# Coding Exercises Solutions / Explanations

Austin McTier

8/15/2023

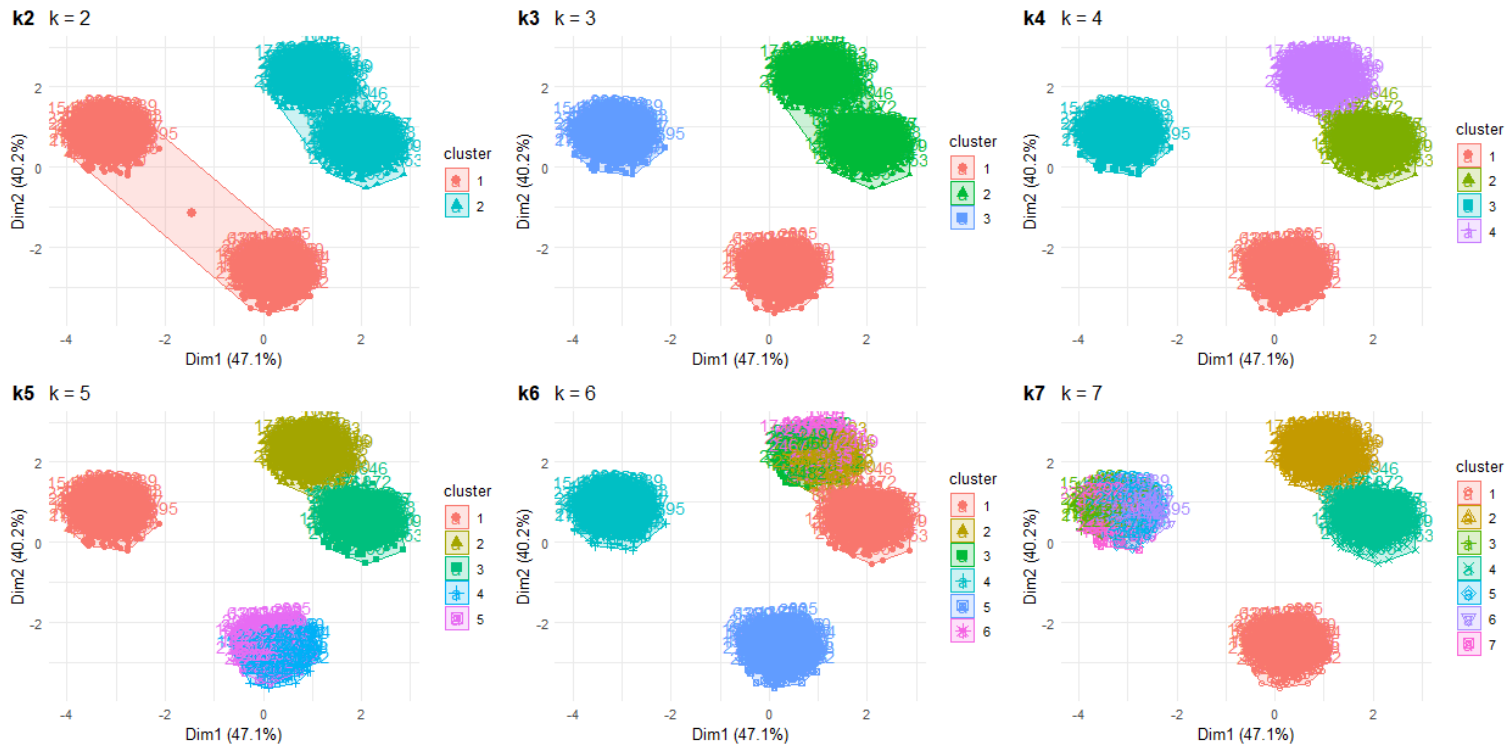
***1) Unsupervised + supervised learning.***

---

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

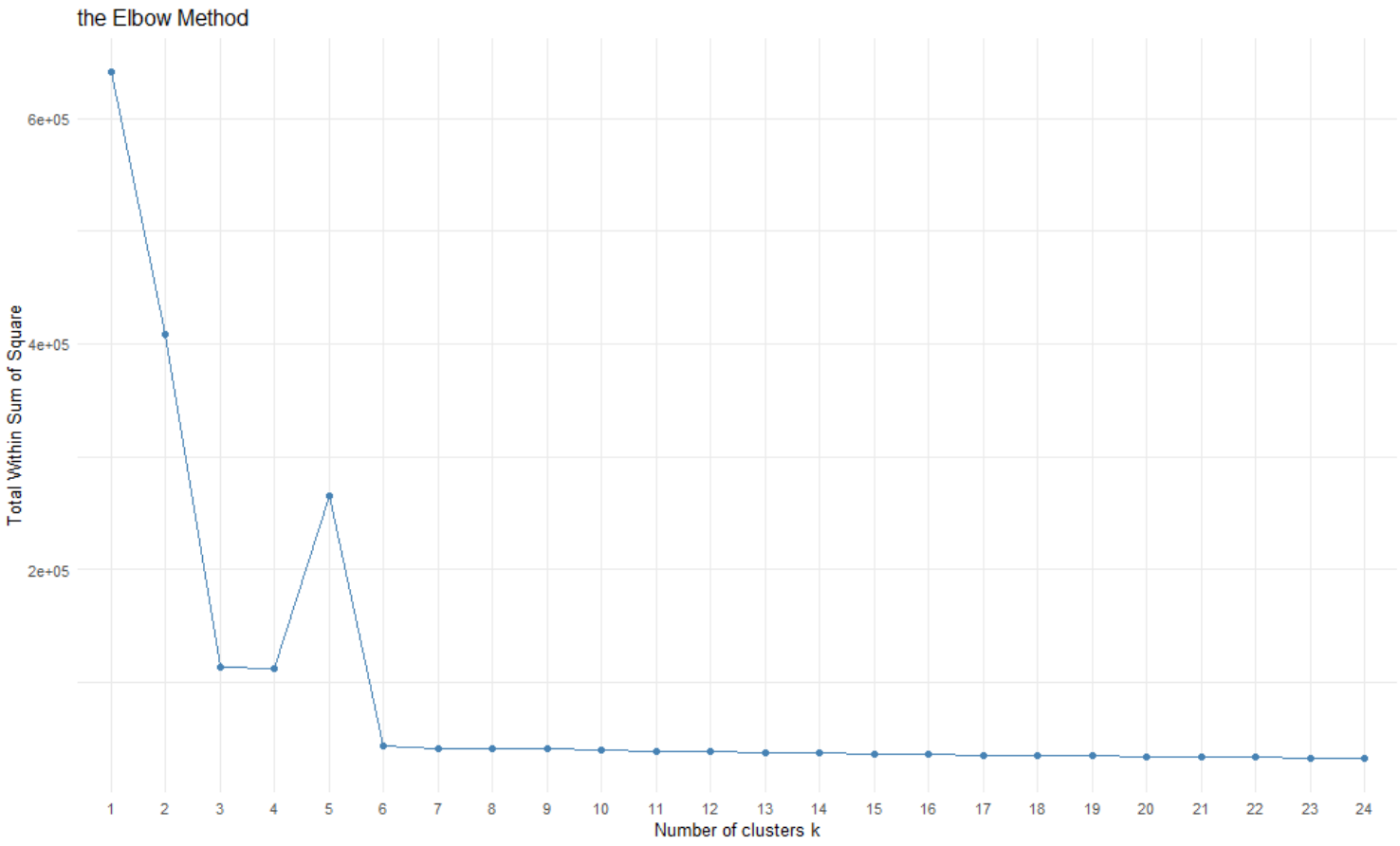
*i) Perform any clustering of your choice to determine the optimal # of clusters in the data*

**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)**



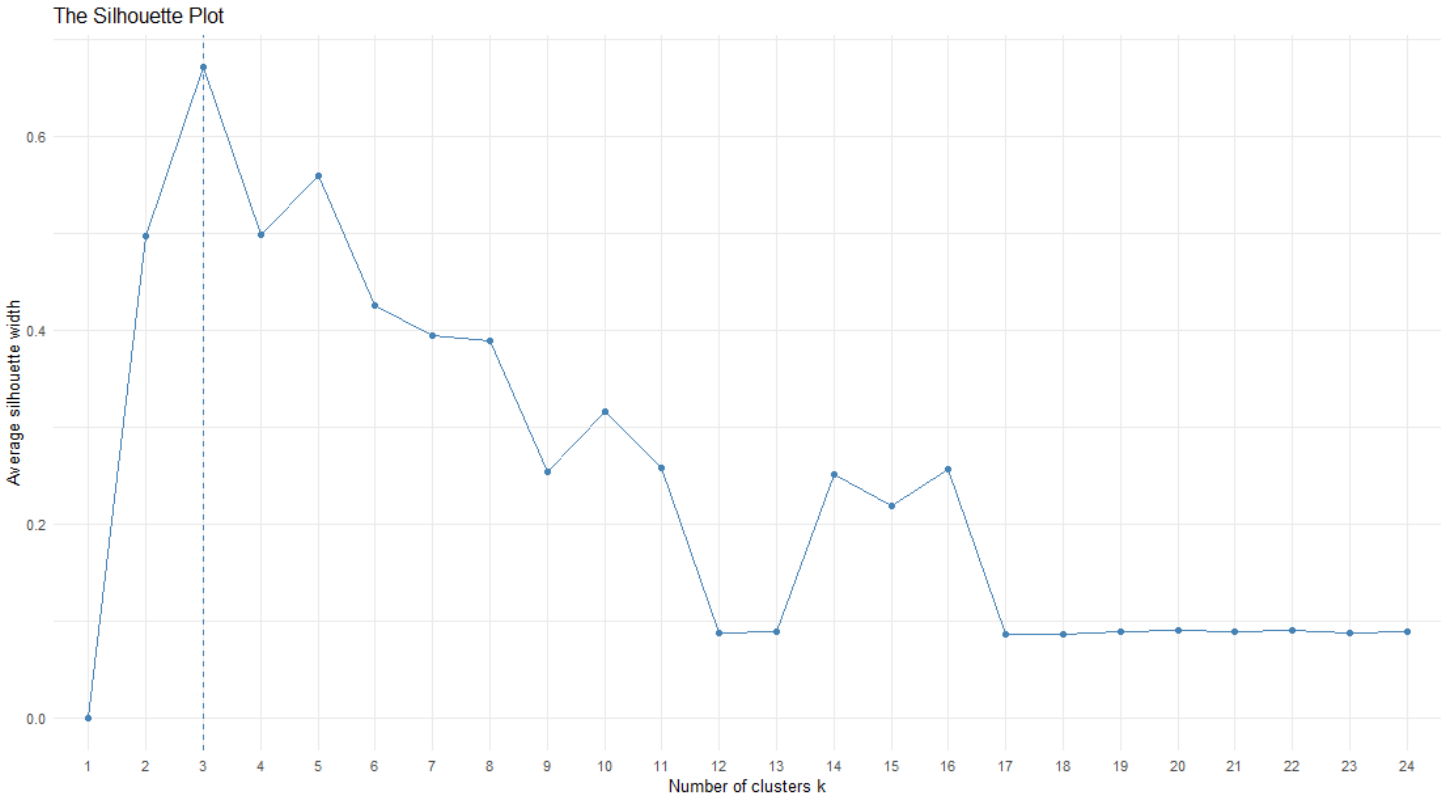
**It seems like either k = 3 or k = 4 could work; k=5 is also an option. Now I will go through a couple of different methods to see what the optimal number of clusters could be.**

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

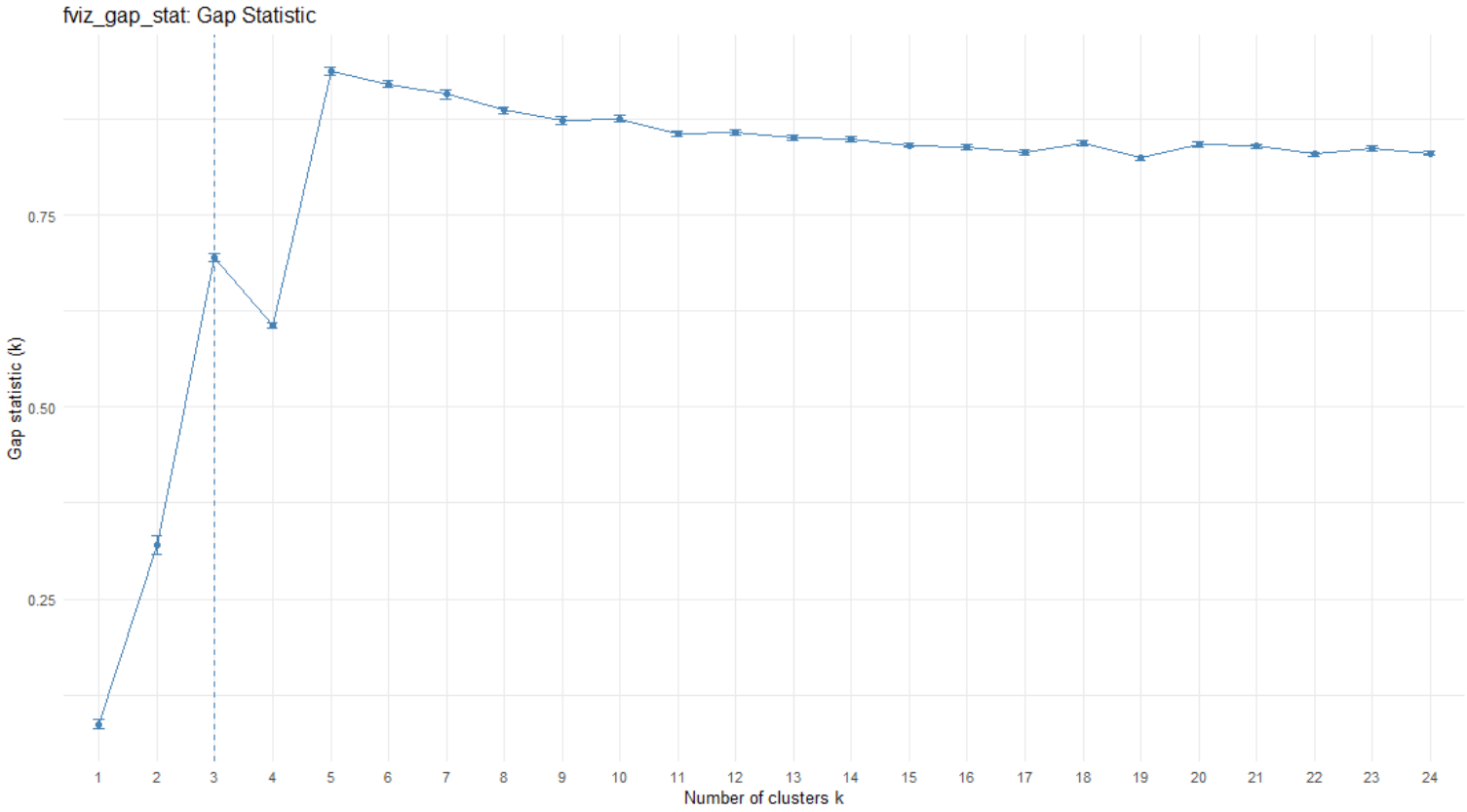



B. **The Silhouette Method - computes the average silhouette of observations for different values of k. The optimal number of clusters k is the one that maximize the average silhouette over a range of possible values for k. (Figure 1_1B) (Next Page)**

NO

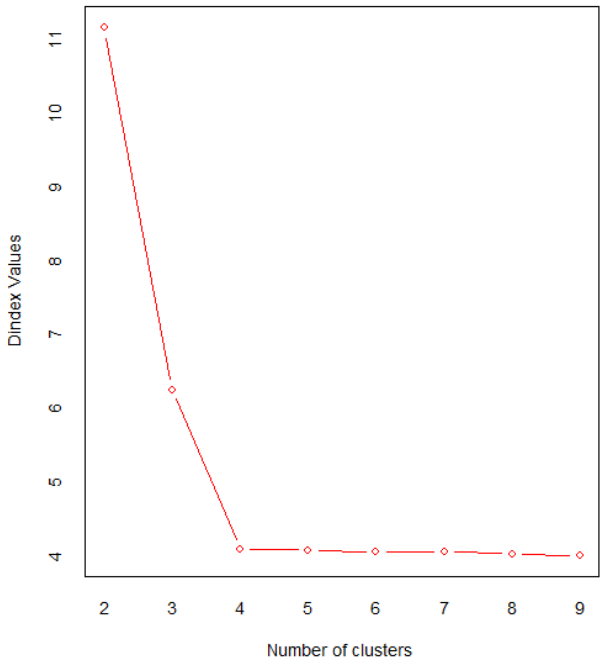


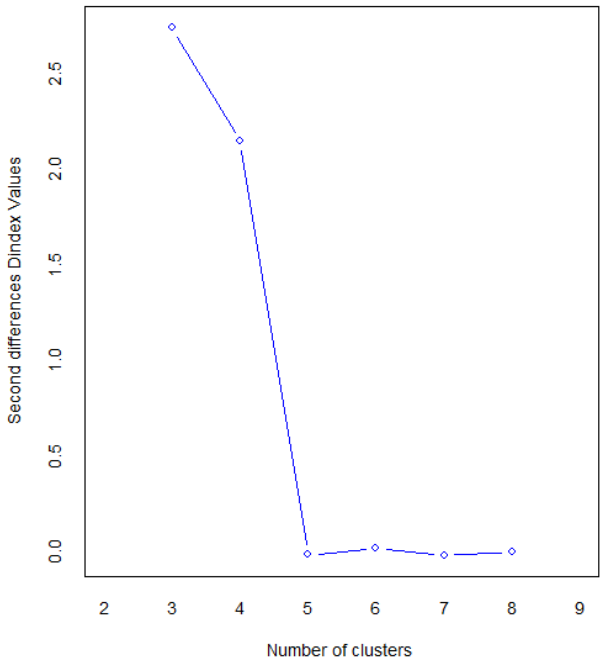

## C. The Gap Statistic (Figure 1_1C)



**D. 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.**

```
> res.nbclust <- NbClust(data, distance = "euclidean",
+                        min.nc = 2, max.nc = 9,
+                        method = "complete", index ="all")
*** : The Hubert index is a graphical method of determining the number of clusters.
                In the plot of Hubert index, we seek a significant knee that corresponds to a
                significant increase of the value of the measure i.e the significant peak in Hubert
                index second differences plot.

*** : The D index is a graphical method of determining the number of clusters.
                In the plot of D index, we seek a significant knee (the significant peak in Dindex
                second differences plot) that corresponds to a significant increase of the value of
                the measure.

*******************************************************************
* Among all indices:
* 1 proposed 2 as the best number of clusters
* 10 proposed 3 as the best number of clusters
* 11 proposed 4 as the best number of clusters
* 1 proposed 7 as the best number of clusters

                    ***** Conclusion *****

* According to the majority rule, the best number of clusters is  4


*******************************************************************
```



**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)**

```
Clustering Methods:
 hierarchical kmeans pam

Cluster sizes:
 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24

Validation Measures:
                                 2         3         4         5         6         7         8         9        10

hierarchical Connectivity     0.0000    0.0000    0.0000    8.4258   20.2845   26.1425   34.3294   34.3294   37.2583
             Dunn             0.3980    0.5290    0.4972    0.2654    0.2724    0.2724    0.2724    0.2724    0.2724
             Silhouette       0.4976    0.6708    0.6978    0.6089    0.5205    0.5035    0.4764    0.4635    0.3289
kmeans       Connectivity     0.0000    0.0000    0.0000  343.6861  652.4444  781.9841  910.2718  974.8147 1327.8377
             Dunn             0.3980    0.5290    0.4972    0.0785    0.0842    0.0842    0.0908    0.1054    0.1059
             Silhouette       0.4976    0.6708    0.6978    0.5603    0.4269    0.4252    0.4224    0.4233    0.2529
pam          Connectivity     0.0000    0.0000    0.0000  321.6520  703.9544 1041.7837 1401.4722 1586.4774 1746.2524
             Dunn             0.3962    0.5290    0.4972    0.0845    0.0845    0.0845    0.0845    0.0869    0.0869
             Silhouette       0.4609    0.6708    0.6978    0.5592    0.4219    0.2586    0.0875    0.0857    0.0816

Optimal Scores:

             Score  Method       Clusters
Connectivity 0.0000 hierarchical 2
Dunn         0.5290 hierarchical 3
Silhouette   0.6978 hierarchical 4

>
```

**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**

```
>
> # Fit a multinomial logistic regression model
> multi <- multinom(K ~ V1 + V2 + V3 + V4 + V5 + V6 + V7 + V8,
+             data = train)
# weights:  30 (18 variable)
initial  value 1922.571505
final  value 0.000000
converged
>
> # Show results for multinomial regression model
> summary(multi)
Call:
multinom(formula = K ~ V1 + V2 + V3 + V4 + V5 + V6 + V7 + V8,
    data = train)

Coefficients:
  (Intercept)        V1         V2         V3        V4        V5        V6        V7        V8
2   11.986667  23.80738   44.25992 -133.43415 -56.09053 -198.9547 -146.7909 148.78772 -232.5035
3   -5.613333 -98.54698 -122.02298   18.54567 -57.18392  130.4883  224.6839 -54.90041   74.1362

Std. Errors:
  (Intercept) V1 V2 V3 V4 V5 V6 V7 V8
2           0  0  0  0  0  0  0  0  0
3           0  0  0  0  0  0  0  0  0

Residual Deviance: 0
AIC: 36
```

## We then fit a random forest model

```
> # Fit a random forest model
> rf <- randomForest(K ~ V1 + V2 + V3 + V4 + V5 + V6 + V7 + V8,
+                    data = train, proximity=TRUE)
> # Show results for random Forest model
> print(rf)

Call:
 randomForest(formula = K ~ V1 + V2 + V3 + V4 + V5 + V6 + V7 +      V8, data = train, proximity = TRUE)
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 2

        OOB estimate of  error rate: 0%
Confusion matrix:
    1   2   3 class.error
1 424   0   0           0
2   0 883   0           0
3   0   0 443           0
```

## 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

```
> # predict both models on the test data
> pred_mn <- predict(multi, test)
> pred_rf <- predict(rf, test)
>
> # Use a confusion matrix to determine how well the models classify new data
> confusionMatrix(pred_rf, test$K)
Confusion Matrix and Statistics

          Reference
Prediction   1   2   3
         1 201   0   0
         2   0 367   0
         3   0   0 182

Overall Statistics

               Accuracy : 1
                 95% CI : (0.9951, 1)
    No Information Rate : 0.4893
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 1

 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: 1 Class: 2 Class: 3
Sensitivity             1.000   1.0000   1.0000
Specificity             1.000   1.0000   1.0000
Pos Pred Value          1.000   1.0000   1.0000
Neg Pred Value          1.000   1.0000   1.0000
Prevalence              0.268   0.4893   0.2427
Detection Rate          0.268   0.4893   0.2427
Detection Prevalence    0.268   0.4893   0.2427
Balanced Accuracy       1.000   1.0000   1.0000
```

```
> confusionMatrix(pred_mn, test$K)
Confusion Matrix and Statistics

          Reference
Prediction   1   2   3
         1 201   0   0
         2   0 367   0
         3   0   0 182

Overall Statistics

               Accuracy : 1
                 95% CI : (0.9951, 1)
    No Information Rate : 0.4893
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 1

 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: 1 Class: 2 Class: 3
Sensitivity             1.000   1.0000   1.0000
Specificity             1.000   1.0000   1.0000
Pos Pred Value          1.000   1.0000   1.0000
Neg Pred Value          1.000   1.0000   1.0000
Prevalence              0.268   0.4893   0.2427
Detection Rate          0.268   0.4893   0.2427
Detection Prevalence    0.268   0.4893   0.2427
Balanced Accuracy       1.000   1.0000   1.0000
```

**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*

```
> # OLS regression model
> model <- lm(`sine(x)` ~ poly(x,19), train)
> summary(model)

Call:
lm(formula = `sine(x)` ~ poly(x, 19), data = train)

Residuals:
       Min         1Q     Median         3Q        Max
-3.378e-14 -2.290e-16 -1.500e-17  2.460e-16  3.257e-14

Coefficients:
                Estimate Std. Error    t value Pr(>|t|)
(Intercept)    7.799e-03  7.148e-17  1.091e+14   <2e-16 ***
poly(x, 19)1   1.456e+01  1.891e-15  7.697e+15   <2e-16 ***
poly(x, 19)2   2.208e-01  1.891e-15  1.167e+14   <2e-16 ***
poly(x, 19)3  -1.162e+01  1.891e-15 -6.142e+15   <2e-16 ***
poly(x, 19)4  -3.833e-02  1.891e-15 -2.026e+13   <2e-16 ***
poly(x, 19)5   1.781e+00  1.891e-15  9.419e+14   <2e-16 ***
poly(x, 19)6   2.105e-03  1.891e-15  1.113e+12   <2e-16 ***
poly(x, 19)7  -1.157e-01  1.891e-15 -6.116e+13   <2e-16 ***
poly(x, 19)8  -9.243e-05  1.891e-15 -4.887e+10   <2e-16 ***
poly(x, 19)9   4.136e-03  1.891e-15  2.187e+12   <2e-16 ***
poly(x, 19)10  2.389e-06  1.891e-15  1.263e+09   <2e-16 ***
poly(x, 19)11 -9.754e-05  1.891e-15 -5.158e+10   <2e-16 ***
poly(x, 19)12 -4.108e-08  1.891e-15 -2.172e+07   <2e-16 ***
poly(x, 19)13  1.564e-06  1.891e-15  8.270e+08   <2e-16 ***
poly(x, 19)14  4.416e-10  1.891e-15  2.335e+05   <2e-16 ***
poly(x, 19)15 -1.901e-08  1.891e-15 -1.005e+07   <2e-16 ***
poly(x, 19)16 -5.493e-12  1.891e-15 -2.904e+03   <2e-16 ***
poly(x, 19)17  1.763e-10  1.891e-15  9.320e+04   <2e-16 ***
poly(x, 19)18  5.852e-14  1.891e-15  3.094e+01   <2e-16 ***
poly(x, 19)19 -1.280e-12  1.891e-15 -6.767e+02   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 1.891e-15 on 680 degrees of freedom
Multiple R-squared:      1,     Adjusted R-squared:      1
F-statistic: 5.151e+30 on 19 and 680 DF,  p-value: < 2.2e-16
```

**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**

```
> # Predicts the values with confidence interval
> pred <- predict(model, test)
>
> # Root mean square error to determine performance
> rmse <- sqrt(mean((test$`sine(x)` - pred)^2))
> rmse
[1] 1.374052e-15
>
> # Normalized RMSE = RMSE / std dev
> nrmse <- rmse/sd(pred)
> nrmse
[1] 1.944557e-15
```

**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 $y = \beta_0 + \beta_1 X + \varepsilon$, 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***

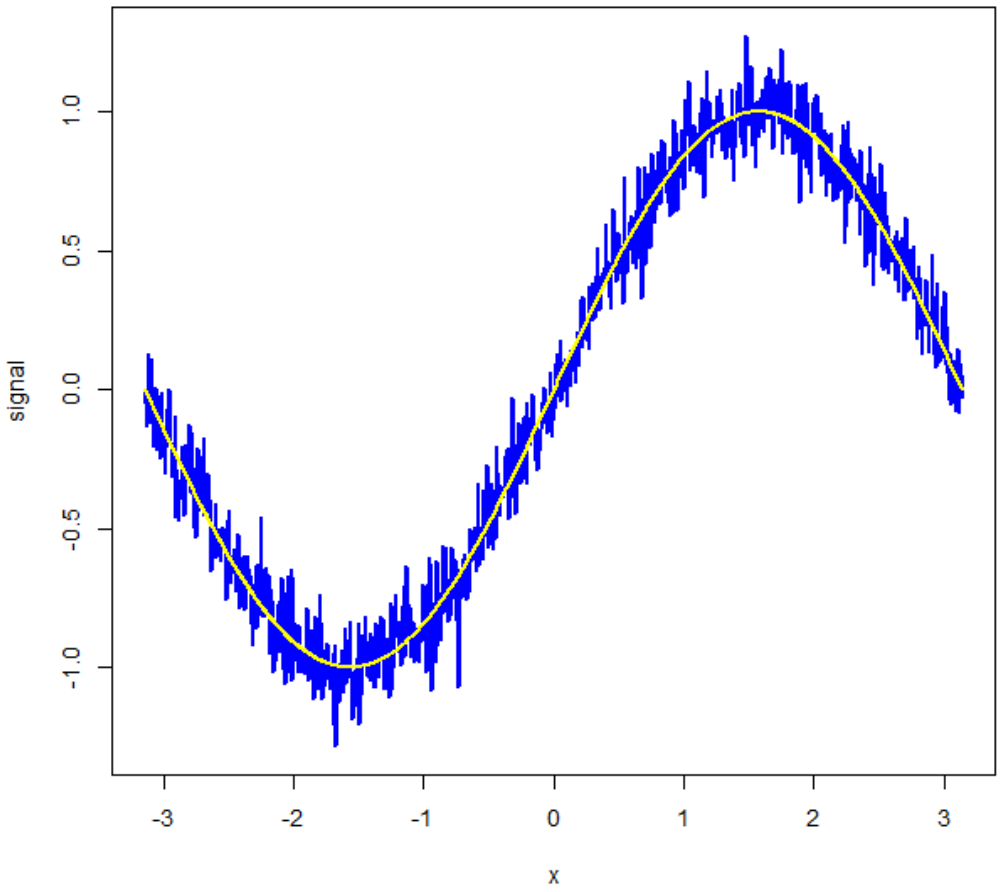
*nrmse <- rmse/sd(pred)*

*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***

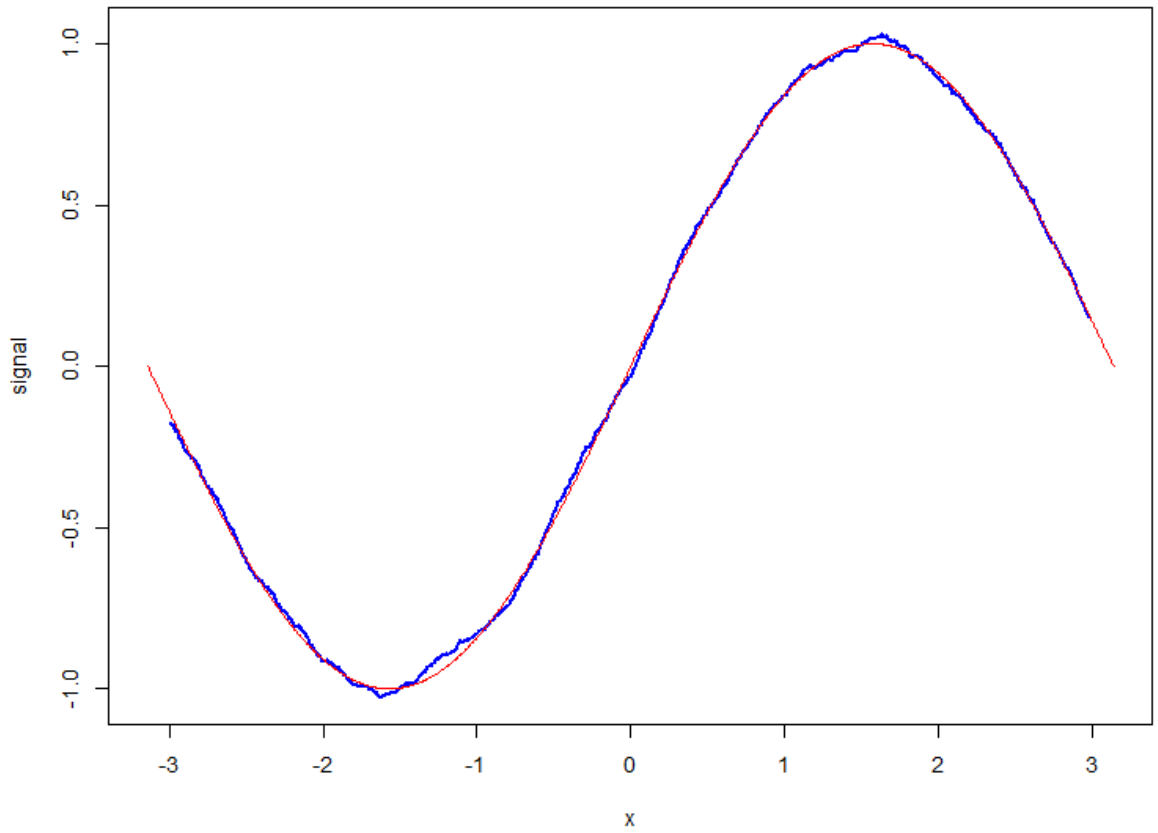
**Will use R's filter() function to smooth noise and remove background signals. Could use R's fft() function for Fourier filtering, but I am personally not as familiar with that one. So using a 50-point moving average for the filter**

*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**

```
> fit <- auto.arima(dataNoisy$`sine(x)`)
> fit
Series: dataNoisy$`sine(x)`
ARIMA(0,1,1)

Coefficients:
          ma1
      -0.8052
s.e.   0.0159

sigma^2 = 0.01118:  log likelihood = 826.94
AIC=-1649.87   AICc=-1649.86   BIC=-1640.06
> KalmanForecast(n.ahead = 10, fit$model)
$pred
 [1] 0.0191204 0.0191204 0.0191204 0.0191204 0.0191204 0.0191204 0.0191204 0.0191204
 [9] 0.0191204 0.0191204

$var
 [1] 1.000000 1.037962 1.075925 1.113887 1.151850 1.189812 1.227775 1.265737 1.303699
[10] 1.341662
```

***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:**

```
> # Trying out a seasonal naive fit model first, then trying out an auto.arima w/ best choices for p,d, and q.
> fit <- snaive(genpi_1000_100000, h=50)
> fit
     Point Forecast       Lo 80    Hi 80         Lo 95    Hi 95
1001              8   2.7688733 13.23113  -0.000317941 16.00032
1002              8   0.6020696 15.39793  -3.314158136 19.31416
1003              8  -1.0605773 17.06058  -5.856957151 21.85696
1004              8  -2.4622535 18.46225  -8.000635882 24.00064
1005              8  -3.6971550 19.69715  -9.889254758 25.88925
1006              8  -4.8135913 20.81359 -11.596696736 27.59670
1007              8  -5.8402604 21.84026 -13.166851681 29.16685
1008              8  -6.7958607 22.79586 -14.628316271 30.62832
1009              8  -7.6933802 23.69338 -16.000953823 32.00095
1010              8  -8.5422752 24.54228 -17.299226699 33.29923
1011              8  -9.3496846 25.34968 -18.534052814 34.53405
1012              8 -10.1211545 26.12115 -19.713914301 35.71391
1013              8 -10.8610956 26.86110 -20.845556556 36.84556
1014              8 -11.5730840 27.57308 -21.934448721 37.93445
1015              8 -12.2600667 28.26007 -22.985098150 38.98510
1016              8 -12.9245069 28.92451 -24.001271764 40.00127
1017              8 -13.5684880 29.56849 -24.986155909 40.98616
1018              8 -14.1937911 30.19379 -25.942474407 41.94247
1019              8 -14.8019528 30.80195 -26.872577421 42.87258
1020              8 -15.3943099 31.39431 -27.778509516 43.77851
1021              8 -15.9720342 31.97203 -28.662062548 44.66206
1022              8 -16.5361592 32.53616 -29.524817354 45.52482
1023              8 -17.0876025 33.08760 -30.368176978 46.36818
1024              8 -17.6271825 33.62718 -31.193393471 47.19339
1025              8 -18.1556336 34.15563 -32.001589705 48.00159
1026              8 -18.6736173 34.67362 -32.793777296 48.79378
1027              8 -19.1817318 35.18173 -33.570871452 49.57087
1028              8 -19.6805208 35.68052 -34.333703363 50.33370
1029              8 -20.1704795 36.17048 -35.083030622 51.08303
1030              8 -20.6520611 36.65206 -35.819546035 51.81955
1031              8 -21.1256810 37.12568 -36.543885123 52.54389
1032              8 -21.5917215 37.59172 -37.256632542 53.25663
1033              8 -22.0505352 38.05054 -37.958327604 53.95833
1034              8 -22.5024483 38.50245 -38.649469058 54.64947
1035              8 -22.9477631 38.94776 -39.330519229 55.33052
1036              8 -23.3867604 39.38676 -40.001907646 56.00191
1037              8 -23.8197016 39.81970 -40.664034202 56.66403
1038              8 -24.2468308 40.24683 -41.317271944 57.31727
1039              8 -24.6683759 40.66838 -41.961969528 57.96197
1040              8 -25.0845504 41.08455 -42.598453398 58.59845
1041              8 -25.4955543 41.49555 -43.227029715 59.22703
1042              8 -25.9015759 41.90158 -43.847986081 59.84799
1043              8 -26.3027919 42.30279 -44.461593073 60.46159
1044              8 -26.6993692 42.69937 -45.068105628 61.06811
1045              8 -27.0914649 43.09146 -45.667764273 61.66776
1046              8 -27.4792276 43.47923 -46.260796246 62.26080
1047              8 -27.8627980 43.86280 -46.847416497 62.84742
1048              8 -28.2423091 44.24231 -47.427828602 63.42783
1049              8 -28.6178871 44.61789 -48.002225587 64.00223
1050              8 -28.9896518 44.98965 -48.570790678 64.57079
```

**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**

```
> next50 <- c(6,9,6,8,1,7,3,5,3,0,9,7,3,6,7,9,2,9,8,8,5,4,6,4,1,4,7,4,7,6,0,2,0,2,1,0,7,8,6,8,9,3,6,1,6,9,9,9,3,3)
pxAcc = 1000
> # Analyze accuracy of the seasonal naive fit model
> accuracy(fit, next50)
                      ME     RMSE     MAE  MPE MAPE      MASE       ACF1
Training set  0.005005005 4.081870 3.33033 -Inf  Inf 1.0000000 -0.4952251
Test set     -2.880000000 4.093898 3.20000 -Inf  Inf 0.9608656         NA
>
```

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

```
> # Trying auto arima function to see what best ARIMA model would work for this time series set
> fit <- auto.arima(genpi_1000_100000)
> fit
Series: genpi_1000_100000
ARIMA(0,0,0) with non-zero mean

Coefficients:
        mean
      4.6470
s.e.  0.0904

sigma^2 = 8.187:  log likelihood = -2469.69
AIC=4943.37   AICc=4943.38   BIC=4953.19
> pred <- predict(fit, 50)
> ME <- mean(next50 - pred$pred)
> RMSE <- sqrt(mean((next50 - pred$pred)^2))
> ME
[1] 0.473
>
> RMSE
[1] 2.947767
```

**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:**

```
> # look at the pairwise correlation between all 5 time series
> rcorr(cbind(genpi_1000_1000, genpi_1000_5000, genpi_1000_10000, genpi_1000_50000, genpi_1000_100000))
                 genpi_1000_1000 genpi_1000_5000 genpi_1000_10000 genpi_1000_50000 genpi_1000_100000
genpi_1000_1000             1.00            0.05            -0.02            0.03             0.06
genpi_1000_5000             0.05            1.00            -0.03            0.00             0.01
genpi_1000_10000           -0.02           -0.03             1.00           -0.04             0.08
genpi_1000_50000            0.03            0.00            -0.04            1.00            -0.02
genpi_1000_100000           0.06            0.01             0.08           -0.02             1.00

n= 1000


P
                 genpi_1000_1000 genpi_1000_5000 genpi_1000_10000 genpi_1000_50000 genpi_1000_100000
genpi_1000_1000                  0.1065          0.5807           0.3585           0.0504
genpi_1000_5000  0.1065                          0.4149           0.9482           0.8205
genpi_1000_10000 0.5807          0.4149                           0.2442           0.0088
genpi_1000_50000 0.3585          0.9482          0.2442                            0.5327
genpi_1000_100000 0.0504         0.8205          0.0088           0.5327
```

**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:**

```
# Create correlation plot to visualize the correlation between all 5 time series
mydata <- data.frame(cbind(genpi_1000_1000, genpi_1000_5000, genpi_1000_10000, genpi_1000_50000, genpi_1000_100000))
corrplot(cor(mydata))
```