Appendix A.2 Code Documentation

1. Simulation

1.1 SNP Expression Level Data Generation

```
#Function to generate a correlated gamma cluster
gamma_generator <- function(n_obs = 10, n_vars = 5, mean = 1, precision = 5, rho = 0.5, n_marker = 1) {

  #Generate random related mean and variance
  mean      <- rep(mean, n_vars) + runif(n_vars, -mean/(mean + 1), mean/(mean + 1))
  precision <- rep(precision, n_vars) + runif(n_vars, -precision/3, precision/3)
  cor_mat   <- genCorMat(n_vars, cors = rep(rho, n_vars * (n_vars - 1)/2))

  #Return data as tibble
  genCorGen(n_obs, n_vars, params1 = mean, params2 = precision,
            dis = "gamma", corMatrix = cor_mat, method = "copula",
            cnames = str_c(sprintf("marker_%i_", n_marker), 1:n_vars), idname = "id", wide = TRUE) %>%
    as_tibble()


}

#Check cors, all good! go from 0 - 0.9, based on real data very few negative
gamma_generator(n_obs = 10, n_vars = 20, mean = 1, precision = 5, rho = 0.1) %>% dplyr::select(-id) %>%

#Testing Viz, all looks good. range mean (0, 1); precision (1, 2)
gamma_generator(n_obs = 100,
                n_vars = 100,
                mean = 1,
                precision = 2,
                rho = 0.5) %>%
  pivot_longer(-id, names_to = "marker", values_to = "expression") %>%
  ggplot(aes(x = expression, colour = marker, fill = marker)) +
  geom_density(alpha = 0.4, adjust = 0.8) +
  theme(legend.position = "none") +
  scale_fill_viridis_d() +
  scale_colour_viridis_d()# +
 # xlim(c(0, 3))

#Visualize correlation - looks good
DataExplorer::plot_correlation(gamma_generator(n_obs = 100,
                                               n_vars = 25,
                                               mean = 1,
                                               precision = 2,
                                               rho = 0.1) %>%
                                 dplyr::select(-id),
                               type = "continuous")

prob_transform <- function(x) {
  exp(x)/(1 + exp(x))
}
```

```r
#Build data generator function
#Set Params
cluster_prop <- 1/5 #a rational number/divisor for clusters
n_true_preds <- 5
n_vars       <- 50
p_threshold  <- c(0.4, 0.6)

data_generator <- function(n_obs = 250, n_vars = 20, rho = 0.1, method = "linear") {

  #Check if valid inputs
  if((rho < 0) | (rho > 1)) {stop("rho must be between (0, 1)")}

  #Initialize Params
  n_cluster  <- n_vars * cluster_prop
  n_preds    <- (cluster_prop)^(-1)
  #Randomize Mean and Precision
  mean      <- seq(0.1, 1, length = n_cluster)
  precision <- seq(1  , 2, length = n_cluster)

  #Generate predictors - Iterate, store as list, and join by id
  syn.df <- pmap(list(x = mean, y = precision, z = 1:n_cluster),
                 .f = function(x, y, z) {

            gamma_generator(n_obs     = n_obs,
                            n_vars    = n_preds,
                            mean      = x,
                            precision = y,
                            rho       = rho,
                            n_marker  = z)

       }) %>% reduce(left_join, by = "id")

  #Label the data by one of linear, quadratic, non-linear
  if(method %in% "linear") {
      pred_sample <- sample(2:ncol(syn.df), n_true_preds)
      prop_true   <- 0
  while((prop_true < p_threshold[1]) | (prop_true > p_threshold[2])) {
      beta_vec    <- runif(n_true_preds, -1, 1)

      syn.df <- syn.df %>%
                mutate(
                 down_syndrome = (as.matrix(.[,pred_sample]) %*% beta_vec),
                 down_syndrome = prob_transform(down_syndrome) > 0.5
                 )

      prop_true <- mean(syn.df$down_syndrome)
    }


  } else if(method %in% "quadratic") {

      pred_sample <- sample(2:ncol(syn.df), n_true_preds)
      prop_true   <- 0
```

```
  while((prop_true < p_threshold[1]) | (prop_true > p_threshold[2])) {

    beta_vec    <- runif(n_true_preds * 2, -1, 1)

    syn.df <- syn.df %>%
              mutate(
               down_syndrome = (as.matrix(.[,pred_sample])   %*% beta_vec[1:n_true_preds]) +
                               (as.matrix(.[,pred_sample])^2 %*% beta_vec[-c(1:n_true_preds)]),
               down_syndrome = prob_transform(down_syndrome) > 0.5
               )

    prop_true <- mean(syn.df$down_syndrome)
  }

} else if(method %in% "sinusoidal") {

    pred_sample <- sample(2:ncol(syn.df), n_true_preds)
    prop_true   <- 0

  while((prop_true < p_threshold[1]) | (prop_true > p_threshold[2])) {

    beta_vec    <- runif(n_true_preds * 2 , -1, 1)

    syn.df <- syn.df %>%
              mutate(
               down_syndrome = sin(as.matrix(.[,pred_sample]) %*% beta_vec[1:n_true_preds]) +
                               cos(as.matrix(.[,pred_sample]) %*% beta_vec[-c(1:n_true_preds)]),
               down_syndrome = prob_transform(down_syndrome) > 0.5
               )

    prop_true <- mean(syn.df$down_syndrome)
  }

}else if(method %in% "power") {

    pred_sample <- sample(2:ncol(syn.df), n_true_preds)
    prop_true   <- 0

  while((prop_true < p_threshold[1]) | (prop_true > p_threshold[2])) {

    alpha_vec   <- runif(n_true_preds, 0, 1)
    beta_vec    <- runif(n_true_preds, -1, 1)

    syn.df <- syn.df %>%
              mutate(
               down_syndrome = (as.matrix(map2_df(.x = syn.df[,pred_sample],
                                                  .y = alpha_vec,
                                                  ~.x^(.y)))) %*% beta_vec,
               down_syndrome = prob_transform(down_syndrome) > 0.5
               )

    prop_true <- ifelse(is.nan(mean(syn.df$down_syndrome)), 0, mean(syn.df$down_syndrome))
  }
```

```
  } else {
  stop("Invalid method. Choose one of linear, quadratic, sinusoidal, or power.")
  }

  return(list(
         data = syn.df %>%
         mutate(
          down_syndrome = ifelse(down_syndrome == TRUE, "Yes", "No") %>%
                          as.factor()) %>%
         dplyr::select(id, down_syndrome, everything()),
         preds = pred_sample))

####Collect all the garbage
  gc()
}

#Test it out, all seems good!
#data_generator(n_obs = 1000, n_vars = 100, rho = 0.5, method = "quadratic")
```

1.2 Generate Final Data With Parallel Computing

```
#Set Up Parallel Computing
#Cores
nCores   <- detectCores() - 1
registerDoParallel(nCores)
today <- Sys.Date() %>% str_replace_all("-", "_")

#Function
taskFun  <- data_generator

#Set up grid for simulation
sim_grid <- expand.grid(
              n_obs  = c(100, 250, 500),
              n_vars = c(25, 50, 100),
              rho    = seq(0.1, 0.9, by = 0.2),
              method = c("linear", "quadratic", "sinusoidal", "power"),
              stringsAsFactors = FALSE
            )

#100 X 10 = 1000 iterations
null.return <- foreach(i = 1:nrow(sim_grid),
                       .packages = c("tidyverse", "simstudy", "data.table")) %dopar% {

        #Call function
        outSub <- taskFun(n_obs  = sim_grid$n_obs[i],
                          n_vars = sim_grid$n_vars[i],
                          rho    = sim_grid$rho[i],
                          method = sim_grid$method[i]
                          )
        #Save output
        saveRDS(list(data = outSub$data, parameters = sim_grid[i, ], true_markers = outSub$preds),
                sprintf("./data/data_generation_%s/simulation_%i.RDS", today, i))
```

```
}

remove(null.return)
gc()
```

```
#Read in the simulated data to validate everything worked
relative_path <- "./data/data_generation_2020_05_01/"
sim.df <- list.files(path = relative_path) %>%
          enframe() %>%
          rename(data_path  = value,
                 simulation = name) %>%
          mutate(
            data_path = str_c(relative_path, "/", data_path),
            input_files = map(.x = data_path, ~read_rds(.x)),
            data        = input_files %>% map("data"),
            data        = map(.x = data, ~ .x %>% dplyr::select(-id)),
            parameters  = input_files %>% map("parameters"),
            true_preds  = input_files %>% map("true_markers"),
            simulation  = as.factor(simulation)
          ) %>%
          dplyr::select(-c(data_path, input_files))
```

1.3 Model Fitting

```
#Set up tuning grids
#Control
control <- trainControl(method = "cv",
                        number = 5,
                        classProbs = TRUE,
                        summaryFunction = twoClassSummary,
                        returnData = FALSE)

db_control <- trainControl(method = "cv",
                           number = 5,
                           classProbs = FALSE,
                           returnData = FALSE)

#List of tunings grids
tune_list  <- function(data) {

tree.depth <- min(c(floor(sqrt(nrow(data))), 30)) #Max depth for ada boost (rpart)

list(
#Lasso
glmnet     = expand.grid(alpha = 1,
                         lambda = 10^seq(-5, -0.5, length = 40)),

#ADA - little slower than GBM
ada        =  expand.grid(
              iter     = c(100, 250),
              maxdepth = tree.depth,
              nu       = seq(0.05, 0.25, length = 20)
```

```
              ),

#GBM - pretty quick tuning
gbm        =  expand.grid(
              n.trees           = c(100, 250),
              interaction.depth = tree.depth,
              shrinkage         = 10^seq(-2, -0.5, length = 20),
              n.minobsinnode    = 10
              ),

#XGlinear grid
xgbLinear  =  expand.grid(
              nrounds = 250,
              eta     = 10^seq(-2, -0.5, length = 5),
              lambda  = 10^seq(-2, -0.5, length = 5),
              alpha   = 1
              ),

#XG Gradiant Tree
xgbTree    =  expand.grid(
              nrounds   = c(100, 250),
              max_depth = tree.depth,
              eta       = 10^seq(-2, -0.5, length = 5),
              gamma     = c(0, 0.05, 0.1),
              colsample_bytree = 1,
              min_child_weight = 1,
              subsample = 1),

#Deep Boosting
deepboost  =  expand.grid(
              num_iter   = 100,
              tree_depth = tree.depth,
              lambda     = 10^seq(-3, -0.5, length = 5),
              beta       = 10^seq(-3, -0.5, length = 5),
              loss_type  = "e"
              )
    )
}

tune_models <- function(train.df, i) {

#Initialize tuning grids
tune_grid <- tune_list(train.df)

#Fit models (deepboost seperately)
a <- Sys.time()
mod <-  c(map2(.x = names(tune_grid)[-6], .y = tune_grid[-6],
          ~train(down_syndrome ~ .,
              data = train.df,
              method = .x,
              trControl = control,
              metric = "ROC",
              tuneGrid = .y)),
```

```
           list(train(down_syndrome ~ .,
              data = train.df,
              method = "deepboost",
              trControl = db_control,
              tuneGrid = tune_grid$deepboost))
          )
b <- Sys.time()
#Display runtime and iteration
print(b - a)
print(i)
#Give appropriate names
names(mod) <- names(tune_grid)

#Save for CV later
saveRDS((mod %>% map("bestTune")), sprintf("./data/results/sim_tuning_parameters/sim_params_%i.RDS", i))

#Collect all garbage and remove for efficient use of memory
#gc()
}

#Let her rip I guess haha
#Set Up Parallel Computing
#Cores
nCores   <- detectCores() - 1
registerDoParallel(nCores)
index    <- 1:nrow(sim.df)
#Here goes nothing
map(.x = index, ~tune_models(sim.df$data[[.x]], .x))
```

1.3.1 Parallel Model Tuning

1.4 Simulation Diagnoistics

```
#Read in the best parameters
relative_path <- "./data/results/sim_tuning_parameters/"
param.df <- list.files(path = relative_path) %>%
          enframe() %>%
          rename(data_path  = value,
                 simulation = name) %>%
          mutate(
            data_path   = str_c(relative_path, "/", data_path),
            best_params = map(.x = data_path, ~read_rds(.x)),
            simulation  = str_split_fixed(data_path, "_", 5)[,5] %>% parse_number(),
            simulation  = as.factor(simulation)
          ) %>%
          dplyr::select(simulation, everything(), -data_path) %>%
          arrange(simulation)

#Read in the associated simulated data
relative_path <- "./data/data_generation_2020_05_01/"
sim.df <- list.files(path = relative_path) %>%
          enframe() %>%
```

```
          rename(data_path  = value,
                 simulation = name) %>%
          mutate(
            data_path = str_c(relative_path, "/", data_path),
            input_files = map(.x = data_path, ~read_rds(.x)),
            data        = input_files %>% map("data"),
            data        = map(.x = data, ~ .x %>% dplyr::select(-id)),
            parameters  = input_files %>% map("parameters"),
            true_preds  = input_files %>% map("true_markers"),
            simulation  = as.factor(simulation)
          ) %>%
          dplyr::select(-c(data_path, input_files))

#Join for the final simulation data frame
sim.df <- left_join(sim.df, param.df, by = "simulation")
#glimpse(sim.df)
```

1.4.1 Read and Join Results

```
#Set up tuning grids
#Control
control <- trainControl(method = "none",
                        classProbs = TRUE,
                        summaryFunction = twoClassSummary,
                        returnData = FALSE)

db_control <- trainControl(method = "none",
                           classProbs = FALSE,
                           returnData = FALSE)

#Fit models
fit_models <- function(data, parameter_list) {



#Fit models (deepboost seperately)
mod <-  c(map2(.x = names(parameter_list)[-6], .y = parameter_list[-6],
          ~train(down_syndrome ~ .,
              data = data,
              method = .x,
              trControl = control,
              metric = "ROC",
              tuneGrid = .y)),
           list(train(down_syndrome ~ .,
              data = data,
              method = "deepboost",
              trControl = db_control,
              tuneGrid = parameter_list$deepboost))
          )

#Give appropriate names
names(mod) <- names(parameter_list)
```

```r
#Return models
return(mod)

#Collect all garbage and remove for efficient use of memory
#gc()
}

#Test Preds
diagnose <- function(models, test.df) {

  bind_rows(
    error = c(map_dbl(.x = models,
                      ~mean(predict(.x, test.df) != test.df$down_syndrome)),
              metric = "error"),
    auc   = c(map_dbl(.x = models[-6], ~ predict(.x, test.df, type = "prob")[,2] %>%
                                roc(test.df$down_syndrome, .) %>%
                                auc()),
              deepboost = NA,
              metric    = "auc")
  ) %>%
    dplyr::select(metric, everything())

}

#CV
make_cv <- function(df, n) {

  crossv_mc(df, n = n, test = 0.2) %>%
  mutate(
    train = map(train, as_tibble),
    test  = map(test, as_tibble),
  ) %>%
  rename(id = .id) %>%
  dplyr::select(id, everything())

}

#Fit and diagnose each simulation with n 5-fold CV's
sim_diagnose <- function(data, best_params, n = 10, i = 1) {

  #Make CV replicates
  cv.df <- make_cv(data, n)

  #Fit and diagnose each cv replicate, return data frame of results
  map2_df(.x = cv.df$train, .y = cv.df$test,
     ~fit_models(.x, best_params) %>%
         diagnose(., .y) %>%
         pivot_longer(-metric, names_to = "model", values_to = "value") %>%
         mutate(value = as.numeric(value)) %>%
         pivot_wider(values_from = "value", names_from = "model")
        ) %>%

  saveRDS(., sprintf("./data/results/sim_cv_results/cv_result_%i.RDS", i))
```

```
}

#Set up furrr parallel computing
plan(multiprocess)


future_map(.x = 1:nrow(sim.df), ~sim_diagnose(data = sim.df$data[[.x]],
                                   sim.df$best_params[[.x]],
                                   n = 25,
                                   i = .x
                                   ))

#Close connection\
plan(sequential)
```

1.4.2 Fit and Run Diagnostics

```
#Read in the best parameters
relative_path <- "./data/results/sim_cv_results/"
cv.df  <- list.files(path = relative_path) %>%
          enframe() %>%
          rename(data_path  = value,
                 simulation = name) %>%
          mutate(
            data_path   = str_c(relative_path, "/", data_path),
            cv_results  = map(.x = data_path, ~read_rds(.x)),
            simulation  = str_split_fixed(data_path, "_", 5)[,5] %>% parse_number(),
            simulation  = as.factor(simulation)
          ) %>%
          dplyr::select(simulation, everything(), -data_path) %>%
          arrange(simulation)

#Read in the associated simulated data
relative_path <- "./data/data_generation_2020_05_01/"
sim.df <- list.files(path = relative_path) %>%
          enframe() %>%
          rename(data_path  = value,
                 simulation = name) %>%
          mutate(
            data_path = str_c(relative_path, "/", data_path),
            input_files = map(.x = data_path, ~read_rds(.x)),
            data        = input_files %>% map("data"),
            data        = map(.x = data, ~ .x %>% dplyr::select(-id)),
            parameters  = input_files %>% map("parameters"),
            true_preds  = input_files %>% map("true_markers"),
            simulation  = as.factor(simulation)
          ) %>%
          dplyr::select(-c(data_path, input_files))

#Working data and diagnostics
sim.df <- left_join(sim.df, cv.df, by = "simulation")
```

```r
#Working long data for density viz
res.long <- sim.df %>%
            dplyr::select(parameters, cv_results) %>%
            mutate(
              cv_results = map(.x = cv_results,
                               ~.x %>%
                                 pivot_longer(-metric,
                                              names_to = "model",
                                              values_to = "value",
                                              values_drop_na = TRUE)),
              parameters  = map(.x = parameters,
                               ~.x %>% as_tibble())
            ) %>%
            unnest(parameters:cv_results) %>%
                        mutate(
              method = as.factor(method) %>%
                       fct_relevel("linear", "quadratic",
                                   "sinusoidal", "power") %>%
                       fct_recode("Linear"     = "linear",
                                  "Quadratic"  = "quadratic",
                                  "Sinusoidal" = "sinusoidal",
                                  "Power"      = "power"),
              model  = as.factor(model) %>%
                       fct_relevel("glmnet", "xgbLinear",
                                   "ada", "gbm", "xgbTree",
                                   "deepboost"),
              n_obs  = as.factor(n_obs) %>%
                       fct_recode("100 Observations" = "100",
                                  "250 Observations" = "250",
                                  "500 Observations" = "500"),
              n_vars = as.factor(n_vars) %>%
                       fct_recode("25  SNP's" = "25",
                                  "50  SNP's" = "50",
                                  "100 SNP's" = "100")

            )

#Working long data with mean cv values
res.mean <- sim.df %>%
            dplyr::select(parameters, cv_results) %>%
            mutate(
              cv_results = map(.x = cv_results,
                               ~.x %>%
                                 pivot_longer(-metric,
                                              names_to = "model",
                                              values_to = "value",
                                              values_drop_na = TRUE) %>%
                                 group_by(metric, model) %>%
                                 summarise(
                                   mean     = mean(value),
                                   variance = sd(value)^2
                                 ) %>%
                                 ungroup() %>%
```

```
                                mutate(
                                  model  = as.factor(model)
                                 )),
              parameters  = map(.x = parameters,
                                ~.x %>% as_tibble())
            ) %>%
            unnest(parameters:cv_results) %>%
                        mutate(
              method = as.factor(method) %>%
                       fct_relevel("linear", "quadratic",
                                   "sinusoidal", "power") %>%
                       fct_recode("Linear"     = "linear",
                                  "Quadratic"  = "quadratic",
                                  "Sinusoidal" = "sinusoidal",
                                  "Power"      = "power"),
              model  = as.factor(model) %>%
                       fct_relevel("glmnet", "xgbLinear",
                                   "ada", "gbm", "xgbTree",
                                   "deepboost"),
              n_obs  = as.factor(n_obs) %>%
                       fct_recode("100 Observations" = "100",
                                  "250 Observations" = "250",
                                  "500 Observations" = "500"),
              n_vars = as.factor(n_vars) %>%
                       fct_recode("25  SNP's" = "25",
                                  "50  SNP's" = "50",
                                  "100 SNP's" = "100")

            )
```

```
auc_ggplot <- function(obs_level, y_lower) {
  ggplot() +
  geom_point(data = res.long %>%
                    filter(metric %in% "auc") %>%
                    filter(n_obs %in% obs_level),
             aes(x = rho, y = value, colour = model, fill = model),
             alpha = 0.08,
             size = 0.8,
             position = "jitter") +
  stat_smooth(data = res.mean %>%
                     filter(metric %in% "auc") %>%
                     filter(n_obs %in% obs_level),
              aes(x = rho, y = mean, colour = model, fill = model),
              geom = "line",
              alpha = 0.56,
              size = 1,
              span = 1,
              se = FALSE,
              method = "loess") +
  labs(
    x = "Correlation",
    y = "CV Mean Area Under the ROC",
    title = sprintf("Model CV AUC by Correlation, Method, and SNP's (N = %i)",
```

```
                    obs_level %>% parse_number())
  ) +
  scale_y_continuous(breaks = seq(y_lower, 1, by = 0.05),
                     limits = c(y_lower, 1)) +
  theme(axis.text.x = element_text(angle = 45, vjust = .5)) +
  scale_x_continuous(breaks = seq(0, 1, by = .2)) +
  facet_wrap(~method + n_vars, ncol = 3) +
  scale_colour_viridis_d("Model") +
  scale_fill_viridis_d("Model")
}

#Generate Grobs
auc.gg.list <- map2(.x = levels(res.mean$n_obs), .y = c(0.7, 0.85, 0.9), ~auc_ggplot(.x, .y))

#Snag names in correct syntax
obs.names <- levels(res.mean$n_obs) %>% str_replace_all(" ", "_") %>% str_replace_all("O", "o")

#Save png
map2(.x = auc.gg.list, .y = obs.names,
     ~ggsave(sprintf("./figures/simulation/auc_%s.png", .y), .x))
```

```
error_ggplot <- function(obs_level, y_upper) {
  ggplot() +
  geom_point(data = res.long %>%
                    filter(metric %in% "error") %>%
                    filter(n_obs %in% obs_level),
             aes(x = rho, y = value, colour = model, fill = model),
             alpha = 0.08,
             size = 0.8,
             position = "jitter") +
  stat_smooth(data = res.mean %>%
                   filter(metric %in% "error") %>%
                   filter(n_obs %in% obs_level),
              aes(x = rho, y = mean, colour = model, fill = model),
              geom = "line",
              alpha = 0.56,
              size = 1,
              span = 1,
              se = FALSE,
              method = "loess") +
  labs(
    x = "Correlation",
    y = "CV Error",
    title = sprintf("Model CV Error by Correlation, Method, and SNP's (N = %i)",
                    obs_level %>% parse_number())
  ) +
  scale_y_continuous(breaks = seq(0, y_upper, by = 0.1),
                     limits = c(0, y_upper)) +
  facet_wrap(~method + n_vars, ncol = 3) +
  scale_colour_viridis_d("Model") +
  scale_fill_viridis_d("Model")
}
```

```
#Generate Grobs
error.gg.list <- map2(.x = levels(res.mean$n_obs), .y = c(0.4, 0.3, 0.2), ~error_ggplot(.x, .y))

#Snag names in correct syntax
obs.names <- levels(res.mean$n_obs) %>% str_replace_all(" ", "_") %>% str_replace_all("0", "o")

#Save png
map2(.x = error.gg.list, .y = obs.names,
     ~ggsave(sprintf("./figures/simulation/error_%s.png", .y), .x))
```

1.4.3 Visualize and Save Results

```
sim.cor <- sim.df$data[[20]] %>%
  DataExplorer::plot_correlation(., type = "continuous",
                                 title = "Correlation of 100 Synthetic SNP Expression Levels (N = 250, 
                                 theme_config = list(
                                   axis.text.x = element_blank(),
                                   axis.text.y = element_blank(),
                                   plot.title  = element_text(hjust = 0.5)),
                                 ggtheme = theme_bw())


sim.dens <- sim.df$data[[2]] %>%
  gather(variable, value, -down_syndrome) %>%
  ggplot(aes(x = value, colour = variable, fill = variable)) +
  geom_density(alpha = 0.3) +
  theme(legend.position = "none") +
  labs(
    x = "SNP Expression Level",
    y = "Density",
    title = "25 Synthetic SNP Levels by Marker"
  ) +
  scale_fill_viridis_d() +
  scale_colour_viridis_d() +
  xlim(c(0, 2))
```

1.4.4 Correlation example for paper

2. Down Syndrome in Mice Application

2.1 Data Read and Clean

```
down.df <- read_csv("./data/mice_down_syndrome.csv") %>%
  filter(!(MouseID %in% c("3426_13", "3426_14", "3426_15"))) %>%
  dplyr::select( -c("BCL2_N","H3MeK4_N","BAD_N","EGR1_N","H3AcK18_N","pCFOS_N","Bcatenin_N","MEK_N","ELK
  janitor::clean_names() %>%
  rename(
    id = mouse_id,
    down_syndrome = class
  ) %>%
```

```
  mutate(
    down_syndrome = ifelse(down_syndrome == "Control", "No", "Yes") %>% as.factor() %>% fct_relevel("No
  ) %>%
  dplyr::select(id, down_syndrome, everything())
```

2.1.1 Exploration

```
#Plot Correlation structure, try to mimic with simulation
mice.cor <- DataExplorer::plot_correlation(down.df, type = "continuous",
                                 title = "Correlation of 68 Mice SNP Expression Levels",
                                 theme_config = list(
                                   axis.text.x = element_blank(),
                                   axis.text.y = element_blank(),
                                   plot.title  = element_text(hjust = 0.5)),
                                 ggtheme = theme_bw())

ggsave("./figures/mice/snp_correlation.png", mice.cor)

mice.dens <- down.df %>%
  gather(variable, value, -c(id, down_syndrome)) %>%
  ggplot(aes(x = value, colour = variable, fill = variable)) +
  geom_density(alpha = 0.4) +
  theme(legend.position = "none") +
  labs(
    x = "SNP Expression Level",
    y = "Density",
    title = "68 Mice SNP Levels by Marker"
  ) +
  scale_fill_viridis_d() +
  scale_colour_viridis_d() +
  xlim(c(0, 2))

dens.join <- (mice.dens + sim.dens)

ggsave("./figures/mice/density_example.png", dens.join)

sum.df <- down.df %>%
  gather(variable, value, -c(id, down_syndrome)) %>%
  group_by(variable) %>%
  summarize(
    mean = mean(value),
    max  = max(value),
    min  = min(value),
    precision   = sd(value)^2
  )


sum.df %>%
  dplyr::select(mean) %>%
  summary()

sum.df %>%
```

```
  dplyr::select(max) %>%
  summary()

sum.df %>%
  dplyr::select(min) %>%
  summary()

sum.df %>%
  dplyr::select(precision) %>%
  summary()
```

2.1.2 Test/Train Split

```
#80% test train split
set.seed(4)
sample <- sample(1:nrow(down.df), round(nrow(down.df) * 0.8), replace = TRUE)

train.df <- down.df[ ,-1] %>% slice(sample)
test.df  <- down.df[ ,-1] %>% slice(-sample)

#Check that test/train are comparable
mean_abs_dif <- map2_dbl(.x = train.df,
                         .y = test.df,
                          ~abs(mean(.x) - mean(.y)))
#nothing larger than an absolute mean diff of 0.05, good to go

mean(train.df$down_syndrome == "Yes")
mean(test.df$down_syndrome == "Yes")
```

2.2 Model Tuning/Fitting

```
#Set up tuning grids
#Control
control <- trainControl(method = "cv",
                        number = 5,
                        classProbs = TRUE,
                        summaryFunction = twoClassSummary,
                        returnData = FALSE)
tree.depth <- floor(sqrt(nrow(train.df)))

#Lasso
lasso.grid <- expand.grid(alpha = 1,
                          lambda = 10^seq(-5, -2, length = 50))
#GBM - pretty quick tuning
gbm.grid   <- expand.grid(
              n.trees           = c(100, 250, 500),
              interaction.depth = tree.depth,
              shrinkage         = seq(0.05, 0.25, length = 20),
              n.minobsinnode    = 10
              )
```

```
#ADA - little slower than GBM
ada.grid   <- expand.grid(
              iter     = c(100, 250, 500),
              maxdepth = tree.depth,
              nu       = seq(0.05, 0.25, length = 20)
              )

#XGlinear grid
xgl.grid   <- expand.grid(
              nrounds = c(100, 250, 500),
              eta     = seq(0.05, 0.25, length = 5),
              lambda  = seq(0, 0.5, by = 0.1),
              alpha   = 1
              )

#XG Gradiant Tree
xgt.grid  <- expand.grid(
              nrounds   = c(100, 250),
              max_depth = tree.depth,
              eta       = seq(0.05, 0.5, by = 0.05),
              gamma     = seq(0, 0.5, by = 0.1),
              colsample_bytree = 1,
              min_child_weight = 1,
              subsample = 1)

#Deep Boosting
db.grid   <- expand.grid(
              num_iter   = c(50, 100),
              tree_depth = tree.depth,
              lambda     = 10^seq(-3, -0.2, length = 10),
              beta       = 10^seq(-3, 0.2, length = 10),
              loss_type  = "e"
              )
```

```
#Train models
mod.lasso <- train(down_syndrome ~ .,
                   data = train.df,
                   method = "glmnet",
                   trControl = control,
                   metric = "ROC",
                   tuneGrid = lasso.grid)

mod.gbm   <- train(down_syndrome ~ .,
                    data = train.df,
                    method = "gbm",
                    trControl = control,
                    metric = "ROC",
                    tuneGrid = gbm.grid)

mod.ada   <- train(down_syndrome ~ .,
                    data = train.df,
                    method = "ada",
                    trControl = control,
                    metric = "ROC",
```

```
                    tuneGrid = ada.grid)

mod.xgl   <- train(down_syndrome ~ .,
                    data = train.df,
                    method = "xgbLinear",
                    trControl = control,
                    metric = "ROC",
                    tuneGrid = xgl.grid)

mod.xgt   <- train(down_syndrome ~ .,
                    data = train.df,
                    method = "xgbTree",
                    trControl = control,
                    metric = "ROC",
                    tuneGrid = xgt.grid)

mod.db    <- train(down_syndrome ~ .,
                    data = train.df,
                    method = "deepboost",
                    trControl = trainControl(method = "cv",
                                             number = 5,
                                             classProbs = FALSE),
                 #   metric = "Accuracy",
                    tuneGrid = db.grid)
#2 hours to run - limitation

best.list <- list(glmnet       = mod.lasso,
                  ada          = mod.ada,
                  gbm          = mod.gbm,
                  xgbLinear    = mod.xgl,
                  xgbTree      = mod.xgt,
                  deepboost    = mod.db
                  ) %>%
             map("bestTune")

#Save Params for testing
today <- Sys.Date() %>% str_replace_all("-", "_")
#saveRDS(best.list, sprintf("./data/results/best_train_params_%s.RDS", today))
```

2.3 Model Diagnostics

```
#Read in param vals
best.list <- readRDS("./data/results/best_train_params_deep_2020_04_28.RDS")
#Function to fit all models
fit_models <- function(param.list, train.df) {

  #Fit models (deepboost seperately)
mod <-  c(map2(.x = names(param.list)[-6], .y = param.list[-6],
          ~train(down_syndrome ~ .,
              data = train.df,
              method = .x,
              trControl = trainControl(method = "none",
```

```
                                        classProbs = TRUE),
              metric = "ROC",
              tuneGrid = .y)),
           list(train(down_syndrome ~ .,
              data = train.df,
              method = "deepboost",
              trControl = trainControl(method = "none",
                                        classProbs = FALSE),
              tuneGrid = param.list$deepboost))
          )
#Give appropriate names
names(mod) <- names(param.list)
return(mod)
}

#Test
a <- Sys.time()
test.fit <- fit_models(best.list, train.df)
b <- Sys.time()
(b - a) #47 seconds - not bad

#Test Error
map_dbl(.x = test.fit, ~mean(predict(.x, test.df) != test.df$down_syndrome))

#Test Preds
diagnose <- function(models, test.df) {

  bind_rows(
    error = c(map_dbl(.x = models,
                      ~mean(predict(.x, test.df) != test.df$down_syndrome)),
              metric = "error"),
    auc   = c(map_dbl(.x = models[-6], ~ predict(.x, test.df, type = "prob")[,2] %>%
                                roc(test.df$down_syndrome, .) %>%
                                auc()),
              deepboost = NA,
              metric    = "auc")
  ) %>%
    dplyr::select(metric, everything())

}

#Test
a <- Sys.time()
diagnose(test.fit, test.df)
b <- Sys.time()
(b - a)

#CV
make_cv <- function(df, folds = 5) {

  crossv_kfold(df, k = folds) %>%
  mutate(
    train = map(train, as_tibble),
```

```
    test  = map(test, as_tibble),
  ) %>%
  rename(id = .id) %>%
  dplyr::select(id, everything())

}

#Set Number
set.seed(4)
#future::plan(multiprocess)
N <- 100
start <- Sys.time()
cv.df <- tibble(iteration = 1:N) %>%
         mutate(
           cv = map(.x = iteration, ~make_cv(train.df, 5))
         ) %>% unnest(cols = cv) %>%
 # slice(1:2) %>%
  mutate(
    models      = map(.x = train, ~fit_models(best.list, .x))) %>%
  mutate(
    diagnostics = map2(.x = models, .y = test,  ~diagnose(.x, .y))
  )
end <- Sys.time()
(run.time <- end - start)

#Save results
#result.df <- cv.df %>% dplyr::select(id, iteration, diagnostics) %>% unnest(diagnostics) %>% arrange(m

#saveRDS(result.df, sprintf("./data/results/cv_res_%s.RDS", Sys.Date() %>% str_replace_all("-", "_")))
```

2.4 Visualize and Summarize Results

```
#Read and tidy
result.df <- read_rds("./data/results/cv_results_final_2020_04_28.RDS")

result.long <- result.df %>%
  pivot_longer(-c(id:metric), names_to = "model", values_to = "value", values_drop_na = TRUE) %>%
  mutate(
    value = as.numeric(value),
    )

# AUC
#Box/violin
auc.box.gg <- result.long %>%
  filter(metric %in% "auc") %>%
  mutate(
    model = as.factor(model) %>%
            fct_reorder(value, .desc = TRUE, .fun = mean)
  )
  ggplot(aes(x = model, y = value, fill = model, colour = model)) +
  geom_violin(trim = FALSE, alpha = 0.3) +
  geom_boxplot(width = 0.1, colour = "black", fill = "white", alpha = 0.1) +
```

```
  labs(
    y = "CV Area Under the ROC",
    x = "Model",
    title = "100 5-Fold CV AUC by Model"
  ) +
  scale_colour_viridis_d("Model") +
  scale_fill_viridis_d("Model") +
  ylim(c(0.98, 1.2))

#Density, much better
auc.dens.gg <- result.long %>%
  filter(metric %in% "auc") %>%
  mutate(
    model = as.factor(model) %>%
            fct_reorder(value, .desc = TRUE, .fun = mean)
  ) %>%
  ggplot(aes(x = value, fill = model, colour = model)) +
  geom_density(alpha = 0.3, adjust = 2, size = 1.2) +
  labs(
    x = "CV Area Under the ROC",
    y = "Density",
    title = "100 5-Fold CV AUC by Model"
  ) +
  scale_colour_viridis_d("Model") +
  scale_fill_viridis_d("Model") +
  xlim(c(0.99, 1))

#Save png
#ggsave("./figures/mice/auc_density.png", auc.dens.gg)


error.dens.gg <- result.long %>%
  filter(metric %in% "error") %>%
  mutate(
    model = as.factor(model) %>%
            fct_reorder(value, .desc = FALSE, .fun = mean)
  ) %>%
  ggplot(aes(x = value, fill = model, colour = model)) +
  geom_density(alpha = 0.3, adjust = 2, size = 1.2) +
  labs(
    x = "CV Error",
    y = "Density",
    title = "100 5-Fold CV Error by Model"
  ) +
  scale_colour_viridis_d("Model") +
  scale_fill_viridis_d("Model") +
  guides(fill = guide_legend(nrow = 1),
         colour = guide_legend(nrow = 1))

auc.error.dens.gg <- result.long %>%
  mutate(
    metric = as.factor(metric) %>%
             fct_recode("Error" = "error", "AUC" = "auc") %>%
```

```
            fct_relevel("Error", "AUC"),
    model  = as.factor(model) %>%
            fct_relevel("gbm", "ada", "xgbTree", "deepboost", "xgbLinear", "glmnet")
  ) %>%
  ggplot(aes(x = value, fill = model, colour = model)) +
  geom_density(alpha = 0.3, adjust = 2, size = 1.2) +
  labs(
    x = "CV Error",
    y = "Density",
    title = "100 5-Fold CV Error by Model"
  ) +
  scale_colour_viridis_d("Model") +
  scale_fill_viridis_d("Model") +
  guides(fill = guide_legend(nrow = 1),
         colour = guide_legend(nrow = 1)) +
  facet_wrap(~metric, scales = "free")

#Save png
#ggsave("./figures/mice/error_density.png", error.dens.gg)

#mice.final.dens <- (error.dens.gg + auc.dens.gg)
ggsave("./figures/mice/error_and_auc_density.png", auc.error.dens.gg)


#Table of results
res.table <- result.long %>%
  group_by(model, metric) %>%
  summarise(
    mean = mean(value),
    median = median(value),
    variance = sd(value)^2
  ) %>%
  ungroup() %>%
  mutate(model = as.factor(model) %>%
               fct_reorder(mean, .desc = TRUE, .fun = max)) %>%
  arrange(metric, model) %>%
  knitr::kable(digits = 5)

#kableExtra::save_kable(res.table, file = "./figures/mice/result_table_litera.png", bs_theme = "litera"


#Final train/test results
best.list <- readRDS("./data/results/best_train_params_deep_2020_04_28.RDS")
final.result <- fit_models(best.list, train.df) %>% diagnose(., test.df) %>%
                nest(-metric)


final.kable <- final.result %>%
  pivot_longer(-metric, names_to = "model", values_to = "value") %>%
  mutate(value = as.numeric(value)) %>%
  pivot_wider(values_from = "value", names_from = "model") %>%
  knitr::kable(digits = 5)
```

```
kableExtra::save_kable(final.kable, file = "./figures/mice/final_test_train_table.png", bs_theme = "lit
```