HRM-ML Paper

gaoch

2024-09-16

Table of contents

Preface

This book contains the supplementary information of the manuscript entitled “Precise prediction of synthetic community structure with high-resolution melting curve and machine learning” authored by Chun-Hui Gao, Jiaqi He, et. al.

> **Note**
>
> This is a Quarto book, created from markdown and executable code.
> See Knuth (1984) for additional discussion of literate programming.
> To learn more about Quarto books visit https://quarto.org/docs/books.

1 Single Species Modeling

Investigation of the amplification, melting curve and Ct standard curve of single species PCRs using Linear modeling.

1.1 Global setting

Here we load packages and define several frequently used variables.

```
# load required packages
library(tidyverse)
library(tidymodels)
library(mcmodel)
library(cowplot)

# default theme
theme_set(theme_bw() +
            theme(legend.key.size = unit(0.4,'cm'),
                  legend.key.height = unit(.4, 'cm')))

# global setting
strain_label = c("label_E", "label_P")
strain_name = c("EC","PP")
strain_color = c("red3", "purple3")

# well position
ec_single_well = paste0(rep(LETTERS[1:16], times = 3), rep(1:3, each = 16))
pp_single_well = paste0(rep(LETTERS[1:16], times = 3), rep(4:6, each = 16))
```

```
gradient_matrix_well = paste0(rep(LETTERS[1:16],times = 16), rep(7:22,each = 16))
strain_single_well = list(EC = ec_single_well, PP = pp_single_well)

# set seed
set.seed(0)
```

1.2 Data process

We use one experiment to start the story.

The experiment of melting curve analysis (and RT-PCR) was performed with a QuantStudio Software supported machine. Therefore, the results file used here is the plain text output of full results of the QuantStudio software (V1.5). In `mcmodel` package, it has the `read_quantstudio()` function to read in the data and transform the full record to a `QuantStudioRaw` class object.

```
# read qPCR run results
raw_file = xfun::magic_path("cycle30-experiment1.txt")
quantstudio_raw = read_quantstudio(raw_file)
quantstudio_raw
```

```
An object of class 'QuantStudioRaw':
  Slots: [Sample Setup], [Raw Data], [Amplification Data],
         [Multicomponent Data], [Results], [Reagent Information], [Melt
         Curve Raw Data], Meta;
```

The plate setting used in melting curve analysis is provided in a `csv` file. Original value for each column is the times of two-fold dilution, and we transform them into DNA quantity as related to the original concentration (assumed to be 1). Note: a label value of `16` means not contain this species.

```
# read plate setting
plate_file = xfun::magic_path("modeling-plate-labels.csv")
plate = read.csv(plate_file)
plate2 = plate |>
```

```
  drop_na() |>
  mutate(label_E = ifelse(label_E == 0, 1, ifelse(label_E == 16, 0, 1/2^label_E)),
         label_P = ifelse(label_P == 0, 1, ifelse(label_P == 16, 0, 1/2^label_P)))
```

1.3 Well concentration

A 384-well PCR plate was divided into two parts. 1) the dilution of single species DNA template of *E. coli* (Figure 1.1a) and *P. putida* (Figure 1.1b), 2) the gradient matrix of two-species DNA mixtures (Figure 1.1c).

```
p_concentration = lapply(seq_along(strain_label), function(i){
  plate2  |>
    dplyr::filter(well_position %in% strain_single_well[[i]]) |>
    plot_384_single_concentration(strain_label[[i]], well_size = 1.5) +
    scale_color_gradient(high = strain_color[[i]],
                         low = "white",
                         trans = "log2",
                         na.value = "grey90") +
    coord_equal() +
    theme(legend.position = 'none')
})

plate3 = plate2 |>
  dplyr::filter(well_position %in% gradient_matrix_well)
p_matrix = plot_384_community_structure(plate3) +
  scale_fill_manual(values = c("red3","purple3")) +
  coord_equal() +
  theme(legend.position = 'none')

plot_grid(p_concentration[[1]], p_concentration[[2]], p_matrix,
          ncol = 2, labels = 'auto')
```

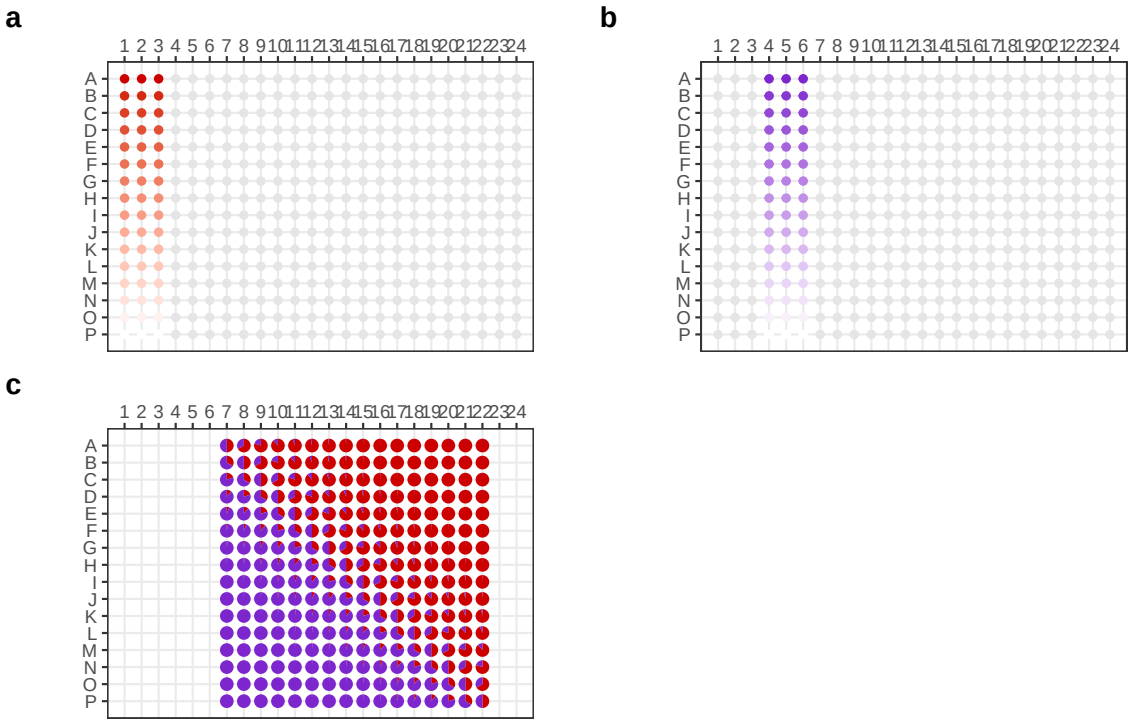


Figure 1.1: Settings of a 384-well PCR plate.

1.4 Amplification plot

```
# plot amplification curve
qs_amplification = get_quantstudio_amplication(quantstudio_raw)
```

```
p_amp = lapply(seq_along(strain_label), function(i){
  df = qs_amplification  |>
    dplyr::filter(well_position %in% strain_single_well[[i]]) |>
    left_join(plate2)

  ggplot(df, aes(cycle, delta_rn, group = well_position)) +
    geom_line(aes(color = .data[[strain_label[[i]]]])) +
    geom_hline(yintercept = 0.124, linetype = 'dashed', color = 'grey') +
    geom_text(x = -Inf, y = 0.124, label = 'Ct threshold',
```

```
                hjust = -.25, vjust = -0.5, color = 'grey') +
    scale_color_gradient(
      high = strain_color[[i]], low = "white", trans = 'log2',
      labels = trans_format("log2", label_number_auto()),
      breaks = 2^c(0, -5, -10, -15)
    ) +
    labs(color = expression(log[2]*Q), y = 'fluorescence') +
    theme(legend.position = "inside",
          legend.position.inside = c(0.38,0.62))
})
```

```
Joining with `by = join_by(well_position)`
Joining with `by = join_by(well_position)`
```

```
plot_grid(plotlist = p_amp, labels = "auto")
```

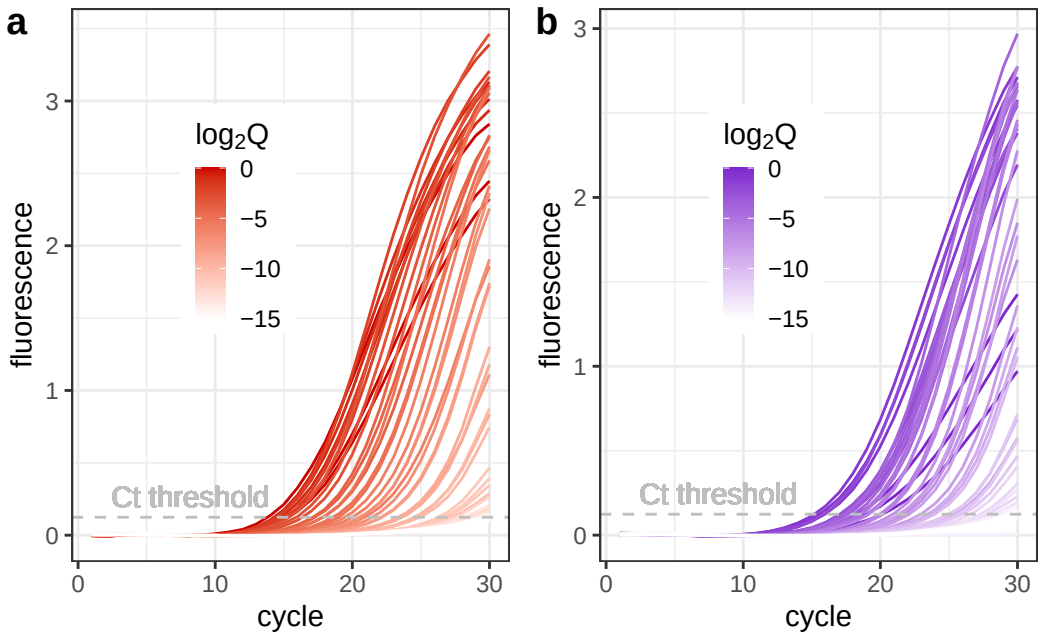


Figure 1.2: Amplification curves of singe species DNAs

1.5 Ct model of single species

```
# check ct calling results
qs_results = get_quantstudio_result(quantstudio_raw) |>
  dplyr::select(well_position, ct) |>
  dplyr::mutate(ct = as.numeric(ct)) |>
  left_join(plate2) |>
  pivot_longer(cols = starts_with("label_"),
               names_to = "strain",
               values_to = "quantity")  |>
  dplyr::filter(quantity != 0, !is.na(ct))
```

```
Warning: There was 1 warning in `dplyr::mutate()`.
i In argument: `ct = as.numeric(ct)`.
Caused by warning:
! NAs introduced by coercion
```

```
Joining with `by = join_by(well_position)`
```

```
mono_ct = qs_results  |>
  dplyr::filter(well_position %in% unlist(strain_single_well))
mono_ct
```

```
# A tibble: 90 x 4
   well_position    ct strain  quantity
   <chr>         <dbl> <chr>      <dbl>
 1 A1             15.4 label_E      1
 2 A2             13.9 label_E      1
 3 A3             15.2 label_E      1
 4 A4             18.3 label_P      1
 5 A5             20.1 label_P      1
 6 A6             19.0 label_P      1
 7 B1             14.5 label_E      0.5
```

```
 8 B2             14.3 label_E      0.5
 9 B3             14.8 label_E      0.5
10 B4             15.2 label_P      0.5
# i 80 more rows
```

```
p_ct = lapply(seq_along(strain_label), function(i){
  qs_results |>
    dplyr::filter(well_position %in% strain_single_well[[i]]) |>
    ggplot(aes(ct, log2(quantity))) +
    geom_smooth(method = 'lm', color = strain_color[[i]]) +
    geom_point(shape = 21, color = strain_color[[i]]) +
    labs(y = expression(log[2]*Q), x = 'Ct')
})

plot_grid(plotlist = p_ct, labels = "auto")
```

```
`geom_smooth()` using formula = 'y ~ x'
`geom_smooth()` using formula = 'y ~ x'
```

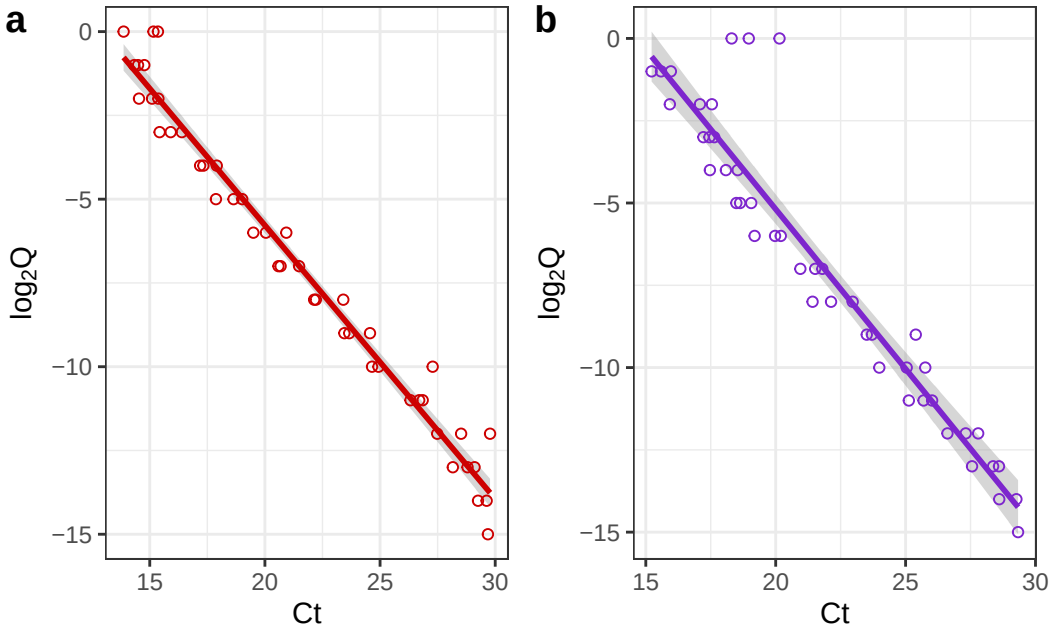


Figure 1.3: Correlation of Ct values and log2-transformed DNA quantities

1.6 Melting curve gradient

```
mc = quantstudio2mc(quantstudio_raw, plate = plate2, primer = "V4")
mc = filterData(mc, from = 80, to = 90) |>
  transformData(step = 0.1)
```

```
plot_mc_single = function(mc, label_color, vline_color){
  # get tm value (median)
  tm = mc_get_tm(mc, npeaks = 1) |> pull(peak_position) |> median()

  df = mc2tbl(mc) |>
    select(-date, -primer) |>
    summarize(derivative = median(derivative),
              .by = all_of(c("temperature", label_color)))

  # plot mc
  ggplot(df,
         aes(temperature, derivative,
             color = .data[[label_color]],
             group = .data[[label_color]])) +
    geom_line() +
    geom_vline(aes(xintercept = I(tm)),
               linetype = 'dashed',
               color = vline_color) +
    geom_text(x = tm, y = Inf, hjust = -0.1, vjust = 2,
              label = paste0("Tm = ", tm, "°C"),
              color = vline_color)  +
    scale_x_continuous(breaks = c(80, 85, 90)) +
    labs(x = "temperature (°C)",
         y = "fluorescence",
         color = expression(log[2]*Q)) +
    theme(legend.position = "inside",
          legend.position.inside = c(0.75,0.5))
}
```

```
p_mc = lapply(seq_along(strain_label), function(i){
  filterData(mc, well_position = strain_single_well[[i]]) |>
    plot_mc_single(strain_label[[i]], strain_color[[i]]) +
    scale_color_gradient(
      high = strain_color[[i]],
      low = "white",
      trans = 'log2',
      labels = trans_format("log2", label_number_auto()),
      breaks = 2^c(0, -5, -10, -15)
    )
})

plot_grid(plotlist = p_mc, ncol = 2, labels = "auto")
```

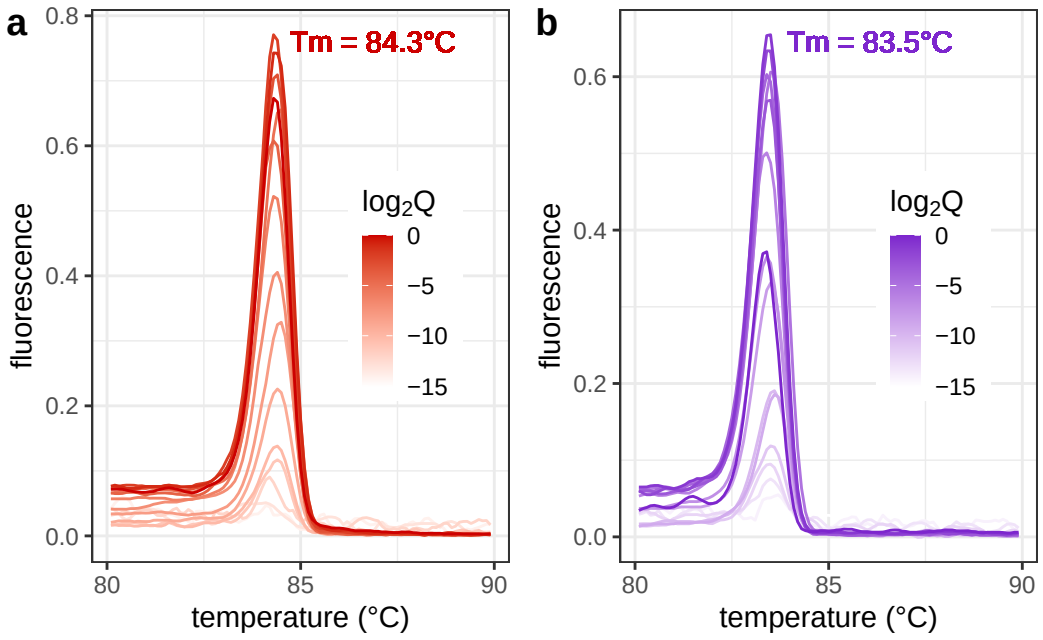


Figure 1.4: Melting curves of single species DNAs

1.7 Peak calling

Detect peak and get peak features.

```
mono_mc = lapply(seq_along(strain_label), function(i){
  x = filterData(mc, well_position = strain_single_well[[i]]) |>
    transformData(step = 0.1)
  df = mc2tbl(x) |>
    select(-date, -primer) |>
    nest(data = c(temperature, derivative)) |>
    pivot_longer(cols = starts_with("label_"),
                 names_to = "strain",
                 values_to = "quantity")
  peaks = lapply(df$data,
                 mcmodel:::detect_tm,
                 zero = "+",
                 npeaks = 1,
                 sortstr = TRUE,
                 threshold = 0) |>
    bind_rows()
  bind_cols(df, peaks)
}) |>
  bind_rows() |>
  dplyr::filter(quantity != 0)

mono_mc$peak_area = sapply(1:nrow(mono_mc), function(i){
  d = mono_mc$data[[i]] |>
    filter(temperature >= mono_mc$peak_start[[i]],
           temperature <= mono_mc$peak_end[[i]])
  pracma::trapz(d$temperature, d$derivative)
})

peak_cols = paste('peak',
                  c('height','area','start','position','end'),
                  sep = "_")
```

```
mono_mc = mono_mc |>
  dplyr::select(well_position, strain, quantity, matches(peak_cols))

mono_mc
```

```
# A tibble: 96 x 8
   well_position strain  quantity peak_height peak_area peak_start peak_position
   <chr>         <chr>      <dbl>       <dbl>     <dbl>      <dbl>         <dbl>
 1 A1            label_E    1           0.644     0.740       82.1          84.3
 2 A2            label_E    1           0.801     0.938       82.3          84.3
 3 A3            label_E    1           0.673     0.791       82.2          84.3
 4 B1            label_E    0.5         0.761     0.898       82.4          84.4
 5 B2            label_E    0.5         0.882     1.06        83            84.8
 6 B3            label_E    0.5         0.751     0.905       82.1          84.3
 7 C1            label_E    0.25        0.771     0.946       82            84.3
 8 C2            label_E    0.25        0.844     1.06        82.1          84.4
 9 C3            label_E    0.25        0.737     0.871       82.4          84.3
10 D1            label_E    0.125       0.691     0.838       82.2          84.3
# i 86 more rows
# i 1 more variable: peak_end <dbl>
```

Correlations of DNA quantity to peak features.

```
plots = lapply(peak_cols, function(x){
  ggplot(mono_mc, aes(log2(quantity), .data[[x]], color = .data$strain)) +
    geom_point(size = 0.2) +
    geom_smooth(method = MASS::rlm) +
    labs(x = expression(log[2]*Q),
         y = sub(pattern = "_", replacement = " ", x)) +
    scale_color_manual(values = strain_color) +
    theme(legend.position = "none")
})

plot_grid(plotlist = plots, align = 'hv', ncol = 3, labels = "auto")
```

```
`geom_smooth()` using formula = 'y ~ x'
`geom_smooth()` using formula = 'y ~ x'
`geom_smooth()` using formula = 'y ~ x'
`geom_smooth()` using formula = 'y ~ x'
`geom_smooth()` using formula = 'y ~ x'
```

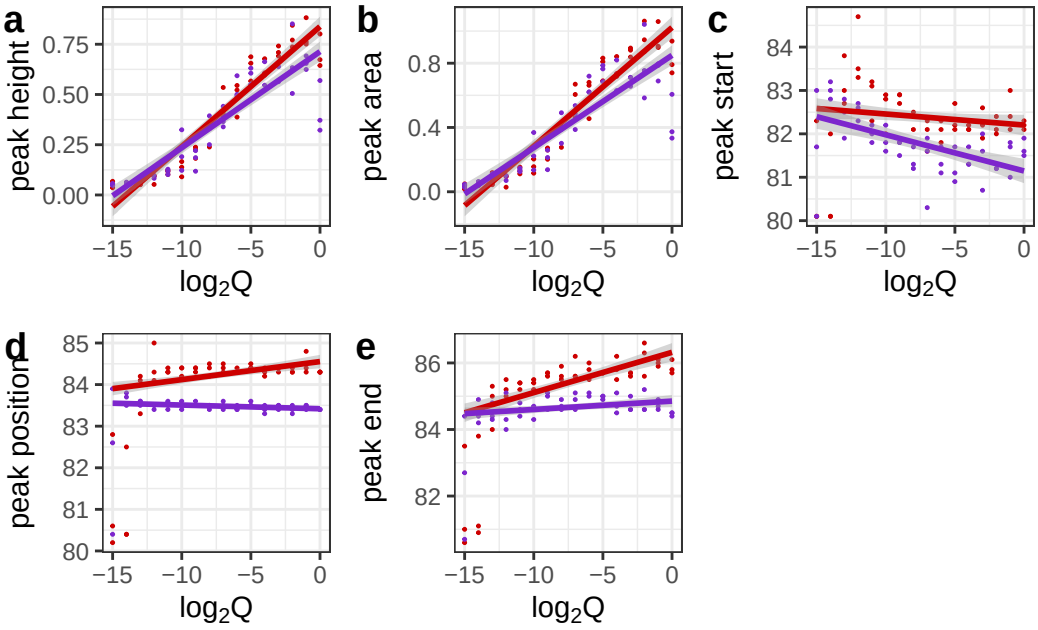


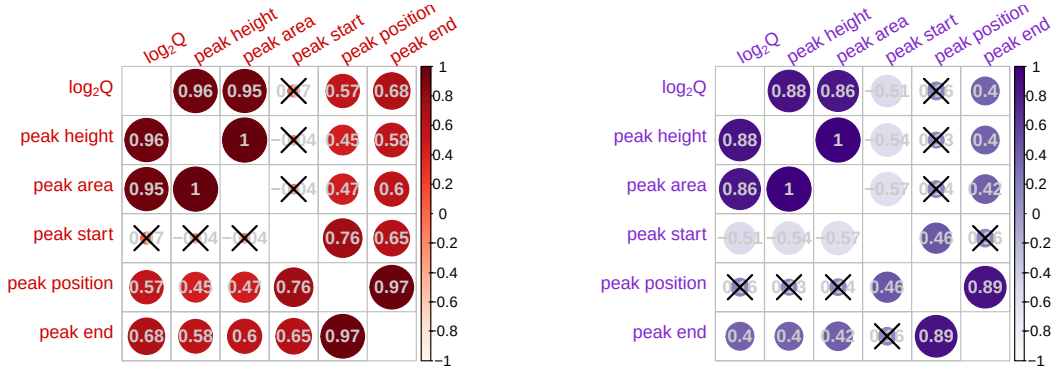
Figure 1.5: Correlations of DNA quantity and peak-associated features extracted from single species melting curves.

```
library(corrplot)
```

```
corrplot 0.92 loaded
```

```
mono_mc_vars = mono_mc |>
  dplyr::mutate(log2Q = log2(quantity)) |>
  dplyr::select(well_position, log2Q, matches(peak_cols)) |>
  dplyr::rename(`$log[2]*Q` = log2Q) |>
  dplyr::rename_with(.fn = function(x) sub("_", " ", x),
                     .cols = matches(peak_cols))
```

```
cols = c('Reds','Purples')
p_mc_corrplot = lapply(seq_along(strain_label), function(i){
  vars = mono_mc_vars  |>
    dplyr::filter(well_position %in% strain_single_well[[i]]) |>
    dplyr::select(-well_position)
  M = cor(vars)
  T = cor.mtest(vars, conf.level = 0.95)
  corrplot::corrplot(M,
                     diag = FALSE, addCoef.col = 'grey80',
                     col = COL1(cols[[i]]),
                     tl.srt = 30,
                     tl.col = strain_color[[i]],
                     p.mat = T$p)
})
```



(a) *E. coli* (b) *P. putida*

Figure 1.6: Correlations of DNA quantity and peak-associated features extracted from single species melting curves.

```
p_peak_height = lapply(seq_along(strain_label), function(i){
  mono_mc |>
    dplyr::filter(well_position %in% strain_single_well[[i]]) |>
    ggplot(aes(peak_height, log2(quantity))) +
    geom_smooth(method = 'lm', color = strain_color[[i]]) +
    geom_point(shape = 21, color = strain_color[[i]]) +
```

```
    labs(x = 'peak height',
         y = expression(log[2]*Q))
})

plot_grid(plotlist = p_peak_height, ncol = 2, labels = "auto")
```

```
`geom_smooth()` using formula = 'y ~ x'
`geom_smooth()` using formula = 'y ~ x'
```

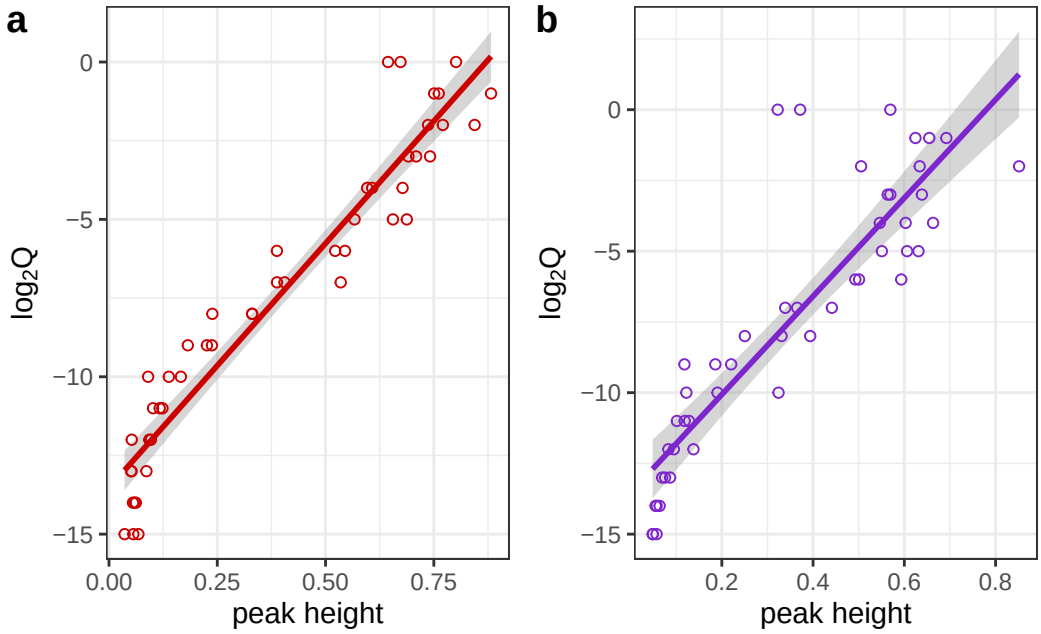


Figure 1.7: Correlation of log2-transformed DNA quantities and peak heights

```
p_peak_area = lapply(seq_along(strain_label), function(i){
  mono_mc |>
    dplyr::filter(well_position %in% strain_single_well[[i]]) |>
    ggplot(aes(peak_area, log2(quantity))) +
    geom_smooth(method = 'lm', color = strain_color[[i]]) +
    geom_point(shape = 21, color = strain_color[[i]]) +
    labs(x = 'peak area', y = expression(log[2]*Q))
```

```
})

plot_grid(plotlist = p_peak_area, ncol = 2, labels = "auto")
```

```
`geom_smooth()` using formula = 'y ~ x'
`geom_smooth()` using formula = 'y ~ x'
```

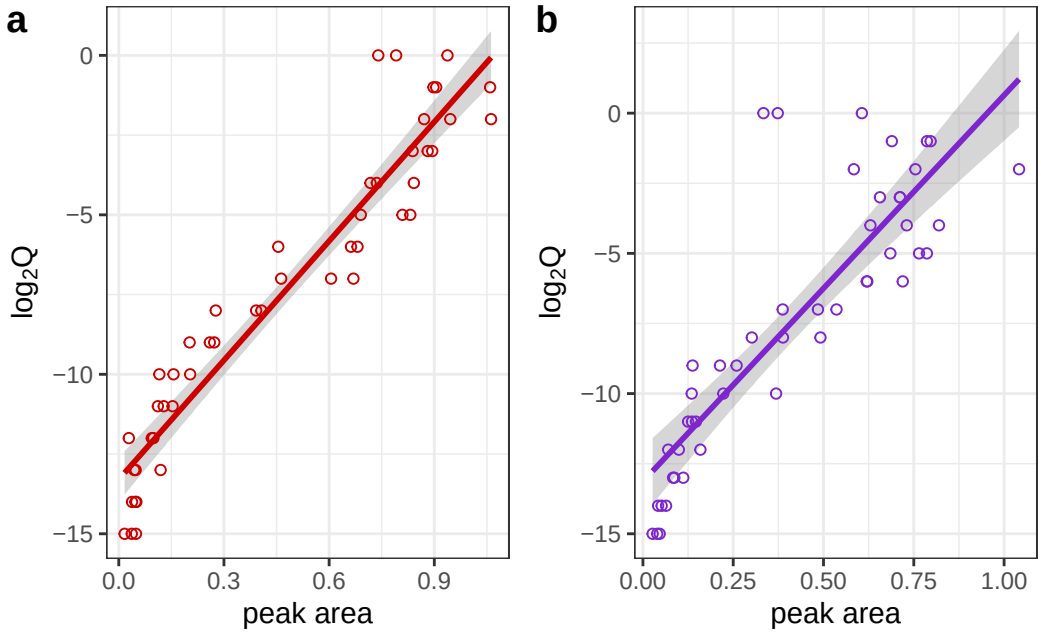


Figure 1.8: Correlation of log2-transformed DNA quantities and peak areas

1.8 Comparision of Ct and MC modelling

```
qs_single_results = qs_results |>
  dplyr::filter(well_position %in% unlist(strain_single_well))
qs_single_results
```

```
# A tibble: 90 x 4
```

```
   well_position    ct strain  quantity
   <chr>         <dbl> <chr>      <dbl>
 1 A1             15.4 label_E      1
 2 A2             13.9 label_E      1
 3 A3             15.2 label_E      1
 4 A4             18.3 label_P      1
 5 A5             20.1 label_P      1
 6 A6             19.0 label_P      1
 7 B1             14.5 label_E      0.5
 8 B2             14.3 label_E      0.5
 9 B3             14.8 label_E      0.5
10 B4             15.2 label_P      0.5
# i 80 more rows
```

```
model_metric = function(formula, data, ...){
  model = lm(formula, data)
  summary = summary(model)
  quosures = enquos(...)
  tibble(data = deparse(substitute(data)),
         formula = paste(as.character(formula), collapse = " "),
         metric = c("r_squared", "adj_r_squared"),
         value = c(summary$r.squared, summary$adj.r.squared)) |>
    dplyr::mutate(!!!quosures, .before = 1)
}
```

```
plot_model_metric = function(model, test_data, color){
  predictions = augment(model, newdata = test_data)
  metrics = metric_set(rmse, rsq, mae)
  model_metrics = metrics(predictions,
                          truth = 'log2quantity',
                          estimate = .fitted)
  annotation = paste(model_metrics[['.metric']],
                     round(model_metrics[['.estimate']], digits = 2),
                     sep = ": ",
                     collapse = "\n")
  ggplot(predictions, aes(`log2quantity`, `.fitted`, color = I(color))) +
```

```
    geom_point(shape = 21) +
    geom_abline(slope = 1, linetype = 'dashed', color = color) +
    annotate("text", x = -Inf, y = Inf, label = annotation,
             color = I(color),
             hjust = -0.1, vjust = 1.1) +
    coord_equal() +
    xlim(c(-16,0)) + ylim(c(-16, 0)) +
    labs(x = expression(log[2]*Q[true]), y = expression(log[2]*Q[pred]))
}

p_model_ct = lapply(seq_along(strain_label), function(i){
  data = mono_ct |>
    dplyr::filter(well_position %in% strain_single_well[[i]]) |>
    mutate(log2quantity = log2(quantity))
  data_split_single = initial_split(data)
  data_train_single = training(data_split_single)
  data_test_single = testing(data_split_single)
  model = lm(log2quantity ~ ct, data_train_single)
  p = plot_model_metric(model, data_test_single,
                        strain_color[[i]]) + labs(subtitle = 'Ct model')
  return(p)
})

p_model_mc = lapply(seq_along(strain_label), function(i){
  data = mono_mc |>
    dplyr::filter(well_position %in% strain_single_well[[i]]) |>
    mutate(log2quantity = log2(quantity))
  data_split_single = initial_split(data)
  data_train_single = training(data_split_single)
  data_test_single = testing(data_split_single)
  model = lm(log2quantity ~ peak_area + peak_height + peak_start + peak_end,
             data_train_single)
  plot_model_metric(model, data_test_single, strain_color[[i]]) +
    labs(subtitle = 'MC model')
})
```

```
plot_grid(plotlist = c(p_model_ct, p_model_mc), align = 'hv', labels = "auto")
```

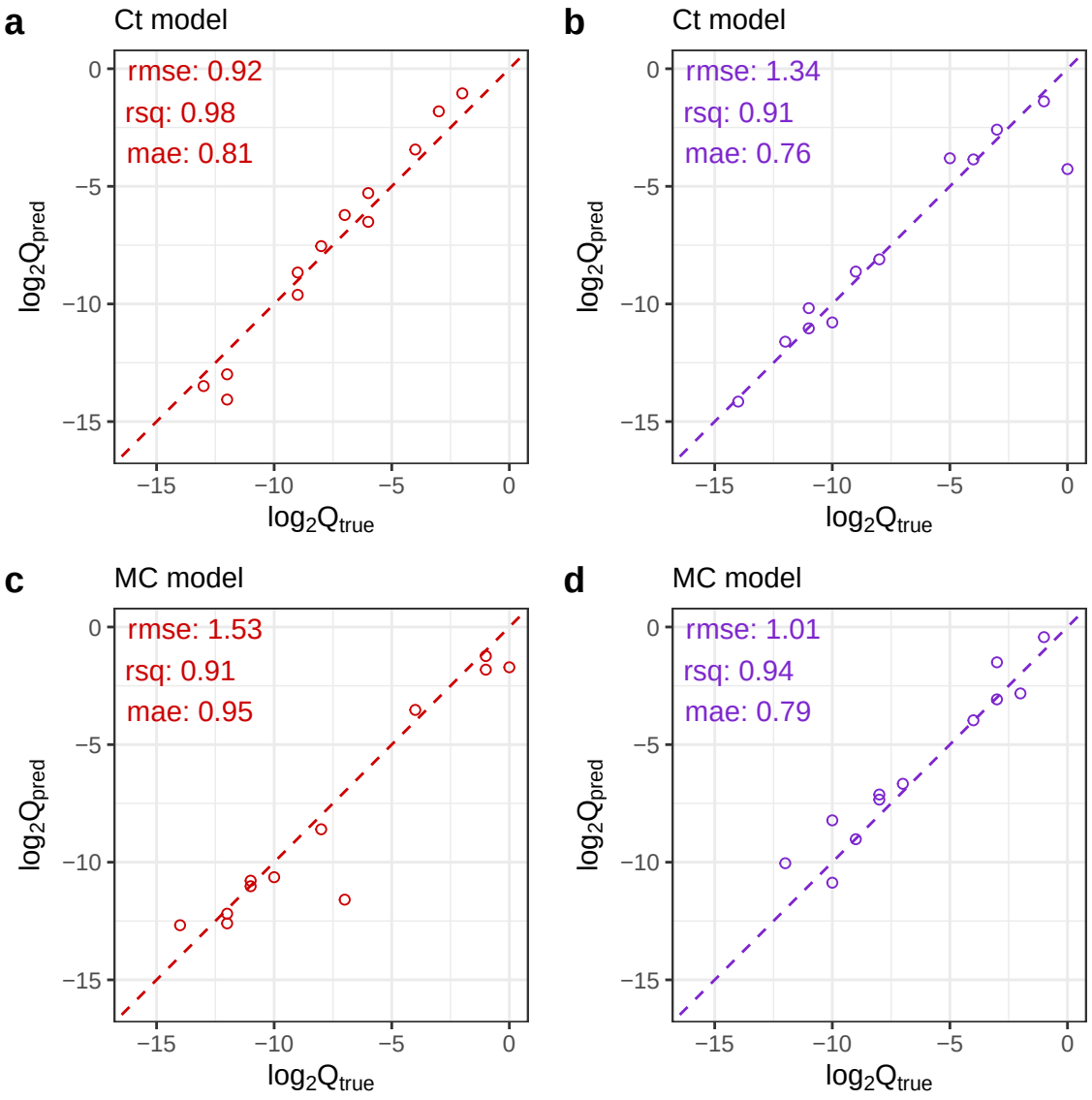


Figure 1.9: Comparison of Ct and Mc linear modelling

1.9 Combined Results

```
plot_grid(plotlist = c(p_amp, p_mc, p_model_ct, p_model_mc),
          align = 'hv',
          ncol = 4, nrow = 2, labels = 'auto')

ggsave("figures/figure1.jpg")
```

```
Saving 9.5 x 5.87 in image
```

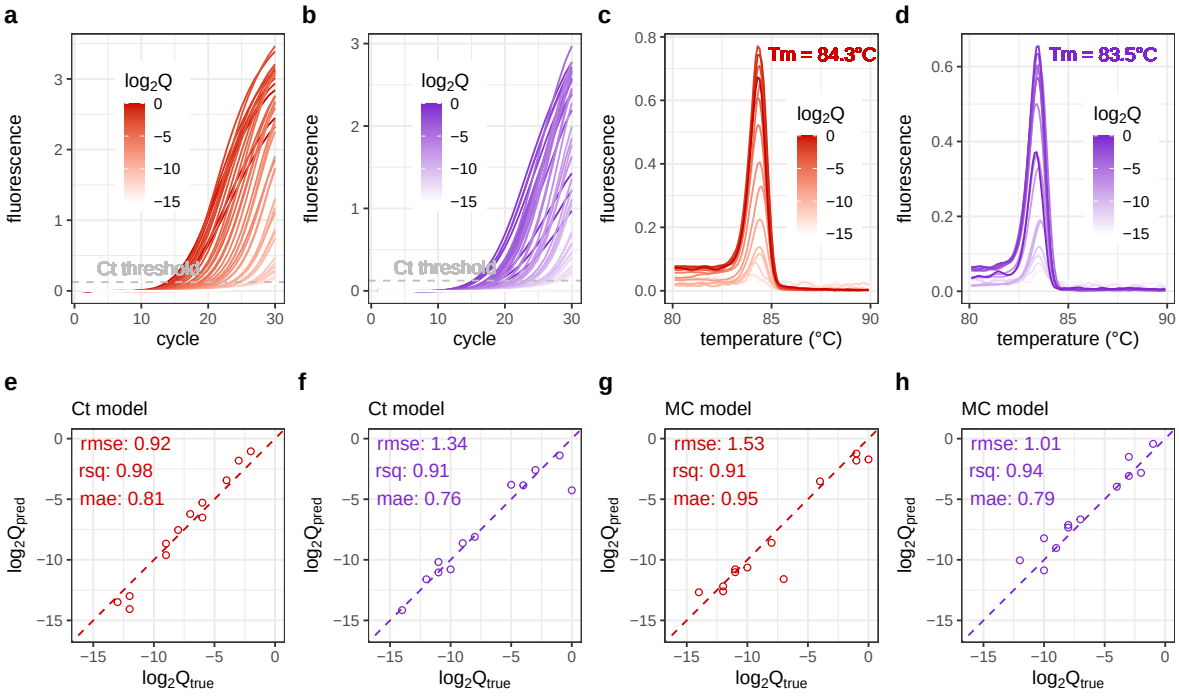


Figure 1.10: Linear modelling and prediction of single species abundance with threshold cycle (Ct) and melting curve features in a two-species SynComs (red, *E. coli.* purple, *P. putida.* Same below).

2 Data preprocess

2.1 Files

The plate setting used in melting curve analysis is provided in a `csv` file. Original value for each column is the times of two-fold dilution, and we transform them into DNA quantity as related to the original concentration (assumed to be 1).

Note: a label value of `16` means not contain this species.

```
library(dplyr)
```

```
Attaching package: 'dplyr'

The following objects are masked from 'package:stats':

    filter, lag

The following objects are masked from 'package:base':

    intersect, setdiff, setequal, union
```

```
# experimental results
files = list.files("data-raw/modeling-qPCR", pattern = ".txt", full.names = TRUE)

# plate setting - layout
plate = read.csv(xfun::magic_path("modeling-plate-labels.csv")) |>
  mutate(label_E = ifelse(label_E == 0, 1, ifelse(label_E == 16, 0, 1/2^label_E)),
         label_P = ifelse(label_P == 0, 1, ifelse(label_P == 16, 0, 1/2^label_P)))
```

2.2 Data Process By `mcmodel`

Extract cycle and repeat from file names.

```
library(stringr)
get_cycle = function(filename){
  str_extract(filename, "cycle[0-9]{2}")  |>
    str_remove("cycle")  |>
    as.numeric()
}

get_repeat = function(filename){
  str_extract(filename, "experiment[0-9]")  |>
    str_remove("experiment")  |>
    as.numeric()
}
```

Read in melting curve and extract data from temperature 80 to 90 °C, transform raw signal to step 0.1 signal by interpolations.

```
library(mcmodel)

mc0512 = lapply(seq_along(files), function(i){
  filename = files[[i]]
  all = read_quantstudio(filename)
  sample = plate  |>
    mutate(cycle = get_cycle(filename), rep = get_repeat(filename))
  mc = quantstudio2mc(all, primer = "V4", plate = sample)  |>
    filterData(from = 80, to = 90)  |>
    transformData(step = 0.1)
  return(mc)
})
```

Transform `MeltingCurve` object to data frame.

```r
data0512 = lapply(mc0512, mc_tbl2wider) |> bind_rows()

head(data0512)
```

```
# A tibble: 6 x 105
  experiment_date well_position label_E label_P cycle   rep  T80.1  T80.2  T80.3
  <date>          <chr>           <dbl>   <dbl> <dbl> <dbl>  <dbl>  <dbl>  <dbl>
1 2023-05-13      A1                  1       0    30     1 0.0680 0.0680 0.0680
2 2023-05-13      A2                  1       0    30     1 0.0778 0.0789 0.0790
3 2023-05-13      A3                  1       0    30     1 0.0718 0.0720 0.0720
4 2023-05-13      A4                  0       1    30     1 0.0623 0.0611 0.0609
5 2023-05-13      A5                  0       1    30     1 0.0339 0.0339 0.0351
6 2023-05-13      A6                  0       1    30     1 0.0344 0.0368 0.0382
# i 96 more variables: T80.4 <dbl>, T80.5 <dbl>, T80.6 <dbl>, T80.7 <dbl>,
#   T80.8 <dbl>, T80.9 <dbl>, T81 <dbl>, T81.1 <dbl>, T81.2 <dbl>, T81.3 <dbl>,
#   T81.4 <dbl>, T81.5 <dbl>, T81.6 <dbl>, T81.7 <dbl>, T81.8 <dbl>,
#   T81.9 <dbl>, T82 <dbl>, T82.1 <dbl>, T82.2 <dbl>, T82.3 <dbl>, T82.4 <dbl>,
#   T82.5 <dbl>, T82.6 <dbl>, T82.7 <dbl>, T82.8 <dbl>, T82.9 <dbl>, T83 <dbl>,
#   T83.1 <dbl>, T83.2 <dbl>, T83.3 <dbl>, T83.4 <dbl>, T83.5 <dbl>,
#   T83.6 <dbl>, T83.7 <dbl>, T83.8 <dbl>, T83.9 <dbl>, T84 <dbl>, ...
```

2.3 Save Modeling Data

Save data frame to disk.

```r
write.csv(data0512, file = "data-clean/20230512.csv", row.names = FALSE)
```

3 Model Selection

We test different models in Python.

3.1 Global setting

Here we load packages and define several frequently used variables.

```
# load required packages
library(tidyverse)
library(tidymodels)
library(mcmodel)
library(cowplot)

# default theme
theme_set(theme_bw() +
            theme(legend.key.size = unit(0.4,'cm'),
                  legend.key.height = unit(.4, 'cm')))

# global setting
strain_label = c("label_E", "label_P")
strain_name = c("EC","PP")
strain_color = c("red3", "purple3")

# well position
ec_single_well = paste0(rep(LETTERS[1:16], times = 3), rep(1:3, each = 16))
pp_single_well = paste0(rep(LETTERS[1:16], times = 3), rep(4:6, each = 16))
gradient_matrix_well = paste0(rep(LETTERS[1:16],times = 16), rep(7:22,each = 16))
```

```
strain_single_well = list(EC = ec_single_well, PP = pp_single_well)

# set seed
set.seed(0)
```

3.2 Data Preparation

```
mc_ml_data = read.csv("data-clean/20230512.csv") |>
  filter(well_position %in% gradient_matrix_well,
                rep == 1,
                cycle == 30) |>
  select(starts_with('label_'), starts_with('T')) |>
  mutate(label_E = log2(label_E), label_P = log2(label_P))

write.csv(mc_ml_data, 'data-clean/model-selection-data.csv')
```

3.3 Models

```
# import linear models
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn.linear_model import ElasticNet
from sklearn.neighbors import KNeighborsRegressor

# import ensemble regressors
from sklearn.ensemble import AdaBoostRegressor
from sklearn.ensemble import BaggingRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.ensemble import RandomForestRegressor
```

```
# import decision tree
from sklearn.tree import DecisionTreeRegressor
from sklearn.svm import LinearSVR

# import additional multioutput regressors
from sklearn.multioutput import RegressorChain, MultiOutputRegressor

base_regressor = GradientBoostingRegressor(random_state=1)
multi_output_gradient_boosting_regression = MultiOutputRegressor(base_regressor)

base_regressor = LinearSVR(dual=True, max_iter=5000)
multi_output_linear_svr_regression = MultiOutputRegressor(base_regressor)

# 创建 Ridge 基础估计器并封装为 MultiOutputRegressor
base_regressor = Ridge(alpha=1.0)
multi_output_ridge_regression = MultiOutputRegressor(base_regressor)

# 创建 Lasso 基础估计器并封装为 MultiOutputRegressor
base_regressor = Lasso(alpha=1.0, max_iter=3000)
multi_output_lasso_regression = MultiOutputRegressor(base_regressor)

# 创建 ElasticNet 基础估计器并封装为 MultiOutputRegressor
base_regressor = ElasticNet(alpha=1.0, l1_ratio=0.5, max_iter=3000)
multi_output_elasticnet_regression = MultiOutputRegressor(base_regressor)

# 创建 Bagging 基础估计器并封装为 MultiOutputRegressor
base_regressor = BaggingRegressor(random_state=1)
multi_output_bagging_regression = MultiOutputRegressor(base_regressor)

# 创建 AdaBoost 基础估计器并封装为 MultiOutputRegressor
base_regressor = AdaBoostRegressor(random_state=1)
multi_output_adaboost_regression = MultiOutputRegressor(base_regressor)

# 创建不同的回归模型对象
models = {
```

```
  'Linear': LinearRegression(),
  'Ridge': multi_output_ridge_regression,
  'Lasso': multi_output_lasso_regression,
  'ElasticNet': multi_output_elasticnet_regression,
  'K-Neighbors': KNeighborsRegressor(),
  'Decision Tree': DecisionTreeRegressor(),
  'RandForest': RandomForestRegressor(),
  'Bagging': multi_output_bagging_regression,
  'AdaBoost': multi_output_adaboost_regression,
  'GradBoost': multi_output_gradient_boosting_regression,
  'SVM': multi_output_linear_svr_regression
  }
```

3.4 Model Metrics

Test different models with same training data, and fetch the model metrics of `r2_score`, `mean_squared_error` and `mean_absolute_error`.

```
# import required modules
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, KFold
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error
import warnings

class meltingCurveExperiment:
  '''
  define a meltingCurveExperiment
  '''

  def __init__(self, file=None, data=None, test_size=0.05):
    # deal with input
    if data is not None:
      if file is not None:
```

```
        warnings.warn("`data` will be used.", UserWarning)
        data = data
    elif file is not None:
      # 如果只提供了 file，则从文件加载数据
      data = pd.read_csv(file)
    else:
      # 如果没有提供 file 和 data，则引发错误
      raise ValueError("You must provide a `file` or `data` argument.")

    data = data.dropna(axis=1, how='all')
    data = data.dropna()
    self.data = data
    X = data.filter(regex = "^T")
    y = data.filter(regex = "^label_")
    self.X_train, self.X_test, self.y_train, self.y_test = (
      train_test_split(X, y, test_size=test_size)
    )

  def cross_validate_regression_models(self, models, cv=10):
    """
    对给定的多个回归模型使用十折交叉验证进行建模，并计算多个参数。

    参数:
      models: 字典，包含不同名称的回归模型对象，键为模型名称，值为模型对象。
      cv: 整数，指定交叉验证的折数，默认为 10。

    返回值:
      results: 字典，包含每个模型的参数值，键为模型名称，值为参数值的字典，
               其中包含 MSE、R^2、MAE 等。
    """
    results = {}
    kf = KFold(n_splits=cv, shuffle=True, random_state=42)

    for model_name, model in models.items():
      rmse_values = []
```

```
    r2_values = []
    mae_values = []

    for train_idx, test_idx in kf.split(self.X_train):
      X_train_fold = self.X_train.iloc[train_idx]
      X_test_fold = self.X_train.iloc[test_idx]
      y_train_fold = self.y_train.iloc[train_idx]
      y_test_fold = self.y_train.iloc[test_idx]

      # 使用模型对训练集进行拟合
      model.fit(X_train_fold, y_train_fold)

      # 使用训练好的模型进行预测
      y_pred_fold = model.predict(X_test_fold)

      # 计算每个折叠的均方误差、R^2 值、平均绝对误差和解释方差得分
      mse_fold = mean_squared_error(y_test_fold, y_pred_fold)
      r2_fold = r2_score(y_test_fold, y_pred_fold)
      mae_fold = mean_absolute_error(y_test_fold, y_pred_fold)

      rmse_values.append(np.sqrt(mse_fold))
      r2_values.append(r2_fold)
      mae_values.append(mae_fold)

      # 将结果存储在字典中
      results[model_name] = {'rmse': rmse_values,
      'rsq': r2_values,
      'mae': mae_values}
      self.results = results


def results_to_df(self, extra = None):
  data_dict = self.results
  raw = []
  for i in data_dict.keys():
```

```
      for j in data_dict[i].keys():
        raw.append({'model':i, 'metric':j, 'value': data_dict[i][j]})
        df = pd.DataFrame(raw)
        df = df.explode('value').reset_index(drop=True)

    if extra is not None:
      for col_name, col_data in extra.items():
        df[col_name] = col_data

    # return
    return(df)
```

3.5 Evaluation

We test different model in Python.

```
exp = meltingCurveExperiment(file='data-clean/model-selection-data.csv')
exp.cross_validate_regression_models(models=models)
exp.results_to_df().to_csv('data-clean/model-selection-metric.csv', index = False)
```

Metric comparison revealed that ensemble machine learning methods, including Bagging, GradBoost and RandForest, have the best prediction performances Figure 3.1.

```
model_ordered = paste('Linear,Lasso,Ridge,ElasticNet,SVM,GradBoost,AdaBoost',
                      'Bagging,K-Neighbors,Decision Tree,RandForest',
                      sep = ",")  |> str_split_1(",")
mc_ml_cv_metric = read.csv('data-clean/model-selection-metric.csv') |>
  mutate(model = factor(model, levels = model_ordered))
metrics = mc_ml_cv_metric$metric |> unique()

p_mc_ml_cv_metric = lapply(metrics, function(x){
  mc_ml_cv_metric |>
    filter(metric == x) |>
    ggplot(aes(model, value)) +
```

```
    geom_boxplot(outliers = FALSE) +
    labs(x = NULL, y = x) +
    theme(axis.text.x = element_text(angle = 45, hjust = 1.1, vjust = 1.1))
})

plot_grid(plotlist = p_mc_ml_cv_metric, ncol = 3, labels = "auto")

ggsave(filename = "figures/figure3a.jpg")
```

```
Saving 8 x 3.2 in image
```

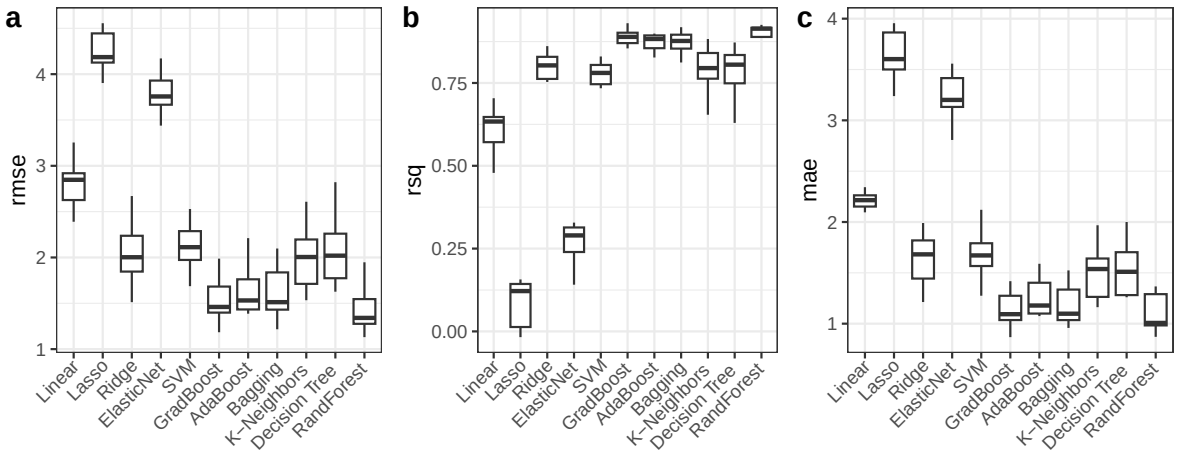


Figure 3.1: Assessment of various machine learning algorisms in modeling and prediction with melting curve data.

4 Dual-species Modelling

We use Random Forest to model dual-species melting curve data.

4.1 Global setting

Here we load packages and define several frequently used variables.

```
# load required packages
library(tidyverse)
library(tidymodels)
library(mcmodel)
library(cowplot)

# default theme
theme_set(theme_bw() +
            theme(legend.key.size = unit(0.4,'cm'),
                  legend.key.height = unit(.4, 'cm')))

# global setting
strain_label = c("label_E", "label_P")
strain_name = c("EC","PP")
strain_color = c("red3", "purple3")

# well position
ec_single_well = paste0(rep(LETTERS[1:16], times = 3), rep(1:3, each = 16))
pp_single_well = paste0(rep(LETTERS[1:16], times = 3), rep(4:6, each = 16))
gradient_matrix_well = paste0(rep(LETTERS[1:16],times = 16), rep(7:22,each = 16))
```

```
strain_single_well = list(EC = ec_single_well, PP = pp_single_well)

# set seed
set.seed(0)
```

4.2 Two-species design

The plate setting used in melting curve analysis is provided in a `csv` file. Original value for each column is the times of two-fold dilution, and we transform them into DNA quantity as related to the original concentration (assumed to be 1). Note: a label value of `16` means not contain this species.

```
# read plate setting
plate_file = xfun::magic_path("modeling-plate-labels.csv")
plate = read.csv(plate_file)
plate2 = plate |>
  drop_na() |>
  mutate(label_E = ifelse(label_E == 0, 1, ifelse(label_E == 16, 0, 1/2^label_E)),
         label_P = ifelse(label_P == 0, 1, ifelse(label_P == 16, 0, 1/2^label_P)))
```

Layout of gradient dilution matrix Figure 4.1.

```
p_concentration = lapply(seq_along(strain_label), function(i){
  plot_384_single_concentration(plate2, strain_label[[i]], well_size = 1.5) +
    scale_color_gradient(high = strain_color[[i]],
                         low = "white",
                         trans = "log2",
                         na.value = "grey90") +
    coord_equal(xlim = c(6.5,22.5)) +
    theme(legend.position = 'none',
          axis.text = element_blank())
})

p_concentration[[1]] = p_concentration[[1]] +
```

```
  labs(x = " ", y = expression(log[2]*Q[1]~(low %->% high)))

p_concentration[[2]] = p_concentration[[2]] +
  labs(x = expression(log[2]*Q[2]~(high %->% low)), y = " ")

plate3 = plate2 |>
  dplyr::filter(well_position %in% gradient_matrix_well)

p_matrix = plot_384_community_structure(plate3) +
  scale_fill_manual(values = c("red3","purple3")) +
  coord_equal(xlim = c(6.5,22.5)) +
  theme(legend.position = 'none',
        axis.text = element_blank()) +
  labs(x = expression(log[2]*Q[2]~(high %->% low)),
       y = expression(log[2]*Q[1]~(low %->% high)))

plot_grid(p_concentration[[1]],
          p_concentration[[2]],
          p_matrix,
          align = 'hv',
          ncol = 3,
          labels = "auto")
```

```
Warning in scale_color_gradient(high = strain_color[[i]], low = "white", : log-2 transformation
log-2 transformation introduced infinite values.
```

```
ggsave(filename = "figures/figure2a.jpg")
```

```
Saving 7 x 2.45 in image
```

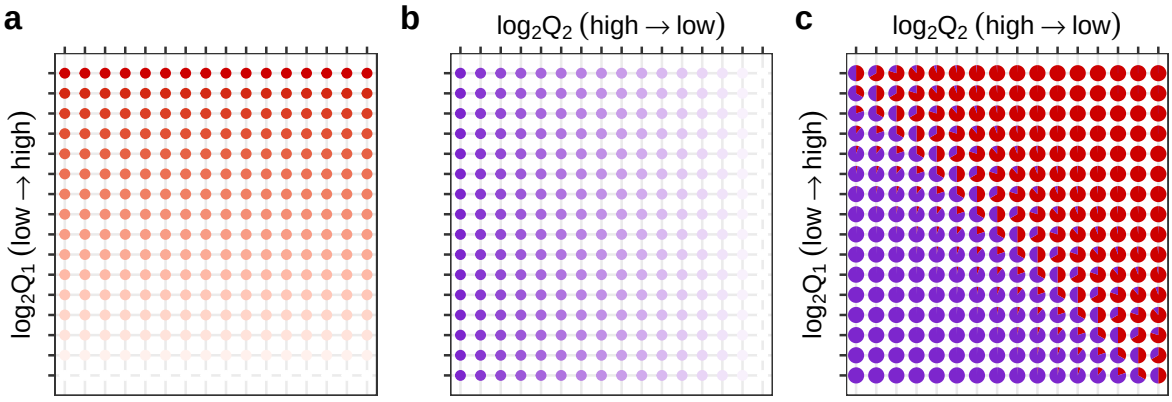


Figure 4.1: Layout of gradient dilution matrix.

Melting curves of gradient dilution matrix Figure 4.2.

```
# read qPCR run results
raw_file = xfun::magic_path("cycle30-experiment1.txt")
quantstudio_raw = read_quantstudio(raw_file)
mc = quantstudio2mc(quantstudio_raw, plate = plate2, primer = "V4")
mc = filterData(mc, from = 80, to = 90) |>
  transformData(step = 0.1)
```

4.3 Gradient Dilution Matrix

```
mc_df = mc2tbl(mc)
mc_df_matrix = mc_df |>
  dplyr::filter(well_position %in% gradient_matrix_well) |>
  dplyr::mutate_at(c('label_E','label_P'),
                   function(x) log2(x) |> as_factor() |> fct_rev())

ggplot(mc_df_matrix, aes(temperature, derivative, group = well_position)) +
  geom_line() +
  scale_x_continuous(position = 'top') +
  facet_grid(label_E ~ label_P, switch = 'y') +
```

```
  labs(x = expression(log[2]*Q[2]~(high %->% low)),
       y = expression(log[2]*Q[1]~(low %->% high))) +
  theme(legend.position = "none",
        panel.spacing = unit(1, "pt"),
        axis.ticks = element_blank(),
        axis.text = element_blank(),)

ggsave("figures/figure2c.jpg")
```

```
Saving 7 x 5.6 in image
```

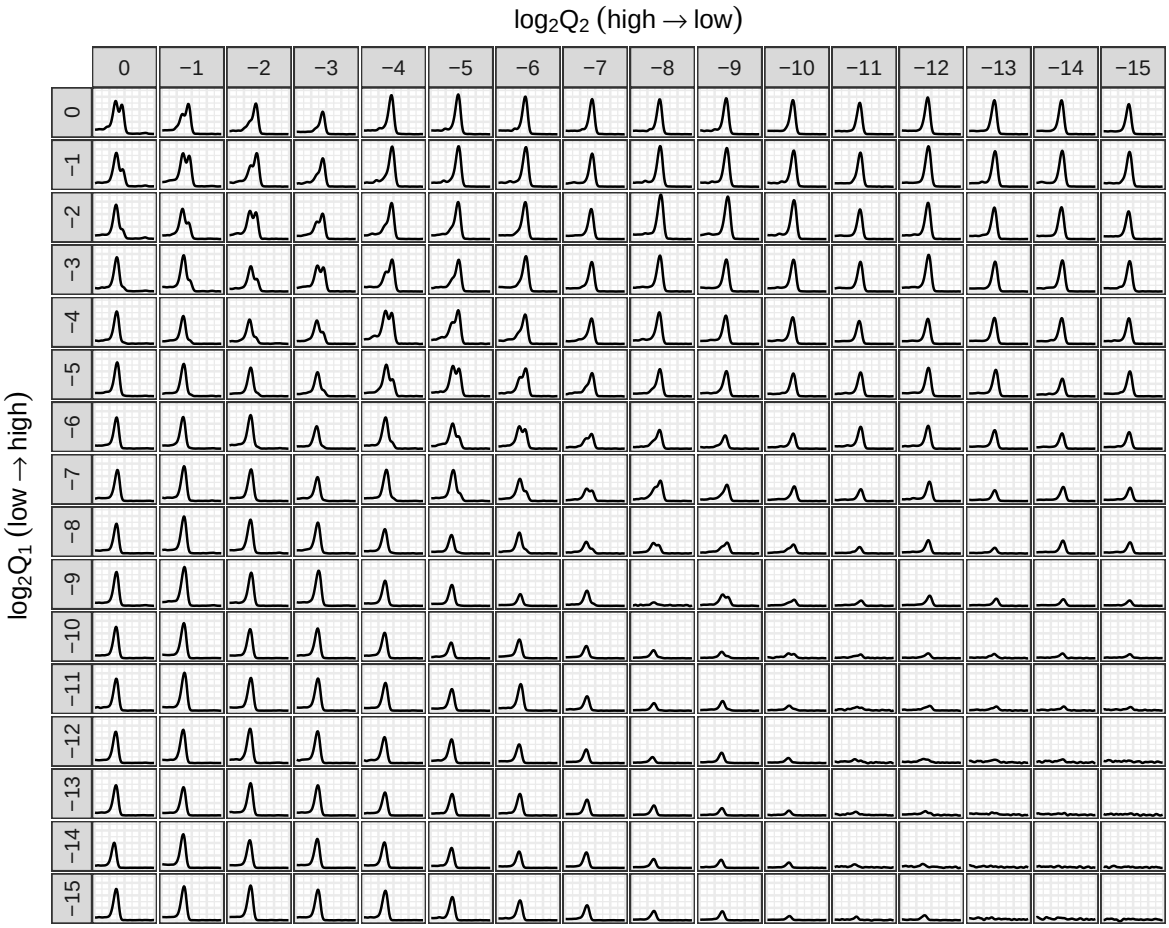


Figure 4.2: Melting curves of gradient dilution matrix.

4.4 Random Forest Modelling

```
# data
mc_ml_data = read.csv("data-clean/20230512.csv") |>
  dplyr::filter(well_position %in% gradient_matrix_well,
                cycle == 30, rep == 2) |>
  mutate(label_E = log2(label_E), label_P = log2(label_P))

data = initial_split(mc_ml_data)
data_train = training(data)
data_test = testing(data)
```

```
lm_spec = linear_reg(engine = "lm")

rf_spec = rand_forest(mode = "regression", trees = 1000) |>
  set_engine("ranger", importance = 'impurity', num.threads = 10)

plot_model_metric = function(predictions, model_metrics, truth_label, color){
  annotation = paste(model_metrics[['.metric']],
                     format(round(model_metrics[['.estimate']], 2), digits = 2),
                     sep = ": ",
                     collapse = "\n")
  ggplot(predictions, aes(.data[[truth_label]], `.pred`, color = I(color))) +
    geom_point(shape = 21) +
    geom_abline(slope = 1, linetype = 'dashed', color = color) +
    annotate("text", x = -Inf, y = Inf, color = I(color), label = annotation,
             hjust = -0.1, vjust = 1.1) +
    coord_equal() +
    xlim(c(-16,0)) + ylim(c(-16, 0)) +
    labs(x = expression(log[2]*Q[true]), y = expression(log[2]*Q[pred]))
}
```

```
p_lm_predict = lapply(seq_along(strain_label), function(i){
  train = data_train |> select(matches(strain_label[[i]]), starts_with("T"))
  test = data_test  |> select(matches(strain_label[[i]]), starts_with("T"))
```

```
  recipe = recipe(as.formula(paste(strain_label[[i]], '.', sep = '~')),
                  data = train)

  fit = workflow() |>
    add_recipe(recipe) |>
    add_model(lm_spec) |>
    fit(train)

  prediction = augment(fit, new_data = test)

  metric = prediction |>
    metrics(truth = strain_label[[i]], estimate = .pred)

  plot_model_metric(prediction, metric, strain_label[[i]], strain_color[[i]]) +
    labs(subtitle = 'Linear')
})
```

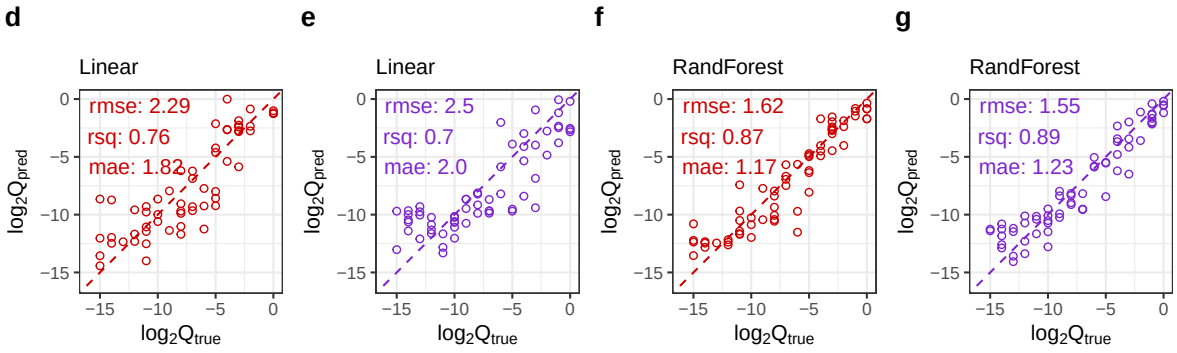
```
Warning in predict.lm(object = object$fit, newdata = new_data, type =
"response", : prediction from rank-deficient fit; consider predict(.,
rankdeficient="NA")

Warning in predict.lm(object = object$fit, newdata = new_data, type =
"response", : prediction from rank-deficient fit; consider predict(.,
rankdeficient="NA")
```

```
p_rf_predict =  lapply(seq_along(strain_label), function(i){
  train = data_train |> select(matches(strain_label[[i]]), starts_with("T"))
  test = data_test  |> select(matches(strain_label[[i]]), starts_with("T"))
  recipe = recipe(as.formula(paste(strain_label[[i]], '.', sep = '~')),
                  data = train)

  fit = workflow()  |>
    add_recipe(recipe)  |>
    add_model(rf_spec) |>
    fit(train)
```

```
  prediction = augment(fit, new_data = test)

  metric = prediction |>
    metrics(truth = strain_label[[i]], estimate = .pred)

  plot_model_metric(prediction, metric, strain_label[[i]], strain_color[[i]]) +
    labs(subtitle = 'RandForest')
})


plot_grid(plotlist = c(p_lm_predict, p_rf_predict), ncol = 4, labels = letters[4:7])
```

```
Warning: Removed 6 rows containing missing values or values outside the scale range
(`geom_point()`).
```

```
Warning: Removed 3 rows containing missing values or values outside the scale range
(`geom_point()`).
```




```
ggsave(filename = "figures/figure3d.jpg")
```

```
Saving 8.5 x 2.97 in image
```

5 Model Optimization

5.1 Global setting

Here we load packages and define several frequently used variables.

```
# load required packages
library(tidyverse)
library(tidymodels)
library(mcmodel)
library(cowplot)

# default theme
theme_set(theme_bw() +
            theme(legend.key.size = unit(0.4,'cm'),
                  legend.key.height = unit(.4, 'cm')))

# global setting
strain_label = c("label_E", "label_P")
strain_name = c("EC","PP")
strain_color = c("red3", "purple3")

# well position
ec_single_well = paste0(rep(LETTERS[1:16], times = 3), rep(1:3, each = 16))
pp_single_well = paste0(rep(LETTERS[1:16], times = 3), rep(4:6, each = 16))
gradient_matrix_well = paste0(rep(LETTERS[1:16],times = 16), rep(7:22,each = 16))
strain_single_well = list(EC = ec_single_well, PP = pp_single_well)
```

```r
# set seed
set.seed(0)
```

5.2 Data, Functions

```r
all_data = read.csv("data-clean/20230512.csv") |>
  filter(well_position %in% gradient_matrix_well) |>
  mutate(label_E = log2(label_E), label_P = log2(label_P))
```

```r
# get metric with specified parameters
rf_metric = function(train_data,
                     test_data = NULL,
                     y = "label_E",
                     X = starts_with("T"),
                     ...){
  model = rand_forest(mode = "regression", trees = 1000) |>
    set_engine("ranger", importance = 'impurity', num.threads = 10)
  train_data = train_data |> select(all_of(y), all_of(X))
  formula = as.formula(paste(y, ".", sep = "~"))
  fit = workflow() |>
    add_recipe(recipe(formula, data = train_data)) |>
    add_model(model) |>
    fit(train_data)
  prediction = augment(fit, new_data = test_data)
  extra = enquos(...)
  metric = prediction |>
    metrics(truth = y, estimate = .pred) |>
    mutate(!!!extra)
  return(metric)
}

# to plot metric
plot_metric = function(data, x, y = ".estimate", metric = 'rsq', color = "black"){
```

```
  data |> filter(.metric == metric) |>
    ggplot(aes(.data[[x]], .data[[y]], color = I(color))) +
    geom_point(shape = 21) +
    geom_smooth(method = "loess") +
    labs(x = x, y = metric)
}
```

5.3 Size of Training Data

Use one experiment data to train, and the other experiment data to test.

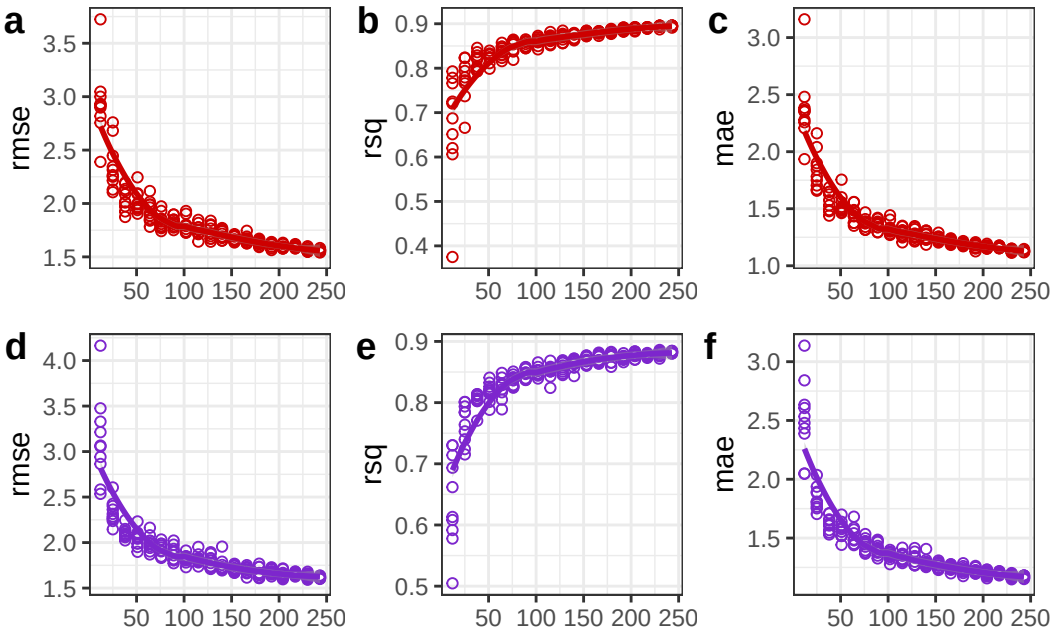
```
rep2 = all_data |>
  dplyr::filter(rep == 2,
                cycle == 30) |>
  dplyr::select(starts_with('label_'), starts_with('T'))

rep3 = all_data |>
  dplyr::filter(rep == 3,
                cycle == 30) |>
  dplyr::select(starts_with('label_'), starts_with('T'))
```

Using different size of training data to fit and evaluate the prediction result of same test data.

```
prop = rep(seq(0.05, 0.95, by = 0.05), each = 10)
rf_rep_metric = lapply(prop, function(p){
  lapply(seq_along(strain_label), function(i){
    data_split = initial_split(rep2, prop = p)
    ylab = strain_label[[i]]
    train = training(data_split) |> select(matches(ylab), starts_with("T"))
    test = rep3 |> select(any_of(ylab), starts_with("T"))
    rf_metric(train, test, ylab, prop = nrow(train), species = ylab)
  }) |> bind_rows()
}) |> bind_rows()
```

```
p_grad = lapply(seq_along(strain_label), function(i){
  lapply(c('rmse','rsq','mae'), function(m){
    df = rf_rep_metric |>
      dplyr::filter(species == strain_label[[i]])
    plot_metric(df, x = "prop", metric = m, color = I(strain_color[[i]])) +
      labs(x = NULL, y = m)
  })
}) |> unlist(recursive = FALSE)

plot_grid(plotlist = p_grad, ncol = 3, align = "hv", labels = "auto")
```

```
`geom_smooth()` using formula = 'y ~ x'
`geom_smooth()` using formula = 'y ~ x'
`geom_smooth()` using formula = 'y ~ x'
`geom_smooth()` using formula = 'y ~ x'
`geom_smooth()` using formula = 'y ~ x'
`geom_smooth()` using formula = 'y ~ x'
```



5.4 Thermo Cycles

```
thermo_cycle_metric = lapply(c(30, 35, 40), function(c){
  mc_ml_data = all_data |>
    filter(cycle == c, rep %in% 2:3)
  lapply(1:10, function(x){
    data = initial_split(mc_ml_data)
    results = lapply(seq_along(strain_label), function(i){
      train = training(data)
      test = testing(data)
      strain = strain_label[[i]]
      rf_metric(train, test, y = strain, cycle = c, species = strain)
    })

    bind_rows(results)
  }) |> bind_rows()
}) |>
  bind_rows() |>
  mutate(cycle = paste(cycle, "x"))
```

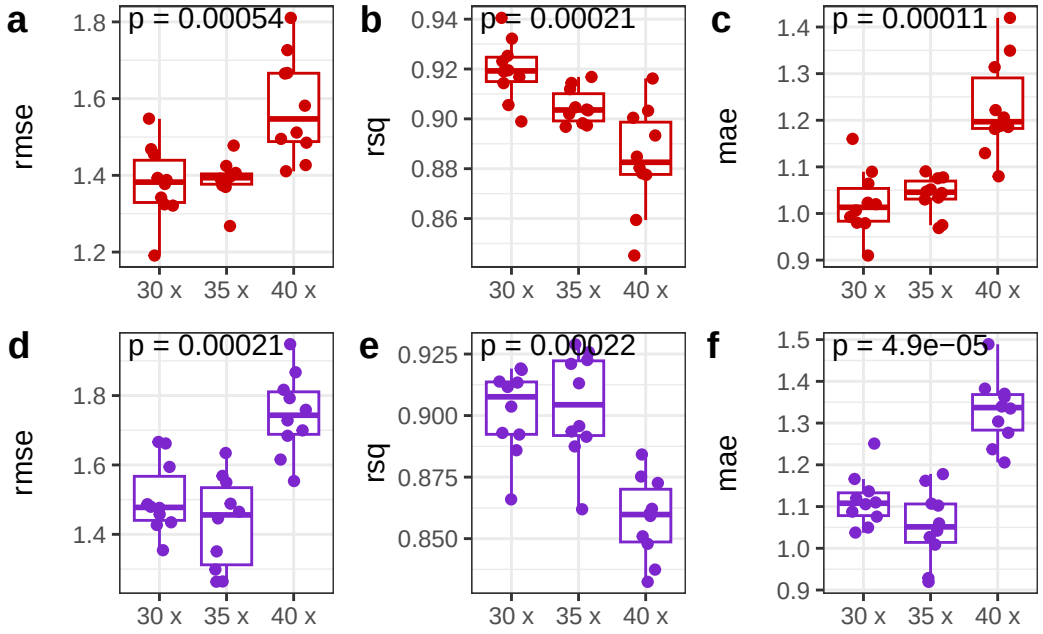
```
library(ggpubr)
```

```
Attaching package: 'ggpubr'
```

```
The following object is masked from 'package:cowplot':

    get_legend
```

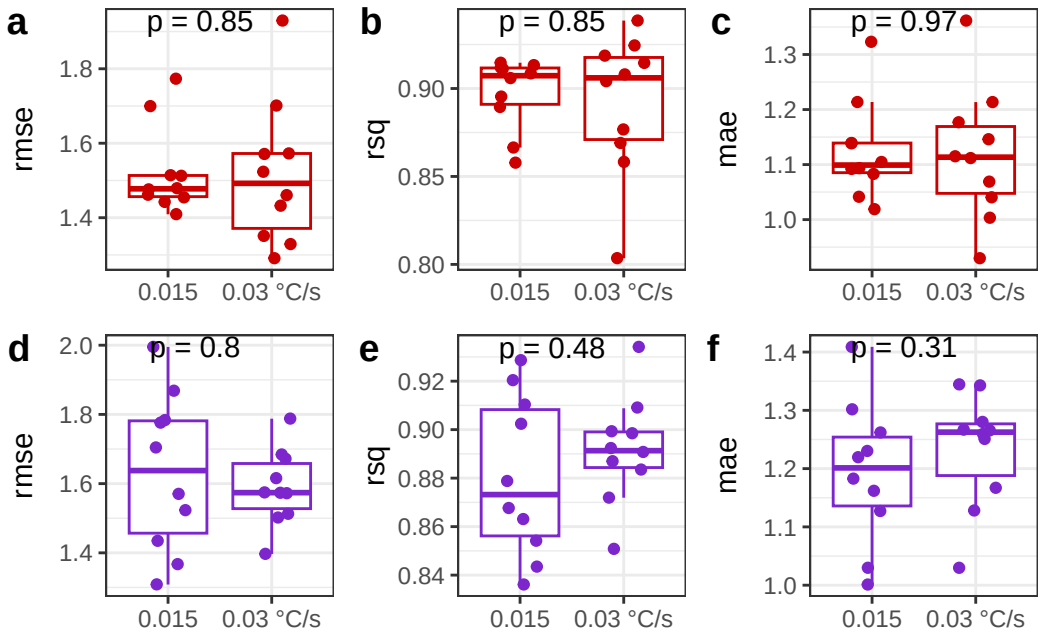
```
# to plot metric
boxplot_metric = function(data, x, y = ".estimate", metric = 'rsq', color = "black"){
  data |> filter(.metric == metric) |>
    ggplot(aes(.data[[x]], .data[[y]], color = I(color))) +
    geom_boxplot(outliers = FALSE) +
```

```
    geom_jitter(width = 0.2) +
    stat_compare_means(label = "p.format", vjust = 0.5) +
    labs(x = NULL, y = metric)
}

p_cycle = lapply(seq_along(strain_label), function(i){
  lapply(c('rmse','rsq','mae'), function(m){
    df = thermo_cycle_metric |>
      dplyr::filter(species == strain_label[[i]])
    boxplot_metric(df, x = "cycle", metric = m, color = I(strain_color[[i]]))
  })
}) |> unlist(recursive = FALSE)

plot_grid(plotlist = p_cycle, align = "hv", ncol = 3, labels = "auto")
```



5.5 Temperature Increment Rate

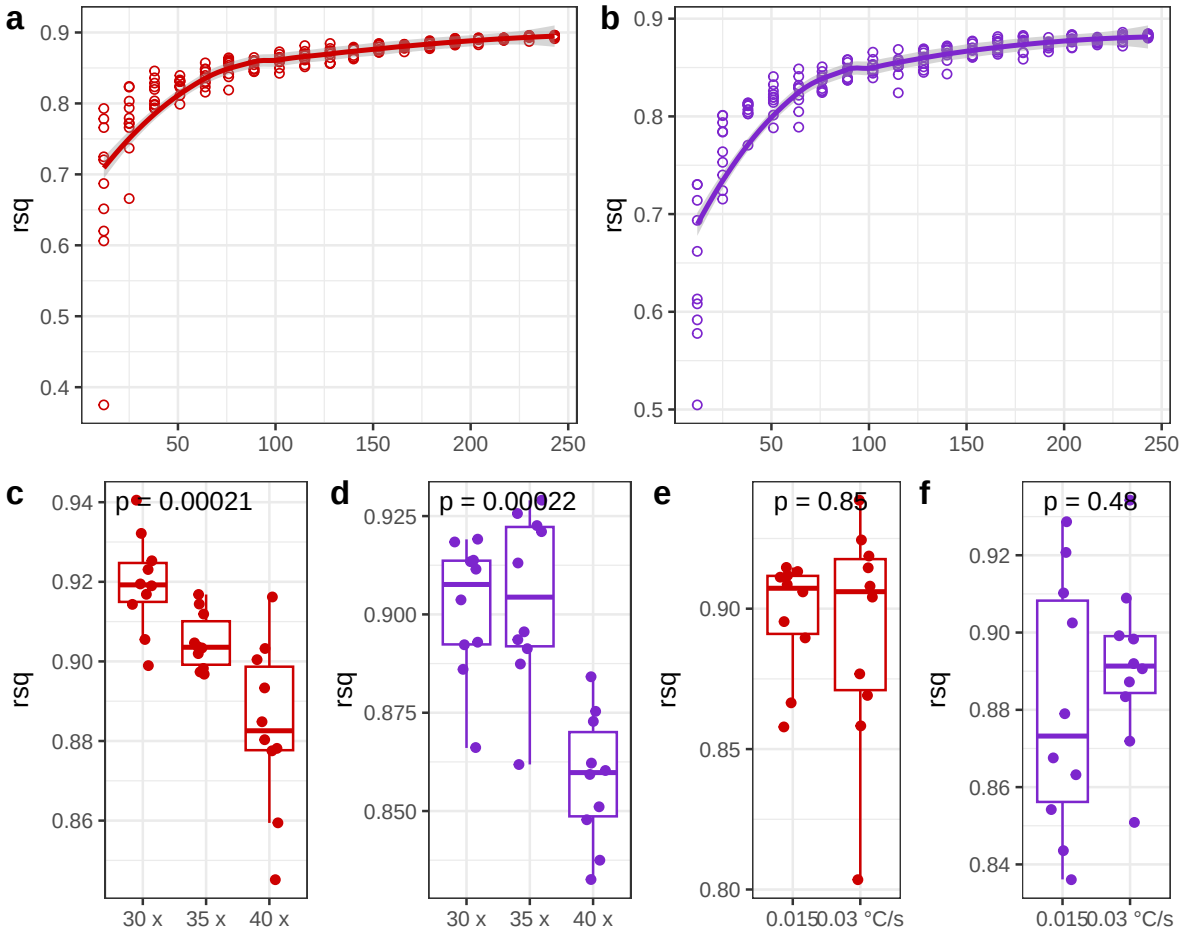
```
rate_map = c(`1` = "0.015", `2` = "0.03 °C/s")
rate_metric = lapply(c(1, 2), function(r){
  mc_ml_data = all_data |>
    filter(cycle == 30, rep == r)
  lapply(1:10, function(x){
    data = initial_split(mc_ml_data)
    results = lapply(seq_along(strain_label), function(i){
      train = training(data)
      test = testing(data)
      strain = strain_label[[i]]
      rf_metric(train, test, y = strain, rate = r, species = strain)
    })

    bind_rows(results)
  }) |> bind_rows()
}) |> bind_rows() |>
  rowwise() |>
  mutate(rate = rate_map[[rate]])


p_rate = lapply(seq_along(strain_label), function(i){
  lapply(c('rmse','rsq','mae'), function(m){
    df = rate_metric |>
      dplyr::filter(species == strain_label[[i]])
    boxplot_metric(df, x = "rate", metric = m, color = I(strain_color[[i]])) +
      labs(x = NULL, y = m)
  })
}) |> unlist(recursive = FALSE)

plot_grid(plotlist = p_rate, align = "hv", ncol = 3, labels = "auto")
```



5.6 Combined Results

```
gglist = c(p_cycle[c(2,5)], p_rate[c(2,5)])
plot_grid(
  plot_grid(plotlist = p_grad[c(2,5)], labels = c("a","b")),
  plot_grid(plotlist = gglist, nrow = 1,
            align = "v",
            rel_widths = c(1.2,1.2,1,1),
            labels = letters[3:6]),
  ncol = 1)
```

```
`geom_smooth()` using formula = 'y ~ x'
`geom_smooth()` using formula = 'y ~ x'
```




```
ggsave("figures/figure4.jpg")
```

```
Saving 7 x 5.6 in image
```

6 Method Evaluation

Comparison of HRM-ML with two golden standards, strain-specific qPCR and 16S rRNA gene sequencing.

6.1 Global setting

Here we load packages and define several frequently used variables.

```
# load required packages
library(tidyverse)
library(tidymodels)
library(mcmodel)
library(cowplot)

# default theme
theme_set(theme_bw() +
            theme(legend.key.size = unit(0.4,'cm'),
                  legend.key.height = unit(.4, 'cm')))

# global setting
strain_label = c("label_E", "label_P")
strain_name = c("EC","PP")
strain_color = c("red3", "purple3")

# well position
ec_single_well = paste0(rep(LETTERS[1:16], times = 3), rep(1:3, each = 16))
pp_single_well = paste0(rep(LETTERS[1:16], times = 3), rep(4:6, each = 16))
```

```
gradient_matrix_well = paste0(rep(LETTERS[1:16],times = 16), rep(7:22,each = 16))
strain_single_well = list(EC = ec_single_well, PP = pp_single_well)

# set seed
set.seed(0)
```

6.2 Functions

```
plot_syncom = function(data){
  ggplot(data, aes(sample, quantity, fill = species)) +
  geom_col() +
  theme(legend.position = "top") +
  scale_x_discrete(labels = function(x) {
    x[seq(2, length(x), 2)] <- ""
    x
  })
}
```

6.3 By 16S rRNA gene sequencing

Use DADA2 to process 16S rRNA gene sequencing data.

```
# raw data
dir = '/Volumes/Data/Projects/MbPL2024051607'
fastq_files = list.files(path = dir, pattern = ".merged.fastq.gz",
                         recursive = TRUE, full.names = TRUE)
sample.names = dirname(fastq_files) |> basename()

# filter and trim sequence
filt_path = file.path("./data-raw/application-ngs")
if (!dir.exists(filt_path)) dir.create(filt_path)
```

```r
library(dada2)

for (i in seq_along(fastq_files)) {
  fastq_filt = file.path(filt_path, basename(fastq_files[i]))
  filterAndTrim(fastq_files[i], fastq_filt,
                maxN = 0, maxEE = 2, truncQ = 2, rm.phix = TRUE,
                compress = TRUE, multithread = TRUE)
}
```

```r
library(dada2)
ngs_path = "data-raw/application-ngs"

# learn error
err = learnErrors(ngs_path, multithread = TRUE, verbose = FALSE)
```

```
104262362 total bases in 412108 reads from 3 samples will be used for learning the error rates.
```

```r
# denoise
dadaFs = derepFastq(ngs_path) |>
  dada(err = err, multithread = TRUE)
```

```
Sample 1 - 138894 reads in 6925 unique sequences.
Sample 2 - 135081 reads in 6290 unique sequences.
Sample 3 - 138133 reads in 6763 unique sequences.
Sample 4 - 146414 reads in 5969 unique sequences.
Sample 5 - 143670 reads in 5975 unique sequences.
Sample 6 - 140401 reads in 6060 unique sequences.
Sample 7 - 143489 reads in 6063 unique sequences.
Sample 8 - 143486 reads in 5979 unique sequences.
Sample 9 - 146415 reads in 5965 unique sequences.
Sample 10 - 142644 reads in 5907 unique sequences.
Sample 11 - 131972 reads in 6048 unique sequences.
Sample 12 - 135238 reads in 6653 unique sequences.
Sample 13 - 142180 reads in 5674 unique sequences.
```

```
Sample 14 - 146281 reads in 6423 unique sequences.
Sample 15 - 136775 reads in 5558 unique sequences.
Sample 16 - 131760 reads in 5484 unique sequences.
Sample 17 - 133423 reads in 5706 unique sequences.
Sample 18 - 146130 reads in 6135 unique sequences.
Sample 19 - 142953 reads in 6105 unique sequences.
Sample 20 - 146291 reads in 6521 unique sequences.
Sample 21 - 146326 reads in 6745 unique sequences.
Sample 22 - 140786 reads in 5726 unique sequences.
Sample 23 - 145401 reads in 6248 unique sequences.
Sample 24 - 142975 reads in 6594 unique sequences.
Sample 25 - 125641 reads in 5031 unique sequences.
Sample 26 - 126698 reads in 4575 unique sequences.
Sample 27 - 135369 reads in 5247 unique sequences.
Sample 28 - 138824 reads in 5184 unique sequences.
Sample 29 - 143340 reads in 4895 unique sequences.
Sample 30 - 141025 reads in 4419 unique sequences.
Sample 31 - 144200 reads in 5132 unique sequences.
Sample 32 - 128555 reads in 4814 unique sequences.
Sample 33 - 147407 reads in 4984 unique sequences.
Sample 34 - 140217 reads in 4539 unique sequences.
Sample 35 - 141868 reads in 4729 unique sequences.
Sample 36 - 147145 reads in 5012 unique sequences.
Sample 37 - 122510 reads in 4288 unique sequences.
Sample 38 - 138030 reads in 4386 unique sequences.
Sample 39 - 135522 reads in 4804 unique sequences.
Sample 40 - 139861 reads in 4406 unique sequences.
```

```
names(dadaFs) = gsub(".merged.fastq.gz", "", names(dadaFs))

# build sequence table
seqtab = makeSequenceTable(dadaFs)

# remove chimera
seqtab.nochim = removeBimeraDenovo(seqtab,
```

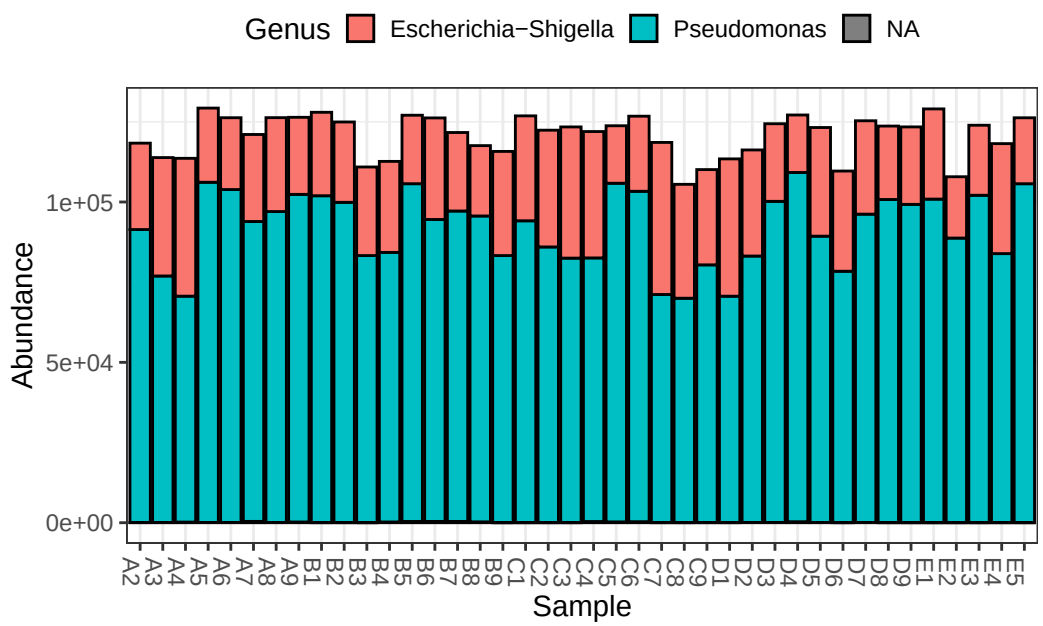
```
                                    method = "consensus",
                                    multithread = TRUE)

# assign taxonomy
silva_train_set = "~/Projects/Silva/silva_nr99_v138.1_train_set.fa.gz"
taxa = assignTaxonomy(seqtab.nochim, silva_train_set, multithread = TRUE)
```

To use Silva database in `assignTaxonomy()`, you should download it at https://benjjneb.github.io/dada2/training.html.

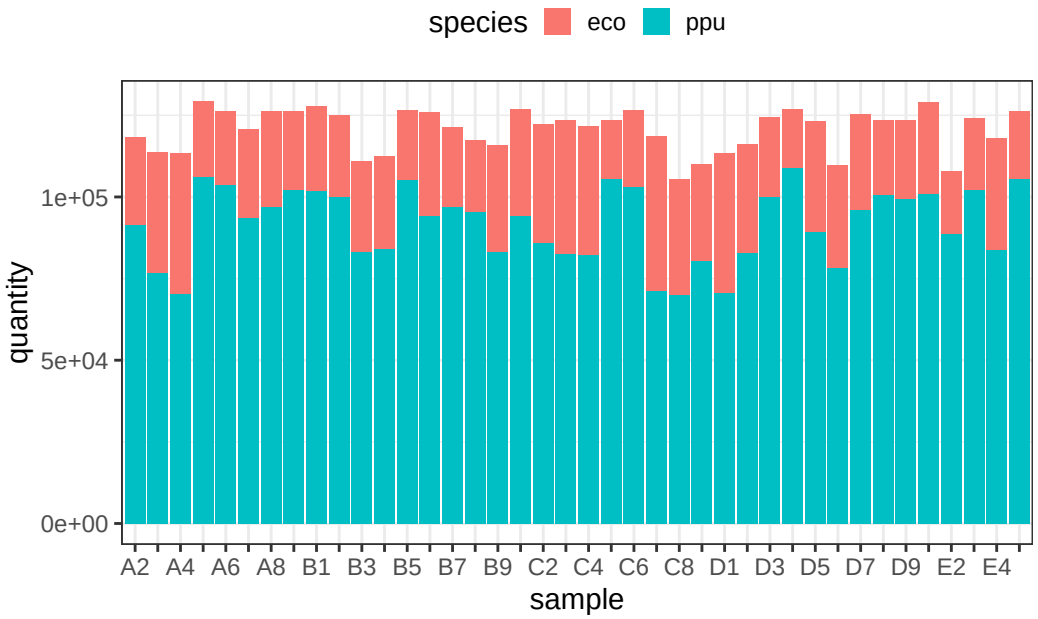
```
# build phyloseq object
library(phyloseq)
```

```
Registered S3 method overwritten by 'vegan':
  method          from
  print.nullmodel parsnip
```

```
ps = phyloseq(otu_table(seqtab.nochim, taxa_are_rows = FALSE), tax_table(taxa))

# merge low abundant noise
taxa_sums = taxa_sums(ps)
threshold = 0.001 * sum(taxa_sums) # threshold is 0.1%
low_abundance = taxa_sums < threshold
ps_merged = merge_taxa(ps, taxa_names(ps)[low_abundance])

# plot community
plot_bar(ps_merged, fill = "Genus") +
  theme(legend.position = "top")
```




```
tax_structure = otu_table(ps_merged) |>
  as.data.frame() |>
  tibble::rownames_to_column("sample") |>
  as_tibble()
colnames(tax_structure)[2:4] = c("ppu","eco","others")
sequencing_result = tax_structure |>
  pivot_longer(cols = -sample, names_to = "species", values_to = "quantity") |>
  filter(species != 'others') |>
  mutate(method = 'NGS')

plot_syncom(sequencing_result)
```



6.4 By strain-specific qPCR

- Build standard curve
- Calculate strain abundance

```
library(mcmodel)

# read qPCR results
result = read_quantstudio(xfun::magic_path("application-qPCR-result.txt")) |>
  get_quantstudio_result() |>
  select(well_position, ct) |>
  mutate(ct = as.numeric(ct))
```

```
Warning: There was 1 warning in `mutate()`.
i In argument: `ct = as.numeric(ct)`.
Caused by warning:
! NAs introduced by coercion
```

```r
# read plate layout
plate = read.csv(xfun::magic_path("application-plate-layout.csv"))

# combine results and plate layout
result = result |> left_join(plate, by = "well_position")
```

```r
std_quantity = tibble(
  well_position = paste0(rep("P", 24), 1:24),
  log2quantity = (rep(2E9, 24)/rep(10^(0:7), each = 3)) |> log2()
)

std_sample = result |>
  filter(target == 'std') |>
  select(well_position, ct)  |>
  left_join(std_quantity, by = "well_position") |>
  select(ct, log2quantity) |>
  na.omit()
```

```r
std_fit = lm(log2quantity ~ ct, std_sample)
summary(std_fit)
```

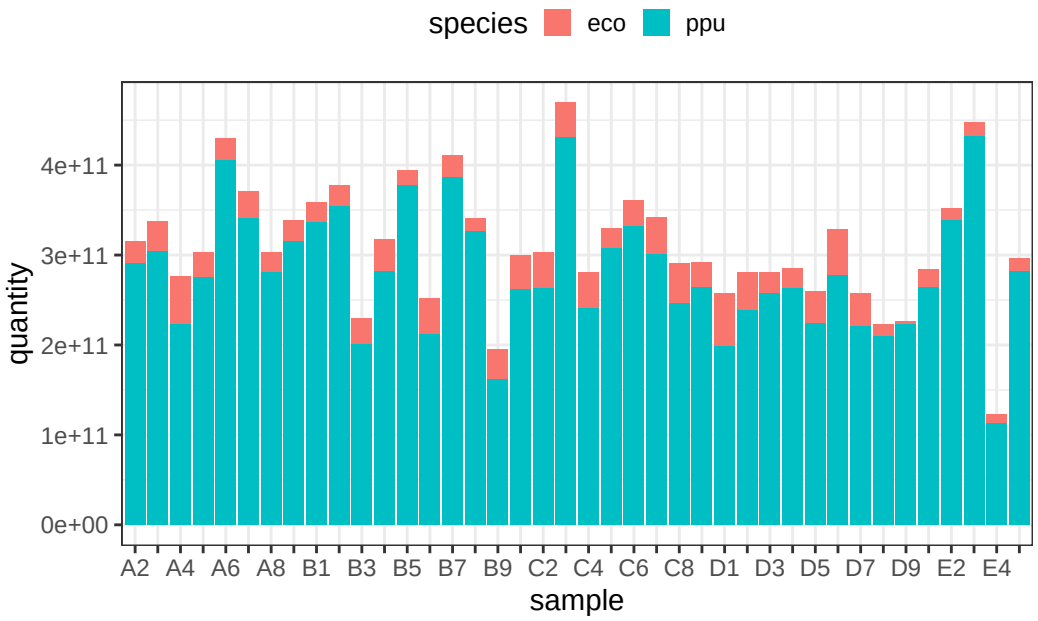
```
Call:
lm(formula = log2quantity ~ ct, data = std_sample)

Residuals:
     Min       1Q   Median       3Q      Max
-0.35894 -0.06547  0.03517  0.09160  0.17017

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 46.71115    0.23581  198.09  < 2e-16 ***
ct          -0.91748    0.01024  -89.58 7.35e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
Residual standard error: 0.1435 on 10 degrees of freedom
Multiple R-squared:  0.9988,    Adjusted R-squared:  0.9986
F-statistic:  8025 on 1 and 10 DF,  p-value: 7.35e-16
```

```r
unk_sample = result |>
  filter(target %in% c('EC','PP'))

unk_sample_predition = broom::augment(std_fit, newdata = unk_sample)
```

```r
qPCR_result = unk_sample_predition |>
  summarise(predict = mean(.fitted, na.rm = TRUE), .by = c(sample, target)) |>
  mutate(species = if_else(target == 'EC', 'eco', 'ppu'),
         quantity = 2^predict,
         method = 'qPCR') |>
  select(sample, species, quantity, method)
```

```r
plot_syncom(qPCR_result)
```



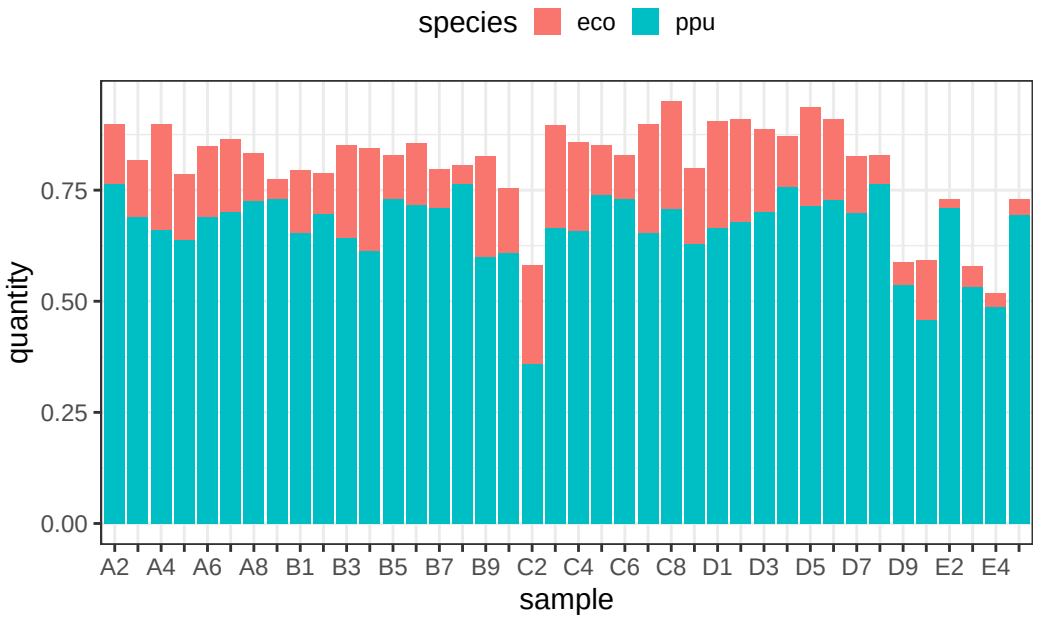
6.5 By HRM-ML method

```
filename = xfun::magic_path("application-qPCR-result.txt")
plate = read.csv(xfun::magic_path("application-plate-layout.csv"))

all = read_quantstudio(filename)
sample = plate |>
  mutate(cycle = 30) |>
  dplyr::filter(target == '16S')
mc240617 = quantstudio2mc(all, plate = sample)  |>
  filterData(from = 75, to = 90, well_position = sample$well_position) |>
  transformData(step = 0.1)



data240617 = mc_tbl2wider(mc240617)

write_csv(data240617, "data-clean/20240617.csv")
```

```
# data
train_data = read.csv("data-clean/20230512.csv") |>
  dplyr::filter(well_position %in% gradient_matrix_well,
                cycle == 30, rep == 1) |>
  dplyr::select(starts_with('label_'), starts_with('T')) |>
  mutate(label_E = log2(label_E), label_P = log2(label_P))

# test data
test_data = read.csv("data-clean/20240617.csv")

library(parsnip)
library(recipes)
library(workflows)
rf_spec = rand_forest(mode = "regression", trees = 1000) |>
  set_engine("ranger", importance = 'impurity', num.threads = 10)

predictions = lapply(seq_along(strain_label), function(i){
```

```
  label = strain_label[[i]]
  train = train_data |> select(matches(label), starts_with("T"))
  recipe = recipe(formula = as.formula(paste(label, '.', sep = '~')),
                  data = train)

  rf_wflow = workflow()  |>
    add_recipe(recipe)  |>
    add_model(rf_spec)

  rf_fit = rf_wflow |>
    fit(train)

  prediction = augment(rf_fit, new_data = test_data) |>
    select(-starts_with("T")) |>
    mutate(label = strain_label[[i]])

  return(prediction)
})

HRM_result = predictions |>
  bind_rows() |>
  summarise(.pred = mean(.pred), .by = c(sample, label)) |>
  mutate(species = if_else(label == 'label_E', 'eco', 'ppu'),
         quantity = 2^.pred,
         method = 'HRM-ML') |>
  select(sample, species, quantity, method)
```

```
HRM_result |> plot_syncom()
```

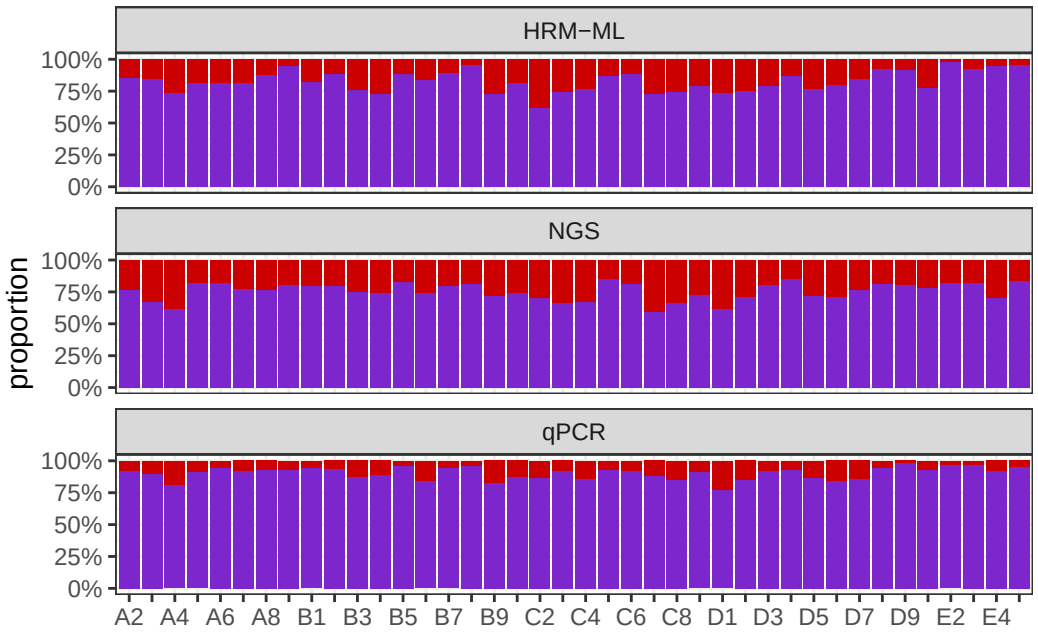


6.6 Comparison

```
three_result = list(qPCR_result, sequencing_result, HRM_result) |>
  bind_rows() |>
  group_by(method, sample) |>
  mutate(prop = quantity/sum(quantity)) |>
  ungroup()
```

```
p_prop = ggplot(three_result, aes(sample, prop, fill = species)) +
  geom_col() +
  facet_wrap(~method, ncol = 1) +
  scale_y_continuous(labels = scales::percent) +
  scale_x_discrete(labels = function(x) {
    x[seq(2, length(x), 2)] <- ""
    x
  }) +
  scale_fill_manual(values = strain_color) +
```

```
  labs(x = NULL, y = "proportion") +
  theme(legend.position = "none")

p_prop
```




```
library(tidyr)
ec_prop = three_result |>
  filter(species == 'eco') |>
  pivot_wider(id_cols = sample, names_from = method, values_from = prop) |>
  na.omit()

cor(ec_prop |> select(-sample), method = "spearman")
```

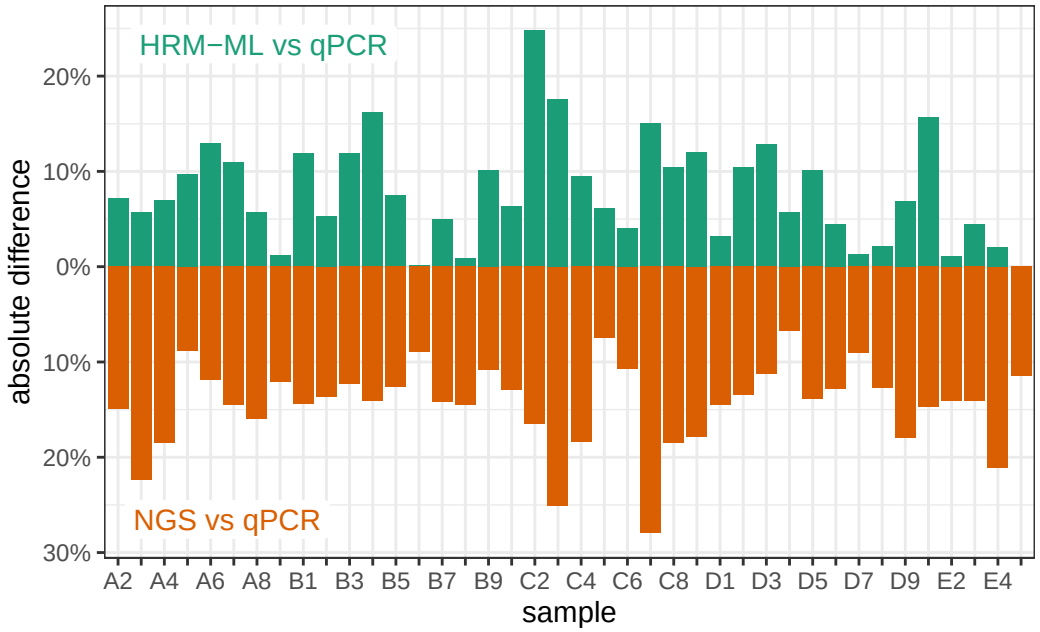
```
            qPCR       NGS    HRM-ML
qPCR   1.0000000 0.7851782 0.7803002
NGS    0.7851782 1.0000000 0.7166979
HRM-ML 0.7803002 0.7166979 1.0000000
```

```r
ec_prop_difference = ec_prop |>
  rowwise() |>
  mutate(qPCR_vs_mc = mean(abs(qPCR - `HRM-ML`)),
         qPCR_vs_ngs = -mean(abs(qPCR - NGS))) |>
  pivot_longer(cols = starts_with('qPCR_vs'),
               names_to = 'comparison',
               values_to = 'abs_distance')

# mean absolute differences
ec_prop_difference |> summarise(average = mean(abs(abs_distance)), .by = comparison)
```

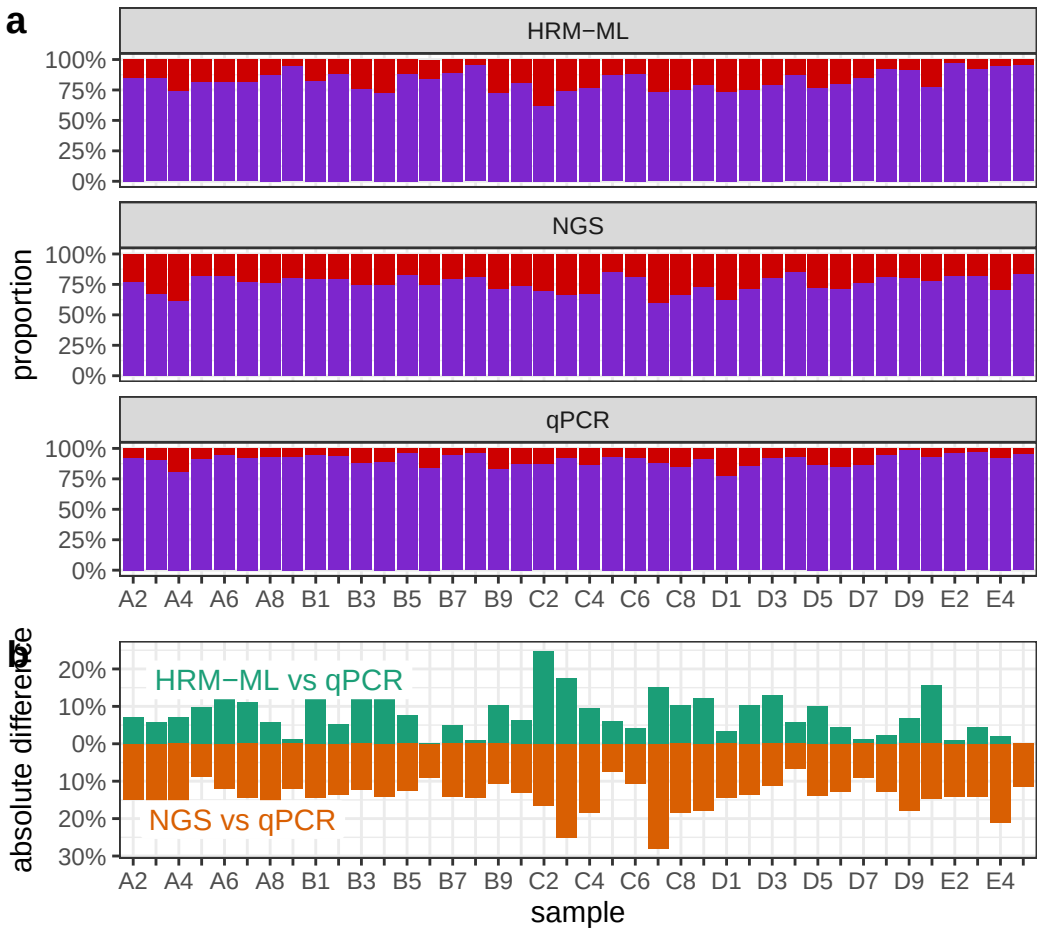
```
# A tibble: 2 x 2
  comparison  average
  <chr>         <dbl>
1 qPCR_vs_mc   0.0790
2 qPCR_vs_ngs  0.144
```

```r
p_diff = ec_prop_difference |>
  ggplot(ggplot2::aes(sample, abs_distance, fill = comparison)) +
  geom_col() +
  scale_x_discrete(labels = function(x) {
    x[seq(2, length(x), 2)] <- ""
    x
  }) +
  annotate("label", x = -Inf, y = Inf,
           label = "HRM-ML vs qPCR",
           hjust = -0.1, vjust = 1.5, label.size = NA,
           color = "#1B9E77") +
  annotate("label", x = -Inf, y = -Inf,
           label = "NGS vs qPCR",
           hjust = -0.1, vjust = -0.5, label.size = NA,
           color = "#D95F02") +
  labs(y = "absolute difference") +
  scale_y_continuous(labels = function(x) scales::percent(abs(x))) +
  scale_fill_manual(values = c("#1B9E77", "#D95F02")) +
```

```
  theme(legend.position = "none")

p_diff
```




```
plot_grid(p_prop, p_diff, align = "v", ncol = 1,
          rel_heights = c(2, 1),
          labels = "auto")
```




```
ggsave("figures/figure5.jpg")
```

```
Saving 5.5 x 4.95 in image
```

References

Knuth, Donald E. 1984. “Literate Programming.” *Comput. J.* 27 (2): 97–111. https://doi.org/10.1093/comjnl/27.2.97.