Week2, Assignment 1

Gabi Rivera

```r
library(caret)
library(naniar) #missing data visualization
library(knitr) #transform to a neat table
library(moments) #Skewness
library(corrplot)
library(tidyverse)
library(scales)
```

Problem 2.1 (20 points)

The soybean data can also be found at the UC Irvine Machine Learning Repository. Data were collected to predict disease in 683 soybeans. The 35 predictors are mostly categorical and include information on the environmental conditions (e.g., temperature, precipitation) and plant conditions (e.g., left spots, mold growth). The outcome labels consist of 19 distinct classes. The data can be loaded via:

```r
library(mlbench)
data(Soybean)
#?Soybean
```

2.1.a (10 points)

Investigate missing values for all the predictors, which predictors have the highest and the lowest number of missing values?

```r
# Table missing data
Soybean[Soybean == ""] <- NA
na_value <- sapply(Soybean, function(x) sum(is.na(x)))
na_df <- data.frame(Column = names(na_value), Missing_Values = na_value)
na_df_sorted <- na_df[order(-na_df$Missing_Values), ]
```

```r
highest_indices <- which(na_value == max(na_value))
lowest_indices <- which(na_value == min(na_value))
highest_columns <- names(na_value)[highest_indices]
lowest_columns <- names(na_value)[lowest_indices]

# Print the filtered columns
cat("Columns with the highest missing values:",
    paste(highest_columns, collapse = ", "), "\n")
```

```
Columns with the highest missing values: hail, sever, seed.tmt, lodging
```

```r
cat("Columns with the lowest missing values:",
    paste(lowest_columns, collapse = ", "), "\n")
```

```
Columns with the lowest missing values: Class, leaves
```

As shown, the predictors that have the highest missing values are hail, sever, seed.tmt, and lodging all having 121 missing values. Class and leaves variables that have zero missing values.

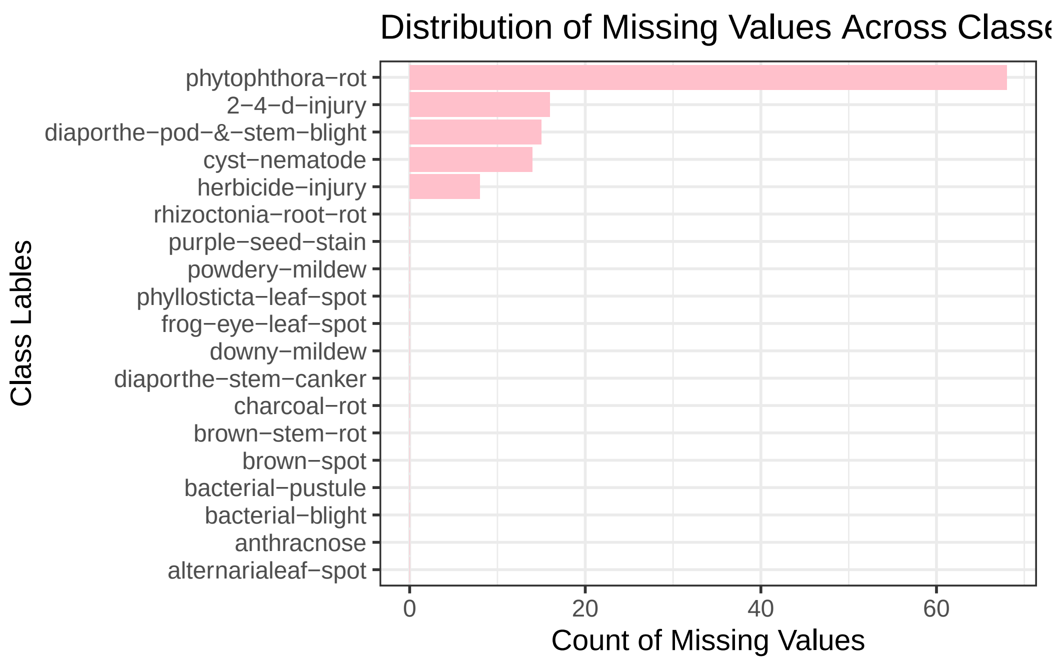
Do the missing values depend on the outcome labels (summarize NAs by class)?

```r
# Summarize Na's by class
Soybean[Soybean == ""] <- NA
complete_rows <- complete.cases(Soybean)
na_summary <- aggregate(!complete_rows ~ Class, data = Soybean, FUN = sum)
```

- *Seems like there are 5 Class labels that have NA counts while the rest have zero NAs.*

Are any of the distributions degenerate in the ways discussed earlier in chapter 3?

```r
# Create a plot of summarized NA's by class
na_summary |>
  ggplot(aes(x = reorder(Class, `!complete_rows`), y = `!complete_rows`)) +
  geom_bar(stat = "identity", fill = "pink") +
  coord_flip() +
  theme_bw() +
  labs(title = "Distribution of Missing Values Across Classes",
       x = "Class Lables",
       y = "Count of Missing Values")
```

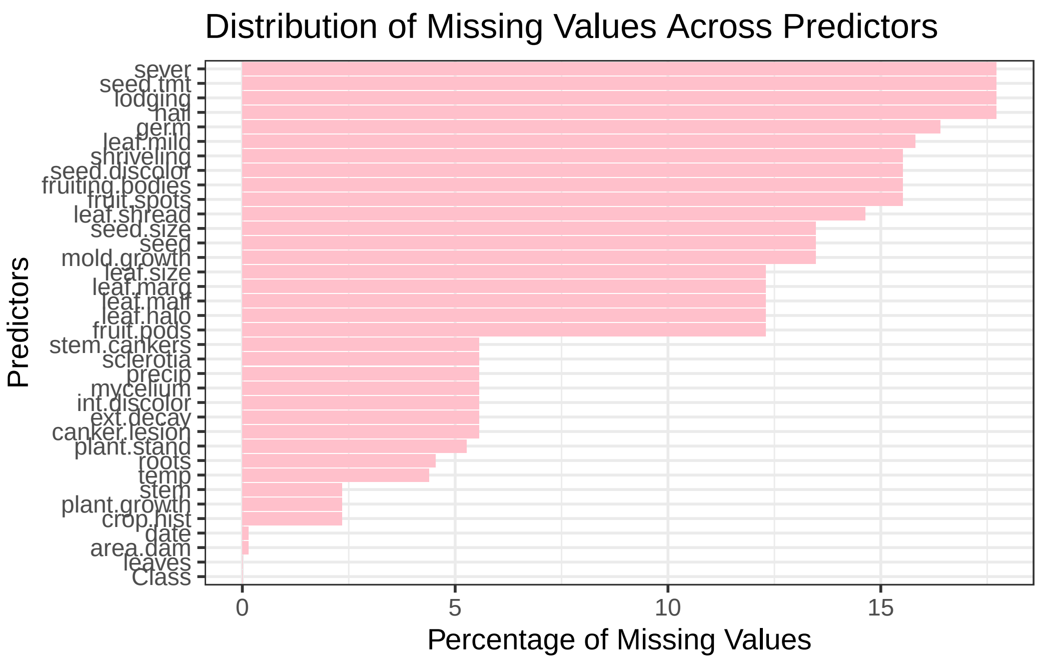


- *Looks like the class phytophthora-rot has the most NAs among the 4 Class labels that contain missing values. This suggest a degenerate distribution as one label shows bias over the rest.*

2.1.b (10 points)

Compute what % of the predictor data is missing?

```r
# Table predictors with corresponding %NA
percent_missing <- colMeans(is.na(Soybean)) * 100
missing_data_summary <- data.frame(
  Predictor = names(percent_missing),
  Percentage_Missing = percent_missing
)

# Create plot for predictors missing values
missing_data_summary |>
  ggplot(aes(x = reorder(Predictor, Percentage_Missing), y = Percentage_Missing)) +
  geom_bar(stat = "identity", fill = "pink") +
  coord_flip() +
  theme_bw() +
  labs(title = "Distribution of Missing Values Across Predictors",
       x = "Predictors",
       y = "Percentage of Missing Values")
```

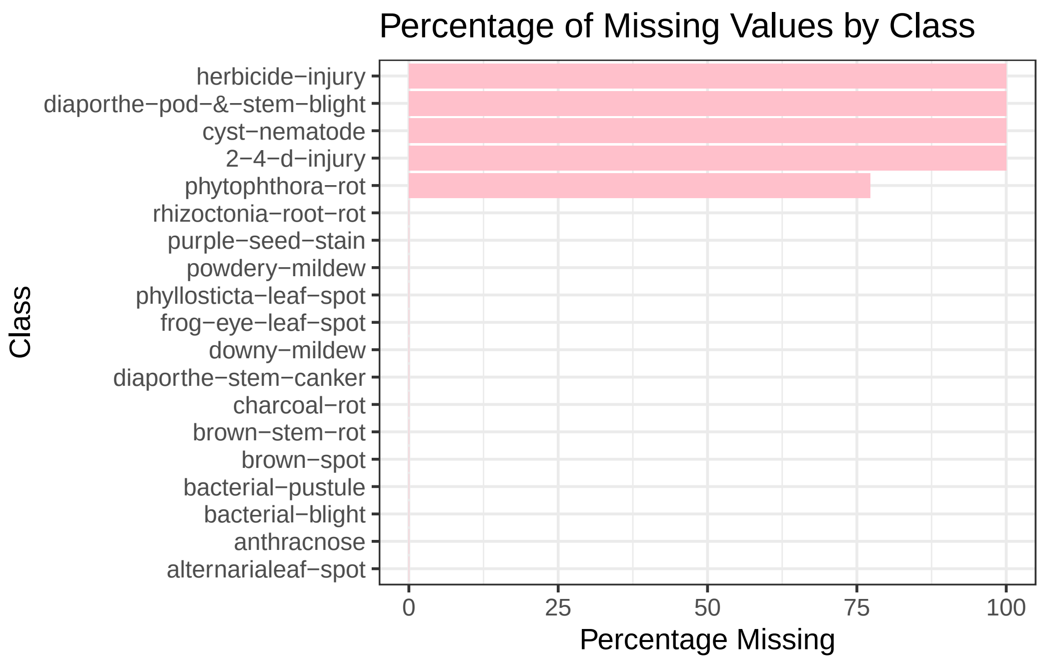


- *As shown in the missing value plot, leaves and class are the variables that have 0 missing values while the rest are ordered in descending order.*

Compute percent of missing data by outcome label and identify classes with highest percent missing values. An example of this calculation: if there are two predictors and 10 rows of data with 5 missing data points then % missing = 5 * 100 / (2*10) = 25%

```
# Calculate the percentage of missing values for each class
total_values <- table(Soybean$Class)
na_summary$Percentage_Missing <- (na_summary$`!complete_rows` /
                                    total_values[na_summary$Class]) * 100

na_summary |>
  ggplot(aes(x = reorder(Class, Percentage_Missing), y = Percentage_Missing)) +
  geom_bar(stat = "identity", fill = "pink") +
  coord_flip() + theme_bw() +
  labs(title = "Percentage of Missing Values by Class", x = "Class",
       y = "Percentage Missing")
```

```
Don't know how to automatically pick scale for object of type <table>.
Defaulting to continuous.
```



Problem 2.2 (10 points)

The caret package contains a QSAR data set from Mente and Lombardo (2005). Here, the ability of a chemical to permeate the blood-brain barrier was experimentally determined for 208 compounds. 134 descriptors were measured for each compound.

2.2.a Load the data:

```r
data(BloodBrain)
#?BloodBrain
```

The numeric outcome is contained in the vector `logBBB` while the predictors are in the data frame `bbbDescr`.

2.2.b (5 points)

Do any of the individual predictors have degenerate distributions

```r
# Identified predictors with skewness higher than -2 and 2
skew_df <- as.data.frame(skewness(bbbDescr, na.rm = FALSE))
colnames(skew_df)[1] <- "Skewness"
skew_df |>
  arrange(desc(Skewness)) |>
  filter(Skewness > 2 | Skewness < -2) |>
  round(2) |>
  head() |>
  knitr::kable()
```

	Skewness
negative	14.32
alert	10.05
a_acid	5.44
vsa_acid	5.44
tcpa	5.37
frac.anion7.	4.32

Checking the distribution through faceted histogram is challenging for 134 variables. Skewness was then used to estimate the predictors that have high skewness score that's greater then 2 and -2. This suggest that yes, some of the predictors have degenerate distribution especially the ones with high skewness score.

2.2.c (5 points)

Generally speaking, are there strong relationships between the predictor data?

Yes, there are strong relationship between some of the predictors. There are 99 predictors that have > +/-50% correlation.

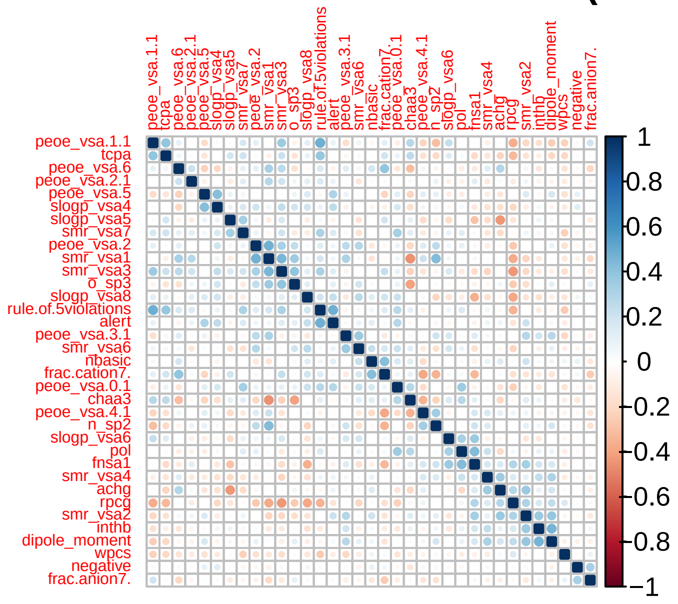
```r
# See dataframe for clear values
cor_matrix <- round(cor(bbbDescr), 2)

# Count the number of predictors that have > .50 and > -.50 correlation score
highCorr <- findCorrelation(cor_matrix, cutoff = .50)
length(highCorr)
```

```
[1] 99
```

If so, how could correlations in the predictor set be reduced?

```r
# Only show predictors at 50% correlation score cutoff
filteredSegData <- bbbDescr[, -highCorr]
correlationsFiltered <- cor(filteredSegData)
corrplot(correlationsFiltered, order = "hclust",
         title = "Correlation Plot Between Predictors (Filtered)",
         tl.cex = .5, number.cex = 0.6, mar = c(1, 1, 1, 1))
```

Correlation Plot Between Predictors (Filtered)



Does this have a dramatic effect on the number of predictors available for modeling?

Yes, applying a cutoff drastically reduced the number of predictors down to ~35 predictors.

Problem 2.3 (10 points)

Consider the permeability data set described in Sect. 1.4. of the textbook. The objective for this data is to use the predictors to model compounds "permeability".

```r
library(AppliedPredictiveModeling)
data(permeability) # this creates two matrices fingerprints and permeability
permeabilitydf <- as_tibble(fingerprints) |>
    mutate(permeability = permeability)
```

2.3.a (5 points)

What data splitting method(s) would you use for these data? Explain.

```r
# Create a dummy outcome column in the data frame
# CreateDataPartition() and createFolds() requires data points

permeabilitydf$dummy_outcome <- factor(rep(1, nrow(permeabilitydf)))

# Explore dataset
#str(permeabilitydf)
```

I prefer to use Stratified Sampling and k-fold Cross-Validation to acquire a wide range of outcome. There is a noticeable large amount of variables and relatively small observation. Stratified sampling will be ideal for large data aspect, simple to implement, and less computationally demanding. K-fold CV give a more robust estimate for model performance even though it is more computationally demanding.

2.3.b (5 points)

Using tools described in this chapter, provide code for implementing your approach(es).

```r
set.seed(503) # set seed for reproducibility

# Stratified Sampling Method
indices <- createDataPartition(y = permeabilitydf$dummy_outcome, p = 0.8,
                               list = FALSE, times = 1)

train_data <- permeabilitydf[indices, ]
test_data <- permeabilitydf[-indices, ]

# k-fold Cross-Validation Method
cvSplits <- createFolds(y = permeabilitydf$dummy_outcome, k = 10, list = FALSE)
train_indices <- vector("numeric", length = 0)
```

```r
test_indices <- vector("numeric", length = 0)

for (i in 1:10) {
  train_indices <- c(train_indices, cvSplits[-i])
  test_indices <- c(test_indices, cvSplits[i])
}

train_data_k <- permeabilitydf[train_indices, ]
test_data_k <- permeabilitydf[test_indices, ]
```

Problem 2.4 (20 points)

Partial least squares was used to model the yield of a chemical manufacturing process (Sect. 1.4). The data can be found in the AppliedPre- dictiveModeling package and can be loaded using:

```r
data(ChemicalManufacturingProcess)
```

The objective of this analysis is to find the number of PLS components that yields the optimal R^2 value. PLS models with 1 through 10 components were each evaluated using five repeats of 10-fold cross-validation and the results are presented in the following table:

2.4.a (7 points)

Using the "one-standard error" method, what number of PLS components provides the most parsimonious model?

```r
# Calculate the upper and lower bounds of resampled R-squared values
pca_table <- pca_table |>
  mutate(UpperBound = Mean + `Std. Error`,
         LowerBound = Mean - `Std. Error`)

# Calculate the one-standard error rule
pca_table <- pca_table |>
  mutate(OneStdErrorRule = Mean - `Std. Error`)

# Identify the row with the highest resampled R-squared value at one-standard
optimal_components <- pca_table |>
  filter(OneStdErrorRule == max(OneStdErrorRule)) |>
  select(Components, Mean, `Std. Error`)
```

```r
optimal_components |>
  tibble() |>
  gt()
```

Components	Mean	Std. Error
4	0.545	0.0308

The resulting number of PLS components that provides the most parsimonious model is 4.

2.4.b (7 points)

If a 10% loss in optimal R2 is acceptable, then what is the optimal number of PLS components?

```r
# Calculate the optimal R-squared value
optimal_r2 <- max(pca_table$Mean)

# Calculate the acceptable R-squared value threshold
acceptable_r2 <- optimal_r2 * (1 - 0.1)

# Identify the number of components closest to the acceptable threshold
optimal_components <- pca_table |>
  mutate(Loss = Mean - acceptable_r2) |>
  filter(Loss == min(Loss)) |>
  select(Components, Mean, `Std. Error`)

optimal_components |>
  tibble() |>
  gt()
```

Components	Mean	Std. Error
1	0.444	0.0272

The optimal number of PLS components is 1 if 10% loss in R^2 is acceptable.

2.4.c (3 points)

Several other models with varying degrees of complexity were trained and tuned and the results are presented in Figure below. If the goal is to select the model that optimizes $\mathbf{R^2}$, then which model(s) would you choose, and why?

I would choose Random Forrest as it has the higher R^2 value and fairly narrow CI. Although the prediction time is the slowest, it has the best optimized r-squared score at ~>0.7. If time is also an issue, I would pick SVM as it has the second highest r-squared improvement for drastically less time compared with Random Forrest. I would settle for KNN if interpretability is also an issue.

2.4.d (3 points)

Prediction time, as well as model complexity are other factors to consider when selecting the optimal model(s). Given each model's prediction time, model complexity, and R2 estimates, which model(s) would you choose, and why?

For my criteria, I would prefer a high r-squared to cover a large portion of variability. I would like to have a lower prediction time to suggest less computational burden. Given a large amount of data, I would prefer the model to be a little bit complex. A simple model is welcome for easier interpretability but I would like to weight more on the models performance, of course with considering the risk of over-fitting. In this case, I choose SVM.

Problem 2.5 (30 points)

Brodnjak-Vonina et al. (2005) develop a methodology for food laboratories to determine the type of oil from a sample. In their procedure, they used a gas chromatograph (an instrument that separates chemicals in a sample) to measure seven different fatty acids in an oil. These measurements would then be used to predict the type of oil in a food sample. To create their model, they used 96 samples of seven types of oils.

```r
library(caret)
data(oil)
```

2.5.a (10 points)

Use the sample function in base R to create a completely random sample of 60 oils. How closely do the frequencies of the random sample match the original samples? Repeat this procedure several times to understand the variation in the sampling process.

```r
set.seed(503)
# Create a random sample of 60 oils
OilType_df <- as.data.frame(oilType)
original_freq <- table(OilType_df$oilType)

num_repeats <- 3
for (i in 1:num_repeats) {
  random_indices <- sample(nrow(OilType_df), 60, replace = FALSE)
  random_sample <- OilType_df[random_indices, , drop = FALSE]
  random_freq <- table(random_sample$oilType)
  accuracy <- sum(random_freq == original_freq) / length(original_freq) * 100

  # Print the comparison of frequencies and accuracy
  cat("Random Sample", i, "vs. Original Frequencies", "Accuracy:", accuracy, "%\n")
  print(cbind(Random_Sample = random_freq, Original = original_freq))
  cat("\n")
}
```

```
Random Sample 1 vs. Original Frequencies Accuracy: 14.28571 %
  Random_Sample Original
A            22       37
B            14       26
C             0        3
D             5        7
E             9       11
F            10       10
G             0        2

Random Sample 2 vs. Original Frequencies Accuracy: 14.28571 %
  Random_Sample Original
A            28       37
B            14       26
C             3        3
D             4        7
E             4       11
F             6       10
G             1        2

Random Sample 3 vs. Original Frequencies Accuracy: 14.28571 %
  Random_Sample Original
A            23       37
B            16       26
```

```
C               3        3
D               4        7
E               7       11
F               7       10
G               0        2
```

Basing on the accuracy score, the results have pretty low accuracy compared to the original frequency. The generated samples are randomized from the original distribution and can sometime be completely different in sequence (i.e. Random Sample 5 - not shown).

2.5.b (5 points)

Use the caret package function `createDataPartition` to create a stratified random sample. How does this compare to the completely random samples?

```r
# Create stratified random sample using createDataPartition
set.seed(503)
outcome <- oilType

stratified_indices <- createDataPartition(outcome, p = 0.625, list = FALSE)
stratified_sample <- OilType_df[stratified_indices, , drop = FALSE]
stratified_freq <- table(stratified_sample$oilType)
accuracy <- sum(stratified_freq == original_freq) / length(original_freq) * 100

# Print the comparison of frequencies for stratified random sample along with accuracy
cat("Stratified Random Sample vs. Original Frequencies", "Accuracy:", accuracy,"%\n")
```

```
Stratified Random Sample vs. Original Frequencies Accuracy: 14.28571 %
```

```r
print(cbind(Stratified_Sample = stratified_freq, Original = original_freq))
```

```
  Stratified_Sample Original
A                24       37
B                17       26
C                 2        3
D                 5        7
E                 7       11
F                 7       10
G                 2        2
```

Unlike the complete random sample that have multiple iteration, stratified random sample created a single output of random samples. When set to the same number of output as complete random sample, the difference of the stratified random sample is the same at 14.2871% accuracy. Meaning that they are preforming relatively the same when randomly sampling.

2.5.c (5 points)

With such a small sample size, what are the options for determining performance of the model? Should a test set be used?

The challenge with small sample size is proper representation of the underlying population. Choosing to apply techniques like cross validation enhances the models performance through repeated use of the available data. In a k-fold validation, the data set is split into k that's equally sized segments. The model gets trained on iteration of K-n folds K times. The performance is then evaluated through tools like R^2 and RMSE that's averaged across the k iterations. Use of test sets are less ideal but stratified random sampling can be used strategically. Observations are stratified before splitting to train and test sets.

2.5.d (10 points)

One method for understanding the uncertainty of a test set is to use a confidence interval. To obtain a confidence interval for the overall accuracy, the based R function binom.test can be used. It requires the user to input the number of samples and the number correctly classified to calculate the interval. For example, suppose a test set sample of 20 oil samples was set aside and 76 were used for model training. For this test set size and a model that is about 80 % accurate (16 out of 20 correct), the confidence interval would be computed using p <- 0.8, n <- 20, binom.test(p * n, n)

Try different sample sizes and accuracy rates to understand the trade-off between the uncertainty in the results, the model performance, and the test set size.

```r
# Define CI (binom.test) function
calculate_ci_width <- function(p, n) {
  t_result <- binom.test(p * n, n)
  ci_width <- t_result$conf.int[2] - t_result$conf.int[1]
  return(ci_width)}

accuracies <- c(0.1, 0.5, 0.9)
  #0.1 * 1:9, going to shorten the results
sample_sizes <- c(20, 640, 5120)
  #20 * 2^(0:8)
```

```r
# Explore different scenarios
for (acc in accuracies) {
  for (size in sample_sizes) {
    ci_width <- calculate_ci_width(acc, size)
    cat("Accuracy:", acc, "| Sample Size:", size, "| CI Width:", ci_width, "\n")}}
```

```
Accuracy: 0.1 | Sample Size: 20 | CI Width: 0.3046342
Accuracy: 0.1 | Sample Size: 640 | CI Width: 0.04803565
Accuracy: 0.1 | Sample Size: 5120 | CI Width: 0.01662982
Accuracy: 0.5 | Sample Size: 20 | CI Width: 0.4560843
Accuracy: 0.5 | Sample Size: 640 | CI Width: 0.07888349
Accuracy: 0.5 | Sample Size: 5120 | CI Width: 0.02758006
Accuracy: 0.9 | Sample Size: 20 | CI Width: 0.3046342
Accuracy: 0.9 | Sample Size: 640 | CI Width: 0.04803565
Accuracy: 0.9 | Sample Size: 5120 | CI Width: 0.01662982
```

As accuracy increase, CI width narrows/decrease. This makes sense as higher accuracy means the model is performing better and so uncertainty in estimations goes down. When sample size increases, the pattern tends to decrease for CI width as well. This makes sense as well as larger sample size results to a more precise accuracy and so CI width goes down in value. Overall, high accuracy with large sample size leads to narrowest CI. Low accuracy with small sample size leads to wider CI.