Coding Exercises Solutions / Explanations

Austin McTier

8/15/2023

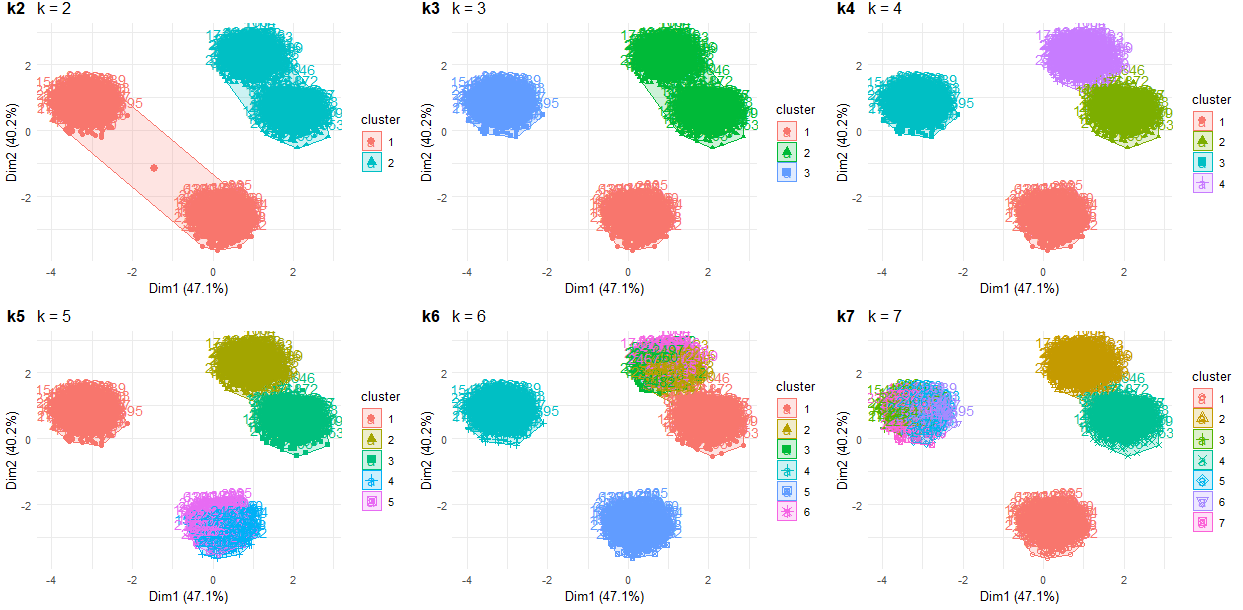
***1) Unsupervised + supervised learning.***

*--------------------------------------*

*Attached is a data file dataClustering.csv which contains a data set of 2500 samples with 8 features.*

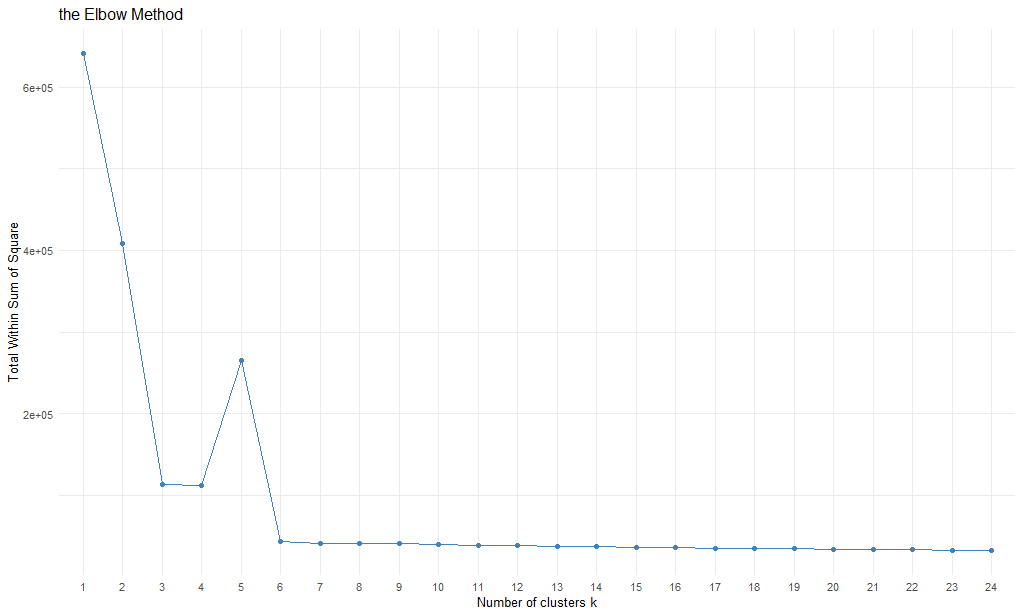
*i) Perform any clustering of your choice to determine the optimal # of clusters in the data*

**I am personally more familiar with K-means clustering that other clustering algorithms, so I will use that for this question. First, I will want to visualize the clusters for different k values to get an idea of what could be the optimal number of clusters (Figure 1\_1)**

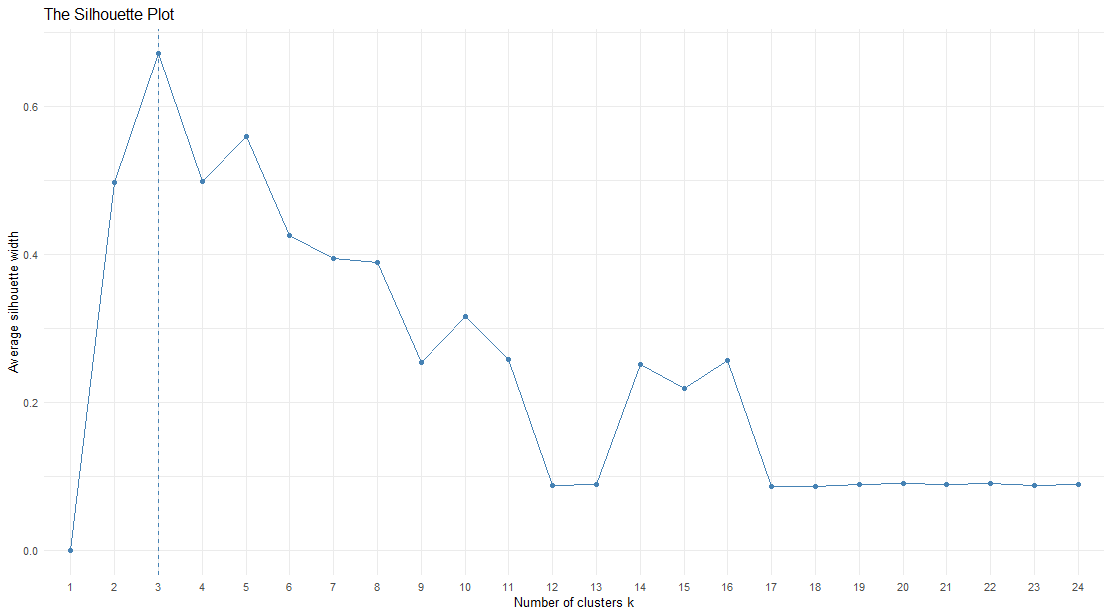
****

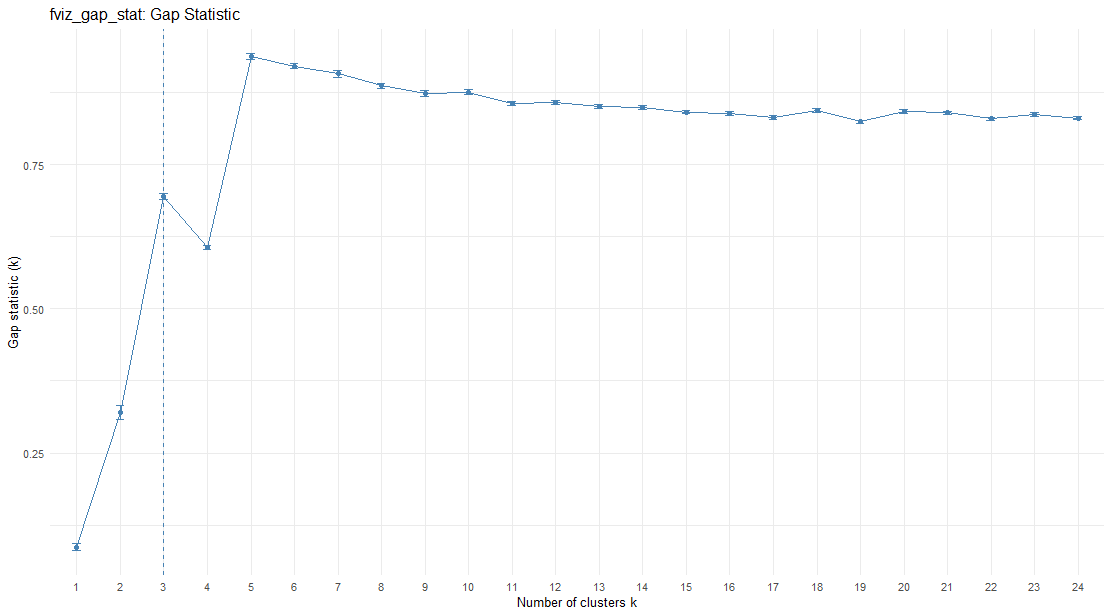
**It seems like either k = 3 or k = 4 could work; k=5 is also an option. Now I will go through a couple of different methods to see what the optimal number of clusters could be.**

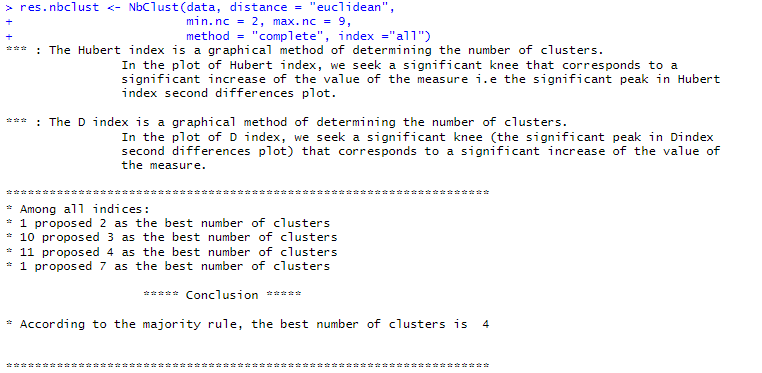
1. **The Elbow Method - sum of squares at each number of clusters is calculated and graphed (Figure 1\_1A)**

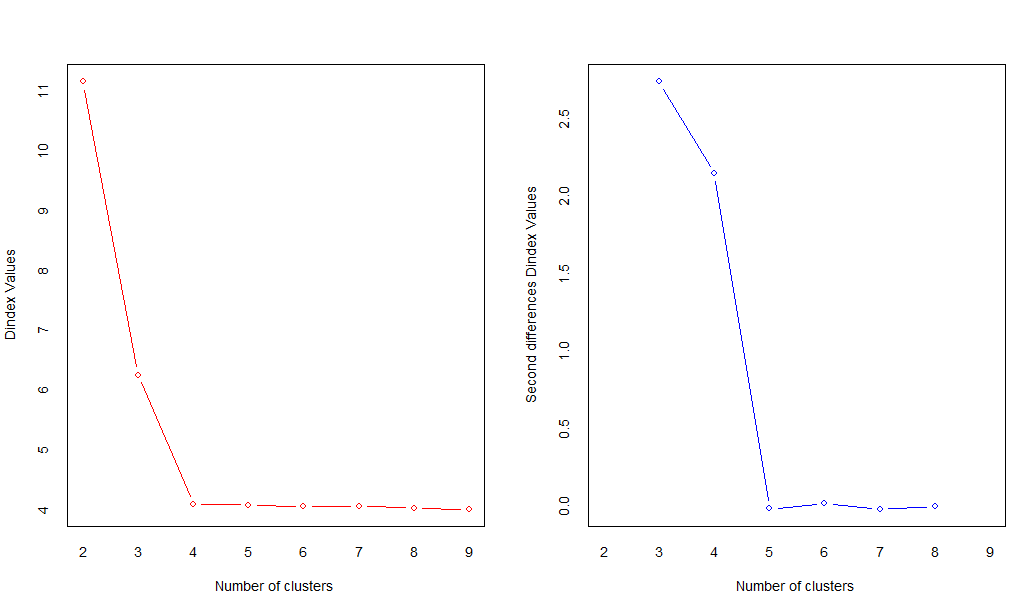
****

1. **The Silhouette Method - computes the average silhouette of observations for different values of k. The optimal number of clusters k is the one that maximize the average silhouette over a range of possible values for k. (Figure 1\_1B) (Next Page)**

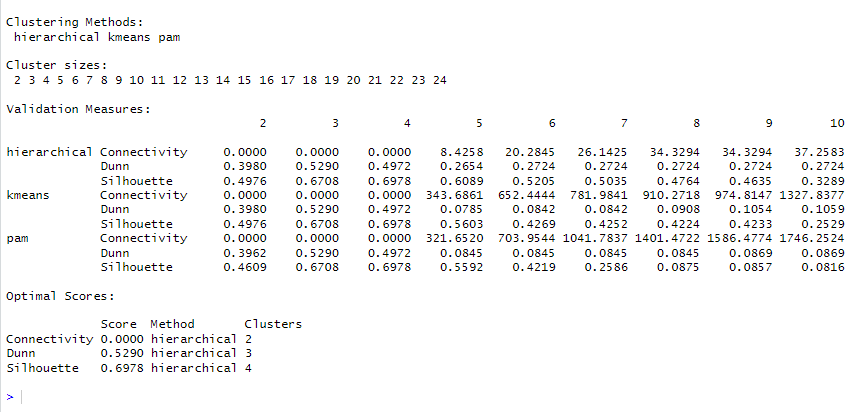
**

1. **The Gap Statistic (Figure 1\_1C)**
2. **NbClust Method – provides 30 indices for determining the relevant number of clusters and proposes to users the best clustering scheme from the different results obtained by varying all combinations of number of clusters, distance measures, and clustering methods.**

****

**

**Based on the results, it seems that the optimal # of clusters is either 3 or 4. I am going to go w/ 3 given the Gap Plot & Silhouette Plot results. I do, however, understand that there are other clustering methods (i.e. Hierarchical, PAM, etc.) that may be better than K-means clustering in this case. So, as an aside, I will check to see if I'm using the optimal clustering method (full image is Figure 1\_Valid)**

**

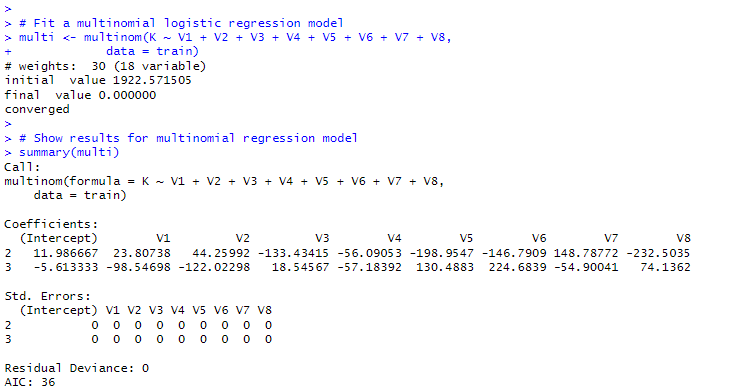
**According to the results, hierarchical clustering would be the preferable clustering method, but 3-4 clusters still appears to be the optimal number of clusters. For the purpose of ii, I will be using 3 clusters**

*ii) Using the result of i) assign clusters labels to each sample, so each sample's label is the cluster to which it belongs. Using these labels as the exact labels, you now have a labeled dataset. Build a classification model that classifies a sample with its corresponding label. Use multinomial regression as a benchmark model, and any ML model (trees, forests, SVM, NN etc.) as a comparison model. Comment on which does better and why.*

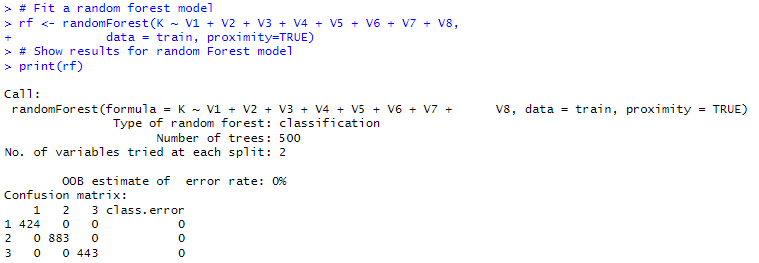
**Given that I used k-means clustering to determine the optimal number of clusters, I will now assign cluster labels to each sample. Now "K" is in our dataset as the cluster label, assigning a datapoint it’s associated cluster label.**

**We now want to build a classification model that classifies a sample with its corresponding label. We will use a multinomial regression as a benchmark model. For a comparison model, I will be using a randomForest model.**

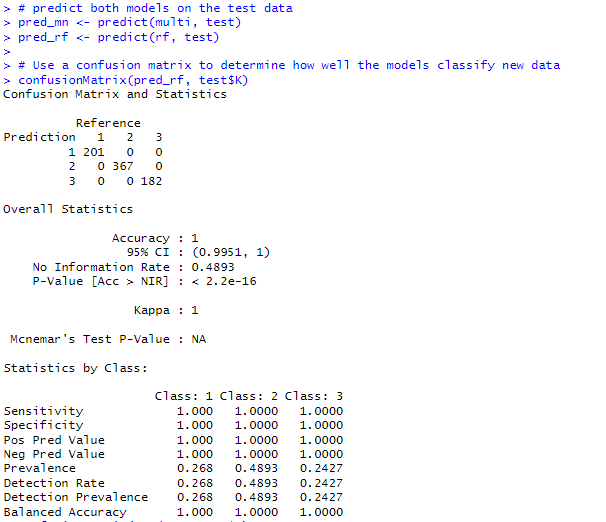
**To do this, we separate the data into training data (70% of the original data) and test data (30% of the original data), to see which model is a better means of classification.**

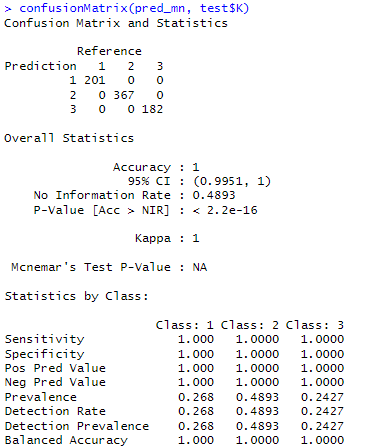
**We then fit a multinomial logistic regression model**

**We then fit a random forest model**

****

**We now put the models to the test by analyzing how they perform on the test data. We use the predict function to see what values are estimated by both models when using the test data. We then analyze both predictions using a confusion matrix, to see how many data points in test are correctly classified by the model**





**For some reason, both models perfectly classify the test data without any misclassification. So both models perform exactly the same. I am unsure if that was suppose to be the result or if I messed something up in my process. I also tried doing so on the entire data, as well as trying 4 clusters instead of 3, and got the same result. Or if it’s something w/ my application of the predict function or confusion matrix**

***2) Prediction + filtering***

*--------------------------------------*

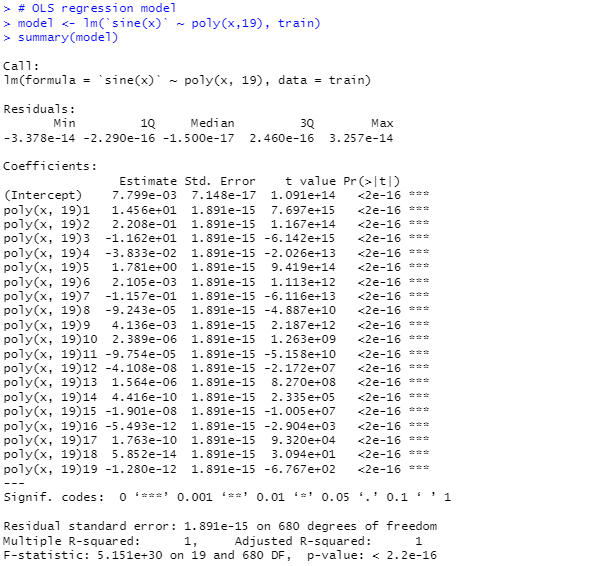
*Attached are 3 files: xvalsSine.csv, cleanSine.csv and noisySine.csv. xvalsSine.csv contains 1000 x-values in the interval -pi/2 to pi/2. cleanSine.csv is a pure sine(x) function for the x values mentioned earlier. noisySine.csv contains sine(x) corrupted by noise.*

*i. Using xvalsSine.csv and cleanSine.csv as a labeled dataset (x,sine(x)) being (value,label) with a random train/test split of 0.7/0.3, build an OLS regression model (you may want to use polynomial basis of a sufficiently large order).*

**First we load the data into the environment and combine cleanSine and xvalsSine, so that we get the full dataset. We then label the columns “sine(x)” and “x”, respectively (Lines 203 – 212 in the code)**

**We then separate the dataset into training data (70%) and test data (30%). After some testing, a polynomial basis of 19 should be of sufficiently large order, since 20 polynomials onward resulted in statistical insignificance for coefficients for those specific polynomial orders. (Shown on next page).**

***Note:*** *I initially tried it with just an order 9 polynomial, and it worked fine for me. So just kept increasing the polynomial order till coefficient values became insignificant*

****

**We then test the model on the test data, and determine how well it does. I will determine so by a normalized Root Mean Squared Error (RMSE) approach**

**A screenshot of a computer code

Description automatically generated**

**So we’ve created an OLS model that fits well to both the training data and the test data**

*(bonus) If you used the normal equations to solve the OLS problem, can you redo it with stochastic gradient descent? (What do you mean by "normal"?)*

**I’m not 100% sure what is meant by “normal equations”. Is this meant to be just a 1 order polynomial?**

**The stochastic gradient descent (sgd) function in R is sgd():**

*sgd.theta <- sgd(`sine(x)` ~ x, train, model="lm")*

*predict(sgd.theta, test, type="link")* **For some reason, the 'predict' function doesn't work w/ sgd, due to something wrong w/ how it's formatted, and how it doesn't play nice w/ predict, resulting in the Matrix multiplication not working. Returns error: "Error in newdata %\*% coef(object) requires numeric/complex matrix/vector arguments" .**

**Lines 242 – 272 was my attempt to produce a predict.sgd function so that it would probably utilize a predict-style function on an sgd output, but alas did not work. So just did the math manually to produce the desired outcome.**

**Since the OLS equation of order 1 is just , I instead did**

*y <- sgd.theta$coefficients[1] + sgd.theta$coefficients[2]\*test[,1]*

*rmse <- sqrt(mean((test$`sine(x)` - y)^2))*

*print(rmse) =* ***0.474417***

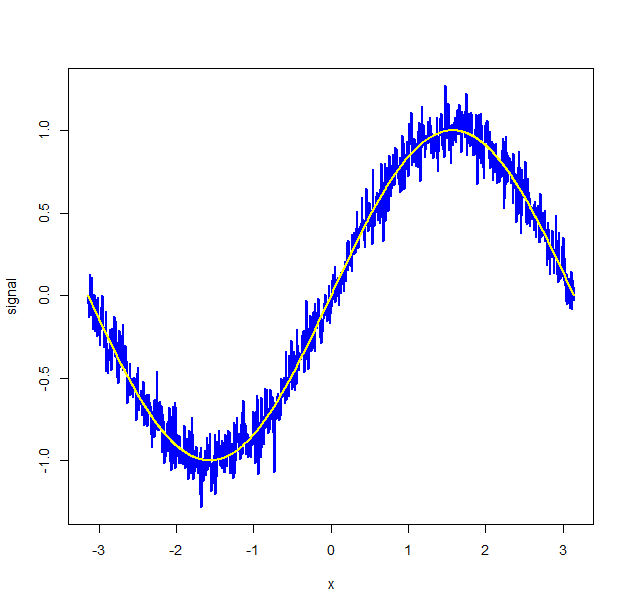
*nrmse <- rmse/sd(pred)*

*print(nrmse) =* ***0.6713944***

**I understand that the purpose of SGD is to find the model parameters that correspond to the best fit between predicted and actual outputs, so can do more research into it’s purpose and utilization, if not in R then in Python.**

*ii. Now, assume you are given the noisySine.csv as a time series with the values of xvalsSine.csv being the time variable. Filter the noisySine.csv data with any filter of your choice and compare against cleanSine.csv to report the error.*

**First we create the noisy sine data by combining noisySine and xvalsSine, similar to what we did in i). Then we plot the clean sine data and the noisy sine data to get an idea of how noisy the latter is. The clean sine data is the yellow line, the noisy sine data is blue.**

****

**We can also estimate the signal-to-noise to get a better idea of how noisy the data is relative to it's clean version.**

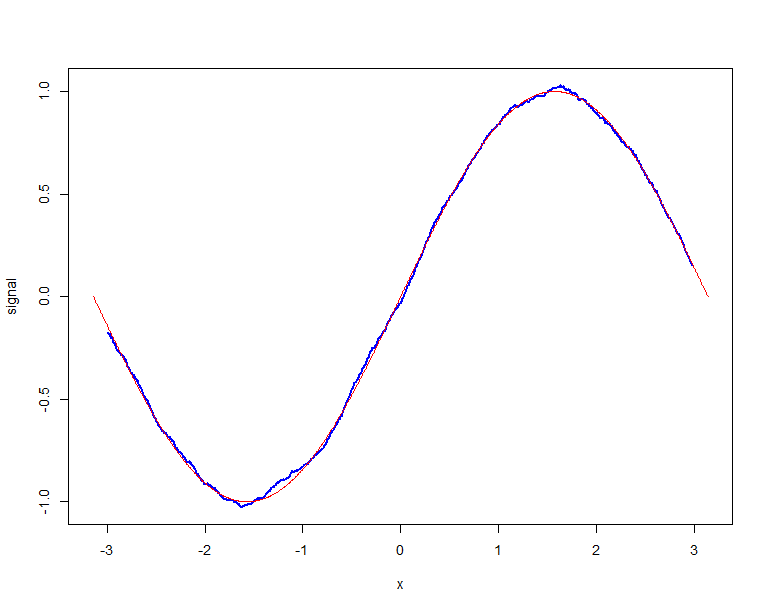
*s\_to\_n <- max(data$`sine(x)`)/sd(dataNoisy$`sine(x)`)*

*s\_to\_n =* ***1.401357***

**Will use R's filter() function to smooth noise and remove background signals. Could use R's fft() function for Fourier filtering, but I am personally not as familiar with that one. So using a 50-point moving average for the filter***mov\_avg\_e = rep(1/50, 50)*

*noisy\_signal\_movavg <- stats::filter(dataNoisy$`sine(x)`, mov\_avg\_e)*

*plot(x = data$x, y = noisy\_signal\_movavg, type = "l", lwd = 2, col = "blue", xlab = "x", ylab = "signal")*

*****lines(x = data$x, y = data$`sine(x)`, lwd = , col = "red")*

**Now look at the signal-to-noise ratio again**

*s\_to\_n\_movavg <- max(data$`sine(x)`)/sd(noisy\_signal\_movavg, na.rm = TRUE)*

*s\_to\_n\_movavg* = **1.385845**

**Looking at the average error and the RMSE, we see that**

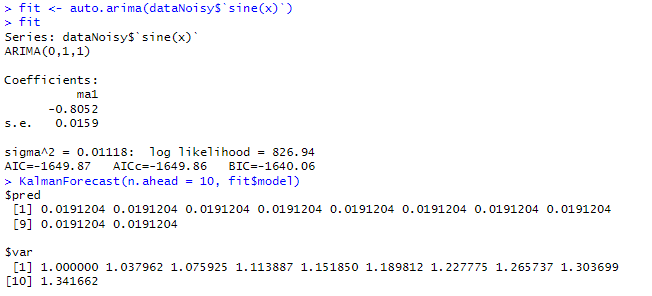
*# average error*

*mean(data$`sine(x)` - noisy\_signal\_movavg, na.rm=T) # =* ***-0.00259983***

*# root mean squared error*

*sqrt(mean((data$`sine(x)` - noisy\_signal\_movavg)^2, na.rm=T)) # =* ***0.01551806***

(*bonus) Can you code a Kalman filter to predict out 10 samples from the noisySine.csv data?*

**Tried the OLS model I did for i) but it gave an error, "Error in KalmanForecast(n.ahead = 10, fit$model) : invalid argument type", so trying an auto.arima instead, which spits out the following for the prediction of the next 10 samples**

***3) Time Series with pi***

*--------------------------------------*

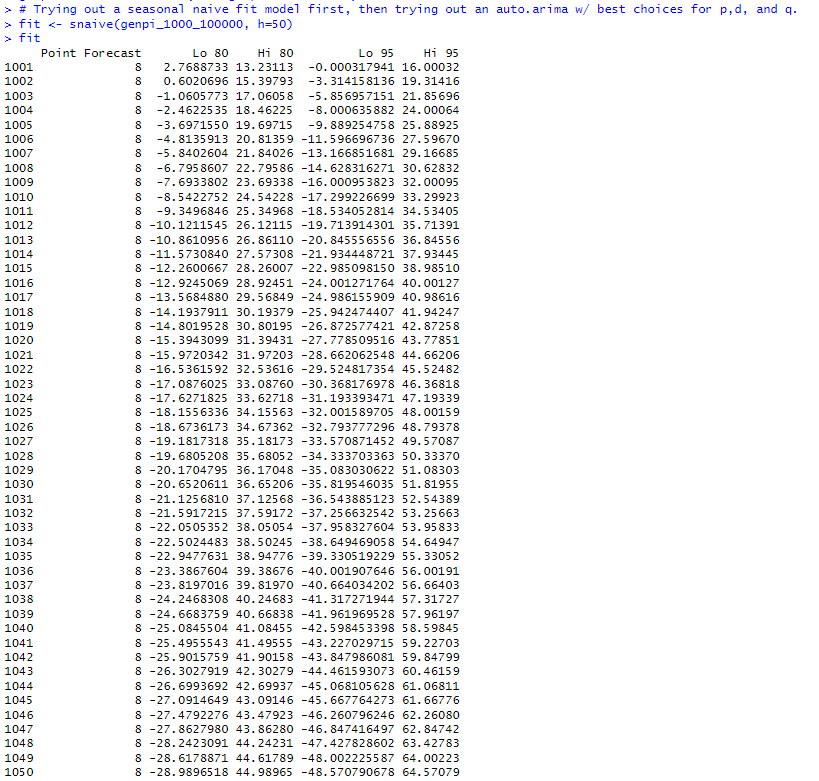
*Attached is a function genPiAppxDigits(numdigits,appxAcc) which returns an approximate value of pi to numdigits digits of accuracy. appxAcc is an integer that controls the approximation accuracy, with a larger number for appxAcc leading to a better approximation.*

*i) Fix numdigits and appxAcc to be sufficiently large, say 1000 and 100000 respectively. Treat each of the 1000 resulting digits of pi as the value of a time series. Thus x[n]=nth digit of pi for n=(1,1000). Build a simple time series forecasting model (any model of your choice)that predicts the next 50 digits of pi. Report your accuracy. Using your results, can you conclude that pi is irrational? If so, how?*

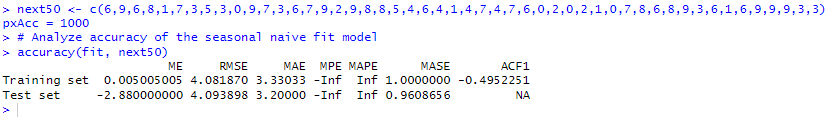
**I was unable to find / produce an adequate equivalent for the function below in R, since R seems to limit the possible number of digits for pi to 1000 digits.**

**Therefore, I use python 3.11 to use the functions and generate the value of pi up to the Nth digit (and for varying appxAcc values), while using R to do the time series analysis.**

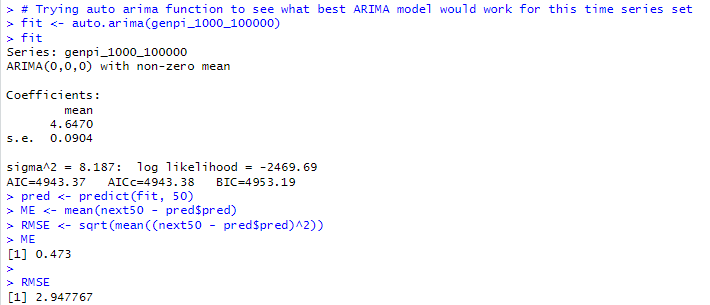
**Generating the value of pi w/ 1000 digits and 100000 approximation accuracy, we then turn the value into a readable time series format, where each digit is a value. (lines 361 – 366). The model I try is a seasonal naïve fit model first. Then I try out an auto.arima, which will fit an arima(p,d,q) model w/ the best choices for p, d, and q. (Lines 369 & 370). The result is then on the next page:**



**I then list the next 50 digits of pi (1001 – 1050 digits), and then use the *accuracy* function to analyze the accuracy of the seasonal naïve fit model**

****

**Now I try using the auto.arima model to see what best ARIMA model would work for this time series set**

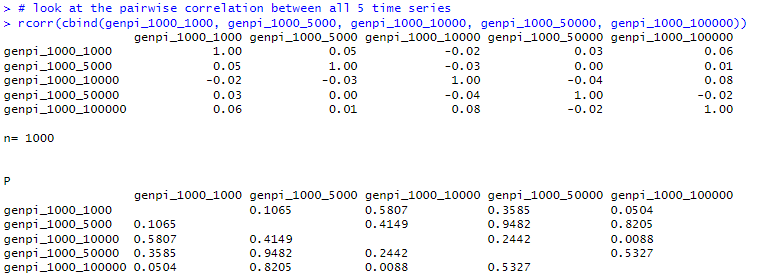
****

**I believe I can conclude that pi is irrational, though I'm uncertain about the strength of my proof. In the seasonal naive forecast I did, the autocorrelation of errors lag 1 (ACF1) is NA, indicating there is no time-based (serial) correlation amongst the digits of Pi as 'time' progresses. Not even insignificant correlation, just no correlation whatsoever (I may be misinterpreting that).**

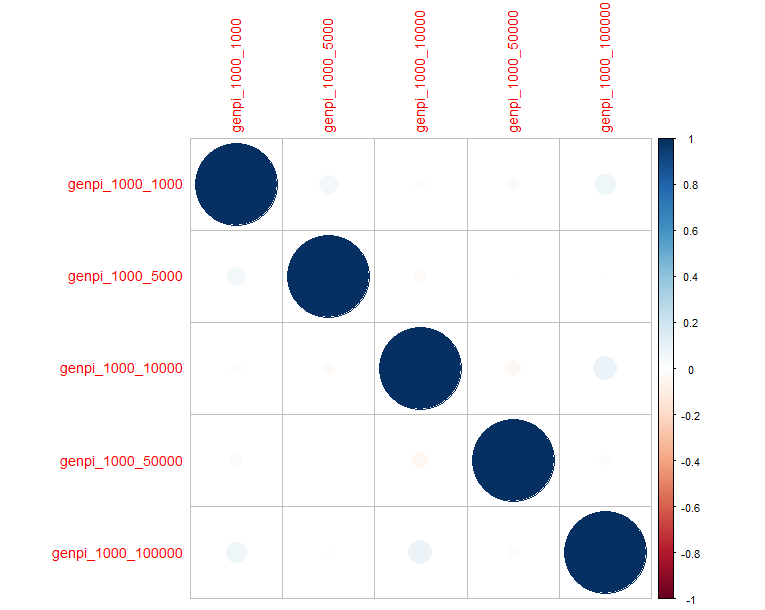
**In addition, the ideal arima model for this time series analysis is ARIMA(0,0,0) meaning the optimal ARIMA model is one with no AR component, no I component, and no MA component, just a flat static forecast of one value into the future. Both models performed terribly on the test data (the next 50 digits of pi).**

*(bonus) Now let's vary appxAcc to be 1000,5000,10000,50000,100000 with fixed numdigits=1000. You thus have 5 time series, each corresponding to a value of appxAcc. Can you find the pairwise correlation between each of the time series?*

**Generating time series data for 1000 digits of pi based on approximation accuracies of 1000, 5000, 10000, 50000, and 100000, respectively, in python, then carrying them to R to produce the time series data (Lines 396-434)**

**We now look at the pairwise correlation with rcorr as a table, and also through visualization w/ a correlation matrix plot. For the former, we do:  
  
**

**For the latter, we get figure 3, and see that there is very low (if not zero or near-zero) correlation between these different approximations of pi:  
  
**

****