Summarizing HTP Temporal Data

singhdj2@ksu.edu

August 8, 2019

Introduction

The objective of today's exercise is to make you familiar with the processing of high-throughput phenotyping data. We will use plant height trait as an example and along the way will learn how to summarize and explore data. Some of the concepts/topics discussed here would be applicable to your research as well.

About the Data

The data was collected from two wheat experiments in South Asia in field season 2018 using an unmanned aerial vehicle a.k.a. drone. There were a total of 1200 wheat test plots (size 1.2m x 3m) in each experiment. The two experiments are identical in terms of genotypes and plot layout except these were planted at two different sowing dates (Optimum on 23-Oct, Late on 11 Nov). The main objective of this field experiment was to assess the effect of sowing dates on plant growth and yield. The data was collected at multiple times during the wheat growing season. Overall this dataset captures the dynamic changes in wheat height during its growth cycle.

Expected Outcome

Through this exercise you will learn:

- How to process plot-level height data from multiple dates
- Model the growth dynamics of height through non-linear regression
- Visualize the results and make sense of the trends
- Compare the effects of sowing dates on wheat height and growth

First we will load all the required packages

First, let us create a logistic regression function to model wheat growth. This is just the definition of a function, we will call it during the later steps.

Quick question: Why is there any need to create functions?

Non-linear Logistic Regression Function

```
runLogReg = function(dates, phenValue){
  # create column name strings for fit stats
  fit.stat.cols <- c("Min.max.accuracy","MAE","MAPE","MSE","RMSE",
                     "NRMSE.mean","NRMSE.median","NRMSE.mean.accuracy", "NRMSE.median.accuracy","Efron.r
  if(sum(!is.na(phenValue))==0){          # skip if no phenotype data
    pred = data.frame(x=NA,y=NA) %>%      # fill NA since no data
```

```
    nest()
  res <- data.frame(phi1=NA, phi2=NA, phi3=NA,
                    flag="no-data", stringsAsFactors = F)
  fit.stats <- as.data.frame(matrix(nrow = 1,ncol = 11,
                                    dimnames = list(1,fit.stat.cols))) # empty fit.stats
  pred.res <- bind_cols(pred,res,fit.stats)      # bind columns
  return(pred.res)                    # return the data frame
  }
if(max(phenValue, na.rm=TRUE) < 60){   ## skip plots that never reach 60cm height
  pred = data.frame(x=NA,y=NA) %>%
    nest()
  res <- data.frame(phi1=NA, phi2=NA, phi3=NA, flag="short.max.ht",
                    stringsAsFactors = F) # flag short plots
  fit.stats <- as.data.frame(matrix(nrow = 1,ncol = 11,
                                    dimnames = list(1,fit.stat.cols))) # empty fit.stats
  pred.res <- bind_cols(pred,res,fit.stats)
  return(pred.res)
}
## get Thermal Time
days = dates
if(sum(!is.na(phenValue)) >= 9) {        # run only plots that have at least 9 time-points
  # add some dummy variables for min and max height at 50-100 ThermalDays before/after the phenotypin
  # buffer raw data to force start values to zero
  days = c(days, c(0,50,75,100), max(days,na.rm=T)+c(50,75,100,150))
  height = c(phenValue, c(0,0,0,0),rep(max(phenValue,na.rm=T),4))   #c(0,0,0,100,100,100))
  ##find initial starting values for phi2 and phi3,  fix phi1 at max pheno value
  phi1 = max(phenValue,na.rm=T)                      # maximum height parameter
  phi2 = coef(lm(logit(height/phi1)~days))[2]        # growth rate
  phi3 = -coef(lm(logit(height/phi1)~days))[1]/phi2  # time at half-max growth
  # Run logistic regression model
  growth_mod <- try(nls(height~ max(phenValue,na.rm=T)/(1+exp(-phi2*(days - phi3))),
                        start=list(phi2=phi2,phi3=phi3), trace=FALSE), silent=TRUE)
  if(class(growth_mod)!="try-error"){              # this condition is to handle errors
    phi2 = coef(growth_mod)[1]
    phi3 = coef(growth_mod)[2]
    ## get predicted value
    pred = getPred(phi1, phi2, phi3) %>%
      nest()
    res = data.frame(phi1, phi2, phi3,flag="data.complete",
                     stringsAsFactors = F)         # combine all growth coeficients
    fit.stats <- accuracy(list(growth_mod),plotit = F)$Fit.criteria # gives pseudo-Rsq and other fit
    colnames(fit.stats) <- fit.stat.cols
    pred.res = bind_cols(pred,res,fit.stats)       # bind growth and predicted values
    return(pred.res)
  } else {                                         # skip if doesn't converge
    pred = data.frame(x=NA,y=NA) %>%
      nest()
    res <- data.frame(phi1=NA,phi2=NA,phi3=NA,flag='try-error',
                      stringsAsFactors = F)
    fit.stats <- as.data.frame(matrix(nrow = 1,ncol = 11,dimnames = list(1,fit.stat.cols))) # empty f
    pred.res <- bind_cols(pred,res,fit.stats)
    return(pred.res)
  }
```

```
  } else {
    pred = data.frame(x=NA,y=NA) %>%
      nest()
    res <- data.frame(phi1=NA,phi2=NA,phi3=NA,flag="few-time-points",
                      stringsAsFactors = F)
    fit.stats <- as.data.frame(matrix(nrow = 1,ncol = 11,
                                      dimnames = list(1,fit.stat.cols))) # empty fit.stats
    pred.res <- bind_cols(pred,res,fit.stats)
    return(pred.res)
  }
}
```

Helper function to extract predictions for desired input 'x' values (e.g. days, thermal days).

```
getPred = function(phi1, phi2, phi3){
  x <- seq(0,2500,by=50)  #construct a range of x values for thermal time
  y <- try(phi1/(1+exp(-phi2*(x - phi3))),silent=TRUE)  #predicted y values  <old: exp(-(phi2+phi3*x)))
  if(class(y)!="try-error"){
    pred <- data.frame(x,y)                                  #create the prediction data frame
    return(pred)
  } else {
    pred <- data.frame(x=NA,y=NA)
    return(pred)
  }
}
```

Now lets get started with our analysis by reading the data file. Enter `help(read.csv)` in the console to know about the *read.csv* function.

```
# read curated data UAV height data
dat.ht <- read.csv("C:/Users/singhdj2/Documents/daljit/r_stuff/tutorials/data/18LDH-tutorial-log-regress
head(dat.ht)
```

```
##   condition plot_id thermal.time DAS phenotype_value
## 1   Optimum    4120         1101  78        48.64035
## 2   Optimum    4119         1101  78        53.21288
## 3   Optimum    4118         1101  78        41.98752
## 4   Optimum    4117         1101  78        47.62004
## 5   Optimum    4116         1101  78        47.49351
## 6   Optimum    4115         1101  78        42.04319
```

There are five columns in the data file. Column *condition* refers to the optimum and late sowing conditions; *plot_id* is unique plot_id for 1200 plots in each condition; *thermal.time* is the cumulative sum of the mean daily air temperature at each condition; DAS is Days after sowing (in Julian Day units); *phenotype_value* is the height value estimated from drone based images.

Before moving forward with your analysis, it is always good to take a look at the data structure and column attributes . Here we will take a quick glimpse at our data file.

```
glimpse(dat.ht)
```

```
## Observations: 29,983
## Variables: 5
## $ condition       <chr> "Optimum", "Optimum", "Optimum", "Optimum", "O...
## $ plot_id         <int> 4120, 4119, 4118, 4117, 4116, 4115, 4114, 4113...
## $ thermal.time    <int> 1101, 1101, 1101, 1101, 1101, 1101, 1101, 1101...
## $ DAS             <int> 78, 78, 78, 78, 78, 78, 78, 78, 78, 78, 78, 78...
## $ phenotype_value <dbl> 48.64035, 53.21288, 41.98752, 47.62004, 47.493...
```

```
# set the condition as 'factor' for plotting convenience
dat.ht <- as_tibble(dat.ht) %>%
  mutate(condition=factor(condition,levels=c("Optimum","Late")))
```

Let us find out how many unique time-points are there per condition?

```
dat.ht %>%
  group_by(condition) %>%
  summarize(time.points=length(unique(thermal.time)))
```

```
## # A tibble: 2 x 2
##   condition time.points
##   <fct>           <int>
## 1 Optimum            15
## 2 Late               11
```

In which condition the plants grow taller? Lets get mean height per time-point for each condition to find out.

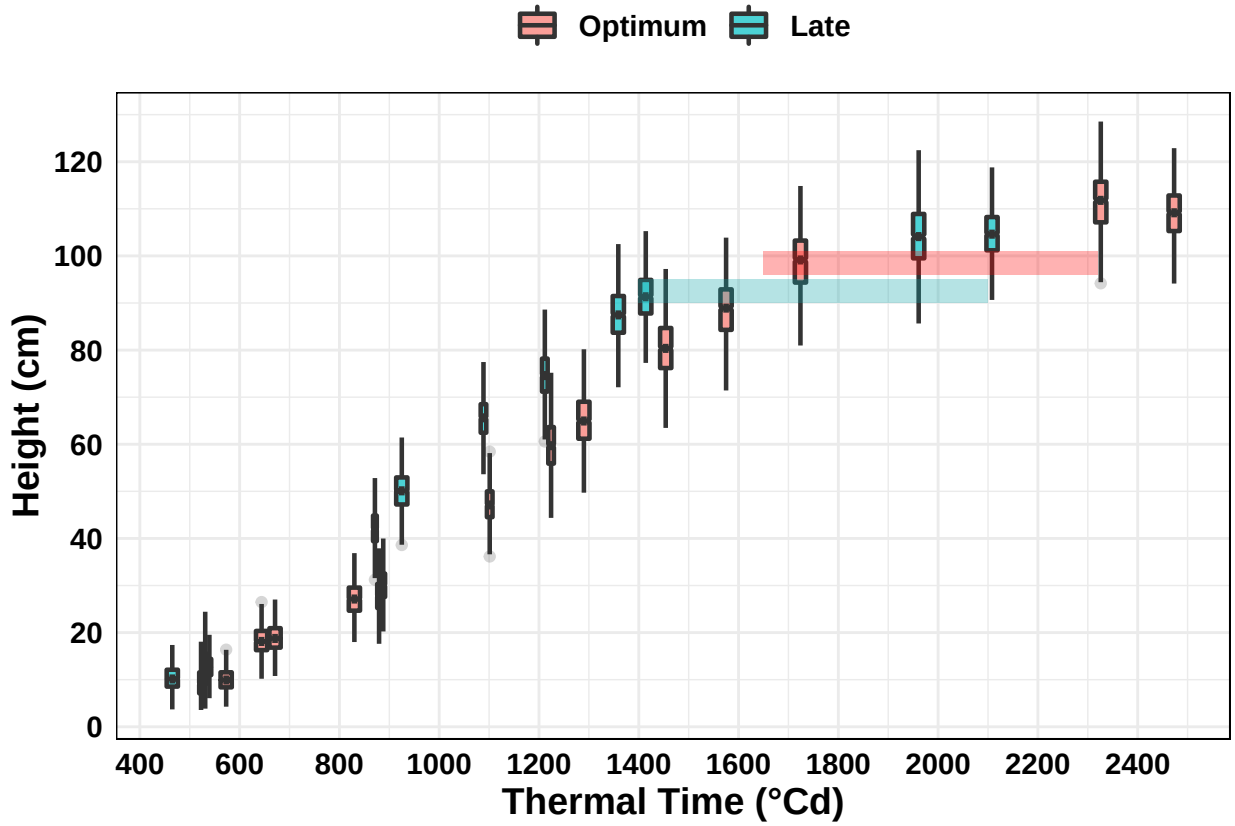
```
dat.ht %>%
  group_by(condition,thermal.time) %>%
  summarise(ht.mean=mean(phenotype_value,na.rm=T))
```

```
## # A tibble: 26 x 3
## # Groups:   condition [2]
```

```
##    condition thermal.time ht.mean
##    <fct>            <int>   <dbl>
##  1 Optimum            546   12.7
##  2 Optimum            573    9.98
##  3 Optimum            644   18.2
##  4 Optimum            671   18.8
##  5 Optimum            830   27.1
##  6 Optimum            881   27.7
##  7 Optimum            898   30.1
##  8 Optimum           1101   47.3
##  9 Optimum           1226   59.8
## 10 Optimum           1290   65.1
## # ... with 16 more rows
```

We can dig a little deeper to look at the raw height trends at both field conditions.

```
x.label = "Thermal Time (°Cd)"
ggplot(data=dat.ht, aes(y = phenotype_value,
                        x = thermal.time,
                        group=interaction(thermal.time,condition),
                        fill=condition)) +
  geom_boxplot(width=25,alpha=0.7,notch = TRUE, size=0.8,
               notchwidth = 0.5, outlier.alpha = 0.2) +
  labs(x=x.label,y="Height (cm)") +
  scale_y_continuous(breaks= seq(0,150,20)) +
  scale_x_continuous(breaks= seq(0,2400,200)) +
  theme_minimal() +
  theme(panel.border = element_rect(colour = "black", fill = NA, size = .5),
        axis.text.x = element_text(colour = "black", size = 11,angle = 0),
        axis.text.y = element_text(colour = "black", size = 11),
        plot.title = element_text(size = 14, face = "bold"),
        text = element_text(size=14,face="bold"),
        legend.position="top") +
  guides(fill=guide_legend(title="")) +
  annotate("rect", xmin=1650, xmax=2320, ymin=96, ymax=101, alpha=0.3, fill="red") +
  annotate("rect", xmin=1400, xmax=2100, ymin=90, ymax=95, alpha=0.3, fill="#0da3a7")
```




```
# We can also save the above plot by using ggsave function.
# create a file name character string
# fName = paste0('../data/18LDH_tutorial-height-trend_boxplot_',Sys.Date(),'.jpg')
# # save the plot on disk
# ggsave(fName, plot = last_plot(), device = "jpg", path = "output/",
#        scale = 1, width = 10, height = 6.5, units = c("in"), dpi=800)
```

Question: By chance, did you notice different height trends at two conditions? Which condition has plants growing faster/slower?

We have data collected at multuple time-points. This rich temporal information can be summarized into a few growth parameters that have a straightforward biological interpretation. To do so, we will deploy our logistic regression function (remember we defined logistic function at the beginning of this script!) for each plot and condition using group_by and do functions from tidyverse package.

```
dat.ht.growth <- dat.ht %>%
  group_by(condition,plot_id) %>%
  do(runLogReg(dates = .$thermal.time, phenValue = .$phenotype_value)) %>%
  ungroup()
head(dat.ht.growth)
```

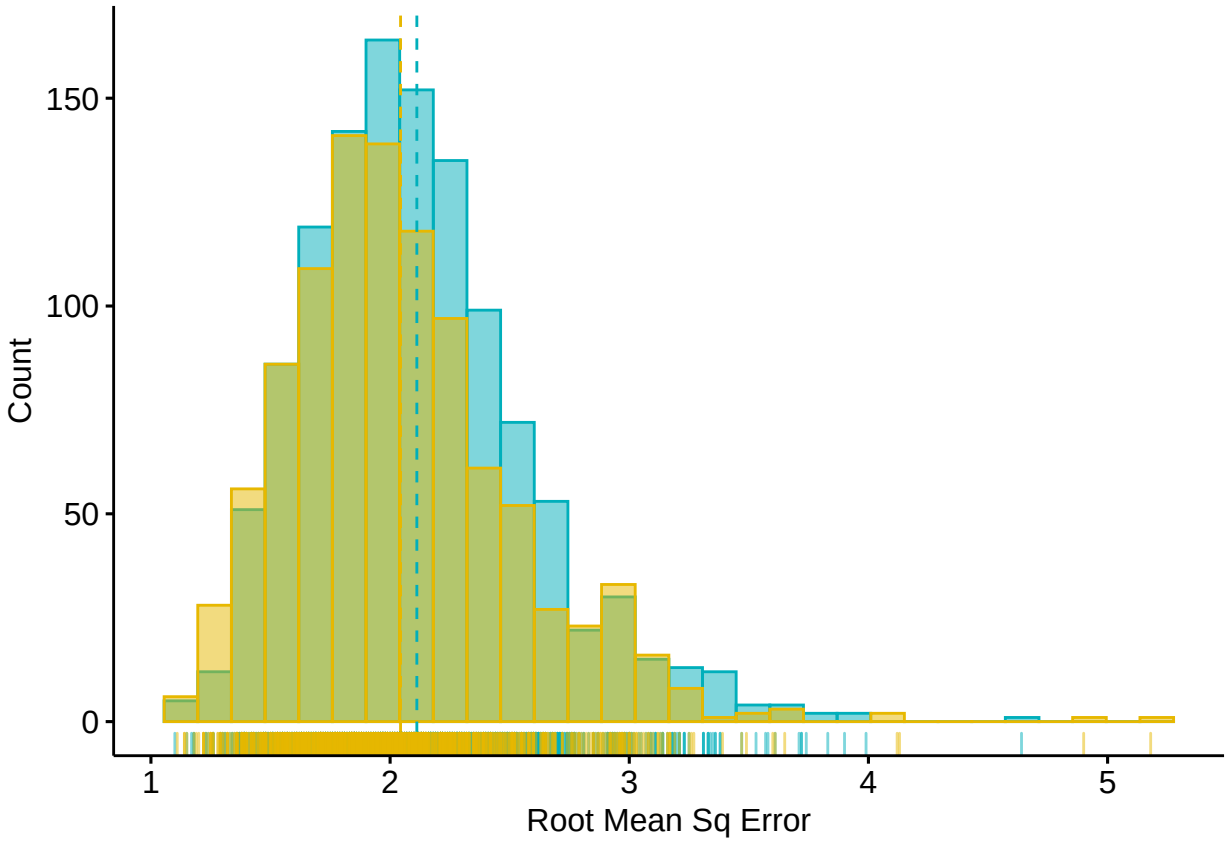
```
## # A tibble: 6 x 18
```

```
##   condition plot_id data   phi1    phi2  phi3 flag  Min.max.accuracy   MAE
##   <fct>       <int> <lis> <dbl>   <dbl> <dbl> <chr>            <dbl> <dbl>
## 1 Optimum      1001 <tib~  95.4 0.00357 1186. data~            0.799  1.13
## 2 Optimum      1002 <tib~ 109.  0.00381 1245. data~            0.782  1.37
## 3 Optimum      1003 <tib~ 107.  0.00333 1135. data~            0.793  1.81
## 4 Optimum      1004 <tib~ 104.  0.00340 1266. data~            0.782  1.65
## 5 Optimum      1005 <tib~ 106.  0.00334 1197. data~            0.786  1.79
## 6 Optimum      1006 <tib~ 112.  0.00309 1204. data~            0.786  2.35
## # ... with 9 more variables: MAPE <dbl>, MSE <dbl>, RMSE <dbl>,
## #   NRMSE.mean <dbl>, NRMSE.median <dbl>, NRMSE.mean.accuracy <dbl>,
## #   NRMSE.median.accuracy <dbl>, Efron.r.squared <dbl>, CV.prcnt <dbl>
```

Note that the parameters phi1, phi2, phi3 correspond to the maximum plant height (cm), growth rate (cm/thermal time), thermal time when plant is halfway its maximum height (measured in degree Cd units), respectively.

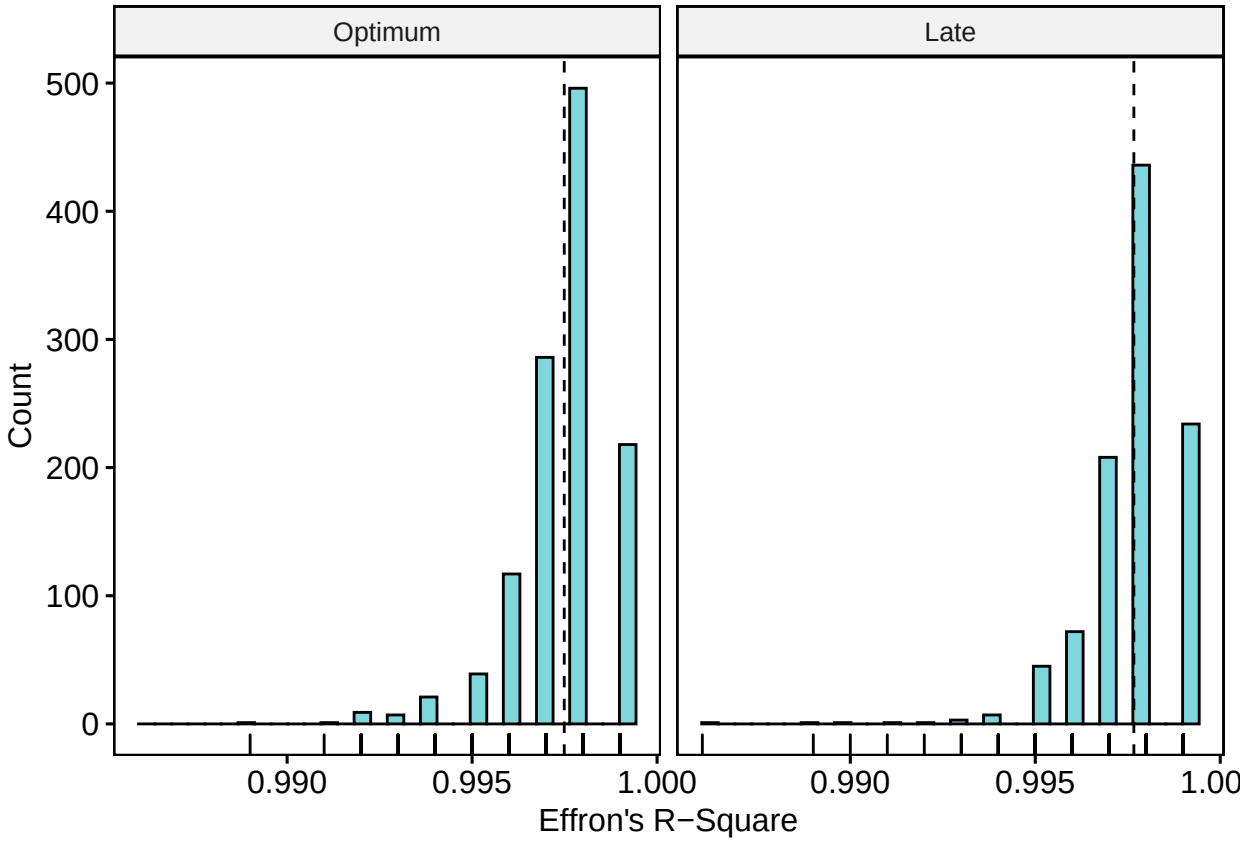
Now that we have succefully summarized our time-points into three biologically interpretable parameters, we can go ahead and take a look at the regression fit summaries to find out how accurately our model was able to fit the original data.

```
gghistogram(dat.ht.growth, x = "RMSE",
   add = "mean", rug = T,
   color = "condition", fill = "condition",
   palette = c("#00AFBB", "#E7B800")) +
  rremove("legend") +
  xlab("Root Mean Sq Error") +
  ylab("Count")
```



Keep in mind that the R-square statistic is a bit tricky for non-linear models. Therefore, Efron.r-sq is used as a workaround approximation for checking the model fits in non-linear models. We will plot Efron.r-sq to check the fit of our regression model.

```
gghistogram(dat.ht.growth, x = "Efron.r.squared",
   add = "mean", rug = T,
   #color = "condition",
   fill = "#00AFBB",
   facet.by = "condition") +
  rremove("legend") +
  xlab("Effron's R-Square") +
  ylab("Count")
```



It looks like our logistic model fit our data really well. Almost 99% of the data had a perfect fit.

Now we are quite sure that our data fits worked well. Now we can ask some biological questions, for example, does sowing date have any effect on plant growth? Lets dig further...

Let us check mean values for each growth parameter.

```
growth.mean <- dat.ht.growth %>%
  group_by(condition) %>%
  summarize_at(vars(contains('phi')),funs(mean(.,na.rm = T)))
growth.mean
```

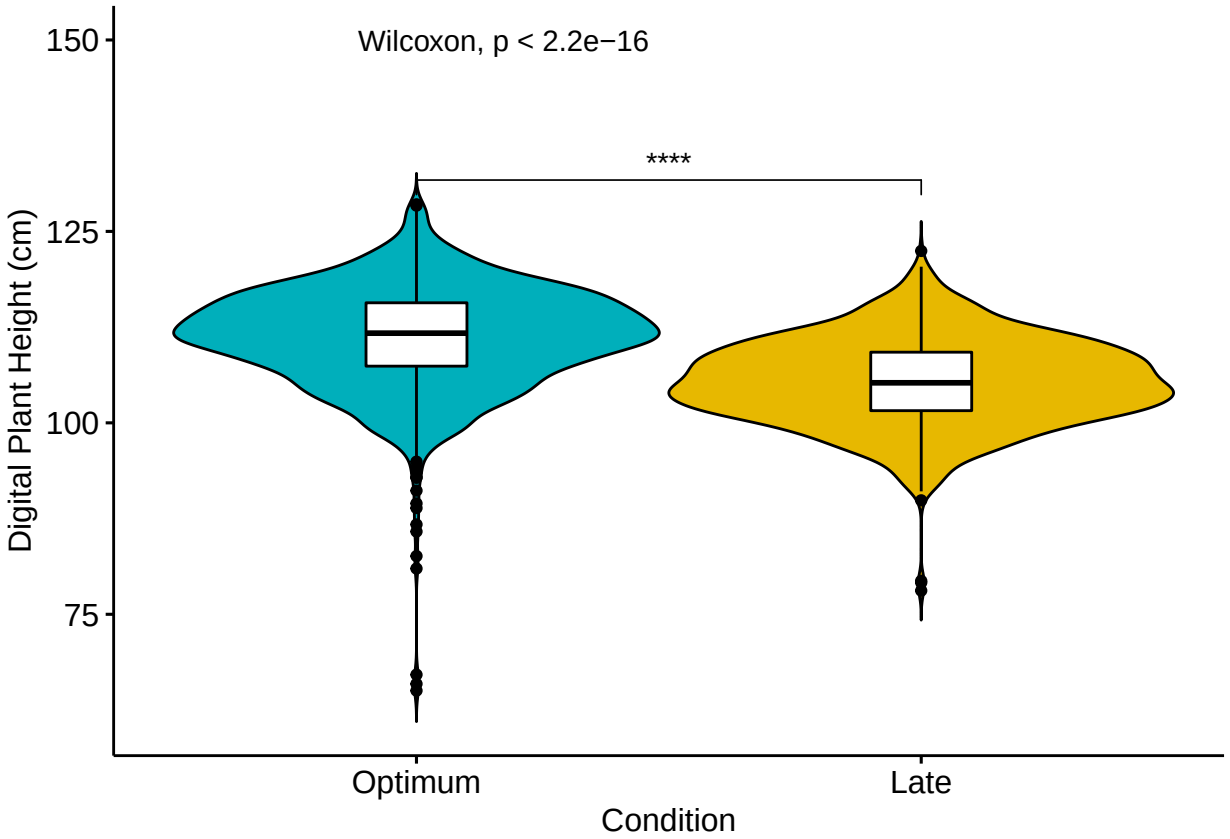
```
## # A tibble: 2 x 4
##   condition  phi1    phi2  phi3
##   <fct>     <dbl>   <dbl> <dbl>
## 1 Optimum    111. 0.00353 1178.
## 2 Late       105. 0.00440  975.
```

Looks like the late sown plants are gowing much faster in short time!

Wait. . . but, are these mean values statistically significant?

We can create violin plots and also perform mean comparisons for phi1 (maximum plant height) in a single step.

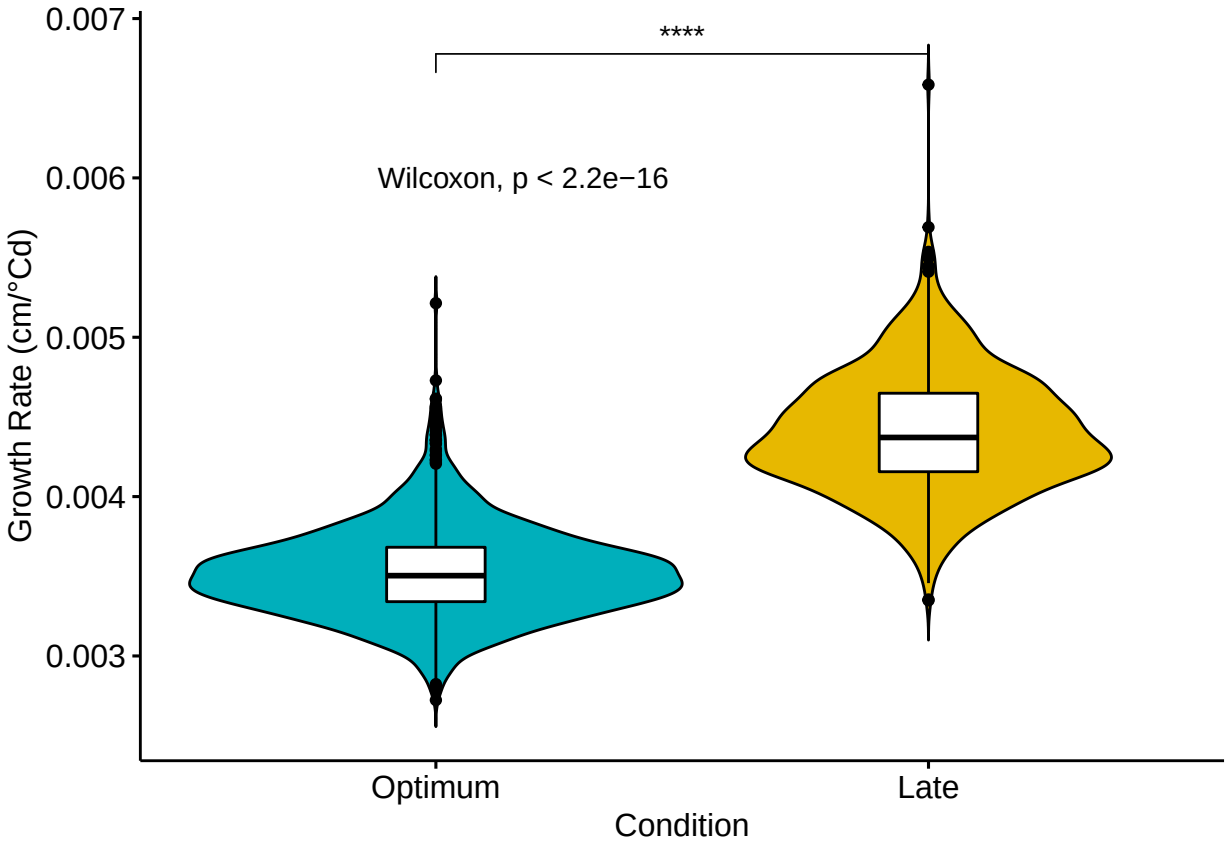
```
# Specify the comparisons you want
my_comparisons <- list( c("Optimum", "Late") )
# plot violins
dat.ht.growth %>% as_tibble() %>%
  ggviolin(., x = "condition", y = "phi1",
           fill = "condition",
           palette = c("#00AFBB", "#E7B800"), #"#FC4E07"),
           #facet.by = "trait_id",
           #scales = "free",
           add = "boxplot", add.params = list(fill = "white")) +
  stat_compare_means(comparisons = my_comparisons, label = "p.signif")+ #Add significance levels
  stat_compare_means(label.y = 150) +
  rremove("legend") +
  xlab("Condition") +
  ylab("Digital Plant Height (cm)")
```



What do you see in these plots? Does this figure answer our question of sowing date effect on plant height?

Violin plots for phi2 (height growth rate)

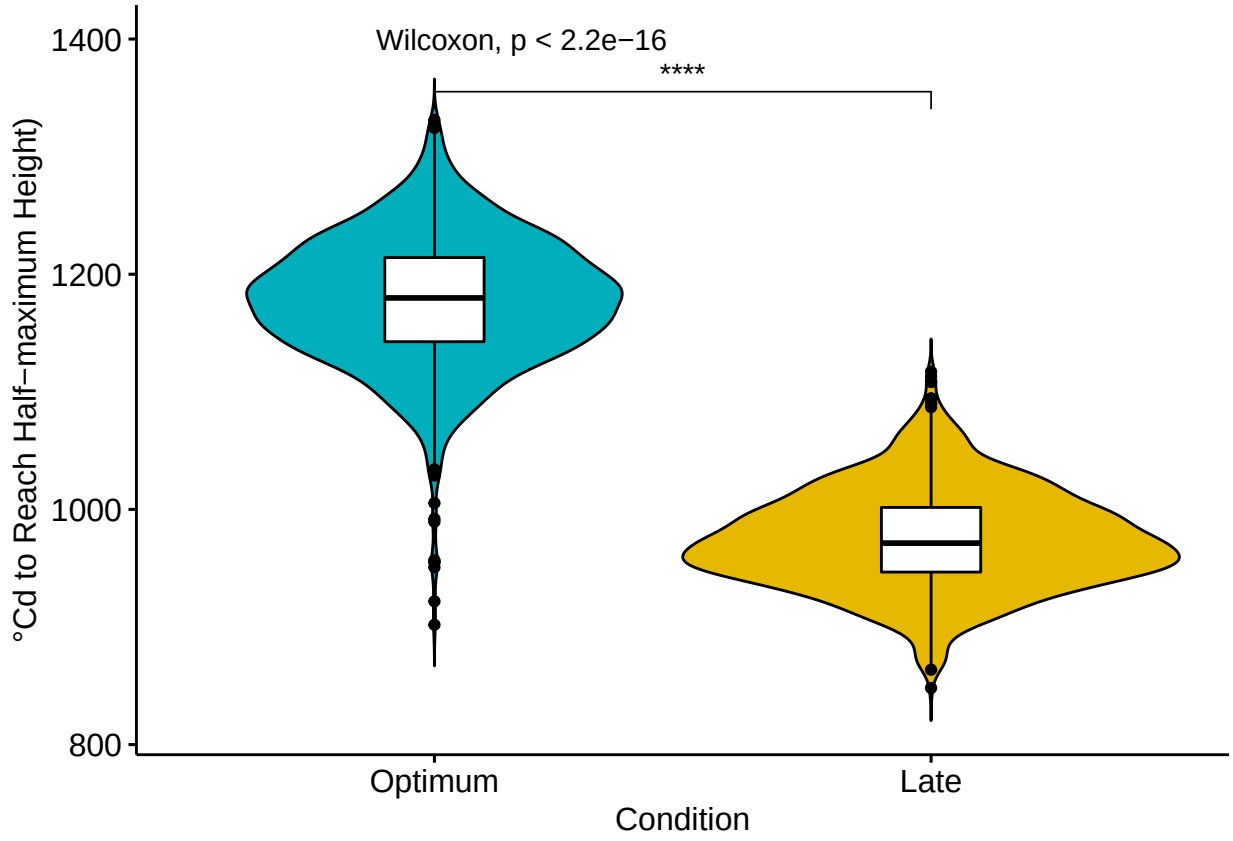
```
dat.ht.growth %>% as_tibble() %>%
  ggviolin(., x = "condition", y = "phi2",
           fill = "condition",
           palette = c("#00AFBB", "#E7B800"), #"#FC4E07"),
           #facet.by = "trait_id",
           #scales = "free",
           add = "boxplot", add.params = list(fill = "white")) +
  stat_compare_means(comparisons = my_comparisons, label = "p.signif")+ #Add significance levels
  stat_compare_means(label.y = 0.006) +
  rremove("legend") +
  xlab("Condition") +
  ylab("Growth Rate (cm/°Cd)")
```



Does the late planted crop grow faster? If so, any guess why?

Violin plots for phi3 (time to reach half maximum height).

```
## phi3 - plot violins with threshold
dat.ht.growth %>% as_tibble() %>%
  ggviolin(., x = "condition", y = "phi3",
           fill = "condition",
           palette = c("#00AFBB", "#E7B800"), #"#FC4E07"),
           #facet.by = "trait_id",
           #scales = "free",
           add = "boxplot", add.params = list(fill = "white")) +
  stat_compare_means(comparisons = my_comparisons, label = "p.signif")+ # Add significance levels
  stat_compare_means(label.y = 1400) +
  rremove("legend") +
  xlab("Condition") +
  ylab("°Cd to Reach Half-maximum Height)")
```



This plot is also consistent with what we see in previous two plots, i.e., the late sown crop grows significantly faster and take less time to reach to its half maximum height.

Concluding remarks

Through this tutorial you have familiarized yourself with a number of programming and statistical concepts namely: R functions, conditional workflows (if, else), tidy-data, exploratory data analysis, graphics, non-linear regression modeling, mean comparisons etc. However, always remeber that the tutorials such this one can only provide a primer or starting point- the real learning actually comes from a goal-oriented program exercises. Just like any other skill, programming is a game of regular practice. I would suggest you to pick a topic of your choice/interest and start playing with some example datasets; or better yet, if you have your own data try that out first.

As always, if you have any questions or interesting ideas to share, please free to reach out to me or other members of the K-State PBG journal club.

Happy Coding!!!

```
sessionInfo()
```

```
## R version 3.5.3 (2019-03-11)
## Platform: x86_64-w64-mingw32/x64 (64-bit)
## Running under: Windows 10 x64 (build 14393)
##
## Matrix products: default
##
## locale:
## [1] LC_COLLATE=English_United States.1252
## [2] LC_CTYPE=English_United States.1252
## [3] LC_MONETARY=English_United States.1252
## [4] LC_NUMERIC=C
## [5] LC_TIME=English_United States.1252
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
##  [1] rcompanion_2.2.2 ggpubr_0.2.1     magrittr_1.5     car_3.0-3
##  [5] carData_3.0-2    nlme_3.1-137     forcats_0.4.0    stringr_1.4.0
##  [9] dplyr_0.8.3      purrr_0.3.2      readr_1.3.1      tidyr_0.8.3
## [13] tibble_2.1.3     ggplot2_3.2.0    tidyverse_1.2.1  pacman_0.5.1
##
## loaded via a namespace (and not attached):
##  [1] matrixStats_0.54.0 lubridate_1.7.4    httr_1.4.0
##  [4] tools_3.5.3        backports_1.1.4    utf8_1.1.4
##  [7] R6_2.4.0           nortest_1.0-4      lazyeval_0.2.2
```

```
## [10] colorspace_1.4-1   withr_2.1.2         tidyselect_0.2.5
## [13] curl_4.0           compiler_3.5.3      cli_1.1.0
## [16] rvest_0.3.4        expm_0.999-4        xml2_1.2.0
## [19] sandwich_2.5-1     labeling_0.3        scales_1.0.0
## [22] lmtest_0.9-37      mvtnorm_1.0-11      multcompView_0.1-7
## [25] digest_0.6.20      foreign_0.8-71      rmarkdown_1.14
## [28] rio_0.5.16         pkgconfig_2.0.2     htmltools_0.3.6
## [31] manipulate_1.0.1   rlang_0.4.0         readxl_1.3.1
## [34] rstudioapi_0.10    generics_0.0.2      zoo_1.8-6
## [37] jsonlite_1.6       zip_2.0.3           modeltools_0.2-22
## [40] Matrix_1.2-17      fansi_0.4.0         Rcpp_1.0.1
## [43] DescTools_0.99.28  munsell_0.5.0       abind_1.4-5
## [46] stringi_1.4.3      multcomp_1.4-10     yaml_2.2.0
## [49] MASS_7.3-51.4      plyr_1.8.4          grid_3.5.3
## [52] parallel_3.5.3     crayon_1.3.4        lattice_0.20-38
## [55] haven_2.1.1        splines_3.5.3       hms_0.5.0
## [58] zeallot_0.1.0      knitr_1.23          pillar_1.4.2
## [61] EMT_1.1            boot_1.3-23         ggsignif_0.5.0
## [64] codetools_0.2-16   stats4_3.5.3        glue_1.3.1
## [67] evaluate_0.14      data.table_1.12.2   modelr_0.1.4
## [70] vctrs_0.2.0        cellranger_1.1.0    gtable_0.3.0
## [73] assertthat_0.2.1   xfun_0.8            openxlsx_4.1.0.1
## [76] coin_1.3-0         libcoin_1.0-4       broom_0.5.2
## [79] survival_2.44-1.1  TH.data_1.0-10
```