UBMI-IFC, UNAM
Coyoacan, CDMX

# Sequece Characterization Test

## Using Genomic-Benchmarks Data

Author: Fuentes David
Tutors:
- PhD. Poot Augusto
- MSc. Pedraza Carlos

# Table of contents

# 1 Downloading data

Python Code

```
# Note: Check 'outputwrap' functions in source 'qmd'
#   notebook, for further insight on output display

# To list available datasets
from genomic_benchmarks.data_check import \
    list_datasets

# To inspect each dataset to select two
from genomic_benchmarks.data_check import info as \
    info_gb

# To download each dataset
from genomic_benchmarks.loc2seq import \
    download_dataset

# To position ourselves in the correct directory
import os
```

```
list_datasets()
```

```
['demo_human_or_worm',
'human_nontata_promoters',
'human_enhancers_ensembl',
'human_enhancers_cohn',
'dummy_mouse_enhancers_ensembl',
'demo_coding_vs_intergenomic_seqs',
'drosophila_enhancers_stark',
'human_ensembl_regulatory',
'human_ocr_ensembl']
```

```
info_gb("human_nontata_promoters", version=0)
```

```
Dataset `human_nontata_promoters` has 2 classes:
negative, positive.

 All lengths of genomic
intervals equals 251.

 Totally 36131 sequences
have been found, 27097 for training and 9034 for
testing.
          train  test
negative  12355  4119
positive  14742  4915
```

```
info_gb("human_ensembl_regulatory", version=0)
```

```
Dataset `human_ensembl_regulatory` has 3 classes:
enhancer, ocr, promoter.

 The length of genomic
intervals ranges from 71 to 802, with average
429.91753643694585 and median 401.0.

 Totally
289061 sequences have been found, 231348 for
training and 57713 for testing.
          train   test
enhancer  85512  21378
ocr       69902  17476
promoter  75934  18859
```

```
info_gb("human_enhancers_cohn", version=0)
```

```
Dataset `human_enhancers_cohn` has 2 classes:
negative, positive.

 All lengths of genomic
intervals equals 500.

 Totally 27791 sequences
have been found, 20843 for training and 6948 for
testing.
          train  test
negative  10422  3474
positive  10421  3474
```

```
info_gb("human_enhancers_ensembl", version=0)
```

```
Dataset `human_enhancers_ensembl` has 2 classes:
negative, positive.

 The length of genomic
intervals ranges from 2 to 573, with average
268.8641324705183 and median 269.0.

 Totally
154842 sequences have been found, 123872 for
training and 30970 for testing.
          train   test
negative  61936  15485
positive  61936  15485
```

```
os.chdir("/path/to/Project/datasets/GenomicBenchmarks")

download_dataset("human_nontata_promoters", version=0)
download_dataset("human_enhancers_cohn", version=0)
```

## 2 Formatting data

The downloaded data came in the form of two directories with many *.txt* files inside, and seeing that at least '*getShape()*' function from '*DNAshapeR*' library required FASTA files to work and it felt right to collect all sequences in a single file, I transfered all sequences of each cis-regulatory element to their single respective FASTA file and gave them a simple header.

Bash Code

```
cd /path/to/Project/datasets/GenomicBenchmarks/

awk 'BEGIN{counter=0}
     {print ">promoter_"counter"|train|positive";
     print $0; counter+=1}' \
     human_nontata_promoters/train/positive/*.txt \
     > promoters_train_positive.fasta

awk 'BEGIN{counter=0}
     {print ">enhancer_"counter"|train|positive";
     print $0; counter+=1}' \
     human_enhancers_cohn/train/positive/*.txt \
     > enhancers_train_positive.fasta
```

## 3 Libraries used

Here is a list of the R libraries used for the following analysis, as well as a reference to my own functions to be used in the characterization step.

R Code

```
# For genome-functions.R
library(stringr)
library(stringi)
library(primes)
# For parallel computing
library(doParallel)
library(foreach)
# For biological functions:
#   - Local/Global alignments
#   - DNA Shape computing
library(Biostrings)
library(DNAshapeR)
# For plotting
library(paletteer)
library(cowplot)
library(ggplot2)
library(dplyr)
library(plyr)
library(see)
# For pretty tables
library(knitr)
library(kableExtra)
# For my own functions
source("/path/to/Project/scripts/genome-functions.R")
```

## 4 Characterizing sequences

Here, we characterize a subset from each of our datasets: 1638 sequences per regulatory element; 3276 in total. However, there's a circumstance about the sequences that has to be noted:

- Both datasets have considerably different elements lengthwise:
  1. All promoters have a length of 251 nucleotides.
  2. All enhancers have a length of 500 nucleotides.
- All features per sequence must be numerical and two-dimensional since we want it to be fed eventually to a simple classifier (like a Support Vector Machine).

```
proj_path <- "path/to/Project/datasets/GenomicBenchmarks"
prom_fastaname <- "promoters_train_positive.fasta"
enha_fastaname <- "enhancers_train_positive.fasta"

prom_path <- paste(proj_path, prom_fastaname, sep = "/")
enha_path <- paste(proj_path, enha_fastaname, sep = "/")

# Scanning sequences
prom_seqs <- scan(prom_path,
                  character(), quote="")[seq(2,29484,2)]
enha_seqs <- scan(enha_path,
                  character(), quote="")[seq(2,20842,2)]
```

Given some previous tests done to 'sequences_characterizer()' I came to the conclusion that parallel computing might provide a higher and more complex set of data in a feasible time span.

```
# Prepairing clusters for parallel computing
corescluster <- makeCluster(6)
registerDoParallel(corescluster)

# Characterizing sequences and exporting to CSV
list_seqs <- list(promoters = prom_seqs,
                  enhancers = enha_seqs)
reg_elems <- c("promoters", "enhancers")
libs <- c("stringr", "stringi", "primes")

for (reg_elem in reg_elems) {
  foreach(i = 1:6) %dopar% {
    required_libs(libs)
    i_start <- ((i - 1) * 273) + 1
    i_final <- i * 273

    if (i > 1) { # Only the first CSV has headers
      write.table(
        sequences_characterizer(
          list_seqs[[reg_elem]][i_start:i_final],
          optim = TRUE, k_max = 6),
        paste(
          "datasets/GB-Testing/test", reg_elem,
          "-training_", i, ".csv", sep = ""),
          row.names = FALSE, col.names = FALSE,
          sep = ",")
```

```
    } else {
      write.csv(
        sequences_characterizer(
          list_seqs[[reg_elem]][i_start:i_final],
          optim = TRUE, k_max = 6),
        paste(
          "datasets/GB-Testing/", reg_elem,
          "-training_", i, ".csv", sep = ""),
          row.names = FALSE)
}}}
```

# 5 Concatenating CSV's

It was decided to produce many files instead of appending over the same CSV table in order to apply a sort of quality control checkup after the parallel computing was done. This, because when forcing the process RAM overload was possible and the function could die mid-process. Since our data was separated in six tables per cis-regulatory element, here we only join each set together in a single CSV.

Bash Code

```
cat datasets/GB-Testing/testpromoters-training_*.csv \
    > datasets/GB-Testing/test-1638-promoters-6mers.csv
cat datasets/GB-Testing/testenhancers-training_*.csv \
    > datasets/GB-Testing/test-1638-enhancers-6mers.csv
```

# 6 Data description

Prior to our primary analysis, it feels reasonable to explain the columns per sequence produced by our tabulator '*sequences_characterizer()*'. From here we'll first describe the ones computed sequence-wise, the ones computed kmer-wise, then the ones computed over each kmer distribution, and finally the ones corresponding to DNA-Shape; this feature comes at last considering '*getShape()*' function makes its' own dataframe. The ones in **black** are already integrated in the table, the ones in **red** are yet to be adjoined:

Per sequence

- From *'genome-functions.R'*:
  - **A, T, C, G** - *Nucleotide Percentages* per sequence.
  - **temp** - *Melting Temperature*: Temperature at which DNA's double helix dissociates into single strands. *It's dependent on GC percentage and sequence length.*
  - **shan** - *Shannon Entropy Coefficient*: Statistical quantifier of information in a system. Measures the uncertainty a set of data has. In this case is a nucleotide-diversity metric. *It's dependent on nucleotide percentages.*
- From *'Biostrings'*:
  - **la_sc** - *Local Alignment Score*
  - **la_id** - *Local Alignment Identity*
  - **ga_sc** - *Global Alignment Score*
  - **ga_id** - *Global Alignment Identity*

Table 1: Overview of each column generated so far by '*sequences-characterizer()*'

| Column | Description | Section Processed |
|---|---|---|
| A | Percentage of Alanines | per Sequence |
| T | Percentage of Thymines | |
| C | Percentage of Cytosines | |
| G | Percentage of Guanines | |
| temp | Melting Temperature | |
| shan | Shannon Coefficient | |
| kN.M_prod | KSG Product | per Kmer: each possible kmer of size *N* is identified on a scale of 1 to $4^N$ denoted by *M*. `i.e. k3.1 (or the first kmer of size 3) would be AAA; k4.2 would be AAAC` |
| kN.M_barc | Barcode Profile | |
| kN.M_pals | Palindrome Profile | |
| kN.M_revc | Reverse Complement Profile | |

Per kmer

Kmer characterization came with many challenges, principally to provide each kmer with a distinct signature value dependent on their sequence structure. This under the assumption that .

- From *'genome-functions.R'*:
  - **kN.M_prod** - *KSG Product*: Its obtained by multiplying **Kmer-Percentage * Shannon Entropy * GC-Percentage** (*in concept*). The justification for this comes from the desire to give each kmer a distictive signal based on their sequence. However many kmers can produce the same value depending on the function, for example, the kmers **'AAAT', 'ATAA', 'CCGC'** and **'TGGG'** all have the *same Entropy Coefficient* (0.811), while **'GGAT', 'CAAG', 'AACC'** and **'TGCT'** all have the *same GC Percentage* (0.5). Additionally, it is a little tricky to *keep the product from becoming zero* when the sequence is either devoid of C/G nucleotides (e.g. **gc_percentage('ATTA') =0**), or lacks nucleotide diversity (e.g. **shannon_entropy('GGGG')=0**).
    Refer to 'Theory & Code Annex', Sections 2 & 3, for more details.
  - **kN.M_barc** - *Barcode Profile*: Its obtained by gen-

erating a vector with size equal to the kmer amount (which is equivalent to '**sequence length**' - '**kmer length**' + 1) and filling it with one prime number per cell in ascending order. Afterwars we use a logical (**1's & 0's**) vector of the same size to represent each kmer's positions in the sequence. *In concept*, if we multiply both vectors we will get a set of prime numbers representing the positions of each kmer inside the sequence. If we multiply as well the elements inside this set, we will obtain an exlusive value for each kmer in each sequence, and only the sequences with the same kmers in the same place will be divisible by the corresponding prime.
  - **kN.M_pals** - *Palindromes' Profile*:
  - **kN.M_revc** - *Reverse Complement Profile*:

Per kmer distribution

Per non-defined kmer

- From *'DNAshapeR'*:
  Generates a set features as vectors (EP, MGW, HelT, Roll & ProT by default) per provided FASTA file. Splits each sequence in sliding kmers of size 5 (pentamers) not defined by me, which therefore makes vector size dependent on sequence-length. The features intended to be used are:
  - **sh_ep** - Shape EP
  - **sh_mgw** - *Minor Groove Width*
  - **sh_helt** - *Helix Twist*
  - **sh_prot** - *Propeller Twist*
  - **sh_roll** - *Roll*

# 7 Primary analysis

Firstly we read our CSV tables into our conviniently named dataframes.

We could have concatenated both tables together, however I prefer to keep them in different files.

R Code

```
setwd("/path/to/Project/")
testdir_path <- "datasets/GB-Testing/"
prom_csvpath <- "test-1638-promoters-6mers.csv"
enha_csvname <- "test-1638-enhancers-6mers.csv"

proms <- read.csv(paste0(testdir_path,
                         prom_csvname),
                         check.names = F)
enhas <- read.csv(paste0(testdir_path,
                         enha_csvname),
                         check.names = F)
CREs <- c("Promoter","Enhancer")
```

First we get an overviwew of the dimensions of our data: NOTE: Replace this with some kind of table

```
deparse(substitute(proms))
deparse(substitute(enhas))
dim(proms)
dim(enhas)
```

```
[1] "proms"                 [1] "enhas"
[1]  1638 21830             [1]  1638 21830
```

It's noticeable the fact that we have way more columns than rows in this test table. Let's get a general glimpse of the first three promoters' rows by displaying the first 3 lines and first 18 columns pertaining to all *sequence-wise* variables computed so far plus the first 12 *kmer-wise* variables, which align with the data related to the first 3 kmers ('**AA**', '**AC**' & '**AG**'):

```
teatable(proms[1:3,1:18])
```

| A | T | C | G | temp | shan |
|---|---|---|---|---|---|
| 0.1673307 | 0.2231076 | 0.3306773 | 0.2788845 | 87.21315 | 1.956136 |
| 0.2629482 | 0.2788845 | 0.2549801 | 0.2031873 | 81.00598 | 1.990374 |
| 0.3625498 | 0.2031873 | 0.2470120 | 0.1872510 | 80.02590 | 1.948719 |

| k2.1_prod | k2.1_barc | k2.1_pals | k2.1_revc | k2.2_prod | k2.2_barc |
|---|---|---|---|---|---|
| 9 | 1.959765 | 2.177403e+09 | 1.374020e+12 | 17.11198 | 1.531862 |
| 17 | 2.633165 | 1.138401e+17 | 2.971115e+17 | 26.44579 | 2.624612 |
| 38 | 5.347025 | 3.981015e+36 | 8.100763e+29 | 24.89016 | 1.855662 |

| k2.2_pals | k2.2_revc | k2.3_prod | k2.3_barc | k2.3_pals | k2.3_revc |
|---|---|---|---|---|---|
| 3.845422e+15 | 3.525451e+11 | 26.44579 | 2.445328 | 4.788062e+13 | 1.305836e+26 |
| 2.607331e+20 | 6.308689e+14 | 24.89016 | 2.513656 | 1.320117e+14 | 6.435389e+19 |
| 1.156904e+25 | 3.573059e+12 | 34.22397 | 3.431445 | 6.011592e+17 | 7.178542e+17 |

We'll get a similar panorama when looking at the first three enhancers' rows (although it seems like *'barcode'* values appear to be significantly larger):

```
teatable(enhas[1:3,1:18])
```

| A | T | C | G | temp | shan |
|---|---|---|---|---|---|
| 0.240 | 0.236 | 0.234 | 0.290 | 85.0392 | 1.993988 |
| 0.204 | 0.286 | 0.210 | 0.300 | 84.4652 | 1.978249 |
| 0.250 | 0.280 | 0.258 | 0.212 | 82.8252 | 1.992923 |

| k2.1_prod | k2.1_barc | k2.1_pals | k2.1_revc | k2.2_prod | k2.2_barc |
|---|---|---|---|---|---|
| 36 | 10.432345 | 1.885437e+37 | 8.632413e+30 | 24.89016 | 6.027803 |
| 20 | 6.153015 | 5.673403e+20 | 1.061673e+53 | 23.33452 | 4.984254 |
| 31 | 9.936447 | 1.640775e+32 | 5.563175e+39 | 46.66905 | 9.301359 |

| k2.2_pals | k2.2_revc | k2.3_prod | k2.3_barc | k2.3_pals | k2.3_revc |
|---|---|---|---|---|---|
| 1.579134e+29 | 1.707234e+31 | 76.22611 | 13.43518 | 3.557032e+41 | 5.830130e+44 |
| 4.485619e+32 | 1.810188e+35 | 73.11484 | 13.29662 | 3.273426e+39 | 3.755909e+39 |
| 9.951038e+37 | 3.612544e+25 | 60.66976 | 12.53356 | 4.255600e+33 | 3.010280e+57 |

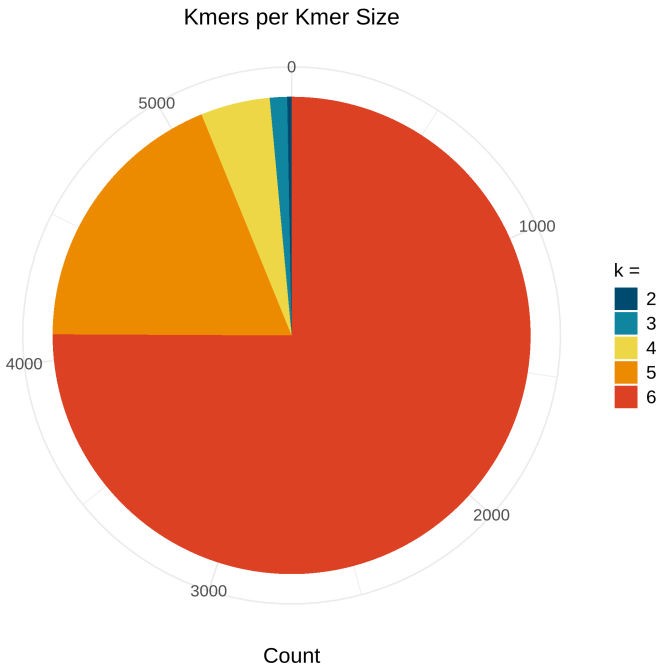
```
mean_prom <- colMeans(proms)
mean_enha <- colMeans(enhas)
sd_prom <- apply(proms, 2, sd)
sd_enha <- apply(enhas, 2, sd)

names_CREs <- rep(CREs, each = length(proms))
cre_summary <- data.frame(Type = factor(names_CREs),
                          Field = rep(names(proms),2),
                          Means = c(mean_prom,mean_enha),
                          StDevs = c(sd_prom,sd_enha))
```

```
ldf <- length(proms)
coffeetable(cre_summary[c(1:10,(ldf+1):(ldf+10)),])
```

| | Type | Field | Means | StDevs |
|---|---|---|---|---|
| 1 | Promoter | A | 1.911256e-01 | 7.145510e-02 |
| 2 | | T | 1.995826e-01 | 7.599340e-02 |
| 3 | | C | 2.962655e-01 | 8.067840e-02 |
| 4 | | G | 3.130263e-01 | 8.524230e-02 |
| 5 | | temp | 8.720208e+01 | 5.379461e+00 |
| 6 | | shan | 1.891724e+00 | 9.437000e-02 |
| 7 | | k2.1_prod | 1.196276e+01 | 8.995306e+00 |
| 8 | | k2.1_barc | 1.500971e+00 | 1.229003e+00 |
| 9 | | k2.1_pals | 2.634405e+51 | 1.065029e+53 |
| 10 | | k2.1_revc | 9.789346e+62 | 3.960331e+64 |
| 21831 | Enhancer | A | 2.653712e-01 | 5.853000e-02 |
| 21832 | | T | 2.678205e-01 | 5.815610e-02 |
| 21833 | | C | 2.351111e-01 | 5.648440e-02 |
| 21834 | | G | 2.316972e-01 | 5.435370e-02 |
| 21835 | | temp | 8.269434e+01 | 3.785299e+00 |
| 21836 | | shan | 1.959202e+00 | 3.892580e-02 |
| 21837 | | k2.1_prod | 4.093346e+01 | 1.835697e+01 |
| 21838 | | k2.1_barc | 1.208127e+01 | 5.751107e+00 |
| 21839 | | k2.1_pals | 2.567318e+112 | 1.038419e+114 |
| 21840 | | k2.1_revc | 3.321646e+121 | 1.344343e+123 |

```
# Get only 'prod' columns of each kmer

k_Ns <- c(n_ki(2), n_ki(3), n_ki(4), n_ki(5), n_ki(6))
k_inds <- c(0)
for (k_n in k_Ns) {
  k_inds <- c(k_inds, tail(k_inds,1) + 1)
  k_inds <- c(k_inds, tail(k_inds,2)[1] + k_n)
}
k_inds <- k_inds[-1]
# print(rev(k_inds)[1])
# print((ldf-6)/4)
# print(ldf)
```

```
kmer_set_sizes <- data.frame(sizes = k_Ns,
  k_sets = c("2","3","4","5","6"))

ggplot(kmer_set_sizes, aes(x = "", y = sizes,
                           fill = k_sets))+
  geom_bar(width = 1, stat = "identity") +
  labs(y = "Count", x = "",
       title = "Kmers per Kmer Size",
       fill = "k =") +
  coord_polar("y", start = 0) +
  theme_minimal() +
  scale_fill_paletteer_d("PNWColors::Bay") +
  theme(legend.position = "right",
        axis.title = element_text(size = 15),
        text = element_text(size = rel(4.25)),
        legend.text = element_text(size = 13),
        legend.title = element_text(size = 13.5),
        axis.text.y = element_text(family = "mono"),
        plot.title = element_text(size = 16, hjust = 0.5))
```



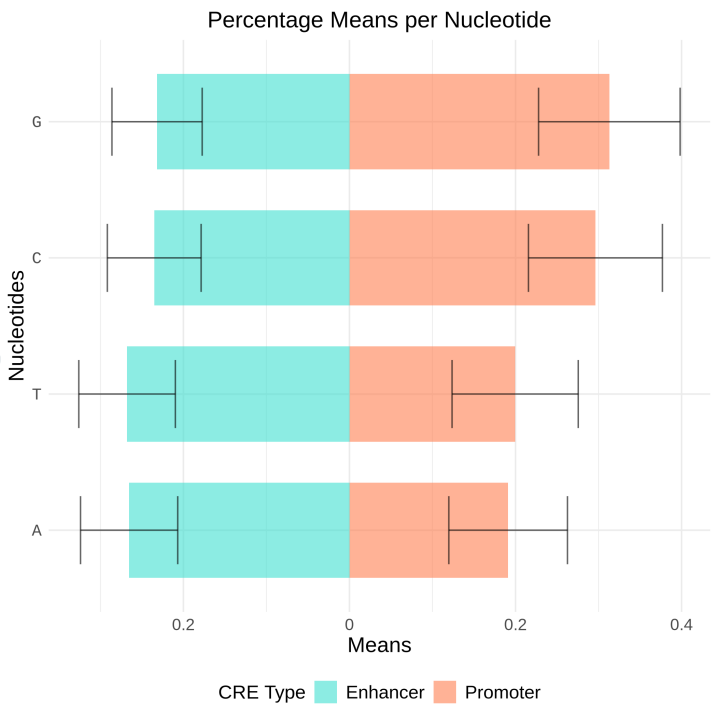

```
pe_arr <- function(seq_data)
  return(array(seq_data, dim = c(rev(k_inds)[1], 2),
    dimnames = list(1:rev(k_inds)[1], c("prom","enha"))))

indxs <- list(
  prod = pe_arr(c(seq(7,ldf-3,4), ldf+seq(7,ldf-3,4))),
  barc = pe_arr(c(seq(8,ldf-2,4), ldf+seq(8,ldf-2,4))),
  pals = pe_arr(c(seq(9,ldf-1,4), ldf+seq(9,ldf-1,4))),
  revc = pe_arr(c(seq(10,ldf,4), ldf+seq(10,ldf,4))))
```

Now, for the sake of text-space optimization, we'll make some plotting functions (principally pyramid, bar and violin plots), in order to visualize our data.

```
# To plot pyramid-plots
pyrplot_ <- function(cre_data, kmer_labels, title,
                     x_label, y_label, y_breaks) {
  CREs <- c("Enhancer", "Promoter")
  field_order <- filter(cre_data, Type==CREs[1])$Field
  fact_field_order <- factor(cre_data$Field, field_order)

  if (missing(kmer_labels)) kmer_labels <- field_order

  ggplot(cre_data) +
    geom_bar(aes(x = fact_field_order,
                 y = ifelse(Type == CREs[1],
                            -Means, Means),
                 fill = paste(Type, "Means")),
             stat = "identity", position = "identity",
             alpha = 0.6, width = 0.7) +
    geom_errorbar(aes(x = fact_field_order,
                      ymin = ifelse(Type == CREs[1],
                                    -Means + StDevs,
                                     Means - StDevs),
                      ymax = ifelse(Type == CREs[1],
                                    -Means - StDevs,
                                     Means + StDevs)),
```

```
                  width = 0.5, alpha = 0.6,
                  colour = "black") +
    coord_flip() +
    scale_x_discrete(labels = kmer_labels) +
    scale_y_continuous(breaks = y_breaks,
                       labels = abs(y_breaks)) +
    scale_fill_manual(values = c("turquoise", "coral"),
                      labels = CREs) +
    labs(y = "Means", x = x_label,
         title = title, fill = "CRE Type") +
    theme_minimal() +
    theme(legend.position = "bottom",
          axis.title = element_text(size = 15),
          text = element_text(size = rel(4.25)),
          legend.text = element_text(size = 13),
          legend.title = element_text(size = 13.5),
          axis.text.y = element_text(family = "mono"),
          plot.title = element_text(size = 16, hjust = 0.5))
}

barplot_ <- function(cre_data, y_breaks, y_axis_title="",
                     fill_legend_title="") {
  ggplot(cre_data) +
    geom_bar(aes(x = factor(Type),
                 y = Means, fill = Type),
             stat = "identity", position = "identity",
             alpha = 0.6) +
    geom_errorbar(aes(x = factor(Type),
                      ymin = Means - StDevs,
                      ymax = Means + StDevs),
                  alpha = 0.6, width = 0.5,
                  colour = "black") +
    labs(y = y_axis_title, x = "",
         fill = fill_legend_title) +
    scale_y_continuous(breaks = y_breaks,
                       labels = y_breaks) +
    scale_fill_manual(values = c("turquoise", "coral")) +
    theme_minimal() +
    theme(legend.position = "none",
          text = element_text(size = rel(4.5)),
          axis.title = element_text(size = 16))
}

hvioplot_ <- function(data, y_var, y_label = "",
                      fill_legend_title = "") {
  # x_breaks <- seq(-0.12,0.12,0.06)
  ggplot(data, aes(x = 0, y = !!sym(y_var), fill = type)) +
    geom_violinhalf(flip = 1, adjust = 0.25,
                    trim = FALSE, scale = "count",
                    position = position_dodge(width = 0),
                    linewidth = 0.1, alpha = 0.6) +
    theme_minimal() +
    # scale_x_continuous(breaks = x_breaks ,
    #                    labels = abs(x_breaks )) +
    scale_fill_manual(values = c("turquoise", "coral")) +
    labs(x = "", y = y_label, fill = fill_legend_title) +
    theme(legend.position = "none",
          text = element_text(size = rel(4.5)),
          axis.title = element_text(size = 13))
}
```

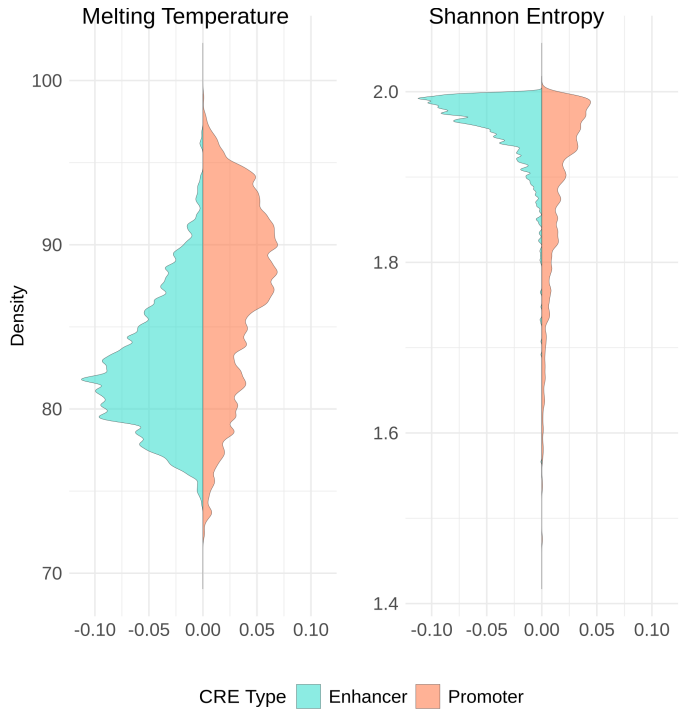
```
pyrplot_(cre_summary[c(1:4, ldf + (1:4)), ],
      x_label = "Nucleotides", y_breaks = seq(-1,1,0.2),
      title = "Percentage Means per Nucleotide")
```



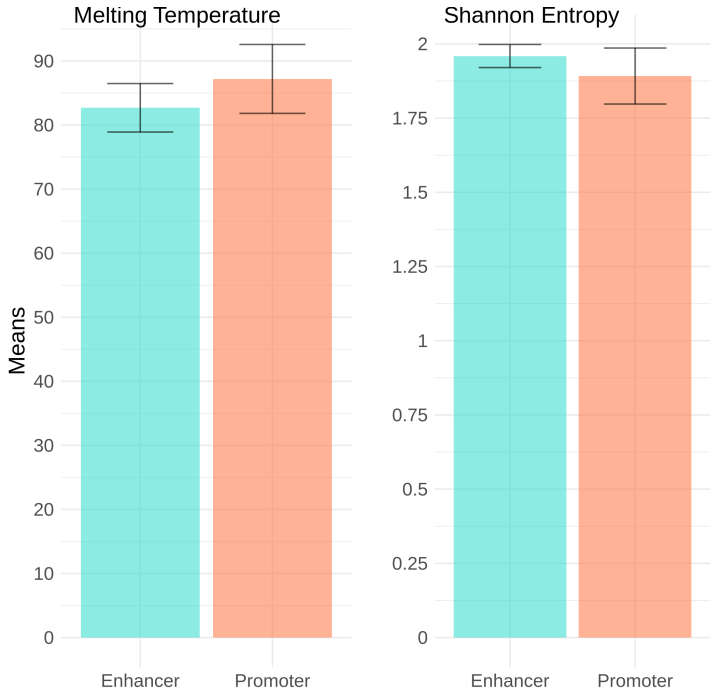

```
data_e <- cbind(type = rep("Enhancer",
                length(enhas$temp)), enhas[,5:6])
data_p <- cbind(type = rep("Promoter",
                length(proms$temp)), proms[,5:6])
subset_tm_sh <- rbind(data_e, data_p)
```

```
# Saving Melting Temperature Violin Plot
tm_violin <- hvioplot_(data = subset_tm_sh,
                       y_var = "temp",
                       y_label = "Density")

# Saving Shannon Violin Plot
sh_violin <- hvioplot_(data = subset_tm_sh,
                       y_var = "shan",
                       fill_legend_title = "CRE Type")

# Get legend
legend_violin <- get_legend_bypass(sh_violin +
  guides(color = guide_legend(nrow = 1)) +
  theme(legend.position = "bottom",
        legend.title = element_text(size = 13.5),
        legend.text = element_text(size = 13)))

# Merging of Violin-Plots
grid_violin <- plot_grid(tm_violin, sh_violin, NULL,
                 ncol = 3, rel_widths = c(1,1,0.1),
                 labels = c("Melting Temperature",
                 "Shannon Entropy"), label_size = 16,
                 label_fontface = "plain", vjust = 1,
                 hjust = c(-0.38, -0.48))
```

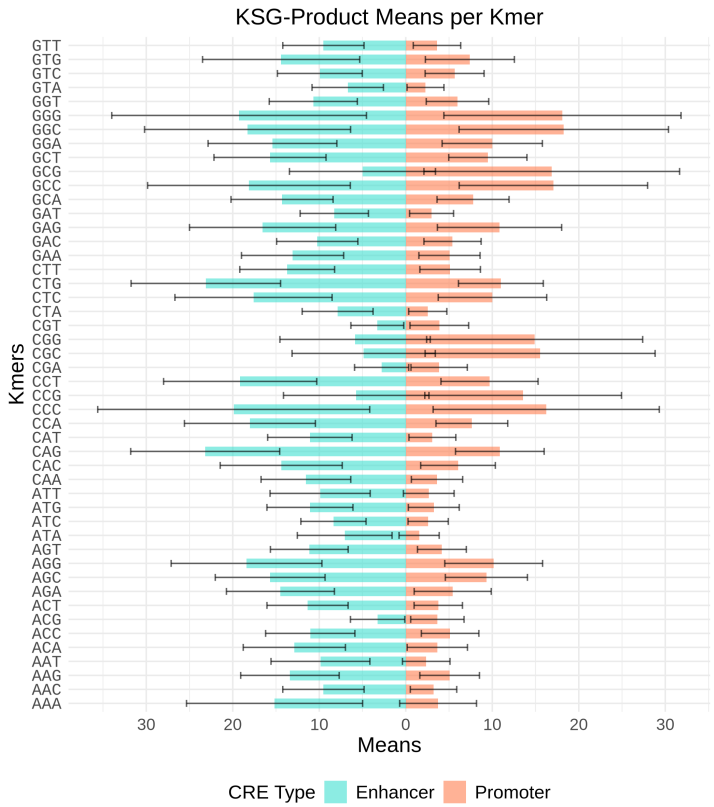
```
# Adding legend at the bottom
plot_grid(grid_violin, legend_violin,
          ncol = 1, rel_heights = c(12,1))
```



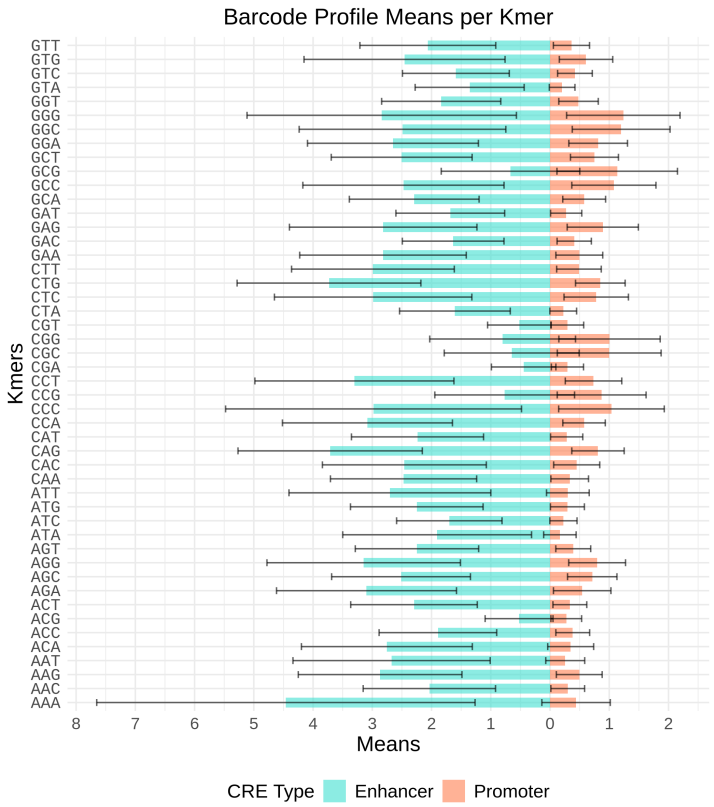

```
# Saving Melting Temperature Bar-Plot
temp_plot <- barplot_(filter(cre_summary, Field=="temp"),
                      y_breaks = seq(0, 100, 10),
                      y_axis_title = "Means")

# Saving Shannon Coefficient Bar-Plot
shan_plot <- barplot_(filter(cre_summary, Field=="shan"),
                      y_breaks = seq(0, 2, 0.25),
                      fill_legend_title = "CRE Type")

# Merging of Bar-Plots
plot_grid(temp_plot, shan_plot,
    ncol = 2, rel_widths = c(1,1),
    vjust = 1, hjust = c(-0.35, -0.48),
    label_size = 16, label_fontface = "plain",
    labels = c("Melting Temperature", "Shannon Entropy"))
```




```
kmer_names <- combi_kmers(k=3)[1:48]

pyrplot_(cre_summary[indxs$prod[17:64,],], kmer_names,
         x_label = "Kmers", y_breaks = seq(-30,30,10),
         title = "KSG-Product Means per Kmer")
```




```
pyrplot_(cre_summary[indxs$barc[17:64,],], kmer_names,
         x_label = "Kmers", y_breaks = seq(-10,10,1),
         title = "Barcode Profile Means per Kmer")
```


Barcode Profile Means per Kmer
Kmers
GTT
GTG
GTC
GTA
GGT
GGG
GGC
GGA
GCT
GCG
GCC
GCA
GAT
GAG
GAC
GAA
CTT
CTG
CTC
CTA
CGT
CGG
CGC
CGA
CCT
CCG
CCC
CCA
CAT
CAG
CAC
CAA
ATT
ATG
ATC
ATA
AGT
AGG
AGC
AGA
ACT
ACG
ACC
ACA
AAT
AAG
AAC
AAA
8
7
6
5
4
3
2
1
0
1
2
Means
CRE Type
Enhancer
Promoter