The necessary packages for R, used here are:

* library(caret)
* library(rpart)
* library(tree)
* library(randomForest)
* library(e1071)
* library(ggplot2)

**Step 1: Read and summarize the data**

Using R, read the file seaflow\_21min.csv and get the overall counts for each category of particle. You may consider using the functions read.csv andsummary.

data <- read.csv("seaflow\_21min.csv")

# Since there is no single identifier (we have composite of file\_id and cell\_id), adding an id column

id <- rownames(data)

data <- cbind(id=id, data)

# Question 1: How many particles labeled "synecho" are in the file provided?

length(data$id[data$pop == 'synecho'])

# Question 2: What is the 3rd Quantile of the field fsc\_small?

summary(data$fsc\_small)[5]

**Step 2: Split the data into test and training sets**

Divide the data into two equal subsets, one for training and one for testing. Make sure to divide the data in an unbiased manner.

You might consider using either the createDataPartition function or the sample function, although there are many ways to achieve the goal.

# Question 3: What is the mean of the variable "time" for your training set?

summary(train$time)[4]

**Step 3: Plot the data**

Plot pe against chl\_small and color by pop

I recommend using the function ggplot in the library ggpplot2 rather than using base R functions, but this is not required.

# Question 4: In the plot of pe vs. chl\_small, the particles labeled ultra appear to be somewhat "mixed" with which two other populations of particles?

plot(x=data$pe, y=data$chl\_small, col=data$pop, pch=substring(data$pop, 1, 1))

**Step 4: Train a decision tree.**

Install the rpart library if you do not have it, and load the library.

Many statistical models in R provide an interface of the form

model <- train(formula, dataframe)

You can then use the model object to make predictions by passing it to the predict function.

A formula object describes the relationship between the independent variables and the response variable, and can be expressed with the syntax

response ~ var1 + var2 + var3

and used with the formula function to construct the formula object itself:

fol <- formula(response ~ var1 + var2 + var3)

The rpart library uses this convention. Assuming your training data is in a data frame called training and you have constructed a formula object called fol, you can construct a decision tree using the following syntax (included here to avoid you struggling with a couple of unusual aspects of R):

model <- rpart(fol, method="class", data=training)

Train a tree as a function of the sensor measurements: fsc\_small + fsc\_perp + fsc\_big + pe + chl\_big + chl\_small

Print the model object using the print function print(model)

The output is a set of decision nodes, one node per line. Each line is indented indicating the height of the tree. Here is a bogus example of a tree:

1) root 33456 22345 nano (0.0016 0.17 0.29 0.25 0.28)

2) chl\_small< 31000 26238 15772 pico (0 0.22 0.4 3.8e-05 0.38)

4) fsc\_perp< 2040 11430 1913 pico (0 8.7e-05 0.83 8.7e-05 0.17) \*

10) chl\_small>=12000 7065 628 nano (0 0.88 0 0 0.12) \*

11) chl\_small< 12000 9000 2232 ultra (0 0.13 0.097 0 0.77) \*

5) fsc\_perp>=2040 14808 5500 ultra (0 0.39 0.064 0 0.55)

3) chl\_small>=31000 9933 780 synecho (0.0058 0.054 0.0044 0.92 0.014)

6) pe>=17532 681 156 nano (0.085 0.77 0 0.069 0.075) \*

7) pe< 17532 9252 146 synecho (0 0.0014 0.0048 0.98 0.0096) \*

To make a prediction, walk down the tree applying the predicates to determine which branch to take. For example, in this bogus tree, a particle with chl\_small=25000 and fsc\_perp=1000 would take branch 2, branch 4, then branch 10, and be classified as nano.

Answer Questions 5, 6, 7.

**Step 5: Evaluate the decision tree on the test data.**

Use the predict function to generate predictions on your test data. Then, compare these predictions with the class labels in the test data itself.

In R, if you write A==B and A and B are vectors, the result is a vector of 1s and 0s. The sum of this vector will be the number of correct predictions. Dividing this sum by the size of the test dataset will give you the accuracy.

Answer Question 8.

**Step 6: Build and evaluate a random forest.**

Load the randomForest library, then call randomForest using the formula object and the data, as you did to build a single tree:

library(randomforest)

model <- randomForest(fol, data=trainingdata)

Evaluate this model on the test data the same way you did for the tree.

Answer Question 9.

Random forests can automatically generate an estimate of variable importance during training by permuting the values in a given variable and measuring the effect on classification. If scrambling the values has little effect on the model's ability to make predictions, then the variable must not be very important.

A random forest can obtain another estimate of variable importance based on the Gini impurity that we discussed in the lecture. The functionimportance(model) prints the mean decrease in gini importance for each variable. The higher the number, the more the gini impurity score decreases by branching on this variable, indicating that the variable is more important.

Call this function and answer Question 10.

**Step 7: Train a support vector machine model and compare results.**

Use the e1071 library and call model <- svm(fol, data=trainingdata).

Answer Question 11.

**Step 8: Construct confusion matrices**

Use the table function to generate a confusion matrix for each of your three methods. Generate predictions using the predict function, then call the table functions like this:

table(pred = predictions, true = testingdata$pop)

Answer Question 12.

**Step 8: Sanity check the data**

As a data scientist, you should never trust the data, especially if you did not collect it yourself. There is no such thing as clean data. You should always be trying to prove your results wrong by finding problems with the data. Richard Feynman calls it "bending over backwards to show how you're maybe wrong." This is even more critical in data science, because almost by definition you are using someone else's data that was collected for some other purpose rather than the experiment you want to do. So of course it's going to have problems.

The measurements in this dataset are all supposed to be continuous (fsc\_small, fsc\_perp, fsc\_big, pe, chl\_small, chl\_big), but one is not. Using plots or R code, figure out which field is corrupted.

Answer Question 13

There is more subtle issue with data as well. Plot time vs. chl\_big, and you will notice a band of the data looks out of place. This band corresponds to data from a particular file for which the sensor may have been miscalibrated. Remove this data from the dataset by filtering out all data associated with file\_id 208, then repeat the experiment for all three methods, making sure to split the dataset into training and test sets after filtering out the bad data.

Answer Question 14