Refactoring Riboviz Analysis Code:

A Personal Journey

24 June 2020
Flic Anderson
The Wallace Lab, University of Edinburgh

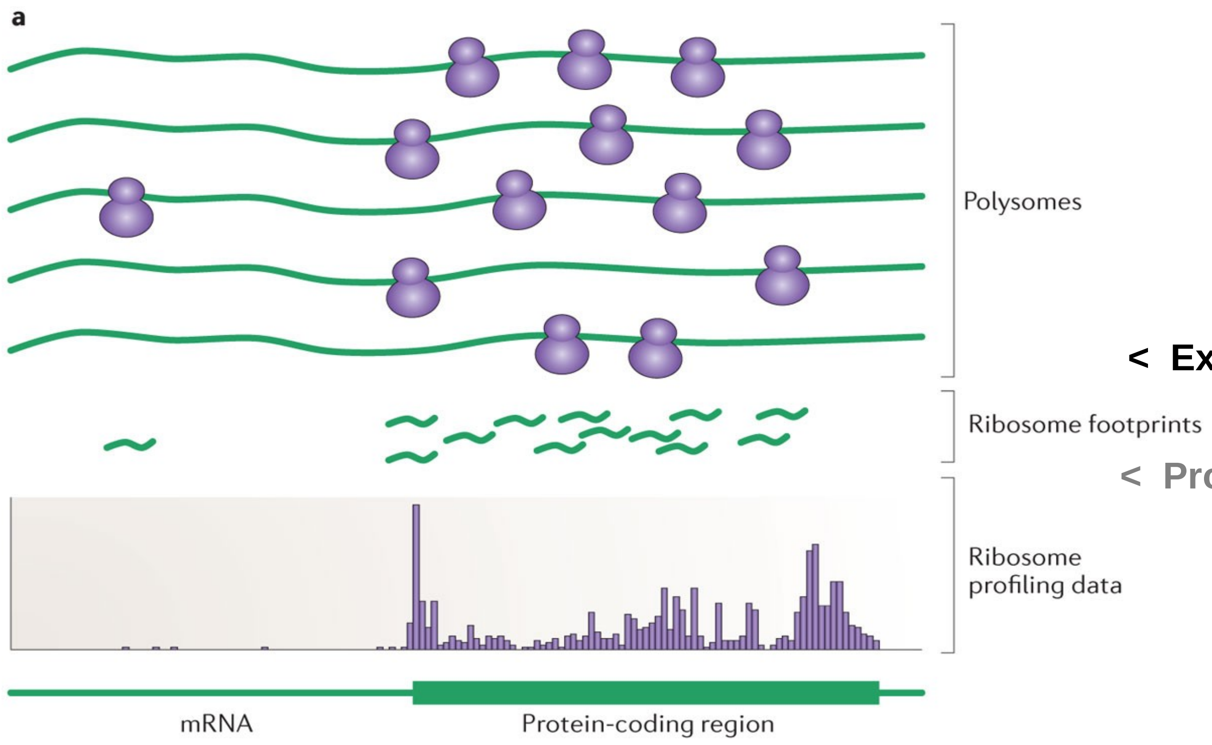
Outline

- Riboviz 101
- Ribosome Profiling Process
- Riboviz Workflow
- Analyis Code Refactoring
  - Example
  - Tips
  - Fails
- Call to action!
- Future Priorities

Riboviz 101

- **Riboviz processes & analyses ribosome profiling data**
- Ribosome profiling data helps unlock details of **active translation**: *mechanics of translation, regulation methods, translational efficiency*
- Developing/improving riboviz = **more researcher time** for biological questions rather than tinkering with pipelines & bespoke analysis code...

Polysomes to Footprints

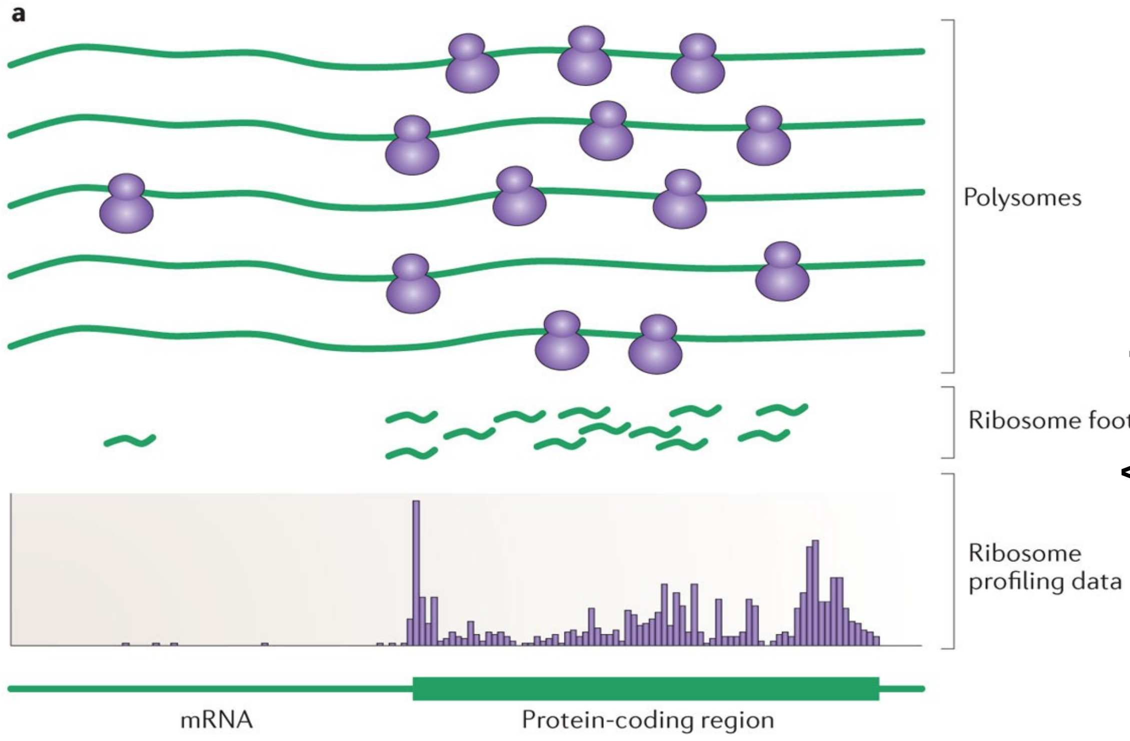


< Experimental Stages

< Processing & Analysis

Ingolia (2014). "Ribosome profiling: new views of translation, from single codons to genome scale". Nature Reviews. Genetics. 15 (3): 205–13. doi:10.1038/nrg3645

Footprints to Profiling Data



< Experimental Stages

< Processing & Analysis

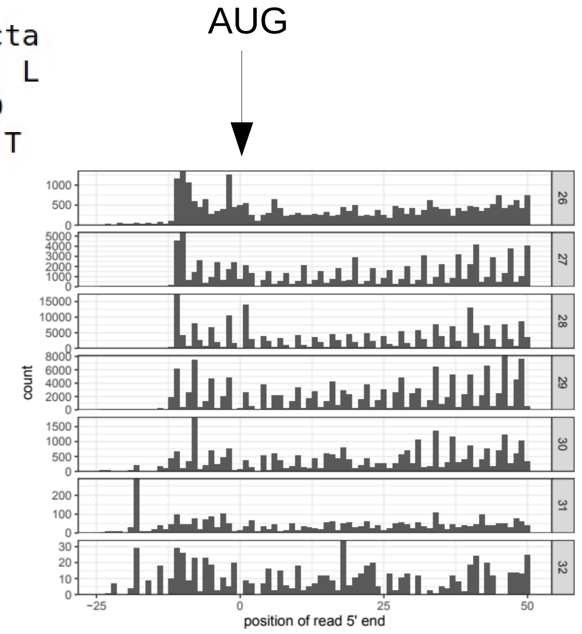
Ingolia (2014). "Ribosome profiling: new views of translation, from single codons to genome scale". Nature Reviews. Genetics. 15 (3): 205–13. doi:10.1038/nrg3645

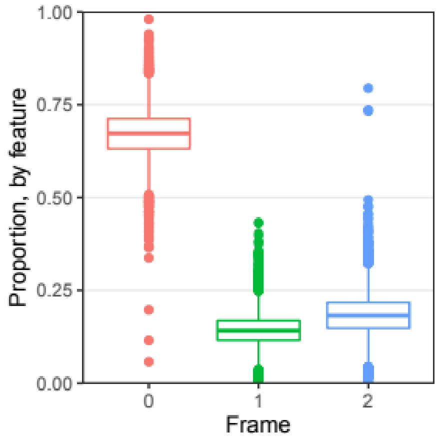
Analysis of Ribosome Profiling Data

- Looking for **3-NT periodicity**: ribosomes moving along transcript 1 codon at a time
- Most reads map to **coding regions** (98.8% in Ingolia et al 2009)
- Reasonable read-lengths (e.g. know should look for appx **~28-30NT**)
- Looking for most reads to be in **one frame**

Ribosome Data & Reading Frame

```
reading frame:            123
                          |||
                          acttacccgggacta
first reading frame         T  Y  P  G  L
second reading frame         L  T  R  D
third reading frame           L  P  G  T
```





G-Sc_2014: WTnone

Reading Frame: https://www.ncbi.nlm.nih.gov/Class/MLACourse/Original8Hour/Genetics/readingframe.html
Data: Guydosh & Green (2014). “Dom34 rescues ribosomes in 3' untranslated regions.” Cell. 156 (5): 950-62. doi: 10.1016/j.cell.2014.02.006.

Footprints to Ribosome Profiling Data

- **Processing**: *lots of steps*
  - Removing adapter sequences
  - Remove UMIs (Unique Molecular Identifiers) & barcodes if present
  - Demultiplex / Deduplicate reads if required
  - Need to filter out contaminant reads
  - Align reads to transcriptome

- **Analysis**: *more steps*
  - Analyse & quantify data:
  - Create outputs (including for quality-control, further analysis)

Riboviz Workflow: Inputs

Organism Specific

Transcript Sequences
.fasta

Genome / Transcriptome Features
.gff

Contaminant Sequences (rRNA)
.fasta

(Additional Organism-Specific Data)

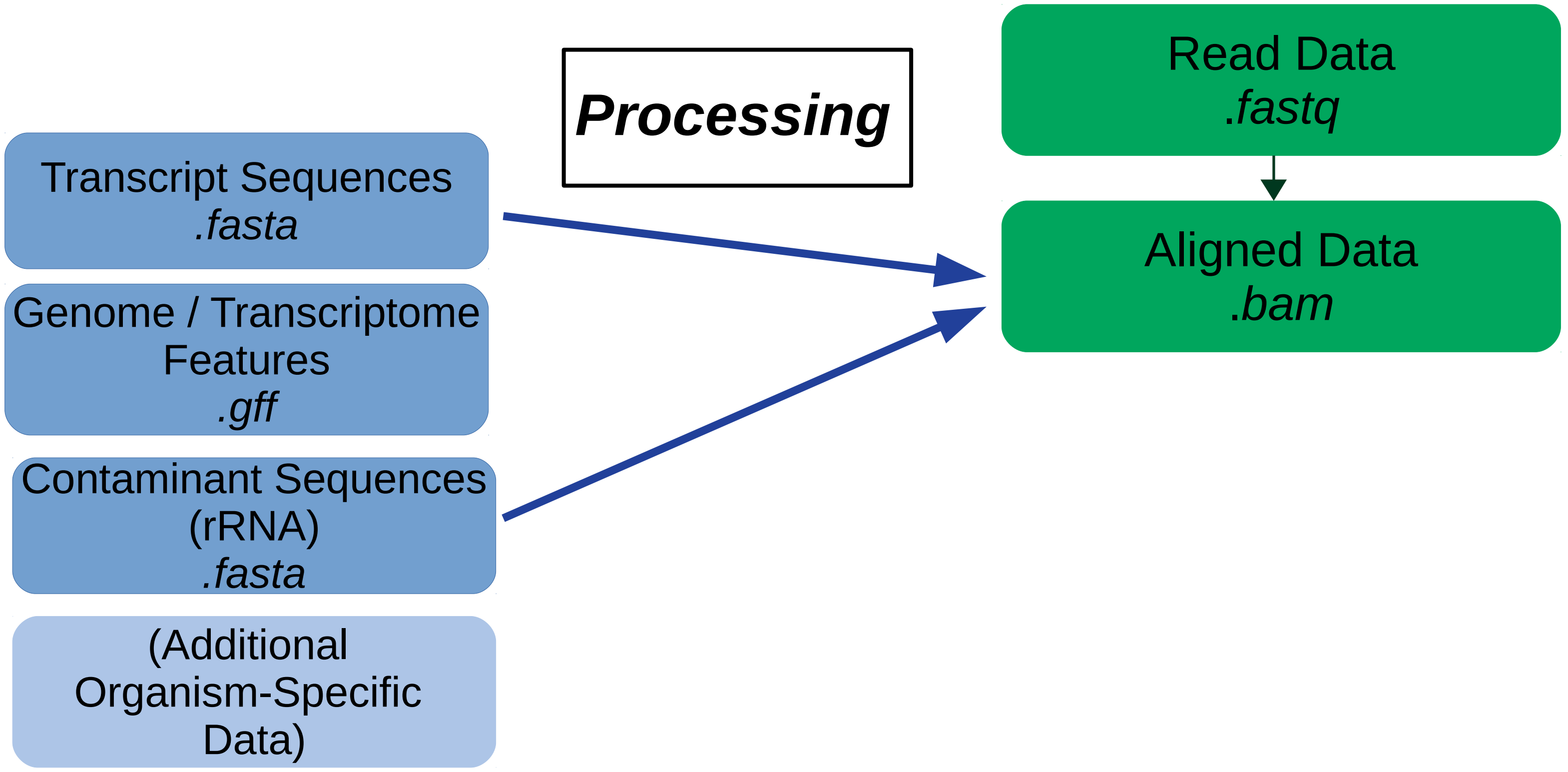
Sample Specific

Read Data
.fastq

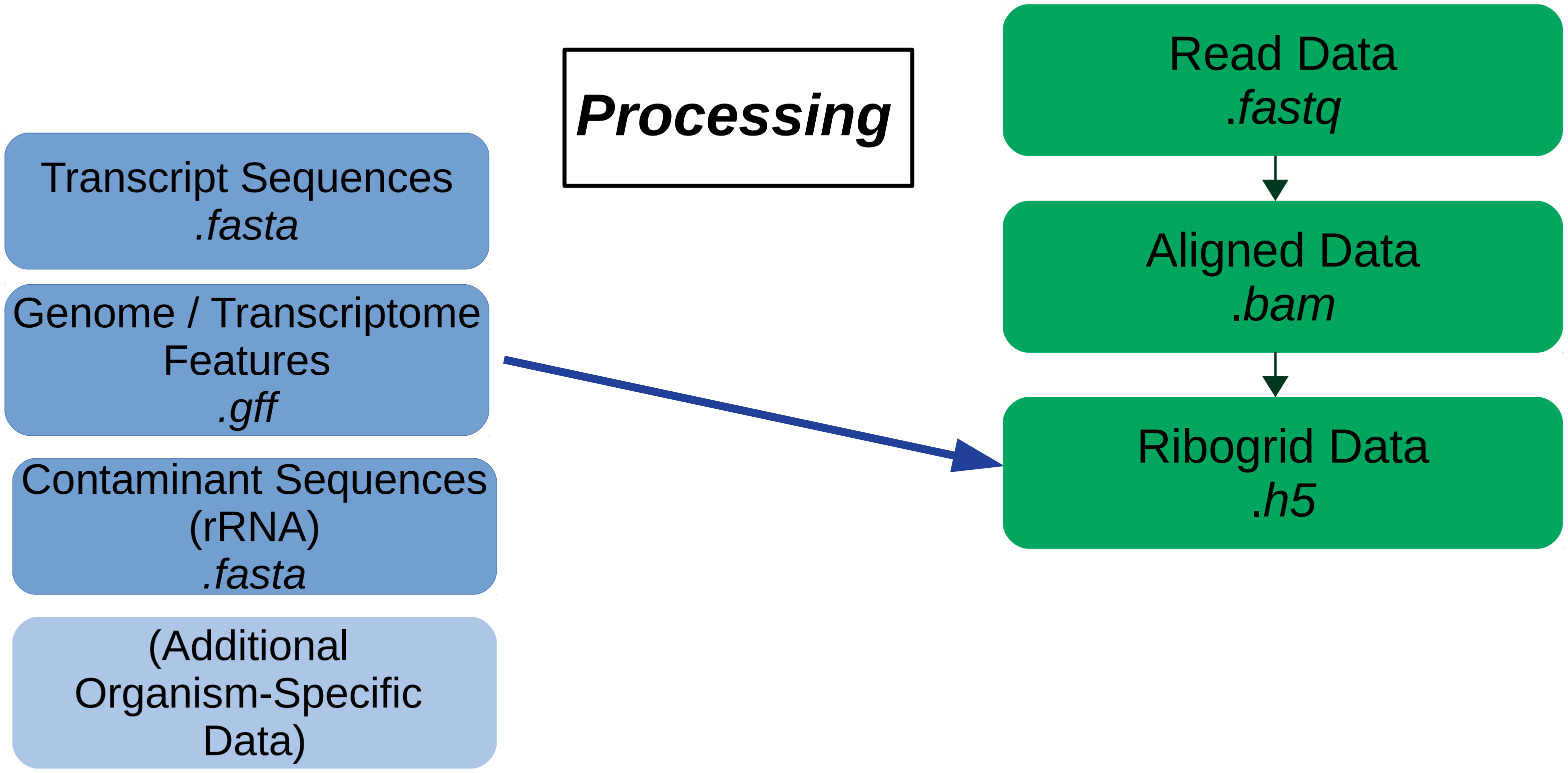
Configuration File
.yaml

Configuration File lists all files & parameters needed to run RiboViz

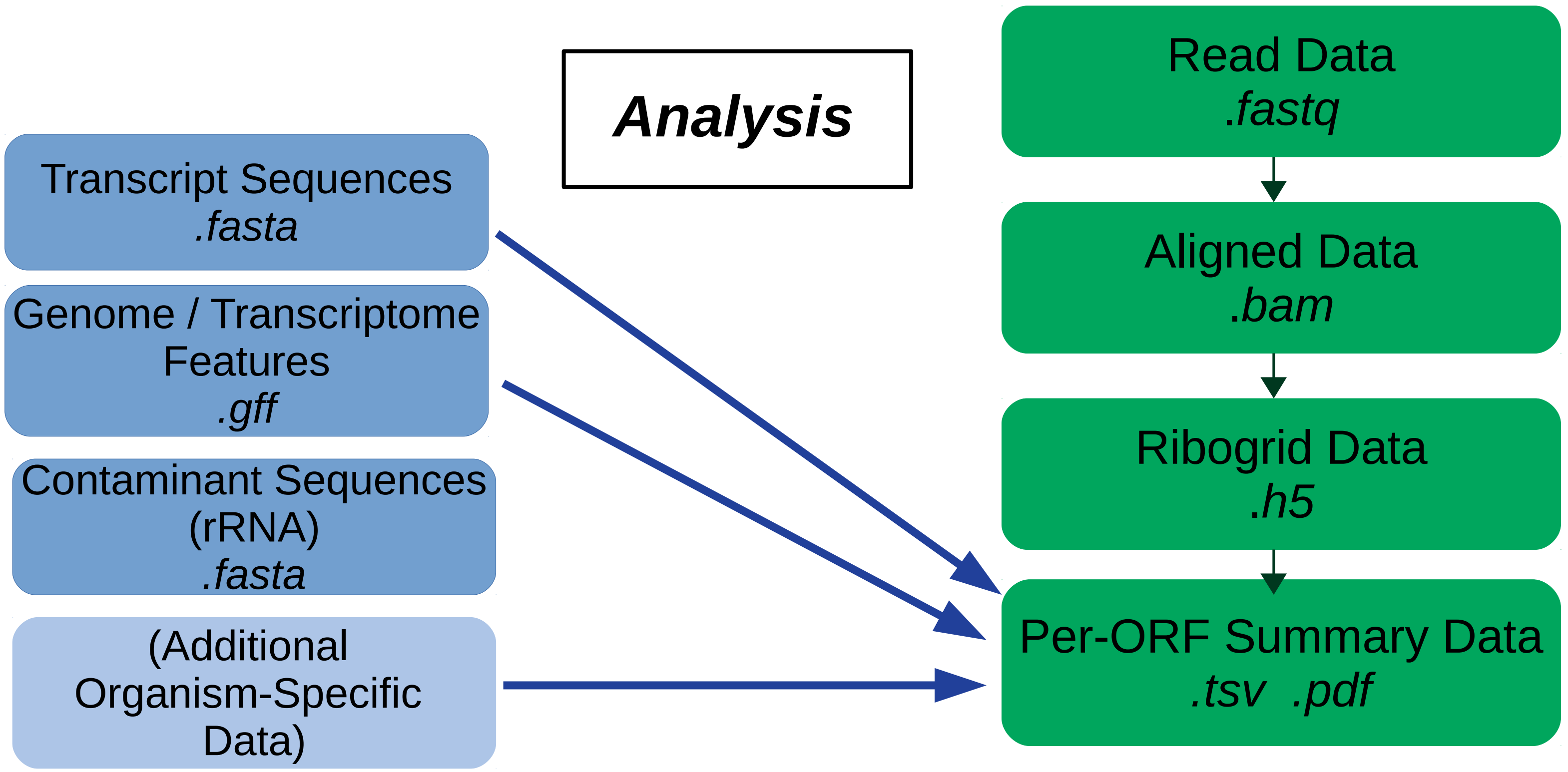
Riboviz Workflow


Processing
Read Data
.fastq
Transcript Sequences
.fasta
Aligned Data
.bam
Genome / Transcriptome Features
.gff
Contaminant Sequences (rRNA)
.fasta
(Additional Organism-Specific Data)

Riboviz Workflow


Processing
Read Data
.fastq
Aligned Data
.bam
Ribogrid Data
.h5
Transcript Sequences
.fasta
Genome / Transcriptome Features
.gff
Contaminant Sequences (rRNA)
.fasta
(Additional Organism-Specific Data)

Riboviz Workflow

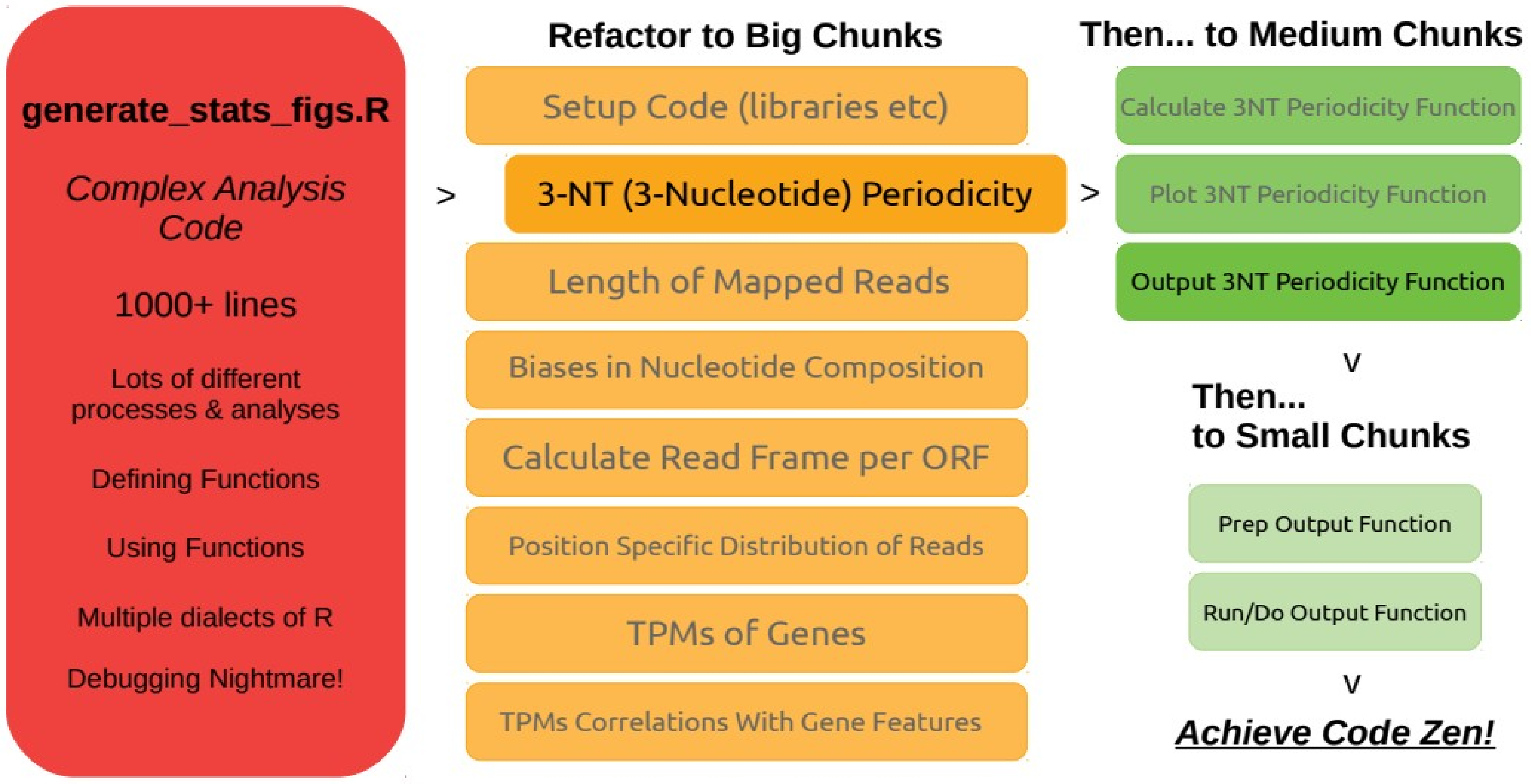

Analysis
Transcript Sequences *.fasta*
Genome / Transcriptome Features *.gff*
Contaminant Sequences (rRNA) *.fasta*
(Additional Organism-Specific Data)
Read Data *.fastq*
Aligned Data *.bam*
Ribogrid Data *.h5*
Per-ORF Summary Data *.tsv .pdf*

Analysis

generate_stats_figs.R:

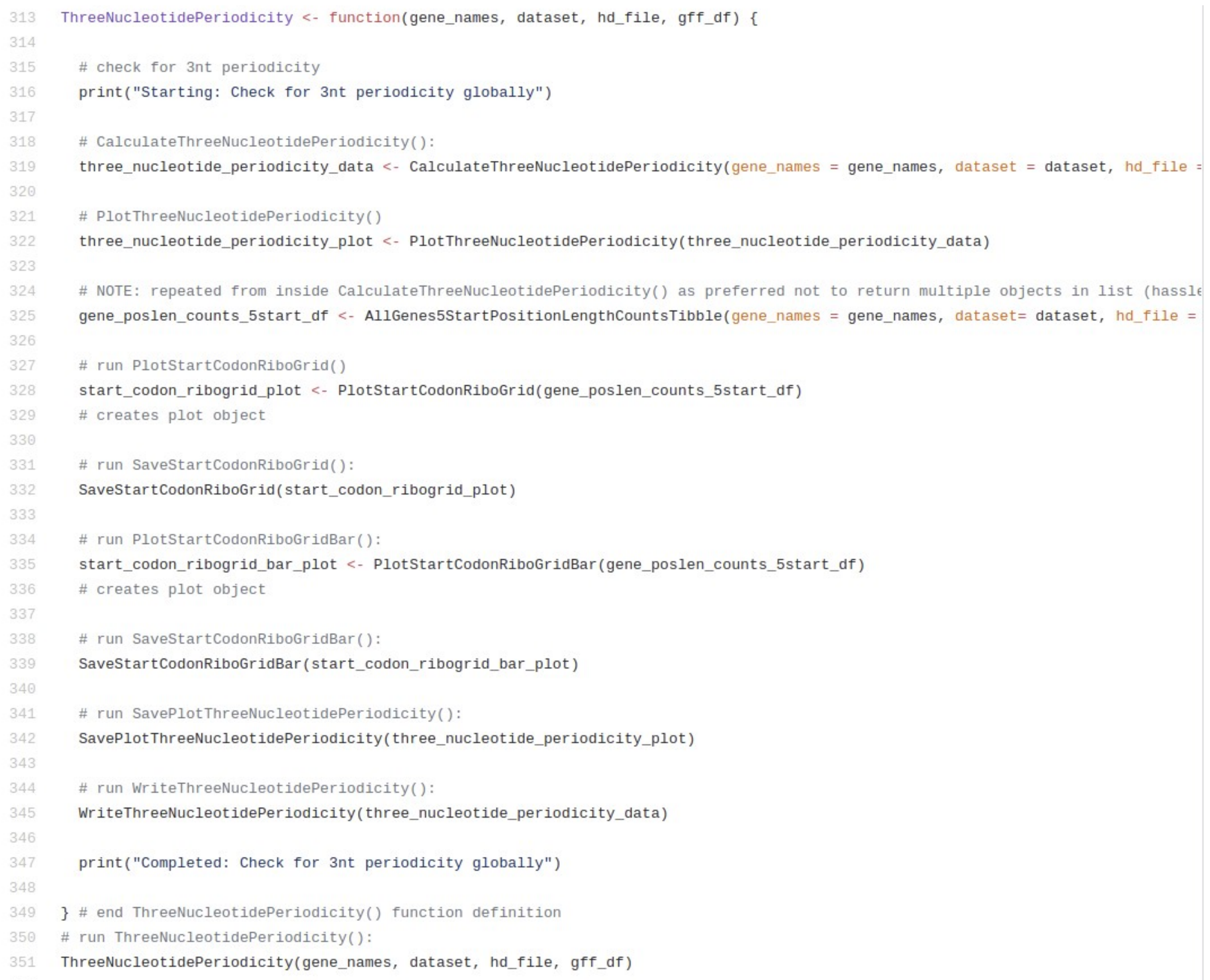
Generates summary statistics, analysis plots & quality control plots

Refactoring Analysis Code

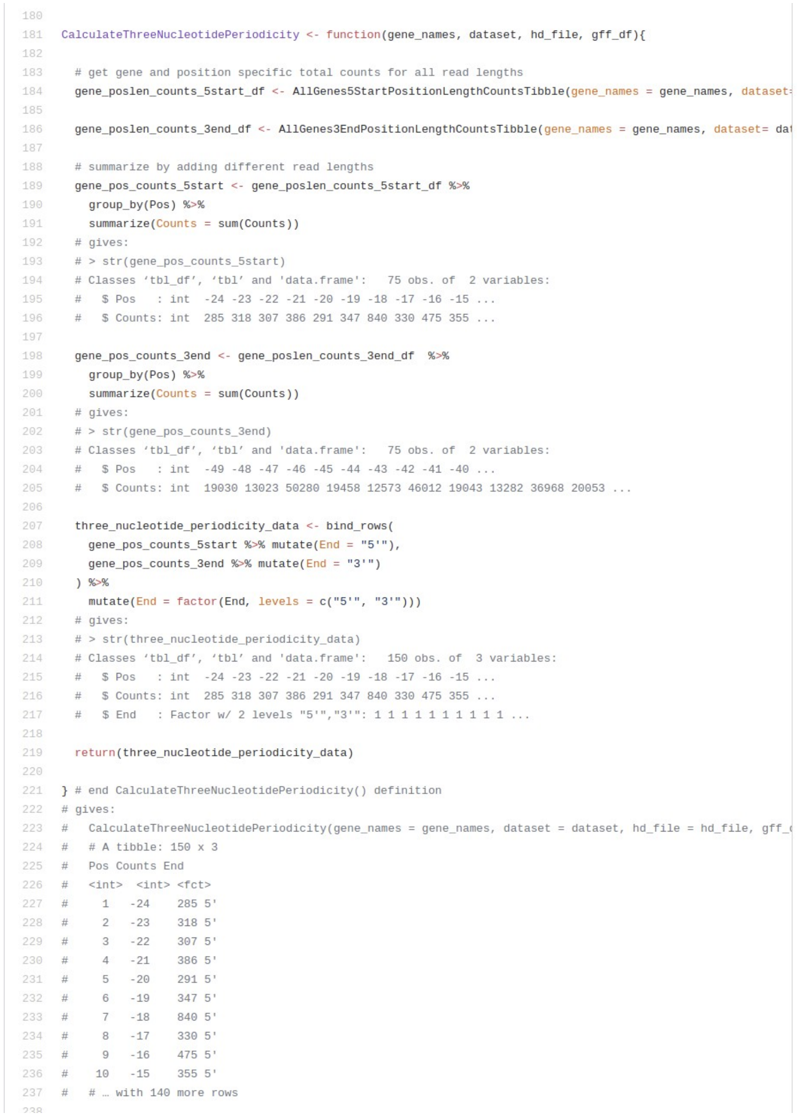

generate_stats_figs.R
Complex Analysis Code
1000+ lines
Lots of different processes & analyses
Defining Functions
Using Functions
Multiple dialects of R
Debugging Nightmare!
>
Refactor to Big Chunks
Setup Code (libraries etc)
3-NT (3-Nucleotide) Periodicity
Length of Mapped Reads
Biases in Nucleotide Composition
Calculate Read Frame per ORF
Position Specific Distribution of Reads
TPMs of Genes
TPMs Correlations With Gene Features
>
Then... to Medium Chunks
Calculate 3NT Periodicity Function
Plot 3NT Periodicity Function
Output 3NT Periodicity Function
v
Then... to Small Chunks
Prep Output Function
Run/Do Output Function
v
Achieve Code Zen!

Anatomy of a Code Chunk...

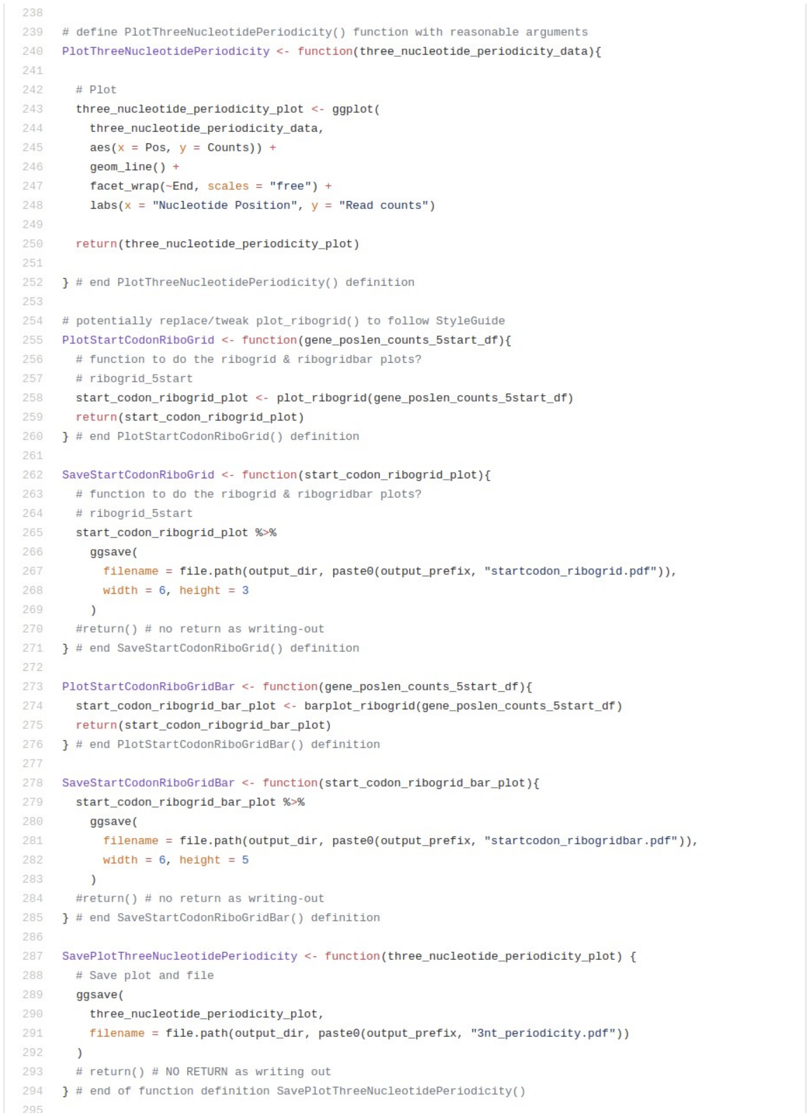
- 3NT Periodicity Big Code Chunk:
  - Calculate! Get start positions & read counts at each position for each gene from the .h5 file, calculate periodicity from this matrix of positions & counts
  - Plots! {ggplot2}
  - Save plots as .pdf
  - Write information out as .tsv (includes provenance info)



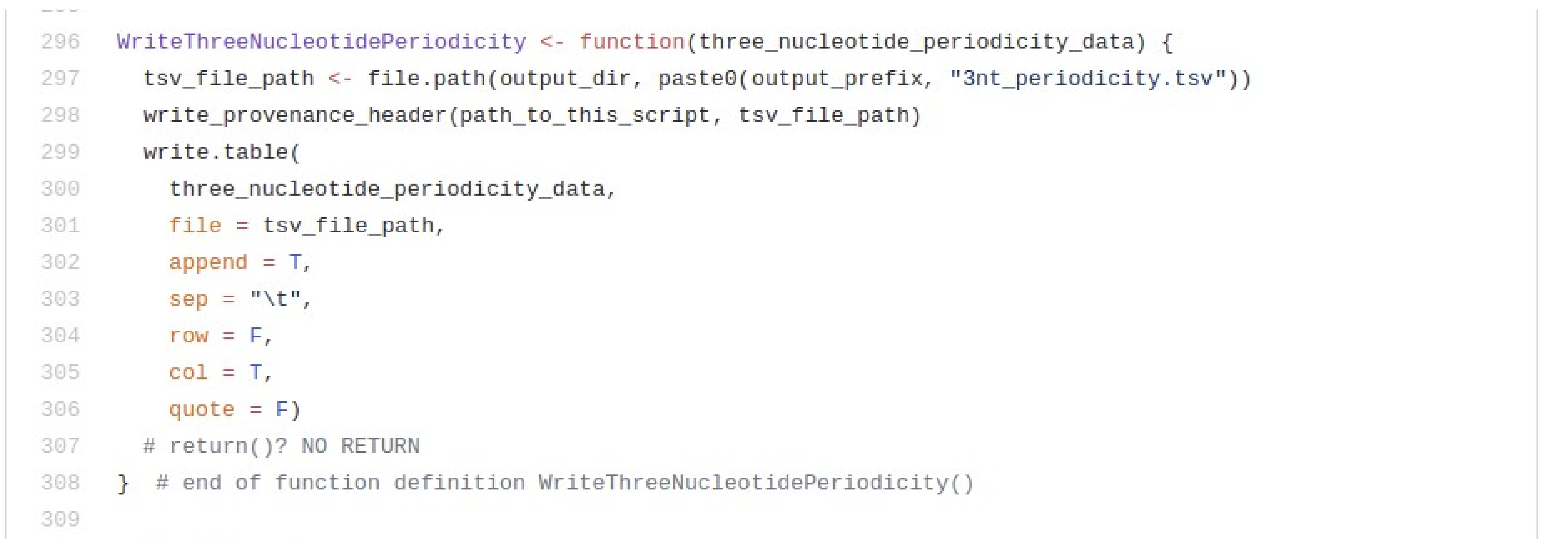
```
ThreeNucleotidePeriodicity <- function(gene_names, dataset, hd_file, gff_df) {

  # check for 3nt periodicity
  print("Starting: Check for 3nt periodicity globally")

  # CalculateThreeNucleotidePeriodicity():
  three_nucleotide_periodicity_data <- CalculateThreeNucleotidePeriodicity(gene_names = gene_names, dataset = dataset, hd_file =

  # PlotThreeNucleotidePeriodicity()
  three_nucleotide_periodicity_plot <- PlotThreeNucleotidePeriodicity(three_nucleotide_periodicity_data)

  # NOTE: repeated from inside CalculateThreeNucleotidePeriodicity() as preferred not to return multiple objects in list (hassle
  gene_poslen_counts_5start_df <- AllGenes5StartPositionLengthCountsTibble(gene_names = gene_names, dataset= dataset, hd_file =

  # run PlotStartCodonRiboGrid()
  start_codon_ribogrid_plot <- PlotStartCodonRiboGrid(gene_poslen_counts_5start_df)
  # creates plot object

  # run SaveStartCodonRiboGrid():
  SaveStartCodonRiboGrid(start_codon_ribogrid_plot)

  # run PlotStartCodonRiboGridBar():
  start_codon_ribogrid_bar_plot <- PlotStartCodonRiboGridBar(gene_poslen_counts_5start_df)
  # creates plot object

  # run SaveStartCodonRiboGridBar():
  SaveStartCodonRiboGridBar(start_codon_ribogrid_bar_plot)

  # run SavePlotThreeNucleotidePeriodicity():
  SavePlotThreeNucleotidePeriodicity(three_nucleotide_periodicity_plot)

  # run WriteThreeNucleotidePeriodicity():
  WriteThreeNucleotidePeriodicity(three_nucleotide_periodicity_data)

  print("Completed: Check for 3nt periodicity globally")

} # end ThreeNucleotidePeriodicity() function definition
# run ThreeNucleotidePeriodicity():
ThreeNucleotidePeriodicity(gene_names, dataset, hd_file, gff_df)
```



```r

CalculateThreeNucleotidePeriodicity <- function(gene_names, dataset, hd_file, gff_df){

  # get gene and position specific total counts for all read lengths
  gene_poslen_counts_5start_df <- AllGenes5StartPositionLengthCountsTibble(gene_names = gene_names, dataset=

  gene_poslen_counts_3end_df <- AllGenes3EndPositionLengthCountsTibble(gene_names = gene_names, dataset= da

  # summarize by adding different read lengths
  gene_pos_counts_5start <- gene_poslen_counts_5start_df %>%
    group_by(Pos) %>%
    summarize(Counts = sum(Counts))
  # gives:
  # > str(gene_pos_counts_5start)
  # Classes ‘tbl_df’, ‘tbl’ and 'data.frame':   75 obs. of  2 variables:
  #   $ Pos   : int  -24 -23 -22 -21 -20 -19 -18 -17 -16 -15 ...
  #   $ Counts: int  285 318 307 386 291 347 840 330 475 355 ...

  gene_pos_counts_3end <- gene_poslen_counts_3end_df  %>%
    group_by(Pos) %>%
    summarize(Counts = sum(Counts))
  # gives:
  # > str(gene_pos_counts_3end)
  # Classes ‘tbl_df’, ‘tbl’ and 'data.frame':   75 obs. of  2 variables:
  #   $ Pos   : int  -49 -48 -47 -46 -45 -44 -43 -42 -41 -40 ...
  #   $ Counts: int  19030 13023 50280 19458 12573 46012 19043 13282 36968 20053 ...

  three_nucleotide_periodicity_data <- bind_rows(
    gene_pos_counts_5start %>% mutate(End = "5'"),
    gene_pos_counts_3end %>% mutate(End = "3'")
  ) %>%
    mutate(End = factor(End, levels = c("5'", "3'")))
  # gives:
  # > str(three_nucleotide_periodicity_data)
  # Classes ‘tbl_df’, ‘tbl’ and 'data.frame':   150 obs. of  3 variables:
  #   $ Pos   : int  -24 -23 -22 -21 -20 -19 -18 -17 -16 -15 ...
  #   $ Counts: int  285 318 307 386 291 347 840 330 475 355 ...
  #   $ End   : Factor w/ 2 levels "5'","3'": 1 1 1 1 1 1 1 1 1 1 ...

  return(three_nucleotide_periodicity_data)

} # end CalculateThreeNucleotidePeriodicity() definition
# gives:
#   CalculateThreeNucleotidePeriodicity(gene_names = gene_names, dataset = dataset, hd_file = hd_file, gff_
#   # A tibble: 150 x 3
#   Pos Counts End
#   <int>  <int> <fct>
#     1   -24    285 5'
#     2   -23    318 5'
#     3   -22    307 5'
#     4   -21    386 5'
#     5   -20    291 5'
#     6   -19    347 5'
#     7   -18    840 5'
#     8   -17    330 5'
#     9   -16    475 5'
#    10   -15    355 5'
#   # … with 140 more rows

```



```r

# define PlotThreeNucleotidePeriodicity() function with reasonable arguments
PlotThreeNucleotidePeriodicity <- function(three_nucleotide_periodicity_data){

  # Plot
  three_nucleotide_periodicity_plot <- ggplot(
    three_nucleotide_periodicity_data,
    aes(x = Pos, y = Counts)) +
    geom_line() +
    facet_wrap(~End, scales = "free") +
    labs(x = "Nucleotide Position", y = "Read counts")

  return(three_nucleotide_periodicity_plot)

} # end PlotThreeNucleotidePeriodicity() definition

# potentially replace/tweak plot_ribogrid() to follow StyleGuide
PlotStartCodonRiboGrid <- function(gene_poslen_counts_5start_df){
  # function to do the ribogrid & ribogridbar plots?
  # ribogrid_5start
  start_codon_ribogrid_plot <- plot_ribogrid(gene_poslen_counts_5start_df)
  return(start_codon_ribogrid_plot)
} # end PlotStartCodonRiboGrid() definition

SaveStartCodonRiboGrid <- function(start_codon_ribogrid_plot){
  # function to do the ribogrid & ribogridbar plots?
  # ribogrid_5start
  start_codon_ribogrid_plot %>%
    ggsave(
      filename = file.path(output_dir, paste0(output_prefix, "startcodon_ribogrid.pdf")),
      width = 6, height = 3
    )
  #return() # no return as writing-out
} # end SaveStartCodonRiboGrid() definition

PlotStartCodonRiboGridBar <- function(gene_poslen_counts_5start_df){
  start_codon_ribogrid_bar_plot <- barplot_ribogrid(gene_poslen_counts_5start_df)
  return(start_codon_ribogrid_bar_plot)
} # end PlotStartCodonRiboGridBar() definition

SaveStartCodonRiboGridBar <- function(start_codon_ribogrid_bar_plot){
  start_codon_ribogrid_bar_plot %>%
    ggsave(
      filename = file.path(output_dir, paste0(output_prefix, "startcodon_ribogridbar.pdf")),
      width = 6, height = 5
    )
  #return() # no return as writing-out
} # end SaveStartCodonRiboGridBar() definition

SavePlotThreeNucleotidePeriodicity <- function(three_nucleotide_periodicity_plot) {
  # Save plot and file
  ggsave(
    three_nucleotide_periodicity_plot,
    filename = file.path(output_dir, paste0(output_prefix, "3nt_periodicity.pdf"))
  )
  # return() # NO RETURN as writing out
} # end of function definition SavePlotThreeNucleotidePeriodicity()

```



```r
WriteThreeNucleotidePeriodicity <- function(three_nucleotide_periodicity_data) {
  tsv_file_path <- file.path(output_dir, paste0(output_prefix, "3nt_periodicity.tsv"))
  write_provenance_header(path_to_this_script, tsv_file_path)
  write.table(
    three_nucleotide_periodicity_data,
    file = tsv_file_path,
    append = T,
    sep = "\t",
    row = F,
    col = T,
    quote = F)
  # return()? NO RETURN
}  # end of function definition WriteThreeNucleotidePeriodicity()

```

Refactoring Tips

- Break it down into **chunks**!
- **Regression tests** are your friend, but fails are not always a failure...
- Develop functions in your main script & then move into **separate functions script**
- Keep it specific: **issue-focussed working**

… Decide how you'll know you're finished...

Refactoring LFMFs

- **Issue proliferation**… Getting lost amongst different issues & losing sight of the main goals
- Schroedinger’s **debugging** issues!
- **Package::function()** is very useful!
- … Remember to **tell collaborators** about & document the new packages you add to the code to solve problems...

“Putting the YOU into USER”

- **User Testing** & User Interviews – *Sam!*
- Existing **Documentation Feedback** – *Siyin* & others already helping with this :)
- Leave the **Code Docs** to the Robots? – {roxygen2} – hoping to chat to *Edward* about this!
- **General Comparisons** - what features of other software doing similar things do you find useful/unhelpful? - <u>**Everyone**</u>!!!

“Testable, Reliable CompYOUtation”

- **Code Testing**:
  - Helping integrate functions from *Ania* & *Siyin’s* projects: want to have good testable code…
  - **{testthat} R package** – would be great to chat with *Xuejia* on how this works
- **More data**!
  - This lab: *Rosey?*
  - Others? Let’s talk :)

Priorities

- **Tests** for the analysis code
- generate_stats_figs.R **finishing touches:** Documentation / better output format / styling
- **New datasets** run & added to example datasets repository
- Integrate **new functions** from Ania & Siyin’s project work
- **User** focus
- Q: How does riboviz compare with **other tools**
- Q: Do we have the right **statistics** for diagnostics?

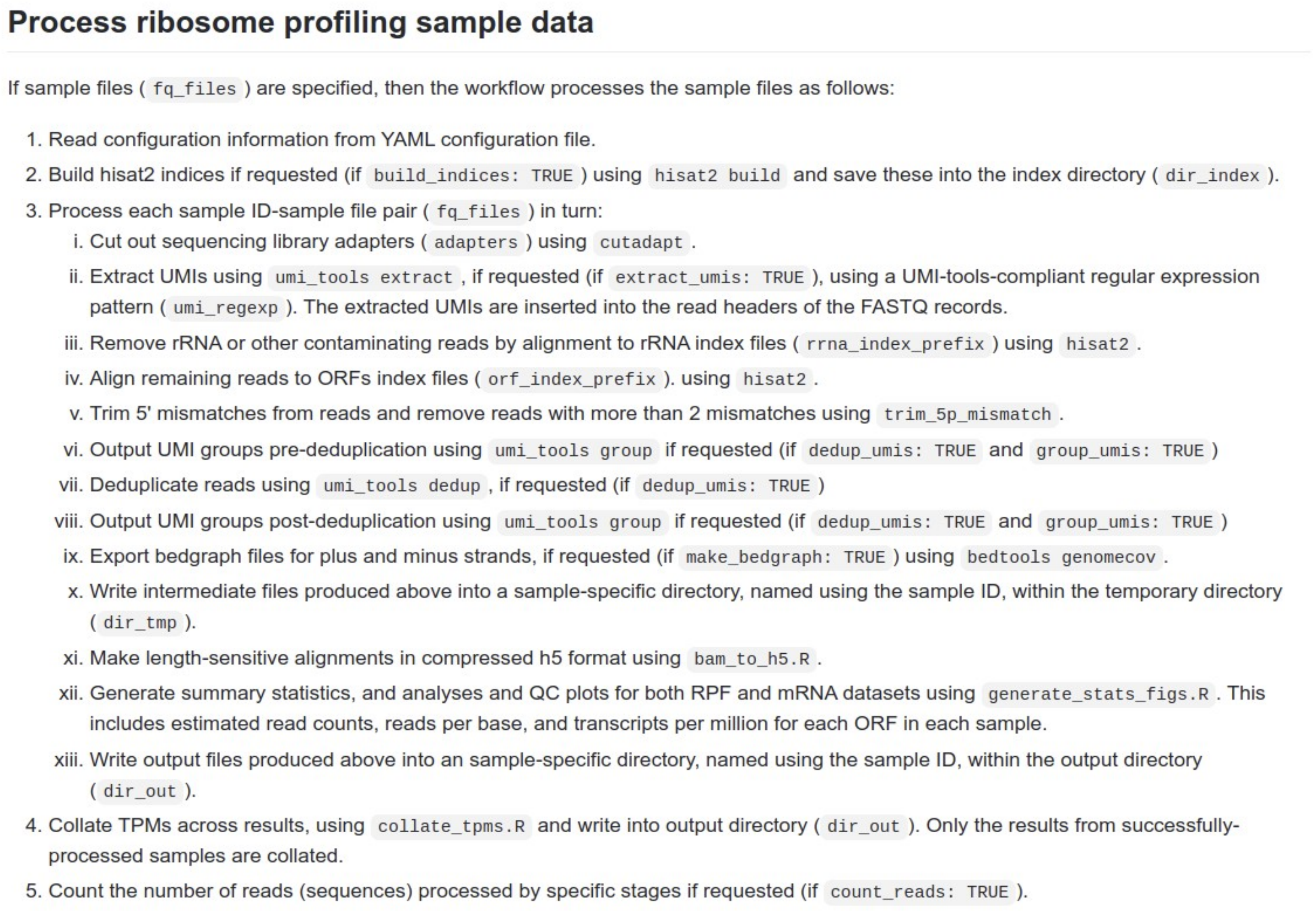
Thanks / Acknowledgements

- BBSRC-NSF funded project
- Collaborative project:
  - Edward Wallace: *The Wallace Lab,* The University of Edinburgh.

    + Siyin Xue, Ania Kurowska
  - Premal Shah, John Favate, Tongji Xing: *The Shah Lab,* Rutgers University.
  - Liana Lareau, Amanda Mok: *The Lareau Lab,* University of California, Berkeley.
  - Kostas Kavousannakis, Mike Jackson: *EPCC,* The University of Edinburgh.
  - Oana Carja, Joshua Plotkin: The University of Pennsylvania

Questions?

EXTRA SLIDES



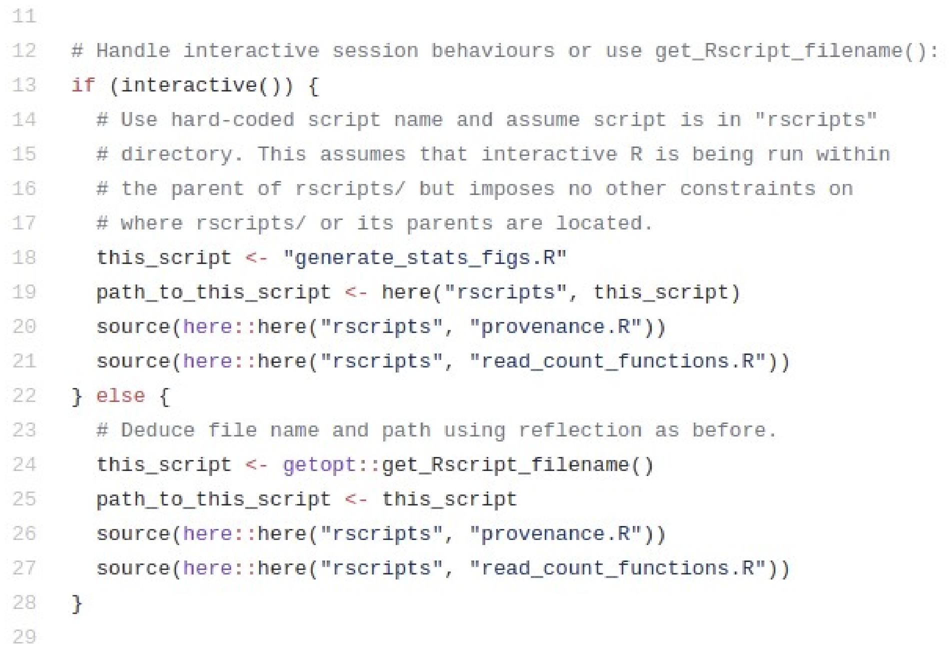
Process ribosome profiling sample data

If sample files (`fq_files`) are specified, then the workflow processes the sample files as follows:

1. Read configuration information from YAML configuration file.
2. Build hisat2 indices if requested (if `build_indices: TRUE`) using `hisat2 build` and save these into the index directory (`dir_index`).
3. Process each sample ID-sample file pair (`fq_files`) in turn:
   i. Cut out sequencing library adapters (`adapters`) using `cutadapt` .
   ii. Extract UMIs using `umi_tools extract` , if requested (if `extract_umis: TRUE`), using a UMI-tools-compliant regular expression pattern (`umi_regexp`). The extracted UMIs are inserted into the read headers of the FASTQ records.
   iii. Remove rRNA or other contaminating reads by alignment to rRNA index files (`rrna_index_prefix`) using `hisat2` .
   iv. Align remaining reads to ORFs index files (`orf_index_prefix`). using `hisat2` .
   v. Trim 5' mismatches from reads and remove reads with more than 2 mismatches using `trim_5p_mismatch` .
   vi. Output UMI groups pre-deduplication using `umi_tools group` if requested (if `dedup_umis: TRUE` and `group_umis: TRUE`)
   vii. Deduplicate reads using `umi_tools dedup` , if requested (if `dedup_umis: TRUE`)
   viii. Output UMI groups post-deduplication using `umi_tools group` if requested (if `dedup_umis: TRUE` and `group_umis: TRUE`)
   ix. Export bedgraph files for plus and minus strands, if requested (if `make_bedgraph: TRUE`) using `bedtools genomecov` .
   x. Write intermediate files produced above into a sample-specific directory, named using the sample ID, within the temporary directory (`dir_tmp`).
   xi. Make length-sensitive alignments in compressed h5 format using `bam_to_h5.R` .
   xii. Generate summary statistics, and analyses and QC plots for both RPF and mRNA datasets using `generate_stats_figs.R` . This includes estimated read counts, reads per base, and transcripts per million for each ORF in each sample.
   xiii. Write output files produced above into an sample-specific directory, named using the sample ID, within the output directory (`dir_out`).
4. Collate TPMs across results, using `collate_tpms.R` and write into output directory (`dir_out`). Only the results from successfully-processed samples are collated.
5. Count the number of reads (sequences) processed by specific stages if requested (if `count_reads: TRUE`).

https://github.com/riboviz/riboviz/blob/develop/docs/user/prep-riboviz-operation.md

*



```r

# Handle interactive session behaviours or use get_Rscript_filename():
if (interactive()) {
  # Use hard-coded script name and assume script is in "rscripts"
  # directory. This assumes that interactive R is being run within
  # the parent of rscripts/ but imposes no other constraints on
  # where rscripts/ or its parents are located.
  this_script <- "generate_stats_figs.R"
  path_to_this_script <- here("rscripts", this_script)
  source(here::here("rscripts", "provenance.R"))
  source(here::here("rscripts", "read_count_functions.R"))
} else {
  # Deduce file name and path using reflection as before.
  this_script <- getopt::get_Rscript_filename()
  path_to_this_script <- this_script
  source(here::here("rscripts", "provenance.R"))
  source(here::here("rscripts", "read_count_functions.R"))
}

```

Fails are not ALWAYS a failure!

- <3 Regression tests!
- Which output is the **correct** output? How do we know?
- How to decide when to make a **new issue** or keep working on a problem?
- **Rollback**...
- Importance of **code testing**… & understandable code!

Updates! UX

- Users:
  - Building user base?
  - Needs / wants?
  - Oven-ready datasets*?

* https://github.com/riboviz/example-datasets

Updates! UX

- Support & Documentation:
  - Existing docs suitable?
  - Document outputs better?
  - Translation UK workshop?

Updates! Testing

- General Testing:
  - New feature development ongoing
  - Bug fixes
  - Code reviews happening
  - New datasets

Updates! Testing

- Methods Testing:
  - Regression tests
  - Expected outputs: simulated data.

First Things First: Setup

- Initial Setup Code:
  - load libraries
  - handle interactive session behaviours if required*
  - source provenance & functions scripts
  - load parameters passed in from main riboviz workflow
  - read in key files (.gff, .fa, .h5)