THE UTILITY OF MIXED-EFFECT MODELS IN THE EVALUATION OF COMPLEX GENOMIC TRAITS IN-VITRO

Supplementary Data: Complete R Code

Nathan Alade, M.S.* Abhinav Nath, Ph.D.† Nina Isoherannen, Ph.D.*
Kenneth Thummel, Ph.D.*

Last compiled on: January 11, 2023

*University of Washington, Department of Pharmaceutics
†University of Washington, Department of Medicinal Chemistry

Contents

1 Correspondence

Corresponding Author: Nathan Alade, M.S.

Address:

- University of Washington School of Pharmacy
- Department of Pharmaceutics
- H-272 Health Sciences Building, Box 357610
- Seattle, Washington 98195

Email: alade@uw.edu

Phone: 206-543-9434

Fax: 206-543-3204

1.0.1 Data Availability:

R Scripts and data necessary to reproduce our analysis are available and can be cloned using the following public repository. (http://nathanalade.github.io/In-Vitro-NLME/)

2 Introduction

2.1 Abstract

The use of human liver microsomes as a model system to evaluate the impact of complex genomic traits (i.e., linkage-disequilibrium patterns, coding, and non-coding variation, etc.) on drug metabolism remains challenging. To overcome these challenges, we propose the use of non-linear mixed effect models (NLME) as an unconventional approach to evaluate the impact of complex genomic traits on the metabolism of xenobiotics in vitro. In this in-silico study, we present a practical use case for such an approach using previously published in-vitro CYP2D6 data and explore the impact of sparse sampling, and experimental error on known kinetic parameter of CYP2D6 mediated formation of 4-hydroxy-atomoxetine in human liver microsomes.

2.2 Objective

This supplement provides the `R` code used to conduct the *in-silico* portion of the study, as well as useful functions for evaluating/visualizing results.

3 R Libraries and Packages

```
# R Packages -----
library(nlme)       # Non-Linear Mixed Effect Modeling
library(tidyverse)  # Data Manipulation and Visualization
library(ggeasy)     # Wrapper for Plot Manipulation
library(ggpubr)     # Additional Tools for Plot Configuration
library(scales)     # Additional Tools for Adjusting Plot Scales
library(cowplot)    # Data Visualization Themes and Tools
library(kableExtra) # Additional Tools for Creating Custom Tables

# Package References -------
knitr::write_bib(c(.packages()), "References/packages.bib")
```

Citation information for R packages [R Core Team [2022]; Wilke [2020]; Wickham et al. [2022b]; Carroll et al. [2021]; Wickham et al. [2022a]; Kassambara [2020]; Zhu [2021]; Pinheiro et al. [2022]; Wickham et al. [2022c]; Wickham and Seidel [2022]; R-tibble; Wickham and Girlich [2022]; Wickham [2021]; Wickham [2016]; Pinheiro and Bates [2000]; Wickham et al. [2019]] used for data can be found in the *References* section.

4 R Session Information

```
# Session Information ------
sessionInfo()
```

```
## R version 4.2.0 (2022-04-22 ucrt)
## Platform: x86_64-w64-mingw32/x64 (64-bit)
## Running under: Windows 10 x64 (build 22000)
##
## Matrix products: default
##
## locale:
## [1] LC_COLLATE=English_United States.utf8
## [2] LC_CTYPE=English_United States.utf8
## [3] LC_MONETARY=English_United States.utf8
## [4] LC_NUMERIC=C
## [5] LC_TIME=English_United States.utf8
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
##  [1] kableExtra_1.3.4 cowplot_1.1.1    scales_1.2.0     ggpubr_0.4.0
##  [5] ggeasy_0.1.3     forcats_0.5.1    stringr_1.4.0    dplyr_1.0.9
##  [9] purrr_0.3.4      readr_2.1.2      tidyr_1.2.0      tibble_3.1.8
## [13] ggplot2_3.3.6    tidyverse_1.3.1  nlme_3.1-157
##
## loaded via a namespace (and not attached):
##  [1] svglite_2.1.0     lubridate_1.8.0   lattice_0.20-45   assertthat_0.2.1
##  [5] digest_0.6.29     utf8_1.2.2        R6_2.5.1          cellranger_1.1.0
##  [9] backports_1.4.1   reprex_2.0.1      evaluate_0.15     httr_1.4.3
## [13] pillar_1.8.0      rlang_1.0.4       readxl_1.4.0      rstudioapi_0.13
## [17] car_3.1-0         rmarkdown_2.14    webshot_0.5.4     munsell_0.5.0
## [21] broom_0.8.0       compiler_4.2.0    modelr_0.1.8      xfun_0.31
## [25] systemfonts_1.0.4 pkgconfig_2.0.3   htmltools_0.5.2   tidyselect_1.1.2
## [29] bookdown_0.31     viridisLite_0.4.0 fansi_1.0.3       crayon_1.5.1
## [33] tzdb_0.3.0        dbplyr_2.2.0      withr_2.5.0       grid_4.2.0
## [37] jsonlite_1.8.0    gtable_0.3.0      lifecycle_1.0.1   DBI_1.1.2
## [41] magrittr_2.0.3    cli_3.3.0         stringi_1.7.6     carData_3.0-5
## [45] ggsignif_0.6.3    fs_1.5.2          xml2_1.3.3        ellipsis_0.3.2
## [49] generics_0.1.2    vctrs_0.4.1       tools_4.2.0       glue_1.6.2
## [53] hms_1.1.1         abind_1.4-5       fastmap_1.1.0     yaml_2.3.5
## [57] colorspace_2.0-3  rstatix_0.7.0     rvest_1.0.2       knitr_1.39
## [61] haven_2.5.0
```

5 Importing Data

CYP2D6 mean estimates and frequencies (Dinh et. al., 2016) were imported and summarized by genotype (***referred to as diplotype in R script***). Additionally, an Eta table and parameter estimate table were also generated to be reference later on in the data analysis.

```
# Atomoxetine Dataset (Dinh et al., 2016)
atx.data <- readxl::read_xlsx("Input Data/Atomoxetine Data - 2016.xlsx")

# Summary of Atomoxetine Data by genotype ----
diplotype.data <- atx.data %>%
  group_by(Diplotype) %>%

  # Summarize mean estimates of the data
  summarise(Vmax = mean(Vmax),
            Km = mean(Km),
            Score = unique(`Activity Score`),
            n = n()) %>%
  ungroup() %>%
  mutate(Freq = n/sum(n),
         Vmax.Eta = ifelse(Diplotype != "1/1",
                           round(Vmax-Vmax[Diplotype == "1/1"],
                                 digits = 2),
                           round(Vmax, digits = 2)),
         Km.Eta = ifelse(Diplotype != "1/1",
                         round(Km-Km[Diplotype == "1/1"],
                               digits = 2),
                         round(Km, digits = 2))) %>%
  select(-n)

# Eta Table (Deviations from the reference group) -----
actual.eta <- diplotype.data %>%
  select(Diplotype, Vmax.Eta, Km.Eta) %>%
  rename("Vmax" = "Vmax.Eta",
         "Km" = "Km.Eta") %>%
  pivot_longer(Vmax:Km, names_to = "Parameter",
               values_to = "Actual") %>%
  arrange(desc(Parameter))

# mean estimates of Vmax and Km for each genotype -----
actual.est <- diplotype.data %>%
  select(Diplotype, Vmax, Km) %>%
  pivot_longer(Vmax:Km, names_to = "Parameter",
               values_to = "Actual") %>%
  mutate(Actual = round(Actual, digits = 2)) %>%
  arrange(desc(Parameter))
```

6 Virtual Population

6.1 Population Simulation Function

A custom simulation function `population.sim()` was scripted to generate a virtual population with a pre-defined distribution of *CYP2D6* genotypes, and to simulate Michaelis-Menten kinetics for each individual. The inputs for the function the `population.sim()` include the following:

- **n** - number of subjects per genotype group (default = 10).
- **CV%_D** - Average standard deviation for each genotype group as a percentage of the true estimate for Vmax and Km (between subject variability for each genotype group) (default = 25).
- **CV%_V** - Average coefficient of variation across all points for simulated Michaelis-Menten kinetics for each individual (residual error).
- **Start** - Starting concentration from Michaelis-Menten simulation (default = 0).
- **End** - Ending concentration for Michaelis menten Kinetics (default = 100 uM).
- **By** - Simulated concentration increment (default = 0.5).
- **pop_freq** - TRUE or FALSE argument indicating whether to use true population frequency (default is FALSE).
- **seed** - Random number generator seed (for reproducibility, default = 23457).
- **data** - Dataframe to be used as reference for population averages. Must include genotype, Vmax, Km, and Frequency columns (default = `diplotype.data`).

Within-group between subject variability was set at 25% CV for each genotype (one standard deviation = 0.25 x mean of true value). Below is a simulated population containing 9000 individuals (1000 per diplotype groups) with mean and standard deviations for each group parameter.

```
population.sim <- function(

  # Pre-define number of subjects per diplotype group
  n = 10,
  `CV%_D` = 25,
  `CV%_V` = 0,

  # Pre-define Michaelis Menten Kinetic Settings
  Start = 0,
  End = 100,
  By = 0.5,

  # Turn off or on use of true population frequency
  ## If set to true "n" will equal total population number
  pop_freq = FALSE,

  # Pre-define reference data, and set seed
  data = diplotype.data,
  seed = 23456){

  # ======== Create Data Containers ========

  datasets <- list()
  pop.data <- tibble()
```

```r
# ======== Create Datasets by Diplotype ========

for (i in seq_along(data[[1]])){

  # Setting Population Specific Frequency
  n_pop <- if_else(pop_freq == FALSE, n,
           ifelse(pop_freq == TRUE,
                  round(n*data[[i, c("Freq")]]), 0))

  # Specify Diplotype for current loop
  Diplotype = rep(data[[i,1]], n_pop)

  # Random assignment of Vmax values N(0,sigma)
  set.seed(seed)
  Vmax_eta <- rnorm(n_pop, sd = `CV%_D`)
  Vmax = data[[i,2]] + (Vmax_eta/100)*data[[i,2]]

  # Random assignment of Km values N(0,sigma)
  set.seed(seed)
  Km_eta <- rnorm(n_pop, sd = `CV%_D`)
  Km = data[[i,3]] + (Km_eta/100)*data[[i,3]]

  # Store each diplotype group in a list container
  temp <- tibble(Diplotype, Vmax, Km)
  datasets[[i]] <- temp

}

# ======== Combine Diplotype Datasets ========

for (i in seq_along(data[[1]])){

  pop.data <- rbind(pop.data, datasets[[i]])

}

# ======== Simulate Michaelis Menten ========

set.seed(seed)

pop.data <- pop.data %>%
  mutate(ID = factor(seq_along(Diplotype),
                     levels = seq_along(Diplotype))) %>%
  relocate(ID) %>%
  expand_grid(S = seq(Start, End, By)) %>%
  mutate(V = round((Vmax*S)/(Km+S), digits = 2))%>%
  group_by(ID) %>%

  # Additon of Residual Error ------------
  mutate(resid_pct = round(rnorm(n = length(S),sd = `CV%_V`),
                           digits = 3),
         V_adj = V+(V*resid_pct/100)) %>%
  ungroup()
```

```
  return(pop.data)

}
```

6.2 Distribution of Virtual Population Parameters

```
dist.plot <- ggplot(

  data = population.sim(n=1000,`CV%_D` = 25, Start = 0.5,
                        By = 5, pop_freq = F) %>%
    group_by(ID) %>%
    filter(S == min(S)),

    aes(x = Vmax))+
  geom_histogram(aes(fill = Diplotype), color = "black", bins = 100)+
    ggeasy::easy_remove_legend_title()+
  theme_bw(base_size = 14)+
  xlab(bquote(V[Max]))+
  ylab("Population (n)")


dist.grb <- dist.plot +
  ggeasy::easy_remove_legend() +
  scale_x_log10()+
  xlab("Log Scale")+
  ylab("Count (n)")


dist.plot2 <- ggplot(

  data = population.sim(n=1000,`CV%_D` = 25, Start = 0.5,
                        By = 5, pop_freq = F) %>%
    group_by(ID) %>%
    filter(S == min(S)),

    aes(x = Km))+
  geom_histogram(aes(fill = Diplotype), color = "black", bins = 100)+
    ggeasy::easy_remove_legend_title()+
  theme_bw(base_size = 14)+
  xlab(bquote(K[M](uM)))+
  ylab("Population (n)")+
  scale_x_log10()


dist.grb2 <- dist.plot +
  ggeasy::easy_remove_legend() +
  scale_x_log10()+
  xlab("Log Scale")+
  ylab("Count (n)")
```

```
## Vmax and Km Distribution Plots
ggpubr::ggarrange(dist.plot +
  annotation_custom(ggplotGrob(dist.grb),
                    xmin = 1000, xmax = 2500,
                    ymin = 250, ymax = 1250),
  dist.plot2, ncol = 1, common.legend = T, legend = "right")
```

6.3 Virtual Population Vmax and Km Distribution Table

```
population.sim(n = 1000, End = 0,`CV%_D` = 25) %>%
  group_by(Diplotype) %>%
  summarise(Vmax_mean = round(mean(Vmax),2),
            Vmax_sd = round(sd(Vmax),2),
            Km_mean = round(mean(Km),2),
            Km_sd = round(sd(Km),2),
            `n` = length(Vmax),
            Vmax = paste0(Vmax_mean," ± ",Vmax_sd),
            Km = paste0(Km_mean," ± ",Km_sd)) %>%
  ungroup() %>%
  left_join(diplotype.data %>%
              mutate(across(where(is.numeric), round, 2)) %>%
              select(Diplotype, Vmax, Km) %>%
              rename("Actual (Vmax)" = "Vmax", "Actual (Km)" = "Km"),
            by = "Diplotype") %>%
  select(Diplotype,`Actual (Vmax)`, Vmax, `Actual (Km)`,Km, n) %>%
  rename("Simulated." = "Vmax",
         "Simulated" = "Km",
         "Actual." = "Actual (Vmax)",
         "Actual" = "Actual (Km)") %>%
  kbl(table.attr = "style='width:60%;'",
      caption = "Simulated Population Parameter Estimates") %>%
  kable_classic("striped", full_width = T) %>%
  add_header_above(c(" ", "Vmax Estimate" = 2, "Km Estimate" = 2, " "))
```

6.4 Incorporation of Experimental Error

Residual error was added to simulated data using a constant coefficient of variation structure, where the observed metabolic rate (V_{obs})for the i^{th}individual at the j^{th} substrate incubation concentration. The residual errors ($_i$) were normally distributed with a mean of 0 and a standard deviation ( ) set at the following values: 0, 0.05, 0.10, or 0.2; corresponding to a coefficient of variation (CV%) of 0%, 5%, 10%, and 20%.

```
# Error Design Datatable -------
error_design.data <- rbind(population.sim(n=3, Start = 0, By = 0.1, `CV%_V` = 0) %>%
         filter(Diplotype == "1/1") %>% mutate(Condition = "0% CV"),
       population.sim(n=3, Start = 0, By = 0.1, `CV%_V` = 5) %>%
         filter(Diplotype == "1/1") %>% mutate(Condition = "5% CV"),
       population.sim(n=3, Start = 0, By = 0.1, `CV%_V` = 10) %>%
         filter(Diplotype == "1/1") %>% mutate(Condition = "10% CV"),
       population.sim(n=3, Start = 0, By = 0.1, `CV%_V` = 20) %>%
         filter(Diplotype == "1/1") %>% mutate(Condition = "20% CV")) %>%
```

```
  filter(S %in% c(0.1, 0.5, 1, 2, 5, 10, 15, 25, 40, 55, 70, 85, 100)) %>%
  mutate(ID2 = paste("Subject",ID),
         `Study Design` = "Rich Design (9 pts)")

# Assigning Condition as Ordered Factor -----------
error_design.data$Condition <- factor(error_design.data$Condition,
                                      levels = c("0% CV","5% CV","10% CV","20% CV"))

# Simulation Grid ---------
error.nls <- nls(formula = V ~ (Vmax*S)/(Km + S), data = error_design.data,
start = list(Vmax = 350, Km = 2))
error.grid <- tibble(S = seq(0,100,0.1),
                V = (coef(error.nls)[[1]]*S)/(coef(error.nls)[[2]]+S))

# Experimental Error Scenarios Plot -----------
ggplot(error_design.data %>% filter(S %in% c(0.1, 0.5, 1, 2.5, 5, 10, 25, 50, 100)),
       aes(x = S, y = V))+
    geom_line(data = error.grid, size = 1, alpha = 0.75)+
    geom_point(aes(y = V_adj, group = ID2, color = ID2, shape = ID2), size = 2.5)+
    facet_wrap(~Condition, ncol = 2, scales = "free")+
    theme_bw(base_size = 14)+
    xlab("[Substrate](uM)")+
    ylab("Reaction Rate (V)\n")+
    scale_color_viridis_d(option = "A", begin = 0.2, end = 0.7)+
    ggeasy::easy_remove_legend_title()+
    theme(legend.position = c(0.9,0.11), legend.box = "horizontal",
          legend.background = element_rect(color = "grey"))+
  labs(title = "Experimental Error Scenarios")
```

7 *In-Silico* Experiments

7.1 Strategic Sampling Approach

A custom function, `update_data()`, was created to generate experimental data *in-silico* according to the specified experimental design and conditions. Two in-silico experimental designs (rich design, and sparse design) for two experimental conditions (UM/EM and IM/PM conditions) were incorporated into the function. Rich design experiments were conducted using 9 overlapping atomoxetine (ATX) incubation concentrations per subject, while sparse design experiments used 3-4 staggered incubation concentrations per subject , with both designs covering similar concentration ranges according to their experimental condition (UM/EM range: 0.10 - 100 uM, IM/PM range: 1-2000 uM). Additionally, each experimental design can subjected to variable degrees of experimental error using the `CV%` input. The inputs for the function the `update_data()` function include the following:

- **n_indiv** - number of subjects per diplotype group (default = 10).
- **CV%_D** - Average standard deviation for each diplotype group as a percentage of the true estimate for Vmax and Km (between subject variability for each diplotype group) (default = 25).
- **CV%** - Average coefficient of variation across all points for simulated Michaelis-Menten kinetics for each individual (residual error).
- **start** - Starting concentration from Michaelis-Menten simulation (default = 0).
- **end** - Ending concentration for Michaelis-Menten kinetics (default = 100 uM).
- **by** - Simulated concentration increment (default = 0.5).
- **Pop_freq** - TRUE or FALSE argument indicating whether to use true population frequency (default is FALSE).
- **Seed** - Random number generator seed (for reproducibility, default = 23457).
- **type** - Experimental design to be simulated (options: rich sampling (9pt) = **"rich"**, sparse sampling (3pt) = **"sparse3pt"**, sparse sampling (4pt) = **"sparse4pt"**, rich sampling for poor metabolizers (16 pt) = **"PM"**, sparse sampling for poor metabolizer (3pt) = **"PM_3pt"**, sparse sampling for poor metabolizer (4pt) = **"PM_4pt"**.

```
update_data <-function(`CV%`, type, n_indiv = 10, `CV%.D` = 25, by = 0.1,
                       start = 0, end = 100, Seed = 23457, Pop.freq = FALSE){

# Sampling Information --------------
# Full Range Set
full_set <- c(0.1, 0.5, 1, 2.5, 5, 10, 25, 50, 100)
PM_range <- c(1, 5, 10, 15, 25, 30, 50, 60, 90, 100, 250, 500, 800, 1000, 1600, 2000)

# Strategic Sampling Sets ----
sparse3pt_set1 <- c(0.1, 2.5, 50)
sparse3pt_set2 <- c(1, 10, 100)
sparse4pt_set1 <- c(0.1, 2.5, 25, 50)
sparse4pt_set2 <- c(1, 10, 50, 100)


PM_3pt_set1 <- c(10,100,1000)
PM_3pt_set2 <- c(25,250,2000)

PM_4pt_set1 <- c(1,10,100,1000)
PM_4pt_set2 <- c(5,25,250,2000)
```

```r
PM_range_set1 <- c(1,10,25,50,100,400,1000,2000)
PM_range_set2 <- c(5,15,30,60,90,200,800,1600)



# Rich Sampling Simulation ----------------
Rich <- population.sim(n = n_indiv, `CV%_D` = `CV%.D`, `CV%_V` = `CV%`, By = by,
                       Start = start, End = end, seed = Seed, pop_freq = Pop.freq) %>%
  filter(S %in% full_set) %>%
  mutate(Diplotype = as_factor(Diplotype),
         V = round(V_adj, digits = 3)) %>%
  select(ID, Diplotype, S, V)

# Sparse Sampling Simulation (3 point) ------------
Sparse3pt <- population.sim(n = n_indiv, `CV%_D` = `CV%.D`, `CV%_V` = `CV%`, By = by,
                            Start = start, End = end, seed = Seed, pop_freq = Pop.freq) %>%
  filter(S %in% full_set) %>%
  mutate(Set = if_else(as.numeric(ID) %% 2 == 0, "Set 2", "Set 1")) %>%
  filter(if_else(Set == "Set 1", S %in% sparse3pt_set1, S %in% sparse3pt_set2)) %>%
  mutate(Diplotype = as_factor(Diplotype),
         V = round(V_adj, digits = 3)) %>%
  select(ID, Diplotype, S, V)

# Sparse Sampling Simulation (4 point) --------------
Sparse4pt <- population.sim(n = n_indiv, `CV%_D` = `CV%.D`, `CV%_V` = `CV%`, By = by,
                            Start = start, End = end, seed = Seed, pop_freq = Pop.freq) %>%
  filter(S %in% full_set) %>%
  mutate(Set = if_else(as.numeric(ID) %% 2 == 0, "Set 2", "Set 1")) %>%
  filter(if_else(Set == "Set 1", S %in% sparse4pt_set1, S %in% sparse4pt_set2)) %>%
  mutate(Diplotype = as_factor(Diplotype),
         V = round(V_adj, digits = 3)) %>%
  select(ID, Diplotype, S, V)


PM <- population.sim(n = n_indiv, `CV%_D` = `CV%.D`, `CV%_V` = `CV%`, By = by,
                     Start = start, End = 2000, seed = Seed, pop_freq = Pop.freq) %>%
  filter(S %in% c(PM_range)) %>%
  mutate(Diplotype = as_factor(Diplotype),
         V = round(V_adj, digits = 3)) %>%
  select(ID, Diplotype, S, V)


PM_3pt <- population.sim(n = n_indiv, `CV%_D` = `CV%.D`, `CV%_V` = `CV%`, By = by,
                         Start = start, End = 2000, seed = Seed, pop_freq = Pop.freq) %>%
  filter(S %in% c(PM_range)) %>%
  mutate(Set = if_else(as.numeric(ID) %% 2 == 0, "Set 2", "Set 1")) %>%
  filter(if_else(Set == "Set 1", S %in% PM_3pt_set1, S %in% PM_3pt_set2)) %>%
  mutate(Diplotype = as_factor(Diplotype),
         V = round(V_adj, digits = 3)) %>%
  select(ID, Diplotype, S, V)

PM_4pt <- population.sim(n = n_indiv, `CV%_D` = `CV%.D`, `CV%_V` = `CV%`, By = by,
                         Start = start, End = 2000, seed = Seed, pop_freq = Pop.freq) %>%
```

```
  filter(S %in% c(PM_range)) %>%
  mutate(Set = if_else(as.numeric(ID) %% 2 == 0, "Set 2", "Set 1")) %>%
  filter(if_else(Set == "Set 1", S %in% PM_4pt_set1, S %in% PM_4pt_set2)) %>%
  mutate(Diplotype = as_factor(Diplotype),
         V = round(V_adj, digits = 3)) %>%
  select(ID, Diplotype, S, V)

# Output Selection ----------
switch(type,
       "rich" = Rich,
       "sparse3pt" = Sparse3pt,
       "sparse4pt" = Sparse4pt,
       "PM" = PM,
       "PM_3pt" = PM_3pt,
       "PM_4pt" = PM_4pt)

}
```

7.2 Representative Experimental Design Plots

Strategic sampling was accomplished by evenly dividing individuals falling under the same CYP2D6 genotype into two groups (Group 1 and Group 2). Each group was designated a pre-defined set of incubation concentrations to be used for the *in-silico* experiment with the total range of incubation concentrations dependent on the type of CYP2D6 metabolizer (as described in the *Strategic Sampling Approach* section above). Note: strategic sampling was not used for rich sample designs, meaning all subject were subjected to the same set of incubations concentrations.

Extensive and Ultra-Rapid Metabolizers

Incubation Concentrations - Rich (9pt) - Set: 0.1, 0.5, 1, 2.5, 5, 10, 25, 50, and 100 uM

Incubation Concentrations - Sparse (3pt) - Set 1: 0.1, 2.5, and 50 uM - Set 2: 1, 10, and 100 uM

Incubation Concentrations - Sparse (4pt) - Set 1: 0.1, 2.5, 25, and 50 uM - Set 2: 1, 10, 50, and 100 uM

Intermediate and Poor Metabolizers

Incubation Concentrations - Rich (16pt) - Set: 1, 5, 10, 15, 25, 30, 50, 60, 90, 100, 250, 500, 800, 1000, 1600, and 2000 uM

Incubation Concentrations - Sparse (3pt) - Set 1: 10,100, and 1000 uM - Set 2: 25, 250, and 2000 uM

Incubation Concentrations - Sparse (4pt) - Set 1: 1,10,100, and 1000 uM - Set 2: 5,25,250, and 2000 uM

```
# Generating Representative Data sets =========
## Representative data comes from 2 wild-type subjects.

rich_design.data <- update_data(`CV%` = 0, type = "rich") %>%
  filter(Diplotype == "1/1", ID %in% c(1,2)) %>%
  mutate(ID2 = paste("Subject",ID),
         `Study Design` = "Rich Design (9 pts)")

sparse_3pt_design.data <- update_data(`CV%` = 0, type = "sparse3pt")%>%
  filter(Diplotype == "1/1", ID %in% c(1,2)) %>%
  mutate(ID2 = paste("Subject",ID),
         `Study Design` = "Sparse Design (3 pts)")
```

```r
sparse_4pt_design.data <-  update_data(`CV%` = 0, type = "sparse4pt")%>%
  filter(Diplotype == "1/1", ID %in% c(1,2)) %>%
  mutate(ID2 = paste("Subject",ID),
         `Study Design` = "Sparse Design (4 pts)")

PM_design.data <- update_data(`CV%` = 0, type = "PM") %>%
  filter(Diplotype == "4/5", ID %in% c(81,82)) %>%
  mutate(ID2 = paste("Subject",ID),
         `Study Design` = "Rich Design (16 pts)")

PM3pt_design.data <- update_data(`CV%` = 0, type = "PM_3pt") %>%
  filter(Diplotype == "4/5", ID %in% c(81,82)) %>%
  mutate(ID2 = paste("Subject",ID),
         `Study Design` = "Sparse Design (3 pts)")

PM4pt_design.data <- update_data(`CV%` = 0, type = "PM_4pt") %>%
  filter(Diplotype == "4/5", ID %in% c(81,82)) %>%
  mutate(ID2 = paste("Subject",ID),
         `Study Design` = "Sparse Design (4 pts)")

# Solution Grid
design.nls <- nls(formula = V ~ (Vmax*S)/(Km + S), data = rich_design.data,
start = list(Vmax = 350, Km = 2))
grid <- tibble(S = seq(0,100,0.1),
               V = (coef(design.nls)[[1]]*S)/(coef(design.nls)[[2]]+S))

design_PM.nls <- nls(formula = V ~ (Vmax*S)/(Km + S), data = PM_design.data,
start = list(Vmax = 20, Km = 50))
grid_pm <- tibble(S = seq(0,2000,0.1),
               V = (coef(design_PM.nls)[[1]]*S)/(coef(design_PM.nls)[[2]]+S))

# UM/EM STRATEGIC SAMPLING DESIGN FIGURE ================
## Representative plots contain data for 2 wild-type subjects.

# Rich Design--------
D1<- ggplot(rich_design.data, aes(x = S, y = V))+
  geom_line(data = grid, size = 1.5, alpha = 0.75)+
  geom_point(aes(color=ID2, shape = ID2), size = 3.5)+
  theme_bw(base_size = 14)+
  ggeasy::easy_remove_legend_title()+
  theme(legend.position = c(0.7,0.2))+
  facet_wrap(~`Study Design`, ncol = 1)+
  labs(title = "Conventional")+
  xlab("[Substrate](uM)")+
  ylab("Reaction Rate (V)\n")+
  scale_color_viridis_d(option = "A", begin = 0.2, end = 0.6)

# Sparse Design (3pt)----------
D2<- ggplot(sparse_3pt_design.data, aes(x = S, y = V))+
  geom_line(data = grid, size = 1.5, alpha = 0.75)+
  geom_point(aes(color=ID2, shape = ID2), size = 3.5)+
  theme_bw(base_size = 14)+
  ggeasy::easy_remove_legend_title()+
```

```
  theme(legend.position = c(0.7,0.2))+
  facet_wrap(~`Study Design`, ncol = 1)+
  labs(title = "Strategic Scenario (2)")+
  xlab("[Substrate](uM)")+
  ylab("Reaction Rate (V)\n")+
  scale_color_viridis_d(option = "A", begin = 0.2, end = 0.6)

# Sparse Design (4pt)------------
D3 <- ggplot(sparse_4pt_design.data, aes(x = S, y = V))+
  geom_line(data = grid, size = 1.5, alpha = 0.75)+
  geom_point(aes(color=ID2, shape = ID2), size = 3.5)+
  theme_bw(base_size = 14)+
  ggeasy::easy_remove_legend_title()+
  theme(legend.position = c(0.7,0.2))+
  facet_wrap(~`Study Design`, ncol = 1)+
  labs(title = "Strategic Scenario (1)")+
  xlab("[Substrate](uM)")+
  ylab("Reaction Rate (V)\n")+
  scale_color_viridis_d(option = "A", begin = 0.2, end = 0.6)


# IM/PM STRATEGIC SAMPLING DESIGN FIGURE ================
## Representative plots contain data for 2 wild-type subjects.

# PM Sparse Design (4pt)---------
D4 <- ggplot(PM4pt_design.data, aes(x = S, y = V))+
  geom_line(data = grid_pm, size = 1.5, alpha = 0.75)+
  geom_point(aes(color=ID2, shape = ID2), size = 3.5)+
  theme_bw(base_size = 14)+
  ggeasy::easy_remove_legend_title()+
  theme(legend.position = c(0.7,0.2))+
  facet_wrap(~`Study Design`, ncol = 1)+
  labs(title = "")+
  xlab("[Substrate](uM)")+
  ylab("Reaction Rate (V)\n")+
  scale_color_viridis_d(option = "A", begin = 0.2, end = 0.6)

# Extensize metabolizer model---------
D5 <- D3 + labs(title = "") #4pt
D6 <- D2 + labs(title = "") #3pt
D7 <- D1 + labs(title = "Extensive Metabolizer") #9pt


# PM Sparse Design (3pt)---------
D8 <- ggplot(PM3pt_design.data, aes(x = S, y = V))+
  geom_line(data = grid_pm, size = 1.5, alpha = 0.75)+
  geom_point(aes(color=ID2, shape = ID2), size = 3.5)+
  theme_bw(base_size = 14)+
  ggeasy::easy_remove_legend_title()+
  theme(legend.position = c(0.7,0.2))+
  facet_wrap(~`Study Design`, ncol = 1)+
  labs(title = "")+
  xlab("[Substrate](uM)")+
```

```
  ylab("Reaction Rate (V)\n")+
  scale_color_viridis_d(option = "A", begin = 0.2, end = 0.6)

# PM Rich Design (16pt)---------
D9 <- ggplot(PM_design.data, aes(x = S, y = V))+
  geom_line(data = grid_pm, size = 1.5, alpha = 0.75)+
  geom_point(aes(color=ID2, shape = ID2), size = 3.5)+
  theme_bw(base_size = 14)+
  ggeasy::easy_remove_legend_title()+
  theme(legend.position = c(0.7,0.2))+
  facet_wrap(~`Study Design`, ncol = 1)+
  labs(title = "Poor Metabolizers")+
  xlab("[Substrate](uM)")+
  ylab("Reaction Rate (V)\n")+
  scale_color_viridis_d(option = "A", begin = 0.2, end = 0.6)


# Combined Plots -----------
## Extensive Metabolizers
ggpubr::ggarrange(D7,D5,D6,ncol = 2, nrow = 2, labels = "AUTO")
## Poor Metabolizers
ggpubr::ggarrange(D9,D4,D8, ncol = 2, nrow = 2, labels = "AUTO")
```

8 Population Michaelis-Menten Modeling (UM/EM)

8.1 Generation of Experimental Data (Base Model - 0% CV)

```
# Population Datasets (CV = 0%)
rich.data <- update_data(`CV%` = 0, type = "rich") %>%
  filter(!Diplotype %in% c("4/41", "4/5"))
sparse3pt.data <- update_data(`CV%` = 0, type = "sparse3pt") %>%
  filter(!Diplotype %in% c("4/41", "4/5"))
sparse4pt.data <- update_data(`CV%` = 0, type = "sparse4pt") %>%
  filter(!Diplotype %in% c("4/41", "4/5"))
```

8.2 Individual-Fits of Experimental Data (Base Model - 0% CV)

A grouped data frame was created for each experiment where reaction rate (V) is predicted by substrate concentration (S) according to subject (ID). Individual fits (un-weighted) and parameter estimates for the data sets were obtained using non-linear least squares function `nlsList()`. Values obtained from the `nlsList()` were used as initial estimates for the non-linear mixed effect model.

```
# Grouped Data frame -------------
rich_data.grp <- groupedData(V~S|ID, rich.data)
sparse3pt.grp <- groupedData(V~S|ID, sparse3pt.data)
sparse4pt.grp <- groupedData(V~S|ID, sparse4pt.data)

# Fit Model ((Non-Linear Least Squares) ------------
rich_fit.nls <- nlsList(V~SSmicmen(S,Vmax,Km), data = rich_data.grp)
sparse3pt_fit.nls <- nlsList(V~SSmicmen(S,Vmax,Km), data = sparse3pt.grp)
sparse4pt_fit.nls <- nlsList(V~SSmicmen(S,Vmax,Km), data = sparse4pt.grp)
```

8.3 Non-Linear Mixed Effect Modeling (Base Model - 0% CV)

Non-linear mixed effect modeling was conducted using the `nlme()` package version 3.1 (Bates et al., 2000). The `nlsList()` fits were used as initial estimates for the NLME model. Individual level random effects were applied to both Vmax and Km. A diagonal variance-covariance structure was assigned to each model using the `pdDiag()` function.

```
# Fit Model ((Non-Linear Mixed-Effect) ------------
rich.nlme <- nlme(rich_fit.nls, random = pdDiag(Vmax + Km ~ 1))
sparse3pt.nlme <- nlme(sparse3pt_fit.nls, random = pdDiag(Vmax + Km ~ 1))
sparse4pt.nlme <- nlme(sparse4pt_fit.nls, random = pdDiag(Vmax + Km ~ 1))


# Model Fixed Effect comparisons
fixef.table <- rbind(fixef(rich.nlme),
      fixef(sparse3pt.nlme),
      fixef(sparse4pt.nlme)) %>%
  as_tibble() %>%
  mutate(Model = c("Rich","Sparse (3pt)","Sparse (4pt)")) %>%
  relocate(Model)

fixef.table %>%
  mutate(across(where(is.numeric), round, 2)) %>%
```

```
  kbl( table.attr = "style='width:60%;'",
       caption =  "Base Model: Mixed Effect Michaelis-Menten") %>%
  kable_classic("striped")
```

8.4 Non-Linear Mixed Effect Modeling (Covariate Model - 0% CV)

CYP2D6 genotype was incorporated as a categorical covariate in the model by adding it as a fixed effect that impacts both Vmax and Km. The extracted population level fixed effects for Vmax and Km from the base model (described in the previous section) were used as initial estimates for the covariate model.

```
#Models with genotype as a Covariate -----
rich.fxf <- fixef.table[1,2:3]
sparse3pt.fxf <- fixef.table[2,2:3]
sparse4pt.fxf <- fixef.table[3,2:3]


rich_covar.nlme <- update(rich.nlme, fixed = Vmax + Km ~ Diplotype,
                      start = c(rich.fxf[[1]], rep(0,6),
                                rich.fxf[[2]], rep(0,6)))

sparse3pt_covar.nlme <- update(sparse3pt.nlme, fixed = Vmax + Km ~ Diplotype,
                      start = c(sparse3pt.fxf[[1]], rep(0,6),
                                sparse3pt.fxf[[2]], rep(0,6)))

sparse4pt_covar.nlme <- update(sparse4pt.nlme, fixed = Vmax + Km ~ Diplotype,
                      start = c(sparse4pt.fxf[[1]], rep(0,6),
                                sparse4pt.fxf[[2]], rep(0,6)))
```

8.5 Base Model and Covariate Model Evaluation (0% CV)

The impact of adding CYP2D6 genotype as a (covariate on both Vmax and Km) and the whether adding covariates to the model improved fit compared to baseline was assessed using analysis of variance `anova()`.

8.5.1 Overall Impact of adding covariates to base model

```
anova(rich.nlme, rich_covar.nlme)
anova(sparse3pt.nlme, sparse3pt_covar.nlme)
anova(sparse4pt.nlme, sparse4pt_covar.nlme)
```

8.5.2 Impact of adding covariate to Vmax and Km

```
anova(rich_covar.nlme) %>%
  kbl(caption = "Rich Model (w/Covariates)",
      table.attr = "style='width:60%;'") %>%
  kable_classic(full_width = T, "striped")

anova(sparse3pt_covar.nlme)%>%
  kbl(caption = "Sparse 3pt Model (w/Covariates)",
      table.attr = "style='width:60%;'") %>%
  kable_classic(full_width = T, "striped")

anova(sparse4pt_covar.nlme)%>%
```

```
  kbl(caption = "Sparse 4pt Model (w/Covariates)",
      table.attr = "style='width:60%;'") %>%
  kable_classic(full_width = T, "striped")
```

8.6 Summary Tables of Fixed Effects Estimates (Covariate Model - 0% CV)

8.6.1 Table of Covariate Model Estimates

```
# Covariate Analysis Summary Table ------------
covaref.table <- rbind(fixef(rich_covar.nlme),
      fixef(sparse3pt_covar.nlme),
      fixef(sparse4pt_covar.nlme)) %>%
  as_tibble() %>%
  mutate(`Covariate Model` = c("Rich","Sparse (3pt)","Sparse (4pt)")) %>%
  pivot_longer(`Vmax.(Intercept)`:`Km.Diplotype2/4`,
               names_to = "Parameter",
               values_to = "Value") %>%
  pivot_wider(names_from = `Covariate Model`, values_from = Value) %>%
  separate(Parameter, sep = "\\.", into = c("Parameter","Diplotype"), remove = T) %>%
  mutate(Diplotype = recode(Diplotype, "(Intercept)" = "Reference (1/1)"),
         across(where(is.numeric), round, 2))
```

8.6.2 Reference Group (*1/*1) Parameter Estimates

```
# Reference Estimates -----------
covaref.table %>%
  filter(Diplotype == "Reference (1/1)")%>%
  kbl(caption = "Reference Population Estimates",
      table.attr = "style='width:60%;'") %>%
  kable_classic(full_width = T, "striped")
```

8.6.3 Variant Group ETAs Table

```
# CYP2D6 Genotype Effects --------
covaref.table%>%
  kbl(caption = "Michaelis Menten Mixed-Effect Model (w/Covariates)",
      table.attr = "style='width:60%;'") %>%
  kable_classic(full_width = T, "striped") %>%
  pack_rows("Vmax Estimate (Population)", 1,1) %>%
  pack_rows("Covariate (Vmax Eta)", 2, 7)%>%
  pack_rows("Km Estimate (Population)", 8,8) %>%
  pack_rows("Covariate (Km Eta)", 9, 14)
```

8.6.4 Covariate Model 95% Confidence Interval Table (0% CV)

```
# Function to Extract Confidence Interval Info ----------
extract_CI <- function(int.data, name){

temp <- int.data$fixed %>%
  as_tibble() %>%
  mutate(across(where(is.numeric), round, 2),
         `CI (95%)`  = paste0("(",lower,", ",upper,")")) %>%
```

```
  select(est., `CI (95%)`)

  colnames(temp) <- c(paste(name, " Estimate"), paste(name, " CI (95%)"))
  temp
}

# Confidence Interval by Sampling Scenario -------
rich_covar.intervals <- intervals(rich_covar.nlme, which = "fixed")
sparse3pt_covar.intervals <- intervals(sparse3pt_covar.nlme, which = "fixed")
sparse4pt_covar.intervals <- intervals(sparse4pt_covar.nlme, which = "fixed")


# Summary Table of Confidence Intervals by Diplotype and Scenario -----------
covar_ci.table <- cbind(extract_CI(rich_covar.intervals, name = "Rich"),
                        extract_CI(sparse3pt_covar.intervals, name = "Sparse 3pt"),
                        extract_CI(sparse4pt_covar.intervals, name = "Sparse 4pt")) %>%
  mutate(Parameter = rownames(rich_covar.intervals$fixed)) %>%
  relocate(Parameter) %>%
  separate(Parameter, remove = T, sep = "\\.", into = c("Parameter", "Diplotype"))%>%
  mutate(Diplotype = recode(Diplotype, "(Intercept)" = "Reference (1/1)"))
  colnames(covar_ci.table)<- c("Parameter", "Diplotype",
                               "Estimate","CI (95%)",
                               "Estimate", "CI (95%)",
                               "Estimate", "CI (95%)")
covar_ci.table%>%
  kbl(caption = "Michaelis Menten Mixed-Effect Model (w/Covariates)") %>%
  kable_classic(full_width = T, "striped")%>%
  pack_rows("Vmax Estimate (Population)", 1,1) %>%
  pack_rows("Covariate (Vmax Eta)", 2, 7)%>%
  pack_rows("Km Estimate (Population)", 8,8) %>%
  pack_rows("Covariate (Km Eta)", 9, 14) %>%
  add_header_above(c(" ", " ",
                     "Rich (9pt)" = 2,
                     "Sparse (3pt)" = 2,
                     "Sparse (4pt)" = 2))
```

9 Population Modeling w/Residual Error (UM/EM)

Data sets with include increasing degrees of residual error (5, 10, and 20% CV) were generated using the `update_data()` function described earlier (see *In-Silico Experiments* section for full description). Additonally, the same methodology for model analysis used in the the previous section (*Population Michaelis-Menten Modeling (UM/EM)*) was applied here to evaluate the impact of increasing experimental error on parameter estimates.

9.1 Generation of Experimental Data (Base Model - 5-20% CV)

```
# Population Data sets (CV = 5%) ----------------
rich_CV5.data <- update_data(`CV%` = 5, type = "rich")%>%
  filter(!Diplotype %in% c("4/41", "4/5"))
sparse3pt_CV5.data <- update_data(`CV%` = 5, type = "sparse3pt")%>%
  filter(!Diplotype %in% c("4/41", "4/5"))
sparse4pt_CV5.data <- update_data(`CV%` = 5, type = "sparse4pt")%>%
  filter(!Diplotype %in% c("4/41", "4/5"))

# Population Data sets (CV = 10%) ----------------
rich_CV10.data <- update_data(`CV%` = 10, type = "rich")%>%
  filter(!Diplotype %in% c("4/41", "4/5"))
sparse3pt_CV10.data <- update_data(`CV%` = 10, type = "sparse3pt")%>%
  filter(!Diplotype %in% c("4/41", "4/5"))
sparse4pt_CV10.data <- update_data(`CV%` = 10, type = "sparse4pt")%>%
  filter(!Diplotype %in% c("4/41", "4/5"))

# Population Data sets (CV = 20%) ----------------
rich_CV20.data <- update_data(`CV%` = 20, type = "rich")%>%
  filter(!Diplotype %in% c("4/41", "4/5"))
sparse3pt_CV20.data <- update_data(`CV%` = 20, type = "sparse3pt")%>%
  filter(!Diplotype %in% c("4/41", "4/5"))
sparse4pt_CV20.data <- update_data(`CV%` = 20, type = "sparse4pt")%>%
  filter(!Diplotype %in% c("4/41", "4/5"))
```

9.2 Inidividual-Fits of Experimental Data (Base Model - 5-20% CV)

```
# Grouped Data frame ============================
rich_CV5_data.grp <- groupedData(V~S|ID, rich_CV5.data)
sparse3pt_CV5.grp <- groupedData(V~S|ID, sparse3pt_CV5.data)
sparse4pt_CV5.grp <- groupedData(V~S|ID, sparse4pt_CV5.data)

rich_CV10_data.grp <- groupedData(V~S|ID, rich_CV10.data)
sparse3pt_CV10.grp <- groupedData(V~S|ID, sparse3pt_CV10.data)
sparse4pt_CV10.grp <- groupedData(V~S|ID, sparse4pt_CV10.data)

rich_CV20_data.grp <- groupedData(V~S|ID, rich_CV20.data)
sparse3pt_CV20.grp <- groupedData(V~S|ID, sparse3pt_CV20.data)
sparse4pt_CV20.grp <- groupedData(V~S|ID, sparse4pt_CV20.data)


# Fit Model ((Non-Linear Least Squares) =====================
rich_CV5.nls <- nlsList(V~SSmicmen(S,Vmax,Km), data = rich_CV5_data.grp)
sparse3pt_CV5.nls <- nlsList(V~SSmicmen(S,Vmax,Km), data = sparse3pt_CV5.grp)
```

```
sparse4pt_CV5.nls <- nlsList(V~SSmicmen(S,Vmax,Km), data = sparse4pt_CV5.grp)

rich_CV10.nls <- nlsList(V~SSmicmen(S,Vmax,Km), data = rich_CV10_data.grp)
sparse3pt_CV10.nls <- nlsList(V~SSmicmen(S,Vmax,Km), data = sparse3pt_CV10.grp)
sparse4pt_CV10.nls <- nlsList(V~SSmicmen(S,Vmax,Km), data = sparse4pt_CV10.grp)

rich_CV20.nls <- nlsList(V~SSmicmen(S,Vmax,Km), data = rich_CV20_data.grp)
sparse3pt_CV20.nls <- nlsList(V~SSmicmen(S,Vmax,Km), data = sparse3pt_CV20.grp)
sparse4pt_CV20.nls <- nlsList(V~SSmicmen(S,Vmax,Km), data = sparse4pt_CV20.grp)
```

9.3 Non-Linear Mixed Effect Modeling (Base Model - 5-20% CV)

```
# Fit Model ((Non-Linear Mixed-Effect) ====================
rich_CV5.nlme <- nlme(rich_CV5.nls, random = pdDiag(Vmax + Km ~ 1))
sparse3pt_CV5.nlme <- nlme(sparse3pt_CV5.nls, random = pdDiag(Vmax + Km ~ 1))
sparse4pt_CV5.nlme <- nlme(sparse4pt_CV5.nls, random = pdDiag(Vmax + Km ~ 1))

rich_CV10.nlme <- nlme(rich_CV10.nls, random = pdDiag(Vmax + Km ~ 1))
sparse3pt_CV10.nlme <- nlme(sparse3pt_CV10.nls, random = pdDiag(Vmax + Km ~ 1))
sparse4pt_CV10.nlme <- nlme(sparse4pt_CV10.nls, random = pdDiag(Vmax + Km ~ 1))

rich_CV20.nlme <- nlme(rich_CV20.nls, random = pdDiag(Vmax + Km ~ 1))
sparse3pt_CV20.nlme <- nlme(sparse3pt_CV20.nls, random = pdDiag(Vmax + Km ~ 1))
sparse4pt_CV20.nlme <- nlme(sparse4pt_CV20.nls, random = pdDiag(Vmax + Km ~ 1))
```

9.4 Impact of Residual Error on Fixed Effects

9.4.1 Summary Table of Fixed Effects (w/Residual Error)

```
# Model Fixed Effect comparisons----------
fixef_CV.table <- rbind(
      fixef(rich.nlme),
      fixef(sparse3pt.nlme),
      fixef(sparse4pt.nlme),
      fixef(rich_CV5.nlme),
      fixef(sparse3pt_CV5.nlme),
      fixef(sparse4pt_CV5.nlme),
      fixef(rich_CV10.nlme),
      fixef(sparse3pt_CV10.nlme),
      fixef(sparse4pt_CV10.nlme),
      fixef(rich_CV20.nlme),
      fixef(sparse3pt_CV20.nlme),
      fixef(sparse4pt_CV20.nlme)) %>%
  as_tibble() %>%
  mutate(Model = c("Rich (0% CV)","Sparse (0% CV, 3pt)","Sparse (0% CV, 4pt)",
                   "Rich (5% CV)","Sparse (5% CV, 3pt)","Sparse (5% CV, 4pt)",
                   "Rich (10% CV)","Sparse (10% CV, 3pt)","Sparse (10% CV, 4pt)",
                   "Rich (20% CV)","Sparse (20% CV, 3pt)","Sparse (20% CV, 4pt)"),
         across(where(is.numeric),round,3)) %>%
  relocate(Model)

# Fixed Effect Table across conditions ------------------
fixef_CV.table %>%
```

```
  kbl(caption = "Population Estimates (w/ Residual Error)",
      table.attr = "style='width:60%;'") %>%
  kable_classic(full_width = T, "striped")%>%
  pack_rows(" ", 1,3) %>%
  pack_rows(" ", 4, 6)%>%
  pack_rows(" ", 7, 9)%>%
  pack_rows(" ", 10, 12)
```

9.4.2 Summary Table of Fixed Effects (w/Residual Error & 95% CI)

```
# Function to extract confidence intervals from model ---------
extract_CI2 <- function(int.data, name){

temp <- int.data$fixed %>%
  as_tibble() %>%
  mutate(across(where(is.numeric), round, 2),
         `CI (95%)`  = paste0("(",lower,", ",upper,")")) %>%
  select(est., `CI (95%)`)

  colnames(temp) <- c("Estimate","CI (95%)")
  temp$Model <- name
  temp$Parameter <- c("Vmax","Km")
  temp
}

# Confidence Interval Data ---------
rich_CV5.intervals <- intervals(rich_CV5.nlme, which = "fixed")
sparse3pt_CV5.intervals <- intervals(sparse3pt_CV5.nlme, which = "fixed")
sparse4pt_CV5.intervals <- intervals(sparse4pt_CV5.nlme, which = "fixed")

rich_CV10.intervals <- intervals(rich_CV10.nlme, which = "fixed")
sparse3pt_CV10.intervals <- intervals(sparse3pt_CV10.nlme, which = "fixed")
sparse4pt_CV10.intervals <- intervals(sparse4pt_CV10.nlme, which = "fixed")

rich_CV20.intervals <- intervals(rich_CV20.nlme, which = "fixed")
sparse3pt_CV20.intervals <- intervals(sparse3pt_CV20.nlme, which = "fixed")
sparse4pt_CV20.intervals <- intervals(sparse4pt_CV20.nlme, which = "fixed")



# Confidence Interval Tables ----------
CV5_ci.table <- rbind(extract_CI2(rich_CV5.intervals, name = "Rich (5% CV)"),
                      extract_CI2(sparse3pt_CV5.intervals, name = "Sparse 3pt (5% CV)"),
                      extract_CI2(sparse4pt_CV5.intervals, name = "Sparse 4pt (5% CV)")) %>%
  relocate(Model, Parameter) %>%
  arrange(desc(Parameter))

CV10_ci.table <- rbind(extract_CI2(rich_CV10.intervals, name = "Rich (10% CV)"),
                      extract_CI2(sparse3pt_CV10.intervals, name = "Sparse 3pt (10% CV)"),
                      extract_CI2(sparse4pt_CV10.intervals, name = "Sparse 4pt (10% CV)")) %>%
  relocate(Model, Parameter) %>%
  arrange(desc(Parameter))
```

```
CV20_ci.table <- rbind(extract_CI2(rich_CV20.intervals, name = "Rich (20% CV)"),
                       extract_CI2(sparse3pt_CV20.intervals, name = "Sparse 3pt (20% CV)"),
                       extract_CI2(sparse4pt_CV20.intervals, name = "Sparse 4pt (20% CV)")) %>%
  relocate(Model, Parameter) %>%
  arrange(desc(Parameter))


# Summary Table of Estimates with CIs ----------
rbind(CV5_ci.table, CV10_ci.table, CV20_ci.table) %>%
  arrange(desc(Parameter))%>%
  kbl(caption = "Population Estimates (w/ Residual Error & 95% CI)",
      table.attr = "style='width:60%;'") %>%
  kable_classic(full_width = T, "striped")%>%
  pack_rows("Vmax Estimates", 1,9) %>%
  pack_rows("", 1, 3) %>%
  pack_rows("", 4, 6) %>%
  pack_rows("", 7, 9) %>%
  pack_rows("Km Estimates", 10, 18) %>%
  pack_rows(" ", 10, 12) %>%
  pack_rows(" ", 13, 15) %>%
  pack_rows(" ", 16, 18)
```

9.5 Non-Linear Mixed Effect Modeling (Covariate Model - 5-20% CV)

```
# Models with Diplotype as a Covariate -----
rich_CV5.fxf <- fixef_CV.table[1,2:3]
sparse3pt_CV5.fxf <- fixef_CV.table[2,2:3]
sparse4pt_CV5.fxf <- fixef_CV.table[3,2:3]

rich_CV10.fxf <- fixef_CV.table[4,2:3]
sparse3pt_CV10.fxf <- fixef_CV.table[5,2:3]
sparse4pt_CV10.fxf <- fixef_CV.table[6,2:3]

rich_CV20.fxf <- fixef_CV.table[7,2:3]
sparse3pt_CV20.fxf <- fixef_CV.table[8,2:3]
sparse4pt_CV20.fxf <- fixef_CV.table[9,2:3]


# NLME Covariate Model w/5% CV ==================
rich_CV5_covar.nlme <- update(rich_CV5.nlme, fixed = Vmax + Km ~ Diplotype,
                    start = c(rich_CV5.fxf[[1]], rep(0,6),
                              rich_CV5.fxf[[2]], rep(0,6)))

sparse3pt_CV5_covar.nlme <- update(sparse3pt_CV5.nlme, fixed = Vmax + Km ~ Diplotype,
                    start = c(sparse3pt_CV5.fxf[[1]], rep(0,6),
                              sparse3pt_CV5.fxf[[2]], rep(0,6)))

sparse4pt_CV5_covar.nlme <- update(sparse4pt_CV5.nlme, fixed = Vmax + Km ~ Diplotype,
                    start = c(sparse4pt_CV5.fxf[[1]], rep(0,6),
                              sparse4pt_CV5.fxf[[2]], rep(0,6)))


# NLME Covariate Model w/10% CV ==================
```

```
rich_CV10_covar.nlme <- update(rich_CV10.nlme, fixed = Vmax + Km ~ Diplotype,
                     start = c(rich_CV10.fxf[[1]], rep(0,6),
                               rich_CV10.fxf[[2]], rep(0,6)))

sparse3pt_CV10_covar.nlme <- update(sparse3pt_CV10.nlme, fixed = Vmax + Km ~ Diplotype,
                     start = c(sparse3pt_CV10.fxf[[1]], rep(0,6),
                               sparse3pt_CV10.fxf[[2]], rep(0,6)))

sparse4pt_CV10_covar.nlme <- update(sparse4pt_CV10.nlme, fixed = Vmax + Km ~ Diplotype,
                     start = c(sparse4pt_CV10.fxf[[1]], rep(0,6),
                               sparse4pt_CV10.fxf[[2]], rep(0,6)))


# NLME Covariate Model w/20% CV ==================
rich_CV20_covar.nlme <- update(rich_CV20.nlme, fixed = Vmax + Km ~ Diplotype,
                     start = c(rich_CV20.fxf[[1]], rep(0,6),
                               rich_CV20.fxf[[2]], rep(0,6)))

sparse3pt_CV20_covar.nlme <- update(sparse3pt_CV20.nlme, fixed = Vmax + Km ~ Diplotype,
                     start = c(sparse3pt_CV20.fxf[[1]], rep(0,6),
                               sparse3pt_CV20.fxf[[2]], rep(0,6)))

sparse4pt_CV20_covar.nlme <- update(sparse4pt_CV20.nlme, fixed = Vmax + Km ~ Diplotype,
                     start = c(sparse4pt_CV20.fxf[[1]], rep(0,6),
                               sparse4pt_CV20.fxf[[2]], rep(0,6)))
```

9.6 Covariate Models Evaluation (5-20% CV)

```
# Summary of Covariate Models (w/ variability) -------
anova(rich_CV5_covar.nlme)
anova(sparse3pt_CV5_covar.nlme)
anova(sparse4pt_CV5_covar.nlme)

anova(rich_CV10_covar.nlme)
anova(sparse3pt_CV10_covar.nlme)
anova(sparse4pt_CV10_covar.nlme)

anova(rich_CV20_covar.nlme)
anova(sparse3pt_CV20_covar.nlme)
anova(sparse4pt_CV20_covar.nlme)
```

10 Weighting Non-Linear Mixed Effect Model (UM/EM)

10.1 Evaluating Heteroscedacity of Residual Error (UM/EM)

Heterogeneity of variance was examined qualitatively by plotting standardized residules (Pearson). Heteroscedacity was corrected by modeleling residual variance as a power function of mean fitted value using the `varPower()` function.

```
# Model Comparison ---------
# 0% CV
plot(rich_covar.nlme, main = "Rich (9pt, 0% CV)")
plot(rich_covar.nlme %>% update(weights = varPower()),
     main = "Rich (9pt, 0% CV, weighted)")

plot(sparse3pt_covar.nlme, main = "Sparse (3pt, 0% CV)")
plot(sparse3pt_covar.nlme %>% update(weights = varPower()),
     main = "Sparse (3pt, 0% CV, weighted)")

plot(sparse4pt_covar.nlme, main = "Sparse (4pt, 0% CV)")
plot(sparse4pt_covar.nlme %>% update(weights = varPower()),
     main = "Sparse (4pt, 0% CV, weighted)")

# 5% CV
plot(rich_CV5_covar.nlme, main = "Rich (9pt, 5% CV)")
plot(rich_CV5_covar.nlme %>% update(weights = varPower()),
     main = "Rich (9pt, 5% CV, weighted)")

plot(sparse3pt_CV5_covar.nlme, main = "Sparse (3pt, 5% CV)")
plot(sparse3pt_CV5_covar.nlme %>% update(weights = varPower()),
     main = "Sparse (3pt, 5% CV, weighted)")

plot(sparse4pt_CV5_covar.nlme, main = "Sparse (4pt, 5% CV)")
plot(sparse4pt_CV5_covar.nlme %>% update(weights = varPower()),
     main = "Sparse (4pt, 5% CV, weighted)")

# 10% CV
plot(rich_CV10_covar.nlme, main = "Rich (9pt, 10% CV)")
plot(rich_CV10_covar.nlme %>% update(weights = varPower()),
     main = "Rich (9pt, 10% CV, weighted)")

plot(sparse3pt_CV10_covar.nlme, main = "Sparse (3pt, 10% CV)")
plot(sparse3pt_CV10_covar.nlme %>% update(weights = varPower()),
     main = "Sparse (3pt, 10% CV, weighted)")

plot(sparse4pt_CV10_covar.nlme, main = "Sparse (4pt, 10% CV)")
plot(sparse4pt_CV10_covar.nlme %>% update(weights = varPower()),
     main = "Sparse (4pt, 10% CV, weighted)")

# 20% CV
plot(rich_CV20_covar.nlme, main = "Rich (9pt, 20% CV)")
plot(rich_CV20_covar.nlme %>% update(weights = varPower()),
     main = "Rich (9pt, 20% CV, weighted)")
```

```
plot(sparse3pt_CV20_covar.nlme, main = "Sparse (3pt, 20% CV)")
plot(sparse3pt_CV20_covar.nlme %>% update(weights = varPower()),
     main = "Sparse (3pt, 20% CV, weighted)")

plot(sparse4pt_CV20_covar.nlme, main = "Sparse (4pt, 20% CV)")
plot(sparse4pt_CV20_covar.nlme %>% update(weights = varPower()),
     main = "Sparse (4pt, 20% CV, weighted)")
```

10.2 Adjusting Residual Error Models

```
# Adjusting Residual Error Model ----
rich_CV5_covar.nlme <- rich_CV5_covar.nlme %>% update(weights = varPower())
sparse3pt_CV5_covar.nlme <- sparse3pt_CV5_covar.nlme %>% update(weights = varPower())
sparse4pt_CV5_covar.nlme  <- sparse4pt_CV5_covar.nlme  %>% update(weights = varPower())
rich_CV10_covar.nlme <- rich_CV10_covar.nlme %>% update(weights = varPower())
sparse3pt_CV10_covar.nlme <- sparse3pt_CV10_covar.nlme %>% update(weights = varPower())
sparse4pt_CV10_covar.nlme <- sparse4pt_CV10_covar.nlme %>% update(weights = varPower())
rich_CV20_covar.nlme <- rich_CV20_covar.nlme %>% update(weights = varPower())
sparse3pt_CV20_covar.nlme <- sparse3pt_CV20_covar.nlme %>% update(weights = varPower())
sparse4pt_CV20_covar.nlme <- sparse4pt_CV20_covar.nlme %>% update(weights = varPower())
```

10.3 Covariate Model Summary Data

```
# Creating covariate fixed effect table ------
covaref_CV.table <- rbind(fixef(rich_CV5_covar.nlme),
      fixef(sparse3pt_CV5_covar.nlme),
      fixef(sparse4pt_CV5_covar.nlme),
      fixef(rich_CV10_covar.nlme),
      fixef(sparse3pt_CV10_covar.nlme),
      fixef(sparse4pt_CV10_covar.nlme),
      fixef(rich_CV20_covar.nlme),
      fixef(sparse3pt_CV20_covar.nlme),
      fixef(sparse4pt_CV20_covar.nlme)) %>%
  as_tibble() %>%
  mutate(`Covariate Model` = c("Rich (5% CV)","Sparse (5% CV, 3pt)","Sparse (5% CV, 4pt)",
                  "Rich (10% CV)","Sparse (10% CV, 3pt)","Sparse (10% CV, 4pt)",
                  "Rich (20% CV)","Sparse (20% CV, 3pt)","Sparse (20% CV, 4pt)")) %>%
  pivot_longer(`Vmax.(Intercept)`:`Km.Diplotype2/4`,
               names_to = "Parameter",
               values_to = "Value") %>%
  pivot_wider(names_from = `Covariate Model`, values_from = Value) %>%
  separate(Parameter, sep = "\\.", into = c("Parameter","Diplotype"), remove = T) %>%
  mutate(Diplotype = recode(Diplotype, "(Intercept)" = "Reference (1/1)"),
         across(where(is.numeric), round, 2))

# Complete Covariate Effects Table --------------------
covar_Vmax.table <- cbind(covaref.table, covaref_CV.table %>%
                            select(-Parameter, -Diplotype)) %>% filter(Parameter == "Vmax")
covar_Km.table <- cbind(covaref.table, covaref_CV.table %>%
                            select(-Parameter, -Diplotype)) %>% filter(Parameter == "Km")
```

```r
# Vmax Estimates ------------------------------
final_vmax.est <- covar_Vmax.table %>%
  pivot_longer(cols = Rich:`Sparse (20% CV, 4pt)`,
               names_to = "Condition",
               values_to = "Estimate (Eta)") %>%
  group_by(Condition) %>%
  mutate(Predicted = if_else(Diplotype != "Reference (1/1)",
                             `Estimate (Eta)` + `Estimate (Eta)`[Diplotype == "Reference (1/1)"],
                             `Estimate (Eta)`),
         Diplotype = recode(Diplotype, "Reference (1/1)" = "Diplotype1/1")) %>%
  ungroup() %>%
  separate(Diplotype, sep = "Diplotype", remove = T, into = c("Type","Diplotype")) %>%
  select(-Type) %>%
  left_join(diplotype.data %>% select(-Km), by = "Diplotype") %>%
  mutate(`Error (%)` = round(abs((Predicted-Vmax)/Vmax)*100, digits = 2),
         Vmax = round(Vmax, digits = 2),
         Condition = factor(Condition, levels =
                              c("Rich",
                                "Rich (5% CV)",
                                "Rich (10% CV)",
                                "Rich (20% CV)",
                                "Sparse (3pt)",
                                "Sparse (5% CV, 3pt)",
                                "Sparse (10% CV, 3pt)",
                                "Sparse (20% CV, 3pt)",
                                "Sparse (4pt)",
                                "Sparse (5% CV, 4pt)",
                                "Sparse (10% CV, 4pt)",
                                "Sparse (20% CV, 4pt)"))) %>%
  select(Parameter, Diplotype, Condition, Vmax, Predicted, `Error (%)`) %>%
  rename("Predicted (Vmax)" = "Predicted") %>%
  arrange(Condition)


# Km Estimates ---------------------------------
final_km.est <- covar_Km.table %>%
  pivot_longer(cols = Rich:`Sparse (20% CV, 4pt)`,
               names_to = "Condition",
               values_to = "Estimate (Eta)") %>%
  group_by(Condition) %>%
  mutate(Predicted = if_else(Diplotype != "Reference (1/1)",
                             `Estimate (Eta)` + `Estimate (Eta)`[Diplotype == "Reference (1/1)"],
                             `Estimate (Eta)`),
         Diplotype = recode(Diplotype, "Reference (1/1)" = "Diplotype1/1")) %>%
  ungroup() %>%
  separate(Diplotype, sep = "Diplotype", remove = T, into = c("Type","Diplotype")) %>%
  select(-Type) %>%
  left_join(diplotype.data %>% select(-Vmax), by = "Diplotype") %>%
  mutate(`Error (%)` = round(abs((Predicted-Km)/Km)*100, digits = 2),
         Km = round(Km, digits = 2),
         Condition = factor(Condition, levels =
                              c("Rich",
                                "Rich (5% CV)",
```

```
                                "Rich (10% CV)",
                                "Rich (20% CV)",
                                "Sparse (3pt)",
                                "Sparse (5% CV, 3pt)",
                                "Sparse (10% CV, 3pt)",
                                "Sparse (20% CV, 3pt)",
                                "Sparse (4pt)",
                                "Sparse (5% CV, 4pt)",
                                "Sparse (10% CV, 4pt)",
                                "Sparse (20% CV, 4pt)"))) %>%
select(Parameter, Diplotype, Condition, Km, Predicted, `Error (%)`) %>%
rename("Predicted (Km)" = "Predicted") %>%
arrange(Condition)
```

11 Population Michaelis-Menten Modeling (IM/PM)

11.1 Non-Linear Mixed Effect Modeling (Covariate Model - 0% CV)

```
# Full Population Simulation rich at 0% CV ------------

# Rich Sampling (16 pt, PM) ---------------
PM.data <- update_data(n_indiv = 10, `CV%` = 0, type = "PM") %>%
  filter(Diplotype %in% c("4/41", "4/5"))
PM.grp <- groupedData(V~S|ID, PM.data)
PM.nls <- nlsList(V~SSmicmen(S,Vmax,Km), data = PM.grp)
PM.nlme <- nlme(PM.nls, random = pdDiag(Vmax + Km ~ 1))
PM_covar.nlme <- update(PM.nlme,
                        fixed = Vmax + Km ~ Diplotype,
                    start = c(fixed.effects(PM.nlme)[[1]], rep(0,1),
                              fixed.effects(PM.nlme)[[2]], rep(0,1)))

# Sparse Sampling (3pt, PM) -------------
PM_3pt.data <- update_data(n_indiv = 10, `CV%` = 0, type = "PM_3pt") %>%
  filter(Diplotype %in% c("4/41", "4/5"))
PM_3pt.grp <- groupedData(V~S|ID, PM_3pt.data)
PM_3pt.nls <- nlsList(V~SSmicmen(S,Vmax,Km), data = PM_3pt.grp)
PM_3pt.nlme <- nlme(PM_3pt.nls, random = pdDiag(Vmax + Km ~ 1))
PM_3pt_covar.nlme <- update(PM_3pt.nlme,
                        fixed = Vmax + Km ~ Diplotype,
                    start = c(fixed.effects(PM_3pt.nlme)[[1]], rep(0,1),
                              fixed.effects(PM_3pt.nlme)[[2]], rep(0,1)))

# Sparse Sampling (4pt, PM) ---------------
PM_4pt.data <- update_data(n_indiv = 10, `CV%` = 0, type = "PM_4pt") %>%
  filter(Diplotype %in% c("4/41", "4/5"))
PM_4pt.grp <- groupedData(V~S|ID, PM_4pt.data)
PM_4pt.nls <- nlsList(V~SSmicmen(S,Vmax,Km), data = PM_4pt.grp)
PM_4pt.nlme <- nlme(PM_4pt.nls, random = pdDiag(Vmax + Km ~ 1))
PM_4pt_covar.nlme <- update(PM_4pt.nlme,
                        fixed = Vmax + Km ~ Diplotype,
                        start = c(fixed.effects(PM_4pt.nlme)[[1]], rep(0,1),
                        fixed.effects(PM_4pt.nlme)[[2]], rep(0,1)))
```

11.2 Non-Linear Mixed Effect Modeling (Covariate Model - 5-20% CV)

```
# 5% Residual Error --------------------------

PM_CV5.data <- update_data(n_indiv = 10, `CV%` = 5, type = "PM") %>%
  filter(Diplotype %in% c("4/41", "4/5"))
PM_CV5_data.grp <- groupedData(V~S|ID, PM_CV5.data)
PM_CV5.nls <- nlsList(V~SSmicmen(S,Vmax,Km), data = PM_CV5_data.grp)
PM_CV5.nlme <- nlme(PM_CV5.nls, random = pdDiag(Vmax + Km ~ 1))
PM_CV5_covar.nlme <- update(PM_CV5.nlme,
                        fixed = Vmax + Km ~ Diplotype,
                    start = c(fixed.effects(PM_CV5.nlme)[[1]], rep(0,1),
                              fixed.effects(PM_CV5.nlme)[[2]], rep(0,1)))
```

```
PM_3pt_CV5.data <- update_data(n_indiv = 10, `CV%` = 5, type = "PM_3pt") %>%
  filter(Diplotype %in% c("4/41", "4/5"))
PM_3pt_CV5_data.grp <- groupedData(V~S|ID, PM_3pt_CV5.data)
PM_3pt_CV5.nls <- nlsList(V~SSmicmen(S,Vmax,Km), data = PM_3pt_CV5_data.grp)
PM_3pt_CV5.nlme <- nlme(PM_3pt_CV5.nls, random = pdDiag(Vmax + Km ~ 1))
PM_3pt_CV5_covar.nlme <- update(PM_3pt_CV5.nlme,
                            fixed = Vmax + Km ~ Diplotype,
                    start = c(fixed.effects(PM_3pt_CV5.nlme)[[1]], rep(0,1),
                              fixed.effects(PM_3pt_CV5.nlme)[[2]], rep(0,1)))


PM_4pt_CV5.data <- update_data(n_indiv = 10, `CV%` = 5, type = "PM_4pt") %>%
  filter(Diplotype %in% c("4/41", "4/5"))
PM_4pt_CV5_data.grp <- groupedData(V~S|ID, PM_4pt_CV5.data)
PM_4pt_CV5.nls <- nlsList(V~SSmicmen(S,Vmax,Km), data = PM_4pt_CV5_data.grp)
PM_4pt_CV5.nlme <- nlme(PM_4pt_CV5.nls, random = pdDiag(Vmax + Km ~ 1))
PM_4pt_CV5_covar.nlme <- update(PM_4pt_CV5.nlme,
                            fixed = Vmax + Km ~ Diplotype,
                    start = c(fixed.effects(PM_4pt_CV5.nlme)[[1]], rep(0,1),
                              fixed.effects(PM_4pt_CV5.nlme)[[2]], rep(0,1)))



# 10% Residual Error --------------------

PM_CV10.data <- update_data(n_indiv = 10, `CV%` = 10, type = "PM") %>%
  filter(Diplotype %in% c("4/41", "4/5"))
PM_CV10_data.grp <- groupedData(V~S|ID, PM_CV10.data)
PM_CV10.nls <- nlsList(V~SSmicmen(S,Vmax,Km), data = PM_CV10_data.grp)
PM_CV10.nlme <- nlme(PM_CV10.nls, random = pdDiag(Vmax + Km ~ 1))
PM_CV10_covar.nlme <- update(PM_CV10.nlme,
                            fixed = Vmax + Km ~ Diplotype,
                    start = c(fixed.effects(PM_CV10.nlme)[[1]], rep(0,1),
                              fixed.effects(PM_CV10.nlme)[[2]], rep(0,1)))


PM_3pt_CV10.data <- update_data(n_indiv = 10, `CV%` = 10, type = "PM_3pt") %>%
  filter(Diplotype %in% c("4/41", "4/5"))
PM_3pt_CV10_data.grp <- groupedData(V~S|ID, PM_3pt_CV10.data)
PM_3pt_CV10.nls <- nlsList(V~SSmicmen(S,Vmax,Km), data = PM_3pt_CV10_data.grp)
PM_3pt_CV10.nlme <- nlme(PM_3pt_CV10.nls, random = pdDiag(Vmax + Km ~ 1))
PM_3pt_CV10_covar.nlme <- update(PM_3pt_CV10.nlme,
                            fixed = Vmax + Km ~ Diplotype,
                    start = c(fixed.effects(PM_3pt_CV10.nlme)[[1]], rep(0,1),
                              fixed.effects(PM_3pt_CV10.nlme)[[2]], rep(0,1)))


PM_4pt_CV10.data <- update_data(n_indiv = 10, `CV%` = 10, type = "PM_4pt") %>%
  filter(Diplotype %in% c("4/41", "4/5"))
PM_4pt_CV10_data.grp <- groupedData(V~S|ID, PM_4pt_CV10.data)
PM_4pt_CV10.nls <- nlsList(V~SSmicmen(S,Vmax,Km), data = PM_4pt_CV10_data.grp)
PM_4pt_CV10.nlme <- nlme(PM_4pt_CV10.nls, random = pdDiag(Vmax + Km ~ 1))
PM_4pt_CV10_covar.nlme <- update(PM_4pt_CV10.nlme,
```

```
                        fixed = Vmax + Km ~ Diplotype,
                    start = c(fixed.effects(PM_4pt_CV10.nlme)[[1]], rep(0,1),
                              fixed.effects(PM_4pt_CV10.nlme)[[2]], rep(0,1)))


# 20% Residual Error ----------


PM_CV20.data <- update_data(n_indiv = 10, `CV%` = 20, type = "PM") %>%
  filter(Diplotype %in% c("4/41", "4/5"))
PM_CV20_data.grp <- groupedData(V~S|ID, PM_CV20.data)
PM_CV20.nls <- nlsList(V~SSmicmen(S,Vmax,Km), data = PM_CV20_data.grp)
PM_CV20.nlme <- nlme(PM_CV20.nls, random = pdDiag(Vmax + Km ~ 1))
PM_CV20_covar.nlme <- update(PM_CV20.nlme,
                        fixed = Vmax + Km ~ Diplotype,
                    start = c(fixed.effects(PM_CV20.nlme)[[1]], rep(0,1),
                              fixed.effects(PM_CV20.nlme)[[2]], rep(0,1)))

PM_3pt_CV20.data <- update_data(n_indiv = 10, `CV%` = 20, type = "PM_3pt") %>%
  filter(Diplotype %in% c("4/41", "4/5"))
PM_3pt_CV20_data.grp <- groupedData(V~S|ID, PM_3pt_CV20.data)
PM_3pt_CV20.nls <- nlsList(V~SSmicmen(S,Vmax,Km), data = PM_3pt_CV20_data.grp)
PM_3pt_CV20.nlme <- nlme(PM_3pt_CV20.nls, random = pdDiag(Vmax + Km ~ 1))
PM_3pt_CV20_covar.nlme <- update(PM_3pt_CV20.nlme,
                        fixed = Vmax + Km ~ Diplotype,
                    start = c(fixed.effects(PM_3pt_CV20.nlme)[[1]], rep(0,1),
                              fixed.effects(PM_3pt_CV20.nlme)[[2]], rep(0,1)))


PM_4pt_CV20.data <- update_data(n_indiv = 10, `CV%` = 20, type = "PM_4pt") %>%
  filter(Diplotype %in% c("4/41", "4/5"))
PM_4pt_CV20_data.grp <- groupedData(V~S|ID, PM_4pt_CV20.data)
PM_4pt_CV20.nls <- nlsList(V~SSmicmen(S,Vmax,Km), data = PM_4pt_CV20_data.grp)
PM_4pt_CV20.nlme <- nlme(PM_4pt_CV20.nls, random = pdDiag(Vmax + Km ~ 1))
PM_4pt_CV20_covar.nlme <- update(PM_4pt_CV20.nlme,
                        fixed = Vmax + Km ~ Diplotype,
                    start = c(fixed.effects(PM_4pt_CV20.nlme)[[1]], rep(0,1),
                              fixed.effects(PM_4pt_CV20.nlme)[[2]], rep(0,1)))
```

12 Weighting Non-Linear Mixed Effect Model (IM/PM)

12.1 Evaluating Heteroscedacity of Residual Error (IM/PM)

```
# Model Comparison ---------
# 0% CV
plot(PM_covar.nlme, main = "PM Rich (16pt, 0% CV)")
plot(PM_covar.nlme %>% update(weights = varPower()),
     main = "PM Rich (16pt, 0% CV, weighted)")

plot(PM_3pt_covar.nlme, main = "PM Sparse (3pt, 0% CV)")
plot(PM_3pt_covar.nlme %>% update(weights = varPower()),
     main = "PM Sparse (3pt, 0% CV, weighted)")

plot(PM_4pt_covar.nlme, main = "PM Sparse (4pt, 0% CV)")
plot(PM_4pt_covar.nlme %>% update(weights = varPower()),
     main = "PM Sparse (4pt, 0% CV, weighted)")

# 5% CV
plot(PM_CV5_covar.nlme, main = "PM Rich (16pt, 5% CV)")
plot(PM_CV5_covar.nlme %>% update(weights = varPower()),
     main = "Rich (16pt, 5% CV, weighted)")

plot(PM_3pt_CV5_covar.nlme, main = "PM Sparse (3pt, 5% CV)")
plot(PM_3pt_CV5_covar.nlme %>% update(weights = varPower()),
     main = "PM Sparse (3pt, 5% CV, weighted)")

plot(PM_4pt_CV5_covar.nlme, main = "PM Sparse (4pt, 5% CV)")
plot(PM_4pt_CV5_covar.nlme %>% update(weights = varPower()),
     main = "PM Sparse (4pt, 5% CV, weighted)")

# 10% CV
plot(PM_CV10_covar.nlme, main = "PM Rich (16pt, 10% CV)")
plot(PM_CV10_covar.nlme %>% update(weights = varPower()),
     main = "PM Rich (16pt, 10% CV, weighted)")

plot(PM_3pt_CV10_covar.nlme, main = "PM Sparse (3pt, 10% CV)")
plot(PM_3pt_CV10_covar.nlme %>% update(weights = varPower()),
     main = "PM Sparse (3pt, 10% CV, weighted)")

plot(PM_4pt_CV10_covar.nlme, main = "PM Sparse (4pt, 10% CV)")
plot(PM_4pt_CV10_covar.nlme %>% update(weights = varPower()),
     main = "PM Sparse (4pt, 10% CV, weighted)")

# 20% CV
plot(PM_CV20_covar.nlme, main = "PM Rich (16pt, 20% CV)")
plot(PM_CV20_covar.nlme %>% update(weights = varPower()),
     main = "PM Rich (16pt, 20% CV, weighted)")

plot(PM_3pt_CV20_covar.nlme, main = "PM Sparse (3pt, 20% CV)")
plot(PM_3pt_CV20_covar.nlme %>% update(weights = varPower()),
     main = "PM Sparse (3pt, 20% CV, weighted)")

plot(PM_4pt_CV20_covar.nlme, main = "PM Sparse (4pt, 20% CV)")
```

```
plot(PM_4pt_CV20_covar.nlme %>% update(weights = varPower()),
     main = "PM Sparse (4pt, 20% CV, weighted)")
```

12.2 Adjusting PM Variance Structure

```
PM_3pt_covar.nlme <- PM_3pt_covar.nlme %>% update(weights = varPower())
PM_4pt_covar.nlme <- PM_4pt_covar.nlme %>% update(weights = varPower())

PM_CV5_covar.nlme <- PM_CV5_covar.nlme %>% update(weights = varPower())
PM_3pt_CV5_covar.nlme <- PM_3pt_CV5_covar.nlme %>% update(weights = varPower())
PM_4pt_CV5_covar.nlme <- PM_4pt_CV5_covar.nlme  %>% update(weights = varPower())

PM_CV10_covar.nlme  <- PM_CV10_covar.nlme %>% update(weights = varPower())
PM_3pt_CV10_covar.nlme <- PM_3pt_CV10_covar.nlme %>% update(weights = varPower())
PM_4pt_CV10_covar.nlme <- PM_4pt_CV10_covar.nlme %>% update(weights = varPower())

PM_CV20_covar.nlme <- PM_CV20_covar.nlme %>% update(weights = varPower())
PM_3pt_CV20_covar.nlme <- PM_3pt_CV20_covar.nlme %>% update(weights = varPower())
PM_4pt_CV20_covar.nlme <- PM_4pt_CV20_covar.nlme %>% update(weights = varPower())
```

12.3 PM Covariate Model Summary Data

```
# Summary Data Table ------------
PM_covaref_CV.table <- rbind(
      fixef(PM_covar.nlme),
      fixef(PM_3pt_covar.nlme),
      fixef(PM_4pt_covar.nlme),
      fixef(PM_CV5_covar.nlme),
      fixef(PM_3pt_CV5_covar.nlme),
      fixef(PM_4pt_CV5_covar.nlme),
      fixef(PM_CV10_covar.nlme),
      fixef(PM_3pt_CV10_covar.nlme),
      fixef(PM_4pt_CV10_covar.nlme),
      fixef(PM_CV20_covar.nlme),
      fixef(PM_3pt_CV20_covar.nlme),
      fixef(PM_4pt_CV20_covar.nlme)) %>%
  as_tibble() %>%
  mutate(`Covariate Model` = c("Rich","Sparse (3pt)","Sparse (4pt)",
                   "Rich (5% CV)","Sparse (5% CV, 3pt)","Sparse (5% CV, 4pt)",
                   "Rich (10% CV)","Sparse (10% CV, 3pt)","Sparse (10% CV, 4pt)",
                   "Rich (20% CV)","Sparse (20% CV, 3pt)","Sparse (20% CV, 4pt)")) %>%
  pivot_longer(`Vmax.(Intercept)`:`Km.Diplotype4/5`,
               names_to = "Parameter",
               values_to = "Value") %>%
  pivot_wider(names_from = `Covariate Model`, values_from = Value) %>%
  separate(Parameter, sep = "\\.", into = c("Parameter","Diplotype"), remove = T) %>%
  mutate(Diplotype = recode(Diplotype, "(Intercept)" = "Reference (4/41)"),
         across(where(is.numeric), round, 2))

# Complete Covariate Effects Table --------------------
PM_covar_Vmax.table <- PM_covaref_CV.table %>%filter(Parameter == "Vmax")
PM_covar_Km.table <- PM_covaref_CV.table %>% filter(Parameter == "Km")
```

```r
# Vmax Estimates -------------------------------
PM_final_vmax.est <- PM_covar_Vmax.table %>%
  pivot_longer(cols = Rich:`Sparse (20% CV, 4pt)`,
               names_to = "Condition",
               values_to = "Estimate (Eta)") %>%
  group_by(Condition) %>%
  mutate(Predicted = if_else(Diplotype != "Reference (4/41)",
                             `Estimate (Eta)` + `Estimate (Eta)`[Diplotype == "Reference (4/41)"],
                             `Estimate (Eta)`),
         Diplotype = recode(Diplotype, "Reference (4/41)" = "Diplotype4/41")) %>%
  ungroup() %>%
  separate(Diplotype, sep = "Diplotype", remove = T, into = c("Type","Diplotype")) %>%
  select(-Type) %>%
  left_join(diplotype.data %>% select(-Km), by = "Diplotype") %>%
  mutate(`Error (%)` = round(abs((Predicted-Vmax)/Vmax)*100, digits = 2),
         Vmax = round(Vmax, digits = 2),
         Condition = factor(Condition, levels =
                              c("Rich",
                                "Rich (5% CV)",
                                "Rich (10% CV)",
                                "Rich (20% CV)",
                                "Sparse (3pt)",
                                "Sparse (5% CV, 3pt)",
                                "Sparse (10% CV, 3pt)",
                                "Sparse (20% CV, 3pt)",
                                "Sparse (4pt)",
                                "Sparse (5% CV, 4pt)",
                                "Sparse (10% CV, 4pt)",
                                "Sparse (20% CV, 4pt)"))) %>%
  select(Parameter, Diplotype, Condition, Vmax, Predicted, `Error (%)`) %>%
  rename("Predicted (Vmax)" = "Predicted") %>%
  arrange(Condition)


# Km Estimates ---------------------------------
PM_final_km.est <- PM_covar_Km.table %>%
  pivot_longer(cols = Rich:`Sparse (20% CV, 4pt)`,
               names_to = "Condition",
               values_to = "Estimate (Eta)") %>%
  group_by(Condition) %>%
  mutate(Predicted = if_else(Diplotype != "Reference (4/41)",
                             `Estimate (Eta)` + `Estimate (Eta)`[Diplotype == "Reference (4/41)"],
                             `Estimate (Eta)`),
         Diplotype = recode(Diplotype, "Reference (4/41)" = "Diplotype4/41")) %>%
  ungroup() %>%
  separate(Diplotype, sep = "Diplotype", remove = T, into = c("Type","Diplotype")) %>%
  select(-Type) %>%
  left_join(diplotype.data %>% select(-Vmax), by = "Diplotype") %>%
  mutate(`Error (%)` = round(abs((Predicted-Km)/Km)*100, digits = 2),
         Km = round(Km, digits = 2),
         Condition = factor(Condition, levels =
                              c("Rich",
```

```
                                "Rich (5% CV)",
                                "Rich (10% CV)",
                                "Rich (20% CV)",
                                "Sparse (3pt)",
                                "Sparse (5% CV, 3pt)",
                                "Sparse (10% CV, 3pt)",
                                "Sparse (20% CV, 3pt)",
                                "Sparse (4pt)",
                                "Sparse (5% CV, 4pt)",
                                "Sparse (10% CV, 4pt)",
                                "Sparse (20% CV, 4pt)"))) %>%
select(Parameter, Diplotype, Condition, Km, Predicted, `Error (%)`) %>%
rename("Predicted (Km)" = "Predicted") %>%
arrange(Condition)
```

13 Final Model Evaluation (Qualitative Checks)

13.1 Combining Data Across All Study Design

```
all_km.est <- rbind(final_km.est, PM_final_km.est)
all_vmax.est <- rbind(final_vmax.est, PM_final_vmax.est)
```

13.2 Predicted vs Actual Figure (Vmax)

```
Vmax.plot <- ggplot(data = all_vmax.est, aes(x = Vmax, y = `Predicted (Vmax)`))+
  geom_point(size = 3, alpha = 0.85, aes(fill = Diplotype), shape = 21)+
  geom_abline(slope = 1, intercept = 0, color = "red")+
  geom_abline(slope = 1, intercept = 0.2, color = "red", linetype = "dashed")+
  geom_abline(slope = 1, intercept = -0.2, color = "red", linetype = "dashed")+
  scale_y_log10()+
  scale_x_log10()+
  facet_wrap(.~Condition, ncol = 4, scales = "free")+
  xlab("Actual (Vmax)")+
  theme_bw(base_size = 12)+
  ggeasy::easy_move_legend("top")+
  scale_fill_viridis_d()

Vmax.plot
```

13.3 Predicted vs Actual Figure (Km)

```
Km.plot <- ggplot(data = all_km.est, aes(x = Km, y = `Predicted (Km)`))+
  geom_point(size = 3, alpha = 0.85, aes(fill = Diplotype), shape = 21)+
  geom_abline(slope = 1, intercept = 0, color = "red")+
  geom_abline(slope = 1, intercept = 0.2, color = "red", linetype = "dashed")+
  geom_abline(slope = 1, intercept = -0.2, color = "red", linetype = "dashed")+
  scale_y_log10()+
  scale_x_log10()+
  facet_wrap(.~Condition, ncol = 4, scales = "free")+
  xlab("Actual (Km)")+
  theme_bw(base_size = 12)+
  ggeasy::easy_move_legend("top")+
  scale_fill_viridis_d()

Km.plot
```

14 Summary Tables for Covariate Model Estimates (w/ 95% CI)

14.1 t-Table Extractor Function

```
# Function to grab t-Table output from NLME model objects -----------

get_tTable <- function(data.nlme, condition, PM = FALSE){

  label <- if_else(PM == TRUE, "4/41", "1/1")

summary(data.nlme)$tTable %>%
  as_tibble() %>%
  mutate(Parameter = row.names(summary(data.nlme)$tTable)) %>%
  relocate(Parameter) %>%
  separate(Parameter, remove = T, sep = "\\.", into = c("Parameter", "Diplotype"))%>%
  mutate(Diplotype = recode(Diplotype, "(Intercept)" = paste0("Diplotype",label))) %>%
  group_by(Parameter) %>%
  mutate(Value = if_else(Diplotype == paste0("Diplotype",label),
                         Value,
                         Value+Value[Diplotype == paste0("Diplotype",label)])) %>%
  separate(Diplotype, remove = T, sep = "Diplotype", into = c(".", "Diplotype")) %>%
  select(Parameter, Diplotype, Value, Std.Error) %>%
  mutate(`RSE(%)` = round((Std.Error/Value)*100, digits = 2),
         Condition = condition)

}
```

14.2 Final Model Parameter Estimates and Standard Error Datasets

```
# Standard Error Tables

## EM and UM Population -------
rich_CV0_covar.tTable <- get_tTable(rich_covar.nlme, "Rich (0% CV)")
sparse3pt_CV0_covar.tTable <- get_tTable(sparse3pt_covar.nlme, "Sparse (0% CV, 3pt)")
sparse4pt_CV0_covar.tTable <- get_tTable(sparse4pt_covar.nlme, "Sparse (0% CV, 4pt)")

rich_CV5_covar.tTable <- get_tTable(rich_CV5_covar.nlme, "Rich (5% CV)")
sparse3pt_CV5_covar.tTable <- get_tTable(sparse3pt_CV5_covar.nlme, "Sparse (5% CV, 3pt)")
sparse4pt_CV5_covar.tTable <- get_tTable(sparse4pt_CV5_covar.nlme, "Sparse (5% CV, 4pt)")

rich_CV10_covar.tTable <- get_tTable(rich_CV10_covar.nlme, "Rich (10% CV)")
sparse3pt_CV10_covar.tTable <- get_tTable(sparse3pt_CV10_covar.nlme, "Sparse (10% CV, 3pt)")
sparse4pt_CV10_covar.tTable <- get_tTable(sparse4pt_CV10_covar.nlme, "Sparse (10% CV, 4pt)")

rich_CV20_covar.tTable <- get_tTable(rich_CV20_covar.nlme, "Rich (20% CV)")
sparse3pt_CV20_covar.tTable <- get_tTable(sparse3pt_CV20_covar.nlme, "Sparse (20% CV, 3pt)")
sparse4pt_CV20_covar.tTable <- get_tTable(sparse4pt_CV20_covar.nlme, "Sparse (20% CV, 4pt)")


## PM Population -------
PM_CV0_covar.tTable <- get_tTable(PM_covar.nlme, "Rich (0% CV)", T)
PM_3pt_CV0_covar.tTable <- get_tTable(PM_3pt_covar.nlme, "Sparse (0% CV, 3pt)", T)
PM_4pt_CV0_covar.tTable <- get_tTable(PM_4pt_covar.nlme, "Sparse (0% CV, 4pt)", T)
```

```
PM_CV5_covar.tTable <- get_tTable(PM_CV5_covar.nlme, "Rich (5% CV)", T)
PM_3pt_CV5_covar.tTable <- get_tTable(PM_3pt_CV5_covar.nlme, "Sparse (5% CV, 3pt)", T)
PM_4pt_CV5_covar.tTable <- get_tTable(PM_4pt_CV5_covar.nlme, "Sparse (5% CV, 4pt)", T)

PM_CV10_covar.tTable <- get_tTable(PM_CV10_covar.nlme, "Rich (10% CV)", T)
PM_3pt_CV10_covar.tTable <- get_tTable(PM_3pt_CV10_covar.nlme, "Sparse (10% CV, 3pt)", T)
PM_4pt_CV10_covar.tTable <- get_tTable(PM_4pt_CV10_covar.nlme, "Sparse (10% CV, 4pt)", T)

PM_CV20_covar.tTable <- get_tTable(PM_CV20_covar.nlme, "Rich (20% CV)", T)
PM_3pt_CV20_covar.tTable <- get_tTable(PM_3pt_CV20_covar.nlme, "Sparse (20% CV, 3pt)", T)
PM_4pt_CV20_covar.tTable <- get_tTable(PM_4pt_CV20_covar.nlme, "Sparse (20% CV, 4pt)", T)
```

14.3 Combining Datasets across Experimental Conditions

```
## Combined tTable Outpout -----------------
complete_CV_covar.tTable <- rbind(

      # EM and UM Population Estimates -------
      rich_CV5_covar.tTable,
      sparse3pt_CV5_covar.tTable,
      sparse4pt_CV5_covar.tTable,

      rich_CV10_covar.tTable,
      sparse3pt_CV10_covar.tTable,
      sparse4pt_CV10_covar.tTable,

      rich_CV20_covar.tTable,
      sparse3pt_CV20_covar.tTable,
      sparse4pt_CV20_covar.tTable,

      #PM and IM Population Estimates -----
      PM_CV5_covar.tTable,
      PM_3pt_CV5_covar.tTable,
      PM_4pt_CV5_covar.tTable,

      PM_CV10_covar.tTable,
      PM_3pt_CV10_covar.tTable,
      PM_4pt_CV10_covar.tTable,

      PM_CV20_covar.tTable,
      PM_3pt_CV20_covar.tTable,
      PM_4pt_CV20_covar.tTable)
```

14.4 Confidence Interval Data (All Conditions)

```
# Confidence Interval Table ---------

## EM and UM Population Intervals
rich_CV5_covar.intervals <- intervals(rich_CV5_covar.nlme, which = "fixed")
sparse3pt_CV5_covar.intervals <- intervals(sparse3pt_CV5_covar.nlme, which = "fixed")
sparse4pt_CV5_covar.intervals <- intervals(sparse4pt_CV5_covar.nlme, which = "fixed")
```

```
rich_CV10_covar.intervals <- intervals(rich_CV10_covar.nlme, which = "fixed")
sparse3pt_CV10_covar.intervals <- intervals(sparse3pt_CV10_covar.nlme, which = "fixed")
sparse4pt_CV10_covar.intervals <- intervals(sparse4pt_CV10_covar.nlme, which = "fixed")

rich_CV20_covar.intervals <- intervals(rich_CV20_covar.nlme, which = "fixed")
sparse3pt_CV20_covar.intervals <- intervals(sparse3pt_CV20_covar.nlme, which = "fixed")
sparse4pt_CV20_covar.intervals <- intervals(sparse4pt_CV20_covar.nlme, which = "fixed")


## PM and IM Population Intervals
PM_covar.intervals <- intervals(PM_covar.nlme, which = "fixed")
PM_3pt_covar.intervals <- intervals(PM_3pt_covar.nlme, which = "fixed")
PM_4pt_covar.intervals <- intervals(PM_4pt_covar.nlme, which = "fixed")

PM_CV5_covar.intervals <- intervals(PM_CV5_covar.nlme, which = "fixed")
PM_3pt_CV5_covar.intervals <- intervals(PM_3pt_CV5_covar.nlme, which = "fixed")
PM_4pt_CV5_covar.intervals <- intervals(PM_4pt_CV5_covar.nlme, which = "fixed")

PM_CV10_covar.intervals <- intervals(PM_CV10_covar.nlme, which = "fixed")
PM_3pt_CV10_covar.intervals <- intervals(PM_3pt_CV10_covar.nlme, which = "fixed")
PM_4pt_CV10_covar.intervals <- intervals(PM_4pt_CV10_covar.nlme, which = "fixed")

PM_CV20_covar.intervals <- intervals(PM_CV20_covar.nlme, which = "fixed")
PM_3pt_CV20_covar.intervals <- intervals(PM_3pt_CV20_covar.nlme, which = "fixed")
PM_4pt_CV20_covar.intervals <- intervals(PM_4pt_CV20_covar.nlme, which = "fixed")
```

14.5 Confidence Interval Extractor Function

```
# Function to extract and tidy confidence interval estimates ---------
extract_CI3 <- function(int.data, name, PM = FALSE){

 label <- if_else(PM == TRUE, "4/41", "1/1")

 temp <- int.data$fixed %>%
  as_tibble() %>%
  mutate(Parameter = rownames(int.data$fixed)) %>%
  separate(Parameter, remove = T, sep = "\\.", into = c("Parameter", "Diplotype"))%>%
  mutate(Diplotype = recode(Diplotype, "(Intercept)" = paste0("Diplotype",label))) %>%
  group_by(Parameter) %>%
    mutate(lower = if_else(Diplotype != paste0("Diplotype",label),
                           lower + est.[Diplotype == paste0("Diplotype",label)],
                           lower),
           upper = if_else(Diplotype != paste0("Diplotype",label),
                           upper + est.[Diplotype == paste0("Diplotype",label)],
                           upper),
           est. = if_else(Diplotype != paste0("Diplotype",label),
                          est. + est.[Diplotype == paste0("Diplotype",label)],
                          est.),
           across(where(is.numeric), round, 2),
           `CI (95%)`  = paste0("(",lower,", ",upper,")")) %>%
  ungroup() %>%
  separate(Diplotype, remove = T, sep = "Diplotype", into = c(".", "Diplotype"))%>%
```

```
  select(Parameter, Diplotype, est., `CI (95%)`)

  colnames(temp) <- c("Parameter", "Diplotype",
                      paste(name, " Estimate"),
                      paste(name, " CI (95%)"))
  temp

}
```

14.6 Generating Confidence Interval Datatables

```
# Combined Tables
CV0_covar_ci.table <- cbind(
  extract_CI3(rich_covar.intervals, name = "Rich (0% CV)"),
  rich_CV0_covar.tTable$`RSE(%)`,
  extract_CI3(sparse3pt_covar.intervals, name = "Sparse 3pt (0% CV)")[,3:4],
  sparse3pt_CV0_covar.tTable$`RSE(%)`,
  extract_CI3(sparse4pt_covar.intervals, name = "Sparse 4pt (0% CV)")[,3:4],
  sparse4pt_CV0_covar.tTable$`RSE(%)`) %>%
  left_join(actual.est, by = c("Diplotype","Parameter")) %>%
  relocate(Actual, .after = Diplotype)

CV5_covar_ci.table <- cbind(
  extract_CI3(rich_CV5_covar.intervals, name = "Rich (5% CV)"),
  rich_CV5_covar.tTable$`RSE(%)`,
  extract_CI3(sparse3pt_CV5_covar.intervals, name = "Sparse 3pt (5% CV)")[,3:4],
  sparse3pt_CV5_covar.tTable$`RSE(%)`,
  extract_CI3(sparse4pt_CV5_covar.intervals, name = "Sparse 4pt (5% CV)")[,3:4],
  sparse4pt_CV5_covar.tTable$`RSE(%)`) %>%
  left_join(actual.est, by = c("Diplotype","Parameter")) %>%
  relocate(Actual, .after = Diplotype)

CV10_covar_ci.table <- cbind(
  extract_CI3(rich_CV10_covar.intervals, name = "Rich (10% CV)"),
  rich_CV10_covar.tTable$`RSE(%)`,
  extract_CI3(sparse3pt_CV10_covar.intervals, name = "Sparse 3pt (10% CV)")[,3:4],
  sparse3pt_CV10_covar.tTable$`RSE(%)`,
  extract_CI3(sparse4pt_CV10_covar.intervals, name = "Sparse 4pt (10% CV)")[,3:4],
  sparse4pt_CV10_covar.tTable$`RSE(%)`) %>%
  left_join(actual.est, by = c("Diplotype","Parameter")) %>%
  relocate(Actual, .after = Diplotype)

CV20_covar_ci.table <- cbind(
  extract_CI3(rich_CV20_covar.intervals, name = "Rich (20% CV)"),
  rich_CV20_covar.tTable$`RSE(%)`,
  extract_CI3(sparse3pt_CV20_covar.intervals, name = "Sparse 3pt (20% CV)")[,3:4],
  sparse3pt_CV20_covar.tTable$`RSE(%)`,
  extract_CI3(sparse4pt_CV20_covar.intervals, name = "Sparse 4pt (20% CV)")[,3:4],
  sparse4pt_CV20_covar.tTable$`RSE(%)`) %>%
  left_join(actual.est, by = c("Diplotype","Parameter")) %>%
  relocate(Actual, .after = Diplotype)
```

```
PM_CV0_covar_ci.table <- cbind(
  extract_CI3(PM_covar.intervals, name = "Rich (0% CV)", T),
  PM_CV0_covar.tTable$`RSE(%)`,
  extract_CI3(PM_3pt_covar.intervals, name = "Sparse 3pt (0% CV)", T)[,3:4],
  PM_3pt_CV0_covar.tTable$`RSE(%)`,
  extract_CI3(PM_4pt_covar.intervals, name = "Sparse 4pt (0% CV)", T)[,3:4],
  PM_4pt_CV0_covar.tTable$`RSE(%)`) %>%
  left_join(actual.est, by = c("Diplotype","Parameter")) %>%
  relocate(Actual, .after = Diplotype)

PM_CV5_covar_ci.table <- cbind(
  extract_CI3(PM_CV5_covar.intervals, name = "Rich (5% CV)", T),
  PM_CV5_covar.tTable$`RSE(%)`,
  extract_CI3(PM_3pt_CV5_covar.intervals, name = "Sparse 3pt (5% CV)", T)[,3:4],
  PM_3pt_CV5_covar.tTable$`RSE(%)`,
  extract_CI3(PM_4pt_CV5_covar.intervals, name = "Sparse 4pt (5% CV)", T)[,3:4],
  PM_4pt_CV5_covar.tTable$`RSE(%)`) %>%
  left_join(actual.est, by = c("Diplotype","Parameter")) %>%
  relocate(Actual, .after = Diplotype)

PM_CV10_covar_ci.table <- cbind(
  extract_CI3(PM_CV10_covar.intervals, name = "Rich (10% CV)", T),
  PM_CV10_covar.tTable$`RSE(%)`,
  extract_CI3(PM_3pt_CV10_covar.intervals, name = "Sparse 3pt (10% CV)", T)[,3:4],
  PM_3pt_CV10_covar.tTable$`RSE(%)`,
  extract_CI3(PM_4pt_CV10_covar.intervals, name = "Sparse 4pt (10% CV)", T)[,3:4],
  PM_4pt_CV10_covar.tTable$`RSE(%)`) %>%
  left_join(actual.est, by = c("Diplotype","Parameter")) %>%
  relocate(Actual, .after = Diplotype)

PM_CV20_covar_ci.table <- cbind(
  extract_CI3(PM_CV20_covar.intervals, name = "Rich (20% CV)", T),
  PM_CV20_covar.tTable$`RSE(%)`,
  extract_CI3(PM_3pt_CV20_covar.intervals, name = "Sparse 3pt (20% CV)", T)[,3:4],
  PM_3pt_CV20_covar.tTable$`RSE(%)`,
  extract_CI3(PM_4pt_CV20_covar.intervals, name = "Sparse 4pt (20% CV)", T)[,3:4],
  PM_4pt_CV20_covar.tTable$`RSE(%)`) %>%
  left_join(actual.est, by = c("Diplotype","Parameter")) %>%
  relocate(Actual, .after = Diplotype)
```

14.6.1 Renaming Datatable Columns

```
colnames(CV0_covar_ci.table) <- c("Parameter","Diplotype","Actual",
                                  "Estimate", "CI (95%)", "RSE(%)",
                                  "Estimate", "CI (95%)", "RSE(%)",
                                  "Estimate", "CI (95%)", "RSE(%)")

colnames(CV5_covar_ci.table) <- c("Parameter","Diplotype","Actual",
                                  "Estimate", "CI (95%)", "RSE(%)",
                                  "Estimate", "CI (95%)", "RSE(%)",
                                  "Estimate", "CI (95%)", "RSE(%)")

colnames(CV10_covar_ci.table) <- c("Parameter","Diplotype","Actual",
```

```
                                  "Estimate", "CI (95%)", "RSE(%)",
                                  "Estimate", "CI (95%)", "RSE(%)",
                                  "Estimate", "CI (95%)", "RSE(%)")

colnames(CV20_covar_ci.table) <- c("Parameter","Diplotype","Actual",
                                  "Estimate", "CI (95%)", "RSE(%)",
                                  "Estimate", "CI (95%)", "RSE(%)",
                                  "Estimate", "CI (95%)", "RSE(%)")

colnames(PM_CV0_covar_ci.table) <- c("Parameter","Diplotype","Actual",
                                  "Estimate", "CI (95%)", "RSE(%)",
                                  "Estimate", "CI (95%)", "RSE(%)",
                                  "Estimate", "CI (95%)", "RSE(%)")

colnames(PM_CV5_covar_ci.table) <- c("Parameter","Diplotype","Actual",
                                  "Estimate", "CI (95%)", "RSE(%)",
                                  "Estimate", "CI (95%)", "RSE(%)",
                                  "Estimate", "CI (95%)", "RSE(%)")

colnames(PM_CV10_covar_ci.table) <- c("Parameter","Diplotype","Actual",
                                  "Estimate", "CI (95%)", "RSE(%)",
                                  "Estimate", "CI (95%)", "RSE(%)",
                                  "Estimate", "CI (95%)", "RSE(%)")

colnames(PM_CV20_covar_ci.table) <- c("Parameter","Diplotype","Actual",
                                  "Estimate", "CI (95%)", "RSE(%)",
                                  "Estimate", "CI (95%)", "RSE(%)",
                                  "Estimate", "CI (95%)", "RSE(%)")
```

14.6.2 Covariate Model (0% CV) CI Table

```
CV0_covar_ci.table %>%
  kbl(caption = "EM and UM Covariate Model Estimates (w/ 0% Residual Error & 95% CI)") %>%
  kable_classic(full_width = T, "striped")%>%
  add_header_above(c(" " = 1," " = 1, " " = 1,
                     "Rich" = 3,
                     "Sparse (3pt)" = 3,
                     "Sparse (4pt)" = 3)) %>%
  pack_rows("Vmax Estimate (Wild-Type)", 1,1) %>%
  pack_rows("Variant Estimates", 2, 7)%>%
  pack_rows("Km Estimate (Wild-Type)", 8,8) %>%
  pack_rows("Variant Estimates", 9, 14)
```

14.6.3 Covariate Model (5% CV) CI Table

```
CV5_covar_ci.table %>%
  kbl(caption = "EM and UM Covariate Model Estimates (w/ 5% Residual Error & 95% CI)") %>%
  kable_classic(full_width = T, "striped")%>%
  add_header_above(c(" " = 1," " = 1, " " = 1,
                     "Rich (5% CV)" = 3,
                     "Sparse (3pt, 5% CV)" = 3,
                     "Sparse (4pt, 5% CV)" = 3)) %>%
  pack_rows("Vmax Estimate (Wild-Type)", 1,1) %>%
```

```
  pack_rows("Variant Estimates", 2, 7)%>%
  pack_rows("Km Estimate (Wild-Type)", 8,8) %>%
  pack_rows("Variant Estimates", 9, 14)
```

14.6.4 Covariate Model (10% CV) CI Table

```
CV10_covar_ci.table %>%
  kbl(caption = "EM and UM Covariate Model Estimates (w/ 10% Residual Error & 95% CI)") %>%
  kable_classic(full_width = T, "striped")%>%
  add_header_above(c(" " = 1," " = 1, " " = 1,
                     "Rich (10% CV)" = 3,
                     "Sparse (3pt, 10% CV)" = 3,
                     "Sparse (4pt, 10% CV)" = 3)) %>%
  pack_rows("Vmax Estimate (Wild-Type)", 1,1) %>%
  pack_rows("Variant Estimates", 2, 7)%>%
  pack_rows("Km Estimate (Wild-Type)", 8,8) %>%
  pack_rows("Variant Estimates", 9, 14)
```

14.6.5 Covariate Model (20% CV) CI Table

```
CV20_covar_ci.table %>%
  kbl(caption = "EM and UM Covariate Model Estimates (w/ 20% Residual Error & 95% CI)") %>%
  kable_classic(full_width = T, "striped")%>%
  add_header_above(c(" " = 1," " = 1, " " = 1,
                     "Rich (20% CV)" = 3,
                     "Sparse (3pt, 20% CV)" = 3,
                     "Sparse (4pt, 20% CV)" = 3)) %>%
  pack_rows("Vmax Estimate (Wild-Type)", 1,1) %>%
  pack_rows("Variant Estimates", 2, 7)%>%
  pack_rows("Km Estimate (Wild-Type)", 8,8) %>%
  pack_rows("Variant Estimates", 9, 14)
```

14.6.6 PM Covariate Model (0% CV) CI Table

```
PM_CV0_covar_ci.table %>%
  kbl(caption = "IM and PM Covariate Model Estimates (w/ 0% Residual Error & 95% CI)") %>%
  kable_classic(full_width = T, "striped")%>%
  add_header_above(c(" " = 1," " = 1, " " = 1,
                     "Rich" = 3,
                     "Sparse (3pt)" = 3,
                     "Sparse (4pt)" = 3)) %>%
  pack_rows("Vmax Estimate (Variant)", 1,2) %>%
  pack_rows("Km Estimate (Variant)", 3,4)
```

14.6.7 PM Covariate Model (5% CV) CI Table

```
PM_CV5_covar_ci.table %>%
  kbl(caption = "IM and PM Covariate Model Estimates (w/ 5% Residual Error & 95% CI)") %>%
  kable_classic(full_width = T, "striped")%>%
  add_header_above(c(" " = 1," " = 1, " " = 1,
                     "Rich (5% CV)" = 3,
                     "Sparse (3pt, 5% CV)" = 3,
```

```
                  "Sparse (4pt, 5% CV)" = 3)) %>%
  pack_rows("Vmax Estimate (Variant)", 1,2) %>%
  pack_rows("Km Estimate (Variant)", 3,4)
```

14.6.8 PM Covariate Model (10% CV) CI Table

```
PM_CV10_covar_ci.table %>%
  kbl(caption = "IM and PM Covariate Model Estimates (w/ 10% Residual Error & 95% CI)") %>%
  kable_classic(full_width = T, "striped")%>%
  add_header_above(c(" " = 1," " = 1, " " = 1,
                  "Rich (10% CV)" = 3,
                  "Sparse (3pt, 10% CV)" = 3,
                  "Sparse (4pt, 10% CV)" = 3)) %>%
  pack_rows("Vmax Estimate (Variant)", 1,2) %>%
  pack_rows("Km Estimate (Variant)", 3,4)
```

14.6.9 PM Covariate Model (20% CV) CI Table

```
PM_CV20_covar_ci.table %>%
  kbl(caption = "IM and PM Covariate Model Estimates (w/ 20% Residual Error & 95% CI)") %>%
  kable_classic(full_width = T, "striped")%>%
  add_header_above(c(" " = 1," " = 1, " " = 1,
                  "Rich (20% CV)" = 3,
                  "Sparse (3pt, 20% CV)" = 3,
                  "Sparse (4pt, 20% CV)" = 3)) %>%
  pack_rows("Vmax Estimate (Variant)", 1,2) %>%
  pack_rows("Km Estimate (Variant)", 3,4)
```

15 Final Model Predictions

15.1 NLME Predictor Function

The predictor function `sim_nlme()` was scripted to generate tidy data frames of NLME predictions given the specified substrate concentration range. The inputs for this function are as followed:

- **tTable** - extracted NLME model tTable estimates (see *Final Model Parameter Estimates and Standard Error Datasets* section).
- **label** - Specifies the condition being simulated (ex. "Rich 0% CV").
- **Group** - Specifies the CYP2D6 metabolizer category (ex. "IM & PM").
- **Start** - Substrate concentration to start simulation (default = 0 uM).
- **End** - Last substrate concentration (default = 100 uM).
- **by** - Simulated concentration increment (default = 0.1)

```
sim_nlme <- function(tTable, label, Group, Start = 0, End = 100, by = 0.1){

  diplotypes_contained <- unique(tTable$Diplotype)

  # Simulate Diplotype Rates -----
  tTable %>%
  select(Parameter, Diplotype, Value) %>%
  pivot_wider(names_from =  Parameter,
              values_from =  Value) %>%
  expand_grid(S = seq(Start,End,by)) %>%
  mutate(V = (Vmax*S)/(Km+S),
         Condition = label,
         Group = Group)

}
```

15.2 Update Data Function (Extended)

`update_data_full()` is an extended version of the previous `update_data()` function; it differs by retaining Km and Vmax information in the output. It was created as an intermediate function for the revised `update_data2()` function (see below).

```
update_data_full <-function(`CV%`, type, n_indiv = 10, `CV%.D` = 25, by = 0.1,
                        start = 0, end = 100, Seed = 23457, Pop.freq = FALSE){

# Sampling Information --------------
# Full Range Set
full_set <- c(0.1, 0.5, 1, 2.5, 5, 10, 25, 50, 100)
PM_range <- c(1, 5, 10, 15, 25, 30, 50, 60, 90, 100, 250, 500, 800, 1000, 1600, 2000)

# Strategic Sampling Sets ----
sparse3pt_set1 <- c(0.1, 2.5, 50)
sparse3pt_set2 <- c(1, 10, 100)
sparse4pt_set1 <- c(0.1, 2.5, 25, 50)
sparse4pt_set2 <- c(1, 10, 50, 100)
```

```r
PM_3pt_set1 <- c(10,100,1000)
PM_3pt_set2 <- c(25,250,2000)

PM_4pt_set1 <- c(1,10,100,1000)
PM_4pt_set2 <- c(5,25,250,2000)

PM_range_set1 <- c(1,10,25,50,100,400,1000,2000)
PM_range_set2 <- c(5,15,30,60,90,200,800,1600)


# Rich Sampling Simulation ----------------
Rich <- population.sim(n = n_indiv, `CV%_D` = `CV%.D`, `CV%_V` = `CV%`, By = by,
                       Start = start, End = end, seed = Seed, pop_freq = Pop.freq) %>%
  filter(S %in% full_set) %>%
  mutate(Diplotype = as_factor(Diplotype),
         V = round(V_adj, digits = 3)) %>%
  select(ID, Diplotype, Vmax, Km, V, S)

# Sparse Sampling Simulation (3 point) ------------
Sparse3pt <- population.sim(n = n_indiv, `CV%_D` = `CV%.D`, `CV%_V` = `CV%`, By = by,
                            Start = start, End = end, seed = Seed, pop_freq = Pop.freq) %>%
  filter(S %in% full_set) %>%
  mutate(Set = if_else(as.numeric(ID) %% 2 == 0, "Set 2", "Set 1")) %>%
  filter(if_else(Set == "Set 1", S %in% sparse3pt_set1, S %in% sparse3pt_set2)) %>%
  mutate(Diplotype = as_factor(Diplotype),
         V = round(V_adj, digits = 3)) %>%
  select(ID, Diplotype, Vmax, Km, V, S)

# Sparse Sampling Simulation (4 point) --------------
Sparse4pt <- population.sim(n = n_indiv, `CV%_D` = `CV%.D`, `CV%_V` = `CV%`, By = by,
                            Start = start, End = end, seed = Seed, pop_freq = Pop.freq) %>%
  filter(S %in% full_set) %>%
  mutate(Set = if_else(as.numeric(ID) %% 2 == 0, "Set 2", "Set 1")) %>%
  filter(if_else(Set == "Set 1", S %in% sparse4pt_set1, S %in% sparse4pt_set2)) %>%
  mutate(Diplotype = as_factor(Diplotype),
         V = round(V_adj, digits = 3)) %>%
  select(ID, Diplotype, Vmax, Km, V, S)


PM <- population.sim(n = n_indiv, `CV%_D` = `CV%.D`, `CV%_V` = `CV%`, By = by,
                     Start = start, End = 2000, seed = Seed, pop_freq = Pop.freq) %>%
  filter(S %in% c(PM_range)) %>%
  mutate(Diplotype = as_factor(Diplotype),
         V = round(V_adj, digits = 3)) %>%
  select(ID, Diplotype, Vmax, Km, V, S)


PM_3pt <- population.sim(n = n_indiv, `CV%_D` = `CV%.D`, `CV%_V` = `CV%`, By = by,
                         Start = start, End = 2000, seed = Seed, pop_freq = Pop.freq) %>%
  filter(S %in% c(PM_range)) %>%
  mutate(Set = if_else(as.numeric(ID) %% 2 == 0, "Set 2", "Set 1")) %>%
  filter(if_else(Set == "Set 1", S %in% PM_3pt_set1, S %in% PM_3pt_set2)) %>%
  mutate(Diplotype = as_factor(Diplotype),
```

```
        V = round(V_adj, digits = 3)) %>%
  select(ID, Diplotype, Vmax, Km, V, S)

PM_4pt <- population.sim(n = n_indiv, `CV%_D` = `CV%.D`, `CV%_V` = `CV%`, By = by,
                         Start = start, End = 2000, seed = Seed, pop_freq = Pop.freq) %>%
  filter(S %in% c(PM_range)) %>%
  mutate(Set = if_else(as.numeric(ID) %% 2 == 0, "Set 2", "Set 1")) %>%
  filter(if_else(Set == "Set 1", S %in% PM_4pt_set1, S %in% PM_4pt_set2)) %>%
  mutate(Diplotype = as_factor(Diplotype),
         V = round(V_adj, digits = 3)) %>%
  select(ID, Diplotype, Vmax, Km, V, S)

# Output Selection ----------
switch(type,
       "rich" = Rich,
       "sparse3pt" = Sparse3pt,
       "sparse4pt" = Sparse4pt,
       "PM" = PM,
       "PM_3pt" = PM_3pt,
       "PM_4pt" = PM_4pt)

}
```

15.3 Update Data Function (Version 2)

Revised vs of `update_data()` designed to allow user to include more information (i.e., Condition, and Group) to the simulated data output.

```
update_data2 <- function(`CV%`, type, Condition, Group,
                         n_indiv = 10,
                         `CV%.D` = 25,
                         by = 0.1,
                         start = 0,
                         end = 100,
                         Seed = 23457,
                         Pop.freq = FALSE){
  update_data_full(`CV%`, type, n_indiv = n_indiv,
              `CV%.D` = `CV%.D`, by = by,
              start = start, end = end,
              Seed = Seed, Pop.freq = Pop.freq) %>%
    mutate(Condition = Condition,
           Group = Group) %>%
    filter(if_else(Group == "EM & UM",
                   !Diplotype %in% c("4/41", "4/5"),
                   Diplotype %in% c("4/41", "4/5")))
}
```

15.4 Sensitivity Analysis Data (Population Level)

```
# Population Level Simulated Data Based on Experimental Conditions ---
sensitivity.data <- rbind(sim_nlme(rich_CV0_covar.tTable,"Rich (0% CV)", "EM & UM"),
      sim_nlme(rich_CV5_covar.tTable,"Rich (5% CV)", "EM & UM"),
```

```
      sim_nlme(rich_CV10_covar.tTable,"Rich (10% CV)", "EM & UM"),
      sim_nlme(rich_CV20_covar.tTable,"Rich (20% CV)", "EM & UM"),
      sim_nlme(sparse3pt_CV0_covar.tTable,"Sparse (3pt, 0% CV)", "EM & UM"),
      sim_nlme(sparse3pt_CV5_covar.tTable,"Sparse (3pt, 5% CV)", "EM & UM"),
      sim_nlme(sparse3pt_CV10_covar.tTable,"Sparse (3pt, 10% CV)", "EM & UM"),
      sim_nlme(sparse3pt_CV20_covar.tTable,"Sparse (3pt, 20% CV)", "EM & UM"),
      sim_nlme(sparse4pt_CV0_covar.tTable,"Sparse (4pt, 0% CV)", "EM & UM"),
      sim_nlme(sparse4pt_CV5_covar.tTable,"Sparse (4pt, 5% CV)", "EM & UM"),
      sim_nlme(sparse4pt_CV10_covar.tTable,"Sparse (4pt, 10% CV)", "EM & UM"),
      sim_nlme(sparse4pt_CV20_covar.tTable,"Sparse (4pt, 20% CV)", "EM & UM"),
      sim_nlme(PM_CV0_covar.tTable, "Rich (0% CV)", "IM & PM", End = 2000),
      sim_nlme(PM_CV5_covar.tTable, "Rich (5% CV)", "IM & PM", End = 2000),
      sim_nlme(PM_CV10_covar.tTable, "Rich (10% CV)", "IM & PM", End = 2000),
      sim_nlme(PM_CV20_covar.tTable, "Rich (20% CV)", "IM & PM", End = 2000),
      sim_nlme(PM_3pt_CV0_covar.tTable, "Sparse (3pt, 0% CV)", "IM & PM", End = 2000),
      sim_nlme(PM_3pt_CV5_covar.tTable, "Sparse (3pt, 5% CV)", "IM & PM", End = 2000),
      sim_nlme(PM_3pt_CV10_covar.tTable, "Sparse (3pt, 10% CV)", "IM & PM", End = 2000),
      sim_nlme(PM_3pt_CV20_covar.tTable, "Sparse (3pt, 20% CV)", "IM & PM", End = 2000),
      sim_nlme(PM_4pt_CV0_covar.tTable, "Sparse (4pt, 0% CV)", "IM & PM", End = 2000),
      sim_nlme(PM_4pt_CV5_covar.tTable, "Sparse (4pt, 5% CV)", "IM & PM", End = 2000),
      sim_nlme(PM_4pt_CV10_covar.tTable, "Sparse (4pt, 10% CV)", "IM & PM", End = 2000),
      sim_nlme(PM_4pt_CV20_covar.tTable, "Sparse (4pt, 20% CV)", "IM & PM", End = 2000))

sensitivity.data$Condition <- factor(sensitivity.data$Condition,
                                     levels = unique(sensitivity.data$Condition))
```

15.5 Sensitivity Analysis Data (Individual Level)

```
# Individual Level Simulated Data Based on Experimental Conditions --
sensitivity.pop <- rbind(update_data2(0,"rich", "Rich (0% CV)", "EM & UM"),
      update_data2(5,"rich", "Rich (5% CV)", "EM & UM"),
      update_data2(10,"rich", "Rich (10% CV)", "EM & UM"),
      update_data2(20,"rich", "Rich (20% CV)", "EM & UM"),
      update_data2(0,"sparse3pt", "Sparse (3pt, 0% CV)", "EM & UM"),
      update_data2(5,"sparse3pt", "Sparse (3pt, 5% CV)", "EM & UM"),
      update_data2(10,"sparse3pt", "Sparse (3pt, 10% CV)", "EM & UM"),
      update_data2(20,"sparse3pt", "Sparse (3pt, 20% CV)", "EM & UM"),
      update_data2(0,"sparse4pt", "Sparse (4pt, 0% CV)", "EM & UM"),
      update_data2(5,"sparse4pt", "Sparse (4pt, 5% CV)", "EM & UM"),
      update_data2(10,"sparse4pt", "Sparse (4pt, 10% CV)", "EM & UM"),
      update_data2(20,"sparse4pt", "Sparse (4pt, 20% CV)", "EM & UM"),
      update_data2(0,"PM", "Rich (0% CV)", "IM & PM"),
      update_data2(5,"PM", "Rich (5% CV)", "IM & PM"),
      update_data2(10,"PM", "Rich (10% CV)", "IM & PM"),
      update_data2(20,"PM", "Rich (20% CV)", "IM & PM"),
      update_data2(0,"PM_3pt", "Sparse (3pt, 0% CV)", "IM & PM"),
      update_data2(5,"PM_3pt", "Sparse (3pt, 5% CV)", "IM & PM"),
      update_data2(10,"PM_3pt", "Sparse (3pt, 10% CV)", "IM & PM"),
      update_data2(20,"PM_3pt", "Sparse (3pt, 20% CV)", "IM & PM"),
      update_data2(0,"PM_4pt", "Sparse (4pt, 0% CV)", "IM & PM"),
      update_data2(5,"PM_4pt", "Sparse (4pt, 5% CV)", "IM & PM"),
      update_data2(10,"PM_4pt", "Sparse (4pt, 10% CV)", "IM & PM"),
      update_data2(20,"PM_4pt", "Sparse (4pt, 20% CV)", "IM & PM"))
```

```
sensitivity.pop$Condition <- factor(sensitivity.pop$Condition,
                                    levels = unique(sensitivity.pop$Condition))
```

16 Publication Figures

16.1 4-OH Atomoxetine Formation (EM)

```
sensitivity_mid_end.plot <- ggplot(data = sensitivity.data %>%
                                     filter(S <=100,
                                       !Diplotype %in% c("1/2x2", "4/41", "4/5")),
                                   aes(x = S, y = V, group = Diplotype))+
  geom_line(data = sensitivity.pop %>%
              filter(S<=100, !Diplotype %in% c("1/2x2", "4/41", "4/5")),
            color = "grey",
            size = 0.5,
            aes(x = S, y = V, group = ID))+

  geom_point(data = sensitivity.pop %>%
               filter(S<=100, !Diplotype %in% c("1/2x2", "4/41", "4/5")),
             aes(x = S, y = V, group = ID, color = Diplotype),
             alpha = 0.4,
             size = 2)+
  geom_line(aes(color = Diplotype), size = 1)+

  facet_wrap(~Condition, ncol = 4, nrow = 3, scales = "free")+
  theme_bw()+
  scale_color_viridis_d()+
  xlab("Atomoxetine (µM)")+
  ylab("4-OH-Atomoxetine Formation\n(pmol/min/mg protein)")+
  ggeasy::easy_move_legend(to = "top")

sensitivity_mid_end.plot+
  ggeasy::easy_add_legend_title("Genotype")
```

16.2 4-OH Atomoxetine Formation (UM)

```
# Rapid Metabolizers -------
sensitivity_high_end.plot <- ggplot(data = sensitivity.data %>%
         filter(Diplotype == "1/2x2"),
       aes(x = S, y = V, group = Diplotype))+
  geom_line(data = sensitivity.pop %>%
              filter(Diplotype == "1/2x2"),
            color = "grey",
            aes(x = S, y = V, group = ID),
            size = 1)+

    geom_point(data = sensitivity.pop %>%
                filter(Diplotype == "1/2x2"),
              aes(x = S, y = V, group = ID,color = Diplotype),
              alpha = 0.4,
              size = 2.5)+
    geom_line(aes(color = Diplotype), size = 1.65)+

  facet_wrap(Group~Condition, ncol = 4, nrow = 3, scales = "free")+
  sjPlot::theme_sjplot()+
  scale_color_viridis_d(option = "B", begin = 0)+
```

```
  xlab("Atomoxetine (µM)")+
  ylab("4-OH-Atomoxetine Formation\n(pmol/min/mg protein)")+
  ggeasy::easy_move_legend(to = "top")

sensitivity_high_end.plot +
  ggeasy::easy_add_legend_title("Genotype")
```

16.3 4-OH Atomoxetine Formation (IM/PM)

```
# Poor Metabolizers ----
sensitivity_low_end.plot <- ggplot(data = sensitivity.data %>%
         filter(Diplotype %in% c("4/41", "4/5")),
       aes(x = S, y = V, group = Diplotype))+
  geom_line(data = sensitivity.pop %>%
              filter(Diplotype %in% c("4/41", "4/5")),
            color = "grey",
            size = 1,
            aes(x = S, y = V, group = ID))+
  geom_line(aes(color = Diplotype), size = 1.65)+
  geom_point(data = sensitivity.pop %>%
               filter(Diplotype %in% c("4/41", "4/5")),
             aes(x = S, y = V, group = ID,color = Diplotype),
             alpha = 0.4,
             size = 2.5)+
  facet_wrap(Group~Condition, ncol = 4, nrow = 3, scales = "free")+
  sjPlot::theme_sjplot()+
  scale_color_viridis_d(option = "A", begin = 0.2, end = 0.5)+
  # scale_x_log10(breaks = trans_breaks("log10", function(x) 10^x),
  #       labels = trans_format("log10", math_format(10^.x)))+
  xlab("Atomoxetine (µM)")+
  ylab("4-OH-Atomoxetine Formation\n(pmol/min/mg protein)")+
  ggeasy::easy_move_legend(to = "top")

sensitivity_low_end.plot+
  ggeasy::easy_add_legend_title("Genotype")
```

16.4 CYP2D6 Intrinsic Clearance Predicted vs Actual

```
# Data frame with intrinsic clearance estimates for all conditions ----
sensitivity.CL <-  sensitivity.data %>%
  filter(S == 0) %>%
  mutate(CLint = round(Vmax/Km,digits = 2)) %>%
  select(-S, -V) %>%
  left_join(diplotype.data %>%
              mutate(CLint_actual = round(Vmax/Km, digit = 2)) %>%
              select(Diplotype, CLint_actual),by = "Diplotype") %>%
  mutate(CLint_ratio = CLint/CLint_actual)%>%
  group_by(Diplotype) %>%
  mutate(standard_resid = (CLint-CLint_actual)/sqrt(CLint_actual),
  Obs = seq_along(standard_resid),
  `Simulated Error` = if_else(Condition %in% c("Rich (0% CV)",
                                               "Sparse (3pt, 0% CV)",
```

```
                                          "Sparse (4pt, 0% CV)"),
                                "Control\n(0% CV)",
                                "Variable Error\n(5-20% CV)")) %>%
          ungroup()


CL.plot <- ggplot(data = sensitivity.CL, aes(x = CLint_actual, y = CLint))+
  geom_point(size = 3.5, alpha = 0.65, aes(fill = Condition), shape = 21)+
  geom_abline(slope = 1, intercept = 0, color = "red")+
  geom_abline(slope = 1, intercept = 0.2, color = "red", linetype = "dashed")+
  geom_abline(slope = 1, intercept = -0.2, color = "red", linetype = "dashed")+
   geom_abline(slope = 1, intercept = 0.15, color = "blue", linetype = "dashed")+
  geom_abline(slope = 1, intercept = -0.15, color = "blue", linetype = "dashed")+
  scale_y_log10()+
  scale_x_log10()+
  # facet_wrap(.~Condition, ncol = 4, scales = "free")+
  xlab("Intrinsic Clearance (Actual)")+
  ylab("Intrinsic Clearance (Predicted)")+
  theme_bw(base_size = 14)+
  ggeasy::easy_move_legend("right")+
  scale_fill_viridis_d()+
  # coord_cartesian(clip = "off")+
  facet_wrap(~Group, ncol = 1, scales = "free")

CL.plot
```

16.5 CYP2D6 Intrinsic Clearance Prediction Residual (1/2)

```
CL_plot2.1 <- ggplot(data = sensitivity.CL,
       aes(y = standard_resid, x = Diplotype))+
  geom_hline(yintercept = 0, size = 1, alpha = 0.5)+
  geom_line(aes(color = Condition, group = Condition))+
  geom_point(size = 3.5, shape = 21, alpha = 0.5, aes(fill = Condition))+
  theme_bw(base_size = 14)+
  facet_grid(`Simulated Error`~., switch = "both")+
  scale_y_continuous(limits = c(-3,3))+
  xlab("CYP2D6 Genotype")+
  ylab(expression(atop("Standardized Residuals", "("*CL[intrinsic]*")")))

CL_plot2.1
```

16.6 CYP2D6 Intrinsic Clearance Prediction Residual (2/2)

```
CL_plot2.2 <- ggplot(data = sensitivity.CL,
       aes(y = fct_reorder(Condition, standard_resid, .fun = median),
           x = standard_resid))+
    stat_boxplot(geom = "errorbar",
                 width = 0.5, size = 0.7) +
    geom_boxplot(size = 0.7)+
    geom_vline(xintercept = 0,
             color = "blue", size = 1)+
    geom_point(size = 2.5, shape = 21, aes(fill = Diplotype))+
```

```
  # stat_summary(geom = "point",
  #              fun = mean,
  #              color = "red", size = 3)+
    scale_x_continuous(limits = c(-3,3))+
    theme_bw(base_size = 14)+
    xlab(expression(atop("Standardized Residuals", "("*CL[intrinsic]*")")))+
    ylab("Experimental Condition")+
    geom_vline(xintercept = c(-2,2),
               linetype = "dashed",
               color = "darkgrey",
               size = 1)+
    ggeasy::easy_add_legend_title("CYP2D6\nGenotype")

CL.plot2.2
```

17 Diagnostic Plots (Final Model)

17.1 Residual Plots (Rich)

```
plot(rich_covar.nlme, main = "Rich (0% CV)")
plot(rich_CV5_covar.nlme, main = "Rich (5% CV)")
plot(rich_CV10_covar.nlme, main = "Rich (10% CV)")
plot(rich_CV20_covar.nlme, main = "Rich (20% CV)")
```

17.2 Residual Plots (Sparse 3pt)

```
plot(sparse3pt_covar.nlme, main = "Sparse (3pt, 0% CV)")
plot(sparse3pt_CV5_covar.nlme, main = "Sparse (3pt, 5% CV)")
plot(sparse3pt_CV10_covar.nlme, main = "Sparse (3pt, 10% CV)")
plot(sparse3pt_CV20_covar.nlme, main = "Sparse (3pt, 20% CV)")
```

17.3 Residual Plots (Sparse 4pt)

```
plot(sparse4pt_covar.nlme, main = "Sparse (4pt, 0% CV)")
plot(sparse4pt_CV5_covar.nlme, main = "Sparse (4pt, 5% CV)")
plot(sparse4pt_CV10_covar.nlme, main = "Sparse (4pt, 10% CV)")
plot(sparse4pt_CV20_covar.nlme, main = "Sparse (4pt, 20% CV)")
```

17.4 Residual Plots (PM Rich)

```
plot(PM_covar.nlme, main = "PM Rich (0% CV)")
plot(PM_CV5_covar.nlme, main = "PM Rich (5% CV)")
plot(PM_CV10_covar.nlme, main = "PM Rich (10% CV)")
plot(PM_CV20_covar.nlme, main = "PM Rich (20% CV)")
```

17.5 Residual Plots (PM Sparse 3pt)

```
plot(PM_3pt_covar.nlme, main = "PM Sparse (3pt, 0% CV)")
plot(PM_3pt_CV5_covar.nlme, main = "PM Sparse (3pt, 5% CV)")
plot(PM_3pt_CV10_covar.nlme, main = "PM Sparse (3pt, 10% CV)")
plot(PM_3pt_CV20_covar.nlme, main = "PM Spase (3pt, 20% CV)")
```

17.6 Residual Plots (PM Sparse 4pt)

```
plot(PM_4pt_covar.nlme, main = "PM Sparse (4pt, 0% CV)")
plot(PM_4pt_CV5_covar.nlme, main = "PM Sparse (4pt, 5% CV)")
plot(PM_4pt_CV10_covar.nlme, main = "PM Sparse (4pt, 10% CV)")
plot(PM_4pt_CV20_covar.nlme, main = "PM Sparse (4pt, 20% CV)")
```

17.7 Predicted vs Observed Plots by CYP2D6 Genotype (UM/EM)

```
plot(rich_covar.nlme,log(V)~log(fitted(.))|Diplotype,
     abline = c(0,1), main = "Rich (0% CV)",
     xlab = "log(Fitted values)")
```

```
plot(rich_CV5_covar.nlme, log(V)~log(fitted(.))|Diplotype,
     abline = c(0,1), main = "Rich (5% CV)",
     xlab = "log(Fitted values)")

plot(rich_CV10_covar.nlme, log(V)~log(fitted(.))|Diplotype,
     abline = c(0,1), main = "Rich (10% CV)",
     xlab = "log(Fitted values)")

plot(rich_CV20_covar.nlme, log(V)~log(fitted(.))|Diplotype,
     abline = c(0,1), main = "Rich (20% CV)",
     xlab = "log(Fitted values)")

plot(sparse3pt_covar.nlme, log(V)~log(fitted(.))|Diplotype,
     abline = c(0,1), main = "Sparse (3pt, 0% CV)",
     xlab = "log(Fitted values)")

plot(sparse3pt_CV5_covar.nlme, log(V)~log(fitted(.))|Diplotype,
     abline = c(0,1), main = "Sparse (3pt, 5% CV)",
     xlab = "log(Fitted values)")

plot(sparse3pt_CV10_covar.nlme, log(V)~log(fitted(.))|Diplotype,
     abline = c(0,1), main = "Sparse (3pt, 10% CV)",
     xlab = "log(Fitted values)")

plot(sparse3pt_CV20_covar.nlme, log(V)~log(fitted(.))|Diplotype,
     abline = c(0,1), main = "Sparse (3pt, 20% CV)",
     xlab = "log(Fitted values)")

plot(sparse4pt_covar.nlme, log(V)~log(fitted(.))|Diplotype,
     abline = c(0,1), main = "Sparse (4pt, 0% CV )",
     xlab = "log(Fitted values)")

plot(sparse4pt_CV5_covar.nlme, log(V)~log(fitted(.))|Diplotype,
     abline = c(0,1), main = "Sparse (4pt, 5% CV)",
     xlab = "log(Fitted values)")

plot(sparse4pt_CV10_covar.nlme, log(V)~log(fitted(.))|Diplotype,
     abline = c(0,1), main = "Sparse (4pt, 10% CV)",
     xlab = "log(Fitted values)")

plot(sparse4pt_CV20_covar.nlme, log(V)~log(fitted(.))|Diplotype,
     abline = c(0,1), main = "Sparse (4pt, 20% CV)",
     xlab = "log(Fitted values)")
```

17.8 Predicted vs Observed Plots by CYP2D6 Genotype (IM/PM)

```
plot(PM_covar.nlme, log(V)~log(fitted(.))|Diplotype,
     abline = c(0,1), xlab = "log(Fitted values)",
     main = "PM Rich (0% CV)")

plot(PM_CV5_covar.nlme, log(V)~log(fitted(.))|Diplotype,
     abline = c(0,1), xlab = "log(Fitted values)",
     main = "PM Rich (5% CV)")
```

```
plot(PM_CV10_covar.nlme, log(V)~log(fitted(.))|Diplotype,
     abline = c(0,1), xlab = "log(Fitted values)",
     main = "PM Rich (10% CV)")

plot(PM_CV20_covar.nlme, log(V)~log(fitted(.))|Diplotype,
     abline = c(0,1), xlab = "log(Fitted values)",
     main = "PM Rich (20% CV)")

plot(PM_3pt_covar.nlme, log(V)~log(fitted(.))|Diplotype,
     abline = c(0,1), xlab = "log(Fitted values)",
     main = "PM Sparse (3pt, 0% CV)")

plot(PM_3pt_CV5_covar.nlme, log(V)~log(fitted(.))|Diplotype,
     abline = c(0,1), xlab = "log(Fitted values)",
     main = "PM Sparse (3pt, 5% CV)")

plot(PM_3pt_CV10_covar.nlme, log(V)~log(fitted(.))|Diplotype,
     abline = c(0,1), xlab = "log(Fitted values)",
     main = "PM Sparse (3pt 10% CV)")

plot(PM_3pt_CV20_covar.nlme, log(V)~log(fitted(.))|Diplotype,
     abline = c(0,1), xlab = "log(Fitted values)",
     main = "PM Sparse (3pt, 20% CV)")

plot(PM_4pt_covar.nlme, log(V)~log(fitted(.))|Diplotype,
     abline = c(0,1), xlab = "log(Fitted values)",
     main = "PM Sparse (4pt, 0% CV)")

plot(PM_4pt_CV5_covar.nlme, log(V)~log(fitted(.))|Diplotype,
     abline = c(0,1), xlab = "log(Fitted values)",
     main = "PM Sparse (4pt, 5% CV)")

plot(PM_4pt_CV10_covar.nlme, log(V)~log(fitted(.))|Diplotype,
     abline = c(0,1), xlab = "log(Fitted values)",
     main = "PM Sparse (4pt, 10% CV)")

plot(PM_4pt_CV20_covar.nlme, log(V)~log(fitted(.))|Diplotype,
     abline = c(0,1), xlab = "log(Fitted values)",
     main = "PM Sparse (4pt, 20% CV)")
```

18 Bootstrap Analysis

18.1 Virtual Population Function

An updated version of a previous function `population.sim()` was generated called `population.sim2()`. The updated function provides greater flexibility substrate concentration range below 0.50 uM.

```
# ========== POPULATION SIMULATION FUNCTION =================#
## Updated to include low end point of 0.10 uM

population.sim2 <- function(

  # Pre-define number of subjects per diplotype group
  n = 10,
  `CV%_D` = 25,
  `CV%_V` = 0,

  # Pre-define Michaelis Menten Kinetic Settings
  Start = 0.5,
  End = 2000,
  By = 0.5,

  # Turn off or on use of true population frequency
  ## If set to true "n" will equal total population number
  pop_freq = FALSE,

  # Pre-define reference data, and set seed
  data = diplotype.data,
  seed = 23457){

  # ======== Create Data Containers ========

  datasets <- list()
  pop.data <- tibble()

  # ======== Create Datasets by Diplotype ========

  for (i in seq_along(data[[1]])){

    # Setting Population Specific Frequency
    n_pop <- if_else(pop_freq == FALSE, n,
                     ifelse(pop_freq == TRUE,
                            round(n*data[[i, c("Freq")]]), 0))

    # Specify Diplotype for current loop
    Diplotype = rep(data[[i,1]], n_pop)

    # Random assignment of Vmax values N(0,sigma)
    set.seed(seed)
    Vmax_eta <- rnorm(n_pop, sd = `CV%_D`)
    Vmax = data[[i,2]] + (Vmax_eta/100)*data[[i,2]]

    # Random assignment of Km values N(0,sigma)
    set.seed(seed)
    Km_eta <- rnorm(n_pop, sd = `CV%_D`)
```

```
    Km = data[[i,3]] + (Km_eta/100)*data[[i,3]]

    # Store each diplotype group in a list container
    temp <- tibble(Diplotype, Vmax, Km)
    datasets[[i]] <- temp

  }

  # ======== Combine Diplotype Datasets ========

  for (i in seq_along(data[[1]])){

    pop.data <- rbind(pop.data, datasets[[i]])

  }

  # ======== Simulate Michaelis Menten ========

  set.seed(seed)

  pop.data <- pop.data %>%
    mutate(ID = factor(seq_along(Diplotype),
                       levels = seq_along(Diplotype))) %>%
    relocate(ID) %>%
    expand_grid(S = c(0.1, seq(Start, End, By))) %>%
    mutate(V = round((Vmax*S)/(Km+S), digits = 2))%>%
    group_by(ID) %>%

    # Additon of Residual Error
    mutate(resid_pct = round(rnorm(n = length(S),sd = `CV%_V`),
                             digits = 3),
           V_adj = V+(V*resid_pct/100)) %>%
    ungroup()


  return(pop.data)

}
```

18.2 Generating Virtual Population (with Residual Error)

Michaelis-Menten kinetics were simulated for each subject in the virtual population across the 4 experimental designs (0%, 5%, 10%, and 20% CV).

```
# ========== CREATE BOOTSTRAP VIRTUAL POPULATIONS ============#

# Virtual Populations
CV0_virtual.pop <- population.sim2(n = 1000, `CV%_V` = 0, End = 2000)
CV5_virtual.pop <- population.sim2(n = 1000, `CV%_V` = 5, End = 2000)
CV10_virtual.pop <- population.sim2(n = 1000, `CV%_V` = 10, End = 2000)
CV20_virtual.pop <- population.sim2(n = 1000, `CV%_V` = 20, End = 2000)


# Sample Population Containers
```

```
CV0_Sample_populations = list()
CV5_Sample_populations = list()
CV10_Sample_populations = list()
CV20_Sample_populations = list()
```

18.3 Bootstrap Sampling of Virtual Population

Using a `for()` loop, a total of 10 bootstrapped populations were generated for each experiental design (0-20% CV) by randomly sampling (with replacement) 1% of the corresponding virtual population (10 subject per genotype group, n = 90).

```
## Bootstrap for-loop
B = 10 ## number of bootstraps

for (i in 1:B) {

  # Bootstrap for 0% CV Population ---------
  CV0_Sample_populations[[i]] <- CV0_virtual.pop %>%
    group_by(Diplotype) %>%
    filter(ID %in% sample(ID, 10, replace = T)) %>%
    ungroup()

  # Bootstrap for 5% CV Population ---------
  CV5_Sample_populations[[i]] <- CV5_virtual.pop %>%
    group_by(Diplotype) %>%
    filter(ID %in% sample(ID, 10, replace = T)) %>%
    ungroup()

  # Bootstrap for 10% CV Population ---------
  CV10_Sample_populations[[i]] <- CV10_virtual.pop %>%
    group_by(Diplotype) %>%
    filter(ID %in% sample(ID, 10, replace = T)) %>%
    ungroup()

  # Bootstrap for 20% CV Population ---------
  CV20_Sample_populations[[i]] <- CV20_virtual.pop %>%
    group_by(Diplotype) %>%
    filter(ID %in% sample(ID, 10, replace = T)) %>%
    ungroup()

  }

# Combining Population Data -----------
Bootstrap.populations <- list(CV0_Sample_populations,
                              CV5_Sample_populations,
                              CV10_Sample_populations,
                              CV20_Sample_populations)

names(Bootstrap.populations)<- c("CV0_Pops",
                                 "CV5_Pops",
                                 "CV10_Pops",
                                 "CV20_Pops")

# saveRDS(Bootstrap.populations, "R Output/Bootsrap Populations Data.rds")
```

18.4 Sampling Function

Strategic sampling designs were implemented using a custom function `update_pop()`, which functions similarly to the `update_data()` function mentioned previously, however, is designed to work more efficiently the `Bootstrap.populations` data which is a list of dataframes). The inputs for the `update_pop()` function include the following:

- **pop_data** - A dataframe or indexed dataframe from a list of dataframes.
- **type** - Specifies the strategic sampling design to be implemented. Options include: *"rich", "sparse3pt", "sparse4pt", "PM", "PM_3pt", "PM_4pt".*

```
#======== CREATE BOOTSTRAP EXPERIMENTAL DATASETS =========#

# Function to transform virtual population into sample population

update_pop <-function(pop_data, type){

  # Sampling Information --------------
  # Full Range Set
  full_set <- c(0.1, 0.5, 1, 2.5, 5, 10, 25, 50, 100)
  PM_range <- c(1, 5, 10, 15, 25, 30, 50, 60, 90, 100, 250, 500, 800, 1000, 1600, 2000)

  # Strategic Sampling Sets ----
  sparse3pt_set1 <- c(0.1, 2.5, 50)
  sparse3pt_set2 <- c(1, 10, 100)
  sparse4pt_set1 <- c(0.1, 2.5, 25, 50)
  sparse4pt_set2 <- c(1, 10, 50, 100)


  PM_3pt_set1 <- c(10,100,1000)
  PM_3pt_set2 <- c(25,250,2000)

  PM_4pt_set1 <- c(1,10,100,1000)
  PM_4pt_set2 <- c(5,25,250,2000)

  PM_range_set1 <- c(1,10,25,50,100,400,1000,2000)
  PM_range_set2 <- c(5,15,30,60,90,200,800,1600)

  ID_key <- pop_data %>%
    select(Diplotype, ID) %>%
    unique() %>%
    group_by(Diplotype) %>%
    mutate(n = 1:n()) %>%
    ungroup()

  PM_Diplotype <- ID_key %>%
    select(Diplotype) %>%
    filter(Diplotype %in% c("4/5","4/41")) %>%
    unique()

  EM_Diplotype <- ID_key %>%
    select(Diplotype) %>%
    filter(!Diplotype %in% c("4/5","4/41")) %>%
```

```
    unique()

  # Rich Sampling Simulation ----------------
  Data <- pop_data %>%

    # Filter S Based on Experimental Model Type
    filter(S %in% switch(type,
                         "rich" = full_set,
                         "sparse3pt" = full_set,
                         "sparse4pt" = full_set,
                         "PM" = PM_range,
                         "PM_3pt" = PM_range,
                         "PM_4pt" = PM_range)) %>%

    #Assign Sampling Key
    left_join(ID_key, by = c("Diplotype", "ID")) %>%

    # Strategic Sampling
    mutate(Set = if_else(n %% 2 == 0, "Set 2", "Set 1")) %>%
    filter(if_else(Set == "Set 1",
                   S %in% switch(type,
                                 "rich" = full_set,
                                 "sparse3pt" = sparse3pt_set1,
                                 "sparse4pt" = sparse4pt_set1,
                                 "PM" = PM_range_set1,
                                 "PM_3pt" = PM_3pt_set1,
                                 "PM_4pt" = PM_4pt_set1),

                   S %in% switch(type,
                                 "rich" = full_set,
                                 "sparse3pt" = sparse3pt_set2,
                                 "sparse4pt" = sparse4pt_set2,
                                 "PM" = PM_range_set2,
                                 "PM_3pt" = PM_3pt_set2,
                                 "PM_4pt" = PM_4pt_set2))) %>%
    mutate(Diplotype = as_factor(Diplotype),
           V = round(V_adj, digits = 3)) %>%
    select(ID, Diplotype, S, V, Set) %>%

    # Refine Inidvidual in dataset according to model
    filter(Diplotype %in% switch(type,
                         "rich" = EM_Diplotype[[1]],
                         "sparse3pt" = EM_Diplotype[[1]],
                         "sparse4pt" = EM_Diplotype[[1]],
                         "PM" = PM_Diplotype[[1]],
                         "PM_3pt" = PM_Diplotype[[1]],
                         "PM_4pt" = PM_Diplotype[[1]]))

  # Data Output ----------
  Data

}
```

18.5 In-Silico Experiments (Rich and Strategic Sampling)

Using a `for()` loop, each virtual population was subject to all 3 sampling strategies (rich, sparse 3pt, and sparse 4pt); with sampling range varying according to CYP2D6 genotype as described previously (see *Strategic Sampling Approach*). Data was combined as a list of a list of dataframes with the first level (n = 4) being %CV specific data, the second level (n = 3) being sampling strategy with each variable at this level containing data for 10 unique population (n = 120 total datasets).

```
# Create Experimental Data sets Populations ---------

CV0_Boot.data <- list()
CV5_Boot.data <- list()
CV10_Boot.data <- list()
CV20_Boot.data <- list()

# For Loop to apply strategic sampling to all dataframes ----------
for (i in 1:B) {

  CV0_Boot.data$rich[[i]] <- update_pop(Bootstrap.populations$CV0_Pops[[i]], type = "rich")
  CV0_Boot.data$sparse3pt[[i]] <- update_pop(Bootstrap.populations$CV0_Pops[[i]], type = "sparse3pt")
  CV0_Boot.data$sparse4pt[[i]] <- update_pop(Bootstrap.populations$CV0_Pops[[i]], type = "sparse4pt")
  CV0_Boot.data$PM[[i]] <- update_pop(Bootstrap.populations$CV0_Pops[[i]], type = "PM")
  CV0_Boot.data$PM_3pt[[i]] <- update_pop(Bootstrap.populations$CV0_Pops[[i]], type = "PM_3pt")
  CV0_Boot.data$PM_4pt[[i]] <- update_pop(Bootstrap.populations$CV0_Pops[[i]], type = "PM_4pt")

  cat('CV0 Iteration', i, "of", B, "complete...\n")

  CV5_Boot.data$rich[[i]] <- update_pop(Bootstrap.populations$CV5_Pops[[i]], type = "rich")
  CV5_Boot.data$sparse3pt[[i]] <- update_pop(Bootstrap.populations$CV5_Pops[[i]], type = "sparse3pt")
  CV5_Boot.data$sparse4pt[[i]] <- update_pop(Bootstrap.populations$CV5_Pops[[i]], type = "sparse4pt")
  CV5_Boot.data$PM[[i]] <- update_pop(Bootstrap.populations$CV5_Pops[[i]], type = "PM")
  CV5_Boot.data$PM_3pt[[i]] <- update_pop(Bootstrap.populations$CV5_Pops[[i]], type = "PM_3pt")
  CV5_Boot.data$PM_4pt[[i]] <- update_pop(Bootstrap.populations$CV5_Pops[[i]], type = "PM_4pt")

  cat('CV5 Iteration', i, "of", B, "complete...\n")

  CV10_Boot.data$rich[[i]] <- update_pop(Bootstrap.populations$CV10_Pops[[i]], type = "rich")
  CV10_Boot.data$sparse3pt[[i]] <- update_pop(Bootstrap.populations$CV10_Pops[[i]], type = "sparse3pt")
  CV10_Boot.data$sparse4pt[[i]] <- update_pop(Bootstrap.populations$CV10_Pops[[i]], type = "sparse4pt")
  CV10_Boot.data$PM[[i]] <- update_pop(Bootstrap.populations$CV10_Pops[[i]], type = "PM")
  CV10_Boot.data$PM_3pt[[i]] <- update_pop(Bootstrap.populations$CV10_Pops[[i]], type = "PM_3pt")
  CV10_Boot.data$PM_4pt[[i]] <- update_pop(Bootstrap.populations$CV10_Pops[[i]], type = "PM_4pt")

  cat('CV10 Iteration', i, "of", B, "complete...\n")

  CV20_Boot.data$rich[[i]] <- update_pop(Bootstrap.populations$CV20_Pops[[i]], type = "rich")
  CV20_Boot.data$sparse3pt[[i]] <- update_pop(Bootstrap.populations$CV20_Pops[[i]], type = "sparse3pt")
  CV20_Boot.data$sparse4pt[[i]] <- update_pop(Bootstrap.populations$CV20_Pops[[i]], type = "sparse4pt")
  CV20_Boot.data$PM[[i]] <- update_pop(Bootstrap.populations$CV20_Pops[[i]], type = "PM")
  CV20_Boot.data$PM_3pt[[i]] <- update_pop(Bootstrap.populations$CV20_Pops[[i]], type = "PM_3pt")
  CV20_Boot.data$PM_4pt[[i]] <- update_pop(Bootstrap.populations$CV20_Pops[[i]], type = "PM_4pt")

  cat('CV20 Iteration', i, "of", B, "complete...\n")
```

```
}

# Combine all boostrap data into a single data structure ---------
Complete_Bootstrap.data <- list(CV0_Boot.data,
                           CV5_Boot.data,
                           CV10_Boot.data,
                           CV20_Boot.data)

names(Complete_Bootstrap.data) <-c("CV0","CV5","CV10","CV20")

# Save the completed bootstrap data sets ------
# saveRDS(Complete_Bootstrap.data, "R Output/Complete Bootstrap Data.rds")
```

18.6 NLME Fitting Function

`fit_nlme()` is a custom function used to simultaneously fit non-linear least squares, and non-linear mixed effect models (with and without covariates). The inputs for the function include the following:

- **data** - Data to be used for model fitting
- **type** - Type of CYP2D6 metabolizer. Options include: *"EM"* for extensive metabolizer and ultra-rapid metabolizer data, and *"PM"* for intermediate and poor metabolizer data (default = "EM").
- **which** - Specifies the desired model output. Options include: *"nls"* = non-linear least squares, *"base.model"* for NLME with out covariates, *"covar.model"* for NLME with covariates (CYP2D6 genotype); default = "covar.model".

```
# ==== Fitting Mixed Effect Model ================#

fit_nlme <- function(data, type = "EM", which = "covar.model"){

n <- if_else(type == "PM", 1, 6)

# Initial Screen ----
data0.grp <- groupedData(V~S|ID, data)
fit0.nls <- nlsList(V~SSmicmen(S,Vmax,Km), data = data0.grp)

# Extract Erroneous Subjects
error.vmax <- coef(fit0.nls) %>%
  filter(is.na(Vmax)) %>%
  row.names() %>%
  as.numeric()

error.km <- coef(fit0.nls) %>%
  filter(is.na(Km)) %>%
  row.names() %>%
  as.numeric()

error.list <- c(error.vmax, error.km)

# Clean Data set
data.grp <- data0.grp %>% filter(!ID %in% error.list)
```

```
# Base Model Fits
fit.nls <- nlsList(V~SSmicmen(S,Vmax,Km), data = data.grp)

base.nlme <- nlme(fit.nls, random = pdDiag(Vmax + Km ~ 1))
covar.nlme <- update(base.nlme, fixed = Vmax + Km ~ Diplotype,
                           start = c(fixef(base.nlme)[[1]], rep(0,n),
                                     fixef(base.nlme)[[2]], rep(0,n)),
                     weights = varPower())

# Output ----------
switch (which,
  "nls" = fit.nls,
  "base.model" = base.nlme,
  "covar.model" = covar.nlme)

}
```

18.7 NLME Estimate Extraction Function

`extract_nlme_est()` is a custom function used to fit, extract, and tidy NLME parameter estimates from experimental data. This function is optimized to work within the context of a `for()` loop, and contains the `fit_nlme()` function as a dependency. The inputs for the function include the following:

- **exp.data** - Experimental data that NLME estimates are desired from.
- **condition** - Label used to tag experimental condition from which the experimental data belongs.
- **PM** - TRUE or FALSE argument specifying where data belongs to IM/PM CYP2D6 genotype (TRUE), or UM/EM CYP2D6 genotype (FALSE); default = "FALSE".
- **pop.num** - Label specifying which bootstrapped population is being analyzed (default = 1).

```
extract_nlme_est <- function(exp.data, condition, PM = FALSE, pop.num = 1){

  # Input ----------
  model.type <- if_else(PM == TRUE, "PM", "EM")
  nlme.model <- fit_nlme(exp.data, type = model.type)
  int.data <- intervals(nlme.model, which = "fixed")
  label <- if_else(PM == TRUE, "4/41", "1/1")

  # Tidy Data -------
  temp <- int.data$fixed %>%
    as_tibble() %>%
    mutate(Parameter = rownames(int.data$fixed)) %>%
    separate(Parameter, remove = T, sep = "\\.", into = c("Parameter", "Diplotype"))%>%
    mutate(Diplotype = recode(Diplotype, "(Intercept)" = paste0("Diplotype",label))) %>%
    group_by(Parameter) %>%
    mutate(est. = if_else(Diplotype != paste0("Diplotype",label),
                          est. + est.[Diplotype == paste0("Diplotype",label)],
                          est.),
           across(where(is.numeric), round, 2)) %>%
    ungroup() %>%
    separate(Diplotype, remove = T, sep = "Diplotype", into = c(".", "Diplotype"))%>%
    select(Parameter, Diplotype, est.)
```

```
  colnames(temp) <- c("Parameter", "Diplotype", paste(condition))

  # Output ----
  temp %>%
    pivot_longer( 3, names_to = "Condition", values_to = "Estimate") %>%
    mutate(Population = pop.num)

}


extract_nlme_est(exp.data = Complete_Bootstrap.data$CV0$sparse3pt[[3]],
                 condition = "Sparse (3pt, CV 5%)", PM = FALSE, pop.num = 3)
```

18.8 Non-Linear Mixed Effect Modeling (Bootstrap Populations)

Using a `for()` loop, bootstrapped parameter estimates were extracted from experimental data (`Complete_Bootstrap.data`) using the `extract_nlme.est()` function. Parameter estimate for all experimental designs and conditions for each population were stored in object `Complete_Bootstrap.est` as a list. A collapsed version was saved as `Complete Bootstrap table.RDS` for data analysis and user reproducibility purposes.

```
Complete_Bootstrap.est <- list()

for (i in 1:B) {


  # CV 0% Condition -------------
  CV0_est.table <- rbind(
  extract_nlme_est(exp.data = Complete_Bootstrap.data$CV0$rich[[i]],
                   condition = "Rich (CV 0%)", pop.num = i),
  extract_nlme_est(exp.data = Complete_Bootstrap.data$CV0$sparse3pt[[i]],
                   condition = "Sparse (3pt, CV 0%)", pop.num = i),
  extract_nlme_est(exp.data = Complete_Bootstrap.data$CV0$sparse4pt[[i]],
                   condition = "Sparse (4pt, CV 0%)", pop.num = i),
  extract_nlme_est(exp.data = Complete_Bootstrap.data$CV0$PM[[i]],
                   condition = "Rich (CV 0%)", PM = TRUE, pop.num = i),
  extract_nlme_est(exp.data = Complete_Bootstrap.data$CV0$PM_3pt[[i]],
                   condition = "Sparse (3pt, CV 0%)", PM = TRUE, pop.num = i),
  extract_nlme_est(exp.data = Complete_Bootstrap.data$CV0$PM_4pt[[i]],
                   condition = "Sparse (4pt, CV 0%)", PM = TRUE, pop.num = i))

  cat('CV0% - Iteration', i, "of", B, "complete...\n")

  # # # CV 5% Condition -------------
  CV5_est.table <- rbind(
  extract_nlme_est(exp.data = Complete_Bootstrap.data$CV5$rich[[i]],
                   condition = "Rich (CV 5%)", pop.num = i),
  extract_nlme_est(exp.data = Complete_Bootstrap.data$CV5$sparse3pt[[i]],
                   condition = "Sparse (3pt, CV 5%)", pop.num = i),
  extract_nlme_est(exp.data = Complete_Bootstrap.data$CV5$sparse4pt[[i]],
                   condition = "Sparse (4pt, CV 5%)", pop.num = i),
  extract_nlme_est(exp.data = Complete_Bootstrap.data$CV5$PM[[i]],
                   condition = "Rich (CV 5%)", PM = TRUE, pop.num = i),
```

```
  extract_nlme_est(exp.data = Complete_Bootstrap.data$CV5$PM_3pt[[i]],
                   condition = "Sparse (3pt, CV 5%)", PM = TRUE, pop.num = i),
  extract_nlme_est(exp.data = Complete_Bootstrap.data$CV5$PM_4pt[[i]],
                   condition = "Sparse (4pt, CV 5%)", PM = TRUE, pop.num = i))

  cat('CV5% - Iteration', i, "of", B, "complete...\n")

  # # CV 10% Condition -------------
  CV10_est.table <- rbind(
  extract_nlme_est(exp.data = Complete_Bootstrap.data$CV10$rich[[i]],
                   condition = "Rich (CV 10%)", pop.num = i),
  extract_nlme_est(exp.data = Complete_Bootstrap.data$CV10$sparse3pt[[i]],
                   condition = "Sparse (3pt, CV 10%)", pop.num = i),
  extract_nlme_est(exp.data = Complete_Bootstrap.data$CV10$sparse4pt[[i]],
                   condition = "Sparse (4pt, CV 10%)", pop.num = i),
  extract_nlme_est(exp.data = Complete_Bootstrap.data$CV10$PM[[i]],
                   condition = "Rich (CV 10%)", PM = TRUE, pop.num = i),
  extract_nlme_est(exp.data = Complete_Bootstrap.data$CV10$PM_3pt[[i]],
                   condition = "Sparse (3pt, CV 10%)", PM = TRUE, pop.num = i),
  extract_nlme_est(exp.data = Complete_Bootstrap.data$CV10$PM_4pt[[i]],
                   condition = "Sparse (4pt, CV 10%)", PM = TRUE, pop.num = i))

  cat('CV10% - Iteration', i, "of", B, "complete...\n")

  # CV 20% Condition -------------
# Problematic fails to converge, requires further attention ##

CV20_est.table <- rbind(
extract_nlme_est(exp.data = Complete_Bootstrap.data$CV20$rich[[i]],
                 condition = "Rich (CV 20%)", pop.num = i),
extract_nlme_est(exp.data = Complete_Bootstrap.data$CV20$sparse3pt[[i]],
                 condition = "Sparse (3pt, CV 20%)", pop.num = i),
extract_nlme_est(exp.data = Complete_Bootstrap.data$CV20$sparse4pt[[i]],
                 condition = "Sparse (4pt, CV 20%)", pop.num = i),
extract_nlme_est(exp.data = Complete_Bootstrap.data$CV20$PM[[i]],
                 condition = "Rich (CV 20%)", PM = TRUE, pop.num = i),
extract_nlme_est(exp.data = Complete_Bootstrap.data$CV20$PM_3pt[[i]],
                 condition = "Sparse (3pt, CV 20%)", PM = TRUE, pop.num = i),
extract_nlme_est(exp.data = Complete_Bootstrap.data$CV20$PM_4pt[[i]],
                 condition = "Sparse (4pt, CV 20%)", PM = TRUE, pop.num = i))

cat('CV20% - Iteration', i, "of", B, "complete...\n")

  Complete_Bootstrap.est[[i]] <- rbind(CV0_est.table,
                                       CV5_est.table,
                                       CV10_est.table,
                                       CV20_est.table)

  # Progress Report -------
  cat('==== ROUND', i, "of", B, "COMPLETE! ===\n")

  }
```

```r
Complete_Bootstrap.table <- bind_rows(Complete_Bootstrap.est)

# saveRDS(R Output/Complete Bootstrap Table.rds")
```

19 Model Evaluation (Bootstrap Analysis)

Non-parametric bootstrap analysis based on a total of 120 generated data sets (10 re-sampled populations per experimental condition) was performed to evaluate the precision of the final model parameters (see separate *Bootstrap Analysis* section above). Mean parameter estimates and 95% confidence intervals for each condition were summarized by genotype. Confidence intervals were calculated using the a custom confidence interval function `CI()` detailed below based on (Equation 6); where confidence interval (CI) is equal to the mean value of the sample population plus or minus the t-score at significance level(0.05) for N-1 degrees of freedom multiplied by the standard error of the sample population mean. The inputs for the `CI()` function are as followed:

- **x** - A vector of values for which a confidence interval is to be calculated.
- **alpha** - Alpha level for confidence interval (default = 0.05)
- **which** - Specifies the desired output. Options: confidence interval rang = **"ci"**, lower interval estimate = **"lower"**, upper interval estimate = **"upper"**, mean estimate = **"est"**, all estimates (mean [lower - upper]) = **"all"**.

19.1 Confidence Interval (95%) Function

```
# Import Bootstrap Data ----------
Bootstrap.table <- read_rds("R Output/Complete Bootstrap Table.rds")

# function to calculate confidence interval of bootstrap estimates -----
CI <- function(x, alpha = 0.05, which = "ci"){

  mean <- mean(x)
  n <- length(x)
  std_dev <- sd(x)
  std_error <- std_dev/sqrt(n)
  degrees_of_freedom = n - 1
  t_score <- qt(p=alpha/2, df=degrees_of_freedom, lower.tail=F)
  margin_error <- t_score * std_error
  lower <- mean - margin_error
  upper <- mean + margin_error
  ci <- paste0("(", round(lower, 2),
               " - ",round(upper, 2),")")
  all <- paste0(round(mean, 2),
                "  (", round(lower, 2),
                " - ",round(upper, 2),")")
  # Output ------
  switch (which,
    "ci" = ci,
    "lower" = lower,
    "upper" = upper,
    "est" = mean,
    "all" = all)
}
```

19.2 Bootstrap Analysis Summary Datatable

```
Summary.boot <- Bootstrap.table %>%
  mutate(Index = 1:length(Population)) %>%
  filter(Index != 315) %>% # un-physiologic scenario
```

```
  group_by(Parameter, Diplotype, Condition) %>%
  summarize(Est = round(mean(Estimate), 2),
            sd = round(sd(Estimate), 2),
            Lower = CI(Estimate, which = "lower"),
            Upper = CI(Estimate, which = "upper"),
            `CI (95%)` = CI(Estimate, which = "ci"),
            all = CI(Estimate, which = "all"),
            n = length(Estimate)) %>%
  ungroup() %>%
  left_join(actual.est, by = c("Diplotype", "Parameter")) %>%
  mutate(Residual = Actual - Est)
```

19.3 Bootstrapped Vmax Estimates (w/95% CI) Plot

```
# Vmax Plot --------------
ggplot(data = Summary.boot %>%
         filter(Parameter == "Vmax"),
       aes(x = Est, y = Condition))+
  geom_vline(aes(xintercept = Actual),
             color = "blue", size = 0.7, alpha = 0.7)+
  geom_vline(aes(xintercept = c(Actual*1.25)), color = "red",
             linetype = "dashed", size = 0.7)+
  geom_vline(aes(xintercept = c(Actual*0.8)), color = "red",
             linetype = "dashed", size = 0.7)+
  geom_vline(aes(xintercept = c(Actual*1.3)), alpha = 0)+
  geom_vline(aes(xintercept = c(Actual*0.7)), alpha = 0)+
  geom_errorbar(aes(xmin = Lower, xmax = Upper), width = 0.6)+
  geom_pointrange(aes(xmin = Lower, xmax = Upper))+
  facet_wrap(~Diplotype, scales = "free_x", ncol = 3, nrow = 3)+
  theme_bw()+
  ggtitle("Bootstrapped Confidence Intervals (95%) of Vmax")+
  xlab(bquote(paste('V'['max']*" Estimate")))+
  ylab("Experimental Condition")
```

19.4 Bootstrapped Km Estimates (w/95% CI) Plot

```
# Km Plot ---------
ggplot(data = Summary.boot %>%
         filter(Parameter == "Km"),
       aes(x = Est, y = Condition))+
  geom_vline(aes(xintercept = Actual),
             color = "blue", size = 0.7, alpha = 0.7)+
  geom_vline(aes(xintercept = c(Actual*1.25)), color = "red",
             linetype = "dashed", size = 0.7)+
  geom_vline(aes(xintercept = c(Actual*0.8)), color = "red",
             linetype = "dashed", size = 0.7)+
  geom_vline(aes(xintercept = c(Actual*1.3)), alpha = 0)+
  geom_vline(aes(xintercept = c(Actual*0.7)), alpha = 0)+
  geom_errorbar(aes(xmin = Lower, xmax = Upper), width = 0.6)+
  geom_pointrange(aes(xmin = Lower, xmax = Upper))+
  facet_wrap(~Diplotype, scales = "free_x", ncol = 3, nrow = 3)+
  theme_bw()+
```

```
ggtitle("Bootstrapped Confidence Intervals (95%) of Km")+
xlab(bquote(paste('K'['m']*" Estimate")))+
ylab("Experimental Condition")
```

References

Jonathan Carroll, Alicia Schep, and Jonathan Sidi. *ggeasy: Easy Access to ggplot2 Commands*, 2021. URL https://github.com/jonocarroll/ggeasy. R package version 0.1.3.

Alboukadel Kassambara. *ggpubr: ggplot2 Based Publication Ready Plots*, 2020. URL https://rpkgs.datanovia.com/ggpubr/. R package version 0.4.0.

José Pinheiro, Douglas Bates, and R Core Team. *nlme: Linear and Nonlinear Mixed Effects Models*, 2022. URL https://svn.r-project.org/R-packages/trunk/nlme/. R package version 3.1-157.

José C. Pinheiro and Douglas M. Bates. *Mixed-Effects Models in S and S-PLUS*. Springer, New York, 2000. doi: 10.1007/b98882.

R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2022. URL https://www.R-project.org/.

Hadley Wickham. *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York, 2016. ISBN 978-3-319-24277-4. URL https://ggplot2.tidyverse.org.

Hadley Wickham. *tidyverse: Easily Install and Load the Tidyverse*, 2021. URL https://CRAN.R-project.org/package=tidyverse. R package version 1.3.1.

Hadley Wickham and Maximilian Girlich. *tidyr: Tidy Messy Data*, 2022. URL https://CRAN.R-project.org/package=tidyr. R package version 1.2.0.

Hadley Wickham and Dana Seidel. *scales: Scale Functions for Visualization*, 2022. URL https://CRAN.R-project.org/package=scales. R package version 1.2.0.

Hadley Wickham, Mara Averick, Jennifer Bryan, Winston Chang, Lucy D'Agostino McGowan, Romain François, Garrett Grolemund, Alex Hayes, Lionel Henry, Jim Hester, Max Kuhn, Thomas Lin Pedersen, Evan Miller, Stephan Milton Bache, Kirill Müller, Jeroen Ooms, David Robinson, Dana Paige Seidel, Vitalie Spinu, Kohske Takahashi, Davis Vaughan, Claus Wilke, Kara Woo, and Hiroaki Yutani. Welcome to the tidyverse. *Journal of Open Source Software*, 4(43):1686, 2019. doi: 10.21105/joss.01686.

Hadley Wickham, Winston Chang, Lionel Henry, Thomas Lin Pedersen, Kohske Takahashi, Claus Wilke, Kara Woo, Hiroaki Yutani, and Dewey Dunnington. *ggplot2: Create Elegant Data Visualisations Using the Grammar of Graphics*, 2022a. URL https://CRAN.R-project.org/package=ggplot2. R package version 3.3.6.

Hadley Wickham, Romain François, Lionel Henry, and Kirill Müller. *dplyr: A Grammar of Data Manipulation*, 2022b. URL https://CRAN.R-project.org/package=dplyr. R package version 1.0.9.

Hadley Wickham, Jim Hester, and Jennifer Bryan. *readr: Read Rectangular Text Data*, 2022c. URL https://CRAN.R-project.org/package=readr. R package version 2.1.2.

Claus O. Wilke. *cowplot: Streamlined Plot Theme and Plot Annotations for ggplot2*, 2020. URL https://wilkelab.org/cowplot/. R package version 1.1.1.

Hao Zhu. *kableExtra: Construct Complex Table with kable and Pipe Syntax*, 2021. URL https://CRAN.R-project.org/package=kableExtra. R package version 1.3.4.