Refactoring Riboviz Analysis Code:

A Personal Journey

24 June 2020

Flic Anderson
The Wallace Lab, University of Edinburgh

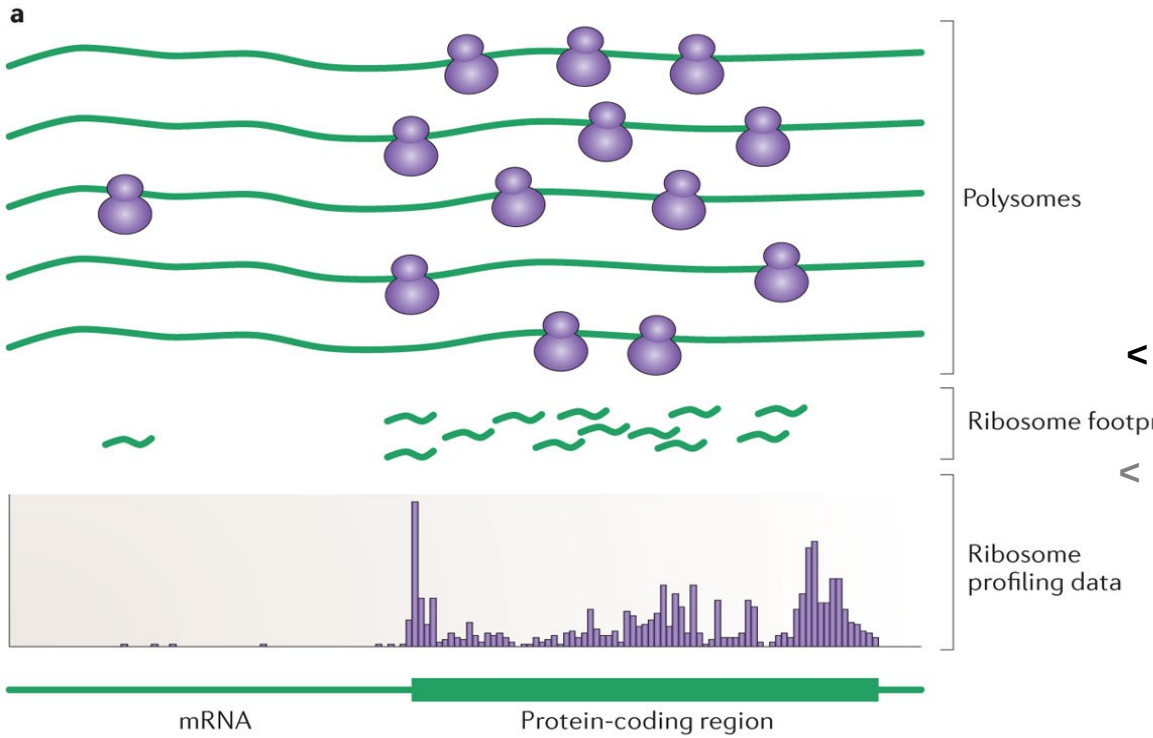
Outline

- 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

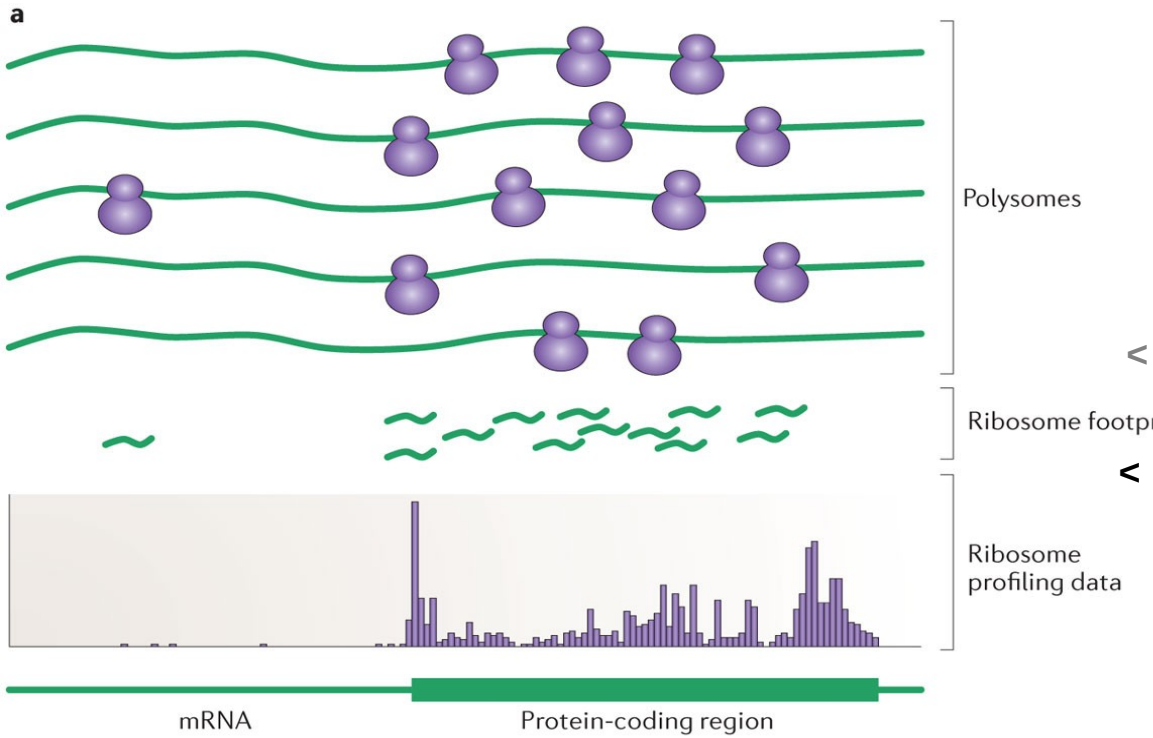


< Experimental Stages

< 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

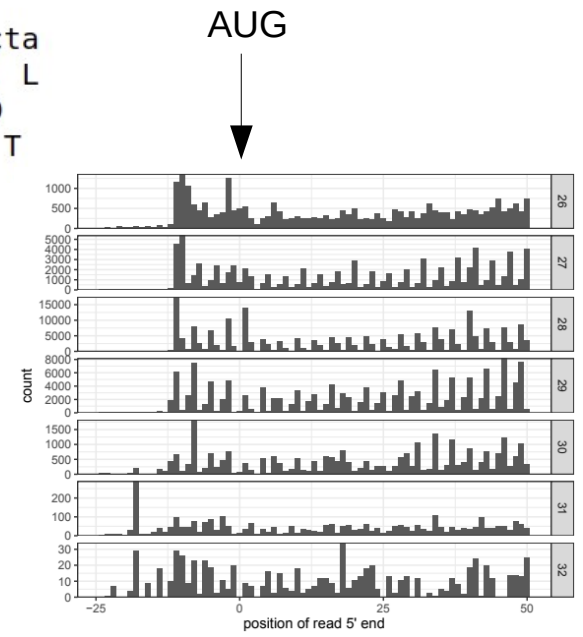
Ingolia (2014). "Ribosome profiling: new views of translation, from single codons to genome scale". Nature Reviews. Genetics. 15 (3): 205–13. doi:10.1038/nrg3645

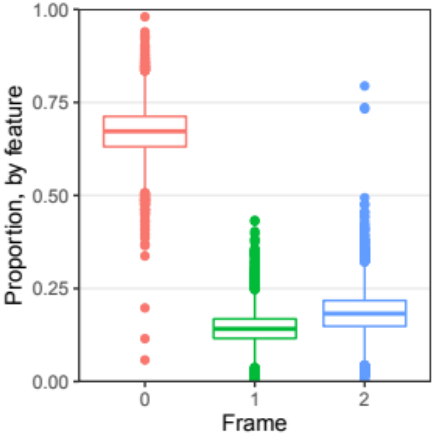
Analysis of Ribosome Profiling Data

- 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)

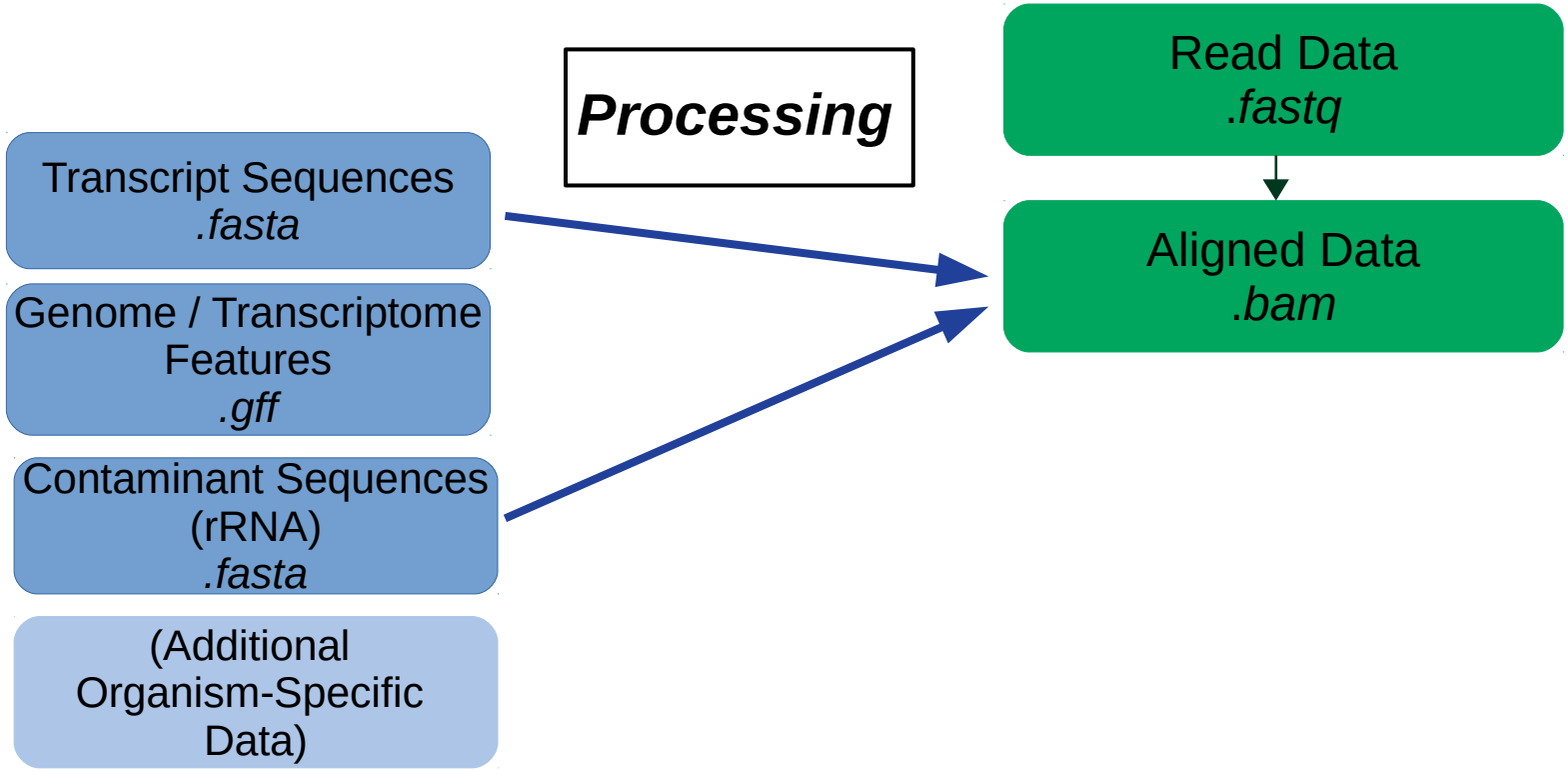
Sample Specific

Read Data
.fastq

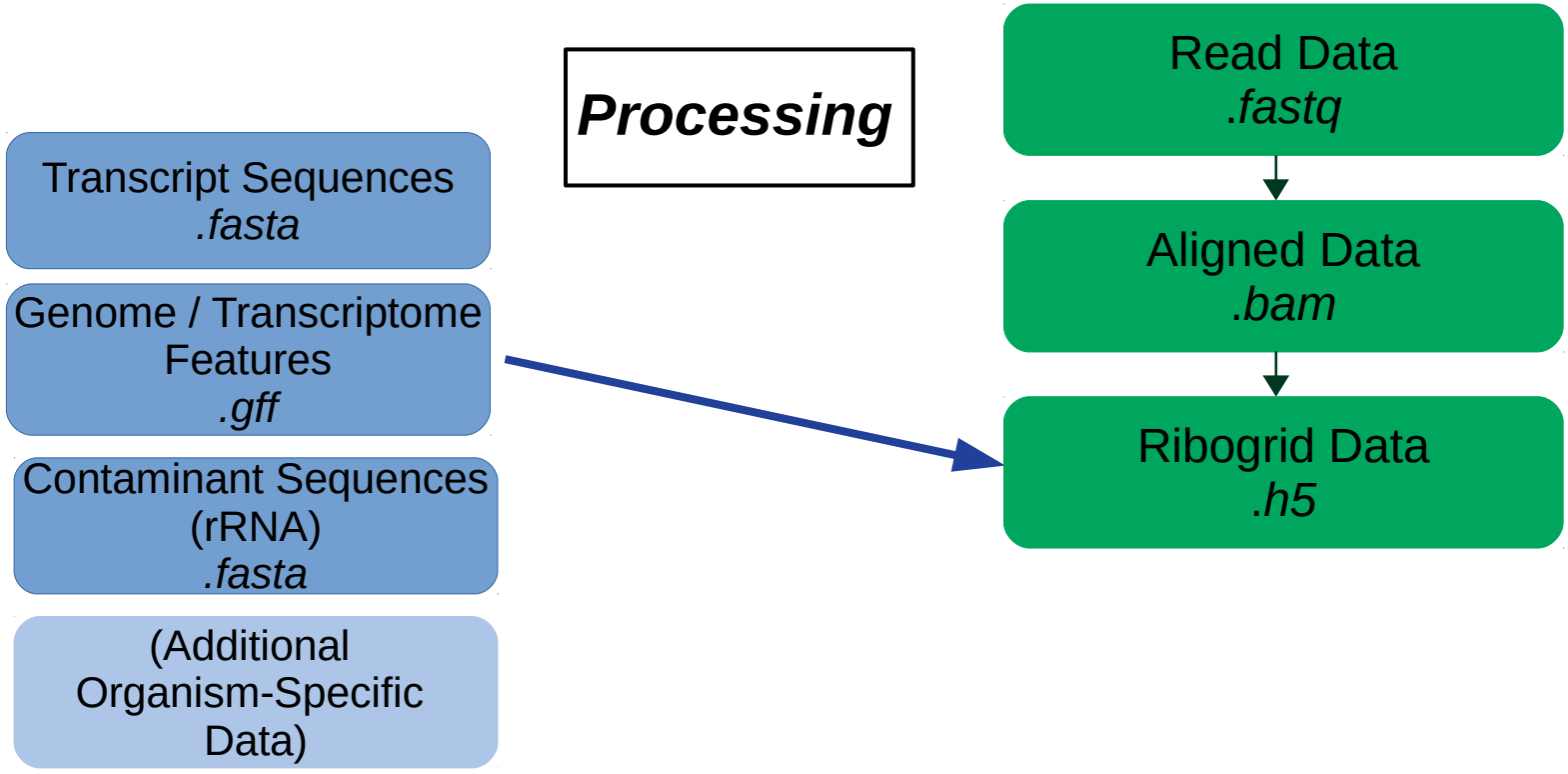
Configuration File
.yaml

Configuration File lists all files & parameters needed to run RiboViz

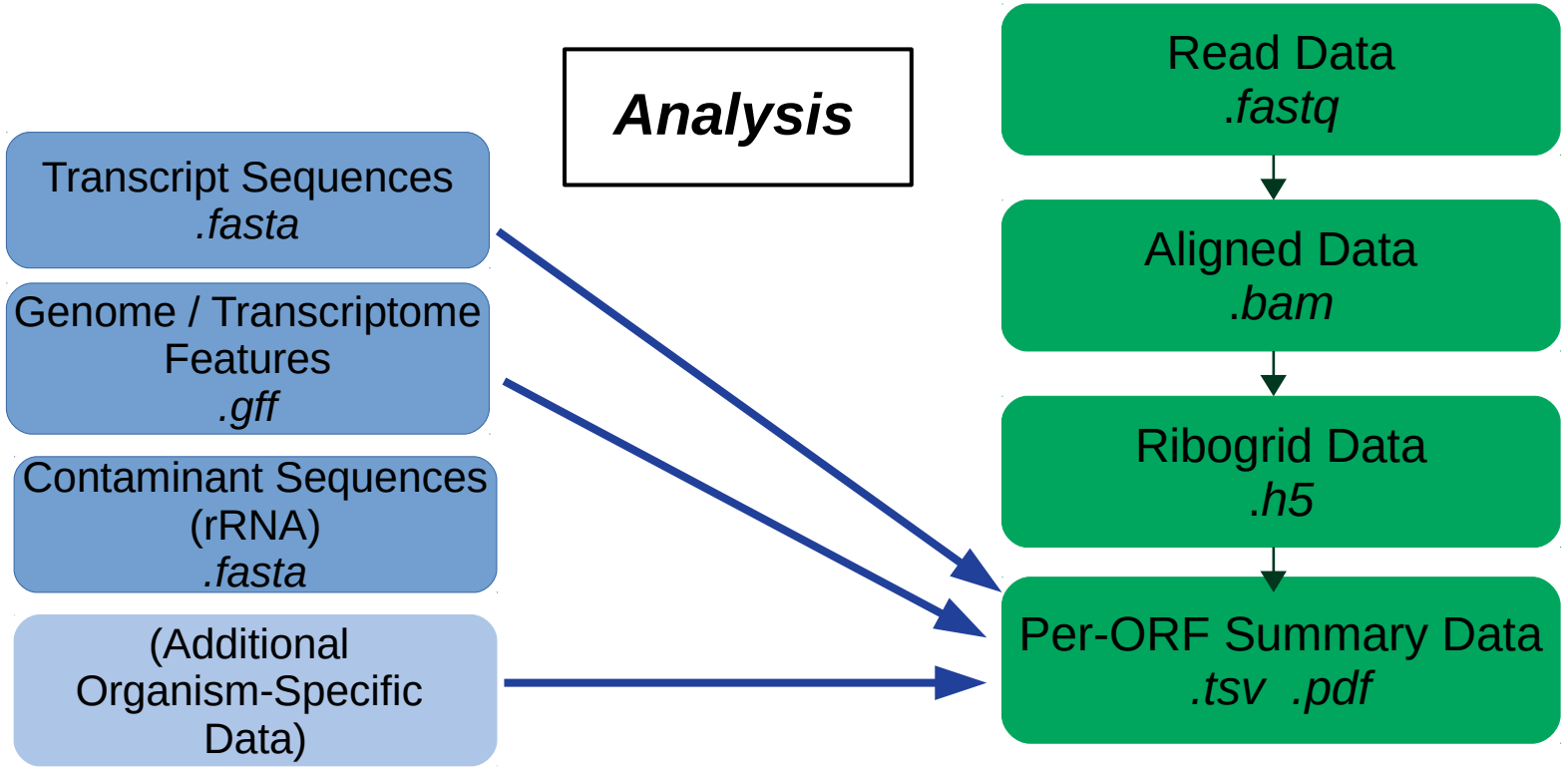
Riboviz Workflow



Riboviz Workflow



Riboviz Workflow

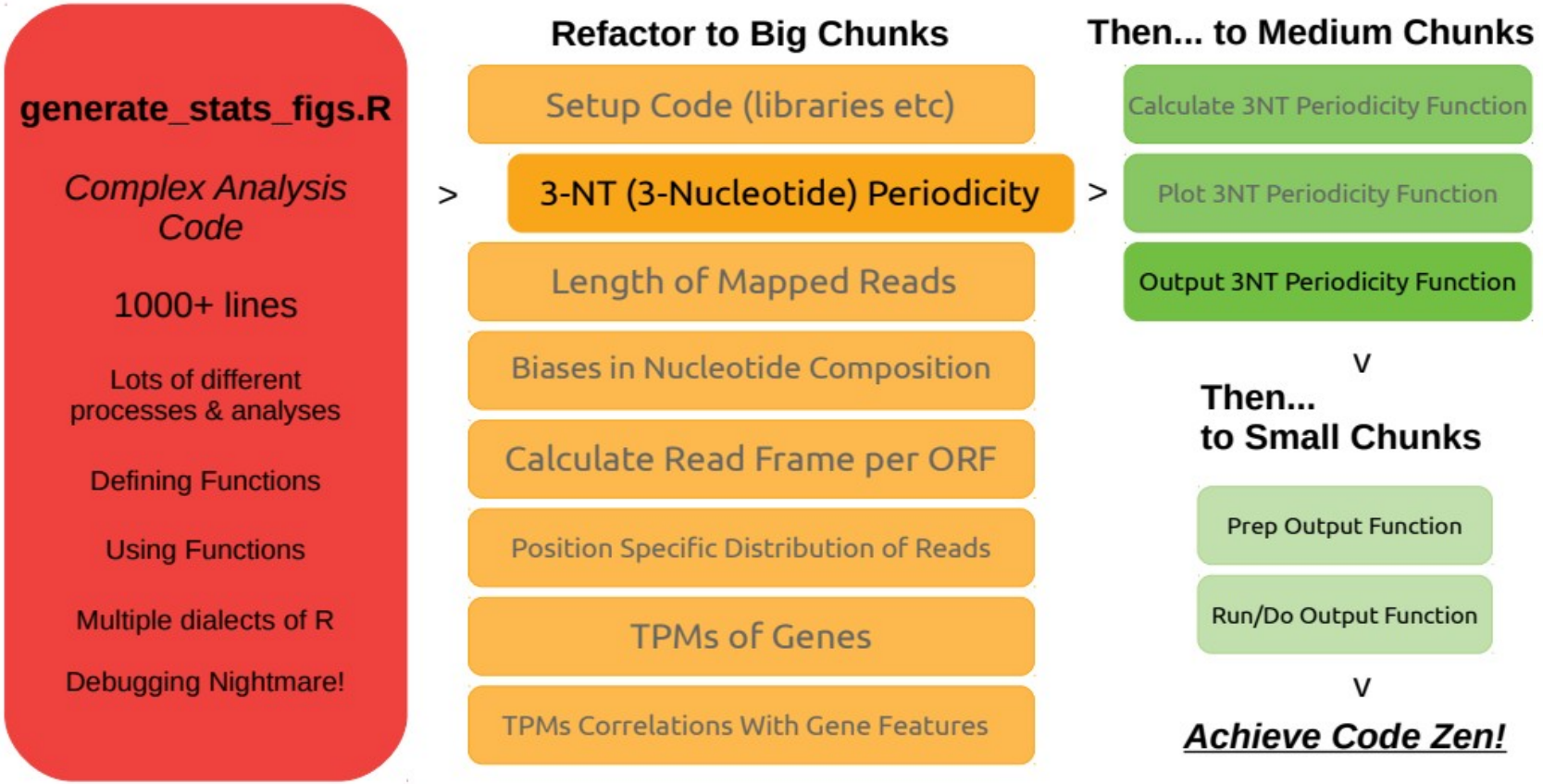


Analysis

generate_stats_figs.R:

Generates summary statistics, analysis plots & quality control plots

Refactoring Analysis Code



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)

```r
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

```

```
# 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()

```

```
296 WriteThreeNucleotidePeriodicity <- function(three_nucleotide_periodicity_data) {
297   tsv_file_path <- file.path(output_dir, paste0(output_prefix, "3nt_periodicity.tsv"))
298   write_provenance_header(path_to_this_script, tsv_file_path)
299   write.table(
300     three_nucleotide_periodicity_data,
301     file = tsv_file_path,
302     append = T,
303     sep = "\t",
304     row = F,
305     col = T,
306     quote = F)
307   # return()? NO RETURN
308 }  # end of function definition WriteThreeNucleotidePeriodicity()
309
```

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? - **Everyone**!!!

“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

*

```
11
12  # Handle interactive session behaviours or use get_Rscript_filename():
13  if (interactive()) {
14    # Use hard-coded script name and assume script is in "rscripts"
15    # directory. This assumes that interactive R is being run within
16    # the parent of rscripts/ but imposes no other constraints on
17    # where rscripts/ or its parents are located.
18    this_script <- "generate_stats_figs.R"
19    path_to_this_script <- here("rscripts", this_script)
20    source(here::here("rscripts", "provenance.R"))
21    source(here::here("rscripts", "read_count_functions.R"))
22  } else {
23    # Deduce file name and path using reflection as before.
24    this_script <- getopt::get_Rscript_filename()
25    path_to_this_script <- this_script
26    source(here::here("rscripts", "provenance.R"))
27    source(here::here("rscripts", "read_count_functions.R"))
28  }
29
```

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)

Refactoring Riboviz Analysis Code:

A Personal Journey

24 June 2020

Flic Anderson
The Wallace Lab, University of Edinburgh

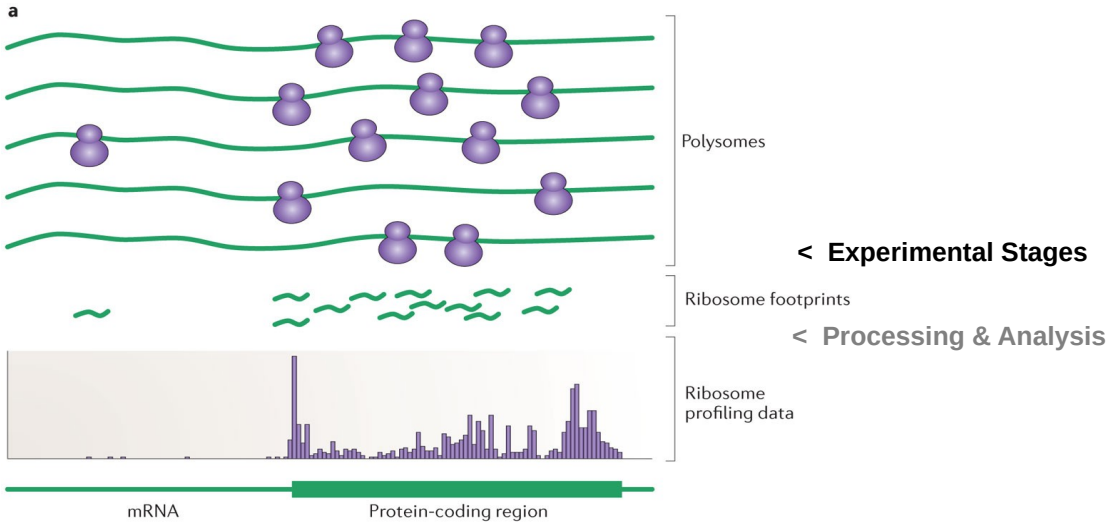
Outline

- 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

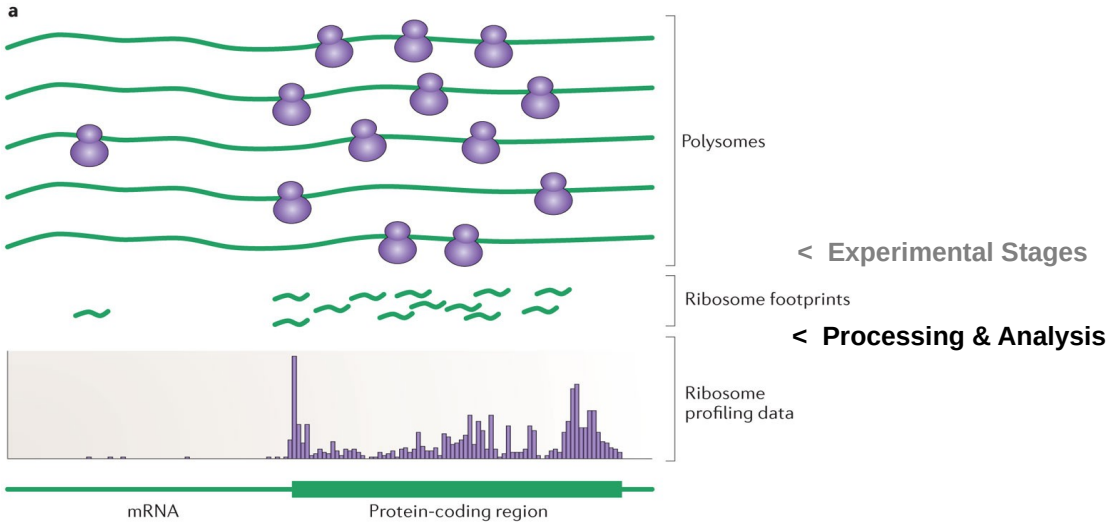


Ingolia (2014). "Ribosome profiling: new views of translation, from single codons to genome scale". Nature Reviews. Genetics. 15 (3): 205–13. doi:10.1038/nrg3645

- Lyse the cells to get at the mRNA molecules bound to ribosomes.
- 'Stop' the translation process: e.g. with cycloheximide or other means
- Digest the non-protected RNA using a nuclease
- Strip away the ribosomes and proteins
- Size-select for these previously 'masked' fragments of mRNA
- Add adapters
- Reverse-transcribe to complimentary DNA
- Amplify
- Sequence

Ingolia NT, Ghaemmaghami S, Newman JR, Weissman JS (2009). "Genome-wide analysis in vivo of translation with nucleotide resolution using ribosome profiling". Science. 324 (5924): 218–23. doi:10.1126/science.1168978

Footprints to Profiling Data



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

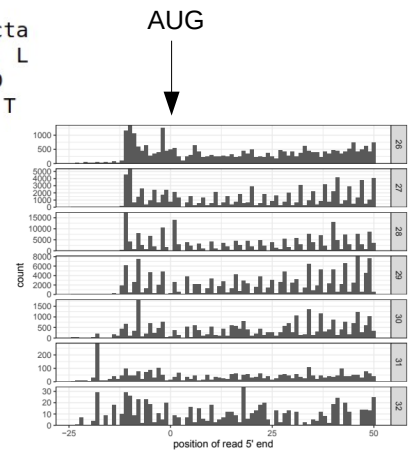
Take these reads & submit to further processing & analysis to get plots & information.

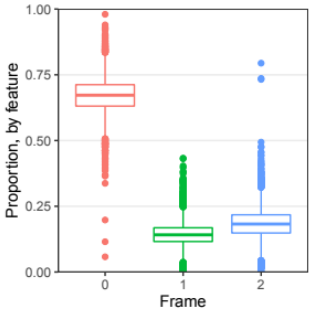
Analysis of Ribosome Profiling Data

- 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.

Ribosome profiling data is really useful for investigating translational mechanics particularly because of the information it provides about the active open reading frame!

Can locate the start codon ‘AUG’ (from annotation position information from a .gff or sequence info) & determine what should be the reading frame based on that.

Can determine which reading frame(s) are being actively translated from the counts of reads sequenced which match each frame. Can look at if this is altered by particular characteristics of the gene or sequence for example.

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)

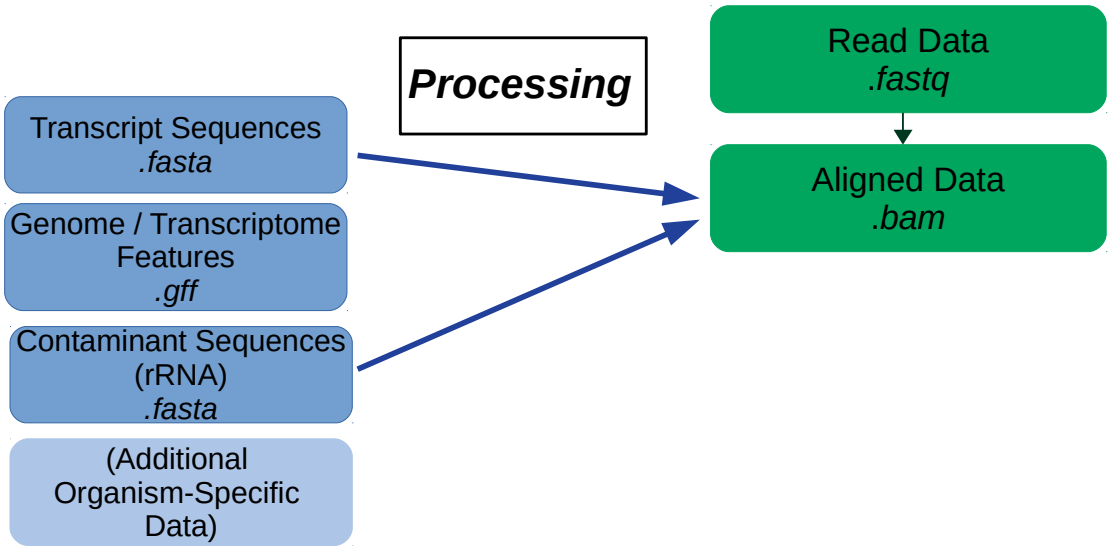
Sample Specific

Read Data
.fastq

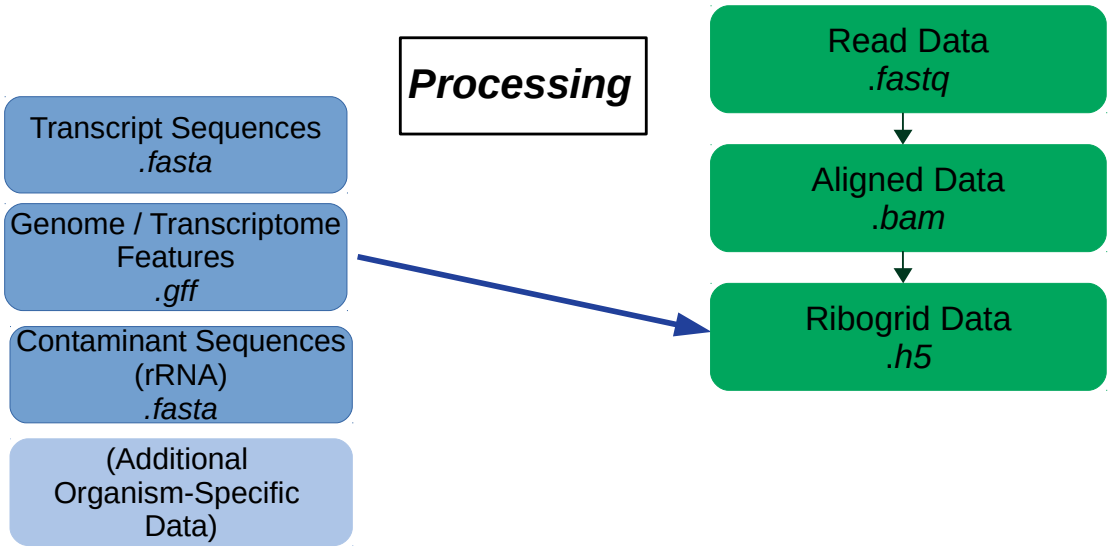
Configuration File
.yaml

Configuration File lists all files & parameters needed to run RiboViz

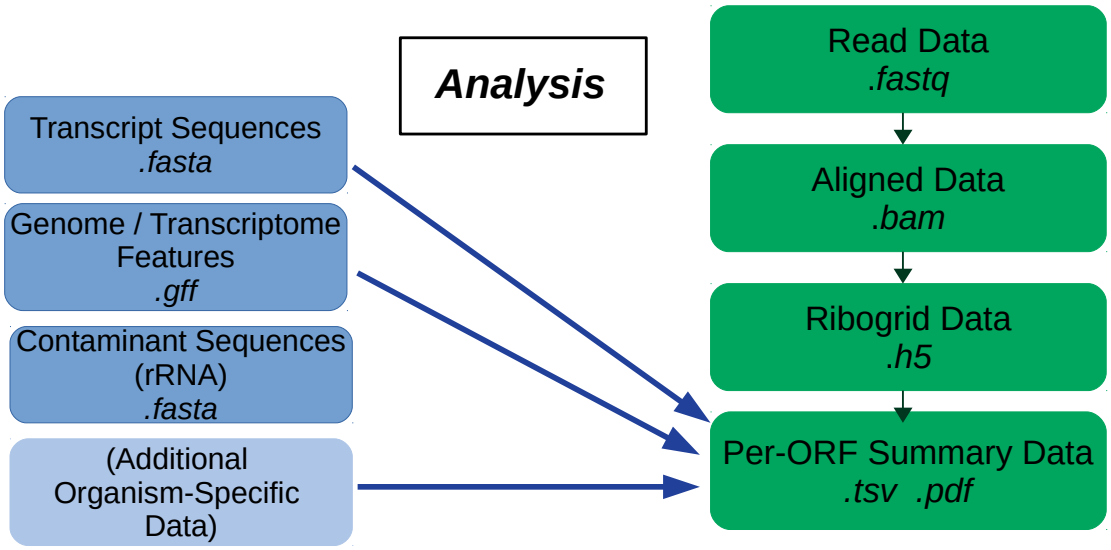
Riboviz Workflow

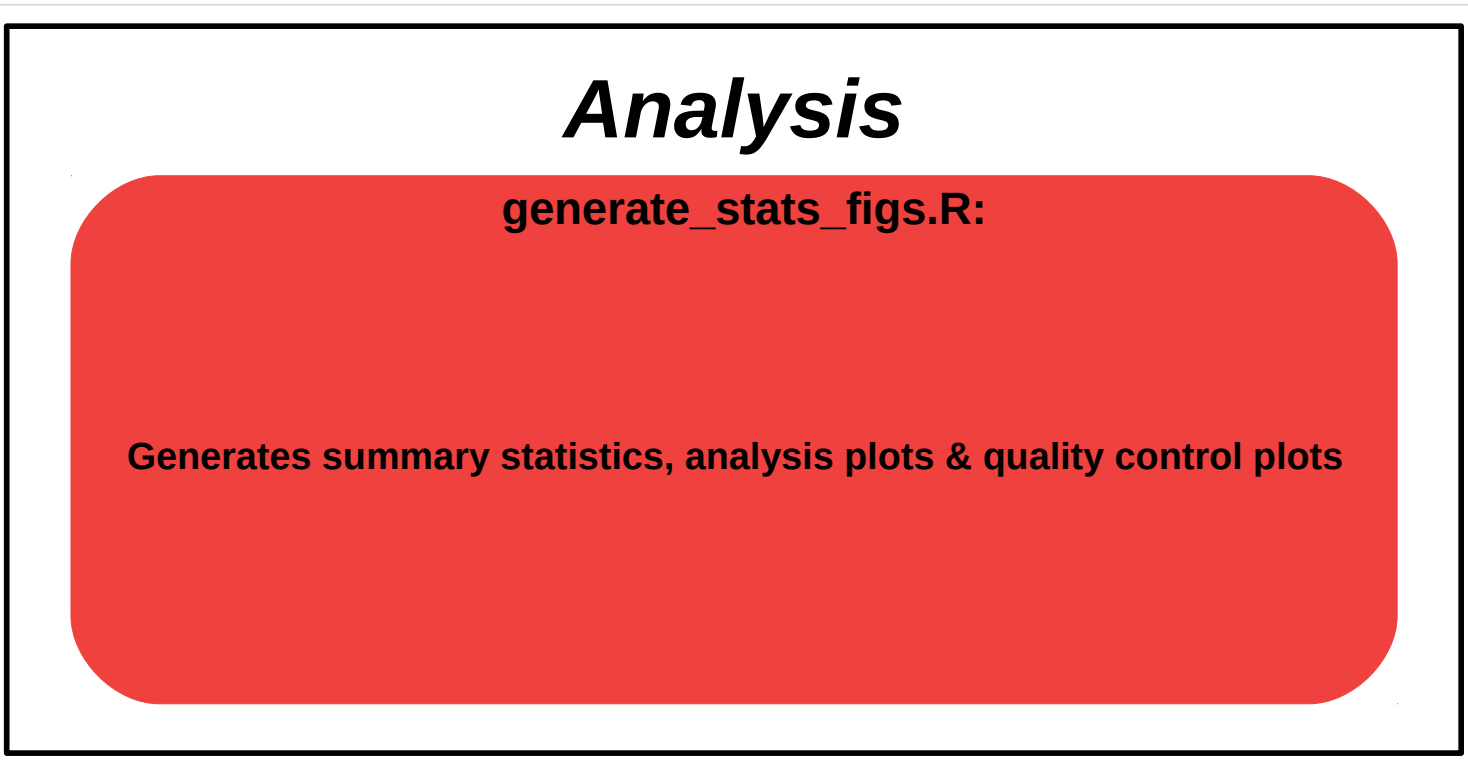


Riboviz Workflow



Riboviz Workflow





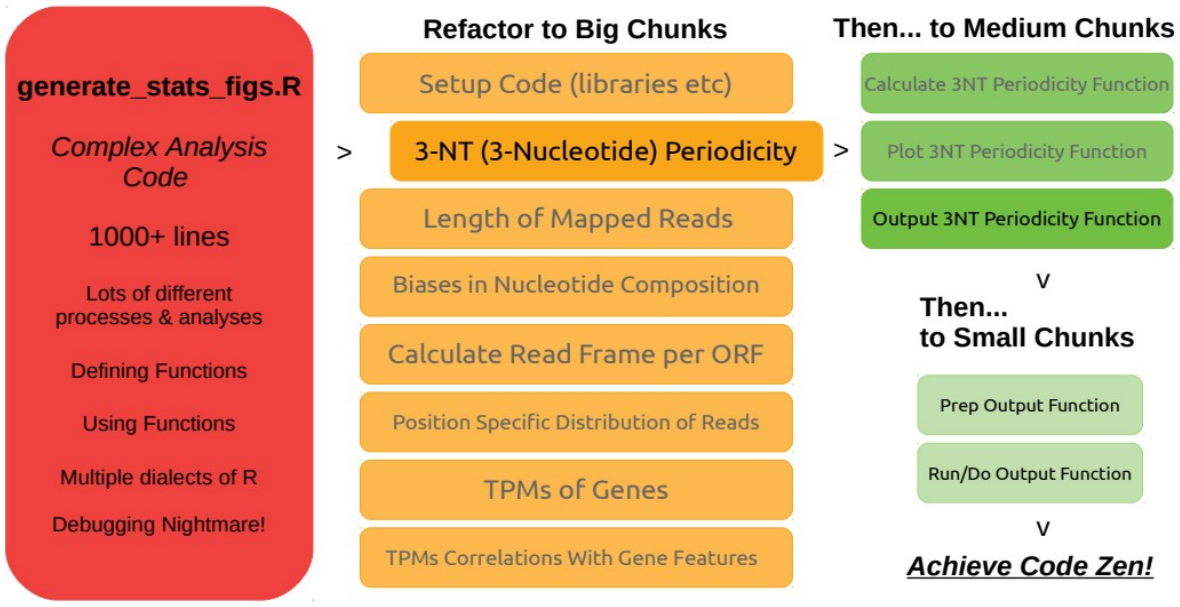
1 key script which handles this:
generate_stats_figs.R

WHAT IT DOES:

Generates summary statistics, analysis plots & quality control plots in a sample

It's run once per sample as part of the riboviz workflow & creates the same (comparable!) outputs for each.

Refactoring Analysis Code



- Setup Code

LARGE CHUNKS:
- 3nt periodicity
- Lengths all mapped reads
- Biases in nucleotide composition
- Calculate read frame for every orf
- RPF position specific distribution of reads
- mRNA position specific distribution of reads
- TPMs of genes
- TPMs correlations with gene features

MEDIUM CHUNKS:
- CalculateFunction - computational code
- PlotFunction - plot creation
- SaveFunction - PDF writeouts of plots
- WriteFunction - .TSV file creation

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)

```r
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)
```


3-NT Periodicity Big chunk code.

- **Easy to read** (maybe not on this screen, sorry)
- Functions & objects **different styling** (not my focus just now in this talk but lots of interesting / helpful stuff to say about it)
- **Print statements** help show where we're at in the logs if something goes wrong
- If things DO go wrong, **error messages benefit from middle-size functions with clear naming**
- Aiming to go with nice tidyverse stuff: e.g. consistent variable names

- NOTE: the medium level functions are defined before this chunk or it doesn't work :)

```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

```


CALCULATE (medium) block does all the heavy lifting.

Lots of functions used here which are defined in (small blocks in) read_count_functions.R script & sourced in - this keeps the structure of this script quite easy to read.

- **{magrittr} pipes (%>%)** where they help make code more understanable vs nested(functions(in(functions)))

- **Returning things explicitly** using return() to avoid unexpected behaviour! Also helps highlight the important objects.

I found it handy to put example output as comments to give an idea of the objects returned & what they might look like. Quite useful while refactoring, but maybe not necessary longer-term & can be removed with a styling pass.

```
# 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()
```


PLOT & SAVE blocks

Same kind of thing: medium block where we've ot plotting code for each type of plot as a separate function.

'Design choices': when multiple blocks look quite similar (e.g. SaveStartCodonRiboGrid + SaveStartCodonRiboGridBar which save out ggplot objects) can be difficult to know when to make 1 function with differing parameters, or keep as 2 separate functions which do almost the same thing…

Decided in this case to keep separate & avoid losing time 'over-optimising', also easy to read so possibly not necessary to fix what ain't broke :)

```
296 WriteThreeNucleotidePeriodicity <- function(three_nucleotide_periodicity_data) {
297   tsv_file_path <- file.path(output_dir, paste0(output_prefix, "3nt_periodicity.tsv"))
298   write_provenance_header(path_to_this_script, tsv_file_path)
299   write.table(
300     three_nucleotide_periodicity_data,
301     file = tsv_file_path,
302     append = T,
303     sep = "\t",
304     row = F,
305     col = T,
306     quote = F)
307   # return()? NO RETURN
308 }  # end of function definition WriteThreeNucleotidePeriodicity()
309
```


WRITE medium block

Nice legible chunk of code: we've got a function with lots of parameters – write.table() - written across multiple lines to avoid going over recommended 80 chars length
Makes it easy to read & quick to identify/change any of those parameters.
Good to do that generally!

- Writing out as .tsv also super helpful because **regression tests can't compare content of .pdfs,** just if .pdfs exist, but CAN compare values in .tsv files – if anything changes, it's easy to know!

Note: #return()? comment handy note to self while I was thinking about what each block should output

Note: I included some "# end of function …" comments at the end after the last '}' as it lets me see the end of that block clearly – something that really helped working on a lot of long code blocks. Stops me getting lost while scrolling, too!
Again, could be easily removed during a re-style.

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...

Think about WWMJD: What Would Mike Jackson Do?
Break it down: Big chunks, Medium chunks, Small chunks, success!
Otherwise you spend hours looking bleakly at a page of impenetrable code & have no idea how to make a dent in it...

Regression Tests: SO useful at telling you something has suddenly stopped working or gives different answers!

Functions in another script, called from the 'main' one. Avoids visual clutter, keeps the script easy to read & understand. Work in main first: easier to write them in the same script then move rather than troubleshooting everything + environment issues

Specific **issue tickets** & issue **branches** keep you focussed on ONE thing: open branch, do work, merge branch, close issue, move on.

A good issue ticket tells you how you'll **know when it's done**, also… Or it's easy to keep tinkering forever… Refactoring turns into styling & then into adding more testing & so on...

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...

Issue proliferation: I got lost and delayed as I tried fixing other issues as they came up & the issues THOSE issues created and… MADNESS. Avoid.

Interactive function behaviour: some functions behave differently when run interactively vs in a sourced script… Makes debugging SUPER tricky!

Package::function() You WILL forget which snazzy package that important function came from otherwise

Also you will probably forget add it to the documentation and tell people to install it and There Will Be Errors.

“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? - **Everyone**!!!

User Testing & Interviews:
- First impressions
- Ease/difficulties of use
- Analyses you want to see
- What doesn’t work / isn’t helpful?

Getting folks to feedback on & improve existing documentation: learned quite a lot from Ania, & now Siyin, & other new users!

Documentation using {roxygen2}:
- Better code documentation is better for users (if nothing else, it ensures developers really THINK about their code before releasing it into the wild!!)

What processing/analysis software tools do you use now that you find useful?
- What features => useful / annoying?
- Any generalisations we can take to riboviz work?

“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?

- Would be good to compare riboviz with other tools & get a feeling for how easy they are to use, what outputs they give & whether their results 'match' - something I'd like to do ahead of a riboviz paper!

- Hard to know without knowing what more users expect from riboviz and what questions they're trying to answer with it. More idea of this as we get more users onboard (esp. user interviews etc)
-

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

*

```
11
12  # Handle interactive session behaviours or use get_Rscript_filename():
13  if (interactive()) {
14    # Use hard-coded script name and assume script is in "rscripts"
15    # directory. This assumes that interactive R is being run within
16    # the parent of rscripts/ but imposes no other constraints on
17    # where rscripts/ or its parents are located.
18    this_script <- "generate_stats_figs.R"
19    path_to_this_script <- here("rscripts", this_script)
20    source(here::here("rscripts", "provenance.R"))
21    source(here::here("rscripts", "read_count_functions.R"))
22  } else {
23    # Deduce file name and path using reflection as before.
24    this_script <- getopt::get_Rscript_filename()
25    path_to_this_script <- this_script
26    source(here::here("rscripts", "provenance.R"))
27    source(here::here("rscripts", "read_count_functions.R"))
28  }
29
```


get_Rscript_filename() useful function which produces the name of the script being run.

It doesn’t act the same way if it’s being run interactively, which

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!

If the original code wasn't tested, how do we know which output is *correct* when we're comparing original output to new output from new code?

Currently we've got 1 section of 'old' code which we've had to roll back to because 'new' code produces different outputs & we need to investigate this!

This is now a new issue, instead of holding back the rest of the refactoring work.

Importance of code testing… & understandable code!

Updates! UX

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

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

Previously spoke about Challenges: UX & Testing Reliability. Now checking in on those.

UX:

- Building user base I'm not sufficiently active enough on that, but our collaborators are currently running datasets with version 2.0-beta ahead of v2.0 release!
- New users (inc Siyin!) posting questions & issues on github – that's a good sign of the start of a building community!
- What do users need/want? Haven't survey'd anyone yet… Qs for Sam! :)
- Onboarding with oven-ready datasets? YES! Example-datasets repo, PR for a Cryptococcus dataset Siyin has just run on the Eddie cluster Progress!

Updates! UX

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

UX contd.: Support & Docs:

- Had comments from Ania, now Siyin reading & giving feedback, adding to & improving.
- New documentation like running on Eddie (UoE-specific, certainly, but really useful to know how to set up on any cluster, potentially help other researchers do the same on theirs?)
- Documentation of outputs? Still on my TODO list, but NOW I've worked through refactoring, it's going to be easier to document. Also, #32 led to fixing #155 (.h5 issue) & now fixed the .Rmd issue, so can start to work on getting markdown output & win a 1-page .html option ?
- Translation UK sadly postponed until 2021…
- Version 2.0-beta? Out now, & working towards v2.0 release! Will include the refactored code!

Updates! Testing

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

- New features like Nextflow workflow implementation alongside (sort of original) python implementation!
- .h5 file access more reliable & can now be parallelised (e.g. plays better with clusters like Eddie)
- Pull requests reviewed before being merged (more than one person checks it works), lots of this happened recently – very exciting to be part of tbh
- New datasets through means new potential data weirdness that might break riboviz. Want to know it works over a wide range of different organisms/ data sizes/etc & will work for other groups' data & not just this lab.
  - e.g. Mouse data? Drosophila? Bacteria? Human? Transcriptome vs Genome files...

Updates! Testing

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

As I mentioned, the regression tests have been super useful at showing what's different between old & new code

But, this has underlined that we didn't have tests for the code in the first place. Some investigations to be done on some of the analysis code to confirm whether 'new' code is more correct or not…?

{testthat} unit testing R package for future? Would be really helpful to implement this for the generate_stats_figs.R code to make sure it DOES do what we expect! (Plus it promises to "make testing as fun as possible", so who could say no?!)

Close to getting the Lareau Lab simulated ribosome profiling data through riboviz: going to be interesting to see what this shows us about riboviz!

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)

- Didn’t need too much work
- Some libraries in main script, most in another sourced script (read_count_functions.R)
- SuppressMessages(libray(...)) avoids lots of loading messages
- Talk a bit about the interactive session behaviours issue shortly
- Parameters come from configuration .yaml file & are passed in to generate_stats_figs.R via riboviz pipeline (python script or nextflow workflow)
- Parameters handled using option parseing, allows us to have set defaults
- Read in: .gff info on sequence annotations, .fa has the coding sequences, .h5 has the sample reads