CSDA Coursework R scripts:

Task 1 question 1:

```
# manual GenBank CDS parser


parse_gb_cds <- function(gb_file){
  lines      <- readLines(gb_file)
  # find FEATURES section
  feat_start <- grep("^FEATURES", lines)
  origin_lin <- grep("^ORIGIN",   lines)
  feats      <- lines[(feat_start+1):(origin_lin-1)]

  # keep only the lines that define CDS
  cds_lines  <- feats[grepl("^\\s{5}CDS", feats)]

  # pull out the raw location token
  locs_raw   <- sub("^\\s*CDS\\s+([^\\s]+).*", "\\1", cds_lines)

  # strip out outer wrappers, e.g. join(), complement()
  locs_clean <- gsub("join\\(|complement\\(|\\)", "", locs_raw)

  # split on commas if there are multiple parts, then parse
  lapply(strsplit(locs_clean, ","), function(parts){
    rng <- as.numeric(unlist(strsplit(parts, "\\.\\.")))
    matrix(rng, ncol=2, byrow=TRUE)  # each row is a start–end
  })
}
```

```
gb_file <- "C:/Users/LEGION/Downloads/ku574722.gb"
cds_regions <- parse_gb_cds(gb_file)

# flatten to a simple list of length-2 numeric vectors:
cds_list <- do.call(c, lapply(cds_regions, function(mat){
  apply(mat, 1, as.numeric)
}))
print(cds_list)

# Compute nucleotide frequencies in coding vs. non-coding
library(seqinr)
seq <- toupper(read.fasta("C:/Users/LEGION/Downloads/ku574722.fasta")[[1]])
L   <- length(seq)

# Build a logical mask of coding positions
is_coding <- rep(FALSE, L)
coords    <- c(cds_list)  # or cds_mat as below
cds_mat   <- matrix(coords, ncol=2, byrow=TRUE)
for(i in seq_len(nrow(cds_mat))) {
  s <- cds_mat[i,1]; e <- min(cds_mat[i,2], L)
  is_coding[s:e] <- TRUE
}

# Extract coding and noncoding letters
coding    <- seq[is_coding]
```

```
noncoding <- seq[!is_coding]

# Count A/C/G/T frequencies
freqs <- function(x) {
  t <- table(x)
  f <- setNames(numeric(4), c("A","C","G","T"))
  f[names(t)] <- as.numeric(t)
  f / sum(f)
}

p_coding    <- freqs(coding)
p_noncoding <- freqs(noncoding)

print(p_coding)
print(p_noncoding)

# Define & initialize 2-state HMM
library(HMM)

states  <- c("Coding","NonCoding")
symbols <- c("A","C","G","T")

# Emission matrix
emissionProbs <- rbind(
  Coding    = p_coding,
  NonCoding = p_noncoding
```

```
)

# Transition matrix with switch-prob = 0.033
p <- 0.033
transProbs <- matrix(c(1-p, p,
                       p,   1-p),
                     nrow=2, byrow=TRUE,
                     dimnames=list(states, states))

startProbs <- c(Coding=0.5, NonCoding=0.5)

hmm <- initHMM(States=states,
               Symbols=symbols,
               startProbs=startProbs,
               transProbs=transProbs,
               emissionProbs=emissionProbs)

# Run the Viterbi algorithm
obs <- seq
vpath <- viterbi(hmm, obs)

# Compare predictions vs. true annotation
true_state <- ifelse(is_coding, "Coding", "NonCoding")
cm <- table(Predicted=vpath, True=true_state)
print(cm)
```

```
accuracy <- mean(vpath == true_state)
cat("Accuracy:", round(accuracy, 4), "\n")

# Experiment with different switch probabilities
for(p in c(0.033, 0.1, 0.2, 0.5)) {
  transProbs[] <- c(1-p, p, p, 1-p)
  hmm <- initHMM(states, symbols, startProbs, transProbs, emissionProbs)
  vpath <- viterbi(hmm, obs)
  acc   <- mean(vpath == true_state)
  cat("p =", p, "→ accuracy =", round(acc,4), "\n")
}
```

Task 1 question 2:

```
# Load the DNA sequence

fasta_file <- "C:/Users/LEGION/Downloads/ku574722.fasta"
dna_sequence <- read.fasta(fasta_file)[[1]]
cat("Sequence loaded. Total length:", length(dna_sequence), "nucleotides\n")

# Define functions for entropy calculation
get_triplets <- function(seq) {
  n <- length(seq)
  triplets <- character(n - 2)
  for (i in 1:(n - 2)) {
    triplets[i] <- paste0(seq[i], seq[i+1], seq[i+2])
  }
  return(triplets)
}

# Function to calculate Shannon entropy
calculate_entropy <- function(seq) {
  triplets <- get_triplets(seq)

  triplet_counts <- table(triplets)
  triplet_freqs <- triplet_counts / sum(triplet_counts)

  entropy <- -sum(triplet_freqs * log2(triplet_freqs))
```

```
  return(entropy)
}

# Function to calculate maximum possible entropy for triplets
calculate_max_entropy <- function() {
  max_entropy <- log2(64)
  return(max_entropy)
}

# Calculate entropy for sliding windows
window_size <- 100
step_size <- 100

num_windows <- floor((length(dna_sequence) - window_size) / step_size) + 1
cat("Number of 100-nucleotide windows:", num_windows, "\n")

# Initialize vectors to store results
window_entropies <- numeric(num_windows)
window_positions <- numeric(num_windows)

# Process each window
for (i in 1:num_windows) {
  start_pos <- (i - 1) * step_size + 1
  end_pos <- start_pos + window_size - 1

  # Make sure we don't exceed sequence length
```

```
  if (end_pos > length(dna_sequence)) {
    end_pos <- length(dna_sequence)
  }

  window_seq <- dna_sequence[start_pos:end_pos]

  # Calculate entropy for this window
  if (length(window_seq) >= 3) {
    window_entropies[i] <- calculate_entropy(window_seq)
    window_positions[i] <- start_pos
  }
}

# Calculate maximum possible entropy
max_entropy <- calculate_max_entropy()
cat("Maximum possible entropy for triplets:", max_entropy, "\n")

# Print summary statistics
cat("Mean entropy:", mean(window_entropies, na.rm = TRUE), "\n")
cat("Min entropy:", min(window_entropies, na.rm = TRUE), "\n")
cat("Max entropy:", max(window_entropies, na.rm = TRUE), "\n")

# Load coding regions from GenBank file
parse_gb_cds <- function(gb_file){
  lines      <- readLines(gb_file)
  feat_start <- grep("^FEATURES", lines)
```

```
  origin_lin <- grep("^ORIGIN",   lines)
  feats      <- lines[(feat_start+1):(origin_lin-1)]

  cds_lines  <- feats[grepl("^\\s{5}CDS", feats)]

  locs_raw   <- sub("^\\s*CDS\\s+([^\\s]+).*", "\\1", cds_lines)

  locs_clean <- gsub("join\\(|complement\\(|\\)", "", locs_raw)

  lapply(strsplit(locs_clean, ","), function(parts){
    rng <- as.numeric(unlist(strsplit(parts, "\\.\\.")))
    matrix(rng, ncol=2, byrow=TRUE)
  })
}

# Load CDS regions
gb_file <- "C:/Users/LEGION/Downloads/ku574722.gb"
cds_regions <- parse_gb_cds(gb_file)

# Convert to flat list and then to matrix
cds_list <- do.call(c, lapply(cds_regions, function(mat){
  apply(mat, 1, as.numeric)  # might need a little unlisting
}))
cds_mat <- matrix(cds_list, ncol=2, byrow=TRUE)

# Plot entropy and coding regions
```

```
entropy_df <- data.frame(
  Position = window_positions,
  Entropy = window_entropies
)

# Create a coding region mask for plotting
is_coding <- rep(FALSE, length(dna_sequence))
for(i in 1:nrow(cds_mat)) {
  s <- cds_mat[i,1]
  e <- min(cds_mat[i,2], length(dna_sequence))
  is_coding[s:e] <- TRUE
}

# Prepare data for plotting coding regions
window_is_coding <- logical(num_windows)
for (i in 1:num_windows) {
  start_pos <- (i - 1) * step_size + 1
  end_pos <- start_pos + window_size - 1

  # Consider a window as coding if at least 50% of it is in a coding region
  window_is_coding[i] <- mean(is_coding[start_pos:end_pos]) >= 0.5
}

entropy_df$IsCoding <- window_is_coding

# Plot results
```

```
jpeg("entropy_plot.jpg", width = 1200, height = 800, quality = 100)
par(mfrow = c(2, 1))

# Plot 1: Entropy values along the sequence
plot(entropy_df$Position, entropy_df$Entropy,
     type = "l", col = "blue",
     xlab = "Position in sequence",
     ylab = "Entropy",
     main = "Entropy of DNA Sequence Using Triplets",
     ylim = c(0, max_entropy * 1.1))
abline(h = max_entropy, col = "red", lty = 2)
text(max(window_positions) * 0.1, max_entropy * 1.05,
     paste("Maximum possible entropy:", round(max_entropy, 3)))

# Plot 2: Coding regions vs. Entropy
plot(entropy_df$Position, entropy_df$Entropy,
     type = "l", col = "blue",
     xlab = "Position in sequence",
     ylab = "Entropy",
     main = "Entropy vs. Coding Regions",
     ylim = c(0, max_entropy * 1.1))

# Add colored background for coding regions
for (i in 1:nrow(cds_mat)) {
  start_pos <- cds_mat[i, 1]
  end_pos <- cds_mat[i, 2]
```

```
  rect(start_pos, 0, end_pos, max_entropy * 1.1,
       col = rgb(1, 0, 0, 0.2), border = NA)
}

# Add legend
legend("topright",
       legend = c("Entropy", "Max Entropy", "Coding Regions"),
       col = c("blue", "red", rgb(1, 0, 0, 0.2)),
       lty = c(1, 2, NA),
       pch = c(NA, NA, 15),
       pt.cex = 2)

dev.off()

# Statistical analysis to compare
# Compare entropy in coding vs non-coding regions
coding_entropy <- entropy_df$Entropy[entropy_df$IsCoding]
noncoding_entropy <- entropy_df$Entropy[!entropy_df$IsCoding]

cat("\nComparing entropy in coding vs. non-coding regions:\n")
cat("Mean entropy in coding regions:", mean(coding_entropy, na.rm = TRUE), "\n")
cat("Mean entropy in non-coding regions:", mean(noncoding_entropy, na.rm = TRUE),
"\n")

# Perform t-test
ttest_result <- t.test(coding_entropy, noncoding_entropy)
```

```
cat("\nT-test comparing entropy in coding vs. non-coding regions:\n")
print(ttest_result)

# Calculate correlation between entropy and coding status
point_biserial <- cor(entropy_df$Entropy, as.numeric(entropy_df$IsCoding),
                      use = "complete.obs")
cat("\nCorrelation between entropy and coding status:", point_biserial, "\n")

# Examine entropy variability
# Calculate rolling average to smooth entropy values
window_size_smooth <- 5
rolling_avg <- numeric(length(window_entropies))

for (i in 1:length(window_entropies)) {
  start_idx <- max(1, i - floor(window_size_smooth/2))
  end_idx <- min(length(window_entropies), i + floor(window_size_smooth/2))
  rolling_avg[i] <- mean(window_entropies[start_idx:end_idx], na.rm = TRUE)
}

# Calculate entropy gradient (rate of change)
entropy_gradient <- c(0, diff(rolling_avg))

# Identify regions of significant entropy change
threshold <- quantile(abs(entropy_gradient), 0.95, na.rm = TRUE)
significant_changes <- which(abs(entropy_gradient) > threshold)
```

```
cat("\nPositions with significant entropy changes:\n")
if (length(significant_changes) > 0) {
  for (pos in significant_changes) {
    cat("Position", window_positions[pos],
        "- Entropy change:", round(entropy_gradient[pos], 4),
        "- In coding region:", entropy_df$IsCoding[pos], "\n")
  }
} else {
  cat("No positions with significant entropy changes found.\n")
}
```

Task 2:

```
# Task 2: EM algorithm for vaccine efficacy


# Load needed libraries
library(dplyr)


# Input data: number who avoided and got flu at each site
df <- data.frame(
  site  = 1:5,
  avoid = c(92, 88, 23, 77, 39),
  got   = c(11, 22, 67, 23, 42)
)


# EM function definition
em_vaccine <- function(a, g,
                       init = list(pA = 0.9, pB = 0.5, pi = 0.5),
                       tol  = 1e-8,
                       maxit = 1000) {
  pA <- init$pA
  pB <- init$pB
  pi <- init$pi
  n  <- a + g
  J  <- length(a)

  for (it in seq_len(maxit)) {
```

```
    # E-step: posterior weight that site j used vaccine A
    likA <- pi      * dbinom(a, size = n, prob = pA)
    likB <- (1 - pi) * dbinom(a, size = n, prob = pB)
    w    <- likA / (likA + likB)
    w[1] <- 1  # Site 1 known to be A

    # M-step: update parameters
    pi_new <- mean(w)
    pA_new <- sum(w * a)           / sum(w * n)
    pB_new <- sum((1 - w) * a)     / sum((1 - w) * n)

    # convergence check
    if (max(abs(c(pA_new - pA, pB_new - pB, pi_new - pi))) < tol) {
      pA <- pA_new; pB <- pB_new; pi <- pi_new
      break
    }

    pA <- pA_new; pB <- pB_new; pi <- pi_new
  }

  list(pA = pA, pB = pB, pi = pi, w = w, iterations = it)
}

# Run EM on my data
res <- em_vaccine(df$avoid, df$got,
                  init = list(pA = 92/103, pB = 0.5, pi = 0.5))
```

```
# Attach posteriors and most-likely vaccine to the data frame
df2 <- df %>%
  mutate(
    postA   = res$w,
    vaccine = ifelse(postA > 0.5, "A", "B")
  )

# Print results
cat("\nEM converged in", res$iterations, "iterations\n")
cat(sprintf("Estimated pA = %.4f, pB = %.4f, pi = %.4f\n\n",
            res$pA, res$pB, res$pi))

print(df2)
```

Task 3:

```
# Load required libraries
library(readxl)
library(mice)
library(ggplot2)
library(cluster)
library(factoextra)
library(fpc)

# Read the diabetes dataset
diabetes_data <- read_excel("C:/Users/LEGION/Downloads/diabetes_dataset.xlsx")

# Check for missing insulin values
sum(diabetes_data$Insulin == 0)

# Imputation for missing insulin values
# First, I will replace zeros with NA for insulin
diabetes_data$Insulin[diabetes_data$Insulin == 0] <- NA

# Perform regression imputation
# We'll use predictive mean matching method in mice package
imputation_model <- mice(diabetes_data, method = "pmm", m = 5, seed = 123)
diabetes_imputed <- complete(imputation_model, 1)

# Verify imputation
```

```
cat("Before imputation - records with missing insulin:",
sum(is.na(diabetes_data$Insulin)), "\n")

cat("After imputation - records with missing insulin:",
sum(is.na(diabetes_imputed$Insulin)), "\n")


# Randomly select 150 patients from the imputed dataset

set.seed(456)

selected_indices <- sample(1:nrow(diabetes_imputed), 150)

diabetes_subset <- diabetes_imputed[selected_indices, ]


# Prepare data for clustering (normalize the data)

# Select variables for clustering (all except Outcome)

cluster_vars <- c("Glucose", "Pregnancies", "BloodPressure", "SkinThickness",
                  "Insulin", "BMI", "DiabetesPedigreeFunction", "Age")


# Scale the variables to have mean 0 and standard deviation 1

diabetes_scaled <- scale(diabetes_subset[, cluster_vars])


# Perform k-means clustering with k=2

set.seed(789)

km_result <- kmeans(diabetes_scaled, centers = 2, nstart = 25)


diabetes_subset$Cluster <- km_result$cluster


# Evaluate clustering using external measures

# Create confusion matrix

conf_matrix <- table(diabetes_subset$Cluster, diabetes_subset$Outcome)
```

```
print("Confusion Matrix (Clusters vs Diabetes Outcome):")
print(conf_matrix)

# Calculate F-measure
# Determine cluster assignment based on majority class
cluster_mapping <- sapply(1:2, function(i) {
  if(sum(conf_matrix[i,2])/sum(conf_matrix[i,]) > 0.5) 2 else 1
})

# Recalculate confusion matrix with mapped clusters
predicted_outcome <- ifelse(diabetes_subset$Cluster == 1,
                            cluster_mapping[1],
                            cluster_mapping[2]) - 1

# Compute confusion matrix
conf_stats <- confusionMatrix(factor(predicted_outcome),
                              factor(diabetes_subset$Outcome))

# Display the results
print("Clustering Performance Metrics:")
print(conf_stats)

# Calculate F-measure manually
precision <- conf_stats$byClass["Pos Pred Value"]
recall <- conf_stats$byClass["Sensitivity"]
f_measure <- 2 * precision * recall / (precision + recall)
```

```
print(paste("F-measure:", round(f_measure, 4)))

# Perform silhouette analysis for internal validation
sil <- silhouette(km_result$cluster, dist(diabetes_scaled))
sil_avg <- mean(sil[, 3])
print(paste("Average Silhouette Width:", round(sil_avg, 4)))

# Dunn index - alternative internal validation measure
# Higher values indicate better clustering
dunn_index <- function(clusters, dist_matrix) {
  k <- length(unique(clusters))

  # Calculate minimum inter-cluster distance
  min_inter <- Inf
  for(i in 1:(k-1)) {
    for(j in (i+1):k) {
      points_i <- which(clusters == i)
      points_j <- which(clusters == j)

      # Calculate distances between points in different clusters
      for(pi in points_i) {
        for(pj in points_j) {
          dist_ij <- dist_matrix[pi, pj]
          if(dist_ij < min_inter) {
            min_inter <- dist_ij
          }
```

```
      }
    }
  }
}

# Calculate maximum intra-cluster distance
max_intra <- 0
for(i in 1:k) {
  points_i <- which(clusters == i)
  if(length(points_i) > 1) {
    for(pi in 1:(length(points_i)-1)) {
      for(pj in (pi+1):length(points_i)) {
        dist_ij <- dist_matrix[points_i[pi], points_i[pj]]
        if(dist_ij > max_intra) {
          max_intra <- dist_ij
        }
      }
    }
  }
}

# Calculate Dunn index
if(max_intra > 0) {
  return(min_inter / max_intra)
} else {
  return(NA)
```

```
  }
}

# Calculate distance matrix
dist_matrix <- as.matrix(dist(diabetes_scaled))
dunn_idx <- dunn_index(km_result$cluster, dist_matrix)
print(paste("Dunn Index:", round(dunn_idx, 4)))

# Visualize the clustering results
# PCA plot with clusters
fviz_cluster(km_result, data = diabetes_scaled,
             palette = c("#FC8D62", "#66C2A5"),
             geom = "point",
             ellipse.type = "convex",
             ggtheme = theme_minimal(),
             main = "Cluster Plot of Diabetes Data")

# Plot showing clusters vs actual outcome
ggplot(diabetes_subset, aes(x = Glucose, y = BMI, color = factor(Outcome), shape =
factor(Cluster))) +
  geom_point(size = 3) +
  scale_color_manual(values = c("#377EB8", "#E41A1C"), name = "Diabetes Status",
                     labels = c("Negative", "Positive")) +
  scale_shape_manual(values = c(16, 17), name = "Cluster") +
  theme_minimal() +
  labs(title = "Clustering Results vs Actual Diabetes Status",
```

```
       x = "Glucose Level",
       y = "BMI")

# Additional cluster evaluation to find optimal number of clusters
# Elbow method
fviz_nbclust(diabetes_scaled, kmeans, method = "wss") +
  geom_vline(xintercept = 2, linetype = 2) +
  labs(subtitle = "Elbow Method")

# Silhouette method
fviz_nbclust(diabetes_scaled, kmeans, method = "silhouette") +
  labs(subtitle = "Silhouette Method")
```