

Journal of Visualized Experiments

Heuristic mining of hierarchical genotypes and accessory genome loci in bacterial populations

--Manuscript Draft--

| | |
|--|---|
| Article Type: | Invited Methods Article - JoVE Produced Video |
| Manuscript Number: | JoVE63115R3 |
| Full Title: | Heuristic mining of hierarchical genotypes and accessory genome loci in bacterial populations |
| Corresponding Author: | Joao Carlos Gomes-Neto UNITED STATES |
| Corresponding Author's Institution: | |
| Corresponding Author E-Mail: | jgomesneto2@unl.edu |
| Order of Authors: | Joao Carlos Gomes-Neto Natasha Pavlovikj Andrew Benson |
| Additional Information: | |
| Question | Response |
| Please specify the section of the submitted manuscript. | Genetics |
| Please indicate whether this article will be Standard Access or Open Access. | Standard Access (\$1400) |
| Please indicate the city, state/province, and country where this article will be filmed . Please do not use abbreviations. | Lincoln, NE/USA |
| Please confirm that you have read and agree to the terms and conditions of the author license agreement that applies below: | I agree to the Author License Agreement |
| Please confirm that you have read and agree to the terms and conditions of the video release that applies below: | I agree to the Video Release |
| Please provide any comments to the journal here. | |

TITLE:

Heuristic Mining of Hierarchical Genotypes and Accessory Genome Loci in Bacterial Populations

AUTHORS AND AFFILIATIONS:

Natasha Pavlovikj^{1*}, Joao Carlos Gomes-Neto^{2,3*}, Andrew K. Benson^{2,3}

¹ Department of Computer Science and Engineering, University of Nebraska-Lincoln, Lincoln, Nebraska, United States of America

² Department of Food Science and Technology, University of Nebraska-Lincoln, Lincoln, Nebraska, United States of America

³ Nebraska Food for Health Center, University of Nebraska-Lincoln, Lincoln, Nebraska, United States of America

* These authors have contributed equally to this work and share first authorship

Email addresses of co-authors:

Natasha Pavlovikj (natasha.pavlovikj@huskers.unl.edu)

Joao Carlos Gomes-Neto (jgomesneto2@unl.edu)

Andrew K. Benson (abenson1@unl.edu)

Corresponding author:

Andrew K. Benson (abenson1@unl.edu)

SUMMARY:

This analytical computational platform provides practical guidance for microbiologists, ecologists, and epidemiologists interested in bacterial population genomics. Specifically, the work presented here demonstrated how to perform: i) phylogeny-guided mapping of hierarchical genotypes; ii) frequency-based analysis of genotypes; iii) kinship and clonality analyses; iv) identification of lineage differentiating accessory loci.

ABSTRACT:

Routine and systematic use of bacterial whole-genome sequencing (WGS) is enhancing the accuracy and resolution of epidemiological investigations carried out by Public Health laboratories and regulatory agencies. Large volumes of publicly available WGS can be used to study pathogenic populations at a large scale. Recently, a freely available computational platform called ProkEvo was published to enable reproducible, automated, and scalable hierarchical-based population genomic analyses using bacterial WGS data. This implementation of ProkEvo demonstrated the importance of combining standard genotypic mapping of populations with mining of accessory genomic content for ecological inference. In particular, the work highlighted here used ProkEvo-derived outputs for population-scaled hierarchical analyses using the R programming language. The main objective was to provide a practical guide for microbiologists, ecologists, and epidemiologists by showing how to: i) use a phylogeny-guided mapping of hierarchical genotypes; ii) assess frequency distributions of genotypes as a proxy for ecological fitness; iii) determine kinship relationships and genetic diversity using specific genotypic

classifications; and iv) map lineage differentiating accessory loci. To enhance reproducibility and portability, R markdown files were used to demonstrate the entire analytical approach. The example dataset contained genomic data from 2,365 isolates of the zoonotic foodborne pathogen *Salmonella* Newport. Phylogeny-anchored mapping of hierarchical genotypes (Serovar -> BAPS1 -> ST -> cgMLST) revealed the population genetic structure, highlighting sequence types (STs) as the keystone differentiating genotype. Across the three most dominant lineages, ST5 and ST118 shared a common ancestor more recently than with the highly clonal ST45 phylotype. ST-based differences were further highlighted by the distribution of accessory antimicrobial resistance (AMR) loci. Lastly, a phylogeny-anchored visualization was used to combine hierarchical genotypes and AMR content to reveal the kinship structure and lineage-specific genomic signatures. Combined, this analytical approach provides some guidelines for conducting heuristic bacterial population genomic analyses using pan-genomic information.

INTRODUCTION:

The increasing use of bacterial whole-genome sequencing (WGS) as a basis for routine surveillance and epidemiological inquiry by Public Health laboratories and regulatory agencies has substantially enhanced pathogen outbreak investigations¹⁻⁴. As a consequence, large volumes of de-identified WGS data are now publicly available and can be used to study aspects of the population biology of pathogenic species at an unprecedented scale, including studies based on: population structures, genotype frequencies, and gene/allele frequencies across multiple reservoirs, geographical regions, and types of environments⁵. The most commonly used WGS-guided epidemiological inquiries are based on analyses using only the shared core-genomic content, where the shared (conserved) content alone is used for genotypic classification (e.g., variant calling), and these variants become the basis for epidemiological analysis and tracing^{1,2,6,7}. Typically, bacterial core-genome-based genotyping is carried out with multi-locus sequence typing (MLST) approaches using seven to a few thousand loci⁸⁻¹⁰. These MLST-based strategies encompass mapping of pre-assembled or assembled genomic sequences onto highly curated databases, thereby combining allelic information into reproducible genotypic units for epidemiological and ecological analysis^{11,12}. For instance, this MLST-based classification can generate genotypic information at two levels of resolution: lower-level sequence types (STs) or ST lineages (7 loci), and higher-level core-genome MLST (cgMLST) variants (~ 300-3,000 loci)¹⁰.

MLST-based genotypic classification is computationally portable and highly reproducible between laboratories, making it widely accepted as an accurate sub-typing approach beneath the bacterial species level^{13,14}. However, bacterial populations are structured with species-specific varying degrees of clonality (i.e., genotypic homogeneity), complex patterns of hierarchical kinship between genotypes¹⁵⁻¹⁷, and a wide range of variation in the distribution of accessory genomic content^{18,19}. Thus, a more holistic approach goes beyond discrete classifications into MLST genotypes and incorporates the hierarchical relationships of genotypes at different scales of resolution, along with mapping of accessory genomic content onto genotypic classifications, which facilitates population-based inference^{18,20,21}. Moreover, analyses can also focus on shared patterns of inheritance of accessory genomic loci among even distantly-related genotypes^{21,22}. Overall, the combined approach enables agnostic interrogation of relationships between population structure and the distribution of specific genomic

compositions (e.g., loci) among geospatial or environmental gradients. Such an approach can yield both fundamental and practical information about the ecological characteristics of specific populations that may, in turn, explain their tropism and dispersion patterns across reservoirs, such as food animals or humans.

This systems-based hierarchical population-oriented approach demands large volumes of WGS data for sufficient statistical power to predict distinguishable genomic signatures. Consequently, the approach requires a computational platform capable of processing many thousands of bacterial genomes at once. Recently, ProkEvo was developed and is a freely available, automated, portable, and scalable bioinformatics platform that allows for integrative hierarchical-based bacterial population analyses, including pan-genomic mapping²⁰. ProkEvo allows for the study of moderate-to-large scale bacterial datasets while providing a framework to generate testable and inferable epidemiological and ecological hypotheses and phenotypic predictions that can be customized by the user. This work complements that pipeline in providing a guide on how to utilize ProkEvo-derived output files as input for analyses and interpretation of hierarchical population classifications and accessory genomic mining. The case study presented here utilized the population of *Salmonella enterica* lineage I zoonotic serovar S. Newport as an example and was specifically aimed at providing practical guidelines for microbiologists, ecologists, and epidemiologists on how to: i) use an automated phylogeny-dependent approach to map hierarchical genotypes; ii) assess the frequency distribution of genotypes as a proxy for evaluating ecological fitness; iii) determine lineage-specific degrees of clonality using independent statistical approaches; and iv) map lineage-differentiating AMR loci as an example of how to mine accessory genomic content in the context of the population structure. More broadly, this analytical approach provides a generalizable framework to perform a population-based genomic analysis at a scale that can be used to infer evolutionary and ecological patterns regardless of the targeted species.

PROTOCOL:

1. Install input files

NOTE: The protocol is available here -

https://github.com/jcgneto/jove_bacterial_population_genomics/tree/main/code. The protocol assumes that the researcher has specifically used ProkEvo (or a comparable pipeline) to get the necessary outputs available in this Figshare repository (<https://figshare.com/account/projects/116625/articles/15097503> - login credentials are required – The user must create a free account to have file access!). Of note, ProkEvo automatically downloads genomic sequences from the NCBI-SRA repository and only requires a .txt file containing a list of genome identifications as an input²⁰, and the one used for this work on S. Newport USA isolates is provided here (<https://figshare.com/account/projects/116625/articles/15097503?file=29025729>). Detailed information on how to install and use this bacterial genomics platform is available here (<https://github.com/npavloviki/ProkEvo/wiki/2.-Quick-start>)²⁰

1.1. Generate core-genome phylogeny using FastTree²³ as previously described²⁰, which is not part of the bioinformatics platform²⁰. FastTree requires the Roary²⁴ core-genome alignment as an input file. The phylogeny file is named newport_phylogeny.tree - <https://figshare.com/account/projects/116625/articles/15097503?file=29025690>).

1.2. Generate SISTR²⁵ output containing the information regarding serovars classifications for *Salmonella* and cgMLST variant calling data (sistr_output.csv - <https://figshare.com/account/projects/116625/articles/15097503?file=29025699>).

1.3. Generate BAPS file by fastbaps^{26,27} containing the BAPS levels 1-6 classification of genomes into sub-groups or haplotypes (fastbaps_partition_baps_prior_l6.csv - <https://figshare.com/account/projects/116625/articles/15097503?file=29025684>).

1.4. Generate MLST-based classification of genomes into STs using the MLST program (<https://github.com/tseemann/mlst>)²⁸ (salmonellast_output.csv - <https://figshare.com/account/projects/116625/articles/15097503?file=29025696>).

1.5. Generate ABRicate (<https://github.com/tseemann/abricate>)²⁹ output as a .csv file containing AMR loci mapped per genome (sabricate_resfinder_output.csv - <https://figshare.com/account/projects/116625/articles/15097503?file=29025693>).

NOTE: The user can turn off specific parts of the ProkEvo bioinformatics pipeline (check here for more information - <https://github.com/npavloviki/ProkEvo/wiki/4.2.-Remove-existing-bioinformatics-tool-from-ProkEvo>). The analytical approach presented here provides guidelines for how to conduct a population-based analysis after the bioinformatics pipeline has been run.

2. Download and install the statistical software and integrated development environment (IDE) application

2.1. Download the most up-to-date freely available version of the R software for Linux, Mac, or PC³⁰. Follow the default installation steps.

2.2. Download the most up-to-date freely available version of the RStudio desktop IDE here³¹. Follow the default steps for installation.

NOTE: The next steps are included in the available script, including detailed information of code utilization, and should be run sequentially to generate the outputs and figures presented in this work

(https://github.com/jcgneto/jove_bacterial_population_genomics/blob/main/code/data_analysis_R_code.Rmd). The user may decide to use another programming language to conduct this analytical/statistical analysis such as Python. In that case, use the steps in the scripts as a framework to carry out the analysis.

3. Install and activate data science libraries

177
178 3.1. Install all data science libraries at once as a first step in the analysis. Avoid installing the
179 libraries every time the script needs to be re-run. Use the function `install.packages()` for library
180 installation. Alternatively, the user may click on the **Packages** tab inside of the IDE and
181 automatically install the packages. The code used to install all needed libraries is presented
182 here:

```
183 # Install Tidyverse  
184 install.packages("tidyverse")  
185 # Install skimr  
186 install.packages("skimr")  
187 # Install vegan  
188 install.packages("vegan")  
189 # Install forcats  
190 install.packages("forcats")  
191 # Install naniar  
192 install.packages("naniar")  
193 # Install ggpubr  
194 install.packages("ggpubr")  
195 # Install ggrepel  
196 install.packages("ggrepel")  
197 # Install reshape2  
198 install.packages("reshape2")  
199 # Install RColorBrewer  
200 install.packages("RColorBrewer")  
201 # Install ggtree  
202 if (!requireNamespace("BiocManager", quietly = TRUE))  
203   install.packages("BiocManager")  
204 BiocManager::install("ggtree")  
205 # Installation of ggtree will prompt a question about installation - answer is "a" to  
206 install/update all dependencies
```

207
208 3.2. Activate all the libraries or packages using the `library()` function at the beginning of the
209 script, right after installation. Here is a demonstration on how to activate all necessary
210 packages:

```
211 # Activate the libraries and packages  
212 library(tidyverse)  
213 library(skimr)  
214 library(vegan)  
215 library(forcats)  
216 library(naniar)  
217 library(ggtree)  
218 library(ggpubr)  
219 library(ggrepel)  
220 library(reshape2)
```

```
221 library(RColorBrewer)
```

```
222
```

223 3.3. Suppress outputting the code used for library and package installation and activation by
224 using {r, include = FALSE} in the code chunk, as follows:

```
225 ```{r, include = FALSE}
```

```
226 # Install Tidyverse
```

```
227 install.packages("tidyverse")
```

```
228 ```
```

```
229
```

230 NOTE: This step is optional but avoids showing chunks of unnecessary code in the final html,
231 doc, or pdf report.

```
232
```

233 3.4. For a brief description of the specific functions of all libraries along with some useful
234 links to gather further information, refer to steps 3.4.1–3.4.11.

```
235
```

236 3.4.1. Tidyverse – use this collection of packages used for data science, including data entry,
237 visualization, parsing and aggregation, and statistical modeling. Typically, ggplot2 (data
238 visualization) and dplyr (data wrangling and modeling) are practical packages present in this
239 library³².

```
240
```

241 3.4.2. skimr – use this package for generating summary statistics of data frames, including
242 identification of missing values³³.

```
243
```

244 3.4.3. vegan – use this package for community ecology statistical analyses, such as calculating
245 diversity-based statistics (e.g., alpha and beta-diversity)³⁴.

```
246
```

247 3.4.4. forcats – use this package to work with categorical variables such as re-ordering
248 classifications. This package is part of the Tidyverse library³².

```
249
```

250 3.4.5. naniar – use this package to visualize the distribution of missing values across variables
251 in a data frame, by using the viss_miss() function³⁵.

```
252
```

253 3.4.6. ggtree – use this package for the visualization of phylogenetic trees³⁶.

```
254
```

255 3.4.7. ggpubr – use this package to improve the quality of ggplot2-based visualizations³⁷.

```
256
```

257 3.4.8. ggrepel – use this package for text labeling inside of graphs³⁸.

```
258
```

259 3.4.9. reshape2 – use the melt() function from this package for the transformation of data
260 frames from wide to long format³⁹.

```
261
```

262 3.4.10. RColorBrewer – use this package to manage colors in ggplot2-based visualizations⁴⁰.

```
263
```

264 3.4.11. Use the following basic functions for exploratory data analysis: head() to check the first

observations in a data frame, `tail()` to check the last observations of a data frame, `is.na()` to count the number of rows with missing values across a data frame, `dim()` to check the number of rows and columns in a dataset, `table()` to count observations across a variable, and `sum()` to count the total number of observations or instances.

4. Data entry and analysis

NOTE: A detailed information on each step of this analysis can be found in the available script (https://github.com/icgneto/iove_bacterial_population_genomics/blob/main/code/data_analysis_R_code.Rmd). However, here are some important points to be considered:

4.1. Do all genomic data entry, including all genotypic classifications (serovar, BAPS, ST, and cgMLST) using the `read_csv()` function.

4.2. Rename, create new variables, and select columns of interest from each dataset before multi-dataset aggregation.

4.3. Don't remove missing values from any independent dataset. Wait until all datasets are aggregated to modify or exclude missing values. If new variables are created for each dataset, then missing values are by default categorized into one of the newly generated classifications.

4.4. Check for erroneous characters such as hyphens or interrogations marks and replace them with NA (Not applicable). Do the same for missing values.

4.5. Aggregate data based on the hierarchical order of genotypes (serovar -> BAPS1 -> ST -> cgMLST), and by grouping based on the individual genome identifications.

4.6. Check for missing values using multiple strategies and deal with such inconsistencies explicitly. Only remove a genome or isolate from the data if the classification is unreliable. Otherwise, consider the analysis being done and remove NAs on a case-by-case basis.

NOTE: It is highly recommended to establish a strategy to deal with such values *a priori*. Avoid removing all genomes or isolates with missing values across any variables. For instance, a genome may have ST classification without having cgMLST variant number. In that case, the genome can still be used for the ST-based analysis.

4.7. Once all datasets are aggregated, assign them to a data frame name or object that can be used in multiple locations in the follow-up analysis, to avoid having to generate the same metadata file for every figure in the paper.

5. Conduct analyses and generate visualizations

NOTE: A detailed description of each step needed to produce all the analysis and visualizations can be found in the markdown file for this paper

(https://github.com/jcgneto/jove_bacterial_population_genomics/tree/main/code). Code for each figure is separated in chunks and the entire script should be run sequentially. Additionally, the code for each main and supplementary figure is provided as a separate file (see **Supplementary File 1** and **Supplementary File 2**). Here are some essential points (with snippets of code) to be considered while generating each main and supplementary figures.

5.1. Use ggtree to plot a phylogenetic tree along with genotypic information (**Figure 1**).

5.1.1. Optimize the ggtree figure size, including diameter and width of rings, by changing the numerical values inside of the xlim() and gheatmap(width =) functions, respectively (see example code below).

```
tree_plot <- ggtree(tree, layout = "circular") + xlim(-250, NA)
figure_1 <- gheatmap(tree_plot, d4, offset=.0, width=20, colnames = FALSE)
```

NOTE: For a more detailed comparison of programs that can be used for phylogenetic plotting, check this work²⁰. The work highlighted an attempt made to identify strategies to improve ggtree-based visualizations such as decreasing the dataset size, but branch lengths and tree topology were not as clearly discriminating as compared to phandango⁴¹.

5.1.2. Aggregate all metadata into as few categories as possible to facilitate the choice of coloring panel when plotting multiple layers of data with the phylogenetic tree (https://github.com/jcgneto/jove_bacterial_population_genomics/blob/main/code/figure_1.R.md). Conduct the data aggregation based on the question of interest and domain knowledge.

5.2. Use a bar plot to assess relative frequencies (**Figure 2**).

5.2.1. Aggregate data for both ST lineages and cgMLST variants to facilitate visualizations. Choose an empirical or statistical threshold used for data aggregation, while considering the question being asked.

5.2.2. For an example code that can be used to inspect the frequency distribution of ST lineages to determine the cut-off see below:

```
st_dist <- d2 %>% group_by(ST) %>% # group by the ST column
count() %>% # count the number of observations
arrange(desc(n)) # arrange the counts in decreasing order
```

5.2.3. For an example code showing how minor (low-frequency) STs can be aggregated refer below. As demonstrated below, STs that are not numbered as 5, 31, 45, 46, 118, 132, or 350, are grouped together as "Other STs". Use a similar code for cgMLST variants (https://github.com/jcgneto/jove_bacterial_population_genomics/blob/main/code/figure_2.R.md).

```
d2$st <- ifelse(d2$ST == 5, "ST5", # create a new ST column for which minor STs are aggregated
as Others
ifelse(d2$ST == 31, "ST31",
```

```

353     ifelse(d2$ST == 45, "ST45",
354           ifelse(d2$ST == 46, "ST46",
355                 ifelse(d2$ST == 118, "ST118",
356                       ifelse(d2$ST == 132, "ST132", ifelse(d2$ST == 350, "ST350", "Other STs"))))))))
357

```

5.3. Use a nested approach to calculate the proportion of each ST lineage within each BAPS1 sub-group to identify STs that are ancestrally related (belong to the same BAPS1 sub-group) (Figure 3). The code below exemplifies how the ST-based proportion can be calculated across BAPS1 sub-groups

(https://github.com/icgneto/jove_bacterial_population_genomics/blob/main/code/figure_3.R md):

```

364 baps <- d2b %>% filter(serovar == "Newport") %>% # filter Newport serovars select(baps_1, ST)
365 %>% # select baps_1 and ST columns
366 mutate(ST = as.numeric(ST)) %>% # change ST column to numeric
367 drop_na(baps_1, ST) %>% # drop NAs
368 group_by(baps_1, ST) %>% # group by baps_1 and ST
369 summarise(n = n()) %>% # count observations
370 mutate(prop = n/sum(n)*100) # calculate proportions
371

```

5.4. Plot the distribution of AMR loci across ST lineages using the Resfinder-based gene annotation results (Figure 4).

NOTE: Resfinder has been widely used in ecological and epidemiological studies⁴². Annotation of protein-coding genes can vary depending on how often databases are curated and updated. If using the suggested bioinformatics pipeline, the researcher can compare AMR-based loci classifications across different databases²⁰. Be sure to check which databases are continually being updated. Do not use out-of-date or poorly curated databases, in order to avoid miscalls.

5.4.1. Use an empirical or statistical threshold to filter out the most important AMR loci to facilitate visualizations. Provide a raw .csv file containing the calculated proportions of all AMR loci across all ST lineages, such as shown here (<https://figshare.com/account/projects/116625/articles/15097503?file=29025687>).

5.4.2. Calculate the AMR proportion for each ST using the following code (https://github.com/icgneto/jove_bacterial_population_genomics/blob/main/code/figure_4.R md):

```

389 # Calculations for ST45
390 d2c <- data6 %>% filter(st == "ST45") # filter ST45 data first
391 # for ST45, calculate the proportion of AMR loci and only keep proportion greater than 10%
392 d3c <- d2c %>% select(id, gene) %>% # select columns
393 group_by(id, gene) %>% # group by id and gene
394 summarize(count = n()) %>% # count observations
395 mutate(count = replace(count, count == 2, 1)) %>% # replace counts equal to 2 with 1 to only
396 consider one copy of each gene (duplications may not be reliable), but the researcher can decide

```

to exclude or keep them. If the researcher wants to exclude them, then use the filter(count != 2) function or else leave as is

```
filter(count <= 1) # filter counts below or equal to 1
d4c <- d3c %>% group_by(gene) %>% # group by gene
summarize(value = n()) %>% # count observations
mutate(total = table(data1$st)[6]) %>% # get the total counts of st mutate(prop =
(value/total)*100) # calculate proportions
d5c <- d4c %>% mutate(st = "ST45") # create a st column and add ST information
```

5.4.3. After calculations are done for all STs, combine datasets as one data frame, using the following code:

```
# Combine datasets
d6 <- rbind(d5a, d5b, d5c, d5d, d5e, d5f, d5g, d5h) # row bind datasets
```

5.4.4. To export the .csv file containing the calculated proportions, use the code:

```
# Export data table containing ST and AMR loci information
abx_newport_st <- d6 write.csv(abx_newport_st, "abx_newport_st.csv", row.names = FALSE)
```

5.4.5. Before plotting the AMR-based distribution across ST lineages, filter the data based on a threshold to facilitate visualizations, as shown below:

```
# Filter AMR loci with proportion higher than or equal to 10%
d7 <- d6 %>% filter(prop >= 10) # determine the threshold empirically or statistically
```

5.5. Plot the core-genome phylogeny along with the hierarchical genotypic classifications and AMR data in a single plot using ggtree (**Figure 5**).

5.5.1. Optimize the figure size inside ggtree using the abovementioned parameters (see step 5.1.1.).

5.5.2. Optimize visualizations by aggregating variables, or using binary classification such as gene presence or absence. The more features are added to the plot, the harder the coloring selection process becomes. (https://github.com/jcgneto/jove_bacterial_population_genomics/blob/main/code/figure_5.Rmd).

NOTE: Supplementary figures - detailed description of the entire code can be found here ([https://github.com/jcgneto/jove_bacterial_population_genomics/blob/main/code/data analysis R code.Rmd](https://github.com/jcgneto/jove_bacterial_population_genomics/blob/main/code/data_analysis_R_code.Rmd)):

5.6. Use a scatter plot in ggplot2, without data aggregation, to display the distribution of ST lineages or cgMLST variants while highlighting the most frequent genotypes (**Supplementary Figure 1**) ([https://github.com/jcgneto/jove_bacterial_population_genomics/blob/main/code/supplementary figure s1.Rmd](https://github.com/jcgneto/jove_bacterial_population_genomics/blob/main/code/supplementary_figure_s1.Rmd)).

5.7. Do a nested analysis to assess the composition of ST lineages through the proportion of cgMLST variants in order to get a glimpse of the ST-based genetic diversity, while identifying the most frequent variants and their genetic relationships (i.e., cgMLST variants that belong to the same ST shared an ancestor more recently than those belonging to distinct STs)

(Supplementary Figure 2)

(https://github.com/jcgneto/jove_bacterial_population_genomics/blob/main/code/supplementary_figure_s2.Rmd).

5.8. Use community ecology metric, namely Simpson's D index of diversity, to measure the degree of clonality or genotypic diversity of each of the major ST lineages⁴³ **(Supplementary Figure 3)**.

5.8.1. Calculate the index of diversity across ST lineages at different levels of genotypic resolution including BAPS level 1 through 6 and cgMLST. Below is the code example on how to do this calculation at the BAPS level 1 (BAPS1) of genotypic resolution:

```
# BAPS level 1 (BAPS1)
# drop the STs and BAPS1 with NAs, group by ST and BAPS1 and then calculate Simpson's index
baps1 <- data6 %>%
select(st, BAPS1) %>% # select columns
drop_na(st, BAPS1) %>% # drop NAs
group_by(st, BAPS1) %>% # group by columns
summarise(n = n()) %>% # count observations
mutate(simpson = diversity(n, "simpson")) %>% # calculate diversity
group_by(st) %>% # group by column
summarise(simpson = mean(simpson)) %>% # calculate the mean of the index
melt(id.vars=c("st"), measure.vars="simpson",
variable.name="index", value.name="value") %>% # covert into long format
mutate(strat = "BAPS1") # create a strat column
```

NOTE: A more genetically diverse population (i.e., more variants at different layers of genotypic resolution) has a higher index at the cgMLST level and produces an increasing index-based values going from BAPS level 2 to 6

(https://github.com/jcgneto/jove_bacterial_population_genomics/blob/main/code/supplementary_figure_s3.Rmd).

5.9. Examine the degree of genotypic diversity of ST lineages by plotting the relative frequency of BAPS sub-groups at all levels of resolution (BAPS1-6) **(Supplementary Figure 4)**. The more diverse the population is, the sparser the distribution of BAPS sub-groups (haplotypes) becomes going from BAPS1 (lower level of resolution) to BAPS6 (higher level of resolution)

(https://github.com/jcgneto/jove_bacterial_population_genomics/blob/main/code/supplementary_figure_s4.Rmd).

REPRESENTATIVE RESULTS:

By utilizing the computational platform ProkEvo for population genomics analyses, the first step in bacterial WGS data mining is comprised of examining the hierarchical population structure in the context of a core-genome phylogeny (**Figure 1**). In the case of *S. enterica* lineage I, as exemplified by the *S. Newport* dataset, the population is hierarchically structured as follows: serovar (lowest level of resolution), BAPS1 sub-groups or haplotypes, ST lineages, and cgMLST variants (highest level of resolution)²⁰. This phylogeny-guided analysis of the hierarchical population structure specifically allows for the following points to be examined: i) phylogenetic distribution of SISTR-based misclassified genomes into other serovars in the case of *Salmonella*; ii) genetic or kinship structure of the population; iii) pattern of diversification at different levels of genotypic resolution; iv) identification of major genotypic unit(s) underlying an evolutionary, ecological, or epidemiological patterns; v) ancestral relationships between ST lineages through BAPS1 sub-groups or haplotype composition, and across cgMLST variants within ST lineages; and vi) partial view of the degree of genotypic homogeneity of an ST lineage by the cgMLST variant composition.

[Place **Figure 1** here]

Relative frequencies of all hierarchical genotypes were then used to evaluate the overall distribution and most frequently observed classifications (i.e., genotypes) (**Figure 2**). In **Figure 2C–D**, less frequent (minor) ST lineages or cgMLST variants were aggregated as “Other STs” or “Other cgMLSTs”, respectively, in order to facilitate data visualization (dimensionality reduction). If sampling is systematically done across environments and/or hosts and is appropriately statistically powered, frequency distribution can become a proxy for ecological fitness. That is, the most frequent lineages or variants could then be predicted to have higher fitness, ensuing further investigation to determine the causative genetic determinants underlying such a quantitative trait^{6,30}.

[Place **Figure 2** here]

Alternatively, a scatter-plot can be used to assess the distribution and proportion of both ST lineages or cgMLST variants, without any data aggregation (**Supplementary Figure 1**). This use of a scatter-plot is particularly useful for ST lineages and cgMLST variants because of the typical occurrence of hundredths, if not thousands, classifications for both genotypes. This sparse distribution commonly does not occur for the serovar and BAPS1 levels of resolution, because they are at a lower level of resolution with sequences inheritably collapsing into a few sub-groups or categories.

Next, the ancestral relationships between STs were examined using a nested approach which encompasses assessing the relative frequency of ST lineages by BAPS1 sub-groups or haplotypes (**Figure 3**). ST lineages that belonged to the same BAPS1 sub-group were more likely to have shared a common ancestor more recently than with other STs (i.e., ST5 and ST118 vs. ST45). Similarly, by examining the distribution of cgMLST variants within ST lineages, the degree of

genotypic heterogeneity across STs can be captured, while assessing their genetic composition and revealing the ancestral relationship between cgMLSTs (i.e., closely related cgMLST variants belong to the same ST lineage or clonal complex) (**Supplementary Figure 2**).

[Place **Figure 3** here]

Given that the pattern of *S. Newport* population diversification appeared to be mostly driven by ST composition (**Figure 1**), two statistical approaches were used to assess the ST-based degree of clonality (i.e., genetic homogeneity), including Simpson's D index of diversity (**Supplementary Figure 3**), and the distribution of BAPS sub-groups or haplotypes using BAPS levels 1-6 (**Supplementary Figure 4**). Assessing the degree of clonality of a population can elucidate the following aspects: i) a better understanding of genetic diversity and population structure; ii) fine-tuning analysis of patterns of diversification across major genotypic units such as ST lineages; and iii) be an indicator of the necessity of using accessory genome mining to find cryptic genotypic units that may reveal novel sub-clusters present in the population. The more clonal a population is at the core-genome level, the harder it becomes to differentiate between variants, and the more likely the accessory genome content will be informative to stratify the population into meaningful genotypic units associated with unique ecological distributions^{18,19,21}.

The relative frequency of ST lineage differentiating AMR loci was assessed to identify unique accessory genomic signatures linked to the *S. Newport* population structure (**Figure 4**). This step of the analysis was focused on AMR distribution because it is a Public Health-associated trait, but the same approach can be applied in a supervised (targeted) or agnostic fashion to examine other components of the accessory genome, including metabolic pathways, virulence factors, etc. Noticeably, *mdf(A)_1* and *aac(6')-laa_1* loci appear to be ancestrally-acquired by the *S. Newport* population; whereas, ST45 is predicted to be multi-drug resistant. Strikingly, these data also suggest that the other major ST lineages, ST5 and ST118, are more likely to be multi-drug susceptible when compared to ST45. These points have to be carefully considered because of the biases present in the dataset; however, this represents a potential epidemiological inference that could be made from more robust WGS data collections.

In general, here are some points to be considered when conducting an accessory genome mapping onto hierarchical genotypes: i) consider the frequency distribution as a quantitative trait but be aware that the allelic composition of a locus can alter trait variance. Moreover, the presence of a locus or loci should be indicative of function but not causal, because the phenotype may be polygenic, or vary according to the allelic composition for the causative locus (e.g., a non-synonymous mutation on the active site of a protein is more likely to affect function); ii) loci distribution can demonstrate genes that are fixed in the population (e.g., found in high frequency across all ST lineages) or recently-acquired by specific ST lineages and cgMLST variants, and may reflect the ecological or epidemiological pattern; iii) multi-drug resistance can be predicted from genomics data. And if the distribution of AMR loci, or other pathways, is strongly linked or commonly inherited by specific lineages, then phenotypes can be predicted by inference from hierarchical genotypes, such as in the case of ST lineages^{45,46}; and iv) measuring phenotypes in the laboratory is still deterministic to validate computational predictions.

[Place **Figure 4** here]

Lastly, a phylogeny-anchored visualization was used to systematically integrate the hierarchical population structure data along with ST lineage differentiating AMR loci distribution based on gene occurrence (**Figure 5**). By combining the population structure along with accessory genomic composition, the following set of questions can be addressed in any given dataset: 1) How is the population structured? How do STs relate to each other and ancestrally through BAPS1 sub-groups? How variable is the cgMLST composition across STs? 2) What is the phylogenetic branching pattern and overall tree topology? and 3) How is the accessory genome distributed? Is the accessory genomic composition mostly likely ancestrally-acquired or recently-derived? What is the lineage or variant-specific pattern? What is the phenotypic prediction and ecological inference? Is there niche-transcending vs. niche-specifying genes? How does the observed pattern relate or inform the epidemiology in the case of pathogens? Can lineages or variants be informatively sub-clustered based on accessory genomic content?

[Place **Figure 5** here]

FIGURE AND TABLE LEGENDS:

Figure 1: Phylogeny-guided mapping of hierarchical genotypes for the *S. Newport* population.

A core-genome phylogeny (black centered circle) was used to map hierarchical genotypes, including serovar (lowest level of resolution – innermost colored circle), BAPS level 1 (BAPS1) sub-groups or haplotypes, ST lineages, and cgMLST variants (highest level of resolution – outermost colored circle). Serovars were grouped into Newport (*S. Newport*) or “Other serovars” based on the SISTR algorithmic classification of genomes, which utilized core-genome MLST information, and ran as part of the computational platform ProkEvo. BAPS1 agnostically stratifies the population into sub-groups or clusters of related haplotypes using core-genomic data within ProkEvo. BAPS1 is hierarchically placed between serovar and ST lineages because it accurately captured the ancestral relationships between STs. ST lineages are formed based on canonical MLST analysis using seven genome-scattered loci. Only major or most frequent STs (proportion >1%) were depicted in the graph. Lastly, only the most frequent cgMLST variants (proportion >3.5%) were used to show the entire hierarchical structure for the *S. Newport* population (n = 2,365 USA isolates only). The category “Other STs” or “Other cgMLSTs” comprised of minor or low-frequency lineages or variants, respectively, with thresholding done arbitrarily that should be set empirically or statistically based on the dataset.

Figure 2: Proportion of *S. Newport* hierarchical genotypes at different levels of resolution. (A)

Serovars are phenotypes of the *S. enterica* lineage I population that can be predicted solely from core-genomic data due to the inheritable high linkage disequilibrium between core-loci and O and H antigenic-coding loci (surface proteins). When using ProkEvo, *Salmonella* genomes are automatically classified to serovars using the SISTR program. Although only *S. Newport* (Newport) genomes from NCBI were putatively downloaded, some have been classified as “Other serovars” within ProkEvo. Approximately 2% (48 out of 2,365) of all genomes were classified as other than *S. Newport* serovar. **(B)** The proportion of BAPS level 1 (BAPS1) sub-groups or haplotypes. BAPS1

is inserted between serovar and ST lineages in the hierarchical scheme because it accurately and agnostically captured the ancestral relationships between STs. (C) The proportion of major ST lineages depicted only STs that were > 1% in relative frequency. Minor STs were grouped as “Other STs”. (D) The proportion of major cgMLST variants showed only four predominant cgMLSTs that were >3% in relative frequency. The remainder cgMLSTs were grouped as “Other cgMLSTs”. (B–D) Genomes classified by SISTR as “Other serovars” (2.03%) were filtered out of the data prior to plotting BAPS1, ST, and cgMLST relative frequencies. (C–D) Thresholds used to plot both ST and cgMLST data were arbitrarily defined and should be established empirically on a case-by-case basis.

Figure 3: Distribution of ST lineages nested within BAPS1 sub-groups for the *S. Newport* population. This plot depicts the ST lineage distribution within each BAPS level 1 sub-group or haplotype, excluding genomes classified as “Other serovars” (2.03% of the entire data). Major STs (proportion >1%) for each BAPS1 sub-group are highlighted in each graph. The larger the circle diameter, the higher the proportion for the particular ST lineage.

Figure 4: Distribution of AMR loci across major ST lineages of the *S. Newport* population. Relative frequency-based distribution of a selected number of AMR loci across major ST lineages (>1% of the population). Minor STs were grouped as “Other STs”. Only genomes classified as *S. Newport* by the SISTR algorithm were kept in the analysis. AMR loci with a relative frequency greater than or equal to 10% were selected for data visualization. This is an arbitrary threshold that should be determined for each dataset. The proportions were calculated using a binary matrix composed of gene presence or absence.

Figure 5: Phylogeny-guided mapping of hierarchical genotypes and accessory AMR loci differentiating between major ST lineages within the *S. Newport* population. A core-genome phylogeny (black centered circle) was used to map hierarchical genotypes, including serovar (lowest level of resolution – innermost colored circle), BAPS level 1 (BAPS1) sub-groups or haplotypes, ST lineages, and cgMLST variants (highest level of resolution – outermost colored circle), along with AMR loci colored as dark-blue if present or gray if absent. Serovars were grouped into *Newport* (*S. Newport*) or “Other serovars” based on the SISTR algorithmic classification. BAPS1 is hierarchically placed between serovar and ST lineages because it accurately and agnostically captured the ancestral relationships between STs. ST lineages are formed based on canonical MLST analysis using seven genome-scattered loci. Only major or most frequent STs (proportion >1%) were depicted in the graph. Also, only the most dominant cgMLST variants (proportion >3.5%) were used to show the entire hierarchical structure for the *S. Newport* population (n = 2,365 USA isolates only). The category “Other STs” or “Other cgMLSTs” comprised of minor or low-frequency lineages or variants, respectively, and thresholding was done arbitrarily and should be set based on the dataset. AMR loci with a relative frequency greater than or equal to 10% were selected for data visualization. This specific graph shows a unique distribution of AMR loci predominantly occurring in ST31, ST45, and ST132 lineages.

Supplementary Figure 1: Sparse distribution of ST lineages and cgMLST variants for the *S. Newport* population. (A) The proportion of ST lineages without aggregating low-frequency STs.

STs with proportion >1% are highlighted in the plot. **(B)** The proportion of cgMLST variants without aggregating low-frequency cgMLSTs. cgMLSTs with proportion > 3% are highlighted in the plot. **(A-B)** Thresholds used to plot both ST and cgMLST data were arbitrarily defined and should be established based on the dataset. Genomes classified by SISTR as “Other serovars” (2.03%) were filtered out of the data prior to plotting both ST and cgMLST relative frequencies. The larger the circle diameter, the higher the proportion for either the ST lineage or cgMLST variant.

Supplementary Figure 2: Distribution of cgMLST variants nested within ST lineages for the S. Newport population. This plot depicts the cgMLST variant distribution across ST lineages, excluding genomes classified as “Other serovars” (2.03% of the entire data). Major cgMLSTs (proportion >15%) for each ST lineage are highlighted in each graph. The larger the circle diameter, the higher the proportion for the specific cgMLST variant. Low-frequency STs were grouped as “Other STs”.

Supplementary Figure 3: Simpson’s D-based degree of genetic diversity across ST lineages using BAPS levels 1-6 haplotypes or cgMLST genotypes as input data for the S. Newport population. The degree of clonality or genetic diversity of each ST lineage was calculated across different genotypic layers of resolution, including BAPS levels 1 (lowest level of resolution) to 6 (highest level of resolution) sub-groups or haplotypes, and by additionally using the cgMLST-based distribution of variants. The higher the index value, the higher the degree of genetic diversity. Highly diverse ST lineages have higher index values going from BAPS1 to BAPS6 (i.e., typically the index increases and eventually plateaus when going from BAPS1 to BAPS6). Only genomes classified as S. Newport by the SISTR program were kept in the analysis. Low-frequency STs were grouped as “Other STs”.

Supplementary Figure 4: Distribution of BAPS levels 1-6 sub-groups or haplotypes across major ST lineages of the S. Newport population. Relative frequency-based distribution of BAPS sub-groups or haplotypes, across major ST lineages, from the lowest (BAPS1) to the highest level of resolution (BAPS6). Major STs were selected based on having a proportion >1%. Only genomes classified as S. Newport by the SISTR program were kept in the analysis. The higher the degree of clonality, the less sparse or spread the distribution of BAPS sub-groups or haplotypes becomes when going from BAPS1 to BAPS6. In other words, a more genetically diverse ST lineage has a wider range of BAPS sub-groups at the BAPS level 6 (highest degree of resolution). Low-frequency STs were grouped as “Other STs”.

Supplementary File 1: Links to material list and genomes list

Supplementary File 2: Hierarchical-based bacterial population genomics analysis using R

DISCUSSION:

The utilization of a systems-based heuristic and hierarchical population structure analysis provides a framework to identify novel genomic signatures in bacterial datasets that have the potential to explain unique ecological and epidemiological patterns²⁰. Additionally, the mapping

of accessory genome data onto the population structure can be used to infer ancestrally-acquired and/or recently-derived traits that facilitate the spread of ST lineages or cgMLST variants across reservoirs^{6,20,21,45,46}. More broadly, a global assessment of pan-genomic content distribution in bacterial populations can reveal patterns of diversification that underly the ecological tropisms or geospatial/temporal bottlenecks that a population might have recently withstood^{18,21}. In the case of pathogenic species, by mining the population structure of clinical vs. environmental isolates, genetic determinants associated with zoonotic events may be identified and used to improve diagnostics and surveillance^{33,34}. The same approach can be applied to non-pathogenic species to identify genotypes with desirable niche-specific engrafting properties, as in the case of gastrointestinal probiotic strains used to improve human health^{49–51}. Yet, the utilization of bacterial WGS data for population-based inquiries requires the use of reproducible, automated, and scalable computational platforms like ProkEvo²⁰. Any computational approach comes with its caveats and nuances, but in general, freely available, well-documented, portable, and user-friendly platforms such as ProkEvo can facilitate the work of microbiologists, ecologists, and epidemiologists doing heuristic bacterial population-based genomics.

In the present work, it was demonstrated how to use ProkEvo-derived outputs to conduct a hierarchical population structure analysis that can be used to map and track genotypes of interest at different levels of resolution, along with predicting useful traits from WGS data. This computational protocol was written using the R programming language, but the framework or conceptual approach is generalizable to other languages such as Python through the utilization of the Pandas library, for instance. The input data is generated by ProkEvo²⁰, which prevents some hurdles to be faced in terms of standardizing outputs and data formats for subsequent analysis. With the exception of phylogenies, all other input datasets come in a tabular format that can easily be quality-controlled, aggregated, parsed, and integrated to generate useful reports for data interpretation. However, it is important to highlight a few critical steps to enhance reproducibility while using this protocol: i) make sure the software versions are always updated and tracked; ii) track the versions of the data science libraries being used, and preferably update them over time; iii) quality-control the data using domain knowledge expertise to make sense of the outputs generated by ProkEvo, or a similar pipeline, in light of what is understood for the targeted bacterial population; iv) conduct an exploratory data analysis prior to using any modeling approach; v) aggregate the data based on empirical knowledge and/or statistical assessments; vi) define a strategy to deal with missing values *a priori* and be consistent and completely transparent about it; vii) if using R, try to use all the packages provided by Tidyverse, because this collection facilitates functional programming, portability, optimization, and is freely available; and viii) be aware that visualization approaches can be difficult because it takes some trial and error to get the right type of plot and coloring scheme that is most appropriately applicable for the question being asked and the data being portrayed.

Of note, this protocol comes with some limitations that can be further improved. For instance, ProkEvo has an intrinsic limit to how many genomes can be used for pan-genomic analysis, if the core-genome alignment step is generated concomitantly, while utilizing the Roary program (~ 2,000–3,000 genomes)²⁴. That is a very specific bottleneck in the pipeline that will affect the number of genomes that can be classified into BAPS haplotypes since it depends on core-genome

alignment (i.e., highly computationally demanding step). However, core-genome alignment can be done with other programs⁵², and such algorithms, in theory, could be easily incorporated into ProkEvo. Otherwise, datasets can be strategically split into random subsets, or in another basis such as by considering the population structure of the organism in question. Alternatively, ProkEvo can be run with a single genome to get ST-based annotation, antibiotic resistance and virulence gene composition, and mapping of plasmids, but the pipeline was designed for population-based genomics. Noteworthy, if the BAPS1-6 classifications are not needed, then the core-genome alignment option of Roary can be turned off, and in that case, ProkEvo can be used with many hundredths of thousands of genomes – it is only limited based on the number of computer cores available. An example of how to implement a new program or how to turn off the core-genome alignment option in Roary within ProkEvo can be found in the following GitHub links (<https://github.com/npavloviki/ProkEvo/wiki/4.1.-Add-new-bioinformatics-tool-to-ProkEvo>) and (<https://github.com/npavloviki/ProkEvo/wiki/4.3.-Change-running-options-for-existing-tool-in-ProkEvo>), respectively. In the case of accessory genomic mining, an agnostic analysis depends on the utilization of the pan-genomic .Rtab file generated by Roary²⁴, which was not specifically used here, but instead, it was strategically demonstrated how to map AMR loci with ABRicate using the Resfinder database (<https://github.com/tseemann/abricate>). Nonetheless, there is an option to expand the scope of the accessory genomic mapping by using a pan-genomic file instead, which can be practically viewed as an expansion of the current approach (e.g., more loci included in the tabular dataset as new columns). It is important to mention that the pan-genomic mapping done by ProkEvo only provided binary information in terms of loci composition, and currently, cannot be used for the identification of single nucleotide polymorphisms across genes.

Another limitation of this protocol is the visualization of the phylogenetic tree. Currently, ggtree is the program of choice, but that comes at the cost of being unable to accurately inspect branch lengths and becomes cumbersome when many layers of data need to be added onto the phylogeny. Alternatively, phandango⁴¹ is a user-friendly, scalable web-page formatted GUI (<https://jameshadfield.github.io/phandango/#/>)⁴¹ that could easily be used to achieve the same goal, and further detailed information on how to use it with ProkEvo outputs is recently published²⁰. Other tools like iTOL could also be used for phylogeny-dependent visualization of data⁵³, but they require using a GUI and cannot be incorporated in automated scripts. Also, accurate core-genome phylogenies can be difficult to estimate due to the cryptic dataset-dependent impact of horizontal gene transfer. Programs such as Gubbins⁵⁴ can be used for that purpose, but they also come with certain limitations such as the need of using whole-genome alignment and ST lineage-specific datasets for the correct estimation of phylogenies. Instead, other phylogeny-independent approaches can be deployed, which then end up requiring other types of visualizations to integrate metadata or accessory genomic information, as in the case of multi-dimensional analysis^{55,56}. Lastly, an empirical and arbitrary approach was used to aggregate minor ST lineages and cgMLST variants, in addition to filtering the most important AMR loci to be quantified. This type of data aggregation can be done empirically using domain knowledge expertise, but could also be achieved statistically by defining *a priori* criterion of the proportion of the distribution that should be displayed, or by using distribution-related metrics such as interquartile range, standard deviation, or skewness, to ultimately define a threshold.

Importantly, the definition for minor genotypes is directly influenced by the nature of the data since sample size, and bias in the types of environmental samples can directly influence the genotypic composition. Regardless, the main consideration is that the mapping of accessory genome content onto the population structure allows for potential genetic determinants of ecological diversification to be identified, such as niche-transcending or niche-specifying genes^{57–59}.

Although the available R scripts were designed for automation of the present work, all provided scripts would need to be further developed to become an abstract and deployable data science library, that could for instance be an integral part of the ProkEvo pipeline. Nonetheless, there are some specific advantages of utilizing this approach such as the use of the BAPS level 1 genotyping or clustering scheme. The placement of BAPS level 1 sub-groups or haplotypes between serovar and ST lineages was defined empirically based on the genetic structure of the *Salmonella* population, but it seems to be applicable to other species such as *Campylobacter jejuni* and *Staphylococcus aureus*²⁰. Moreover, BAPS1 accurately captures the ancestral relationship between ST lineages and provides a scalable approach for evolutionary analysis, especially when phylogenetic applications are limited²⁰. Furthermore, the use of a nested approach for examining hierarchical relationships and patterns of diversification facilitates the identification of ancestry between ST lineages using BAPS1 sub-groups, and across cgMLST variants using ST lineages, successively going from lower to higher genotypic resolution in assessing the population structure. It is important to reiterate that the frequency distribution of ST lineages and cgMLST variants, if drawn from a systematically collected and statistically powered sample, can become a proxy for ecological fitness^{1,6,43}. Consequently, dominant ST lineages and cgMLST variants are likely to contain unique genomic features that may be the basis the biological mechanism for their dominance in the population in that particular environment or host.

Herein, two independent statistical metrics were used to assess the degree of clonality of the population, which allows for an auxiliary understanding of the population genetic diversity, which may indicate the past occurrence of sample bias, population bottlenecks, or founder effect. In particular, the agnostic assessment of BAPS levels 1–6 sub-groups across ST lineages can refine the understanding of genetic diversity that cannot typically be resolved by simply looking at *Salmonella* cgMLST variant level generated by SISTR. As mentioned beforehand, other features of the pan-genome can be mapped onto the population structure and files containing plasmid and virulence gene composition, in addition to the utilization of other AMR databases along with agnostic pan-genome dataset, are automatically generated by ProkEvo²⁰. Of note, ProkEvo does not currently allow for differentiation between AMR loci present in the bacterial chromosome vs. plasmids. Ecological and epidemiological metadata can also be easily integrated into this analytical approach by incorporation of other variables in a .csv file containing all the genomic information. In particular, the work presented here specifically complements the utilization of the scalable and portable computational platform ProkEvo, which was designed to be used by researchers focused on heuristic population genomics analyses that facilitate data mining and customization by the user. Other platforms can be used for genotyping, population structure analysis, and/or mapping of accessory genomes such as Enterobase⁵, PATRIC⁶⁰, and BacWGSTdb⁶¹. The latter are excellent resources that facilitate genomics data mining for

researchers who are not seeking to customize and utilize cluster computing for scalable and complex analysis. The analytical approach presented here is specifically tailored for researchers who want to have the flexibility to carry out a population genomics analysis using reproducible scripts on their local machine or by using a cloud- or high-performance computational platform.

In conclusion, the analytical R-based platform presented in this work was targeted at providing a practical guide for microbiologists, ecologists, and epidemiologists on how to: i) use phylogeny-dependent approaches to map hierarchical genotypes; ii) assess the frequency distribution of genotypes as a proxy for evaluating ecological fitness; iii) determine lineage-specific degrees of clonality using independent statistical approaches; and iv) map lineage-differentiating AMR loci as an example of how to mine accessory genomic content in the context of the population structure. The scripts provided here can be used on either a local machine or a high-performance computational platform. For experimental and environmental microbiologists, this approach facilitates studies of datasets aimed at identifying unique traits and candidate pathways for further mechanistic studies that ultimately can be contextualized at the population level. Ecologists can benefit from this approach by being able to analyze moderate-to-large datasets, that in theory, increase the statistical power needed to find signatures of selection in a population while considering kinship relationships and patterns of diversification. Lastly, epidemiologists can harness unique practical information for diagnostics and surveillance by defining genotypic units of interest and predicting Public Health-associated traits such as AMR. More broadly, this analytical guidance provides a generalizable framework to utilize ProkEvo to perform a population-based genomic analysis that can be used to infer evolutionary and ecological patterns for pathogenic and non-pathogenic species since the approach is generalizable to other bacterial species.

ACKNOWLEDGMENTS:

This work was supported by funding provided by the UNL-IANR Agricultural Research Division and the National Institute for Antimicrobial Resistance Research and Education and by the Nebraska Food for Health Center at the Food Science and Technology Department (UNL). This research could only be completed by utilizing the Holland Computing Center (HCC) at UNL, which receives support from the Nebraska Research Initiative. We are also thankful for having access, through the HCC, to resources provided by the Open Science Grid (OSG), which is supported by the National Science Foundation and the U.S. Department of Energy's Office of Science. This work used the Pegasus Workflow Management Software which is funded by the National Science Foundation (grant #1664162).

DISCLOSURES:

The authors have declared that no competing interests exist.

REFERENCES:

1. Grad, Y. H. et al. Genomic epidemiology of the *Escherichia coli* O104:H4 outbreaks in Europe, 2011. *Proceedings of the National Academy of Sciences of the United States of America*. **109** (8), 3065–3070 (2012).
2. Worby, C. J., Chang, H. -H., Hanage, W. P., Lipsitch, M. The distribution of pairwise genetic

- distances: a tool for investigating disease transmission. *Genetics*. **198** (4), 1395–1404 (2014).
3. Leekitcharoenphon, P. et al. Global genomic epidemiology of *Salmonella enterica* serovar Typhimurium DT104. *Applied and Environmental Microbiology*. **82** (8), 2516–2526 (2016).
4. Alba, P. et al. Molecular epidemiology of *Salmonella* Infantis in Europe: insights into the success of the bacterial host and its parasitic pESI-like megaplasmid. *Microbial Genomics*. **6** (5), (2020).
5. Zhou, Z., Alikhan, N. -F., Mohamed, K., Fan, Y., the Agama Study Group, Achtman, M. The EnteroBase user's guide, with case studies on *Salmonella* transmissions, *Yersinia pestis* phylogeny, and *Escherichia* core genomic diversity. *Genome Research*. **30** (1), 138–152 (2020).
6. Azarian, T. et al. Global emergence and population dynamics of divergent serotype 3 CC180 pneumococci. *PLOS Pathogens*. **14** (11), e1007438 (2018).
7. Saltykova, A. et al. Comparison of SNP-based subtyping workflows for bacterial isolates using WGS data, applied to *Salmonella enterica* serotype Typhimurium and serotype 1,4,[5],12:i:-. *PLOS ONE*. **13** (2), e0192504 (2018).
8. Achtman, M. et al. Multi-locus sequence typing as a replacement for serotyping in *Salmonella enterica*. *PLoS Pathogens*. **8** (6), e1002776 (2012).
9. Maiden, M. C. J. et al. Multi-locus sequence typing: A portable approach to the identification of clones within populations of pathogenic microorganisms. *Proceedings of the National Academy of Sciences of the United States of America*. **95** (6), 3140–3145 (1998).
10. Alikhan, N. -F., Zhou, Z., Sergeant, M. J., Achtman, M. A genomic overview of the population structure of *Salmonella*. *PLOS Genetics*. **14** (4), e1007261 (2018).
11. Gupta, A., Jordan, I. K., Rishishwar, L. stringMLST: a fast k-mer based tool for multi-locus sequence typing. *Bioinformatics*. **33** (1), 119–121 (2017).
12. Jolley, K. A., Maiden, M. C. BIGSdb: Scalable analysis of bacterial genome variation at the population level. *BMC Bioinformatics*. **11** (1), 595 (2010).
13. Maiden, M. C. J. et al. MLST revisited: the gene-by-gene approach to bacterial genomics. *Nature Reviews Microbiology*. **11** (10), 728–736 (2013).
14. Maiden, M. C. J. Multilocus sequence typing of bacteria. *Annual Review of Microbiology*. **60** (1), 561–588 (2006).
15. Shapiro, B. J., Polz, M. F. Ordering microbial diversity into ecologically and genetically cohesive units. *Trends in Microbiology*. **22** (5), 235–247 (2014).
16. Cordero, O. X., Polz, M. F. Explaining microbial genomic diversity in light of evolutionary ecology. *Nature Reviews Microbiology*. **12** (4), 263–273 (2014).
17. Achtman, M., Wagner, M. Microbial diversity and the genetic nature of microbial species. *Nature Reviews Microbiology*. **6** (6), 431–440 (2008).
18. Abudahab, K. et al. PANINI: Pangenome neighbour identification for bacterial populations. *Microbial Genomics*. **5** (4) (2019).
19. Laing, C. R., Whiteside, M. D., Gannon, V. P. J. Pan-genome analyses of the species *Salmonella enterica*, and identification of genomic markers predictive for species, subspecies, and serovar. *Frontiers in Microbiology*. **8**, 1345 (2017).
20. Pavlovikj, N., Gomes-Neto, J. C., Deogun, J. S., Benson, A. K. ProkEvo: an automated, reproducible, and scalable framework for high-throughput bacterial population genomics analyses. *PeerJ*. **9**, e11376 (2021).
21. McNally, A. et al. Combined analysis of variation in core, accessory and regulatory genome

- regions provides a super-resolution view into the evolution of bacterial populations. *PLOS Genetics*. **12** (9), e1006280 (2016).
22. Langridge, G. C. et al. Patterns of genome evolution that have accompanied host adaptation in *Salmonella*. *Proceedings of the National Academy of Sciences of the United States of America*. **112** (3), 863–868 (2015).
23. Price, M. N., Dehal, P. S., Arkin, A. P. FastTree 2 – Approximately maximum-likelihood trees for large alignments. *PLoS ONE*. **5** (3), e9490 (2010).
24. Page, A. J. et al. Roary: rapid large-scale prokaryote pan genome analysis. *Bioinformatics*. **31** (22), 3691–3693 (2015).
25. Yoshida, C. E. et al. The Salmonella In silico typing resource (SISTR): An open web-accessible tool for rapidly typing and subtyping draft Salmonella genome assemblies. *PLOS ONE*. **11** (1), e0147101 (2016).
26. Cheng, L., Connor, T. R., Siren, J., Aanensen, D. M., Corander, J. Hierarchical and spatially explicit clustering of DNA sequences with BAPS software. *Molecular Biology and Evolution*. **30** (5), 1224–1228 (2013).
27. Tonkin-Hill, G., Lees, J. A., Bentley, S. D., Frost, S. D. W., Corander, J. Fast hierarchical Bayesian analysis of population structure. *Nucleic Acids Research*. **47** (11), 5539–5549 (2019).
28. Seemann, T. *MLST*. *GitHub*. at <<https://github.com/tseemann/mlst>> (2020).
29. Seemann, T. *ABRicate*. *GitHub*. at <<https://github.com/tseemann/abricate>> (2020).
30. R Core Team *R: A language and environment for statistical computing*. *R Foundation for Statistical Computing, Vienna, Austria*. at <<https://cran.r-project.org>> (2021).
31. RStudio Team *RStudio: Integrated Development for R*. *RStudio, PBC, Boston, MA*. at <<http://www.rstudio.com/>> (2020).
32. Wickham, H. et al. Welcome to the Tidyverse. *Journal of Open Source Software*. **4** (43), 1686 (2019).
33. rOpenSci *rOpenSci: The skimr package*. *GitHub*. at <<https://github.com/ropensci/skimr/>> Berkeley, CA (2021).
34. Oksanen, J. et al. *vegan: Community ecology package*. R package version 2.5-5. at <<https://CRAN.R-project.org/package=vegan>> (2019).
35. Tierney, N. J., Cook, D. H. Expanding tidy data principles to facilitate missing data exploration, visualization and assessment of imputations. *arXiv:1809.02264 [stat]* at <<http://arxiv.org/abs/1809.02264>> (2020).
36. Yu, G. Using ggtree to visualize data on tree-like structures. *Current Protocols in Bioinformatics*. **69** (1) (2020).
37. Kassambara, A. *ggpubr: “ggplot2” Based Publication Ready Plots*. R package version 0.4.0. at <<https://CRAN.R-project.org/package=ggpubr>> (2020).
38. Slowikowski, K. *ggrepel: Automatically Position Non-Overlapping Text Labels with “ggplot2”*. R package version 0.9.1. at <<https://CRAN.R-project.org/package=ggrepel>> (2021).
39. Wickham, H. Reshaping Data with the reshape Package. *Journal of Statistical Software*. **21** (12) (2007).
40. Neuwirth, E. *RColorBrewer: ColorBrewer Palettes*. R package version 1.1-2. at <<https://CRAN.R-project.org/package=RColorBrewer>> (2014).
41. Hadfield, J., Croucher, N. J., Goater, R. J., Abudahab, K., Aanensen, D. M., Harris, S. R.

Phandango: an interactive viewer for bacterial population genomics. *Bioinformatics*. **34** (2), 292–293 (2018).

42. Perron, G. G. et al. Functional characterization of bacteria isolated from ancient arctic soil exposes diverse resistance mechanisms to modern antibiotics. *PLOS ONE*. **10** (3), e0069533 (2015).

43. Mitchell, P. K. et al. Population genomics of pneumococcal carriage in Massachusetts children following introduction of PCV-13. *Microbial Genomics*. **5** (2) (2019).

44. Klemm, E. J. et al. Emergence of host-adapted *Salmonella* Enteritidis through rapid evolution in an immunocompromised host. *Nature Microbiology*. **1** (3), 15023 (2016).

45. Břinda, K. et al. Rapid inference of antibiotic resistance and susceptibility by genomic neighbour typing. *Nature Microbiology*. **5** (3), 455–464 (2020).

46. MacFadden, D. R. et al. Using genetic distance from archived samples for the prediction of antibiotic resistance in *Escherichia coli*. *Antimicrobial Agents and Chemotherapy*. **64** (5) (2020).

47. Mageiros, L. et al. Genome evolution and the emergence of pathogenicity in avian *Escherichia coli*. *Nature Communications*. **12** (1), 765 (2021).

48. Yahara, K. et al. Genome-wide association of functional traits linked with *Campylobacter jejuni* survival from farm to fork. *Environmental Microbiology*. **19** (1), 361–380 (2017).

49. Walter, J., Maldonado-Gómez, M. X., Martínez, I. To engraft or not to engraft: an ecological framework for gut microbiome modulation with live microbes. *Current Opinion in Biotechnology*. **49**, 129–139 (2018).

50. Maldonado-Gómez, M. X. et al. Stable engraftment of *Bifidobacterium longum* AH1206 in the human gut depends on individualized features of the resident microbiome. *Cell Host & Microbe*. **20** (4), 515–526 (2016).

51. Zhao, S. et al. Adaptive evolution within gut microbiomes of healthy people. *Cell Host & Microbe*. **25** (5), 656–667.e8 (2019).

52. Treangen, T. J., Ondov, B. D., Koren, S., Phillippy, A. M. The Harvest suite for rapid core-genome alignment and visualization of thousands of intraspecific microbial genomes. *Genome Biology*. **15** (11), 524 (2014).

53. Letunic, I., Bork, P. Interactive Tree Of Life (iTOL) v5: an online tool for phylogenetic tree display and annotation. *Nucleic Acids Research*. **49** (W1), W293–W296 (2021).

54. Croucher, N. J. et al. Rapid phylogenetic analysis of large samples of recombinant bacterial whole genome sequences using Gubbins. *Nucleic Acids Research*. **43** (3), e15–e15 (2015).

55. Fenske, G. J., Thachil, A., McDonough, P. L., Glaser, A., Scaria, J. Geography shapes the population genomics of *Salmonella enterica* Dublin. *Genome Biology and Evolution*. **11** (8), 2220–2231 (2019).

56. Lees, J. A. et al. Fast and flexible bacterial genomic epidemiology with PopPUNK. *Genome Research*. **29** (2), 304–316 (2019).

57. Cohan, F. M. Towards a conceptual and operational union of bacterial systematics, ecology, and evolution. *Philosophical Transactions of the Royal Society B: Biological Sciences*. **361** (1475), 1985–1996 (2006).

58. Cohan, F. M., Koeppel, A. F. The origins of ecological diversity in prokaryotes. *Current Biology*. **18** (21), R1024–R1034 (2008).

59. Cohan, F. M. Transmission in the origins of bacterial diversity, from ecotypes to phyla. *Microbial Transmission*. **5** (5), 311–343, doi: 10.1128/microbiolspec.MTBP-0014-2016 (2019).

- 1012 60. Davis, J. J. et al. The PATRIC bioinformatics resource center: expanding data and analysis
1013 capabilities. *Nucleic Acids Research*. **48** (D1), D606–D612 (2019).
- 1014 61. Feng, Y., Zou, S., Chen, H., Yu, Y., Ruan, Z. BacWGSTdb 2.0: a one-stop repository for
1015 bacterial whole-genome sequence typing and source tracking. *Nucleic Acids Research*. **49** (D1),
1016 D644–D650 (2021).

1017

Figure 1

Hierarchical population structure - S. Newport

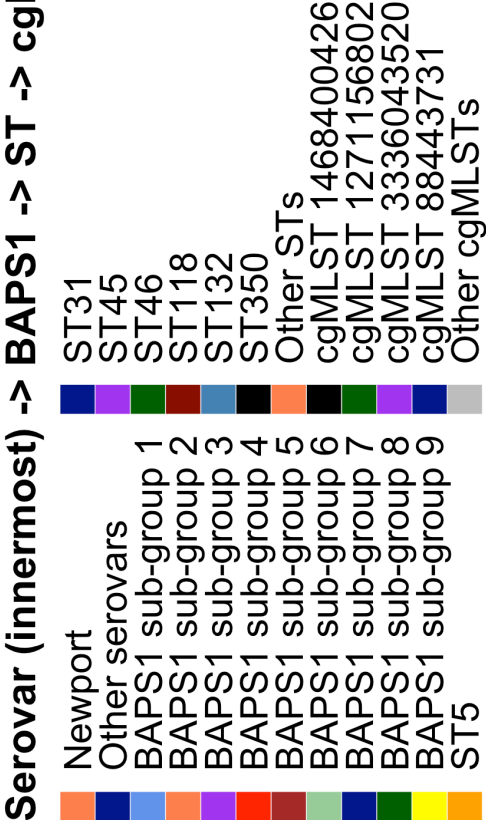
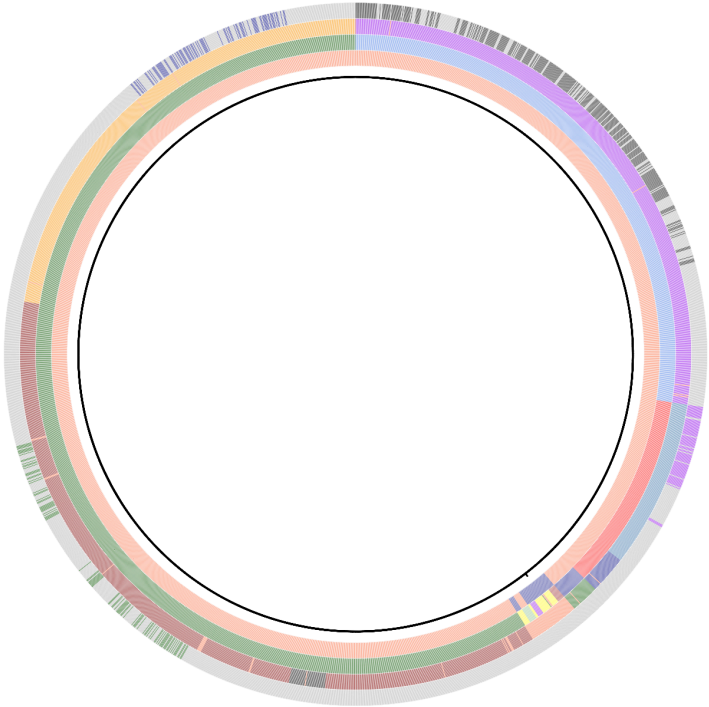
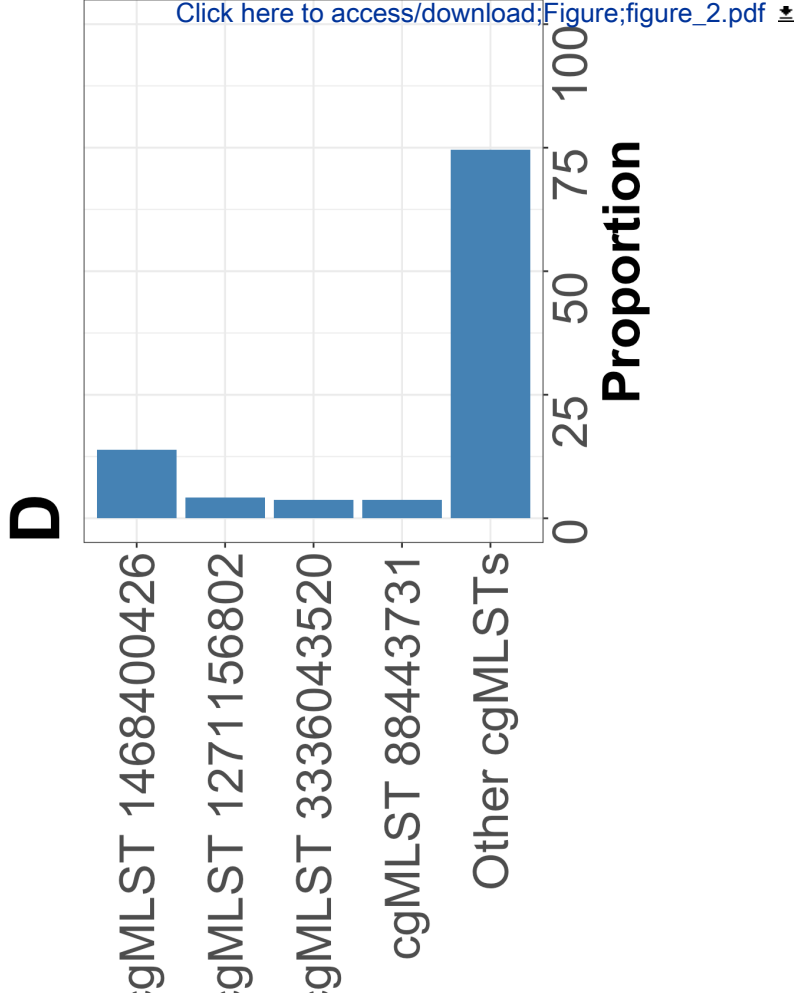
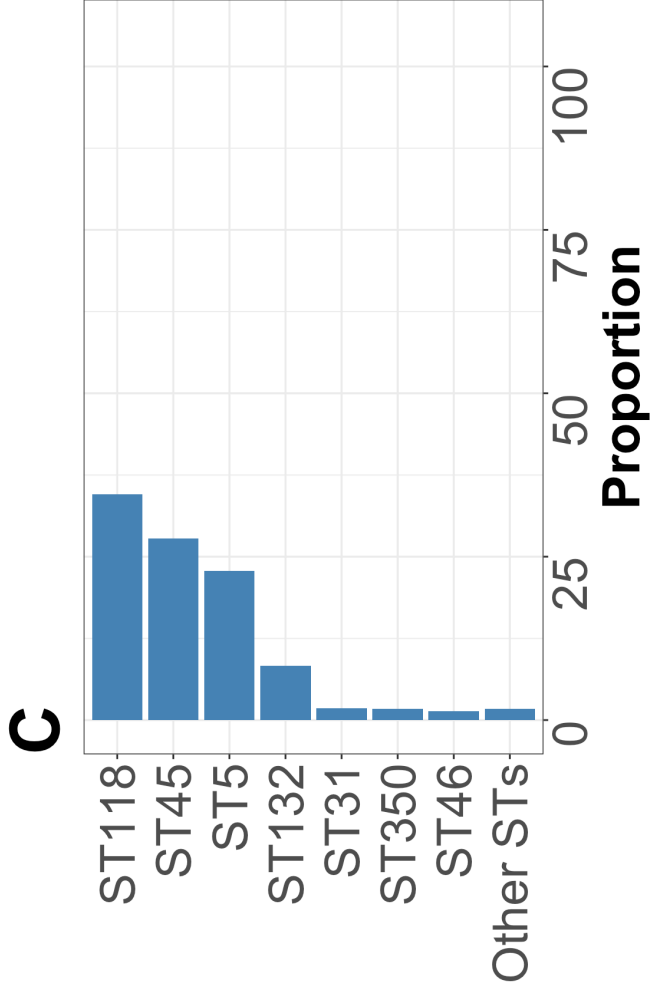
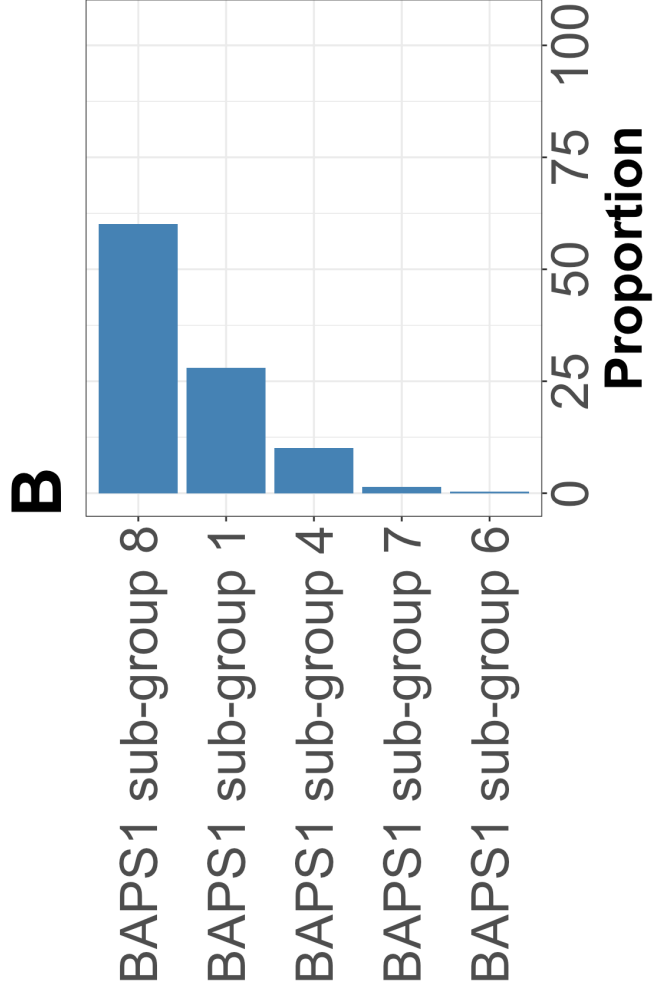
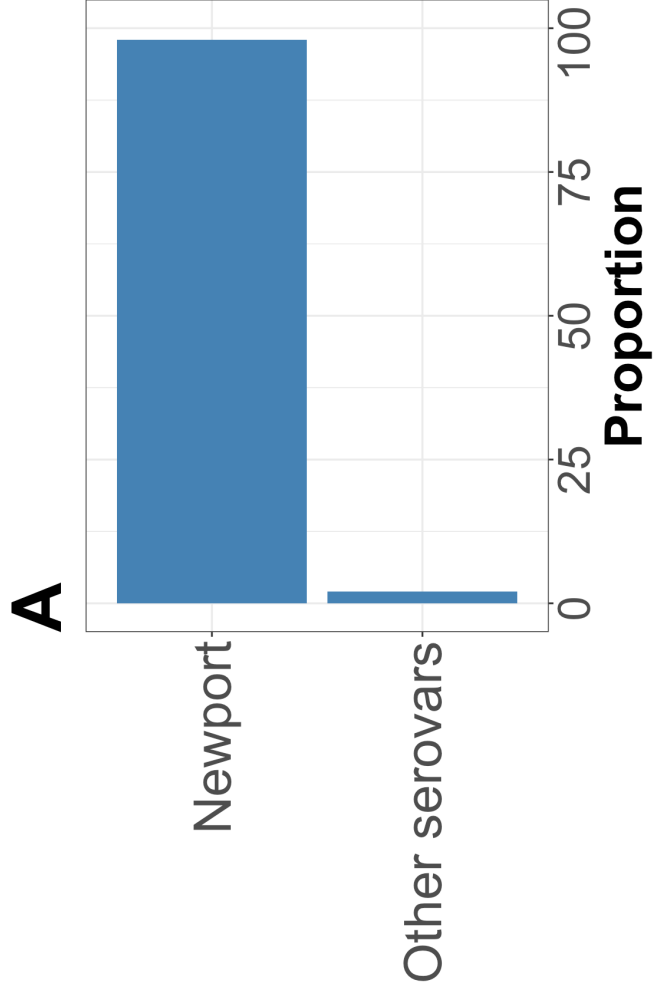


Figure 2



[Click here to access/download;Figure;figure_2.pdf](#)

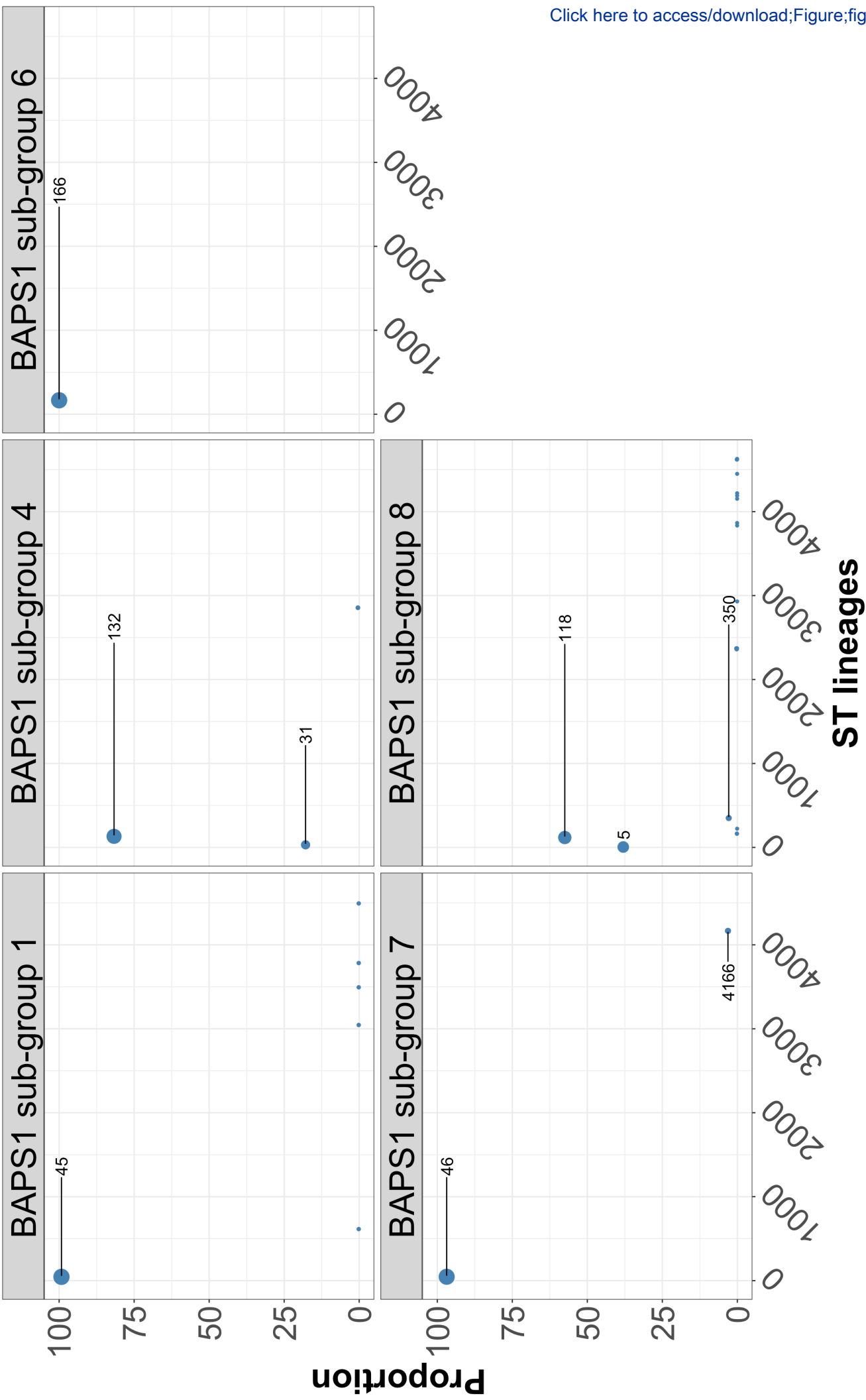


Figure 4

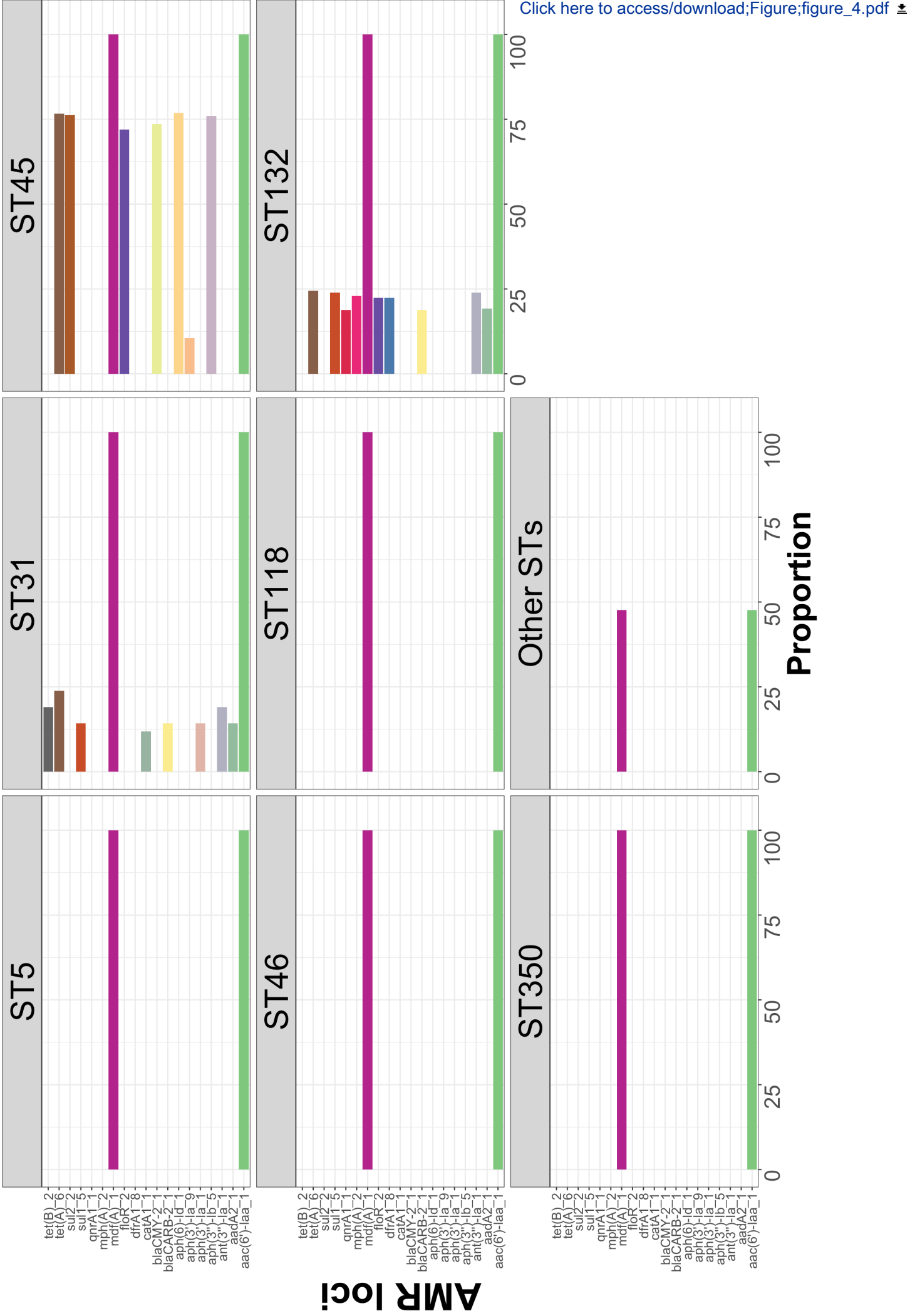
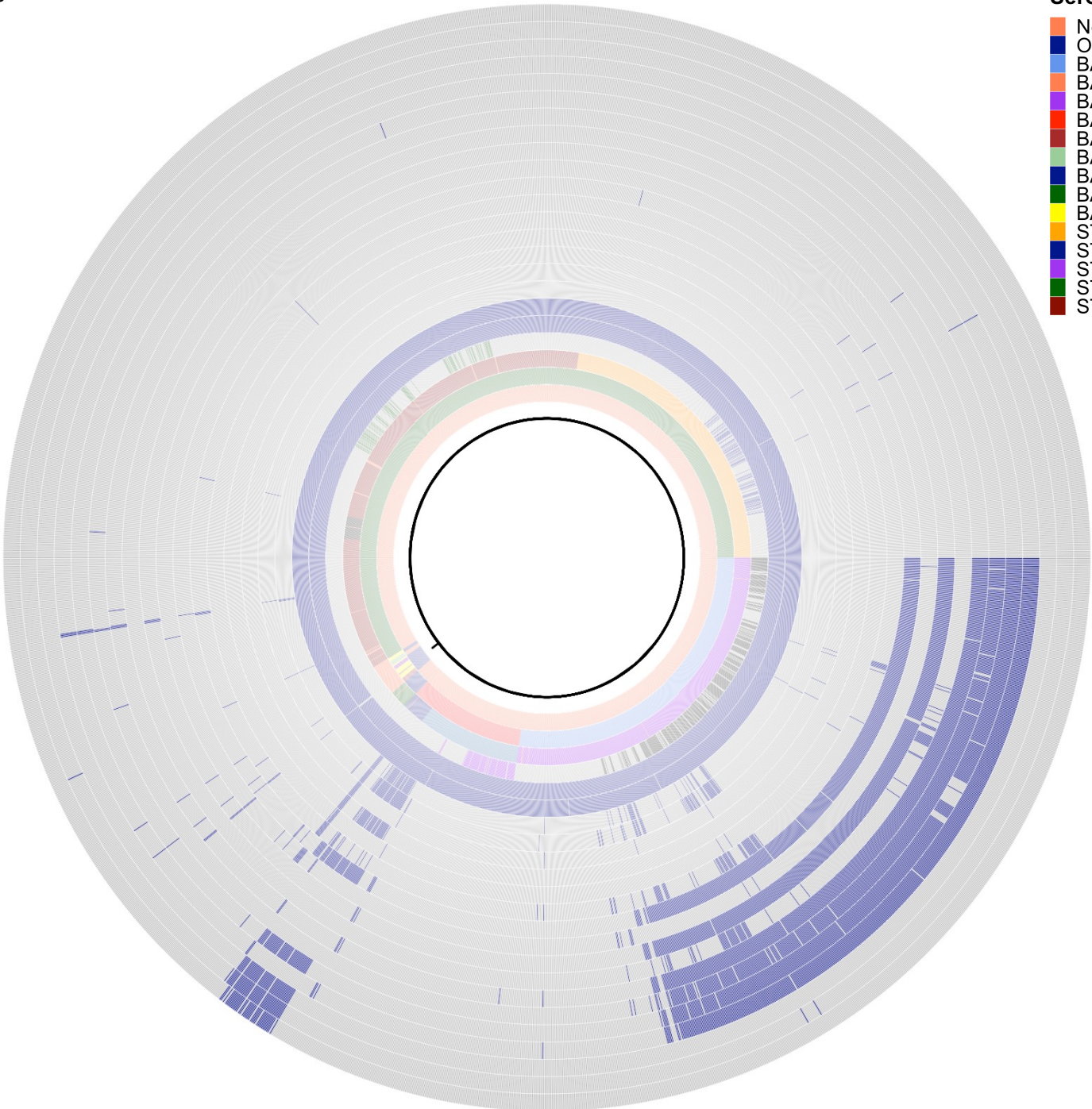


Figure 5



[Click here to access/download;Figure;figure_5.pdf](#)

| Serovar -> BAPS1 -> ST -> cgMLST -> AMR loci | | | |
|--|-------------------------|-------------------------|-----------------------|
| Newport | ST132 | aph(3')-la_1 (present) | aph(6)-ld_1 (present) |
| Other serovars | ST350 | aph(3')-la_1 (absent) | aph(6)-ld_1 (absent) |
| BAPS1 sub-group 1 | Other STs | blaCARB-2_1 (present) | blaCMY-2_1 (present) |
| BAPS1 sub-group 2 | cgMLST 1468400426 | blaCARB-2_1 (absent) | blaCMY-2_1 (absent) |
| BAPS1 sub-group 3 | cgMLST 1271156802 | catA1_1 (present) | floR_2 (present) |
| BAPS1 sub-group 4 | cgMLST 3336043520 | catA1_1 (absent) | floR_2 (absent) |
| BAPS1 sub-group 5 | cgMLST 88443731 | sul1_5 (present) | sul2_2 (present) |
| BAPS1 sub-group 6 | Other cgMLSTs | sul1_5 (absent) | sul2_2 (absent) |
| BAPS1 sub-group 7 | aac(6')-laa_1 (present) | tet(A)_6 (present) | dfrA1_8 (present) |
| BAPS1 sub-group 8 | aac(6')-laa_1 (absent) | tet(A)_6 (absent) | dfrA1_8 (absent) |
| BAPS1 sub-group 9 | mdf(A)_1 (present) | tet(B)_2 (present) | mph(A)_2 (present) |
| ST5 | mdf(A)_1 (absent) | tet(B)_2 (absent) | mph(A)_2 (absent) |
| ST31 | aadA2_1 (present) | aph(3'')-lb_5 (present) | qnrA1_1 (present) |
| ST45 | aadA2_1 (absent) | aph(3'')-lb_5 (absent) | qnrA1_1 (absent) |
| ST46 | ant(3'')-la_1 (present) | aph(3')-la_9 (present) | |
| ST118 | ant(3'')-la_1 (absent) | aph(3')-la_9 (absent) | |



[Click here to access/download](#)

Table of Materials

Jove_table_materials-63115R3.xlsx



Response to reviewers:

First of all, we would like to thank the Editor and reviewers for your comments, because the paper is in much better shape with your suggestions being added to it. Point-by-point responses are listed below. Thank you!

Editorial comments:

Editorial Changes

Changes to be made by the Author(s):

Thank you for your comments and considerations.

1. Please take this opportunity to thoroughly proofread the manuscript to ensure that there are no spelling or grammar issues.

Modified accordingly.

2. Please adjust the numbering of the Protocol to follow the JoVE Instructions for Authors. For example, 1 should be followed by 1.1 and then 1.1.1 and 1.1.2 if necessary. Please refrain from using bullets or dashes.

Modified accordingly.

3. JoVE policy states that the video narrative is objective and not biased towards a particular product featured in the video. The goal of this policy is to focus on science rather than to present a technique as an advertisement for a specific item. To this end, we ask that you please reduce the number of instances of "_____" within your text. The term may be introduced but please use it infrequently and when directly relevant. Otherwise, please refer to the term using generic language. For example, R software, RStudio, ProkEvo, etc.

Modified as best as we could.

We did the best we could here because the R scripts and analytical platform was designed specifically to complement the ProkEvo pipeline (which is freely available). We are not promoting the use of the R software or RStudio, but all the scripts and software are completely free for any user anywhere in the world. We tried as best as we could to minimize the number of times we mentioned the names though, but in certain key places we could not avoid it.

4. The Protocol should contain only action items that direct the reader to do something. Please move the discussion about the protocol to the Discussion.

Modified accordingly.

5. Please revise the text to avoid the use of any personal pronouns (e.g., "we", "you", "our" etc.).

Modified accordingly.

6. Please include a single line space between each step, substep, and note in the protocol section. Please highlight up to 3 pages of the Protocol (including headings and spacing) that identifies the essential steps of the protocol for the video, i.e., the steps that should be visualized to tell the most cohesive story of the Protocol. Remember that non-highlighted Protocol steps will remain in the manuscript, and therefore will still be available to the reader. Please keep in mind that software steps without a graphical user interface (GUI) cannot be filmed.

Modified accordingly.

7. Please refrain from using bullets in the representative results.

Modified accordingly.

8. Please include all the Figure Legends (including supplementary figures and tables) together at the end of the Representative Results in the manuscript text.

Modified accordingly.

Reviewers' comments:

Reviewer #1:

Manuscript Summary:

The manuscript presents a protocol for the analysis of bacterial pangenomes using the ProkEvo workflow previously published by the same authors. It is demonstrated how from a set of whole genome sequencing files one can generate diagrams with population structure, classify bacteria into sequence types, and obtain a list of the identified genes known to confer antimicrobial resistance. The presentation is extensive and accompanied with R scripting code snippets and anticipated results. The protocol may be potentially useful if the authors make it simple to use and demonstrate that it is better in some way than other existing analytical pipelines in the field.

Thank you for your comments and considerations.

Minor Concerns:

1. The original publication in the PeerJ journal presents ProkEvo as a complex analytical workflow that employs many third-party tools. I am not sure whether the presented protocol in this manuscript covers all dependencies in the section where installation steps are described. Maybe a table of dependencies and corresponding

steps to install them will give a better view on the pre-requisites to this protocol to successfully execute.

This is an important point that we have clarified now. ProkEvo is the bioinformatics platform that processes WGS data into .csv outputs that we are now using in this paper to show the R-based statistical analysis. For this paper the user does not need to run ProkEvo but we point it out because the outputs are all generated through it (previous publication of the ProkEvo paper which is cited in this paper). Our R-based scripts basically complement ProkEvo, but its premise and approach could be used with any output that contains the same kind of information (ST classification, AMR mapping, etc.). ProkEvo does not create any problems with dependencies. In fact, for consumers of WGS data, ProkEvo helps tremendously because in a single installation step the user gets the most up-to-date version of all packages contained in it. Now, as for the R scripts, there are R libraries that need to be installed, and those are now all clarified inside of the paper. Thank you for highlighting this point because it helped us clarifying the purpose of this paper more objectively which is to conduct a population-based analysis assuming the user has those outputs to begin with (which are produced by ProkEvo as stated).

2. The authors objectively discussed the shortcomings on the protocol. However, I think 2 major points are still missing or under-described in the capabilities of the protocol: (1) The identification and report of mutations in the AMR genes. Frequently, mutation in the targeted gene drives the antimicrobial resistance. (2) The use of this protocol to analyze a single sample is unclear. For example, contrasting a given sample to the collection of the already annotated strains would be useful.

Thank you for bringing these points up. Answer to point 1: ProkEvo is not set up for identification of mutations or alleles as it is, although it could, but this type of analysis is out of the scope of this paper. We clarified that limitation in the text.

Answer to point 2: Yes, ProkEvo can be run with a single sample to get genotyping and genome annotation including AMR gene and plasmid mapping, but the purpose of this paper is to carry out a population-based analysis. That is also now clarified in the text.

3. The authors should discuss or demonstrate in results as to why their protocol is any different or advantageous compared to the existing tools, for example
PATRIC at <https://patricbrc.org/>
iTOL at <https://itol.embl.de/>
BacWGSTdb at <http://bacdb.cn/BacWGSTdb/>

We have highlighted the potential for users to use web-interface or GUI type of tools in comparison to ours, but there are tremendous advantages in using ProkEvo and these R scripts, such as: customizing your own analysis, deploying the software in high-performance computers or private cloud environments, adding or removing programs, exploring the data and conducting statistical analysis using domain expertise, changing the visualization as the user wishes, etc. We certainly acknowledge the importance of other tools that are out there, and have also used them, and ProkEvo is not here to be a

silver bullet, but instead it is in our view that this approach is very useful for those who would like to deploy a scalable program for WGS analysis and tailor it to their specific needs.

Reviewer #2:

Manuscript Summary:

The authors discussed the application of their ProkEvo computational platform for performing population-based genomic analysis. They used R to discuss the hierarchical analysis based on a sample dataset. In general, the protocol, applications, and results are well-explained in the manuscript.

Thank you for your comments and considerations.

Major Concerns:

I don't have any major concerns.

Minor Concerns:

I have a couple of comments/questions.

1. The authors should include system requirements (processor, memory, etc.) to run their R-based framework.

We clarified that, but the .csv files used as an input are very light, therefore, this analysis could be done virtually in any laptop, and certainly in any high-performance platform.

2. Their computational platform supports analysis for a limited number of genomes. What can be the maximum input size? Is it possible to overcome this limitation?

ProkEvo has one major limitation (with the tools it contains) that we can detect which is the number of genomes used, if the user cares about generating a core-genome alignment through Roary (which is inside of ProkEvo). If the user turns off that option, it scales to many thousands of genomes at once. We don't have a cap right now because we have only run a 10-30,000 at once and had no problem. But we have clarified that in the text.

3. The authors mentioned that their framework can be generalized for python and other languages. How can it be generalized? The manuscript needs to add some explanations to it.

Thank you for bringing that up as well. What we meant with that was that if the user does not want to use R, he or she could simply apply the same principles or concepts with Pandas in Python. But we have clarified that in the text as well.

4. The manuscript should include any limitations regarding time & space complexities? Is it possible to perform parallel computation using their framework?

ProkEvo is already set up for that and the original publication which is cited in this paper demonstrates that very aspect. But that is not part of this paper, because what the work presented in this paper focus on using the .csv files produced by ProkEvo to conduct a statistical analysis of the data, which as it is has not required parallelization, because the output files are really light and easy to work with in a simple laptop setting.

5. If possible, please improve the visualization of the statistics/results.

Thank you for suggesting that, but we have kept our visualizations as they are. We totally respect your comment though.



Food Science and Technology Department

**University of Nebraska-Lincoln
1901 N 21 ST
Lincoln, NE 68588-6205**

November 6th, 2021

Dear Editor,

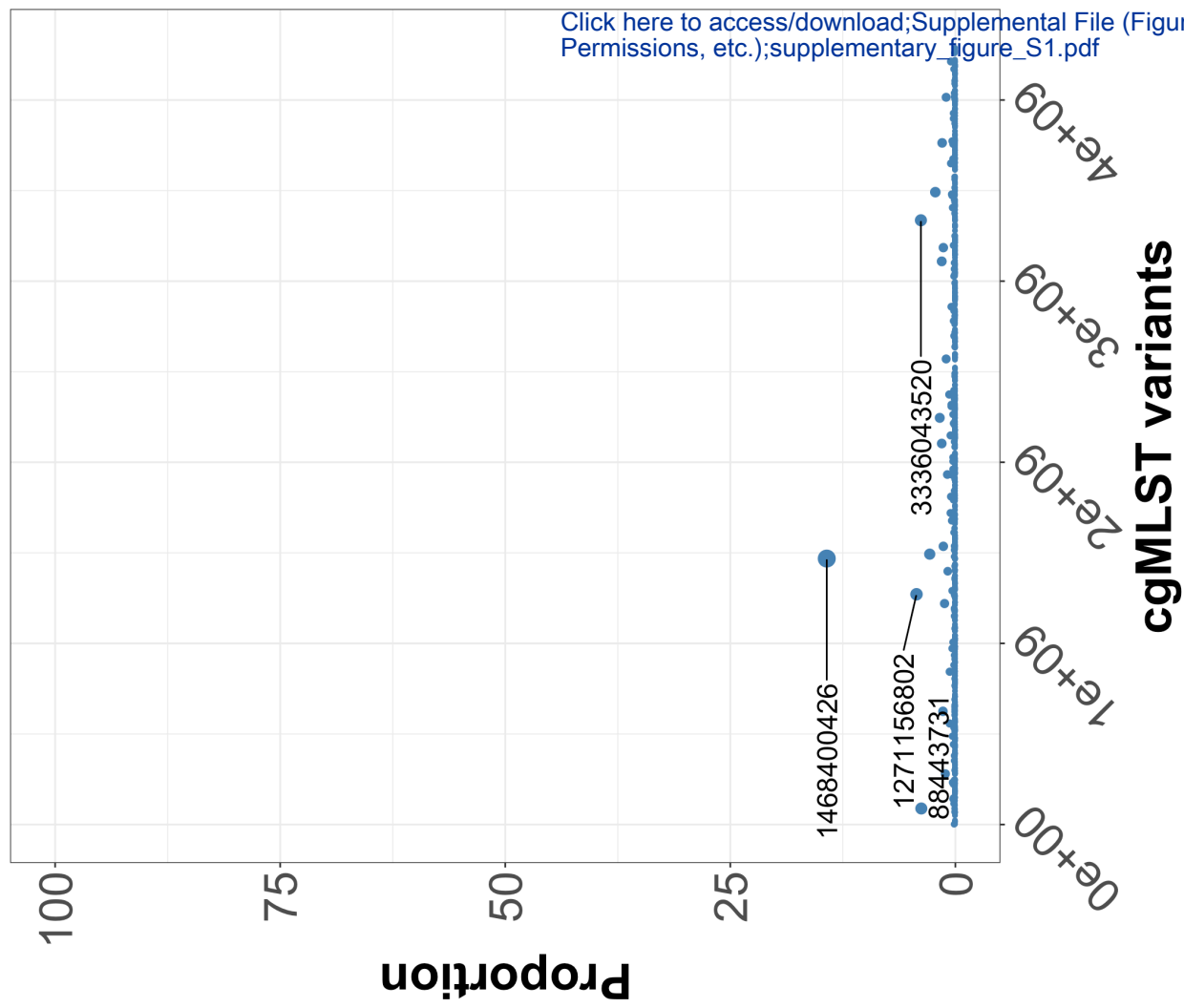
We are very thankful for your final suggestions. After considering all suggested points, we believe this manuscript is more adequately formatted for publication in Jove.

Sincerely,

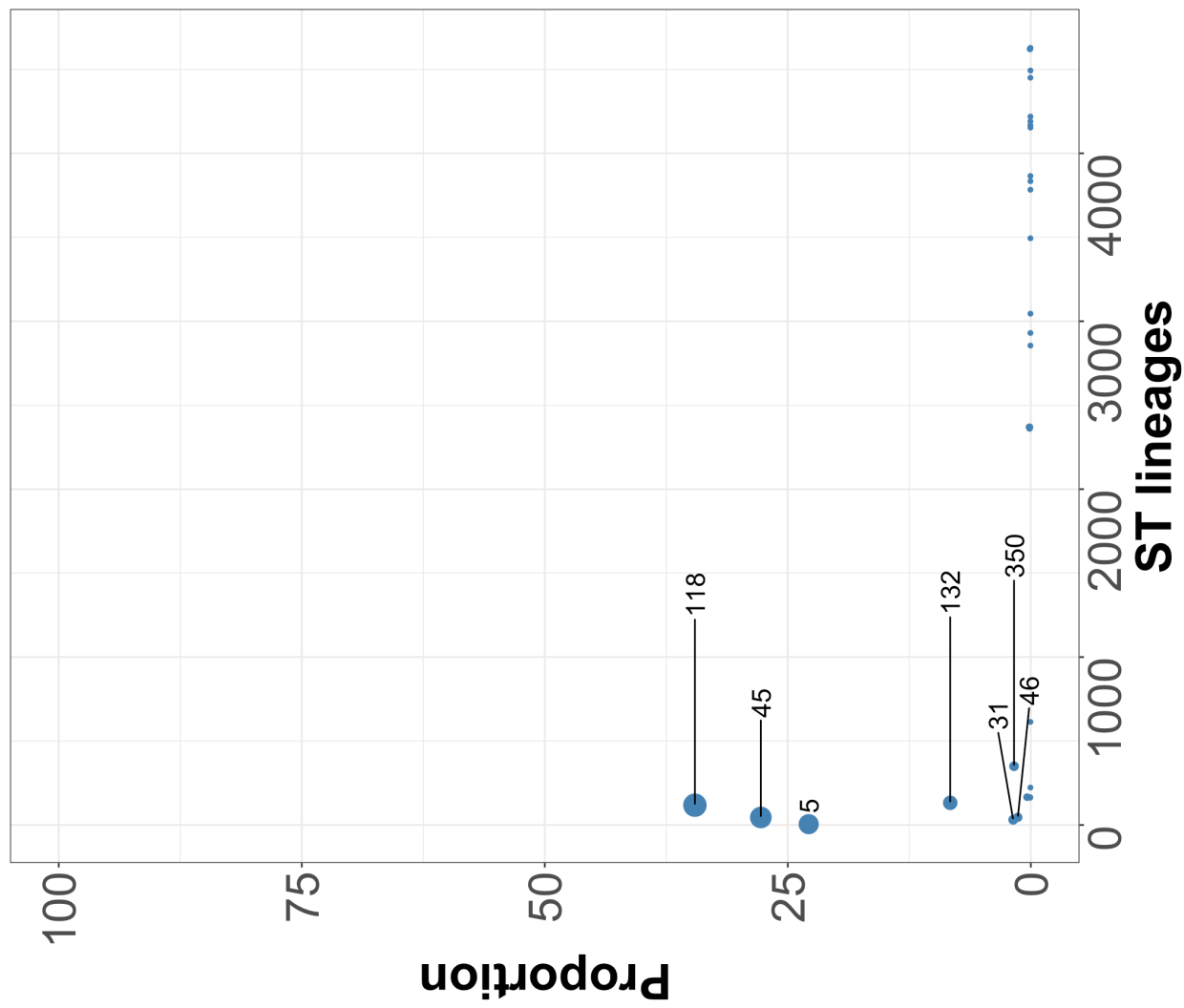
Joao Carlos Gomes Neto
DVM, MS, PhD Postdoctoral researcher

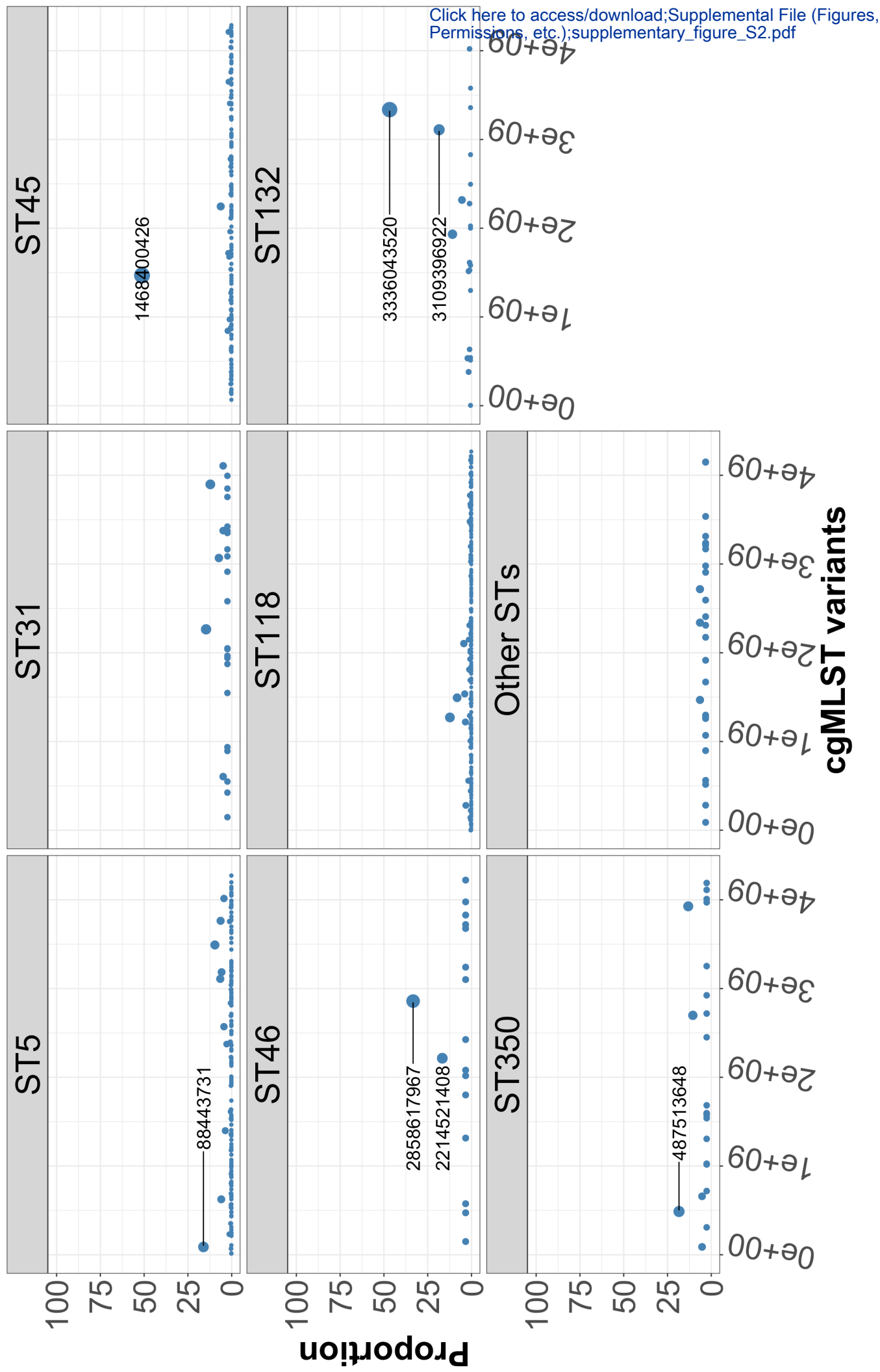
Andrew K. Benson
Professor, Department of Food Science and Technology

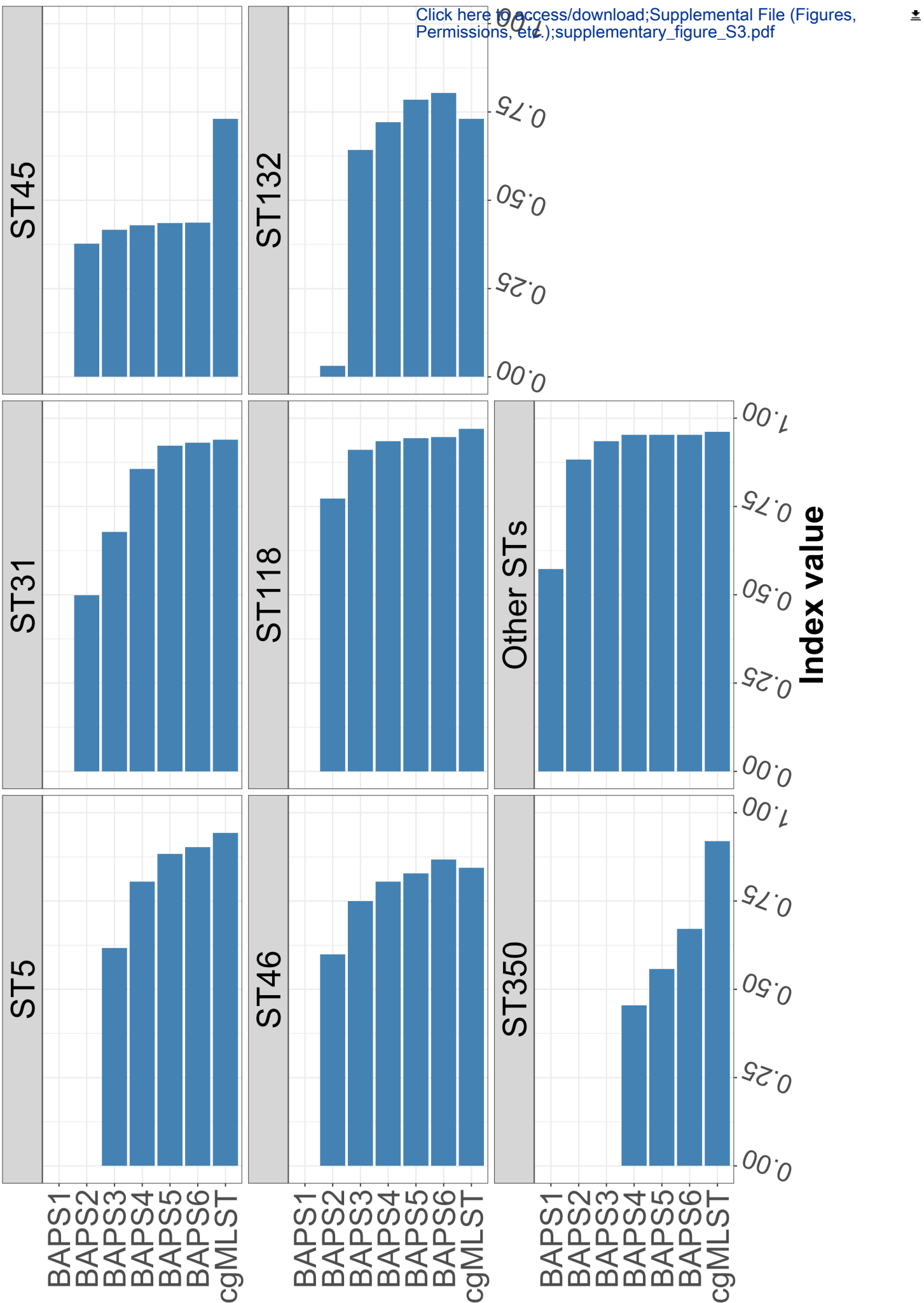
B

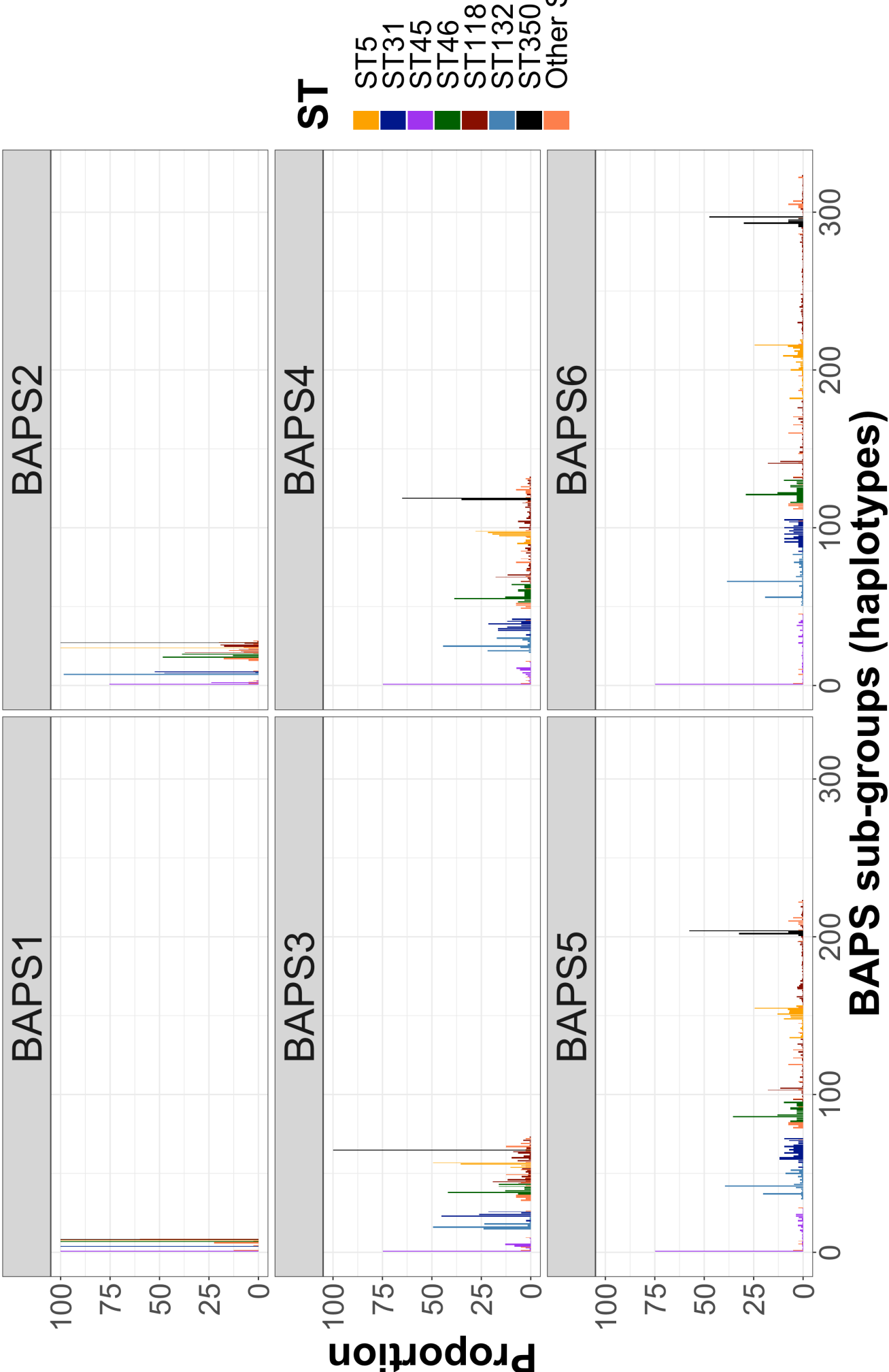


A









Supplementary Materials

Material List.

<https://figshare.com/account/projects/116625/articles/14829225?file=28963260>

Genomes List.

<https://figshare.com/account/home#/projects/116625>

Hierarchical-based bacterial population genomics analysis using R

Gomes-Neto JC, Pavlovikj N, Benson AK

This case study will be done using a Salmonella Newport dataset that is available on NCBI-SRA, and contains 2,365 genomes.

A list of genomes and datasets are all available here:

<https://figshare.com/account/home#/projects/116625> This link to Figshare requires login credentials.

How to install all packages. Packages should be installed only once. From time-to-time new versions will be available and the user is responsible for updating them accordingly. Make sure versions of programs are reported every time you use them.

When beginning running this script, you can remove the # that comes before the `install.packages()` function, but when running it the second time around, and subsequently, comment the function out using #. That way you avoid creating issues with dependencies, and different versions of the program.

```
# install tidyverse
# install.packages("tidyverse")
# install skimr
# install.packages("skimr")
# install vegan
# install.packages("vegan")
# install forcats
# install.packages("forcats")
# install nanian
# install.packages("nanian")
# install ggpubr
# install.packages("ggpubr")
# install ggrepel
# install.packages("ggrepel")
# install reshape2
# install.packages("reshape2")
# install reshape2
# install.packages("RColorBrewer")
# install ggtree
# if (!requireNamespace("BiocManager", quietly = TRUE))
#   install.packages("BiocManager")
# BiocManager::install("ggtree")
# installation of ggtree will prompt a question about installation - answer i
s "a" to install/update all dependencies
```

How to activate packages prior to utilization.

```

# Load previously installed packages
library(tidyverse)

## — Attaching packages ————— tidyverse 1.
3.1 —

## ✓ ggplot2 3.3.5      ✓ purrr  0.3.4
## ✓ tibble  3.1.2      ✓ dplyr  1.0.7
## ✓ tidyr   1.1.3      ✓ stringr 1.4.0
## ✓ readr   1.4.0      ✓ forcats 0.5.1

## — Conflicts ————— tidyverse_conflict
s() —
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()

library(skimr)
library(vegan)

## Loading required package: permute

## Loading required package: lattice

## This is vegan 2.5-7

library(forcats)
library(naniar)

##
## Attaching package: 'naniar'

## The following object is masked from 'package:skimr':
##
##     n_complete

library(ggtree)

## ggtree v3.0.2 For help: https://yulab-smu.top/treedata-book/
##
## If you use ggtree in published research, please cite the most appropriate
paper(s):
##
## 1. Guangchuang Yu. Using ggtree to visualize data on tree-like structures.
Current Protocols in Bioinformatics, 2020, 69:e96. doi:10.1002/cpbi.96
## 2. Guangchuang Yu, Tommy Tsan-Yuk Lam, Huachen Zhu, Yi Guan. Two methods f
or mapping and visualizing associated data on phylogeny using ggtree. Molecul
ar Biology and Evolution 2018, 35(12):3041-3043. doi:10.1093/molbev/msy194
## 3. Guangchuang Yu, David Smith, Huachen Zhu, Yi Guan, Tommy Tsan-Yuk Lam.
ggtree: an R package for visualization and annotation of phylogenetic trees w
ith their covariates and other associated data. Methods in Ecology and Evolut
ion 2017, 8(1):28-36. doi:10.1111/2041-210X.12628

```



```
##
## Attaching package: 'ggtree'

## The following object is masked from 'package:tidyr':
##
##     expand

library(ggpubr)

##
## Attaching package: 'ggpubr'

## The following object is masked from 'package:ggtree':
##
##     rotate

library(ggrepel)
library(reshape2)

##
## Attaching package: 'reshape2'

## The following object is masked from 'package:tidyr':
##
##     smiths

library(RColorBrewer)
```

Enter and quality control all genotypic data including: serovar-predictions (generated by SISTR), BAPS level 1 (generated by fastbaps), ST lineages (generated by mlst), and cgMLST variants (generated by SISTR). All input files were generated by the describe programs which are part of the computational platform called ProkEvo.

```
# enter the BAPS data
baps <- read_csv('~/.Documents/jove_paper/data/fastbaps_partition_baps_prior_16.csv')

##
## — Column specification —————
##
## cols(
##   Isolates = col_character(),
##   `Level 1` = col_double(),
##   `Level 2` = col_double(),
##   `Level 3` = col_double(),
##   `Level 4` = col_double(),
##   `Level 5` = col_double(),
##   `Level 6` = col_double()
## )

# check the first six rows of the dataset
head(baps)
```

```
## # A tibble: 6 x 7
##   Isolates   `Level 1` `Level 2` `Level 3` `Level 4` `Level 5` `Level 6`
##   <chr>      <dbl>    <dbl>    <dbl>    <dbl>    <dbl>    <dbl>
## 1 SRR1002805      8      21      44      67      99     134
## 2 SRR1002816      8      25      60     103     167     235
## 3 SRR1002817      1       1       1       1       1       1
## 4 SRR1002827      1       1       1       1       1       1
## 5 SRR1002828      1       1       1       1       1       1
## 6 SRR1002830      8      25      60     103     167     235

# changing the first two column names because for hierarchical analysis we only use BAPS1
colnames(baps)[1:2] <- c("id", "baps1")
# check the first six rows of the dataset again to see the change in column names
head(baps)

## # A tibble: 6 x 7
##   id      baps1 `Level 2` `Level 3` `Level 4` `Level 5` `Level 6`
##   <chr>    <dbl>    <dbl>    <dbl>    <dbl>    <dbl>    <dbl>
## 1 SRR1002805      8      21      44      67      99     134
## 2 SRR1002816      8      25      60     103     167     235
## 3 SRR1002817      1       1       1       1       1       1
## 4 SRR1002827      1       1       1       1       1       1
## 5 SRR1002828      1       1       1       1       1       1
## 6 SRR1002830      8      25      60     103     167     235

# select columns id and baps_1
baps1 <- baps %>%
  select(id, baps1)
# check the first six rows of the dataset
head(baps1)

## # A tibble: 6 x 2
##   id      baps1
##   <chr>    <dbl>
## 1 SRR1002805      8
## 2 SRR1002816      8
## 3 SRR1002817      1
## 4 SRR1002827      1
## 5 SRR1002828      1
## 6 SRR1002830      8

# quality control baps1 data
skim(baps1)
```

Data summary

| | |
|----------------|-------|
| Name | baps1 |
| Number of rows | 2365 |

Number of columns 2

Column type frequency:

character 1


numeric 1

Group variables None

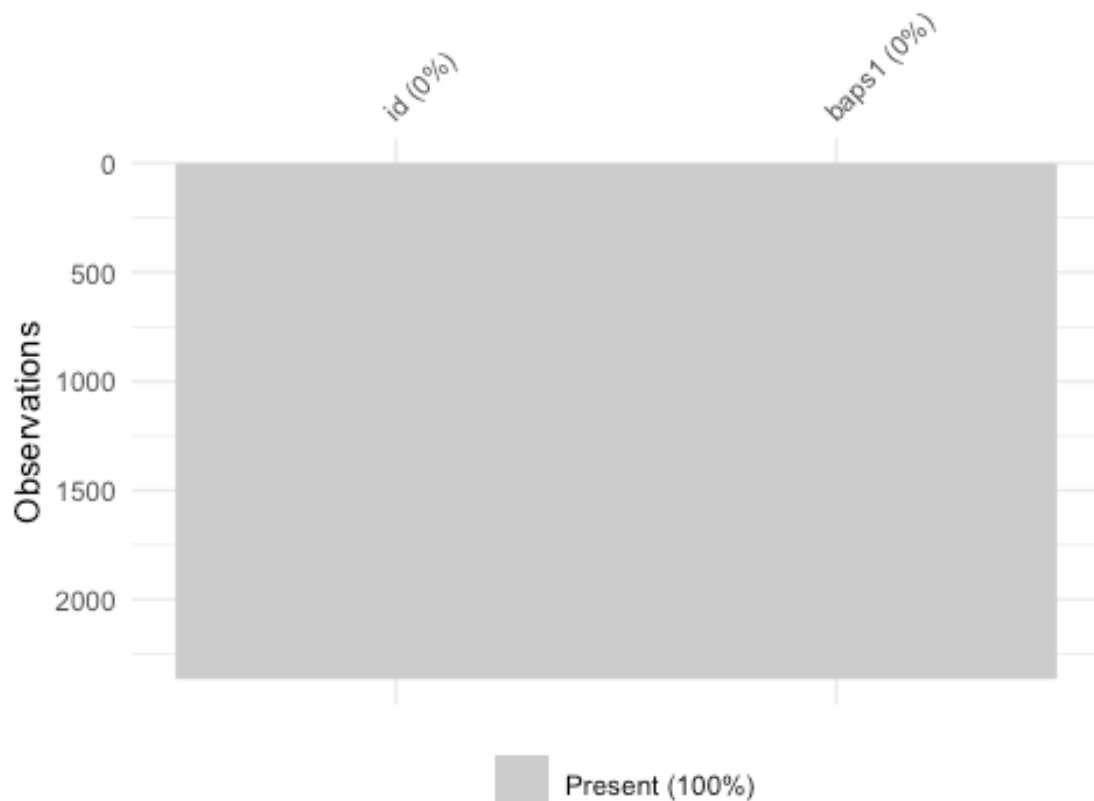
Variable type: character

| skim_variable | n_missing | complete_rate | min | max | empty | n_unique | whitespace |
|---------------|-----------|---------------|-----|-----|-------|----------|------------|
| id | 0 | 1 | 9 | 11 | 0 | 2365 | 0 |

Variable type: numeric

| skim_variable | n_missing | complete_rate | mean | sd | p0 | p2 | p5 | p7 | p10 | hist |
|---------------|-----------|---------------|------|------|----|----|----|----|-----|--|
| baps1 | 0 | 1 | 5.64 | 3.11 | 1 | 1 | 8 | 8 | 9 |  |

```
# using a plotting strategy to check for missing values  
vis_miss(baps1)
```



```
# rename BAPS Level 1 sub-groups or haplotypes
baps1$baps_1 <- ifelse(baps1$baps1 == 1, "BAPS1 sub-group 1", # the ifelse fu
nctions tests for different conditions prior to assigning groups to one categ
ory or another
                      ifelse(baps1$baps1 == 2, "BAPS1 sub-group 2",
                              ifelse(baps1$baps1 == 3, "BAPS1 sub-group 3",
                                      ifelse(baps1$baps1 == 4, "BAPS1 sub-group 4"
,
                              ifelse(baps1$baps1 == 5, "BAPS1 sub-g
roup 5",
                              ifelse(baps1$baps1 == 6, "BAPS
1 sub-group 6",
                              ifelse(baps1$baps1 ==
7, "BAPS1 sub-group 7",
                              ifelse(baps1$b
aps1 == 8, "BAPS1 sub-group 8", "BAPS1 sub-group 9"))))))))
#####-----#####
#####
#####-----#####
#####
# enter MLST results
mlst <- read_csv('~/Documents/jove_paper/data/salmonellast_output.csv')
```

```
##
## — Column specification —————
## cols(
##   FILE = col_character(),
##   SCHEME = col_character(),
##   ST = col_character(),
##   aroC = col_character(),
##   dnaN = col_character(),
##   hemD = col_character(),
##   hisD = col_character(),
##   purE = col_character(),
##   sucA = col_character(),
##   thrA = col_character()
## )

# check the first six rows of the dataset
head(mlst)

## # A tibble: 6 x 10
##   FILE          SCHEME  ST   aroC  dnaN  hemD  hisD  purE  sucA
##   <chr>          <chr>   <chr> <chr> <chr> <chr> <chr> <chr> <chr>
## 1 SRR1425284_contigs.f... senteri... 45    10    7    21    14    15    12
## 2 SRR5760393_contigs.f... senteri... 2930  16    2    45   744    36    39
## 3 SRR5851178_contigs.f... senteri... 118   16    2    45   43    36    39
## 4 SRR5935662_contigs.f... senteri... 118   16    2    45   43    36    39
## 5 SRR6881504_contigs.f... senteri... 118   16    2    45   43    36    39
## 6 SRR5908147_contigs.f... senteri... 132    2   57   15   14    15    20

# generate the id column by deriving it from the FILE column
mlst$id <- sapply(strsplit(as.character(mlst$FILE), '_'), "[", 1)
# check the first six rows of the dataset
head(mlst)

## # A tibble: 6 x 11
##   FILE          SCHEME  ST   aroC  dnaN  hemD  hisD  purE  sucA  thrA  id
##   <chr>          <chr>   <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr>
## 1 SRR1425284_co... senteri... 45    10    7    21    14    15    12    12    SRR142...
## 2 SRR5760393_co... senteri... 2930  16    2    45   744    36    39    42    SRR576...
```

```
## 3 SRR5851178_co... sender... 118 16 2 45 43 36 39 42 S
RR585...
## 4 SRR5935662_co... sender... 118 16 2 45 43 36 39 42 S
RR593...
## 5 SRR6881504_co... sender... 118 16 2 45 43 36 39 42 S
RR688...
## 6 SRR5908147_co... sender... 132 2 57 15 14 15 20 12 S
RR590...
```

```
# select the id and ST columns
```

```
mlst1 <- mlst %>%
```

```
  select(id, ST) # select id and ST columns
```

```
# use the table() function to detect extraneous characters such as "-" or "?"
in the data - those are ST misclassifications
```

```
# check for the presence of ST misclassification in the ST column
```

```
table(mlst1$ST)
```

```
##
```

```
## - 11 112 118 13 132 1370 138 14 15 164 166 1674 19 2129
2132
```

```
## 4 4 2 800 1 192 1 1 2 2 2 9 1 5 1
2
```

```
## 2166 223 23 2362 2370 2371 24 27 2855 2930 3045 31 32 3242 3494
350
```

```
## 1 1 2 3 4 2 1 1 1 1 1 42 1 1 1
40
```

```
## 367 371 3783 3834 3865 40 413 4153 4166 4190 4219 435 4450 4493 45
450
```

```
## 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 643
1
```

```
## 46 4621 4628 4640 471 5 548 582 614 680 816 83 95
```

```
## 31 2 1 1 1 529 1 1 1 1 1 4 1
```

```
# check for missing values
```

```
skim(mlst1)
```

Data summary

Name mlst1

Number of rows 2365

Number of columns 2

Column type frequency:

character 2

Group variables None

Variable type: character

| skim_variable | n_missing | complete_rate | min | max | empty | n_unique | whitespace |
|---------------|-----------|---------------|-----|-----|-------|----------|------------|
| id | 0 | 1 | 9 | 11 | 0 | 2365 | 0 |
| ST | 0 | 1 | 1 | 4 | 0 | 61 | 0 |

```
# mutate the "-" character to NA (missing value)
mlst1 <- mlst1 %>%
  mutate_all(na_if, "-") # fill all - with NAs
# check the first six rows of the dataset
head(mlst1)

## # A tibble: 6 x 2
##   id      ST
##   <chr>   <chr>
## 1 SRR1425284 45
## 2 SRR5760393 2930
## 3 SRR5851178 118
## 4 SRR5935662 118
## 5 SRR6881504 118
## 6 SRR5908147 132

# quality control baps1 data
skim(mlst1)
```

Data summary

| | |
|-------------------|-------|
| Name | mlst1 |
| Number of rows | 2365 |
| Number of columns | 2 |

Column type frequency:

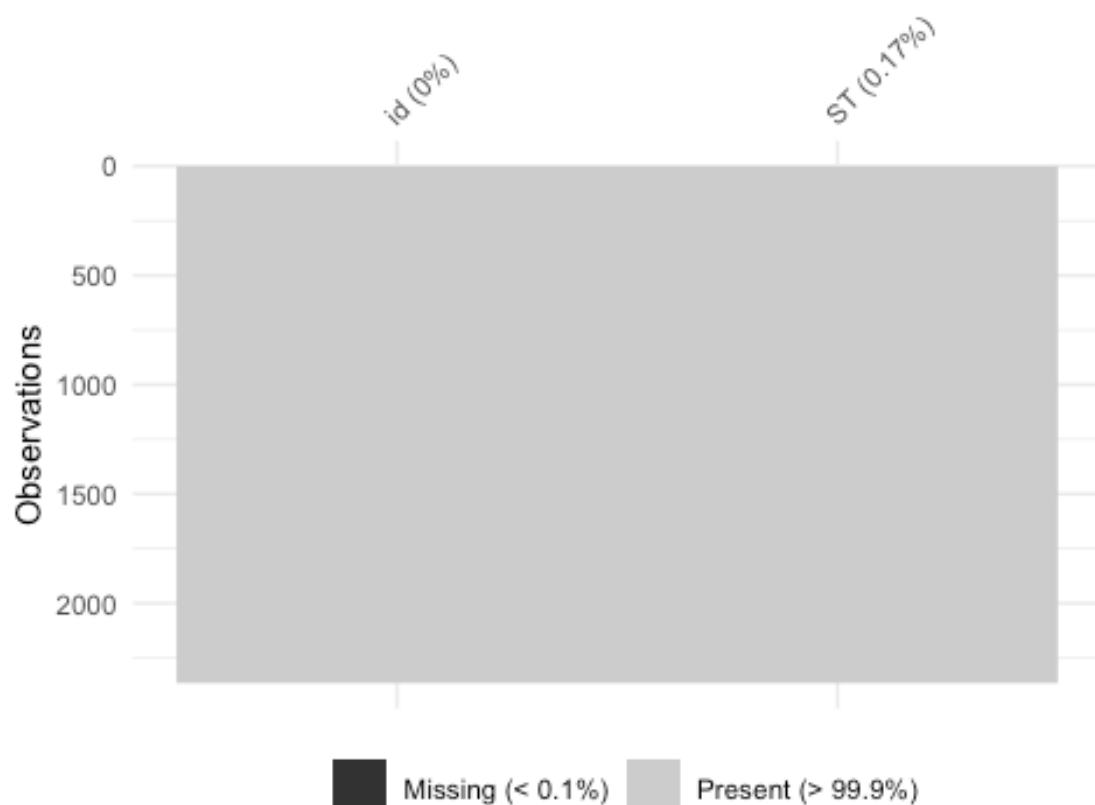
| | |
|-----------|---|
| character | 2 |
|-----------|---|

| | |
|-----------------|------|
| Group variables | None |
|-----------------|------|

Variable type: character

| skim_variable | n_missing | complete_rate | min | max | empty | n_unique | whitespace |
|---------------|-----------|---------------|-----|-----|-------|----------|------------|
| id | 0 | 1 | 9 | 11 | 0 | 2365 | 0 |
| ST | 4 | 1 | 1 | 4 | 0 | 60 | 0 |

```
# using a plotting strategy to check for missing values
vis_miss(mlst1)
```



At this stage missing values are not removed or dealt with until datasets are merged

```
#####-----#####
#####
#####-----#####
#####
```

Enter SISTR results

```
sistr <- read_csv('~/Documents/jove_paper/data/sistr_output.csv')
```

```
##
```

```
## — Column specification —————
```

```
## cols(
##   cgmlst_ST = col_double(),
##   cgmlst_distance = col_double(),
##   cgmlst_genome_match = col_character(),
##   cgmlst_matching_alleles = col_double(),
##   cgmlst_subspecies = col_character(),
##   fasta_filepath = col_character(),
##   genome = col_character(),
##   h1 = col_character(),
##   h2 = col_character(),
##   o_antigen = col_character(),
```



```

## qc_messages = col_character(),
## qc_status = col_character(),
## serogroup = col_character(),
## serovar = col_character(),
## serovar_antigen = col_character(),
## serovar_cgmlst = col_character()
## )

# check the first six rows of the dataset
head(sistr)

## # A tibble: 6 x 16
##   cgmlst_ST cgmlst_distance cgmlst_genome_m... cgmlst_matching_... cgmlst_sub
speci...
##   <dbl>          <dbl> <chr>          <dbl> <chr>
## 1      NA          0.00909 SAL_DA6272AA      327 enterica
## 2 1297108188      0.00303 SAL_BA3042AA      329 enterica
## 3  279966480      0.00606 SRR1122516        328 enterica
## 4  4017665136      0.0606  SAL_EA9357AA      310 enterica
## 5  2221566091      0.00606 SAL_EA2811AA      328 enterica
## 6  3336043520      0.00606 SAL_FA5137AA      328 enterica
## # ... with 11 more variables: fasta_filepath <chr>, genome <chr>, h1 <chr>,
## #   h2 <chr>, o_antigen <chr>, qc_messages <chr>, qc_status <chr>,
## #   serogroup <chr>, serovar <chr>, serovar_antigen <chr>, serovar_cgmlst
<chr>

# generate the id column by deriving it from the genome column
sistr$id <- sapply(strsplit(as.character(sistr$genome), '_'), "[", 1)
# select the id and cgmlst_ST columns
sistr1 <- sistr %>%
  select(id, serovar_cgmlst, cgmlst_ST) # select id, serovar_cgmlst
, and cgmlst_ST columns
# check the first six rows of the dataset
head(sistr1)

## # A tibble: 6 x 3
##   id          serovar_cgmlst cgmlst_ST
##   <chr>          <chr>          <dbl>
## 1 SRR1425284 Newport              NA
## 2 SRR5760393 Newport      1297108188
## 3 SRR5851178 Newport      279966480
## 4 SRR5935662 Newport      4017665136
## 5 SRR6881504 Newport      2221566091
## 6 SRR5908147 Newport      3336043520

# quality control baps1 data
skim(sistr1)

```

Data summary

| | |
|------|--------|
| Name | sistr1 |
|------|--------|

Number of rows 2365
Number of columns 3

Column type frequency:




character 2
numeric 1

Group variables None

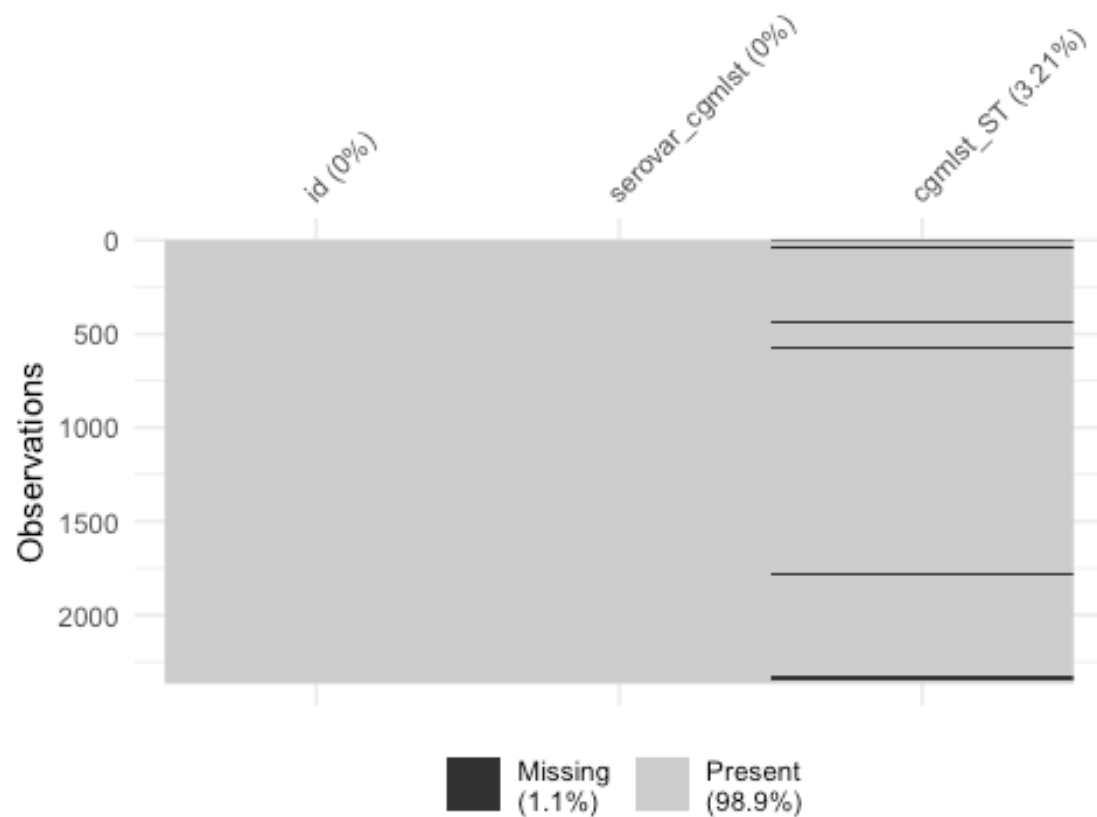
Variable type: character

| skim_variable | n_missing | complete_rate | min | max | empty | n_unique | whitespace |
|----------------|-----------|---------------|-----|-----|-------|----------|------------|
| id | 0 | 1 | 9 | 11 | 0 | 2365 | 0 |
| serovar_cgmlst | 0 | 1 | 4 | 15 | 0 | 28 | 0 |

Variable type: numeric

| skim_variable | n_missing | complete_rate | mean | sd | p0 | p25 | p50 | p75 | p100 | hist |
|---------------|-----------|---------------|-------|-------|-----|-------|-------|-------|-------|---|
| cgmlst | 76 | 0.97 | 19987 | 11508 | 183 | 12711 | 16352 | 31089 | 42880 |  |
| _ST | | | 99689 | 24697 | 768 | 56802 | 71886 | 47600 | 28711 |  |
| | | | | | 5 | | | | |  |

```
# using a plotting strategy to check for missing values  
vis_miss(sistr1)
```



```
# At this stage missing values are not removed or dealt with until datasets are merged
#####-----#####
#####
#####-----#####
#####
# combine baps1, mlst1, and sistr1 datasets
d1 <- left_join(baps1, mlst1, on = "id") # merge datasets on identical ids
## Joining, by = "id"
d2 <- left_join(d1, sistr1, on = "id") # merge datasets on identical ids
## Joining, by = "id"
# check data dimensionality
dim(d2)
## [1] 2365    6
# check the first six rows of the dataset
head(d2)
## # A tibble: 6 x 6
##   id          baps1 baps_1      ST  serovar_cgmlst cgmlst_ST
```

```
##      <chr>      <dbl> <chr>      <chr> <chr>      <dbl>
## 1 SRR1002805      8 BAPS1 sub-group 8 118 Newport 1089389973
## 2 SRR1002816      8 BAPS1 sub-group 8 118 Newport 1837685
## 3 SRR1002817      1 BAPS1 sub-group 1 45 Newport 1468400426
## 4 SRR1002827      1 BAPS1 sub-group 1 45 Newport 1468400426
## 5 SRR1002828      1 BAPS1 sub-group 1 45 Newport 1468400426
## 6 SRR1002830      8 BAPS1 sub-group 8 118 Newport 1837685

# quality control baps1 data
skim(d2)
```

Data summary

| | |
|-------------------|------|
| Name | d2 |
| Number of rows | 2365 |
| Number of columns | 6 |

Column type frequency:

| | |
|-----------|---|
| character | 4 |
| numeric | 2 |

| | |
|-----------------|------|
| Group variables | None |
|-----------------|------|

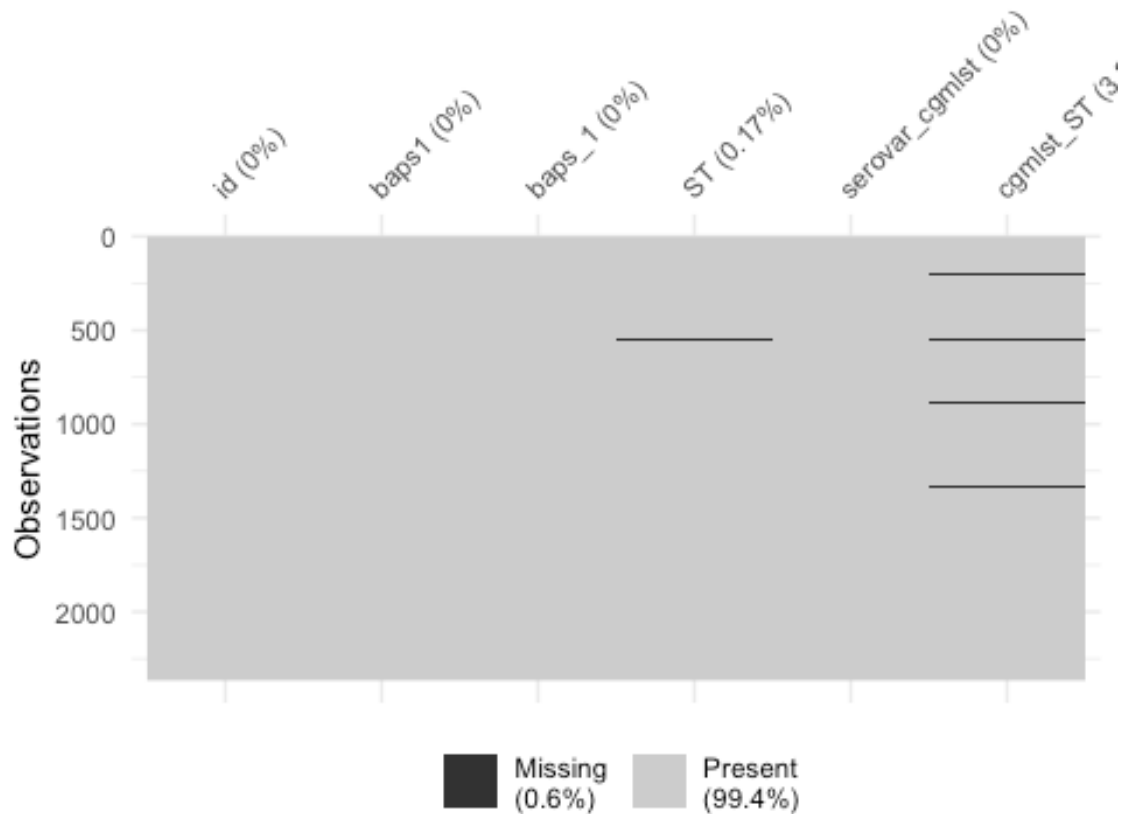
Variable type: character

| skim_variable | n_missing | complete_rate | min | max | empty | n_unique | whitespace |
|----------------|-----------|---------------|-----|-----|-------|----------|------------|
| id | 0 | 1 | 9 | 11 | 0 | 2365 | 0 |
| baps_1 | 0 | 1 | 17 | 17 | 0 | 9 | 0 |
| ST | 4 | 1 | 1 | 4 | 0 | 60 | 0 |
| serovar_cgmlst | 0 | 1 | 4 | 15 | 0 | 28 | 0 |

Variable type: numeric

| skim_variable | n_missing | complete_rate | mean | sd | p0 | p25 | p50 | p75 | p100 | hist |
|---------------|-----------|---------------|------------|------------|--------|----------|----------|----------|----------|------|
| baps1 | 0 | 1.00 | 5.6400e+00 | 3.1100e+00 | 1 | 1 | 8 | 8 | 9 | |
| cgmlst_ST | 76 | 0.97 | 1.9988e+09 | 1.1508e+09 | 183768 | 12711568 | 16352718 | 31089476 | 42880287 | |

```
# using a plotting strategy to check for missing values
vis_miss(d2)
```



```
# At this stage missing values are not removed or dealt with until datasets are merged
```

```
#####-----#####
#####
#####-----#####
#####
```

```
# group all genomes not classified as Newport as "Other serovars"
```

```
# create a new column called serovar that contains the binary classification
to group all SISTR-based misclassified genomes
```

```
d2$serovar <- ifelse(d2$serovar_cgmlst == "Newport", "Newport",
                    "Other serovars") # classify serovar_cgmlst into
```

```
Newport or Others
```

```
# check the first six rows of the dataset
```

```
head(d2)
```

```
## # A tibble: 6 x 7
```

```
##   id          baps1 baps_1          ST   serovar_cgmlst cgmlst_ST serov
##   <chr>      <dbl> <chr>          <chr> <chr>          <dbl> <chr>
## 1 SRR1002805      8 BAPS1 sub-group 8 118   Newport      1089389973 Newport
```

```

rt
## 2 SRR1002816      8 BAPS1 sub-group 8 118      Newport      1837685 Newpo
rt
## 3 SRR1002817      1 BAPS1 sub-group 1 45      Newport      1468400426 Newpo
rt
## 4 SRR1002827      1 BAPS1 sub-group 1 45      Newport      1468400426 Newpo
rt
## 5 SRR1002828      1 BAPS1 sub-group 1 45      Newport      1468400426 Newpo
rt
## 6 SRR1002830      8 BAPS1 sub-group 8 118      Newport      1837685 Newpo
rt

```

```

# check the composition of the serovar column
table(d2$serovar)

```

```

##
##      Newport Other serovars
##      2317      48

```

```

#####-----#####
#####
#####-----#####
#####

```

```

# no transformation is needed for the baps1 dataset - no groupings or aggrega
tions are needed

```

```

#####-----#####
#####
#####-----#####
#####

```

```

# check ST distribution to focus on major STs for all subsequent analyses by
grouping by ST and counting

```

```

st_dist <- d2 %>%
  group_by(ST) %>% # group by the ST column
  count() %>% # count the number of observations
  arrange(desc(n)) # arrange the counts in decreasing order
st_dist

```

```

## # A tibble: 61 x 2
## # Groups:   ST [61]
##   ST      n
##   <chr> <int>
## 1 118    800
## 2 45     643
## 3 5      529
## 4 132    192
## 5 31     42
## 6 350    40
## 7 46     31
## 8 166     9
## 9 19      5

```

```
## 10 11      4
## # ... with 51 more rows

# based on the frequency analysis, the following STs were not aggregated: ST
118, ST45, ST5, ST132, ST31, ST350, and ST46
# create a new st column
d2$st <- ifelse(d2$ST == 5, "ST5", # create a new ST column for which minor S
Ts are aggregated as Others
              ifelse(d2$ST == 31, "ST31",
                    ifelse(d2$ST == 45, "ST45",
                          ifelse(d2$ST == 46, "ST46",
                                ifelse(d2$ST == 118, "ST118",
                                      ifelse(d2$ST == 132, "ST132",
                                            ifelse(d2$ST == 350, "ST350", "Other STs"))))))))
# check the first six rows of the dataset
head(d2)

## # A tibble: 6 x 8
##   id      baps1 baps_1      ST  serovar_cgmlst cgmlst_ST serovar
##   <chr>    <dbl> <chr>    <chr> <chr>          <dbl> <chr>
##   <chr>
## 1 SRR1002805      8 BAPS1 sub-group... 118  Newport          1.09e9 Newport
ST118
## 2 SRR1002816      8 BAPS1 sub-group... 118  Newport          1.84e6 Newport
ST118
## 3 SRR1002817      1 BAPS1 sub-group... 45   Newport          1.47e9 Newport
ST45
## 4 SRR1002827      1 BAPS1 sub-group... 45   Newport          1.47e9 Newport
ST45
## 5 SRR1002828      1 BAPS1 sub-group... 45   Newport          1.47e9 Newport
ST45
## 6 SRR1002830      8 BAPS1 sub-group... 118  Newport          1.84e6 Newport
ST118

#####-----#####
#####
#####-----#####
#####
# check cgMLST distribution by ST to focus on major cgMLSTs for all subsequen
t analyses by grouping by STs and cgMLSTs and counting
cgmlst_dist <- d2 %>%
  group_by(st, cgmlst_ST) %>% # group by st and cgmlst_ST
  count() %>% # count the number of observations
  arrange(desc(n)) # arrange in descending order
cgmlst_dist

## # A tibble: 777 x 3
## # Groups:   st, cgmlst_ST [777]
##   st      cgmlst_ST      n
##   <chr>    <dbl> <int>
```

```

## 1 ST45 1468400426 319
## 2 ST118 1271156802 96
## 3 ST132 3336043520 86
## 4 ST5 88443731 84
## 5 ST118 1492716119 64
## 6 ST5 3491314984 50
## 7 ST45 2245200879 39
## 8 ST118 2103519905 34
## 9 ST132 3109396922 34
## 10 ST5 3108947600 34
## # ... with 767 more rows

# for the purposes of this analysis only the top four most frequent cgMLSTs were selected and respectively renamed
d2 <- mutate(d2, cgmlst = ifelse(cgmlst_ST %in% 1468400426, "cgMLST 1468400426", # create a new cgMLST column while aggregating minor cgMLST variants
                                ifelse(cgmlst_ST %in% 88443731, "cgMLST 88443731",
                                ifelse(cgmlst_ST %in% 1271156802, "cgMLST 1271156802",
                                ifelse(cgmlst_ST %in% 3336043520, "cgMLST 3336043520",
                                "Other cgMLSTs")))))

# check the first six rows of the dataset
head(d2)

## # A tibble: 6 x 9
##   id      baps1 baps_1 ST serovar_cgmlst cgmlst_ST serovar st cgmlst
##   <chr>   <dbl> <chr>   <chr> <chr>          <dbl> <chr>   <chr> <chr>
## 1 SRR100... 8 BAPS1 su... 118 Newport 1.09e9 Newport ST118 Other cg...
## 2 SRR100... 8 BAPS1 su... 118 Newport 1.84e6 Newport ST118 Other cg...
## 3 SRR100... 1 BAPS1 su... 45 Newport 1.47e9 Newport ST45 cgMLST 1...
## 4 SRR100... 1 BAPS1 su... 45 Newport 1.47e9 Newport ST45 cgMLST 1...
## 5 SRR100... 1 BAPS1 su... 45 Newport 1.47e9 Newport ST45 cgMLST 1...
## 6 SRR100... 8 BAPS1 su... 118 Newport 1.84e6 Newport ST118 Other cg...

#####-----#####
#####
#####-----#####
#####

# select only needed columns for all subsequent analyses
d3 <- d2 %>%

```



```

      select(id, serovar, baps_1, st, cgmlst) # select columns of interest which are described within parenthesis
# check data dimensionality to make sure it matches that of d2
dim(d3)

## [1] 2365      5

# check the first six rows of the dataset
head(d3)

## # A tibble: 6 x 5
##   id          serovar baps_1          st    cgmlst
##   <chr>        <chr>   <chr>        <chr> <chr>
## 1 SRR1002805 Newport BAPS1 sub-group 8 ST118 Other cgMLSTs
## 2 SRR1002816 Newport BAPS1 sub-group 8 ST118 Other cgMLSTs
## 3 SRR1002817 Newport BAPS1 sub-group 1 ST45  cgMLST 1468400426
## 4 SRR1002827 Newport BAPS1 sub-group 1 ST45  cgMLST 1468400426
## 5 SRR1002828 Newport BAPS1 sub-group 1 ST45  cgMLST 1468400426
## 6 SRR1002830 Newport BAPS1 sub-group 8 ST118 Other cgMLSTs

# quality control baps1 data
skim(d3)

```

Data summary

| | |
|-------------------|------|
| Name | d3 |
| Number of rows | 2365 |
| Number of columns | 5 |

Column type frequency:

| | |
|-----------|---|
| character | 5 |
|-----------|---|

| | |
|-----------------|------|
| Group variables | None |
|-----------------|------|

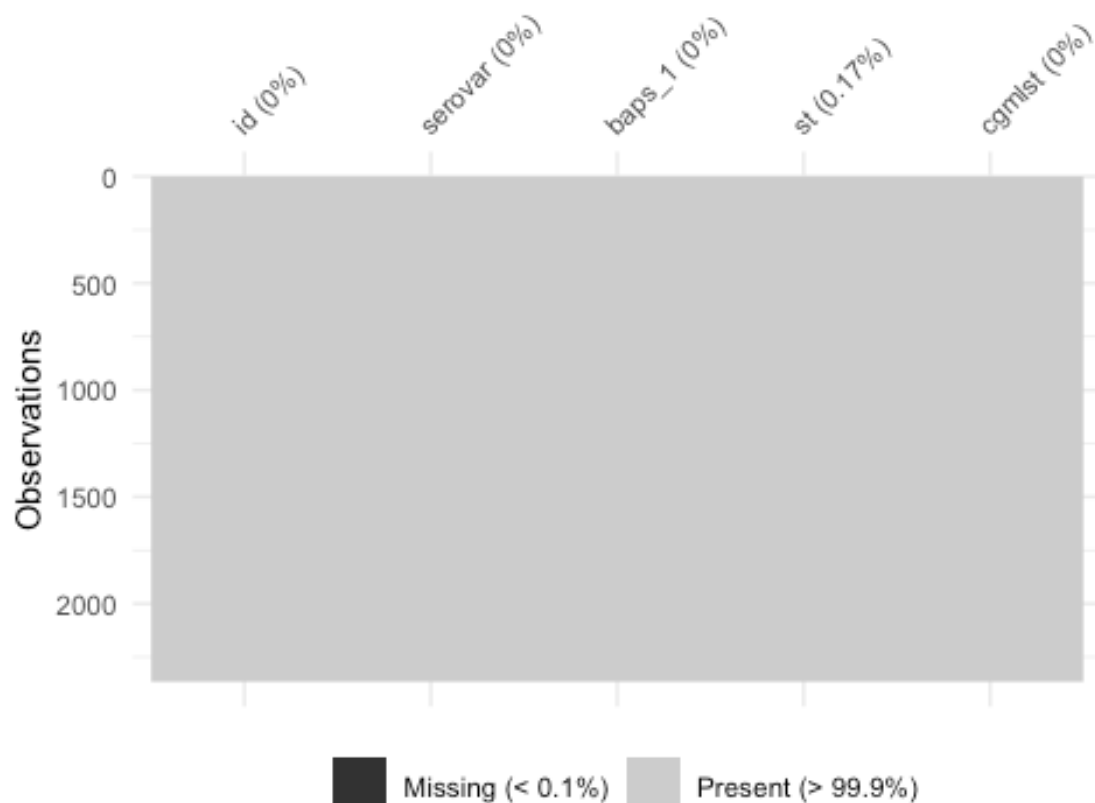
Variable type: character

| skim_variable | n_missing | complete_rate | min | max | empty | n_unique | whitespace |
|---------------|-----------|---------------|-----|-----|-------|----------|------------|
| id | 0 | 1 | 9 | 11 | 0 | 2365 | 0 |
| serovar | 0 | 1 | 7 | 14 | 0 | 2 | 0 |
| baps_1 | 0 | 1 | 17 | 17 | 0 | 9 | 0 |
| st | 4 | 1 | 3 | 9 | 0 | 8 | 0 |
| cgmlst | 0 | 1 | 13 | 17 | 0 | 5 | 0 |

```

# using a plotting strategy to check for missing values
vis_miss(d3)

```



```
# make sure there is no NA in the dataset
sum(is.na(d3))

## [1] 4

# replace NA in the ST column
d3 <- d3 %>% mutate(st = replace_na(st, "Other STs"))
# make sure there is no NA in the dataset
sum(is.na(d3))

## [1] 0

# check the distribution of serovars, baps1, st, and cgmlst classifications
table(d3$serovar)

##
##      Newport Other serovars
##      2317          48

table(d3$baps_1)

##
## BAPS1 sub-group 1 BAPS1 sub-group 2 BAPS1 sub-group 3 BAPS1 sub-group 4
##           648           2           7           235
## BAPS1 sub-group 5 BAPS1 sub-group 6 BAPS1 sub-group 7 BAPS1 sub-group 8
```

```
##          9          9          32          1394
## BAPS1 sub-group 9
##          29

table(d3$st)

##
## Other STs      ST118      ST132      ST31      ST350      ST45      ST46
ST5
##          88          800          192          42          40          643          31
529

table(d3$cgmlst)

##
## cgMLST 1271156802 cgMLST 1468400426 cgMLST 3336043520 cgMLST 88443731
##          97          321          86          85
## Other cgMLSTs
##          1776

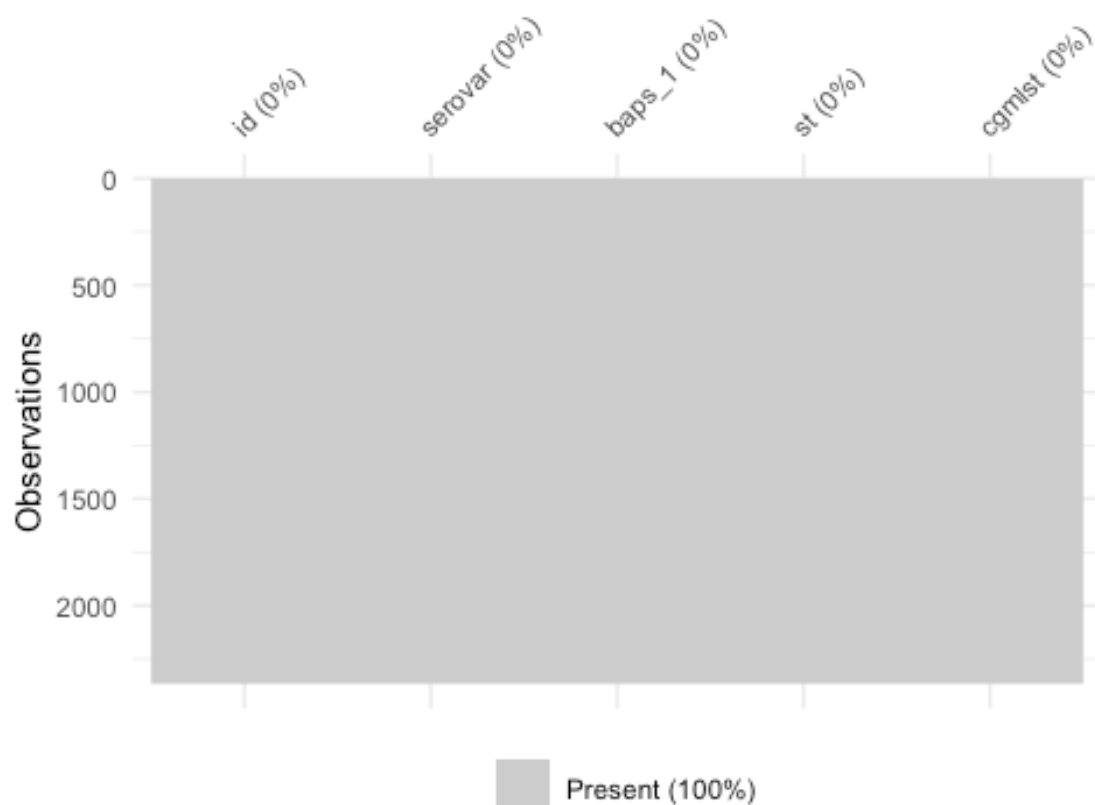
# check the first six rows in the dataset
head(d3)

## # A tibble: 6 x 5
##   id          serovar baps_1          st    cgmlst
##   <chr>      <chr>   <chr>      <chr> <chr>
## 1 SRR1002805 Newport BAPS1 sub-group 8 ST118 Other cgMLSTs
## 2 SRR1002816 Newport BAPS1 sub-group 8 ST118 Other cgMLSTs
## 3 SRR1002817 Newport BAPS1 sub-group 1 ST45 cgMLST 1468400426
## 4 SRR1002827 Newport BAPS1 sub-group 1 ST45 cgMLST 1468400426
## 5 SRR1002828 Newport BAPS1 sub-group 1 ST45 cgMLST 1468400426
## 6 SRR1002830 Newport BAPS1 sub-group 8 ST118 Other cgMLSTs

# check the last six rows in the dataset
tail(d3)

## # A tibble: 6 x 5
##   id          serovar baps_1          st    cgmlst
##   <chr>      <chr>   <chr>      <chr> <chr>
## 1 SRR952683 Newport BAPS1 sub-group 8 ST118 Other cgMLSTs
## 2 SRR953548 Newport BAPS1 sub-group 1 ST45 Other cgMLSTs
## 3 SRR953551 Newport BAPS1 sub-group 1 ST45 cgMLST 1468400426
## 4 SRR980337 Newport BAPS1 sub-group 1 ST45 cgMLST 1468400426
## 5 SRR980338 Newport BAPS1 sub-group 1 ST45 cgMLST 1468400426
## 6 SRR980354 Newport BAPS1 sub-group 1 ST45 cgMLST 1468400426

# check for missing values
vis_miss(d3)
```



```
# check the data characteristics
```

```
str(d3)
```

```
## tibble [2,365 × 5] (S3: tbl_df/tbl/data.frame)
```

```
## $ id      : chr [1:2365] "SRR1002805" "SRR1002816" "SRR1002817" "SRR1002827" ...
```

```
## $ serovar: chr [1:2365] "Newport" "Newport" "Newport" "Newport" ...
```

```
## $ baps_1 : chr [1:2365] "BAPS1 sub-group 8" "BAPS1 sub-group 8" "BAPS1 sub-group 1" "BAPS1 sub-group 1" ...
```

```
## $ st      : chr [1:2365] "ST118" "ST118" "ST45" "ST45" ...
```

```
## $ cgmlst  : chr [1:2365] "Other cgMLSTs" "Other cgMLSTs" "cgMLST 1468400426" "cgMLST 1468400426" ...
```

Figure 1. Visualize the phylogeny-based hierarchical distribution of genotypes.

At the center of the figure is the core-genome phylogeny. Genotypic information is plotted onto it in the following order: Serovar (innermost) BAPS1 ST cgMLST (outermost)

```
# enter the phylogeny data
```

```
tree <- read.tree("~/Documents/jove_paper/data/newport_phylogeny.tree")
```

```
# recall the metadata combining all hierarchical genotypes
```

```
# here I assign d3 to d4 just to keep the formatted metadata read for other steps, in case I need to do
```

```
# any transformation with the data to plot with the phylogenetic tree
```

```

d4 <- d3
# to plot with the phylogeny using ggtree we need to make the id column into
index first
d4 <- column_to_rownames(d4, var = "id")
# create the tree
# adjusting the parameters to make the tree visible or however you wish requi
res some trial and error
tree_plot <- ggtree(tree, layout = "circular") + xlim(-250, NA)
# plot figure 1 - color scheme for each layer of the plot should be chosen ba
sed on the user preferences
figure_1 <- gheatmap(tree_plot, d4, offset=.0, width=20, colnames = FALSE) +
# visualize the tree with metadata
  scale_fill_manual(values = c("coral", "darkblue",
                                "cornflowerblue", "coral", "purple", "red", "b
rown", "darkseagreen3", "darkblue", "darkgreen", "yellow",
                                "orange", "darkblue", "purple", "darkgreen", "
darkred", "steelblue", "black", "coral",
                                "black", "darkgreen", "purple", "darkblue", "g
ray"),
                    breaks = c("Newport", "Other serovars",
                                "BAPS1 sub-group 1", "BAPS1 sub-group 2", "BAP
S1 sub-group 3", "BAPS1 sub-group 4",
                                "BAPS1 sub-group 5", "BAPS1 sub-group 6", "BAP
S1 sub-group 7", "BAPS1 sub-group 8",
                                "BAPS1 sub-group 9",
                                "ST5", "ST31", "ST45", "ST46", "ST118", "ST132
", "ST350", "Other STs",
                                "cgMLST 1468400426", "cgMLST 1271156802", "cgM
LST 3336043520", "cgMLST 88443731", "Other cgMLSTs"),
                    name="Serovar (innermost) -> BAPS1 -> ST -> cgMLST (outer
most)") + # add colors and labels for the scale
  ggtitle(expression(bold(paste("Hierarchical population structure - ", bol
ditalic(S), ". Newport")))) + # add title for the figure
  theme(plot.title = element_text(size = 40, face = "bold"),
        legend.title=element_text(size=24, face = "bold"),
        legend.text=element_text(size=22)) # customize figure's title, legend, f
ont

## Scale for 'y' is already present. Adding another scale for 'y', which will
## replace the existing scale.

## Scale for 'fill' is already present. Adding another scale for 'fill', whic
h
## will replace the existing scale.

figure_1

```

Hierarchical population structure - S. Newport

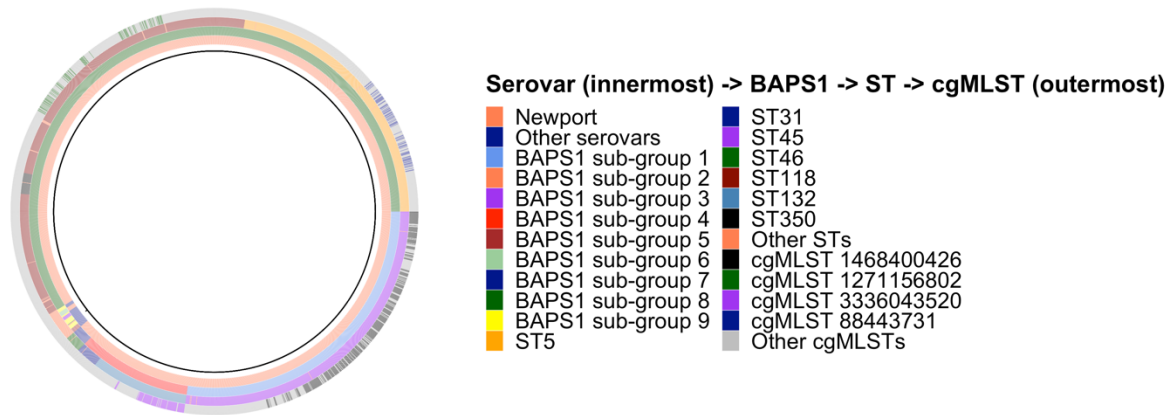


Figure 2. Visualize the frequency-based distribution of hierarchical genotypes. Note - only genomes classified as S. Newport by SISTR within ProkEvo are included in this analysis.

```
# bring back the metadata file
d5 <- d3
#####-----#####
#####
#####-----#####
#####
# plot Serovar distribution
# calculate the relative frequencies for serovar
# drop the serovars with NAs, group by serovar and then calculate the frequencies
serovar_data <- d5 %>%
  drop_na(serovar) %>% # remove NA based on serovar
  select(serovar) %>% # select the serovar column
  group_by(serovar) %>% # group_by serovar
  summarise(n = n()) %>% # count the number of genomes per serovar
# classification
  mutate(prop_serovar = n/sum(n)*100) # calculate the proportion
# of each serovar
# order serovar groups
serovar_data$serovar <- factor(serovar_data$serovar, levels=c("Other serovars", "Newport"))
# plot the data
sero_plot <- ggplot(serovar_data, aes(x = prop_serovar, y = serovar)) + # show serovars on y-axis and proportion on x-axis
```

```

  ylab("") + xlab("Proportion") + xlim(0, 100) + # set labels for axis and limit for x-axis
  ggtitle("A") + # add title for the plot
  theme_bw() + # choose the plot background
  theme(legend.position = "none") + # remove legend
  theme(axis.text.y = element_text(size = 30)) + # set the font size for y-axis text
  theme(axis.title.y = element_text(size = 30, face = "bold")) + # set the font size for y-axis title
  theme(axis.title.x = element_text(size = 30, face = "bold")) + # set the font size for x-axis title
  theme(axis.text.x = element_text(angle = 0, hjust = 0.7, size = 26)) + # set the font size for x-axis text
  theme(plot.title = element_text(size = 40, face = "bold")) + # customize plot's title, legend, font
  geom_col(fill = "steelblue") # fill the bars that represent the values with blue
#####-----#####
#####
#####-----#####
#####
# plot BAPS1 sub-group distribution for Newport data only
# calculate the relative frequencies for serovar and BAPS1 sub-group
# drop the serovars and BAPS1 sub-groups records with NAs, group by serovar and BAPS1 sub-group and then calculate the frequencies
baps_data <- d5 %>%
  drop_na(serovar, baps_1) %>% # drop NAs for serovar and baps_1
  columns
  select(serovar, baps_1) %>% # select serovar and baps_1 columns
  group_by(serovar, baps_1) %>% # group data based on serovar and
  baps_1
  summarise(n = n()) %>% # count the number of observations per group
  group
  mutate(prop = n/sum(n)*100) %>% # calculate the proportion for each group
  filter(serovar != "Other serovars") # filter out genomes classified as Other serovars

## `summarise()` has grouped output by 'serovar'. You can override using the `.groups` argument.

# re-order the baps_1 column
baps_data$baps_1 <- factor(baps_data$baps_1, levels=c("BAPS1 sub-group 6", "BAPS1 sub-group 7", "BAPS1 sub-group 4", "BAPS1 sub-group 1", "BAPS1 sub-group 8"))
# plot the data
baps_plot <- ggplot(baps_data, aes(x = baps_1, y = prop)) + # show BAPS sub-groups on x-axis and proportion on y-axis
  xlab("") + ylab("Proportion") + ylim(0, 105) + # set labels for axis and limit for y-axis

```

```

ggtitle("B") + # add title for the plot
theme_bw() + # setting plot background
theme(legend.position = "none") + # remove Legend
theme(axis.text.y = element_text(size = 30)) + # change font size for y-axis text
theme(axis.title.y = element_text(size = 30, face = "bold")) + # change font size for y-axis title
theme(axis.title.x = element_text(size = 30, face = "bold")) + # change font size for x-axis title
theme(axis.text.x = element_text(angle = 0, size = 30)) + # change font size for x-axis text
theme(plot.title = element_text(size = 40, face = "bold")) + # customize plot's title, legend, font
geom_col(fill = "steelblue") + # fill the bars that represent the values with blue
coord_flip() # flip x and y axis
#####-----#####
#####
#####-----#####
#####
# plot ST distribution
# calculate the relative frequencies for serovar and ST
# drop the serovars and STs with NAs, group by serovar and ST and then calculate the frequencies
st_data <- d5 %>%
  drop_na(serovar, st) %>% # drop NAs for serovar and st columns
  select(serovar, st) %>% # select serovar and st columns
  group_by(serovar, st) %>% # group data based on serovar and st columns
  summarise(n = n()) %>% # count the number of observations
  mutate(prop_st = n/sum(n)*100) %>% # calculate the proportion by groups
  filter(serovar != "Other serovars") # filter out data for Other serovars

## `summarise()` has grouped output by 'serovar'. You can override using the `.groups` argument.

# re-order the st column
st_data$st <- factor(st_data$st, levels=c("Other STs", "ST46", "ST350", "ST31", "ST132", "ST5", "ST45", "ST118"))

# plot the data
st_plot <- ggplot(st_data, aes(x = st, y = prop_st)) + # show STs on x-axis and proportion on y-axis
  xlab("") + ylab("Proportion") + ylim(0, 105) + # set labels for axis and limit for y-axis
  ggtitle("C") + # add title for the plot
  theme_bw() + # set the plot background
  theme(legend.position = "none") + # remove Legends

```



```

  theme(axis.text.y = element_text(size = 28)) + # change font size for y-axis text
  theme(axis.title.y = element_text(size = 30, face = "bold")) + # change font size for y-axis title
  theme(axis.title.x = element_text(size = 30, face = "bold")) + # change font size for x-axis title
  theme(axis.text.x = element_text(angle = 0, hjust = 1, size = 28)) + # change font size for x-axis text
  theme(plot.title = element_text(size = 40, face = "bold")) + # customize plot's title, legend, font
  geom_col(fill = "steelblue") + # fill the bars that represent the values with blue
  coord_flip() # flip x and y axis
#####-----#####
#####
#####-----#####
#####
# plot cgMLST distribution
# calculate the relative frequencies for serovar and cgMLST
# drop the serovars and cgMLSTs with NAs, group by serovar and cgMLST and then calculate the frequencies
cgmlst_data <- d5 %>%
  drop_na(serovar, cgmlst) %>% # drop NAs for serovar and cgmlst columns
  group_by(serovar, cgmlst) %>% # group data based on serovar and cgmlst columns
  summarise(n = n()) %>% # count the number of observations
  mutate(prop_cgmlst = n/sum(n)*100) %>% # calculate the proportions
  filter(serovar != "Other serovars") # filter out data for Other serovars

## `summarise()` has grouped output by 'serovar'. You can override using the `.groups` argument.

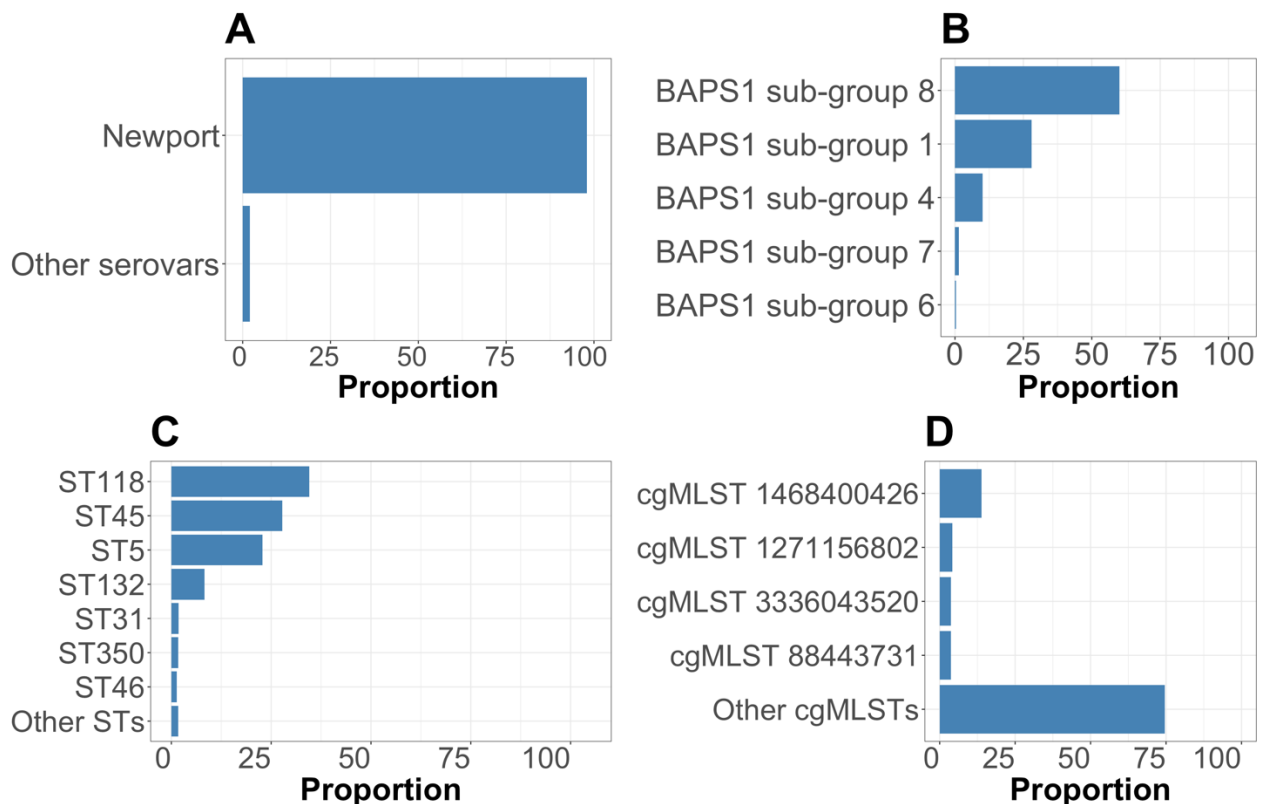
# re-order cgmlst column
cgmlst_data$cgmlst <- factor(cgmlst_data$cgmlst, levels=c("Other cgMLSTs", "cgMLST 88443731", "cgMLST 3336043520", "cgMLST 1271156802", "cgMLST 1468400426"))
# plot the data
cgmlst_plot <- ggplot(cgmlst_data, aes(x = cgmlst, y = prop_cgmlst)) + # show cgMLSTs on x-axis and proportion on y-axis
  xlab("") + ylab("Proportion") + ylim(0, 100) + # set labels for axis and limit for y-axis
  ggtitle("D") + # add title for the plot
  theme_bw() + # set plot background
  theme(legend.position = "none") + # remove legend
  theme(axis.text.y = element_text(size = 28)) + # set the font size for y-axis text
  theme(axis.title.x = element_text(size = 30, face = "bold")) + # set the font

```

```

nt size for x-axis title
  theme(axis.text.x = element_text(angle = 0, hjust = 1, size = 28)) + # set
the font size for x-axis text, orientations, and angle
  theme(plot.title = element_text(size = 40, face = "bold")) + # customize p
lot's title, legend, font
  geom_col(fill = "steelblue") + # fill the bars that represent the values w
ith blue
  coord_flip() # flip x and y axis
#####-----#####
#####
#####-----#####
#####
# combine all plots in one to make Figure 2 based on 2 columns x 2 rows
figure_2 <- ggarrange(sero_plot, baps_plot, st_plot, cgmlst_plot,
                      nrow = 2, ncol = 2, widths = c(10, 10, 10, 10),
                      heights = c(1, 1, 1, 1))
figure_2

```



Supplementary Figure S1. Scatter plot showing the distribution of STs and cgMLSTs. Note - only genomes classified as S. Newport by SISTR within ProkEvo are included in this analysis.

```

# bring the data containing the ST and cgmlst_ST information as numeric values
# assigning d2 to d2b
d2b <- d2

```

```

# plot the ST distribution using a scatter plot
# select Newport serovars only, drop the STs with NAs, group by ST and then calculate the distribution
st_scatter <- d2b %>%
  filter(serovar == "Newport") %>% # filter for Newport data only
  select(ST) %>% # select the ST column
  mutate(ST = as.numeric(ST)) %>% # transform the column to numeric

ic
  drop_na(ST) %>% # remove NA
  group_by(ST) %>% # group by ST
  summarise(n = n()) %>% # count observations
  mutate(prop = n/sum(n)*100) %>% # create a column with proportions

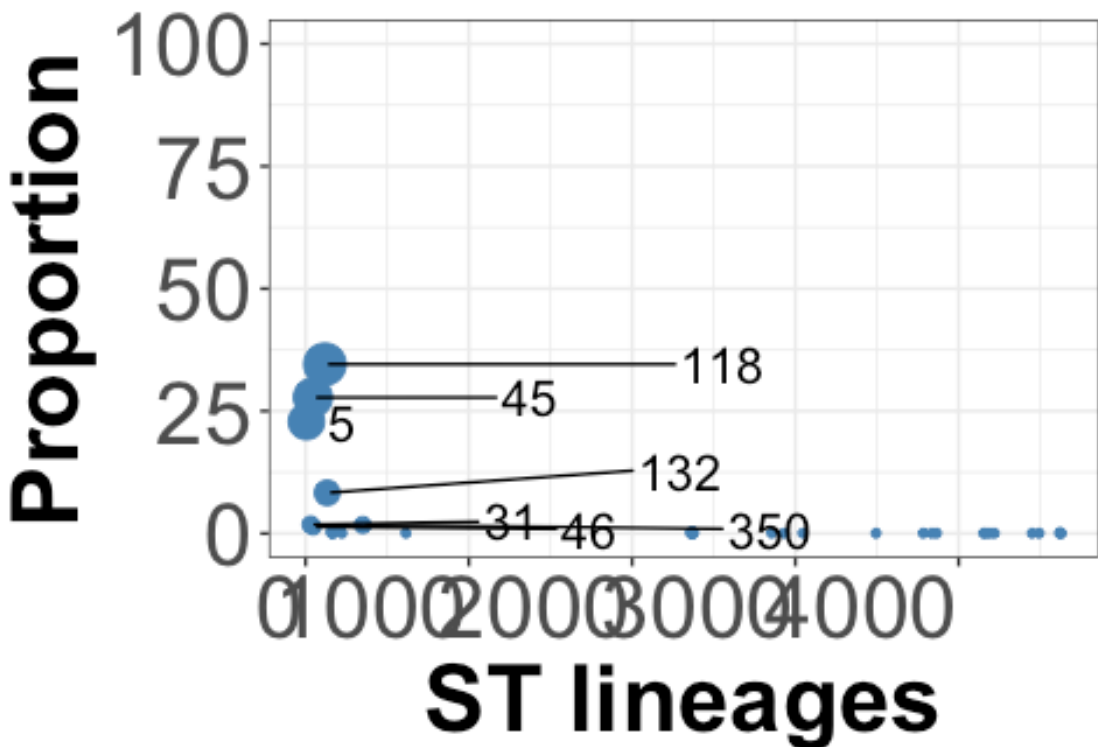
  arrange(desc(prop)) # arrange data in descending order
# check the first six observation of st_scatter
head(st_scatter)

## # A tibble: 6 x 3
##       ST      n prop
##   <dbl> <int> <dbl>
## 1   118   800 34.6
## 2    45   643 27.8
## 3     5   529 22.9
## 4   132   192  8.29
## 5    31    42  1.81
## 6   350    40  1.73

# plot
st_plot <- ggplot(st_scatter, aes(x = ST, y = prop, labels = ST)) + # show STs on x-axis and proportion on y-axis
  xlab("ST lineages") + ylab("Proportion") + ylim(0, 100) + # set labels for axis and limit for y-axis
  ggtitle("A") + # add title for the plot
  theme_bw() + # set plot background
  theme(legend.position = "none") + # remove legends
  theme(axis.text.y = element_text(size = 28)) + # change y-axis text font size
  theme(axis.title.y = element_text(size = 30, face = "bold")) + # change y-axis title font size and face
  theme(axis.title.x = element_text(size = 30, face = "bold")) + # change x-axis title font size and face
  theme(axis.text.x = element_text(angle = 0, hjust = 1, size = 28)) + # change x-axis text font size, angle and orientation
  theme(plot.title = element_text(size = 40, face = "bold")) + # customize figure's title, legend, font
  geom_point(aes(size = prop), color = "steelblue") + # the points that represent the values are blue with size based on the proportion
  geom_text_repel(data=subset(st_scatter, prop > 1), aes(label = ST, size = 30), hjust = -6) # add text/proportion to the plot
st_plot

```

A



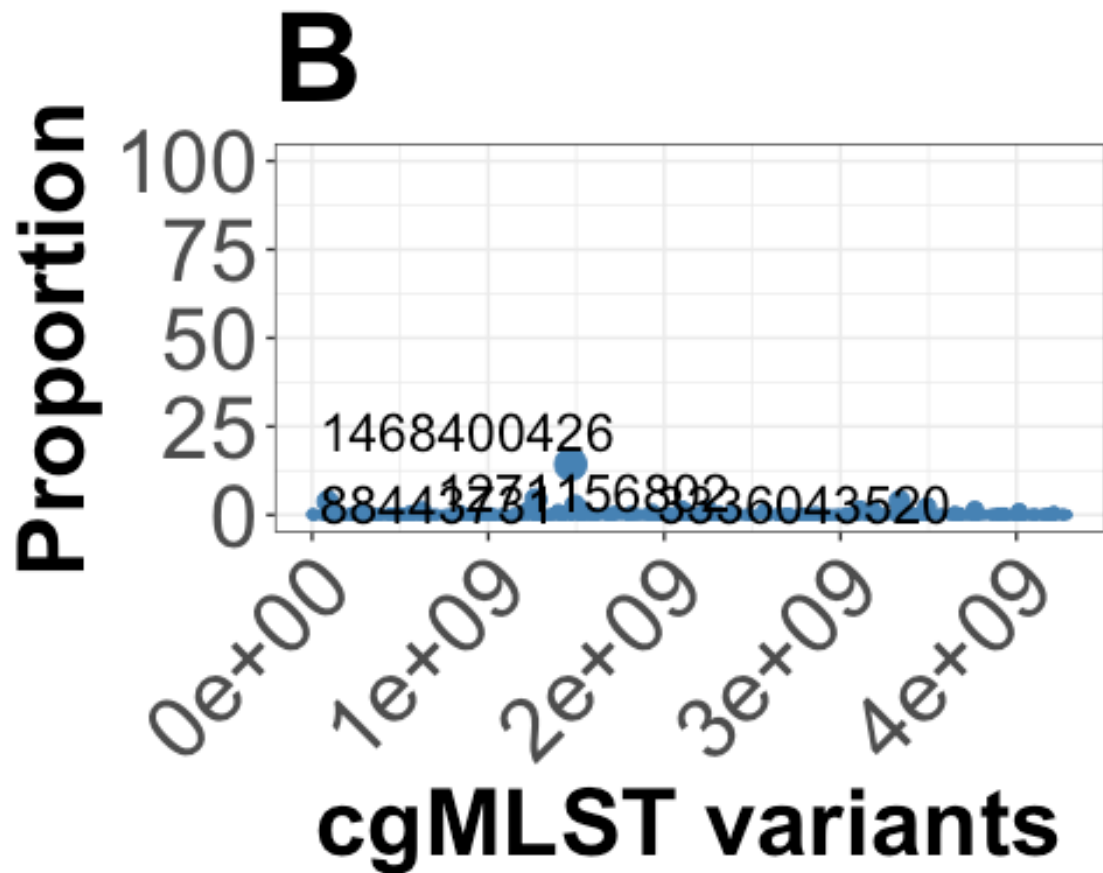
```
#####-----#####
#####
#####-----#####
#####
# plot the cgMLST distribution using a scatter plot
# select Newport serovars only, drop the cgMLSTs with NAs, group by cgMLST and then calculate the distribution
cgmlst_scatter <- d2b %>%
  filter(serovar == "Newport") %>% # select Newport genomes only
  select(cgmlst_ST) %>% # select the cgmlst_ST column
  mutate(cgmlst_ST = as.numeric(cgmlst_ST)) %>% # change column to numeric
  drop_na(cgmlst_ST) %>% # drop NAs
  group_by(cgmlst_ST) %>% # group by cgmlst_ST
  summarise(n = n()) %>% # count observations
  mutate(prop = n/sum(n)*100) %>% # calculate proportions
  arrange(desc(prop)) # arrange in descending order
# check the first six observation of st_scatter
head(cgmlst_scatter)

## # A tibble: 6 x 3
##   cgmlst_ST      n  prop
##   <dbl> <int> <dbl>
```

```
## 1 1468400426    321 14.3
## 2 1271156802     97  4.32
## 3 3336043520     86  3.83
## 4   88443731     85  3.78
## 5 1492716119     64  2.85
## 6 3491314984     50  2.23
```

```
# plot
```

```
cgmlst_plot <- ggplot(cgmlst_scatter, aes(x = cgmlst_ST, y = prop, labels = c
gmlst_ST)) + # show cgMLSTs on x-axis and proportion on y-axis
  xlab("cgMLST variants") + ylab("Proportion") + ylim(0, 100) + # set labels
for axis and limit for y-axis
  ggtitle("B") + # add title for the plot
  theme_bw() + # set plot background
  theme(legend.position = "none") + # remove legends
  theme(axis.text.y = element_text(size = 28)) + # change y-axis text font si
ze
  theme(axis.title.y = element_text(size = 30, face = "bold")) + # change y-a
xis title font size and face
  theme(axis.title.x = element_text(size = 30, face = "bold")) + # change x-a
xis title font size and face
  theme(axis.text.x = element_text(angle = 45, hjust = 1, size = 28)) + # cha
nge x-axis text font size, angle, and orientation
  theme(plot.title = element_text(size = 40, face = "bold")) + # customize f
igure's title, legend, font
  geom_point(aes(size = prop), color = "steelblue") + # the points that rep
resent the values are blue with size based on the proportion
  geom_text_repel(data=subset(cgmlst_scatter, prop > 3), aes(label = cgmlst_S
T, size = 30), hjust = 2) # add text/proportion to the plot
cgmlst_plot
```



```
#####-----#####
#####
#####-----#####
#####
# combine all plots in one to make Figure 3
sup_fig_s1 <- ggarrange(st_plot, cgmlst_plot,
  nrow = 1, ncol = 2, widths = c(10, 10),
  heights = c(1, 1)) # combine plots together in one row and tw
o columns
sup_fig_s1
```

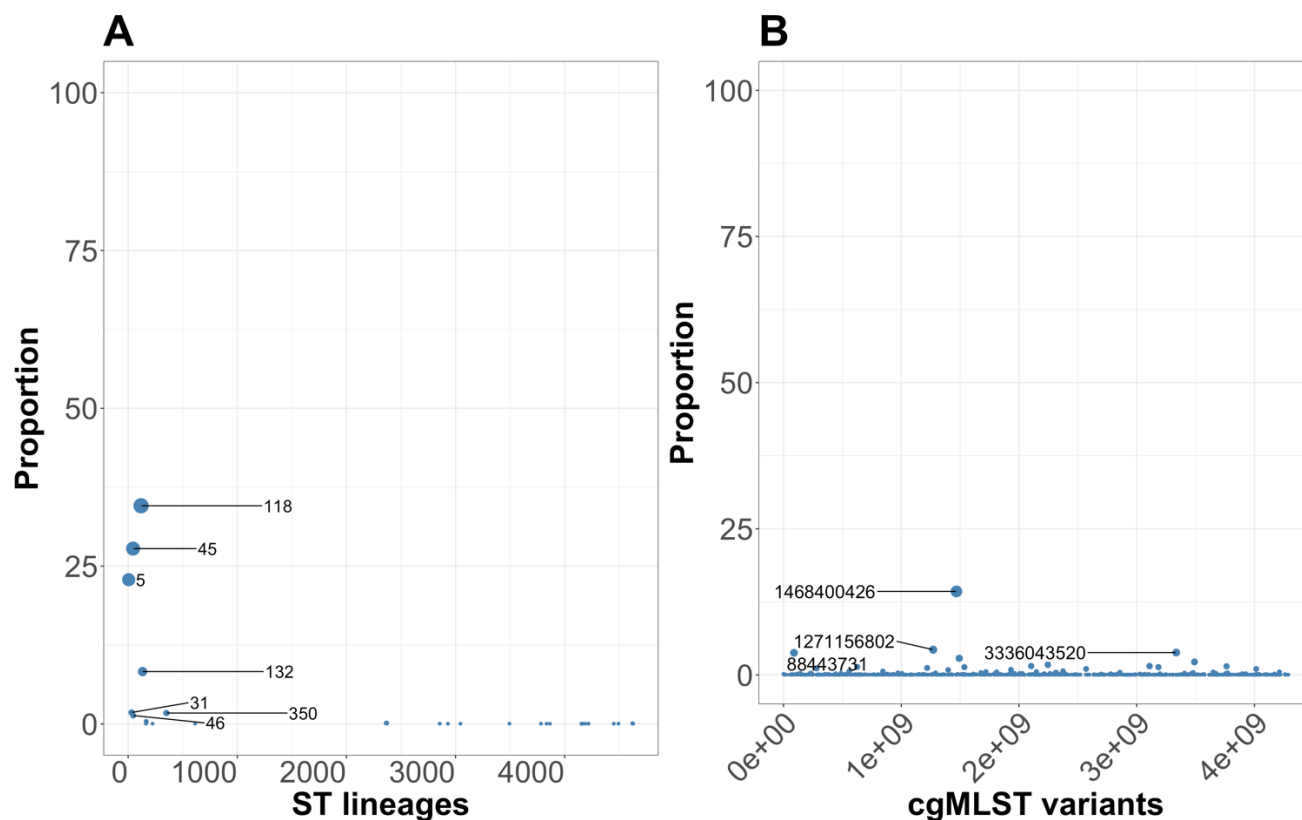


Figure 3. Distribution of STs based on BAPS level 1 sub-groups. Note - only genomes classified as S. Newport by SISTR within ProkEvo are included in this analysis.

```
# bring the data containing the ST information as numeric values
# assigning d2 to d2b
d2b <- d2
# plot the ST distribution using a scatter plot
# select Newport serovars only, drop the STs and BAPS1 sub-groups with NAs, group by ST and BAPS1 sub-groups and then calculate the distribution
baps <- d2b %>%
  filter(serovar == "Newport") %>% # filter Newport serovars
  select(baps_1, ST) %>% # select baps_1 and ST columns
  mutate(ST = as.numeric(ST)) %>% # change ST column to numeric
  drop_na(baps_1, ST) %>% # drop NAs
  group_by(baps_1, ST) %>% # group by baps_1 and ST
  summarise(n = n()) %>% # count observations
  mutate(prop = n/sum(n)*100) # calculate proportions

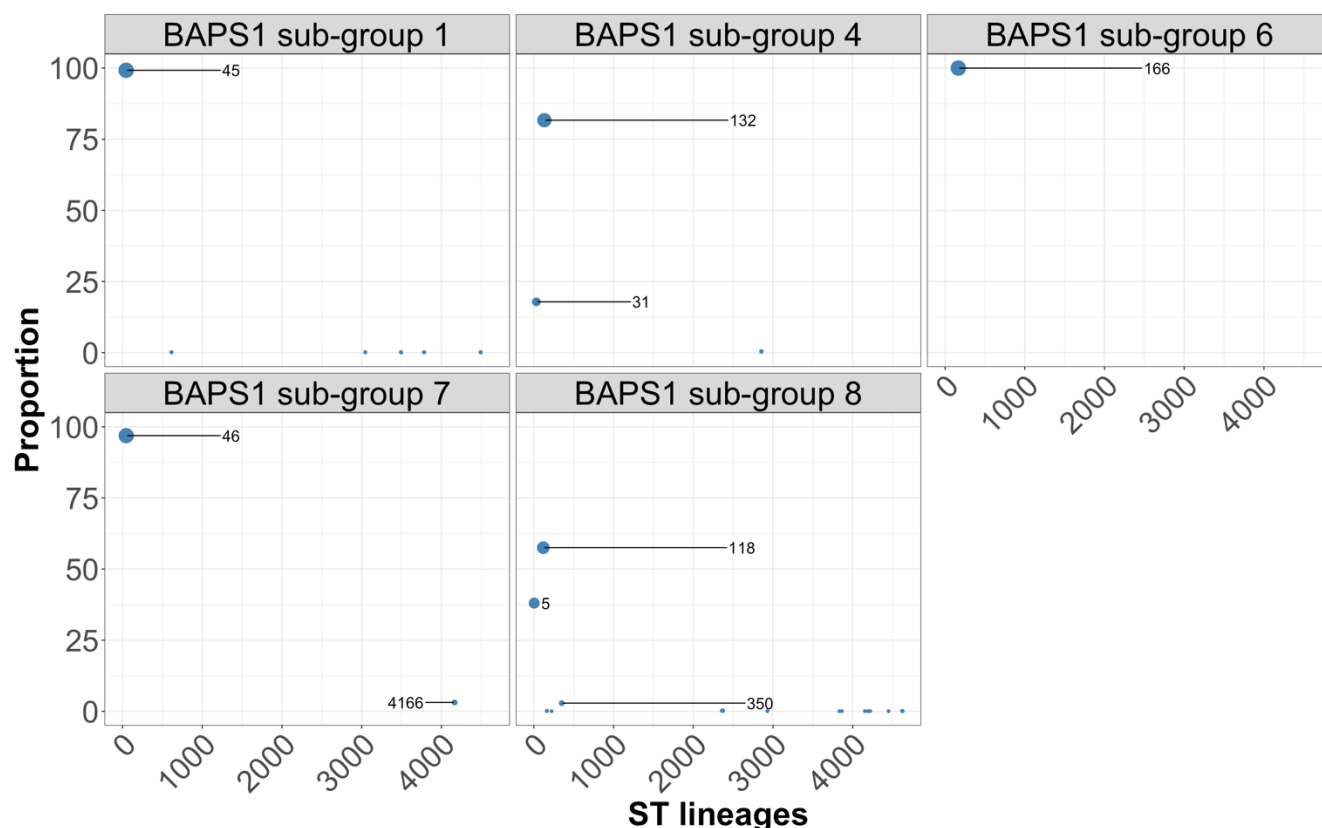
## `summarise()` has grouped output by 'baps_1'. You can override using the `
.groups` argument.

# check the first six observation of baps
head(baps)

## # A tibble: 6 x 4
## # Groups:   baps_1 [1]
```

```
##   baps_1          ST      n   prop
##   <chr>          <dbl> <int> <dbl>
## 1 BAPS1 sub-group 1    45   643 99.2
## 2 BAPS1 sub-group 1   614     1 0.154
## 3 BAPS1 sub-group 1  3045     1 0.154
## 4 BAPS1 sub-group 1  3494     1 0.154
## 5 BAPS1 sub-group 1  3783     1 0.154
## 6 BAPS1 sub-group 1  4493     1 0.154

# plot
figure_3 <- ggplot(baps, aes(x = ST, y = prop, labels = ST)) + # show STs on
x-axis and proportion on y-axis
  xlab("ST lineages") + ylab("Proportion") + ylim(0, 100) + # set labels for
axis and limit for y-axis
  theme_bw() + # set plot background
  theme(legend.position = "none") + # remove legends
  theme(axis.text.y = element_text(size = 28)) + # change y-axis text font si
ze
  theme(axis.title.y = element_text(size = 30, face = "bold")) + # change y-a
xis title font size and face
  theme(axis.title.x = element_text(size = 30, face = "bold")) + # change x-a
xis title font size and face
  theme(axis.text.x = element_text(angle = 45, hjust = 1, size = 28)) + # cha
nge x-axis text font size, angle, and orientation
  theme(strip.text.x = element_text(size = 30, colour = "black", angle = 0))
+ # customize figure's title, legend, font
  geom_point(aes(size = prop), color = "steelblue") + # the points that rep
resent the values are blue with size based on the proportion
  geom_text_repel(data=subset(baps, prop > 1), aes(label = ST, size = 70), hj
ust = -10) + # add text/proportion to the plot
  facet_wrap(~ baps_1) # generate multi-plots for BAPS1 sub-groups
figure_3
```

Supplementary Figure S2. Distribution of cgMLSTs based on the major ST lineages presented in Figures 1 and 2. Note - only genomes classified as *S. Newport* by SISTR within ProkEvo are included in this analysis.

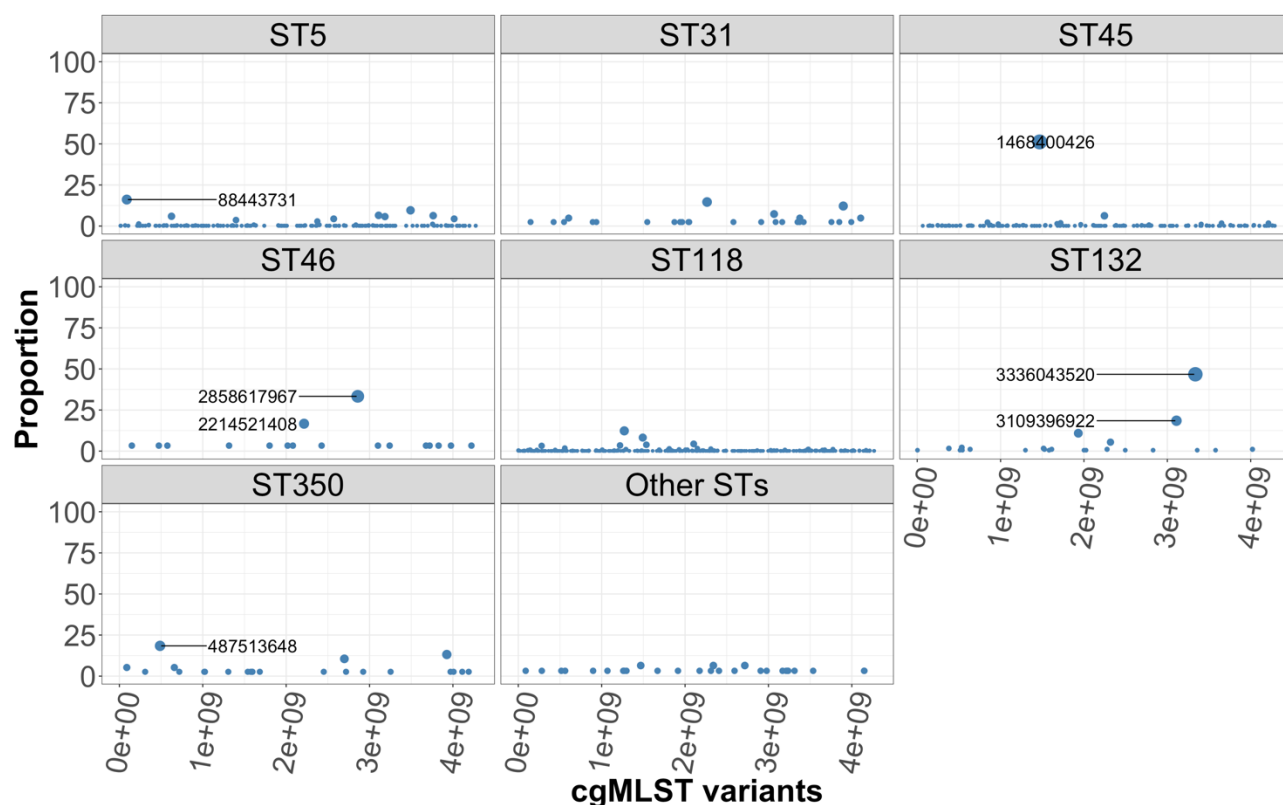
```
# bring the data containing the cgMLST information as numeric values
# assigning d2 to d2b
d2b <- d2
# plot the ST distribution using a scatter plot
# select Newport serovars only, drop the STs and cgMLSTs with NAs, group by ST and cgMLST and then calculate the distribution
cgmlst <- d2b %>%
  filter(serovar == "Newport") %>% # filter for Newport genomes only
  select(st, cgmlst_ST) %>% # select st and cgmlst_ST columns
  drop_na(st, cgmlst_ST) %>% # drop NAs
  group_by(st, cgmlst_ST) %>% # group by st and cgmlst_ST
  summarise(n = n()) %>% # count observations
  mutate(prop = n/sum(n)*100) # calculate proportions

## `summarise()` has grouped output by 'st'. You can override using the `.groups` argument.

# check the first six observation of cgmlst
head(cgmlst)
```

```
## # A tibble: 6 x 4
## # Groups:   st [1]
##   st          cgmlst_ST      n prop
##   <chr>          <dbl> <int> <dbl>
## 1 Other STs      88443731      1  3.23
## 2 Other STs     281971874      1  3.23
## 3 Other STs     516287174      1  3.23
## 4 Other STs     561885424      1  3.23
## 5 Other STs     897193995      1  3.23
## 6 Other STs    1069299601      1  3.23

# re-order ST
cgmlst$st <- factor(cgmlst$st, levels=c("ST5", "ST31", "ST45", "ST46", "ST118",
", "ST132", "ST350", "Other STs"))
# plot
sup_fig_2 <- ggplot(cgmlst, aes(x = cgmlst_ST, y = prop, labels = cgmlst_ST))
+ # show cgMLSTs on x-axis and proportion on y-axis
  xlab("cgMLST variants") + ylab("Proportion") + ylim(0, 100) + # set labels
for axis and limit for y-axis
  theme_bw() + # set plot background
  theme(legend.position = "none") + # remove legend
  theme(axis.text.y = element_text(size = 28)) + # change y-axis text font si
ze
  theme(axis.title.y = element_text(size = 30, face = "bold")) + # change y-a
xis title font size and face
  theme(axis.title.x = element_text(size = 30, face = "bold")) + # change x-a
xis title font size and face
  theme(axis.text.x = element_text(angle = 80, hjust = 1, size = 28)) + # cha
nge x-axis text font size, angle, and orientation
  theme(strip.text.x = element_text(size = 30, colour = "black", angle = 0))
+ # customize figure's title, legend, font
  geom_point(aes(size = prop), color = "steelblue") + # the points that rep
resent the values are blue with size based on the proportion
  geom_text_repel(data=subset(cgmlst, prop > 15), aes(label = cgmlst_ST, size
= 50), hjust = -25) + # add text/proportion to the plot
  facet_wrap(~ st) # generate multi-plots for STs
sup_fig_2
```



Supplementary Figure S3. Simpson's D diversity analysis of ST lineages using cgMLST and BAPS levels 1-6. Note - only genomes classified as *S. Newport* by SISTR within ProkEvo are included in this analysis.

```
# enter BAPS1-6 data
baps <- read_csv('~/Documents/jove_paper/data/fastbaps_partition_baps_prior_1
6.csv')

##
## — Column specification —————
## cols(
##   Isolates = col_character(),
##   `Level 1` = col_double(),
##   `Level 2` = col_double(),
##   `Level 3` = col_double(),
##   `Level 4` = col_double(),
##   `Level 5` = col_double(),
##   `Level 6` = col_double()
## )

# changing column names
colnames(baps)[1:7] <- c("id", "BAPS1", "BAPS2", "BAPS3", "BAPS4", "BAPS5", "
BAPS6")
# assign baps data to d1
data1 <- baps
```

```

# transform BAPS sub-groups to factor (categorical data)
data1 <- data1 %>% mutate(BAPS1 = as.factor(BAPS1))
data1 <- data1 %>% mutate(BAPS2 = as.factor(BAPS2))
data1 <- data1 %>% mutate(BAPS3 = as.factor(BAPS3))
data1 <- data1 %>% mutate(BAPS4 = as.factor(BAPS4))
data1 <- data1 %>% mutate(BAPS5 = as.factor(BAPS5))
data1 <- data1 %>% mutate(BAPS6 = as.factor(BAPS6))
#####-----#####
#####
#####-----#####
#####
# enter the ST data
mlst <- read_csv('~/Documents/jove_paper/data/salmonellast_output.csv')

##
## — Column specification —————
##
## cols(
##   FILE = col_character(),
##   SCHEME = col_character(),
##   ST = col_character(),
##   aroC = col_character(),
##   dnaN = col_character(),
##   hemD = col_character(),
##   hisD = col_character(),
##   purE = col_character(),
##   sucA = col_character(),
##   thrA = col_character()
## )

# generate the id column
mlst$id2 <- sapply(strsplit(as.character(mlst$FILE), '_'), "[", 1)
# select columns of interest
data2 <- mlst %>%
  select(id2, ST) %>% # select columns of interest
  mutate_all(na_if, "-") %>% # transform any - to NA
  mutate_all(na_if, "?") %>% # transform any ? to NA
  rename(id = id2) # rename id2 column to id
# categorize ST data to aggregate minor STs and keep only major ST Lineages separately
data2$st <- ifelse(data2$ST == 5, "ST5", # group major STs separately, and minor STs as Others
  ifelse(data2$ST == 31, "ST31",
    ifelse(data2$ST == 45, "ST45",
      ifelse(data2$ST == 46, "ST46",
        ifelse(data2$ST == 118, "ST118",
          ifelse(data2$ST == 132, "ST132",
            ifelse(data2$ST == 350, "ST350", "Other STs"))))))))
# filter out the numerical ST column
data2 <- data2 %>% select(-ST)

```

```
#####-----#####
#####
#####-----#####
#####
# enter SISTR results
sistr <- read_csv('~/Documents/jove_paper/data/sistr_output.csv')

##
## — Column specification —————
## cols(
##   cgmlst_ST = col_double(),
##   cgmlst_distance = col_double(),
##   cgmlst_genome_match = col_character(),
##   cgmlst_matching_alleles = col_double(),
##   cgmlst_subspecies = col_character(),
##   fasta_filepath = col_character(),
##   genome = col_character(),
##   h1 = col_character(),
##   h2 = col_character(),
##   o_antigen = col_character(),
##   qc_messages = col_character(),
##   qc_status = col_character(),
##   serogroup = col_character(),
##   serovar = col_character(),
##   serovar_antigen = col_character(),
##   serovar_cgmlst = col_character()
## )

# generate the id column
sistr$id <- sapply(strsplit(as.character(sistr$genome), '_'), "[", 1)
# select id and cgmlst_ST columns
data3 <- sistr %>%
  select(id, serovar, cgmlst_ST) %>% # select columns of interest
  mutate_all(na_if, "-") %>% # transform any - to NA
  mutate_all(na_if, "?") # transform any ? to NA
# group serovars as Newport or others
data3$serovar <- ifelse(data3$serovar == "Newport", "Newport",
  "Other serovars") # group genomes to either Newport or other serovars
#####-----#####
#####
#####-----#####
#####
# merge datasets
data4 <- left_join(data1, data2, on = "id") # join datasets based on id

## Joining, by = "id"

data5 <- left_join(data3, data4, on = "id") # join datasets based on id
```

```
## Joining, by = "id"

# filter data for Newport only
data6 <- data5 %>% filter(serovar == "Newport")
# check for missing values
skim(data6)
```

Data summary

| | |
|-------------------|-------|
| Name | data6 |
| Number of rows | 2317 |
| Number of columns | 10 |

Column type frequency:

| | |
|-----------|---|
| character | 3 |
| factor | 6 |
| numeric | 1 |

| | |
|-----------------|------|
| Group variables | None |
|-----------------|------|

Variable type: character

| skim_variable | n_missing | complete_rate | min | max | empty | n_unique | whitespace |
|---------------|-----------|---------------|-----|-----|-------|----------|------------|
| id | 0 | 1 | 9 | 11 | 0 | 2317 | 0 |
| serovar | 0 | 1 | 7 | 7 | 0 | 1 | 0 |
| st | 3 | 1 | 3 | 9 | 0 | 8 | 0 |

Variable type: factor

| skim_variable | n_missing | complete_rate | ordered | n_unique | top_counts |
|---------------|-----------|---------------|---------|----------|-------------------------------------|
| BAPS1 | 0 | 1 | FALSE | 5 | 8: 1393, 1: 648, 4: 235, 7: 32 |
| BAPS2 | 0 | 1 | FALSE | 19 | 24: 532, 1: 486, 21: 300, 7: 189 |
| BAPS3 | 0 | 1 | FALSE | 62 | 1: 483, 57: 263, 56: 187, 45: 155 |
| BAPS4 | 0 | 1 | FALSE | 121 | 1: 483, 98: 149, 69: 143, 97: 114 |
| BAPS5 | 0 | 1 | FALSE | 212 | 1: 483, 103: 143, 155: 131, 104: 93 |
| BAPS6 | 0 | 1 | FALSE | 312 | 1: 483, 141: 143, 216: 131, 142: 93 |

Variable type: numeric

| skim_variable | n_missing | complete | mean | sd | p0 | p25 | p50 | p75 | p100 | hist |
|---------------|-----------|----------|----------------|----------------|-----------------|----------------|----------------|----------------|----------------|---|
| cgmlst_ST | 70 | 0.97 | 19976 02103 | 11492 04978 | 183 768 5 | 12711 56802 | 16127 07568 | 31089 47600 | 42880 28711 |  |

```
# group ST missing values as "Other STs"
data6 <- data6 %>% mutate(st = replace_na(st, "Other STs"))
# check the distribution of serovars to make sure there is only Newport
table(data6$serovar)

##
## Newport
## 2317

#####-----#####
#####
#####-----#####
#####
# cgMLST diversity
# drop the STs and cgMLSTs with NAs, group by ST and cgMLST and then calculate Simpson's index
cgmlst_div <- data6 %>%
  drop_na(st, cgmlst_ST) %>% # drop NAs
  mutate(cgmlst_ST = as.factor(cgmlst_ST)) %>% # transform column
to categorical
  select(st, cgmlst_ST) %>% # select columns of interest
  group_by(st, cgmlst_ST) %>% # group by st and cgmlst_ST
  summarise(n = n()) %>% # count observations
  mutate(simpson = diversity(n, "simpson")) %>% # calculate the index of diversity
  group_by(st) %>% # group by st
  summarise(simpson = mean(simpson)) %>% # get the mean of the simpson index
  melt(id.vars=c("st"), measure.vars="simpson",
        variable.name="index", value.name="value") %>% # convert into long format
  mutate(strat = "cgMLST") # create a strat column

## `summarise()` has grouped output by 'st'. You can override using the `.groups` argument.

#####-----#####
#####
#####-----#####
#####
# BAPS based diversity (calculated across BAPS Levels 1 through 6)
```

```

# BAPS Level 1
# drop the STs and BAPS1 with NAs, group by ST and BAPS1 and then calculate S
impson's index
baps1 <- data6 %>%
  select(st, BAPS1) %>% # select columns
  drop_na(st, BAPS1) %>% # drop NAs
  group_by(st, BAPS1) %>% # group by columns
  summarise(n = n()) %>% # count observations
  mutate(simpson = diversity(n, "simpson")) %>% # calculate diver
sity

  group_by(st) %>% # group by column
  summarise(simpson = mean(simpson)) %>% # calculate the mean of
the index

  melt(id.vars=c("st"), measure.vars="simpson",
        variable.name="index", value.name="value") %>% # covert i
nto long format
  mutate(strat = "BAPS1") # create a strat column

## `summarise()` has grouped output by 'st'. You can override using the `.gro
ups` argument.

# BAPS Level 2
# drop the STs and BAPS2 with NAs, group by ST and BAPS2 and then calculate S
impson's index
baps2 <- data6 %>%
  select(st, BAPS2) %>% # select columns
  drop_na(st, BAPS2) %>% # drop NAs
  group_by(st, BAPS2) %>% # group by columns
  summarise(n = n()) %>% # count observations
  mutate(simpson = diversity(n, "simpson")) %>% # calculate diver
sity

  group_by(st) %>% # group by column
  summarise(simpson = mean(simpson)) %>% # calculate the mean of
the index

  melt(id.vars=c("st"), measure.vars="simpson",
        variable.name="index", value.name="value") %>% # covert i
nto long format
  mutate(strat = "BAPS2") # create a strat column

## `summarise()` has grouped output by 'st'. You can override using the `.gro
ups` argument.

# BAPS Level 3
# drop the STs and BAPS3 with NAs, group by ST and BAPS3 and then calculate S
impson's index
baps3 <- data6 %>%
  select(st, BAPS3) %>% # select columns
  drop_na(st, BAPS3) %>% # drop NAs
  group_by(st, BAPS3) %>% # group by columns
  summarise(n = n()) %>% # count observations
  mutate(simpson = diversity(n, "simpson")) %>% # calculate diver

```



```

sity
      group_by(st) %>% # group by column
      summarise(simpson = mean(simpson)) %>% # calculate the mean of
the index
      melt(id.vars=c("st"), measure.vars="simpson",
           variable.name="index", value.name="value") %>% # covert i
nto Long format
      mutate(strat = "BAPS3") # create a strat column

## `summarise()` has grouped output by 'st'. You can override using the `.gro
ups` argument.

# BAPS Level 4
# drop the STs and BAPS4 with NAs, group by ST and BAPS4 and then calculate S
impson's index
baps4 <- data6 %>%
      select(st, BAPS4) %>% # select columns
      drop_na(st, BAPS4) %>% # drop NAs
      group_by(st, BAPS4) %>% # group by columns
      summarise(n = n()) %>% # count observations
      mutate(simpson = diversity(n, "simpson")) %>% # calculate diver
sity
      group_by(st) %>% # group by column
      summarise(simpson = mean(simpson)) %>% # calculate the mean of
the index
      melt(id.vars=c("st"), measure.vars="simpson",
           variable.name="index", value.name="value") %>% # covert i
nto Long format
      mutate(strat = "BAPS4") # create a strat column

## `summarise()` has grouped output by 'st'. You can override using the `.gro
ups` argument.

# BAPS Level 5
# drop the STs and BAPS5 with NAs, group by ST and BAPS5 and then calculate S
impson's index
baps5 <- data6 %>%
      select(st, BAPS5) %>% # select columns
      drop_na(st, BAPS5) %>% # drop NAs
      group_by(st, BAPS5) %>% # group by columns
      summarise(n = n()) %>% # count observations
      mutate(simpson = diversity(n, "simpson")) %>% # calculate diver
sity
      group_by(st) %>% # group by column
      summarise(simpson = mean(simpson)) %>% # calculate the mean of
the index
      melt(id.vars=c("st"), measure.vars="simpson",
           variable.name="index", value.name="value") %>% # covert i
nto Long format
      mutate(strat = "BAPS5") # create a strat column

```

`summarise()` has grouped output by 'st'. You can override using the `.groups` argument.

BAPS Level 6

drop the STs and BAPS6 with NAs, group by ST and BAPS6 and then calculate Simpson's index

```
baps6 <- data6 %>%  
  select(st, BAPS6) %>% # select columns  
  drop_na(st, BAPS6) %>% # drop NAs  
  group_by(st, BAPS6) %>% # group by columns  
  summarise(n = n()) %>% # count observations  
  mutate(simpson = diversity(n, "simpson")) %>% # calculate diver-
```

sity

```
  group_by(st) %>% # group by column  
  summarise(simpson = mean(simpson)) %>% # calculate the mean of  
the index
```

```
  melt(id.vars=c("st"), measure.vars="simpson",  
        variable.name="index", value.name="value") %>% # covert i  
nto long format
```

```
  mutate(strat = "BAPS6") # create a strat column
```

`summarise()` has grouped output by 'st'. You can override using the `.groups` argument.

```
#####-----#####  
#####
```

```
#####-----#####  
#####
```

Concatenate all datasets

```
data7 <- rbind(cgmlst_div, baps1, baps2, baps3, baps4, baps5, baps6)
```

order the ST column

```
data7$st <- factor(data7$st, levels=c("ST5", "ST31", "ST45", "ST46", "ST118",  
  "ST132", "ST350", "Other STs"))
```

```
#####-----#####  
#####
```

```
#####-----#####  
#####
```

plot Simpson's diversity analysis

order the strat column

```
data7$strat <- factor(data7$strat, levels=c("cgMLST", "BAPS6", "BAPS5", "BAPS  
4", "BAPS3", "BAPS2", "BAPS1"))
```

```
sup_fig_3 <- ggplot(data7, aes(x = strat, y = value)) + # show strata on x-a  
xis and index values on y-axis
```

```
  xlab("") + ylab("Index value") + ylim(0,1) + # set labels for axis and lim  
it for y-axis
```

```
  theme_bw() + # set plot background
```

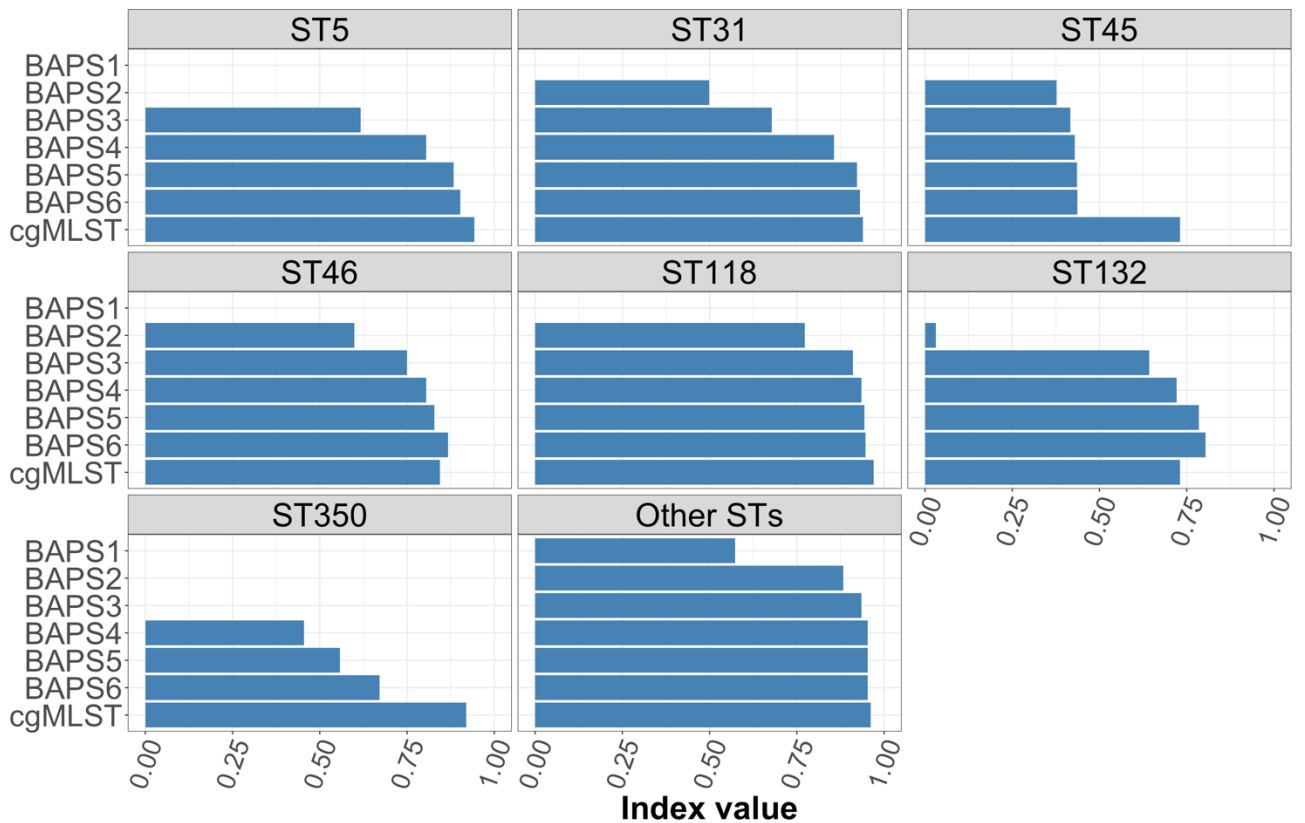
```
  theme(axis.text.y = element_text(size = 28)) + # change y-axis text font si  
ze
```

```
  theme(axis.title.y = element_text(size = 30, face = "bold")) + # change y-a  
xis title font size and face
```

```

  theme(axis.title.x = element_text(size = 30, face = "bold")) + # change x-axis title font size and face
  theme(axis.text.x = element_text(angle = 70, hjust = 1, size = 24)) + # change x-axis text font size, angle, and orientation
  theme(strip.text.x = element_text(size = 30, colour = "black", angle = 0))
+ # customize figure's title, legend, font
  geom_col(fill = "steelblue") + # fill the bars that represent the values with blue
  facet_wrap(~st) + # generate multi-plots for STs
  coord_flip() # flip x and y axis
sup_fig_3

```



Supplementary Figure S4. BAPS levels 1-6 distribution of sub-groups across ST lineages. Note - only genomes classified as S. Newport by SISTR within ProkEvo are included in this analysis.

```

# enter BAPS1-6 data
baps <- read_csv('~/Documents/jove_paper/data/fastbaps_partition_baps_prior_16.csv')

##
## — Column specification —————
## cols(
##   Isolates = col_character(),

```

```

## `Level 1` = col_double(),
## `Level 2` = col_double(),
## `Level 3` = col_double(),
## `Level 4` = col_double(),
## `Level 5` = col_double(),
## `Level 6` = col_double()
## )

# changing column names
colnames(baps)[1:7] <- c("id", "BAPS1", "BAPS2", "BAPS3", "BAPS4", "BAPS5", "
BAPS6")
# assign baps data to d1
data1 <- baps
# transform BAPS sub-groups to factor (categorical data)
data1 <- data1 %>% mutate(BAPS1 = as.factor(BAPS1))
data1 <- data1 %>% mutate(BAPS2 = as.factor(BAPS2))
data1 <- data1 %>% mutate(BAPS3 = as.factor(BAPS3))
data1 <- data1 %>% mutate(BAPS4 = as.factor(BAPS4))
data1 <- data1 %>% mutate(BAPS5 = as.factor(BAPS5))
data1 <- data1 %>% mutate(BAPS6 = as.factor(BAPS6))
#####-----#####
#####
#####-----#####
#####
# enter the ST data
mlst <- read_csv('~/.Documents/jove_paper/data/salmonellast_output.csv')

##
## — Column specification —————
##
## cols(
##   FILE = col_character(),
##   SCHEME = col_character(),
##   ST = col_character(),
##   aroC = col_character(),
##   dnaN = col_character(),
##   hemD = col_character(),
##   hisD = col_character(),
##   purE = col_character(),
##   sucA = col_character(),
##   thrA = col_character()
## )

# generate the id column
mlst$id2 <- sapply(strsplit(as.character(mlst$FILE), '_'), "[", 1)
# select columns of interest and rename -/? to NAs
data2 <- mlst %>%
  select(id2, ST) %>% # select columns
  mutate_all(na_if, "-") %>% # transform any - to NAs
  mutate_all(na_if, "?") %>% # transform any ? to NAs

```

```

        rename(id = id2) # rename column
# categorize ST data to aggregate minor STs and keep only major ST Lineages s
eparately
data2$st <- ifelse(data2$ST == 5, "ST5", # aggregate minor STs as others
        ifelse(data2$ST == 31, "ST31",
        ifelse(data2$ST == 45, "ST45",
        ifelse(data2$ST == 46, "ST46",
        ifelse(data2$ST == 118, "ST118",
        ifelse(data2$ST == 132, "ST132",
",
        ifelse(data2$ST == 350, "ST350", "Other STs"))))))))
# filter out the numerical ST column
data2 <- data2 %>% select(-ST)
#####-----#####
#####
#####-----#####
#####
# enter SISTR results
sistr <- read_csv('~/.Documents/jove_paper/data/sistr_output.csv')

##
## — Column specification —————
##
## cols(
##   cgmlst_ST = col_double(),
##   cgmlst_distance = col_double(),
##   cgmlst_genome_match = col_character(),
##   cgmlst_matching_alleles = col_double(),
##   cgmlst_subspecies = col_character(),
##   fasta_filepath = col_character(),
##   genome = col_character(),
##   h1 = col_character(),
##   h2 = col_character(),
##   o_antigen = col_character(),
##   qc_messages = col_character(),
##   qc_status = col_character(),
##   serogroup = col_character(),
##   serovar = col_character(),
##   serovar_antigen = col_character(),
##   serovar_cgmlst = col_character()
## )

# generate the id column
sistr$id <- sapply(strsplit(as.character(sistr$genome), '_'), "[", 1)
# select id and cgmlst_ST columns and rename -/? to NAs
data3 <- sistr %>%
  select(id, serovar, cgmlst_ST) %>% # select columns
  mutate_all(na_if, "-") %>% # transform any - to NA
  mutate_all(na_if, "?") # transform any ? to NA
# group serovars as Newport or others

```

```

data3$serovar <- ifelse(data3$serovar == "Newport", "Newport",
                        "Other serovars")
#####-----#####
#####
#####-----#####
#####
# merge datasets
data4 <- left_join(data1, data2, on = "id") # join datasets on id
## Joining, by = "id"
data5 <- left_join(data3, data4, on = "id") # join datasets on id
## Joining, by = "id"
# filter data for Newport only
data6 <- data5 %>% filter(serovar == "Newport")
# check for missing values
skim(data6)

```

Data summary

| | |
|-------------------|-------|
| Name | data6 |
| Number of rows | 2317 |
| Number of columns | 10 |

Column type frequency:

| | |
|-----------|---|
| character | 3 |
| factor | 6 |
| numeric | 1 |

| | |
|-----------------|------|
| Group variables | None |
|-----------------|------|

Variable type: character


| skim_variable | n_missing | complete_rate | min | max | empty | n_unique | whitespace |
|---------------|-----------|---------------|-----|-----|-------|----------|------------|
| id | 0 | 1 | 9 | 11 | 0 | 2317 | 0 |
| serovar | 0 | 1 | 7 | 7 | 0 | 1 | 0 |
| st | 2 | 1 | 3 | 9 | 0 | 8 | 0 |

Variable type: factor

| skim_variable | n_missing | complete_rate | ordered | n_unique | top_counts |
|---------------|-----------|---------------|---------|----------|--------------------------------|
| BAPS1 | 0 | 1 | FALSE | 5 | 8: 1393, 1: 648, 4: 235, 7: 32 |

| | | | | | |
|-------|---|---|-------|-----|-------------------------------------|
| BAPS2 | 0 | 1 | FALSE | 19 | 24: 532, 1: 486, 21: 300, 7: 189 |
| BAPS3 | 0 | 1 | FALSE | 62 | 1: 483, 57: 263, 56: 187, 45: 155 |
| BAPS4 | 0 | 1 | FALSE | 121 | 1: 483, 98: 149, 69: 143, 97: 114 |
| BAPS5 | 0 | 1 | FALSE | 212 | 1: 483, 103: 143, 155: 131, 104: 93 |
| BAPS6 | 0 | 1 | FALSE | 312 | 1: 483, 141: 143, 216: 131, 142: 93 |

Variable type: numeric

| skim_v ariabl e | n_mi ssin g | compl ete_rat e | mean | sd | p0 | p25 | p50 | p75 | p100 | hist |
|-----------------------|-------------------|-----------------------|----------------|----------------|-----------------|----------------|----------------|----------------|----------------|---|
| cgmlst _ST | 70 | 0.97 | 19976 02103 | 11492 04978 | 183 768 5 | 12711 56802 | 16127 07568 | 31089 47600 | 42880 28711 |  |

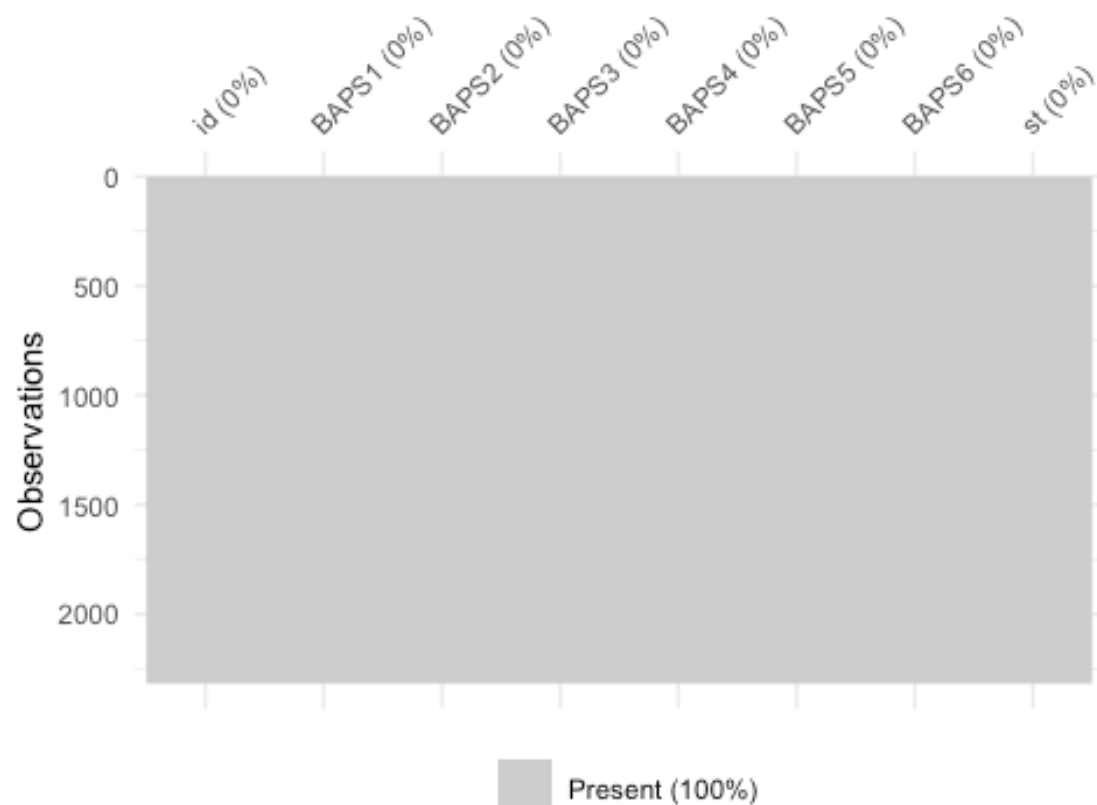
```

# group ST missing values as "Other STs"
data6 <- data6 %>% mutate(st = replace_na(st, "Other STs"))
# check the distribution of serovars to make sure there is only Newport
table(data6$serovar)

##
## Newport
##      2317

# select only needed columns (exclude columns labeled as serovar and cgmlst_ST)
data7 <- data6 %>% select(-c(serovar, cgmlst_ST))
# check for missing values again
vis_miss(data7)

```



```
# another way to check for missing values
skim(data7)
```

Data summary

Name data7
 Number of rows 2317
 Number of columns 8

Column type frequency:

character 2
 factor 6

Group variables None

Variable type: character

| skim_variable | n_missing | complete_rate | min | max | empty | n_unique | whitespace |
|---------------|-----------|---------------|-----|-----|-------|----------|------------|
| id | 0 | 1 | 9 | 11 | 0 | 2317 | 0 |

| | | | | | | | |
|----|---|---|---|---|---|---|---|
| st | 0 | 1 | 3 | 9 | 0 | 8 | 0 |
|----|---|---|---|---|---|---|---|

Variable type: factor

| skim_variable | n_missing | complete_rate | ordered | n_unique | top_counts |
|---------------|-----------|---------------|---------|----------|-------------------------------------|
| BAPS1 | 0 | 1 | FALSE | 5 | 8: 1393, 1: 648, 4: 235, 7: 32 |
| BAPS2 | 0 | 1 | FALSE | 19 | 24: 532, 1: 486, 21: 300, 7: 189 |
| BAPS3 | 0 | 1 | FALSE | 62 | 1: 483, 57: 263, 56: 187, 45: 155 |
| BAPS4 | 0 | 1 | FALSE | 121 | 1: 483, 98: 149, 69: 143, 97: 114 |
| BAPS5 | 0 | 1 | FALSE | 212 | 1: 483, 103: 143, 155: 131, 104: 93 |
| BAPS6 | 0 | 1 | FALSE | 312 | 1: 483, 141: 143, 216: 131, 142: 93 |

```
# yet another way to check for missing values
sum(is.na(data7))

## [1] 0

#####-----#####
#####
#####-----#####
#####

# plot data
# BAPS Levels 1-6 distributions
# BAPS Level 1
# drop the STs and BAPS1 with NAs, group by ST and BAPS1 and then calculate distribution
baps1 <- data7 %>%
  select(st, BAPS1) %>% # select columns
  drop_na(st, BAPS1) %>% # drop NAs for selected columns
  group_by(st, BAPS1) %>% # group by columns
  summarise(n = n()) %>% # count observations
  mutate(total = sum(n)) %>% # sum up values to create a column
  mutate(prop = n/total*100) %>% # calculate proportions
  mutate(group = "BAPS1") %>% # create a group column
  rename(baps = BAPS1) # rename a column

## `summarise()` has grouped output by 'st'. You can override using the `.groups` argument.

# BAPS Level 2
# drop the STs and BAPS2 with NAs, group by ST and BAPS2 and then calculate distribution
baps2 <- data7 %>%
```

```

select(st, BAPS2) %>% # select columns
drop_na(st, BAPS2) %>% # drop NAs for selected columns
group_by(st, BAPS2) %>% # group by columns
summarise(n = n()) %>% # count observations
mutate(total = sum(n)) %>% # sum up values to create a column
mutate(prop = n/total*100) %>% # calculate proportions
mutate(group = "BAPS2") %>% # create a group column
rename(baps = BAPS2) # rename a column

```

`summarise()` has grouped output by 'st'. You can override using the `.groups` argument.

BAPS Level 3

drop the STs and BAPS3 with NAs, group by ST and BAPS3 and then calculate distribution

```

baps3 <- data7 %>%
  select(st, BAPS3) %>% # select columns
  drop_na(st, BAPS3) %>% # drop NAs for selected columns
  group_by(st, BAPS3) %>% # group by columns
  summarise(n = n()) %>% # count observations
  mutate(total = sum(n)) %>% # sum up values to create a column
  mutate(prop = n/total*100) %>% # calculate proportions
  mutate(group = "BAPS3") %>% # create a group column
  rename(baps = BAPS3) # rename a column

```

`summarise()` has grouped output by 'st'. You can override using the `.groups` argument.

BAPS Level 4

drop the STs and BAPS4 with NAs, group by ST and BAPS4 and then calculate distribution

```

baps4 <- data7 %>%
  select(st, BAPS4) %>% # select columns
  drop_na(st, BAPS4) %>% # drop NAs for selected columns
  group_by(st, BAPS4) %>% # group by columns
  summarise(n = n()) %>% # count observations
  mutate(total = sum(n)) %>% # sum up values to create a column
  mutate(prop = n/total*100) %>% # calculate proportions
  mutate(group = "BAPS4") %>% # create a group column
  rename(baps = BAPS4) # rename a column

```

`summarise()` has grouped output by 'st'. You can override using the `.groups` argument.

BAPS Level 5

drop the STs and BAPS5 with NAs, group by ST and BAPS5 and then calculate distribution

```

baps5 <- data7 %>%
  select(st, BAPS5) %>% # select columns
  drop_na(st, BAPS5) %>% # drop NAs for selected columns
  group_by(st, BAPS5) %>% # group by columns

```

```

summarise(n = n()) %>% # count observations
mutate(total = sum(n)) %>% # sum up values to create a column
mutate(prop = n/total*100) %>% # calculate proportions
mutate(group = "BAPS5") %>% # create a group column
rename(baps = BAPS5) # rename a column

```

`summarise()` has grouped output by 'st'. You can override using the `.groups` argument.

BAPS Level 6

drop the STs and BAPS6 with NAs, group by ST and BAPS6 and then calculate distribution

```

baps6 <- data7 %>%
  select(st, BAPS6) %>% # select columns
  drop_na(st, BAPS6) %>% # drop NAs for selected columns
  group_by(st, BAPS6) %>% # group by columns
  summarise(n = n()) %>% # count observations
  mutate(total = sum(n)) %>% # sum up values to create a column
  mutate(prop = n/total*100) %>% # calculate proportions
  mutate(group = "BAPS6") %>% # create a group column
  rename(baps = BAPS6) # rename a column

```

`summarise()` has grouped output by 'st'. You can override using the `.groups` argument.

```

#####-----#####
#####
#####-----#####
#####

```

Concatenate all datasets

```
data8 <- rbind(baps1, baps2, baps3, baps4, baps5, baps6)
```

order BAPS Levels

```
data8$group <- factor(data8$group, levels=c("BAPS1", "BAPS2", "BAPS3",
                                           "BAPS4", "BAPS5", "BAPS6"))
```

```

#####-----#####
#####
#####-----#####
#####

```

plot data

transform baps column to numeric

```
data8 <- data8 %>% mutate(baps = as.numeric(baps))
```

order ST data

```
data8$st <- factor(data8$st, levels=c("ST5", "ST31", "ST45", "ST46", "ST118",
                                       "ST132", "ST350", "Other STs"))
```

```
sup_fig_4 <- ggplot(data8, aes(x = baps, y = prop, fill = st)) + # show BAPS
  Levels on x-axis and proportion on y-axis
```

```
  xlab("BAPS sub-groups (haplotypes)") + ylab("Proportion") + ylim(0, 100) +
  # set labels for axis and limit for y-axis
```

```
  theme_bw() + # set plot background
```

```
  theme(axis.text.x = element_text(angle = 0, size = 25)) + # change x-axis t
```

```

ext font size and angle
  theme(axis.title.x = element_text(size = 35, face = "bold")) + # change x-axis title font size and face
  theme(axis.title.y = element_text(size = 35, face = "bold")) + # change y-axis title font size and face
  theme(axis.text.y = element_text(angle = 0, hjust = 1, size = 25)) + # change y-axis text font size, angle, and orientation
  theme(legend.title=element_text(size=35, face = "bold")) + # customize Legend title
  theme(legend.text=element_text(size=25)) + # customize Legend text
  theme(strip.text.x = element_text(size = 35)) + # customize figure's title, legend, font
  scale_fill_manual(name = "ST", values=c("orange", "darkblue", "purple", "darkgreen", "darkred", "steelblue",
                                           "black", "coral")) + # add colors and labels for the scale
  geom_col(position = "dodge") + # position columns side by side
  facet_wrap(~group, ncol = 2) # generate multi-plots for BAPS with 2 columns
sup_fig_4

```

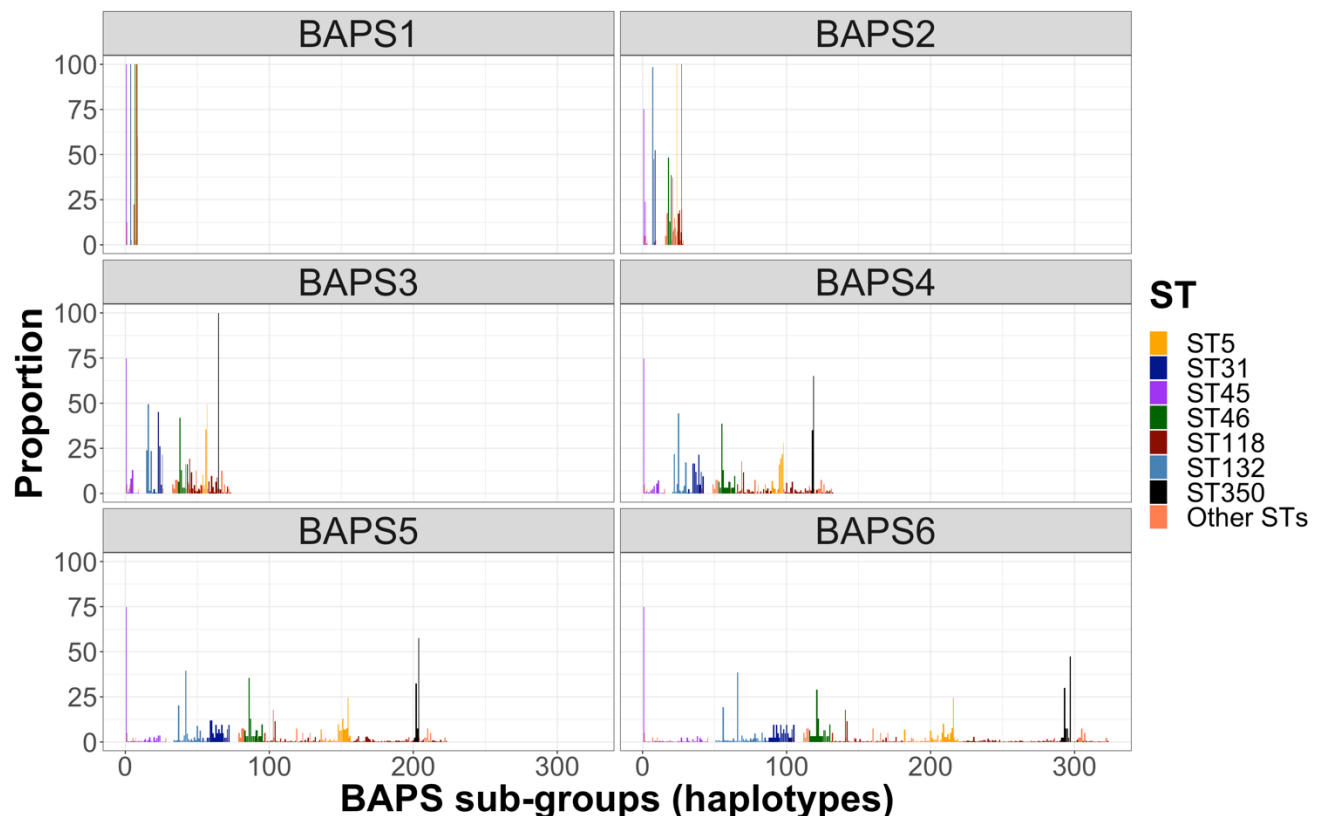


Figure 4. Distribution of Resfinder-annotated AMR loci across ST lineages. Note - only genomes classified as *S. Newport* by SISTR within ProkEvo are included in this analysis.

```

# enter the ST data
mlst <- read_csv('~/Documents/jove_paper/data/salmonellast_output.csv')

```

```

##
## — Column specification —————
##
## cols(
##   FILE = col_character(),
##   SCHEME = col_character(),
##   ST = col_character(),
##   aroC = col_character(),
##   dnaN = col_character(),
##   hemD = col_character(),
##   hisD = col_character(),
##   purE = col_character(),
##   sucA = col_character(),
##   thrA = col_character()
## )

# generate the id column
mlst$id2 <- sapply(strsplit(as.character(mlst$FILE), '_'), "[", 1)
# select columns of interest and rename -/? to NAs
data1 <- mlst %>%
  select(id2, ST) %>% # select columns
  mutate_all(na_if, "-") %>% # transform - to NA
  mutate_all(na_if, "?") %>% # transform ? to NA
  rename(id = id2) # rename column
# categorize ST data to aggregate minor STs and keep only major ST Lineages separately
data1$st <- ifelse(data1$ST == 5, "ST5", # group minor STs as others
  ifelse(data1$ST == 31, "ST31",
    ifelse(data1$ST == 45, "ST45",
      ifelse(data1$ST == 46, "ST46",
        ifelse(data1$ST == 118, "ST118",
          ifelse(data1$ST == 132, "ST132",
            "",
              ifelse(data1$ST == 350, "ST350", "Other STs"))))))))
# filter out the numerical ST column
data1 <- data1 %>% select(-ST)
#####-----#####
#####
#####-----#####
#####
# enter the Resfinder Loci databases (AMR Loci)
abx <- read_csv('~/.Documents/jove_paper/data/sabricate_resfinder_output.csv')

##
## — Column specification —————
##
## cols(
##   `#FILE` = col_character(),
##   SEQUENCE = col_character(),
##   START = col_double(),

```

```

## END = col_double(),
## GENE = col_character(),
## COVERAGE = col_character(),
## COVERAGE_MAP = col_character(),
## GAPS = col_character(),
## `%COVERAGE` = col_double(),
## `%IDENTITY` = col_double(),
## DATABASE = col_character(),
## ACCESSION = col_character(),
## PRODUCT = col_character()
## )

# create the id column
abx$id <- sapply(strsplit(as.character(abx$`#FILE`), '_'), "[", 1)
# change the name of the GENE column
abx <- abx %>% mutate(gene = GENE)
# select columns of interest
abx1 <- abx %>% select(id, gene)
#####-----#####
#####
#####-----#####
#####
# enter SISTR results
sistr <- read_csv('~/Documents/jove_paper/data/sistr_output.csv')

##
## — Column specification —————
##
## cols(
##   cgmlst_ST = col_double(),
##   cgmlst_distance = col_double(),
##   cgmlst_genome_match = col_character(),
##   cgmlst_matching_alleles = col_double(),
##   cgmlst_subspecies = col_character(),
##   fasta_filepath = col_character(),
##   genome = col_character(),
##   h1 = col_character(),
##   h2 = col_character(),
##   o_antigen = col_character(),
##   qc_messages = col_character(),
##   qc_status = col_character(),
##   serogroup = col_character(),
##   serovar = col_character(),
##   serovar_antigen = col_character(),
##   serovar_cgmlst = col_character()
## )

# generate the id column
sistr$id <- sapply(strsplit(as.character(sistr$genome), '_'), "[", 1)
# select id and cgmlst_ST columns and rename -/? to NAs

```

```

data3 <- sistr %>%
  select(id, serovar) %>% # select columns
  mutate_all(na_if, "-") %>% # transform - to NA
  mutate_all(na_if, "?") # transform ? to NA
# group serovars as Newport or others
data3$serovar <- ifelse(data3$serovar == "Newport", "Newport",
                        "Other serovars")
#####-----#####
#####
#####-----#####
#####
# merge datasets
data4 <- left_join(data1, abx1, on = "id") # join datasets based on id

## Joining, by = "id"

data5 <- left_join(data4, data3, on = "id") # join datasets based on id

## Joining, by = "id"

# filter data for Newport only
data6 <- data5 %>% filter(serovar == "Newport")
# check for NAs
skim(data6)

```

Data summary

| | |
|-------------------|-------|
| Name | data6 |
| Number of rows | 9027 |
| Number of columns | 4 |

Column type frequency:

| | |
|-----------|---|
| character | 4 |
|-----------|---|

| | |
|-----------------|------|
| Group variables | None |
|-----------------|------|

Variable type: character

| skim_variable | n_missing | complete_rate | min | max | empty | n_unique | whitespace |
|---------------|-----------|---------------|-----|-----|-------|----------|------------|
| id | 0 | 1 | 9 | 11 | 0 | 2317 | 0 |
| st | 4 | 1 | 3 | 9 | 0 | 8 | 0 |
| gene | 0 | 1 | 6 | 15 | 0 | 119 | 0 |
| serovar | 0 | 1 | 7 | 7 | 0 | 1 | 0 |

```

# group ST missing values as "Other STs"
data6 <- data6 %>% mutate(st = replace_na(st, "Other STs"))

```

```
# check for NAs again
skim(data6)
```

Data summary

```
Name          data6
Number of rows 9027
Number of columns 4
```

Column type frequency:

```
character      4
```

```
Group variables  None
```

Variable type: character

| skim_variable | n_missing | complete_rate | min | max | empty | n_unique | whitespace |
|---------------|-----------|---------------|-----|-----|-------|----------|------------|
| id | 0 | 1 | 9 | 11 | 0 | 2317 | 0 |
| st | 0 | 1 | 3 | 9 | 0 | 8 | 0 |
| gene | 0 | 1 | 6 | 15 | 0 | 119 | 0 |
| serovar | 0 | 1 | 7 | 7 | 0 | 1 | 0 |

```
#####-----#####
#####
#####-----#####
#####
# calculate the proportion of AMR Loci for each ST Lineage
# calculations for ST5
d2a <- data6 %>% filter(st == "ST5")
# for ST5, calculate the proportion of AMR Loci and only keep proportion greater than 10%
d3a <- d2a %>%
  select(id, gene) %>% # select columns
  group_by(id, gene) %>% # group by id and gene
  summarize(count = n()) %>% # count observations
  mutate(count = replace(count, count == 2, 1)) %>% # replace counts equal to 2 by 1
  filter(count <= 1) # filter counts below or equal to 1

## `summarise()` has grouped output by 'id'. You can override using the `.groups` argument.

d4a <- d3a %>%
  group_by(gene) %>% # group by gene
  summarize(value = n()) %>% # count observations
  mutate(total = table(data1$st)[8]) %>% # get the total counts of st
```



```

mutate(prop = (value/total)*100) # calculate proportions
d5a <- d4a %>% mutate(st = "ST5")
#####-----#####
#####
#####-----#####
#####
# calculations for ST31
d2b <- data6 %>% filter(st == "ST31")
# for ST31, calculate the proportion of AMR Loci and only keep proportion gre
ater than 10%
d3b <- d2b %>%
  select(id, gene) %>% # select columns
  group_by(id, gene) %>% # group by id and gene
  summarize(count = n()) %>% # count observations
  mutate(count = replace(count, count == 2, 1)) %>% # replace counts equal
to 2 by 1
  filter(count <= 1) # filter counts below or equal to 1

## `summarise()` has grouped output by 'id'. You can override using the `.gro
ups` argument.

d4b <- d3b %>% group_by(gene) %>% # group by gene
  summarize(value = n()) %>% # count observations
  mutate(total = table(data1$st)[4]) %>% # get the total counts of st
  mutate(prop = (value/total)*100) # calculate proportions
d5b <- d4b %>% mutate(st = "ST31")
#####-----#####
#####
#####-----#####
#####
# calculations for ST45
d2c <- data6 %>% filter(st == "ST45")
# for ST45, calculate the proportion of AMR Loci and only keep proportion gre
ater than 10%
d3c <- d2c %>%
  select(id, gene) %>% # select columns
  group_by(id, gene) %>% # group by id and gene
  summarize(count = n()) %>% # count observations
  mutate(count = replace(count, count == 2, 1)) %>% # replace counts equal
to 2 by 1
  filter(count <= 1) # filter counts below or equal to 1

## `summarise()` has grouped output by 'id'. You can override using the `.gro
ups` argument.

d4c <- d3c %>% group_by(gene) %>% # group by gene
  summarize(value = n()) %>% # count observations
  mutate(total = table(data1$st)[6]) %>% # get the total counts of st
  mutate(prop = (value/total)*100) # calculate proportions
d5c <- d4c %>% mutate(st = "ST45")
#####-----#####

```

```
#####
#####-----#####
#####
# calculations for ST46
d2d <- data6 %>% filter(st == "ST46")
# for ST46, calculate the proportion of AMR Loci and only keep proportion greater than 10%
d3d <- d2d %>%
  select(id, gene) %>% # select columns
  group_by(id, gene) %>% # group by id and gene
  summarize(count = n()) %>% # count observations
  mutate(count = replace(count, count == 2, 1)) %>% # replace counts equal
to 2 by 1
  filter(count <= 1) # filter counts below or equal to 1

## `summarise()` has grouped output by 'id'. You can override using the `.groups` argument.

d4d <- d3d %>% group_by(gene) %>% # group by gene
  summarize(value = n()) %>% # count observations
  mutate(total = table(data1$st)[7]) %>% # get the total counts of st
  mutate(prop = (value/total)*100) # calculate proportions
d5d <- d4d %>% mutate(st = "ST46")
#####-----#####
#####
#####-----#####
#####
# calculations for ST118
d2e <- data6 %>% filter(st == "ST118")
# for ST118, calculate the proportion of AMR Loci and only keep proportion greater than 10%
d3e <- d2e %>%
  select(id, gene) %>% # select columns
  group_by(id, gene) %>% # group by id and gene
  summarize(count = n()) %>% # count observations
  mutate(count = replace(count, count == 2, 1)) %>% # replace counts equal
to 2 by 1
  filter(count <= 1) # filter counts below or equal to 1

## `summarise()` has grouped output by 'id'. You can override using the `.groups` argument.

d4e <- d3e %>% group_by(gene) %>% # group by gene
  summarize(value = n()) %>% # count observations
  mutate(total = table(data1$st)[2]) %>% # get the total counts of st
  mutate(prop = (value/total)*100) # calculate proportions
d5e <- d4e %>% mutate(st = "ST118")
#####-----#####
#####
#####-----#####
#####
```

```

# calculations for ST132
d2f <- data6 %>% filter(st == "ST132")
# for ST132, calculate the proportion of AMR Loci and only keep proportion gr
eater than 10%
d3f <- d2f %>%
  select(id, gene) %>% # select columns
  group_by(id, gene) %>% # group by id and gene
  summarize(count = n()) %>% # count observations
  mutate(count = replace(count, count == 2, 1)) %>% # replace counts equal
to 2 by 1
  filter(count <= 1) # filter counts below or equal to 1

## `summarise()` has grouped output by 'id'. You can override using the `.gro
ups` argument.

d4f <- d3f %>% group_by(gene) %>% # group by gene
  summarize(value = n()) %>% # count observations
  mutate(total = table(data1$st)[3]) %>% # get the total counts of st
  mutate(prop = (value/total)*100) # calculate proportions
d5f <- d4f %>% mutate(st = "ST132")
#####-----#####
#####
#####-----#####
#####
# calculations for ST350
d2g <- data6 %>% filter(st == "ST350")
# for ST350, calculate the proportion of AMR Loci and only keep proportion gr
eater than 10%
d3g <- d2g %>%
  select(id, gene) %>% # select columns
  group_by(id, gene) %>% # group by id and gene
  summarize(count = n()) %>% # count observations
  mutate(count = replace(count, count == 2, 1)) %>% # replace counts equal
to 2 by 1
  filter(count <= 1) # filter counts below or equal to 1

## `summarise()` has grouped output by 'id'. You can override using the `.gro
ups` argument.

d4g <- d3g %>% group_by(gene) %>% # group by gene
  summarize(value = n()) %>% # count observations
  mutate(total = table(data1$st)[5]) %>% # get the total counts of st
  mutate(prop = (value/total)*100) # calculate proportions
d5g <- d4g %>% mutate(st = "ST350")
#####-----#####
#####
#####-----#####
#####
# calculations for Other ST Lineages
d2h <- data6 %>% filter(st == "Other STs")
# for other STs, calculate the proportion of AMR Loci and only keep proportio

```

```

n greater than 10%
d3h <- d2h %>%
  select(id, gene) %>% # select columns
  group_by(id, gene) %>% # group by id and gene
  summarize(count = n()) %>% # count observations
  mutate(count = replace(count, count == 2, 1)) %>% # replace counts equal
to 2 by 1
  filter(count <= 1) # filter counts below or equal to 1

## `summarise()` has grouped output by 'id'. You can override using the `.groups`
argument.

d4h <- d3h %>% group_by(gene) %>% # group by gene
  summarize(value = n()) %>% # count observations
  mutate(total = table(data1$st)[1]) %>% # get the total counts of st
  mutate(prop = (value/total)*100) # calculate proportions
d5h <- d4h %>% mutate(st = "Other STs")
#####-----#####
#####
#####-----#####
#####
# combined datasets
d6 <- rbind(d5a, d5b, d5c, d5d, d5e, d5f, d5g, d5h)
#####-----#####
#####
#####-----#####
#####
# export data table containing ST and AMR Loci information
abx_newport_st <- d6
write.csv(abx_newport_st, "~/Documents/jove_paper/data/abx_newport_st.csv", ro
w.names = FALSE)
#####
# filter AMR Loci with proportion higher than or equal to %
d7 <- d6 %>% filter(prop >= 10)
# order STs
d7$st <- factor(d7$st, levels=c("ST5", "ST31", "ST45", "ST46", "ST118", "ST13
2", "ST350", "Other STs"))
# plot Figure 8
# create a vector object with the number of unique genes or Loci in the data
colourCount = length(unique(d7$gene))
figure_4 <- ggplot(data = d7, mapping = aes(x = prop, y=gene, fill = gene)) +
# show genes on y-axis and proportion on x-axis
  xlab("Proportion") + ylab("AMR loci") + xlim(0, 105) + # set labels fo
r axis and limit for x-axis
  theme_bw() + # set the plot background
  theme(legend.position = "none") + # remove legend
  theme(axis.text.y = element_text(size = 12)) + # change font size for y
-axis text
  theme(axis.title.y = element_text(size = 35, face = "bold")) + # custom
ize y-axis title font size and face

```

```

    theme(axis.title.x = element_text(size = 30, face = "bold")) + # custom
    ize x-axis title font size and face
    theme(axis.text.x = element_text(angle = 0, hjust = 1, size = 20)) + #
    customize x-axis text font size, angle, and orientation
    theme(strip.text.x = element_text(size = 30, colour = "black", angle =
    0)) + # customize figure's title, legend, font
    scale_fill_manual(values = colorRampPalette(brewer.pal(8, "Accent"))(co
    lourCount)) + # add colors for the scale
    geom_bar(stat = "identity") + # show the bars with y values
    facet_wrap(~ st) # generate multi-plots for STs
figure_4

```

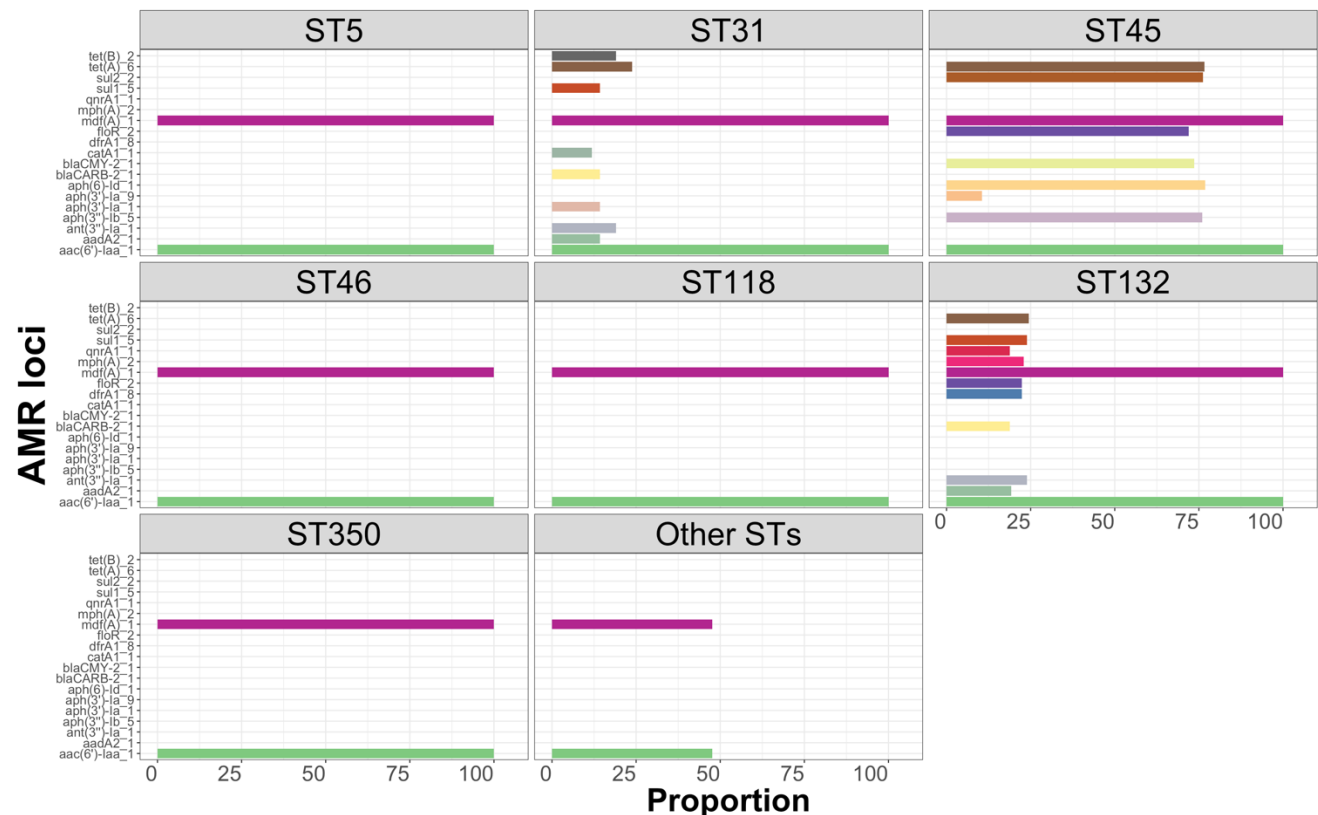


Figure 5 without legend. Phylogeny-guided mapping of hierarchical genotypes along with ST-differentiating AMR loci.

```

# enter the ST data
mlst <- read_csv('~Documents/jove_paper/data/salmonellast_output.csv')

##
## — Column specification —
##
## cols(
##   FILE = col_character(),
##   SCHEME = col_character(),
##   ST = col_character(),
##   aroC = col_character(),

```

```

## dnaN = col_character(),
## hemD = col_character(),
## hisD = col_character(),
## purE = col_character(),
## sucA = col_character(),
## thrA = col_character()
## )

# generate the id column
mlst$id2 <- sapply(strsplit(as.character(mlst$FILE), '_'), "[", 1)
# select columns of interest and rename -/? to NAs
data1 <- mlst %>%
  select(id2, ST) %>% # select columns
  mutate_all(na_if, "-") %>% # change - to NA
  mutate_all(na_if, "?") %>% # change ? to NA
  rename(id = id2) # rename column
# categorize ST data to aggregate minor STs and keep only major ST lineages separately
data1$st <- ifelse(data1$ST == 5, "ST5", # aggregate minor STs as Others
  ifelse(data1$ST == 31, "ST31",
    ifelse(data1$ST == 45, "ST45",
      ifelse(data1$ST == 46, "ST46",
        ifelse(data1$ST == 118, "ST118",
          ifelse(data1$ST == 132, "ST132",
            ",
              ifelse(data1$ST == 350, "ST350", "Other STs"))))))))
# filter out the numerical ST column
data1 <- data1 %>% select(-ST)
#####-----#####
#####
#####-----#####
#####
# enter the Resfinder Loci databases (AMR Loci)
abx <- read_csv('~/Documents/jove_paper/data/sabricate_resfinder_output.csv')

##
## — Column specification —————
## cols(
##   `#FILE` = col_character(),
##   SEQUENCE = col_character(),
##   START = col_double(),
##   END = col_double(),
##   GENE = col_character(),
##   COVERAGE = col_character(),
##   COVERAGE_MAP = col_character(),
##   GAPS = col_character(),
##   `%COVERAGE` = col_double(),
##   `%IDENTITY` = col_double(),
##   DATABASE = col_character(),

```

```
##  ACCESSION = col_character(),
##  PRODUCT = col_character()
## )

# create the id column
abx$id <- sapply(strsplit(as.character(abx$'#FILE'), '_'), "[", 1)
# change the name of the GENE column
abx <- abx %>% mutate(gene = GENE)
# select columns of interest
abx1 <- abx %>% select(id, gene)
#####-----#####
#####
#####-----#####
#####
# join ST and AMR loci data
d1 <- left_join(data1, abx1, on = "id")

## Joining, by = "id"

# check for NAs or missing values
skim(d1)
```

Data summary

| | |
|-------------------|------|
| Name | d1 |
| Number of rows | 9169 |
| Number of columns | 3 |

Column type frequency:

| | |
|-----------|---|
| character | 3 |
|-----------|---|

| | |
|-----------------|------|
| Group variables | None |
|-----------------|------|

Variable type: character

| skim_variable | n_missing | complete_rate | min | max | empty | n_unique | whitespace |
|---------------|-----------|---------------|-----|-----|-------|----------|------------|
| id | 0 | 1 | 9 | 11 | 0 | 2365 | 0 |
| st | 11 | 1 | 3 | 9 | 0 | 8 | 0 |
| gene | 0 | 1 | 6 | 15 | 0 | 127 | 0 |

```
# group ST missing values as "Other STs"
d1 <- d1 %>% mutate(st = replace_na(st, "Other STs"))
# check for NAs again
skim(d1)
```

Data summary

| | |
|-------------------|------|
| Name | d1 |
| Number of rows | 9169 |
| Number of columns | 3 |

Column type frequency:

| | |
|-----------|---|
| character | 3 |
|-----------|---|

| | |
|-----------------|------|
| Group variables | None |
|-----------------|------|

Variable type: character

| skim_variable | n_missing | complete_rate | min | max | empty | n_unique | whitespace |
|---------------|-----------|---------------|-----|-----|-------|----------|------------|
| id | 0 | 1 | 9 | 11 | 0 | 2365 | 0 |
| st | 0 | 1 | 3 | 9 | 0 | 8 | 0 |
| gene | 0 | 1 | 6 | 15 | 0 | 127 | 0 |

```
#####-----#####
#####
#####-----#####
#####
# calculate the proportion of AMR loci for each ST lineage
# calculations for ST5
d2a <- d1 %>% filter(st == "ST5")
# for ST5, calculate the proportion of AMR loci and only keep proportion greater than 10%
d3a <- d2a %>%
  select(id, gene) %>% # select columns
  group_by(id, gene) %>% # group by columns
  summarize(count = n()) %>% # count observations
  mutate(count = replace(count, count == 2, 1)) %>% # replace values in a column
  filter(count <= 1) # filter based on a threshold

## `summarise()` has grouped output by 'id'. You can override using the `.groups` argument.

d4a <- d3a %>%
  group_by(gene) %>% # group by column
  summarize(value = n()) %>% # count observations
  mutate(total = table(data1$st)[8]) %>% # get the total number of observations per ST
  mutate(prop = (value/total)*100) # calculate proportions
d5a <- d4a %>% mutate(st = "ST5")
#####-----#####
```



```
#####
#####-----#####
#####
# calculations for ST31
d2b <- d1 %>% filter(st == "ST31")
# for ST31, calculate the proportion of AMR Loci and only keep proportion gre
ater than 10%
d3b <- d2b %>%
  select(id, gene) %>% # select columns
  group_by(id, gene) %>% # group by columns
  summarize(count = n()) %>% # count observations
  mutate(count = replace(count, count == 2, 1)) %>% # replace values in a c
olumn
  filter(count <= 1) # filter based on a threshold

## `summarise()` has grouped output by 'id'. You can override using the `.gro
ups` argument.

d4b <- d3b %>%
  group_by(gene) %>% # group by column
  summarize(value = n()) %>% # count observations
  mutate(total = table(data1$st)[4]) %>% # get the total number of obs
ervations per ST
  mutate(prop = (value/total)*100) # calculate proportions
d5b <- d4b %>% mutate(st = "ST31")
#####-----#####
#####
#####-----#####
#####
# calculations for ST45
d2c <- d1 %>% filter(st == "ST45")
# for ST45, calculate the proportion of AMR Loci and only keep proportion gre
ater than 10%
d3c <- d2c %>%
  select(id, gene) %>% # select columns
  group_by(id, gene) %>% # group by columns
  summarize(count = n()) %>% # count observations
  mutate(count = replace(count, count == 2, 1)) %>% # replace values in a c
olumn
  filter(count <= 1) # filter based on a threshold

## `summarise()` has grouped output by 'id'. You can override using the `.gro
ups` argument.

d4c <- d3c %>%
  group_by(gene) %>% # group by column
  summarize(value = n()) %>% # count observations
  mutate(total = table(data1$st)[6]) %>% # get the total number of obs
ervations per ST
  mutate(prop = (value/total)*100) # calculate proportions
d5c <- d4c %>% mutate(st = "ST45")
```

```
#####-----#####
#####
#####-----#####
#####
# calculations for ST46
d2d <- d1 %>% filter(st == "ST46")
# for ST46, calculate the proportion of AMR Loci and only keep proportion greater than 10%
d3d <- d2d %>%
  select(id, gene) %>% # select columns
  group_by(id, gene) %>% # group by columns
  summarize(count = n()) %>% # count observations
  mutate(count = replace(count, count == 2, 1)) %>% # replace values in a column
  filter(count <= 1) # filter based on a threshold

## `summarise()` has grouped output by 'id'. You can override using the `.groups` argument.

d4d <- d3d %>%
  group_by(gene) %>% # group by column
  summarize(value = n()) %>% # count observations
  mutate(total = table(data1$st)[7]) %>% # get the total number of observations per ST
  mutate(prop = (value/total)*100) # calculate proportions
d5d <- d4d %>% mutate(st = "ST46")
#####-----#####
#####
#####-----#####
#####
# calculations for ST118
d2e <- d1 %>% filter(st == "ST118")
# for ST118, calculate the proportion of AMR Loci and only keep proportion greater than 10%
d3e <- d2e %>%
  select(id, gene) %>% # select columns
  group_by(id, gene) %>% # group by columns
  summarize(count = n()) %>% # count observations
  mutate(count = replace(count, count == 2, 1)) %>% # replace values in a column
  filter(count <= 1) # filter based on a threshold

## `summarise()` has grouped output by 'id'. You can override using the `.groups` argument.

d4e <- d3e %>%
  group_by(gene) %>% # group by column
  summarize(value = n()) %>% # count observations
  mutate(total = table(data1$st)[2]) %>% # get the total number of observations per ST
  mutate(prop = (value/total)*100) # calculate proportions
```

```

d5e <- d4e %>% mutate(st = "ST118")
#####-----#####
#####
#####-----#####
#####
# calculations for ST132
d2f <- d1 %>% filter(st == "ST132")
# for ST132, calculate the proportion of AMR Loci and only keep proportion gr
eater than 10%
d3f <- d2f %>%
  select(id, gene) %>% # select columns
  group_by(id, gene) %>% # group by columns
  summarize(count = n()) %>% # count observations
  mutate(count = replace(count, count == 2, 1)) %>% # replace values in a c
olumn
  filter(count <= 1) # filter based on a threshold

## `summarise()` has grouped output by 'id'. You can override using the `.gro
ups` argument.

d4f <- d3f %>%
  group_by(gene) %>% # group by column
  summarize(value = n()) %>% # count observations
  mutate(total = table(data1$st)[3]) %>% # get the total number of obs
ervations per ST
  mutate(prop = (value/total)*100) # calculate proportions
d5f <- d4f %>% mutate(st = "ST132")
#####-----#####
#####
#####-----#####
#####
# calculations for ST350
d2g <- d1 %>% filter(st == "ST350")
# for ST350, calculate the proportion of AMR Loci and only keep proportion gr
eater than 10%
d3g <- d2g %>%
  select(id, gene) %>% # select columns
  group_by(id, gene) %>% # group by columns
  summarize(count = n()) %>% # count observations
  mutate(count = replace(count, count == 2, 1)) %>% # replace values in a c
olumn
  filter(count <= 1) # filter based on a threshold

## `summarise()` has grouped output by 'id'. You can override using the `.gro
ups` argument.

d4g <- d3g %>%
  group_by(gene) %>% # group by column
  summarize(value = n()) %>% # count observations
  mutate(total = table(data1$st)[5]) %>% # get the total number of obs
ervations per ST

```

```

      mutate(prop = (value/total)*100) # calculate proportions
d5g <- d4g %>% mutate(st = "ST350")
#####
#####
#####
#####
# calculations for Other ST lineages
d2h <- d1 %>% filter(st == "Other STs")
# for other STs, calculate the proportion of AMR Loci and only keep proportion greater than 10%
d3h <- d2h %>%
  select(id, gene) %>% # select columns
  group_by(id, gene) %>% # group by columns
  summarize(count = n()) %>% # count observations
  mutate(count = replace(count, count == 2, 1)) %>% # replace values in a column
  filter(count <= 1) # filter based on a threshold

## `summarise()` has grouped output by 'id'. You can override using the `.groups` argument.

d4h <- d3h %>% group_by(gene) %>% # group by column
  summarize(value = n()) %>% # count observations
  mutate(total = table(data1$st)[1]) %>% # get the total number of observations per ST
  mutate(prop = (value/total)*100) # calculate proportions
d5h <- d4h %>% mutate(st = "Other STs")
#####
#####
#####
#####
# combined datasets
d6 <- rbind(d5a, d5b, d5c, d5d, d5e, d5f, d5g, d5h)
#####
#####
#####
#####
# filter AMR Loci with proportion higher than or equal to 10%
d7 <- d6 %>% filter(prop >= 10)
# create a list of genes with proportion >= 10%
d8 <- d7 %>% select(gene) %>% unique()
# transform dataframe to list of characters or vector
d9 <- pull(d8, gene)
#####
#####
#####
#####
# spread the abx1 table - from long to wide format
# count gene occurrences
abx2 <- abx1 %>%

```

```

    group_by(id, gene) %>% # group by id and gene
    summarize(count = n()) %>% # count the number of observations
    filter(count <= 1) # filter genes with 0 or 1 counts

## `summarise()` has grouped output by 'id'. You can override using the `.groups` argument.

# spread from long to wide format
abx3 <- spread(abx2, key = gene, value = count)
# select columns of interest
abx4 <- abx3 %>% select(d9)

## Note: Using an external vector in selections is ambiguous.
##  Use `all_of(d9)` instead of `d9` to silence this message.
##  See <https://tidyselect.r-lib.org/reference/faq-external-vector.html>.
## This message is displayed once per session.

## Adding missing grouping variables: `id`

# replace all NAs with zeros
abx4[is.na(abx4)] <- 0
# merge with original data containing the hierarchical genotypes
d4 <- left_join(d3, abx4, on = "id")

## Joining, by = "id"

#####-----#####
#####
#####-----#####
#####

# change binary values from AMR data locus by locus
d4 <- d4 %>% mutate(`aac(6')-Iaa_1` = factor(ifelse(`aac(6')-Iaa_1` == 1, "aac(6')-Iaa_1 (present)", "aac(6')-Iaa_1 (absent)")))
d4 <- d4 %>% mutate(`mdf(A)_1` = factor(ifelse(`mdf(A)_1` == 1, "mdf(A)_1 (present)", "mdf(A)_1 (absent)")))
d4 <- d4 %>% mutate(aadA2_1 = factor(ifelse(aadA2_1 == 1, "aadA2_1 (present)", "aadA2_1 (absent)")))
d4 <- d4 %>% mutate(`ant(3')-Ia_1` = factor(ifelse(`ant(3')-Ia_1` == 1, "ant(3')-Ia_1 (present)", "ant(3')-Ia_1 (absent)")))
d4 <- d4 %>% mutate(`aph(3')-Ia_1` = factor(ifelse(`aph(3')-Ia_1` == 1, "aph(3')-Ia_1 (present)", "aph(3')-Ia_1 (absent)")))
d4 <- d4 %>% mutate(`blaCARB-2_1` = factor(ifelse(`blaCARB-2_1` == 1, "blaCARB-2_1 (present)", "blaCARB-2_1 (absent)")))
d4 <- d4 %>% mutate(catA1_1 = factor(ifelse(catA1_1 == 1, "catA1_1 (present)", "catA1_1 (absent)")))
d4 <- d4 %>% mutate(sul1_5 = factor(ifelse(sul1_5 == 1, "sul1_5 (present)", "sul1_5 (absent)")))
d4 <- d4 %>% mutate(`tet(A)_6` = factor(ifelse(`tet(A)_6` == 1, "tet(A)_6 (present)", "tet(A)_6 (absent)")))
d4 <- d4 %>% mutate(`tet(B)_2` = factor(ifelse(`tet(B)_2` == 1, "tet(B)_2 (present)", "tet(B)_2 (absent)")))

```

```
d4 <- d4 %>% mutate(`aph(3')-Ib_5` = factor(ifelse(`aph(3')-Ib_5` == 1, "aph(3')-Ib_5 (present)", "aph(3')-Ib_5 (absent)")))
d4 <- d4 %>% mutate(`aph(3')-Ia_9` = factor(ifelse(`aph(3')-Ia_9` == 1, "aph(3')-Ia_9 (present)", "aph(3')-Ia_9 (absent)")))
d4 <- d4 %>% mutate(`aph(6)-Id_1` = factor(ifelse(`aph(6)-Id_1` == 1, "aph(6)-Id_1 (present)", "aph(6)-Id_1 (absent)")))
d4 <- d4 %>% mutate(`blaCMY-2_1` = factor(ifelse(`blaCMY-2_1` == 1, "blaCMY-2_1 (present)", "blaCMY-2_1 (absent)")))
d4 <- d4 %>% mutate(`floR_2` = factor(ifelse(floR_2 == 1, "floR_2 (present)", "floR_2 (absent)")))
d4 <- d4 %>% mutate(`sul2_2` = factor(ifelse(sul2_2 == 1, "sul2_2 (present)", "sul2_2 (absent)")))
d4 <- d4 %>% mutate(`dfrA1_8` = factor(ifelse(dfrA1_8 == 1, "dfrA1_8 (present)", "dfrA1_8 (absent)")))
d4 <- d4 %>% mutate(`mph(A)_2` = factor(ifelse(`mph(A)_2` == 1, "mph(A)_2 (present)", "mph(A)_2 (absent)")))
d4 <- d4 %>% mutate(`qnrA1_1` = factor(ifelse(qnrA1_1 == 1, "qnrA1_1 (present)", "qnrA1_1 (absent)")))
#####-----#####
#####
#####-----#####
#####
# enter the phylogeny data
tree <- read.tree("~/Documents/jove_paper/data/newport_phylogeny.tree")
# bring data containing hierarchical genotypes and AMR Loci
d5 <- d4
# to plot with the phylogeny using ggtree we need to make the id column into index first
d6 <- column_to_rownames(d5, var = "id")
# create the tree
# adjusting the parameters to make the tree visible or however you wish requires some trial and error
tree_plot <- ggtree(tree, layout = "circular") + xlim(-50, NA)
# plot figure 1 - color scheme for each layer of the plot should be chosen based on the user preferences
figure_5 <- gheatmap(tree_plot, d6, offset=.0, width=50, colnames = FALSE) +
# visualize the tree with metadata
  scale_fill_manual(values = c("coral", "darkblue", "cornflowerblue", "coral", "purple", "red", "brown", "darkseagreen3", "darkblue", "darkgreen", "yellow", "orange", "darkblue", "purple", "darkgreen", "darkred", "steelblue", "black", "coral", "black", "darkgreen", "purple", "darkblue", "gray", "darkblue", "gray", "darkblue", "gray", "darkblue", "gray", "darkblue", "gray", "darkblue", "gray"))
```

```

        "darkblue", "gray",
        "darkblue", "gray",
        "darkblue", "gray",
        "darkblue", "gray",
        "darkblue", "gray",
        "darkblue", "gray",
        "darkblue", "gray",
        "darkblue", "gray",
        "darkblue", "gray",
        "darkblue", "gray",
        "darkblue", "gray",
        "darkblue", "gray",
        "darkblue", "gray"),
    breaks = c("Newport", "Other serovars",
        "BAPS1 sub-group 1", "BAPS1 sub-group 2", "BAP
S1 sub-group 3", "BAPS1 sub-group 4",
        "BAPS1 sub-group 5", "BAPS1 sub-group 6", "BAP
S1 sub-group 7", "BAPS1 sub-group 8",
        "BAPS1 sub-group 9",
        "ST5", "ST31", "ST45", "ST46", "ST118", "ST132
", "ST350", "Other STs",
        "cgMLST 1468400426", "cgMLST 1271156802", "cgM
LST 3336043520", "cgMLST 88443731", "Other cgMLSTs",
        "aac(6')-Iaa_1 (present)", "aac(6')-Iaa_1 (abs
ent)",
        "mdf(A)_1 (present)", "mdf(A)_1 (absent)",
        "aadA2_1 (present)", "aadA2_1 (absent)",
        "ant(3'')-Ia_1 (present)", "ant(3'')-Ia_1 (abs
ent)",
        "aph(3')-Ia_1 (present)", "aph(3')-Ia_1 (absen
t)",
        "blaCARB-2_1 (present)", "blaCARB-2_1 (absent)
",
        "catA1_1 (present)", "catA1_1 (absent)",
        "sul1_5 (present)", "sul1_5 (absent)",
        "tet(A)_6 (present)", "tet(A)_6 (absent)",
        "tet(B)_2 (present)", "tet(B)_2 (absent)",
        "aph(3'')-Ib_5 (present)", "aph(3'')-Ib_5 (abs
ent)",
        "aph(3')-Ia_9 (present)", "aph(3')-Ia_9 (absen
t)",
        "aph(6)-Id_1 (present)", "aph(6)-Id_1 (absent)
",
        "blaCMY-2_1 (present)", "blaCMY-2_1 (absent)",
        "floR_2 (present)", "floR_2 (absent)",
        "sul2_2 (present)", "sul2_2 (absent)",
        "dfrA1_8 (present)", "dfrA1_8 (absent)",
        "mph(A)_2 (present)", "mph(A)_2 (absent)",
        "qnrA1_1 (present)", "qnrA1_1 (absent)",
    name="Serovar -> BAPS1 -> ST -> cgMLST -> AMR loci") + #

```

```

add colors and labels for the scale
  theme(legend.title=element_text(size=24, face = "bold"),
        legend.text=element_text(size=22)) +
        theme(legend.position = "none") # customize figure's title, legend,
font

## Warning: attributes are not identical across measure variables;
## they will be dropped

## Scale for 'y' is already present. Adding another scale for 'y', which will
## replace the existing scale.

## Scale for 'fill' is already present. Adding another scale for 'fill', whic
h
## will replace the existing scale.

figure_5

```

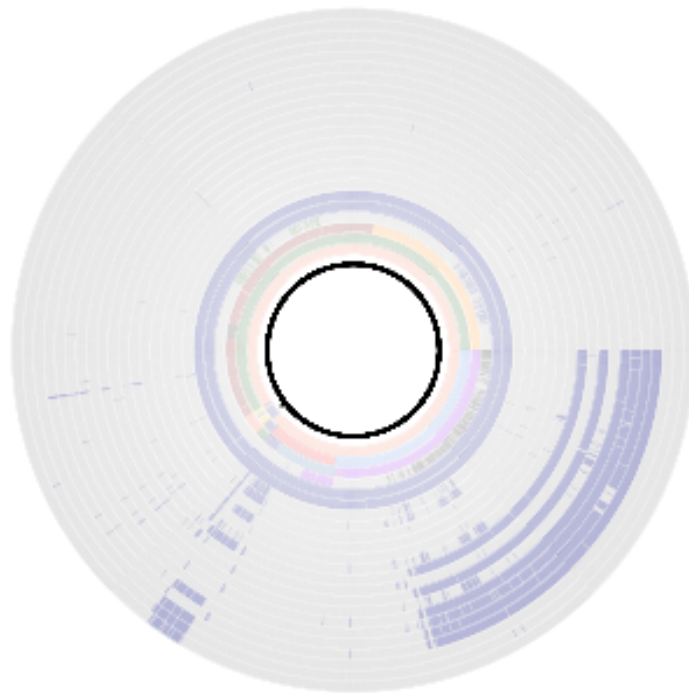


Figure 5

with legend. Phylogeny-guided mapping of hierarchical genotypes along with ST-differentiating AMR loci.

```

# enter the ST data
mlst <- read_csv('~/Documents/jove_paper/data/salmonellast_output.csv')

```



```

##
## — Column specification —————
##
## cols(
##   FILE = col_character(),
##   SCHEME = col_character(),
##   ST = col_character(),
##   aroC = col_character(),
##   dnaN = col_character(),
##   hemD = col_character(),
##   hisD = col_character(),
##   purE = col_character(),
##   sucA = col_character(),
##   thrA = col_character()
## )

# generate the id column
mlst$id2 <- sapply(strsplit(as.character(mlst$FILE), '_'), "[", 1)
# select columns of interest and rename -/? to NAs
data1 <- mlst %>%
  select(id2, ST) %>% # select columns
  mutate_all(na_if, "-") %>% # change - to NA
  mutate_all(na_if, "?") %>% # change ? to NA
  rename(id = id2) # rename column
# categorize ST data to aggregate minor STs and keep only major ST Lineages separately
data1$st <- ifelse(data1$ST == 5, "ST5", # aggregate minor STs as Others
  ifelse(data1$ST == 31, "ST31",
    ifelse(data1$ST == 45, "ST45",
      ifelse(data1$ST == 46, "ST46",
        ifelse(data1$ST == 118, "ST118",
          ifelse(data1$ST == 132, "ST132",
            "",
              ifelse(data1$ST == 350, "ST350", "Other STs"))))))))
# filter out the numerical ST column
data1 <- data1 %>% select(-ST)
#####-----#####
#####
#####-----#####
#####
# enter the Resfinder Loci databases (AMR Loci)
abx <- read_csv('~/.Documents/jove_paper/data/sabricate_resfinder_output.csv')

##
## — Column specification —————
##
## cols(
##   `#FILE` = col_character(),
##   SEQUENCE = col_character(),
##   START = col_double(),

```

```
## END = col_double(),
## GENE = col_character(),
## COVERAGE = col_character(),
## COVERAGE_MAP = col_character(),
## GAPS = col_character(),
## `%COVERAGE` = col_double(),
## `%IDENTITY` = col_double(),
## DATABASE = col_character(),
## ACCESSION = col_character(),
## PRODUCT = col_character()
## )

# create the id column
abx$id <- sapply(strsplit(as.character(abx$`#FILE`), '_'), "[", 1)
# change the name of the GENE column
abx <- abx %>% mutate(gene = GENE)
# select columns of interest
abx1 <- abx %>% select(id, gene)
#####-----#####
#####
#####-----#####
#####
# join ST and AMR Loci data
d1 <- left_join(data1, abx1, on = "id")

## Joining, by = "id"

# check for NAs or missing values
skim(d1)
```

Data summary

| | |
|-------------------|------|
| Name | d1 |
| Number of rows | 9169 |
| Number of columns | 3 |

Column type frequency:

| | |
|-----------|---|
| character | 3 |
|-----------|---|

| | |
|-----------------|------|
| Group variables | None |
|-----------------|------|

Variable type: character

| skim_variable | n_missing | complete_rate | min | max | empty | n_unique | whitespace |
|---------------|-----------|---------------|-----|-----|-------|----------|------------|
| id | 0 | 1 | 9 | 11 | 0 | 2365 | 0 |
| st | 11 | 1 | 3 | 9 | 0 | 8 | 0 |

```

gene          0          1    6   15    0   127    0
# group ST missing values as "Other STs"
d1 <- d1 %>% mutate(st = replace_na(st, "Other STs"))
# check for NAs again
skim(d1)

```

Data summary

```

Name          d1
Number of rows 9169
Number of columns 3

```

Column type frequency:

```

character      3

```

```

Group variables      None

```

Variable type: character

| skim_variable | n_missing | complete_rate | min | max | empty | n_unique | whitespace |
|---------------|-----------|---------------|-----|-----|-------|----------|------------|
| id | 0 | 1 | 9 | 11 | 0 | 2365 | 0 |
| st | 0 | 1 | 3 | 9 | 0 | 8 | 0 |
| gene | 0 | 1 | 6 | 15 | 0 | 127 | 0 |

```

#####-----#####
#####
#####-----#####
#####
# calculate the proportion of AMR Loci for each ST Lineage
# calculations for ST5
d2a <- d1 %>% filter(st == "ST5")
# for ST5, calculate the proportion of AMR Loci and only keep proportion greater than 10%
d3a <- d2a %>%
  select(id, gene) %>% # select columns
  group_by(id, gene) %>% # group by columns
  summarize(count = n()) %>% # count observations
  mutate(count = replace(count, count == 2, 1)) %>% # replace values in a column
  filter(count <= 1) # filter based on a threshold

## `summarise()` has grouped output by 'id'. You can override using the `.groups` argument.

d4a <- d3a %>%
  group_by(gene) %>% # group by column

```

```

    summarize(value = n()) %>% # count observations
    mutate(total = table(data1$st)[8]) %>% # get the total number of observations per ST
    mutate(prop = (value/total)*100) # calculate proportions
d5a <- d4a %>% mutate(st = "ST5")
#####-----#####
#####
#####-----#####
#####
# calculations for ST31
d2b <- d1 %>% filter(st == "ST31")
# for ST31, calculate the proportion of AMR loci and only keep proportion greater than 10%
d3b <- d2b %>%
  select(id, gene) %>% # select columns
  group_by(id, gene) %>% # group by columns
  summarize(count = n()) %>% # count observations
  mutate(count = replace(count, count == 2, 1)) %>% # replace values in a column
  filter(count <= 1) # filter based on a threshold

## `summarise()` has grouped output by 'id'. You can override using the `.groups` argument.

d4b <- d3b %>%
  group_by(gene) %>% # group by column
  summarize(value = n()) %>% # count observations
  mutate(total = table(data1$st)[4]) %>% # get the total number of observations per ST
  mutate(prop = (value/total)*100) # calculate proportions
d5b <- d4b %>% mutate(st = "ST31")
#####-----#####
#####
#####-----#####
#####
# calculations for ST45
d2c <- d1 %>% filter(st == "ST45")
# for ST45, calculate the proportion of AMR loci and only keep proportion greater than 10%
d3c <- d2c %>%
  select(id, gene) %>% # select columns
  group_by(id, gene) %>% # group by columns
  summarize(count = n()) %>% # count observations
  mutate(count = replace(count, count == 2, 1)) %>% # replace values in a column
  filter(count <= 1) # filter based on a threshold

## `summarise()` has grouped output by 'id'. You can override using the `.groups` argument.

```

```

d4c <- d3c %>%
  group_by(gene) %>% # group by column
  summarize(value = n()) %>% # count observations
  mutate(total = table(data1$st)[6]) %>% # get the total number of observations per ST
  mutate(prop = (value/total)*100) # calculate proportions
d5c <- d4c %>% mutate(st = "ST45")
#####-----#####
#####
#####-----#####
#####
# calculations for ST46
d2d <- d1 %>% filter(st == "ST46")
# for ST46, calculate the proportion of AMR Loci and only keep proportion greater than 10%
d3d <- d2d %>%
  select(id, gene) %>% # select columns
  group_by(id, gene) %>% # group by columns
  summarize(count = n()) %>% # count observations
  mutate(count = replace(count, count == 2, 1)) %>% # replace values in a column
  filter(count <= 1) # filter based on a threshold

## `summarise()` has grouped output by 'id'. You can override using the `.groups` argument.

d4d <- d3d %>%
  group_by(gene) %>% # group by column
  summarize(value = n()) %>% # count observations
  mutate(total = table(data1$st)[7]) %>% # get the total number of observations per ST
  mutate(prop = (value/total)*100) # calculate proportions
d5d <- d4d %>% mutate(st = "ST46")
#####-----#####
#####
#####-----#####
#####
# calculations for ST118
d2e <- d1 %>% filter(st == "ST118")
# for ST118, calculate the proportion of AMR Loci and only keep proportion greater than 10%
d3e <- d2e %>%
  select(id, gene) %>% # select columns
  group_by(id, gene) %>% # group by columns
  summarize(count = n()) %>% # count observations
  mutate(count = replace(count, count == 2, 1)) %>% # replace values in a column
  filter(count <= 1) # filter based on a threshold

```

`summarise()` has grouped output by 'id'. You can override using the `.groups` argument.

```
d4e <- d3e %>%
  group_by(gene) %>% # group by column
  summarize(value = n()) %>% # count observations
  mutate(total = table(data1$st)[2]) %>% # get the total number of observations per ST
  mutate(prop = (value/total)*100) # calculate proportions
```

```
d5e <- d4e %>% mutate(st = "ST118")
#####-----#####
#####
#####-----#####
#####
```

calculations for ST132

```
d2f <- d1 %>% filter(st == "ST132")
# for ST132, calculate the proportion of AMR Loci and only keep proportion greater than 10%
```

```
d3f <- d2f %>%
  select(id, gene) %>% # select columns
  group_by(id, gene) %>% # group by columns
  summarize(count = n()) %>% # count observations
  mutate(count = replace(count, count == 2, 1)) %>% # replace values in a column
  filter(count <= 1) # filter based on a threshold
```

`summarise()` has grouped output by 'id'. You can override using the `.groups` argument.

```
d4f <- d3f %>%
  group_by(gene) %>% # group by column
  summarize(value = n()) %>% # count observations
  mutate(total = table(data1$st)[3]) %>% # get the total number of observations per ST
  mutate(prop = (value/total)*100) # calculate proportions
```

```
d5f <- d4f %>% mutate(st = "ST132")
#####-----#####
#####
#####-----#####
#####
```

calculations for ST350

```
d2g <- d1 %>% filter(st == "ST350")
# for ST350, calculate the proportion of AMR Loci and only keep proportion greater than 10%
```

```
d3g <- d2g %>%
  select(id, gene) %>% # select columns
  group_by(id, gene) %>% # group by columns
  summarize(count = n()) %>% # count observations
  mutate(count = replace(count, count == 2, 1)) %>% # replace values in a column
```

```

column
  filter(count <= 1) # filter based on a threshold

## `summarise()` has grouped output by 'id'. You can override using the `.groups` argument.

d4g <- d3g %>%
  group_by(gene) %>% # group by column
  summarize(value = n()) %>% # count observations
  mutate(total = table(data1$st)[5]) %>% # get the total number of observations per ST
  mutate(prop = (value/total)*100) # calculate proportions
d5g <- d4g %>% mutate(st = "ST350")
#####
#####
#####
#####
# calculations for Other ST lineages
d2h <- d1 %>% filter(st == "Other STs")
# for other STs, calculate the proportion of AMR Loci and only keep proportion greater than 10%
d3h <- d2h %>%
  select(id, gene) %>% # select columns
  group_by(id, gene) %>% # group by columns
  summarize(count = n()) %>% # count observations
  mutate(count = replace(count, count == 2, 1)) %>% # replace values in a column
column
  filter(count <= 1) # filter based on a threshold

## `summarise()` has grouped output by 'id'. You can override using the `.groups` argument.

d4h <- d3h %>% group_by(gene) %>% # group by column
  summarize(value = n()) %>% # count observations
  mutate(total = table(data1$st)[1]) %>% # get the total number of observations per ST
  mutate(prop = (value/total)*100) # calculate proportions
d5h <- d4h %>% mutate(st = "Other STs")
#####
#####
#####
#####
# combined datasets
d6 <- rbind(d5a, d5b, d5c, d5d, d5e, d5f, d5g, d5h)
#####
#####
#####
#####
# filter AMR Loci with proportion higher than or equal to %
d7 <- d6 %>% filter(prop >= 10)
# create a list of genes with proportion >= 10%

```

```

d8 <- d7 %>% select(gene) %>% unique()
# transform dataframe to list of characters or vector
d9 <- pull(d8, gene)
#####-----#####
#####
#####-----#####
#####
# spread the abx1 table - from long to wide format
# count gene occurrences
abx2 <- abx1 %>%
  group_by(id, gene) %>% # group by id and gene
  summarize(count = n()) %>% # count the number of observations
  filter(count <= 1) # filter genes with 0 or 1 counts

## `summarise()` has grouped output by 'id'. You can override using the `.groups` argument.

# spread from long to wide format
abx3 <- spread(abx2, key = gene, value = count)
# select columns of interest
abx4 <- abx3 %>% select(d9)

## Adding missing grouping variables: `id`

# replace all NAs with zeros
abx4[is.na(abx4)] <- 0
# merge with original data containing the hierarchical genotypes
d4 <- left_join(d3, abx4, on = "id")

## Joining, by = "id"

#####-----#####
#####
#####-----#####
#####
# change binary values from AMR data Locus by Locus
d4 <- d4 %>% mutate(`aac(6')-Iaa_1` = factor(ifelse(`aac(6')-Iaa_1` == 1, "aac(6')-Iaa_1 (present)", "aac(6')-Iaa_1 (absent)")))
d4 <- d4 %>% mutate(`mdf(A)_1` = factor(ifelse(`mdf(A)_1` == 1, "mdf(A)_1 (present)", "mdf(A)_1 (absent)")))
d4 <- d4 %>% mutate(`aadA2_1` = factor(ifelse(aadA2_1 == 1, "aadA2_1 (present)", "aadA2_1 (absent)")))
d4 <- d4 %>% mutate(`ant(3')-Ia_1` = factor(ifelse(`ant(3')-Ia_1` == 1, "ant(3')-Ia_1 (present)", "ant(3')-Ia_1 (absent)")))
d4 <- d4 %>% mutate(`aph(3')-Ia_1` = factor(ifelse(`aph(3')-Ia_1` == 1, "aph(3')-Ia_1 (present)", "aph(3')-Ia_1 (absent)")))
d4 <- d4 %>% mutate(`blaCARB-2_1` = factor(ifelse(`blaCARB-2_1` == 1, "blaCARB-2_1 (present)", "blaCARB-2_1 (absent)")))
d4 <- d4 %>% mutate(`catA1_1` = factor(ifelse(catA1_1 == 1, "catA1_1 (present)", "catA1_1 (absent)")))
d4 <- d4 %>% mutate(`sul1_5` = factor(ifelse(sul1_5 == 1, "sul1_5 (present)", "

```



```

sul1_5 (absent)))))
d4 <- d4 %>% mutate(`tet(A)_6` = factor(ifelse(`tet(A)_6` == 1, "tet(A)_6 (present)", "tet(A)_6 (absent)")))
d4 <- d4 %>% mutate(`tet(B)_2` = factor(ifelse(`tet(B)_2` == 1, "tet(B)_2 (present)", "tet(B)_2 (absent)")))
d4 <- d4 %>% mutate(`aph(3')-Ib_5` = factor(ifelse(`aph(3')-Ib_5` == 1, "aph(3')-Ib_5 (present)", "aph(3')-Ib_5 (absent)")))
d4 <- d4 %>% mutate(`aph(3')-Ia_9` = factor(ifelse(`aph(3')-Ia_9` == 1, "aph(3')-Ia_9 (present)", "aph(3')-Ia_9 (absent)")))
d4 <- d4 %>% mutate(`aph(6)-Id_1` = factor(ifelse(`aph(6)-Id_1` == 1, "aph(6)-Id_1 (present)", "aph(6)-Id_1 (absent)")))
d4 <- d4 %>% mutate(`blaCMY-2_1` = factor(ifelse(`blaCMY-2_1` == 1, "blaCMY-2_1 (present)", "blaCMY-2_1 (absent)")))
d4 <- d4 %>% mutate(floR_2 = factor(ifelse(floR_2 == 1, "floR_2 (present)", "floR_2 (absent)")))
d4 <- d4 %>% mutate(sul2_2 = factor(ifelse(sul2_2 == 1, "sul2_2 (present)", "sul2_2 (absent)")))
d4 <- d4 %>% mutate(dfrA1_8 = factor(ifelse(dfrA1_8 == 1, "dfrA1_8 (present)", "dfrA1_8 (absent)")))
d4 <- d4 %>% mutate(`mph(A)_2` = factor(ifelse(`mph(A)_2` == 1, "mph(A)_2 (present)", "mph(A)_2 (absent)")))
d4 <- d4 %>% mutate(qnrA1_1 = factor(ifelse(qnrA1_1 == 1, "qnrA1_1 (present)", "qnrA1_1 (absent)")))
#####-----#####
#####
#####-----#####
#####
# enter the phylogeny data
tree <- read.tree("~/Documents/jove_paper/data/newport_phylogeny.tree")
# bring data containing hierarchical genotypes and AMR loci
d5 <- d4
# to plot with the phylogeny using ggtree we need to make the id column into index first
d6 <- column_to_rownames(d5, var = "id")
# create the tree
# adjusting the parameters to make the tree visible or however you wish requires some trial and error
tree_plot <- ggtree(tree, layout = "circular") + xlim(-50, NA)
# plot figure 1 - color scheme for each layer of the plot should be chosen based on the user preferences
figure_5b <- gheatmap(tree_plot, d6, offset=.0, width=50, colnames = FALSE) +
# visualize the tree with metadata
  scale_fill_manual(values = c("coral", "darkblue",
                                "cornflowerblue", "coral", "purple", "red", "brown", "darkseagreen3", "darkblue", "darkgreen", "yellow",
                                "orange", "darkblue", "purple", "darkgreen", "darkred", "steelblue", "black", "coral",
                                "black", "darkgreen", "purple", "darkblue", "gray",
                                "darkblue", "gray",

```

```

        "darkblue", "gray",
        "darkblue", "gray",
        "darkblue", "gray",
        "darkblue", "gray",
        "darkblue", "gray",
        "darkblue", "gray",
        "darkblue", "gray",
        "darkblue", "gray",
        "darkblue", "gray",
        "darkblue", "gray",
        "darkblue", "gray",
        "darkblue", "gray",
        "darkblue", "gray",
        "darkblue", "gray",
        "darkblue", "gray",
        "darkblue", "gray",
        "darkblue", "gray",
        "darkblue", "gray"),
    breaks = c("Newport", "Other serovars",
        "BAPS1 sub-group 1", "BAPS1 sub-group 2", "BAP
S1 sub-group 3", "BAPS1 sub-group 4",
        "BAPS1 sub-group 5", "BAPS1 sub-group 6", "BAP
S1 sub-group 7", "BAPS1 sub-group 8",
        "BAPS1 sub-group 9",
        "ST5", "ST31", "ST45", "ST46", "ST118", "ST132
", "ST350", "Other STs",
        "cgMLST 1468400426", "cgMLST 1271156802", "cgM
LST 3336043520", "cgMLST 88443731", "Other cgMLSTs",
        "aac(6')-Iaa_1 (present)", "aac(6')-Iaa_1 (abs
ent)",
        "mdf(A)_1 (present)", "mdf(A)_1 (absent)",
        "aadA2_1 (present)", "aadA2_1 (absent)",
        "ant(3'')-Ia_1 (present)", "ant(3'')-Ia_1 (abs
ent)",
        "aph(3')-Ia_1 (present)", "aph(3')-Ia_1 (absen
t)",
        "blaCARB-2_1 (present)", "blaCARB-2_1 (absent)
",
        "catA1_1 (present)", "catA1_1 (absent)",
        "sul1_5 (present)", "sul1_5 (absent)",
        "tet(A)_6 (present)", "tet(A)_6 (absent)",
        "tet(B)_2 (present)", "tet(B)_2 (absent)",
        "aph(3'')-Ib_5 (present)", "aph(3'')-Ib_5 (abs
ent)",
        "aph(3')-Ia_9 (present)", "aph(3')-Ia_9 (absen
t)",
        "aph(6)-Id_1 (present)", "aph(6)-Id_1 (absent)
",
        "blaCMY-2_1 (present)", "blaCMY-2_1 (absent)",
        "floR_2 (present)", "floR_2 (absent)",

```

```

        "sul2_2 (present)", "sul2_2 (absent)",
        "dfrA1_8 (present)", "dfrA1_8 (absent)",
        "mph(A)_2 (present)", "mph(A)_2 (absent)",
        "qnrA1_1 (present)", "qnrA1_1 (absent)",
        name="Serovar -> BAPS1 -> ST -> cgMLST -> AMR loci") + #
add colors and labels for the scale
theme(legend.title=element_text(size=24, face = "bold"),
      legend.text=element_text(size=22)) # customize figure's title, legend, font
nt

## Warning: attributes are not identical across measure variables;
## they will be dropped

## Scale for 'y' is already present. Adding another scale for 'y', which will
## replace the existing scale.

## Scale for 'fill' is already present. Adding another scale for 'fill', which
## will replace the existing scale.

figure_5b

```

