RSDexport

Yichen Han

2024-08-10

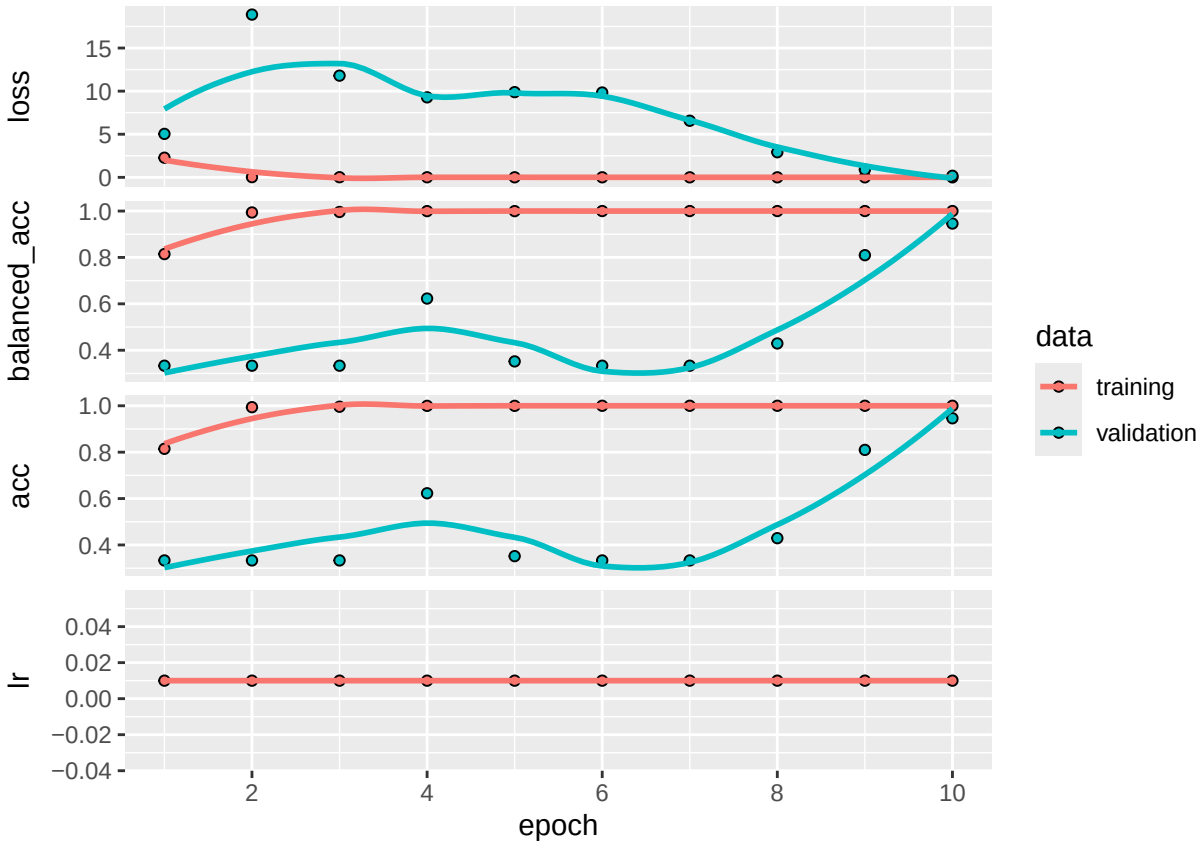
Experiment 39

Spec.region excluded for "abnormal". Functional change not guaranteed.

Call: `GenePermutation(triplets, keyed, num.perm=15000, min.subs=10, max.subs=30, spec.region=30:60)`

Model: `maxlen = 477, batch_size = 64, steps_per_epoch = 45, epochs = 10, step = c(1, 1, 1)`

```
# load outputs/hist39.rds
hist39 <- readRDS(file="outputs/hist39.rds")
plot(hist39)
```




```
checkpoint_path <- file.path("checkpoints")
dir_path <- file.path("outputs")
```

```
run_name <- "rsd-permutation_39"
model <- load_cp(paste0(checkpoint_path, "/", run_name),
                 cp_filter = "last_ep")
```

```
## Using checkpoint checkpoints/rsd-permutation_39/Ep.010-val_loss0.18-val_acc0.946.hdf5
```

Model at last Epoch

```
path_special_test <- file.path("special/test")
path_normal_test <- file.path("normal/test")
path_abnormal_test <- file.path("abnormal/test")
eval_model <- evaluate_model(path_input = c(path_normal_test,
  path_abnormal_test, path_special_test),
  model = model,
  batch_size = 64,
  step = 5,
  vocabulary_label = list(c("normal", "abnormal", "special")),
  number_batches = 10,
  mode = "label_folder",
  verbose = FALSE
)

eval_model
```

```
## [[1]]
## [[1]]$confusion_matrix
##            Truth
## Prediction normal abnormal special
##   normal      254       82       1
##   abnormal      0      163       0
##   special       2       11     255
##
## [[1]]$accuracy
## [1] 0.875
##
## [[1]]$categorical_crossentropy_loss
## [1] 0.6876215
##
## [[1]]$AUC
## NULL
##
## [[1]]$AUPRC
## NULL
```

```
instance <- permute_sequence(triplets, type="ok", min.subs=5, max.subs=30,
                                dict=codon.dict, spec.cond=FALSE, spec.region=NULL)
instance <- permute_sequence(instance, type="func", min.subs=5, max.subs=30,
                                dict=codon.dict, spec.cond=TRUE, spec.region=30:60)
instance_pasted <- paste(instance, collapse = "")
onehot_instance <-  seq_encoding_label(char_sequence = instance_pasted,
```

```
                                          maxlen = 477,
                                          start_ind = 1,
                                          vocabulary = c("A", "C", "G", "T"))
head(onehot_instance[1,,])
```

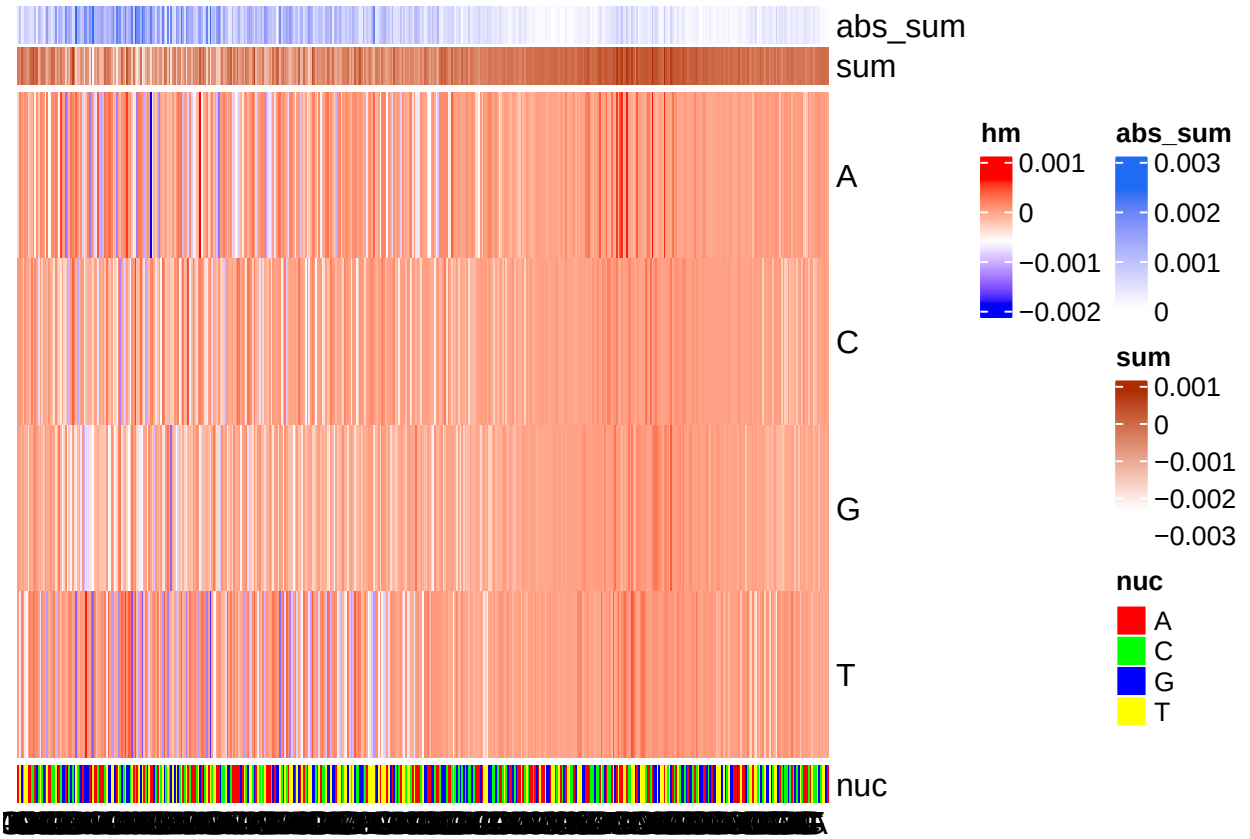
```
##      [,1] [,2] [,3] [,4]
## [1,]    1    0    0    0
## [2,]    0    0    0    1
## [3,]    0    0    1    0
## [4,]    0    1    0    0
## [5,]    0    0    0    1
## [6,]    0    0    0    1
```

```
pred <- predict(model, onehot_instance, verbose = 0)
pred
```

```
##              [,1]         [,2] [,3]
## [1,] 2.989441e-09 3.559504e-18    1
```

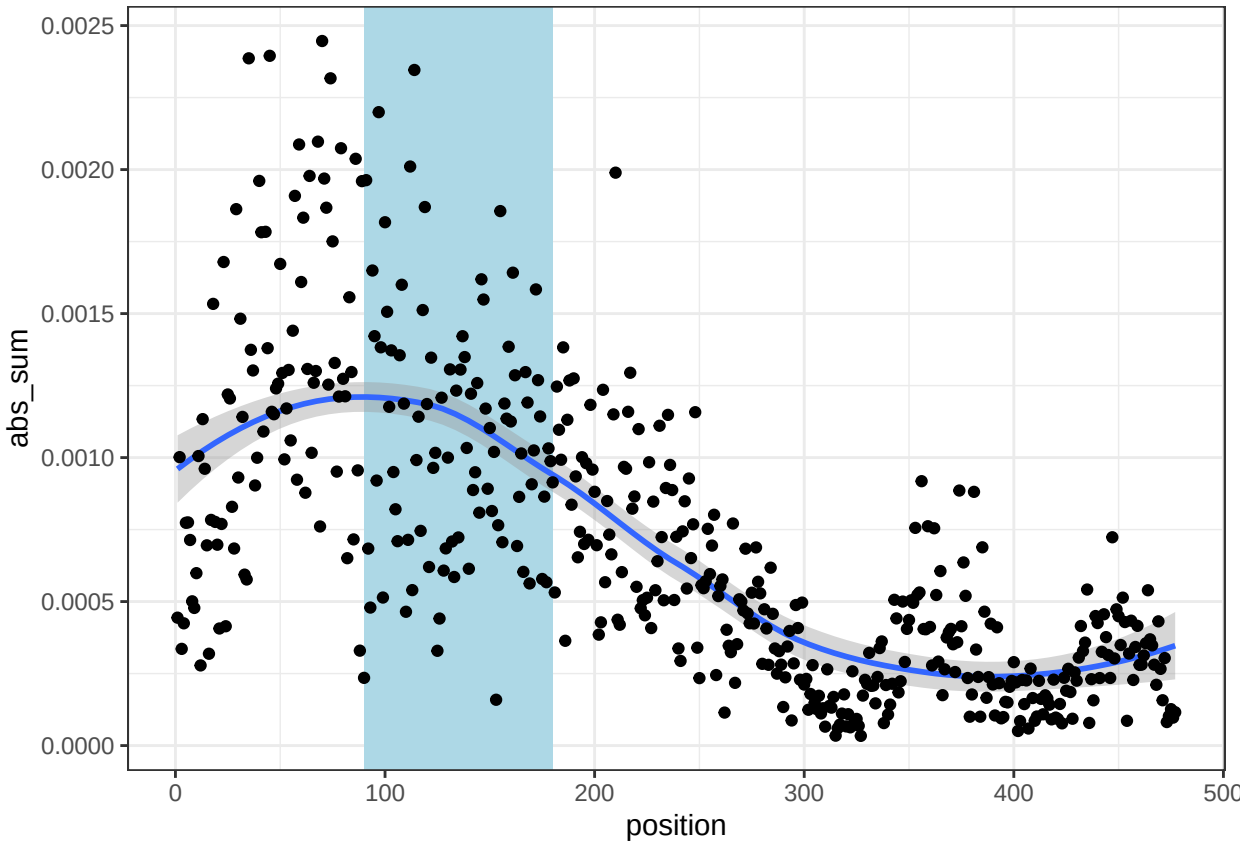
```
ig <- integrated_gradients(
  input_seq = onehot_instance,
  baseline_type = "shuffle",
  target_class_idx = 2,
  model = model,
  num_baseline_repeats = 50)

heatmaps_integrated_grad(integrated_grads = ig,
                         input_seq = onehot_instance)
```

```
## [[1]]
```




```
abs_sum <- rowSums(abs(as.array(ig)))
df <- data.frame(abs_sum = abs_sum, position = 1 : 477)
ggplot(df, aes(x = position, y = abs_sum)) + geom_rect(aes(xmin = 90, xmax = 180, ymin = -Inf, ymax = I
```

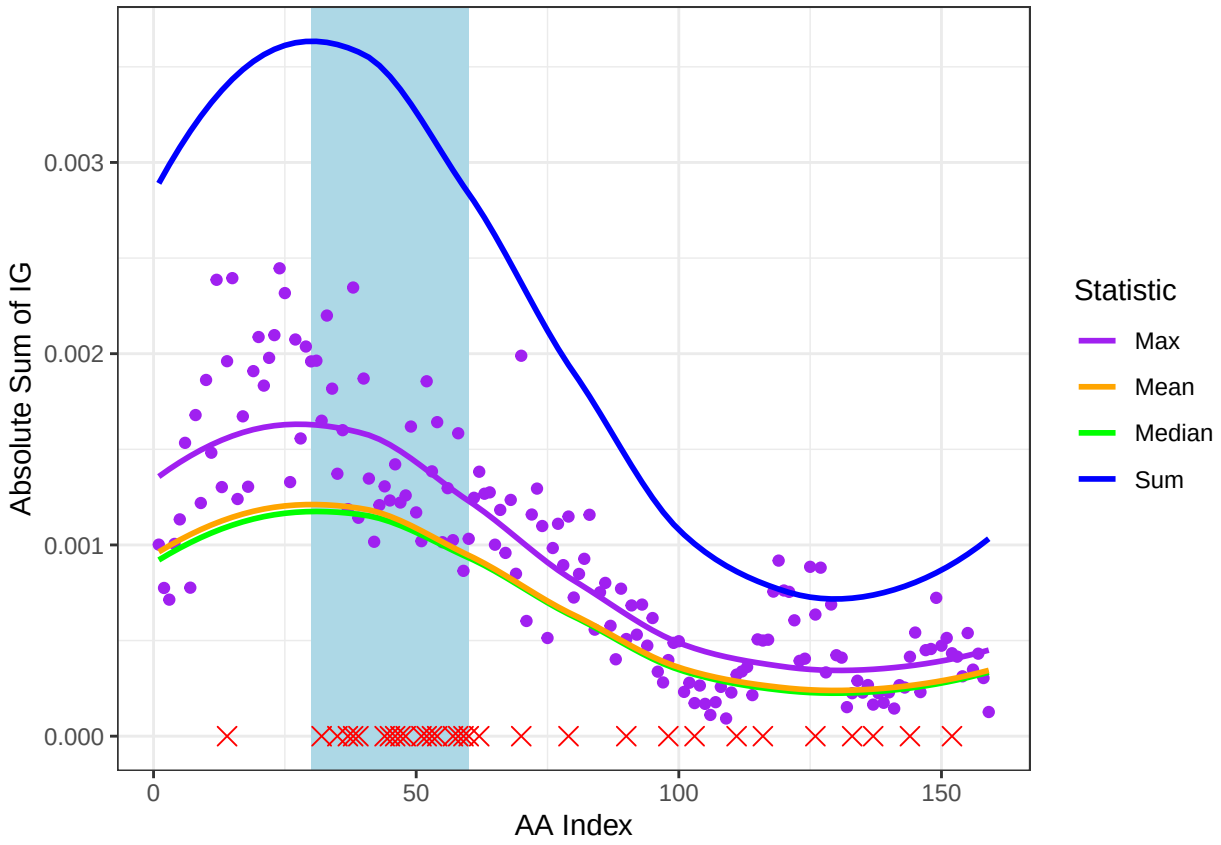
```
## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```




```r
# abs_sum <- rowSums(abs(as.array(ig)))
#df <- data.frame(abs_sum = abs_sum, position = 1 : 477)
#ggplot(df, aes(x = position, y = abs_sum)) + geom_rect(aes(xmin = 90, xmax = 180, ymin = -Inf, ymax =

# modification: df should be summed every 3 rows, position reassigned accordingly.
df_mod <- df %>%
  mutate(group = rep(1:(nrow(df) / 3), each = 3)) %>%
  group_by(group) %>%
  summarise(
    abs_sum_sum = sum(abs_sum),
    abs_sum_median = median(abs_sum),
    abs_sum_mean = mean(abs_sum),
    abs_sum_max = max(abs_sum)
  )

# Split instance (a string) into triplets (every third character)
chars <- strsplit(instance_pasted, "")[[1]]

# Create triplets
trip_instance <- sapply(seq(1, length(chars), by = 3), function(i) {
  paste(chars[i:min(i+2, length(chars))], collapse = "")
})

# Get the index of sequences at which trip_instance != triplets
index <- which(trip_instance != triplets)
```

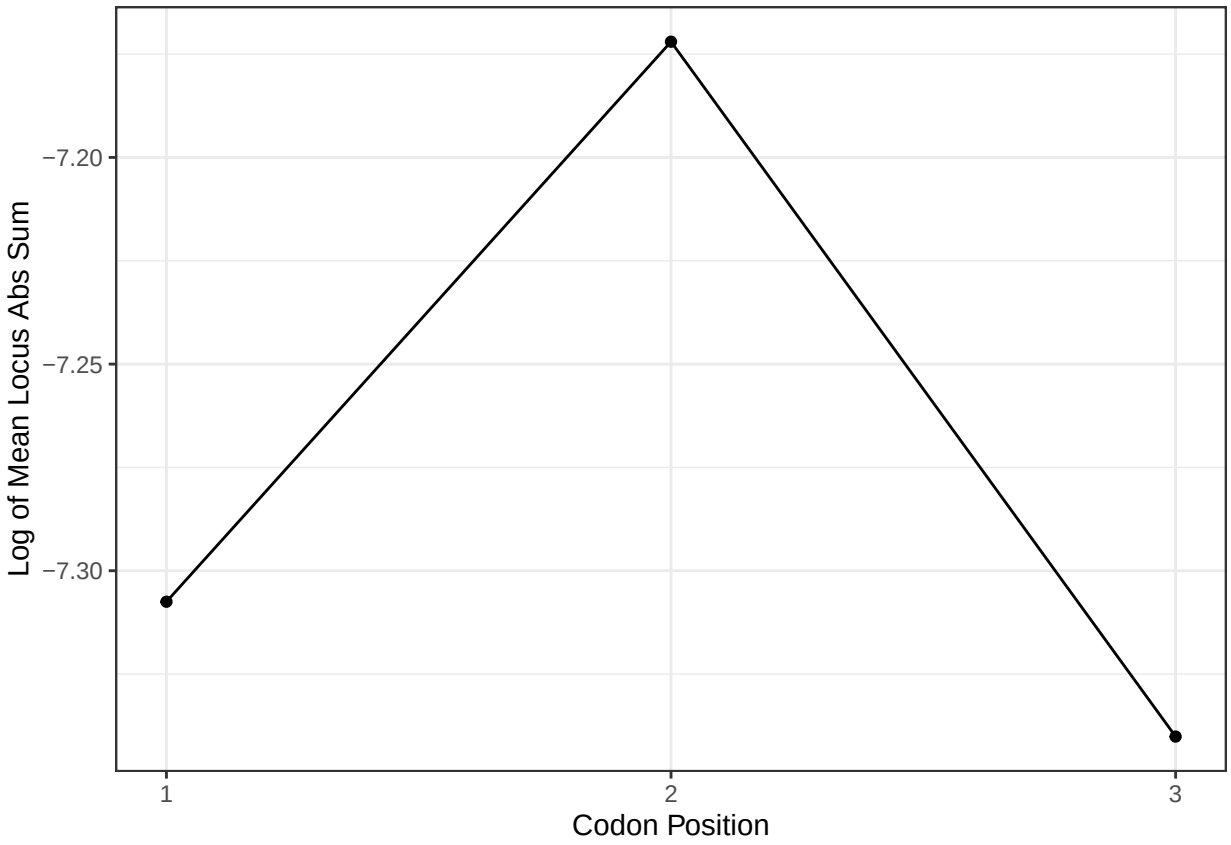
```r
# Plot the results with different smooth lines for sum, median, mean, and max
ggplot(df_mod, aes(x = group)) +
  geom_rect(aes(xmin = 30, xmax = 60, ymin = -Inf, ymax = Inf), fill = "lightblue", alpha = 0.2) +
  # draw points of max
  geom_point(aes(x = group, y = abs_sum_max), color = "purple") +
  geom_smooth(aes(y = abs_sum_sum, color = "Sum"), method = "auto", se = FALSE) +
  geom_smooth(aes(y = abs_sum_median, color = "Median"), method = "auto", se = FALSE) +
  geom_smooth(aes(y = abs_sum_mean, color = "Mean"), method = "auto", se = FALSE) +
  geom_smooth(aes(y = abs_sum_max, color = "Max"), method = "auto", se = FALSE) +
  scale_color_manual(values = c("Sum" = "blue", "Median" = "green", "Mean" = "orange", "Max" = "purple"
  geom_point(data = data.frame(group = index), aes(x = group, y = 0), shape = 4, size = 3, color = "red
  theme_bw() +
  labs(y = "Absolute Sum of IG", color = "Statistic", x = "AA Index")
```

```
## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```

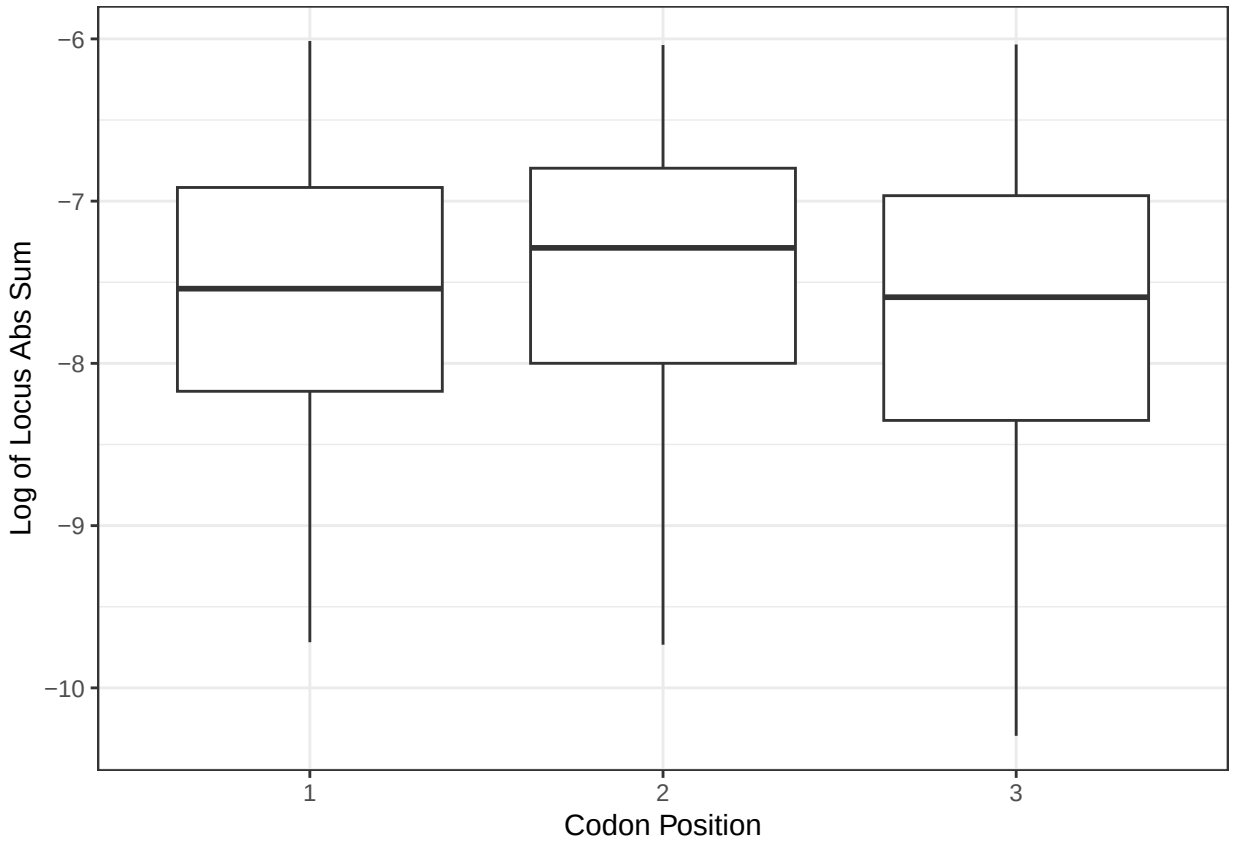



```r
# calculate mean/med of first, second, and third codon positions
mean_codon <- sapply(1:3, function(i) {
  mean(df$abs_sum[seq(i, 477, by = 3)])
})
# plot with line. x:1,2,3, y: mean_codon
ggplot(data = data.frame(x = 1:3, y = log(mean_codon)), aes(x = x, y = y)) +
```

```r
  geom_line() +
  geom_point() +
  theme_bw() +
  labs(x = "Codon Position", y = "Log of Mean Locus Abs Sum") +
  # x tick only 1,2,3, integer
  scale_x_continuous(breaks = 1:3, minor_breaks = NULL, labels = c("1", "2", "3"))
```




```r
# Separate the abs_sum data by codon positions
codon_data <- data.frame(
  position = rep(1:3, each = length(df$abs_sum) / 3),
  abs_sum = unlist(lapply(1:3, function(i) df$abs_sum[seq(i, 477, by = 3)]))
)

# Box plot for each codon position
ggplot(codon_data, aes(x = factor(position), y = log(abs_sum))) +
  geom_boxplot() +
  theme_bw() +
  labs(x = "Codon Position", y = "Log of Locus Abs Sum") +
  scale_x_discrete(labels = c("1", "2", "3"))
```

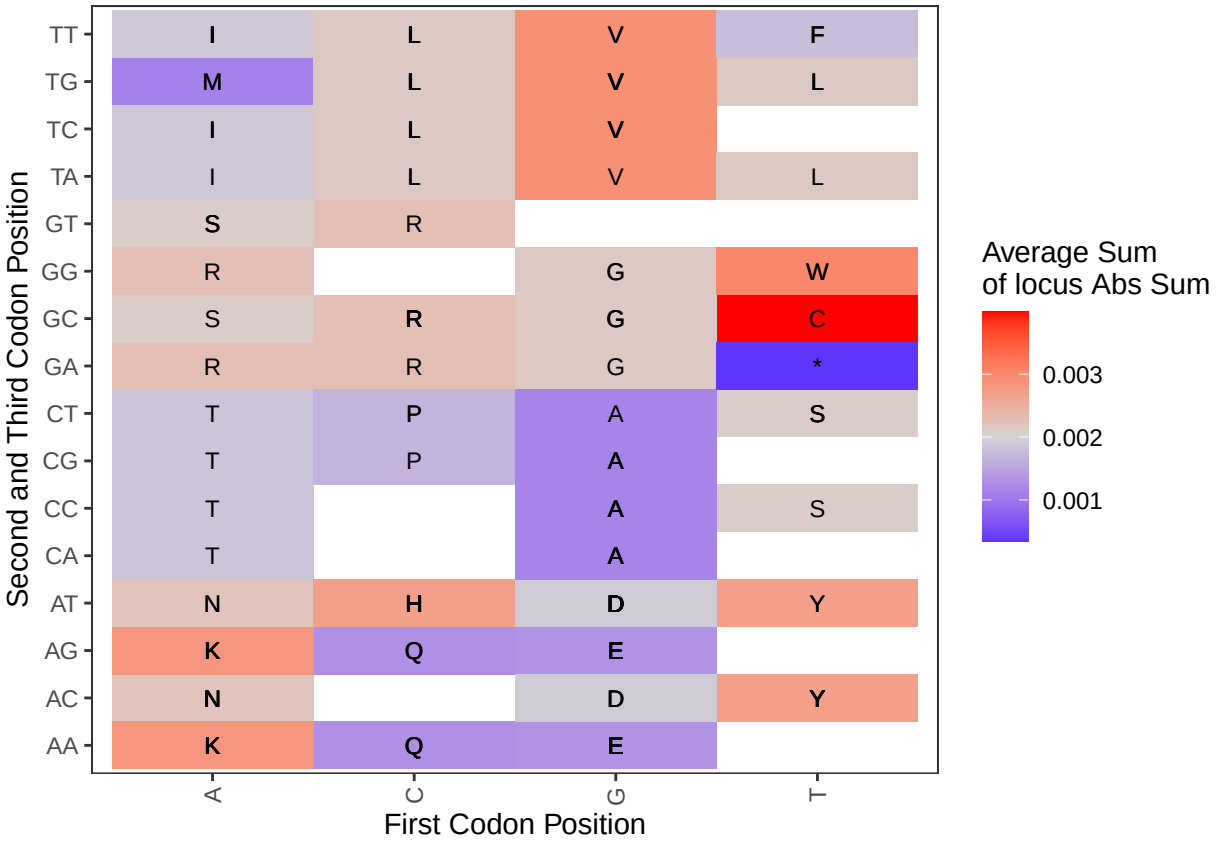



```
lm_model <- lm(log(abs_sum) ~ factor(position), data = codon_data)

# Summarize the linear model
summary(lm_model)
```

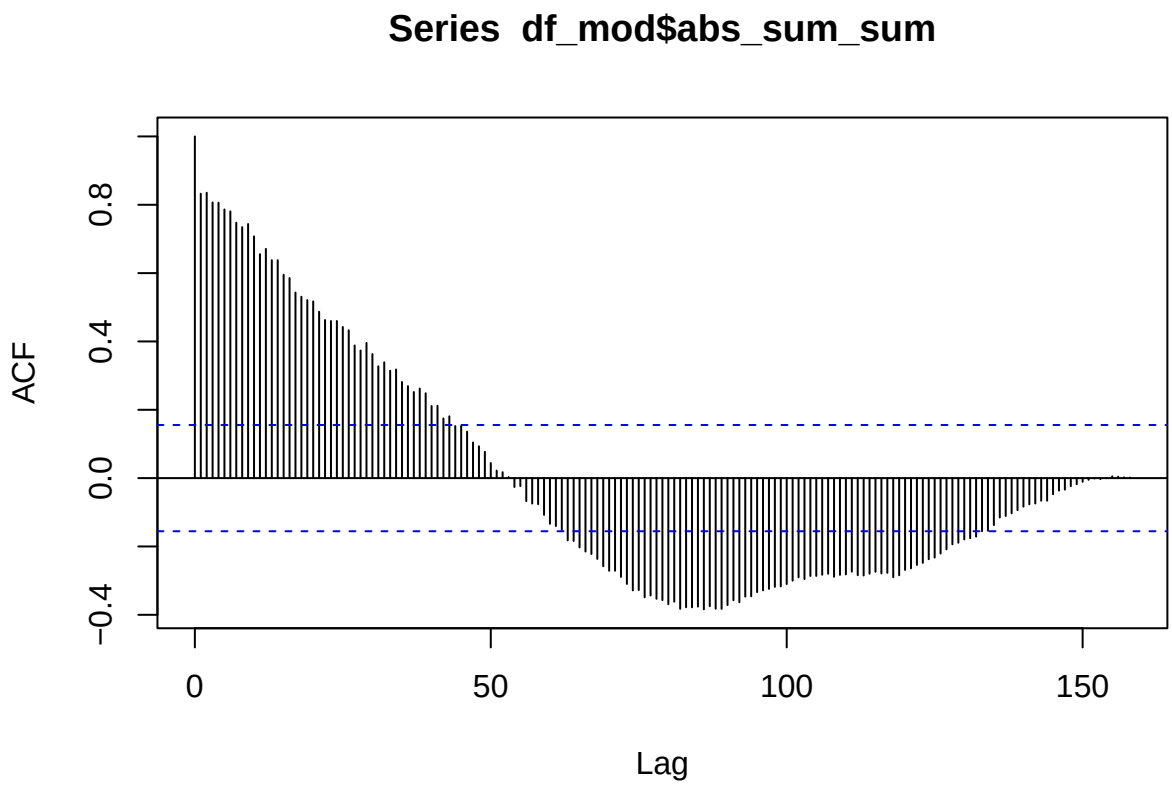
```
##
## Call:
## lm(formula = log(abs_sum) ~ factor(position), data = codon_data)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.62161 -0.55317  0.09533  0.70736  1.63964
##
## Coefficients:
##                   Estimate Std. Error  t value Pr(>|t|)
## (Intercept)       -7.63479    0.06965 -109.624   <2e-16 ***
## factor(position)2  0.15296    0.09849    1.553    0.121
## factor(position)3 -0.03918    0.09849   -0.398    0.691
## ---
## Signif. codes:  0 ’***’ 0.001 ’**’ 0.01 ’*’ 0.05 ’.’ 0.1 ’ ’ 1
##
## Residual standard error: 0.8782 on 474 degrees of freedom
## Multiple R-squared:  0.008887,   Adjusted R-squared:  0.004705
## F-statistic: 2.125 on 2 and 474 DF,  p-value: 0.1205
```

NB: plot generated with considerable randomness and variance.

```r
keyed_instance <- triplets_keying(trip_instance)
df_keyed <- cbind(keyed_instance, trip_instance, df_mod)
# mutate two colmuns: cod1 is the first character in trip_instance, cod23 is the 2-3 chars in trip_inst
df_keyed <- df_keyed %>%
  mutate(cod1 = substr(trip_instance, 1, 1),
         cod23 = substr(trip_instance, 2, 3))
df_keyed <- df_keyed %>%
  group_by(keyed_instance) %>%
  mutate(value = mean(abs_sum_sum))

ggplot(df_keyed, aes(x = cod1, y = cod23, fill = value)) +
  geom_tile() +
  geom_text(aes(label = keyed_instance), color = "black", size = 3) +
  scale_fill_gradient2(low = "blue", mid = "lightgrey", high = "red", midpoint = 0.002) +
  theme_bw() +
  labs(fill = "Average Sum\nof locus Abs Sum",
       x = "First Codon Position",
       y = "Second and Third Codon Position") +
  theme(axis.text.x = element_text(angle = 90, vjust = 0.5, hjust=1),
        strip.text = element_text(face = "bold"),
        panel.grid.major = element_blank(), panel.grid.minor = element_blank())
```



```r
cor_values <- acf(df_mod$abs_sum_sum, lag.max = 159, plot = TRUE)
```



Codon Synonym Similarity

CSS score by Durrant & Bhatt:

$$CSS := \frac{1}{k} \sum_{i=1}^{k} \hat{\sigma}_i$$

Null distribution approximated by random AA substitution.

CSS score using coefficient of variance:

$$CSS_{CV} := \frac{1}{k} \sum_{i=1}^{k} \frac{\hat{\sigma}_i}{\hat{\mu}_i}$$

Positional Codon Synonym Similarity score: perform ok-mutate based on instance. Group aggregated IG scores (sum) by their AA position, and calculate the standard deviation of the mean IG score for each triplet, corrected by its mean and scaled using inverse transformation. => No distribution needed, if near 1, then low variability.

$$\text{PCSS} := \frac{1}{N} \sum_{i=1}^{N} \frac{1}{1 + \alpha \cdot \hat{c}_v}, \quad \hat{c}_v = \frac{\hat{\sigma}_i}{\hat{\mu}_i}, \; N = \frac{n}{3}$$

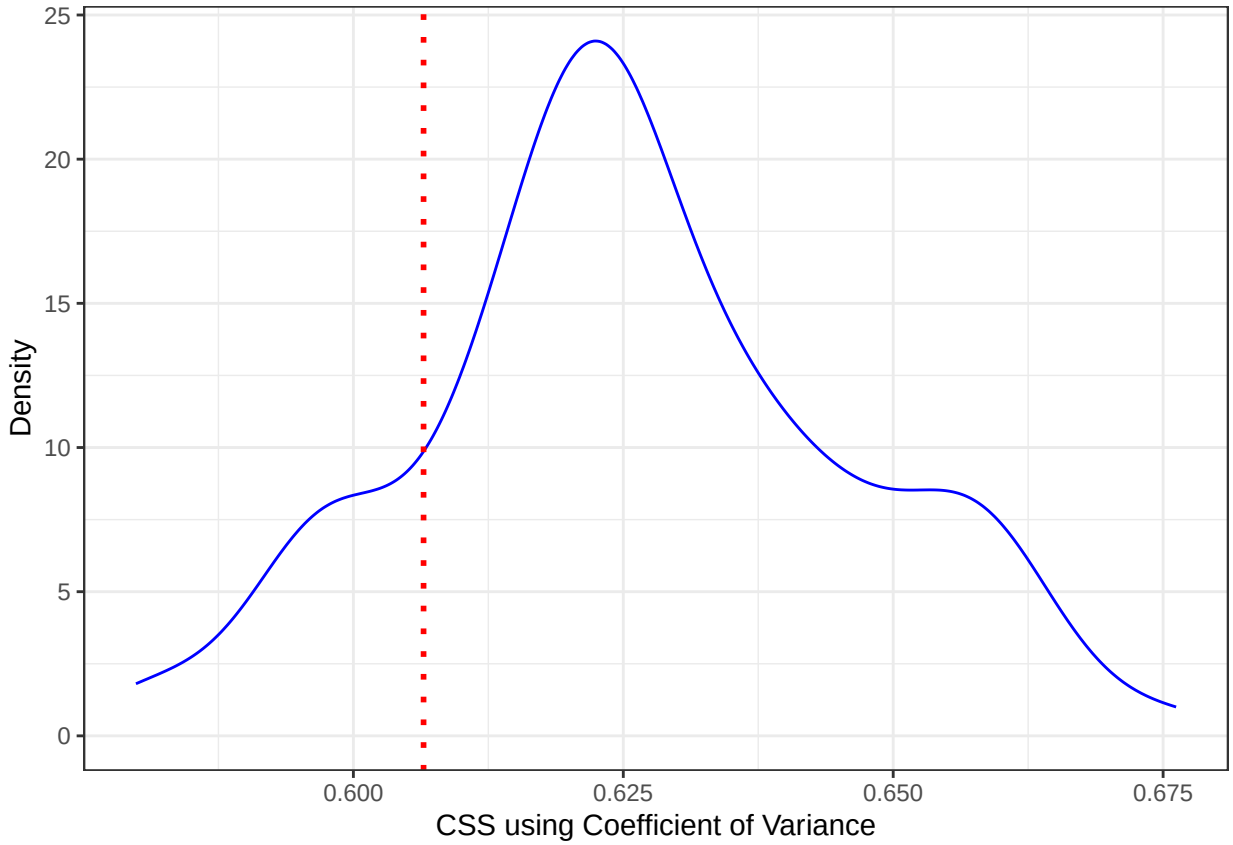
```r
ig_instance <- integrated_gradients(
  input_seq = onehot_instance,
  baseline_type = "shuffle",
  target_class_idx = 3,
  model = model,
  num_baseline_repeats = 50)
```

```r
abs_sum <- rowSums(abs(as.array(ig_instance)))
df <- data.frame(abs_sum = abs_sum, position = 1 : 477)
df_mod <- df %>%
  mutate(group = rep(1:(nrow(df) / 3), each = 3)) %>%
  group_by(group) %>%
  summarise(
    abs_sum_mean = mean(abs_sum)
  )
df_k <- cbind(keyed_instance, trip_instance, df_mod)
# delete row if keyed_instance in "M","W"
df_k <- df_k %>%
  filter(!(keyed_instance %in% c("M", "W", "*")))
# groupby keyed_instance, calculate std of abs_sum_mean using summary
df_std <- df_k %>%
  group_by(keyed_instance) %>%
  summarise(std = sd(abs_sum_mean) / mean(abs_sum_mean))
css_instance_init <- mean(df_std$std, na.rm=TRUE)

# use: permute_sequence(instance, type="ok", min.subs=30, max.subs=80,
#                       dict=codon.dict, spec.cond=FALSE, spec.region=NULL)
# to create 100 instances, and repeat everything from "#begin"
# for each permuted instance.
# Save all css_instance, excluding the original one, into css vector
```

```r
css_values <- numeric(100)

# Loop to create 100 permuted instances
for (i in 1:100) {
  # Create a permuted instance with the specified parameters
  permuted_instance <- permute_sequence(instance, type = "ok", min.subs = 10, max.subs = 30,
                                        dict = codon.dict, spec.cond = FALSE, spec.region = NULL)

  # Start the process for the new permuted instance
  permuted_instance <- paste(permuted_instance, collapse = "")
  chars <- strsplit(permuted_instance, "")[[1]]
  trip_instance <- sapply(seq(1, length(chars), by = 3), function(i) {
    paste(chars[i:min(i+2, length(chars))], collapse = "")
  })
  keyed_instance <- triplets_keying(trip_instance)
  onehot_instance <- seq_encoding_label(char_sequence = permuted_instance,
                                        maxlen = 477,
                                        start_ind = 1,
                                        vocabulary = c("A", "C", "G", "T"))

  ig_instance <- integrated_gradients(
    input_seq = onehot_instance,
    baseline_type = "shuffle",
    target_class_idx = 3,
    model = model,
    num_baseline_repeats = 50
  )

  abs_sum <- rowSums(abs(as.array(ig_instance)))
```

```
  df <- data.frame(abs_sum = abs_sum, position = 1:477)
  df_mod <- df %>%
    mutate(group = rep(1:(nrow(df) / 3), each = 3)) %>%
    group_by(group) %>%
    summarise(
      abs_sum_mean = mean(abs_sum)
    )

  df_k <- cbind(keyed_instance, trip_instance, df_mod)

  # Delete rows where keyed_instance is "M", "W", or "*"
  df_k <- df_k %>%
    filter(!(keyed_instance %in% c("M", "W", "*")))

  # Group by keyed_instance and calculate the standard deviation of abs_sum_mean
  df_std <- df_k %>%
    group_by(keyed_instance) %>%
    summarise(std = sd(abs_sum_mean) / mean(abs_sum_mean))

  # Calculate the css_instance for this permuted instance
  css_instance <- mean(df_std$std, na.rm=TRUE)

  # Store the css_instance in the vector
  css_values[i] <- css_instance
}
```

```
# density curve of css_values
dfcss <- data.frame(CSS = css_values)
ggplot(dfcss, aes(x = CSS)) +
  geom_density(color = "blue", alpha = 0.5) +
  theme_bw() +
  # a dotted vertical line of x=css_instance_init
  geom_vline(xintercept = css_instance_init, linetype = "dotted", color = "red", lwd=1) +
  labs(x = "CSS using Coefficient of Variance", y = "Density")
```




```r
# ok-mutate based on instance, 100 times, saved to pcss_df
# just use permute_sequence, nothing else
pcss_df <- data.frame(permuted = instance_pasted)
for (i in 1:99) {
  permuted_instance <- permute_sequence(instance, type = "ok", min.subs = 80,
                                        max.subs = 120, dict = codon.dict,
                                        spec.cond = FALSE, spec.region = NULL)
  permuted_instance <- paste(permuted_instance, collapse = "")
  pcss_df <- rbind(pcss_df, data.frame(permuted = I(list(permuted_instance))))
}
list_onehot <- lapply(pcss_df$permuted, function(x) {
  seq_encoding_label(char_sequence = x, maxlen = 477, start_ind = 1, vocabulary = c("A", "C", "G", "T"))
})
```

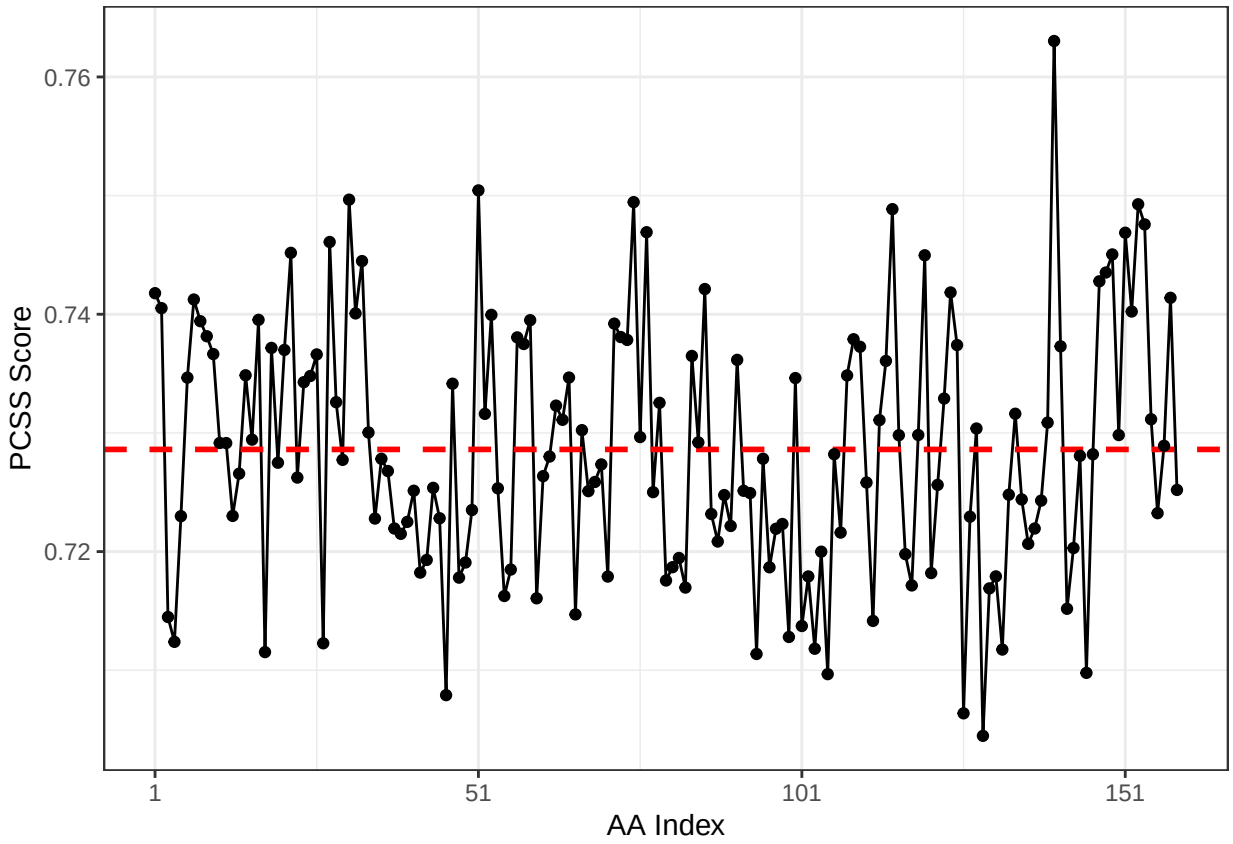
```r
csv_file_path <- "pcssdata.csv"

# Check if the CSV file already exists
if (file.exists(csv_file_path)) {
  # If it exists, read the CSV into result_df
  result_df <- read.csv(csv_file_path)
  message("Loaded result_df from existing CSV file.")
} else {
# Initialize an empty dataframe with the position column
  result_df <- data.frame(position = 1:477)

# Loop through each one-hot encoded instance in the list
```

```
  for (i in seq_along(list_onehot)) {
    onehot_instance <- list_onehot[[i]]

  # Compute Integrated Gradients
    ig <- integrated_gradients(
      input_seq = onehot_instance,
      baseline_type = "shuffle",
      target_class_idx = 2,
      model = model,
      num_baseline_repeats = 50
    )

  # Compute the absolute sum of the IG scores
    abs_sum <- rowSums(abs(as.array(ig)))

  # Add the abs_sum as a new column in the result_df
    result_df[[paste0("abssum", i)]] <- abs_sum
  }
}
```

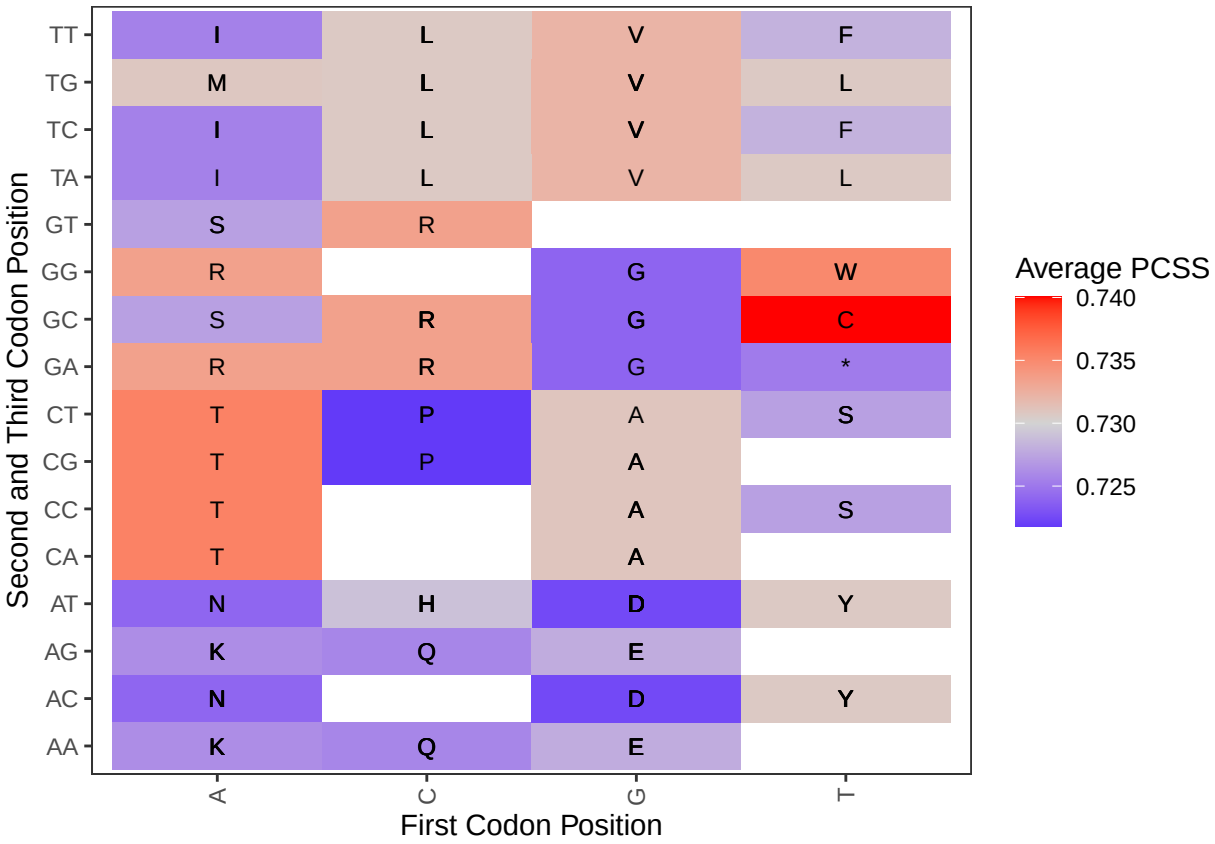
```
## Loaded result_df from existing CSV file.
```

```
calculate_mean_every_three_rows <- function(df) {
  # Calculate the number of groups (each group will consist of three rows)
  n_groups <- nrow(df) %/% 3

  # Initialize an empty list to store the means
  mean_list <- list()

  # Loop through each group and calculate the mean for each column
  for (i in 1:n_groups) {
    # Select the three rows corresponding to the current group
    rows <- df[((i-1) * 3 + 1):(i * 3), ]

    # Calculate the mean for each column in the current group
    mean_row <- colSums(rows)

    # Append the result to the list
    mean_list[[i]] <- mean_row
  }

  # Combine the results into a new dataframe
  mean_df <- do.call(rbind, mean_list)

  return(mean_df)
}

# Apply the function to result_df
result_df_mean <- calculate_mean_every_three_rows(result_df)

# Convert the result to a data frame with appropriate column names
result_df_mean <- as.data.frame(result_df_mean)
names(result_df_mean) <- names(result_df)
```

```r
result_df_mean <- result_df_mean %>%
  select(-X, -position)

# Function to calculate the coefficient of variance (CV) for each row
calculate_cv_rowwise <- function(df) {
  # Apply the function to calculate CV (sd/mean) row-wise
  cv <- apply(df[-1], 1, function(row) {
    row_sd <- sd(row)
    row_mean <- mean(row)
    if (row_mean != 0) {
      return(row_sd / row_mean)
    } else {
      return(NA)  # Handle division by zero
    }
  })
  return(cv)
}
invtrans <- function(x) {
  return(1 / (1 + x))
}

# Apply the function to result_df_mean (excluding the first column "position")
rowwise_std <- calculate_cv_rowwise(result_df_mean)
pcss_final <- data.frame(position=1:length(rowwise_std), key = keyed_instance,
                         trip_instance, sig.cv=invtrans(rowwise_std))

# plot pcss_final, x:position, y:cv, draw a horizontal line for the mean(pcss_final$cv), and mark its y
ggplot(pcss_final, aes(x = position, y = sig.cv)) +
  geom_line() +
  geom_hline(yintercept = mean(pcss_final$sig.cv), linetype = "dashed", lwd=1, color = "red") +
  geom_point() +
  theme_bw() +
  labs(x = "AA Index", y = "PCSS Score") +
  scale_x_continuous(breaks = seq(1, 477, by = 50))
```



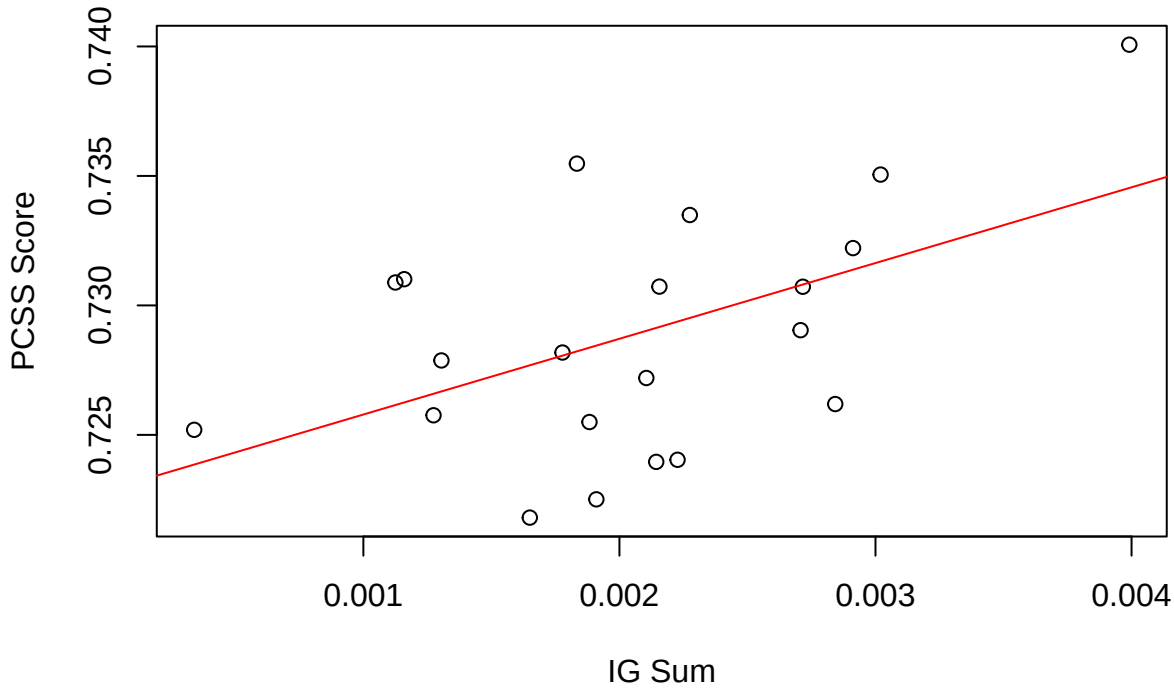

```
pcss_final <- pcss_final %>%
  mutate(cod1 = substr(trip_instance, 1, 1),
         cod23 = substr(trip_instance, 2, 3))
pcss_final <- pcss_final %>%
  group_by(key) %>%
  mutate(value = mean(sig.cv))

ggplot(pcss_final, aes(x = cod1, y = cod23, fill = value)) +
  geom_tile() +
  geom_text(aes(label = key), color = "black", size = 3) +
  scale_fill_gradient2(low = "blue", mid = "lightgrey", high = "red", midpoint = 0.73) +
  theme_bw() +
  labs(fill = "Average PCSS",
       x = "First Codon Position",
       y = "Second and Third Codon Position") +
  theme(axis.text.x = element_text(angle = 90, vjust = 0.5, hjust=1),
        strip.text = element_text(face = "bold"),
        panel.grid.major = element_blank(), panel.grid.minor = element_blank())
```




```
# correlation of df_keyed$abs_sum_sum vs. pcss_final$value
data_cor <- left_join(df_keyed, pcss_final, by = c("group" = "position"))
# keep value.y, vlaue.x unique: groupby keyed_instance and select the first row of each group
data_cor <- data_cor %>%
  group_by(keyed_instance) %>%
  slice(1)
cor.test(data_cor$value.y, data_cor$value.x, method = "pearson")
```

```
##
##  Pearson's product-moment correlation
##
## data:  data_cor$value.y and data_cor$value.x
## t = 2.554, df = 19, p-value = 0.01939
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  0.09447585 0.76933420
## sample estimates:
##       cor
## 0.5055445
```

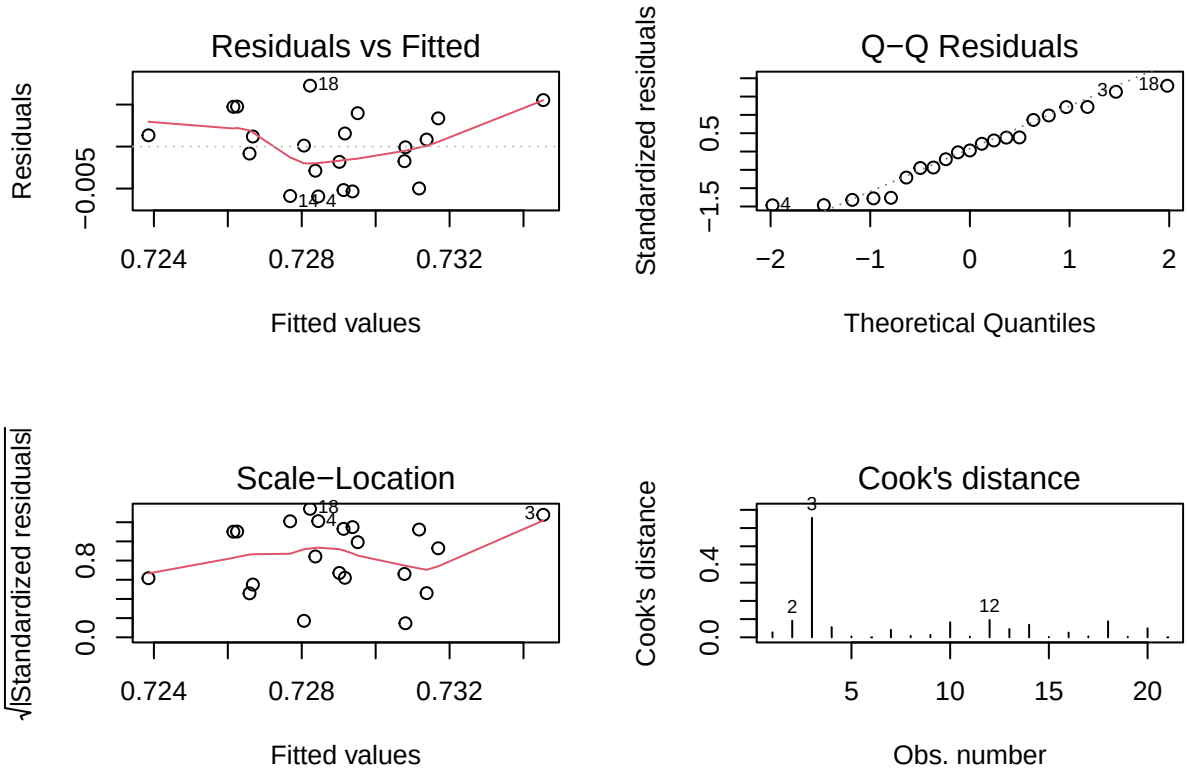
```
plot(data_cor$value.x, data_cor$value.y,
     xlab = "IG Sum",
     ylab = "PCSS Score")
lmpcss <- lm(data_cor$value.y ~ data_cor$value.x)
abline(lmpcss, col = "red")
```




```
summary(lmpcss)
```

```
##
## Call:
## lm(formula = data_cor$value.y ~ data_cor$value.x)
##
## Residuals:
##        Min         1Q     Median         3Q        Max
## -0.0059364 -0.0028727  0.0001192  0.0033585  0.0072505
##
## Coefficients:
##                  Estimate Std. Error t value Pr(>|t|)
## (Intercept)      0.722861   0.002532 285.526   <2e-16 ***
## data_cor$value.x 2.925000   1.145249   2.554   0.0194 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.004148 on 19 degrees of freedom
## Multiple R-squared:  0.2556, Adjusted R-squared:  0.2164
## F-statistic: 6.523 on 1 and 19 DF,  p-value: 0.01939
```

```
# 4x4 grid
par(mfrow=c(2,2))
plot(lmpcss, which=1:4)
```


Residuals vs Fitted
Residuals
−0.005
18
14
4
0.724
0.728
0.732
Fitted values
Q−Q Residuals
Standardized residuals
−1.5
0.5
4
3
18
−2
−1
0
1
2
Theoretical Quantiles
Scale−Location
√|Standardized residuals|
0.0
0.8
18
4
3
0.724
0.728
0.732
Fitted values
Cook's distance
Cook's distance
0.0
0.4
3
2
12
5
10
15
20
Obs. number