Lecture 1

Introduction to the course
Formulas for statistics and metrics

The goal for this course is to provide an introduction to machine learning that is different from the things that you learned in MTH 325 or Data Mining. For this reason, we will be looking at some specific machine learning algorithms in greater depth, code up these algorithms from scratch, so that we have the ability to customize those algorithms as we please. While functions built-into packages are the emphasis of Applied Stats and Data Mining, sometimes algorithms we want to use aren't in specific packages (yet), or they don't allow for the kind of customization we might want or need. We will begin by coding up some basic components, write our own functions and compare them to built-in functions. We will then use our code elements to develop a dozen or so machine learning algorithms and customize those algorithms. These processes will also allow us to apply consistent metrics across various package-based algorithms that may not have all the same metrics built-in, or which are structured differently that don't allow the use of built-in functions from standard packages.

To that end, we are going to start with some basics. Some of this may be review, but we'll also be introducing new concepts along the way.

Mean:

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

```
my_mean <- function(x) {
  sum(x) / length(x)
}

# Example usage
data <- c(1, 2, 3, 4, 5)
my_mean(data)
```

You can apply this same operation to a column of a data frame by extracting a column: data<-mtcars$mpg.

You can use this same function to calculate the proportion of 1's in a column if the column is already coded as 0/1. The average of the 0's and 1's will be the proportion of 1's, since the sum is the number of 1s, and the denominator is the total number in the set: data<-mtcars$am.

Median:

```
my_median <- function(x) {
  sorted_x <- sort(x)
  n <- length(x)
  if (n %% 2 == 1) {
    sorted_x[(n + 1) / 2]
  } else {
    (sorted_x[n / 2] + sorted_x[n / 2 + 1]) / 2
  }
```

```
}

# Example usage
my_median(data)
```

Try this example a column from mtcars.

Standard Deviation:

$$s = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

```
my_sd <- function(x) {
  m <- my_mean(x)
  sqrt(sum((x - m)^2) / (length(x) - 1))
}

# Example usage
my_sd(data)
```

Notice that this is the sample standard deviation function, which is the same as the built-in sd() function in base R. However, there is no built-in population standard deviation function. Modify this function to calculate the population standard deviation instead.

To calculate the variance, the function structure is the same, but we would omit the sqrt().

Summary Tables:
If we want to create a summary table and the values are not already 0/1, then we need to work a little harder.

```
my_table <- function(x) {
  unique_vals <- unique(x)
  freq <- sapply(unique_vals, function(val) sum(x == val))
  names(freq) <- unique_vals
  return(freq)
}

# Example usage
categorical_data <- c("A", "B", "A", "C", "B", "A")
my_table(categorical_data)
```

How does this function do with mtcars$cyl?

Compare to the built-in function:

```
table(mtcars$cyl)
```

We can also aggregate by group:

```
my_aggregate_mean <- function(data, group) {
  unique_groups <- unique(group)
  group_means <- sapply(unique_groups, function(g) my_mean(data[group == g]))
  names(group_means) <- unique_groups
  return(group_means)
}

# Example usage
values <- c(5, 10, 15, 20, 25, 30)
groups <- c("A", "A", "B", "B", "C", "C")
my_aggregate_mean(values, groups)
```

Compare this to the built-in function:

```
aggregate(mpg ~ cyl, data = mtcars, FUN = my_mean)
```

We can subset our data for calculation as well:

```
mtcars[mtcars$cyl == 6, ]
mtcars[1:5, ]
```

Two-sample t-test:

$$t = \frac{(\overline{x}_1 - \overline{x}_2) - (\mu_1 - \mu_2)}{s_p \sqrt{\frac{1}{n_1} + \frac{1}{n_2}}}$$

```
my_t_test <- function(x, y) {
  mean_diff <- my_mean(x) - my_mean(y)
  pooled_sd <- sqrt(((length(x) - 1) * my_sd(x)^2 + (length(y) - 1) * my_sd(y)^2) / (length(x) +
length(y) - 2))
  t_stat <- mean_diff / (pooled_sd * sqrt(1 / length(x) + 1 / length(y)))
  return(t_stat)
}
#this assumes the difference is assumed to be zero.

# Example usage
group1 <- c(5, 6, 7, 8, 9)
group2 <- c(1, 2, 3, 4, 5)
my_t_test(group1, group2)
```

Correlation:

$$r = \frac{1}{n-1} \sum \left(\frac{x - \overline{x}}{s_x} \right) \left(\frac{y - \overline{y}}{s_y} \right)$$

```
my_correlation <- function(x, y) {
  cov_xy <- sum((x - my_mean(x)) * (y - my_mean(y))) / (length(x) - 1)
```

```
  cor_xy <- cov_xy / (my_sd(x) * my_sd(y))
  return(cor_xy)
}

# Example usage
var1 <- c(1, 2, 3, 4, 5)
var2 <- c(5, 4, 3, 2, 1)
my_correlation(var1, var2)
```

Compare to the built-in function:

```
cor(mtcars$hp, mtcars$mpg)
```

All these examples were applied to a single vector, or column of a data frame. What if we want to apply the same operation to all the columns in a data frame?

Some examples:

```
#means
my_mean <- function(x) {
  sum(x) / length(x)
}

apply_means <- function(data) {
  sapply(data, my_mean)
}

# Example usage
data("mtcars")
apply_means(mtcars)

#median
my_median <- function(x) {
  sorted_x <- sort(x)
  n <- length(x)
  if (n %% 2 == 1) {
    sorted_x[(n + 1) / 2]
  } else {
    (sorted_x[n / 2] + sorted_x[n / 2 + 1]) / 2
  }
}

apply_medians <- function(data) {
  sapply(data, my_median)
}

# Example usage
apply_medians(mtcars)

#standard deviation (sample)
```

```r
my_sd <- function(x) {
  m <- my_mean(x)
  sqrt(sum((x - m)^2) / (length(x) - 1))
}

apply_sds <- function(data) {
  sapply(data, my_sd)
}

# Example usage
apply_sds(mtcars)

#Summary table
my_table <- function(x) {
  unique_vals <- unique(x)
  freq <- sapply(unique_vals, function(val) sum(x == val))
  names(freq) <- unique_vals
  return(freq)
}

apply_tables <- function(data) {
  lapply(data, function(column) {
    if (is.factor(column) || is.character(column)) {
      my_table(column)
    } else {
      NULL
    }
  })
}

# Example usage
apply_tables(mtcars)

#Aggregate
my_aggregate_mean <- function(data, group) {
  unique_groups <- unique(group)
  group_means <- sapply(unique_groups, function(g) my_mean(data[group == g]))
  names(group_means) <- unique_groups
  return(group_means)
}

apply_aggregate_means <- function(data, group_column) {
  data <- data.frame(data)
  groups <- data[[group_column]]
  data <- data[ , !(names(data) %in% group_column)]
  lapply(data, function(column) my_aggregate_mean(column, groups))
}
```

```r
# Example usage
apply_aggregate_means(mtcars, "cyl")

#T-tests
# Load data
data("mtcars")

# Define custom mean function
my_mean <- function(x) {
  sum(x) / length(x)
}

# Define custom variance function
my_variance <- function(x) {
  m <- my_mean(x)
  sum((x - m)^2) / (length(x) - 1)
}

# Define custom t-test function
my_t_test <- function(x, y) {
  mean_diff <- my_mean(x) - my_mean(y)
  pooled_sd <- sqrt(((length(x) - 1) * my_variance(x) + (length(y) - 1) * my_variance(y)) / (length(x) +
length(y) - 2))
  t_stat <- mean_diff / (pooled_sd * sqrt(1 / length(x) + 1 / length(y)))
  return(t_stat)
}

# Function to apply t-test to each column in the data frame
apply_t_tests <- function(data, group_column) {
  data <- data.frame(data)
  groups <- data[[group_column]]
  data <- data[ , !(names(data) %in% group_column)]
  unique_groups <- unique(groups)

  if (length(unique_groups) != 2) {
    stop("The group column must have exactly 2 unique values.")
  }

  group1 <- data[groups == unique_groups[1], ]
  group2 <- data[groups == unique_groups[2], ]

  sapply(names(data), function(column) my_t_test(group1[[column]], group2[[column]]))
}

# Subset mtcars to include only cars with 4 and 6 cylinders
subset_mtcars <- mtcars[mtcars$cyl %in% c(4, 6), ]

# Example usage
```

```
apply_t_tests(subset_mtcars, "cyl")
my_correlation <- function(x, y) {
  cov_xy <- sum((x - my_mean(x)) * (y - my_mean(y))) / (length(x) - 1)
  cor_xy <- cov_xy / (my_sd(x) * my_sd(y))
  return(cor_xy)
}

#Correlations
apply_correlations <- function(data) {
  n <- ncol(data)
  cor_matrix <- matrix(NA, n, n)
  colnames(cor_matrix) <- rownames(cor_matrix) <- names(data)
  for (i in 1:(n - 1)) {
    for (j in (i + 1):n) {
      cor_matrix[i, j] <- cor_matrix[j, i] <- my_correlation(data[[i]], data[[j]])
    }
  }
  return(cor_matrix)
}

# Example usage
apply_correlations(mtcars)
```

Some other statistical metrics:
Skewness:

$$\text{Skewness} = \frac{n}{(n-1)(n-2)} \times \sum \left(\frac{x_i - \bar{x}}{s}\right)^3$$

Where:

- n is the number of observations
- x_i is each individual observation
- $\bar{x}$ is the mean of the observations
- s is the standard deviation of the observations.

```
my_skewness <- function(x) {
  n <- length(x)
  mean_x <- mean(x)
  sd_x <- sd(x)
  skewness <- (n/((n-1)*(n-2))) * sum(((x - mean_x) / sd_x)^3)
  return(skewness)
}

# Example usage
data <- c(1, 2, 3, 4, 5, 6, 7, 8, 9)
```

```
my_skewness(data)
```

Kurtosis:

$$K = \frac{1}{N}\sum_{i=1}^{N}(\frac{x_i - \mu}{\sigma})^4$$

```
#this formula uses an alternate formula that is less computationally intensive, but a messier formula.
my_kurtosis <- function(x) {
  n <- length(x)
  mean_x <- mean(x)
  sd_x <- sd(x)
  kurtosis <- (n*(n+1))/((n-1)*(n-2)*(n-3)) * sum(((x - mean_x) / sd_x)^4) - (3*(n-1)^2)/((n-2)*(n-3))
  return(kurtosis)
}

# Example usage
my_kurtosis(data)
```

Coefficient of variation:

```
my_cv <- function(x) {
  mean_x <- mean(x)
  sd_x <- sd(x)
  cv <- sd_x / mean_x
  return(cv)
}

# Example usage
my_cv(data)
```

Mean Absolute Deviation (MAD):

$$MAD = \frac{\sum |x_i - \overline{x}|}{n}$$

```
my_mad <- function(x) {
  mean_x <- mean(x)
  mad <- mean(abs(x - mean_x))
  return(mad)
}

# Example usage
my_mad(data)
```

Harmonic Mean:

$$\text{Harmonic Mean} = \frac{n}{\sum\left[\frac{1}{x_i}\right]}$$

```
my_harmonic_mean <- function(x) {
  harmonic_mean <- length(x) / sum(1 / x)
  return(harmonic_mean)
}

# Example usage
my_harmonic_mean(data)
```

Geometric Mean:

$$\overline{X}_{geom} = \sqrt[n]{\prod_{i=1}^{n} x_i} = \sqrt[n]{x_1.x_1.\ldots\ldots x_n}$$

```
my_geometric_mean <- function(x) {
  geometric_mean <- prod(x)^(1/length(x))
  return(geometric_mean)
}

# Example usage
my_geometric_mean(data)
```

Median Absolute Deviation:

```
my_median_absolute_deviation <- function(x) {
  med_x <- median(x)
  mad <- median(abs(x - med_x))
  return(mad)
}

# Example usage
my_median_absolute_deviation(data)
```

Trimmed Mean:

The trimmed mean is a compromise between the mean and the median, by removing extreme values from both ends of the dataset. A trim of 10% removes 10% from the top and the bottom of the dataset (thus, a trim of 50% leaves no data left to average).

```
my_trimmed_mean <- function(x, trim = 0.1) {
  sorted_x <- sort(x)
  n <- length(x)
  k <- floor(trim * n)
  trimmed_x <- sorted_x[(k + 1):(n - k)]
  trimmed_mean <- mean(trimmed_x)
  return(trimmed_mean)
}
```

```
# Example usage
my_trimmed_mean(data, 0.1)
```

Some of these variation formulas we will see again when we do residual analysis with regression.