Exploring Codon Frequency Biases and Redundancies Across Organisms using R

By: Patrick Blaney

DA DA5020 37951 Collect/Store/Retrieve Data SEC 02 Spring 2018

Overview:

One of the most important aspects of the biological life is the built-in safeguards that allow organisms to fully function despite a mutation to their to a critical component of their DNA. These mutations usually only occur at a single point, called which is called a base, within the entire genome of the organism and a high percentage of them have no effect on functionality of a gene. The danger of these mutations comes when they happen within a coding DNA sequence (CDS) region. This single point error can alter the codons that are read to produce the amino acids of a specific protein and result in the protein losing its functionality.

This is where codon redundancy comes into play. Each codon consists of three bases which can be either A, T, G, or C. This leads to many different possible combinations that can make up one codon. Since the number of amino acids produced by these codons is even more limited, there are cases where one amino acid is produced by more than one codon. The redundancy built into this process allows the organism to have an extra layer of protection against a single point mutation causing a crucial protein to lose its functionality.

The aim of this project is to look over all the CDS regions of various organisms in order to identify if there is a bias in codon sequence frequency for those amino acids that have high redundancy.

Collection of Data:

This aspect of the project took the most care and exploration as there are many different websites that offer free access to highly accurate DNA sequences for a wide range of organisms. For this project, the target sequences needed for analysis were those that only contained CDS regions.

The three organisms that will be explored are E. coli K-12, Schizosaccharomyces pombe, and C. elegans. They were selected based on a few criteria: Genome size, belonging to a unique taxonomic Kingdom, and accuracy of CDS.

All three organisms have a very well known and experimentally proven genome which encompasses all the CDS. Additionally, they all have a total genome size that is reasonable to work with for the scope of this project in terms of memory and computational power/time. The genome size for E. coli K-12, Schizosaccharomyces pombe, and C. elegans are: 4,639,221, 12,462,637, and 100,258,171, respectively. I believe it is essential for the quality of the project to include the drastic change in size as these numbers correspond to the difference in taxonomy as well. These three represent the Bacteria, Fungi, and Animalia Kingdoms, respectively. This is an opportunity to show that how genome and organism complexity may be related

Given the size of each organism's genome, the file sizes will also grow in size (4.7 to 52 MB after being decompressed) and will require more computational time for processing. One step in particular requires a significantly longer completion time even though it does produce the expected output. There is a comment regarding this step within the provided R Markdown file.

STEP 1: Identify a website that has target information in a form that is easily accessible to anyone.

Initially, I was going to utilize the National Center for Biotechnology Information (NCBI) website which is one of the most trusted sources but I ran into difficulty when trying to create an automated process for effectively retrieving the CDSs by way of scraping the sequences directly from the NCBI web page. The main issue was that all the sequences of interest were not easily isolated with a CSS selector. Below are two screenshots that compare what the web page looks like naturally and how it looks when using SelectorGadget, the Google Chrome extension:

```
gene            337..2799
                /locus_tag="CXP41_00010"
CDS             337..2799
                /locus_tag="CXP41_00010"
                /inference="COORDINATES: similar to AA
                sequence:RefSeq:WP_005124053.1"
                /note="Derived by automated computational analysis using
                gene prediction method: Protein Homology."
                /codon_start=1
                /transl_table=11
                /product="bifunctional aspartokinase I/homoserine
                dehydrogenase I"
                /protein_id="AUG14806.1"
                /translation="MRVLKFGGTSVANAERFLRVADILESNARQGQVATVLSAPAKIT
                NHLVAMIEKTISGQDALPNISDAERIFAELLTGLAAAQPGFPLAQLKTFVDQEFAQIK
                HVLHGISLLGQCPDSINAALICRGEKMSIAIMAGVLEARGHNVTVIDPVEKLLAVGHY
                LESTVDIAESTRRIAASRIPADHMVLMAGFTAGNEKGELVVLGRNGSDYSAAVLAACL
                RADCCEIWTDVDGVYTCDPRQVPDARLLKSMSYQEAMELSYFGAKVLHPRTITPIAQF
                QIPCLIKNTGNPQAPGTLIGASRDEDELPVKGISNLNNMAMFSVSGPGMKGMVGMAAR
                VFAAMSRARISVVLITQSSSEYSISFCVPQSDCVRAERAMQEEFYLELKEGLLEPLAV
                TERLAIISVVGDGMRTLRGISAKFFAALARANINIVAIAQGSSERSISVVVNNDDATT
                GVRVTHQMLFNTDQVIEVFVIGVGGVGGALLEQLKRQQSWLKNKHIDLRVCGVANSKA
                LLTNVHGLNLENWQEELAQAKEPFNLGRLIRLVKEYHLLNPVIVDCTSSQAVADQYAD
                FLREGFHVVTPNKKANTSSMDYYHQLRYAAEKSRRKFLYDTNVGAGLPVIENLQNLLN
                AGDELMKFSGILSGSLSYIFGKLDEGMSFSEATTLAREMGYTEPDPRDDLSGMDVARK
                LLILARETGRELELADIEIEPVLPAEFNAEGDVAAFMANLSQLDDLFAARVAKARDEG
                KVLRYVGNIDEDGVCRVKIAEVDGNDPLFKVKNGENALAFYSHYYQPLPLVLRGYGAG
                NDVTAAGVFADLLRTLSWKLGV"
```

```
     CDS             337..2799
                     /locus_tag="CXP41_00010"
                     /inference="COORDINATES: similar to AA
                     sequence:RefSeq:WP_005124053.1"
                     /note="Derived by automated computational analysis using
                     gene prediction method: Protein Homology."
                     /codon_start=1
                     /transl_table=11
                     /product="bifunctional aspartokinase I/homoserine
                     dehydrogenase I"
                     /protein_id="AUG14806.1"
                     /translation="MRVLKFGGTSVANAERFLRVADILESNARQGQVATVLSAPAKIT
                     NHLVAMIEKTISGQDALPNISDAERIFAELLTGLAAAQPGFPLAQLKTFVDQEFAQIK
                     HVLHGISLLGQCPDSINAALICRGEKMSIAIMAGVLEARGHNVTVIDPVEKLLAVGHY
                     LESTVDIAESTRRIAASRIPADHMVLMAGFTAGNEKGELVVLGRNGSDYSAAVLAACL
                     RADCCEIWTDVDGVYTCDPRQVPDARLLKSMSYQEAMELSYFGAKVLHPRTITPIAQF
                     QIPCLIKNTGNPQAPGTLIGASRDEDELPVKGISNLNNMAMFSVSGPGMKGMVGMAAR
                     VFAAMSRARISVVLITQSSSEYSISFCVPQSDCVRAERAMQEEFYLELKEGLLEPLAV
                     TERLAIISVVGDGMRTLRGISAKFFAALARANINIVAIAQGSSERSISVVVNNDDATT
                     GVRVTHQMLFNTDQVIEVFVIGVGGVGGALLEQLKRQQSWLKNKHIDLRVCGVANSKA
                     LLTNVHGLNLENWQEELAQAKEPFNLGRLIRLVKEYHLLNPVIVDCTSSQAVADQYAD
                     FLREGFHVVTPNKKANTSSMDYYHQLRYAAEKSRRKFLYDTNVGAGLPVIENLQNLLN
                     AGDELMKFSGILSGSLSYIFGKLDEGMSFSEATTLAREMGYTEPDPRDDLSGMDVARK
                     LLILARETGRELELADIEIEPVLPAEFNAEGDVAAFMANLSQLDDLFAARVAKARDEG
                     KVLRYVGNIDEDGVCRVKIAEVDGNDPLFKVKNGENALAFYSHYYQPLPLVLRGYGAG
                     NDVTAAGVFADLLRTLSWKLGV"
```

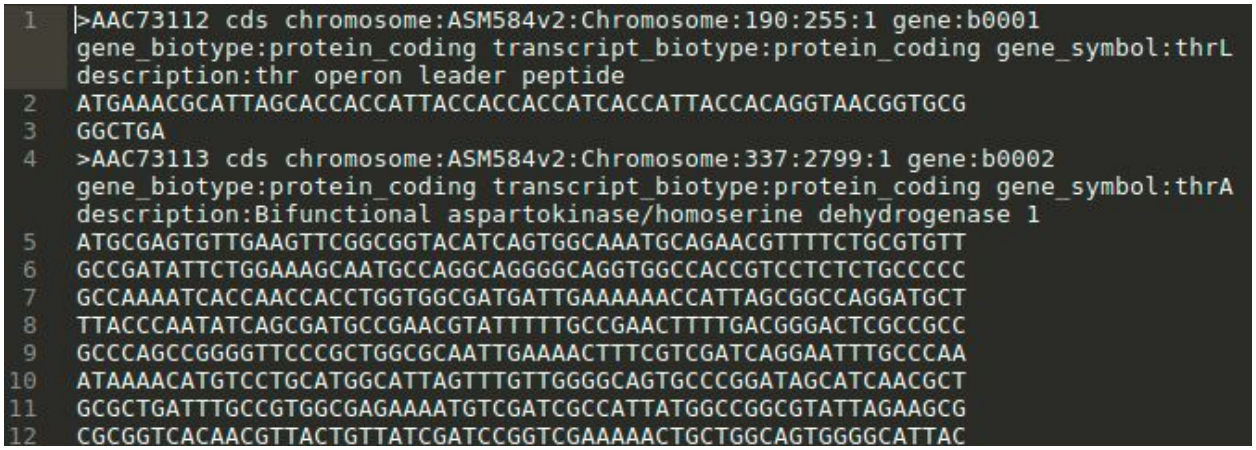
From here you can see that the CDS regions are displayed but they displayed as the amino acid sequence, I need the nucleotide sequence. Additionally, I attempted to use the “CDS” hyperlinks to direct me to the web page containing the nucleotide sequence but this also proved to be an issue as they are technically pseudo-links due.

I then shifted my idea from web-scraping to downloading a file that contains all the CDS regions for an organism in FASTA format. It was during this search that I decided I would utilize access to Ensembl’s extensive FTP website as my source for all data on the organisms I decided to analyze for this project. Below is a screenshot of the web page interface to Ensembl’s FTP Download:

STEP 2: View the data to ensure that it is in the desired format.

From this page, the hyperlinks associated with each species will bring the user directly to the location of that file within the FTP repository. Also, there is a very important note at the bottom of the image that lets the user know that the files will all be compressed into a gzip (*.gz) file. Upon opening the zip file, the FASTA formatted file will consist of a header that starts with a “>” and contains general information about

each CDS region followed by the nucleotide sequence for that region. Below is an image of a FASTA formatted entry:



STEP 3: Decide which method is best used to pull data from this website.

Now that I have confirmed that I have the data I want in a format that I can work with, I must decide what the best method would be to download these files into my R environment. Fortunately, the “curl” pagack of R has a modified version of the “download.file()” called “curl_download()” which supports interactions with FTPs very well. The function “curl_download()” takes two arguments: One being the URL of the location of the file and the second being the name it will be saved under within the user’s current working directory of the R session. I decided this method would be my preferred process as it is both simplified and reliable.

Storage of Data:

Now that the file has been pulled down from the Ensembl website into the R environment, the lines containing the sequences can be extracted and organized into a

dataframe, along with other pieces of metadata, for easy storage into a SQLite database.

STEP 1: Unzip the *.gz file that was downloaded to allow access to FASTA file.

As noted earlier, all the files that are downloaded from Ensembl's FTP website are compressed into gzip files to save space as these files can become quite large if dealing with very large genomes. Fortunately, the R package "R.Utils" has a function "gunzip()" that will decompress any gzip file, give a new name to the unzipped file, and remove the old compressed version to save memory. This function takes two strings as arguments: The name of the origin gzip file and the new desired name. Now I have the raw CDS FASTA file available for R to interact with.

STEP 2: Isolate individual nucleotide sequences from the FASTA file.

Given nature of the FASTA format, parsing the file should be straightforward but to simplify the process even further, I made use of the function "read.fasta()" that is part of the "seqinr" package in R. For this particular case, this function will take three arguments: The name of the FASTA file, set "as.string" and "seqonly" to TRUE. This will output only the nucleotide sequences, headers are excluded, as individual strings in a list of strings. Having the sequences stored as strings will allow for easy division of the the sequence into individual codons in a step downstream.

STEP 3: Compile sequences into a data frame along with background metadata.

With the objective of comparing the makeup of all the CDS regions between different organisms, it is most fitting to use a relational database, particularly a SQLite

database. The top level table that will be stored in the database will contain various pieces of metadata pertaining the each organism. This information will be used to explore any potential correlations between themselves and codon frequency bias. Below is the populated data frame titled “Organism_Overview”:

OrganismName	GenomeSize	TotalCDS	Taxonomy
E. coli K-12	4,639,221	4,140	Bacteria
Schizosaccharomyces pombe	12,462,637	5,146	Fungi
C. elegans	100,258,171	33,391	Animalia

The “GenomeSize” column contains integer values of the total number of base pairs within the DNA of the organism. I specifically kept this as a numeric value so that some arithmetic could be possibly be applied to it in the downstream analysis. The “TotalCDS” column contains a count of all the CDS regions that are within the downloaded FASTA file. This integer is of interest so to compare the amount of CDS per genome size for each organism. The “Taxonomy” column will be populated with the name of the Kingdom of each organism to show how things might change when comparing vastly different species. The primary key for this table is the organism’s name. This will also be a foreign key in all other second level tables. All of the metadata in this data frame was sourced individually from websites that are cited in the References section at the end of this report.

The second level tables will contain all the sequence strings for that organism and an associated CDS ID that was artificially created to provide a primary key for each

table. Below is the structure of the data frames that contain CDS information for each organism of interest:

OrganismName	CDSID	CDS
C.elegans	ce1	“ATGGTG….”
C.elegans	ce2	“AAATGTTG…”

The organism name populated for each row in the column was through use of the “rep()” function within R. This function takes two arguments: A string you would like to repeat and the number of times you would like it repeated. The CDS ID was created to uniquely identify all the CDS strings associate with each organism which inherently provides the table with a primary key, even though it is artificial. These CDS IDs were generated by combining the initials of the organism with the index of the individual sequence. Below demonstrates the R code used:

```
# Create CDS ID for each unique CDS string for each organism, these will be used as a primary
# key for each following organism CDS Information tables
```{r,}
ecoli_cds_id <- tibble(
  "initials" = rep("ec",length(ecoli_cds)),
  "cds number" = c(1:length(ecoli_cds))
)

schizosaccharomyces_pombe_cds_id <- tibble(
  "initials" = rep("sp",length(schizosaccharomyces_pombe_cds)),
  "cds number" = c(1:length(schizosaccharomyces_pombe_cds))
)

celegans_cds_id <- tibble(
  "initials" = rep("ce",length(celegans_cds)),
  "cds number" = c(1:length(celegans_cds))
)
```

```

Additionally, instead of creating one large table of all organisms and CDSs together, I used a table for each organism just as a preference. The CDS column contains all the

sequences found in the FASTA file which are stored as strings. In order to retrieve the individual sequence string from the list of all the CDS strings for each organism to store in a row, they must first be unlisted individually as they are added. Below is a screenshot of the code used to populate the column:

```
"CDS" = c(unlist(ecoli_cds[1:length(ecoli_cds)]))
```

STEP 4: Create SQLite database and populate tables into this database

Now that all the tables have been created for each organism with a relation to the "Organism_Overview" table, they need to be stored into a relational database using R. For this project, I used a SQLite database and interacted with it using the "RSQLite" package. Below is a screenshot of the R code used to connect with the database and insert tables into it.

```
db <- dbConnect(SQLite(), dbname = "coding_DNA_sequences.sqlite")

dbWriteTable(conn = db,
             name = "Organism_Overview",
             value = organism_overview,
             row.names = FALSE,
             overwrite = TRUE
)

dbWriteTable(conn = db,
             name = "Ecoli_CDS_Information",
             value = ecoli_cds_information,
             row.names = FALSE,
             overwrite = TRUE
)

dbWriteTable(conn = db,
             name = "Schizosaccharomyces_Pombe_CDS_Information",
             value = schizosaccharomyces_pombe_cds_information,
             row.names = FALSE,
             overwrite = TRUE
)

dbWriteTable(conn = db,
             name = "Celegans_CDS_Information",
             value = celegans_cds_information,
             row.names = FALSE,
             overwrite = TRUE
)
```

The first line creates the connection to the database as well as gives it a name for your current working directory. Each line after that utilizes the “dbWriteTable()” function to move a pre-written data frame easily into the database. Notice that each time the table is written, the arguments “row.names = FALSE” and “overwrite = TRUE” are passed, this is to ensure that now extra rows are added with just the column name and the overwrite is to ensure that an update to the table upstream of this steam can easily be passed through using the same script.

Retrieving Data for Analysis

Now that the target sequences are neatly organized into a table within a database, the analysis can begin by querying the database.

STEP 1: Pull out the CDSs from the SQLite database for each organism.

This query will be written using SQL coupled along with the “dbSendQuery()” function. This function takes two arguments: The variable of the database being queried, and the SQL request written as a string. The screenshot below demonstrates:

```
180  celegans_query <- dbSendQuery(db,
181    "SELECT [CDS]
182    FROM Celegans_CDS_Information"
183  )
```

The column CDS contains all the information of interest and I want all the observations for each specific organism. Therefore, I SELECT the [CDS] from organism’s table. I then store this query for CDS strings in a variable that will then be passed to the

function “dbFetch()”. This function takes a variable that stores a query of the database specified in the “dbSendQuery()” function. Below is a screenshot of the R code:

```
185  celegans_cds_strings <- dbFetch(celegans_query)
186  ```
```

The fetched query is then assigned to a new variable that will contain only a column of CDS strings for that respective organism.

Downstream Processing and Analysis

Now all the CDS strings have been retrieved from the database, the first steps towards the analysis can begin. The focus of the analysis is to output a chart of the total count of each codon throughout all the CDS within each organism. Along with each graph, a table containing the numbers being represented in the chart is also created. The final deliverable will be the 3 graphs as well as some tables of filtered data frames to show example of conclusion.

STEP 1: Extract all individual codons from CDS strings for each organism and store in vector form.

Since all the CDS are string data type, the simplest solution was to extract each codon as a string using the “str_extract_all()” function from the “tidyverse” R package. Below is a screenshot example of the R code used:

```
195  schizosaccharomyces_pombe_cds_codons <-
196    str_extract_all(schizosaccharomyces_pombe_cds_strings, "[ATGC]{3}", simplify = TRUE) %>%
197    as.vector()
```

The first argument passed is the variable that holds the CDS strings for the organism. The second argument is a regular expression that is used to capture all possible codons by setting the pattern to any combination of three of the letters A, T, G, or C. The final parts of the code organize the output to be one vector of all codons.

STEP 2: Plot the total count of all unique codons using a bar graph.

The benefit of converting the extracted codons into a vector in the previous step pays off here as it allows for a way to plot the total count of each unique codon with only three lines of R code, as seen below:

```
207  # E. coli K-12
208  ggplot() +
209    geom_bar(aes(ecoli_cds_codons)) +
210    coord_flip() +
```

With this as the base, the desired output can be easily viewed. Extra lines were added to the code to improve the visibility and presentation of the graphs.

STEP 3: Create one vector containing strings of all 64 possible codons and another vector containing the names of each amino acid that corresponds to the each of the 64 possibilities. The image below shows the vectors that were manually populated:

```r
possible_codons <- c(
  "ATT","ATC","ATA",
  "CTT","CTC","CTA","CTG","TTA","TTG",
  "GTT","GTC","GTA","GTG",
  "TTT","TTC",
  "ATG",
  "TGT","TGC",
  "GCT","GCC","GCA","GCG",
  "GGT","GGC","GGA","GGG",
  "CCT","CCC","CCA","CCG",
  "ACT","ACC","ACA","ACG",
  "TCT","TCC","TCA","TCG","AGT","AGC",
  "TAT","TAC",
  "TGG",
  "CAA","CAG",
  "AAT","AAC",
  "CAT","CAC",
  "GAA","GAG",
  "GAT","GAC",
  "AAA","AAG",
  "CGT","CGC","CGA","CGG","AGA","AGG",
  "TAA","TAG","TGA")
```

```r
corresponding_amino_acid <- c(
  rep("Isoleucine", 3),
  rep("Leucine", 6),
  rep("Valine", 4),
  rep("Phenylalanine", 2),
  "Methionine",
  rep("Cysteine", 2),
  rep("Alanine", 4),
  rep("Glycine", 4),
  rep("Proline", 4),
  rep("Threonine", 4),
  rep("Serine", 6),
  rep("Tyrosine", 2),
  "Tryptophan",
  rep("Glutamine", 2),
  rep("Asparagine", 2),
  rep("Histidine", 2),
  rep("Glutamic acid", 2),
  rep("Aspartic acid", 2),
  rep("Lysine", 2),
  rep("Arginine", 6),
  rep("Stop codon", 3)
)
```

STEP 4: Write a function that loops through all 64 possible codons and sums the total number of times that codon occurs within each organism. The function will then append each new total to initialize empty vector.

```
285 count_codons <- function(all_codons) {
286   count_of_each <- integer()
287   for (i in 1:64){
288     count_of_each[i] <-  sum(str_count(all_codons, string = possible_codons[i]))
289     print(count_of_each[i])
290   }
291   return(count_of_each)
292 }
```

The chosen name of the function is “count_codons” and it takes one argument: A vector of many strings. An empty vector is first initialized which will be populated with the total count of each occurence of a possible codon. This step helps increase the speed of the for loop that is used next. The loop iterates through 64 times and each time it counts all occurrences found that match the regular expression. The regular expression in this case is the codon string that is found where the index matches the number of the current iteration through the loop. A sum of all the matches is then recorded and added to the empty vector at the index equal to the number of the current iteration. Finally, the function returns the vector that has a count of occurance for all 64 possible codons.

STEP 5: Create data frame that matches up all possible codons with their associated count and corresponding amino acid.

This data frame will be paired with bar graph that visualizes the total count to give a context of which codons have a greater redundancy then others. This data frame can also be filtered even more specifically using the R package “tidyverse” and its

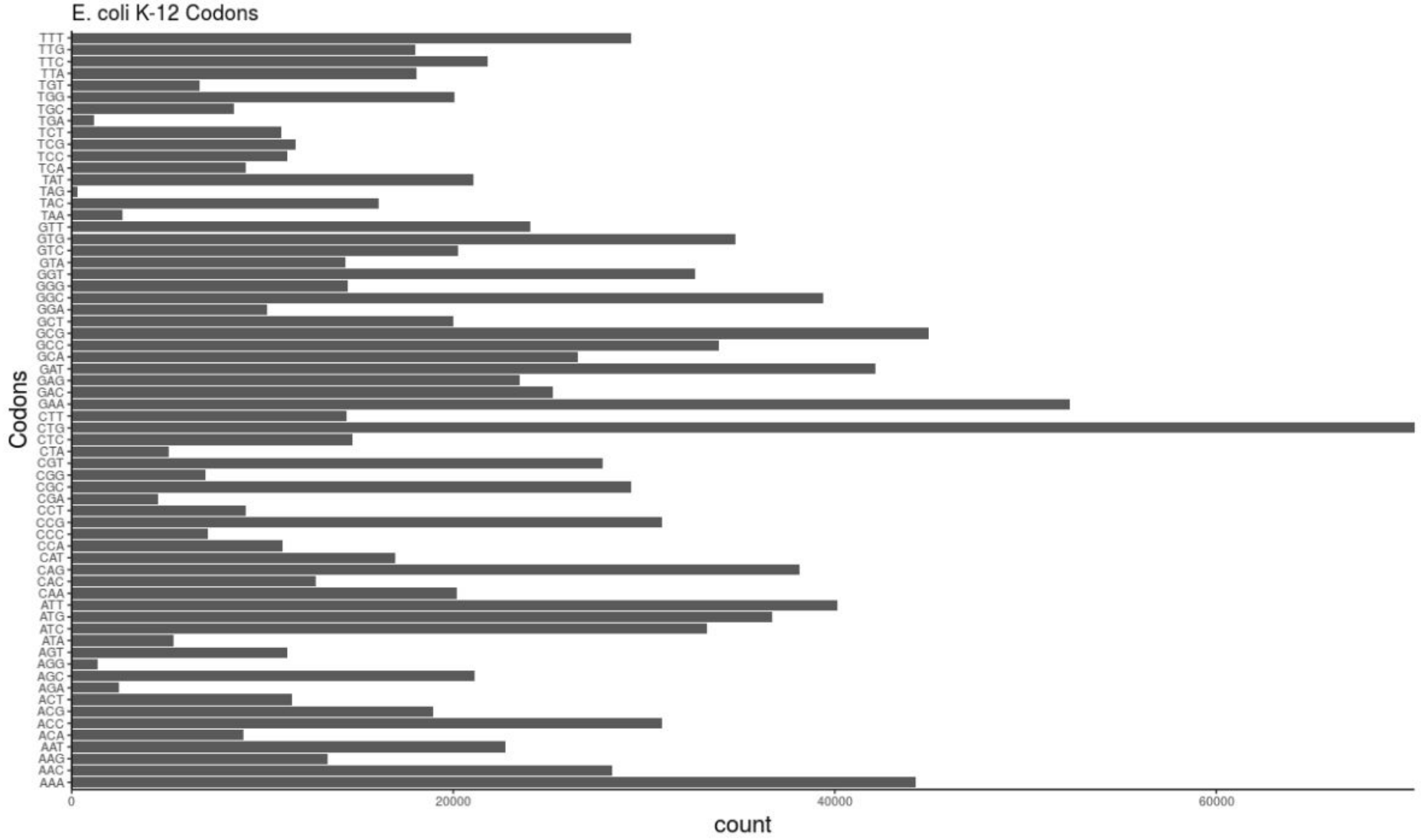
associated functions “select()” and “filter()”. The following is an example of the code and the resulting data frame:

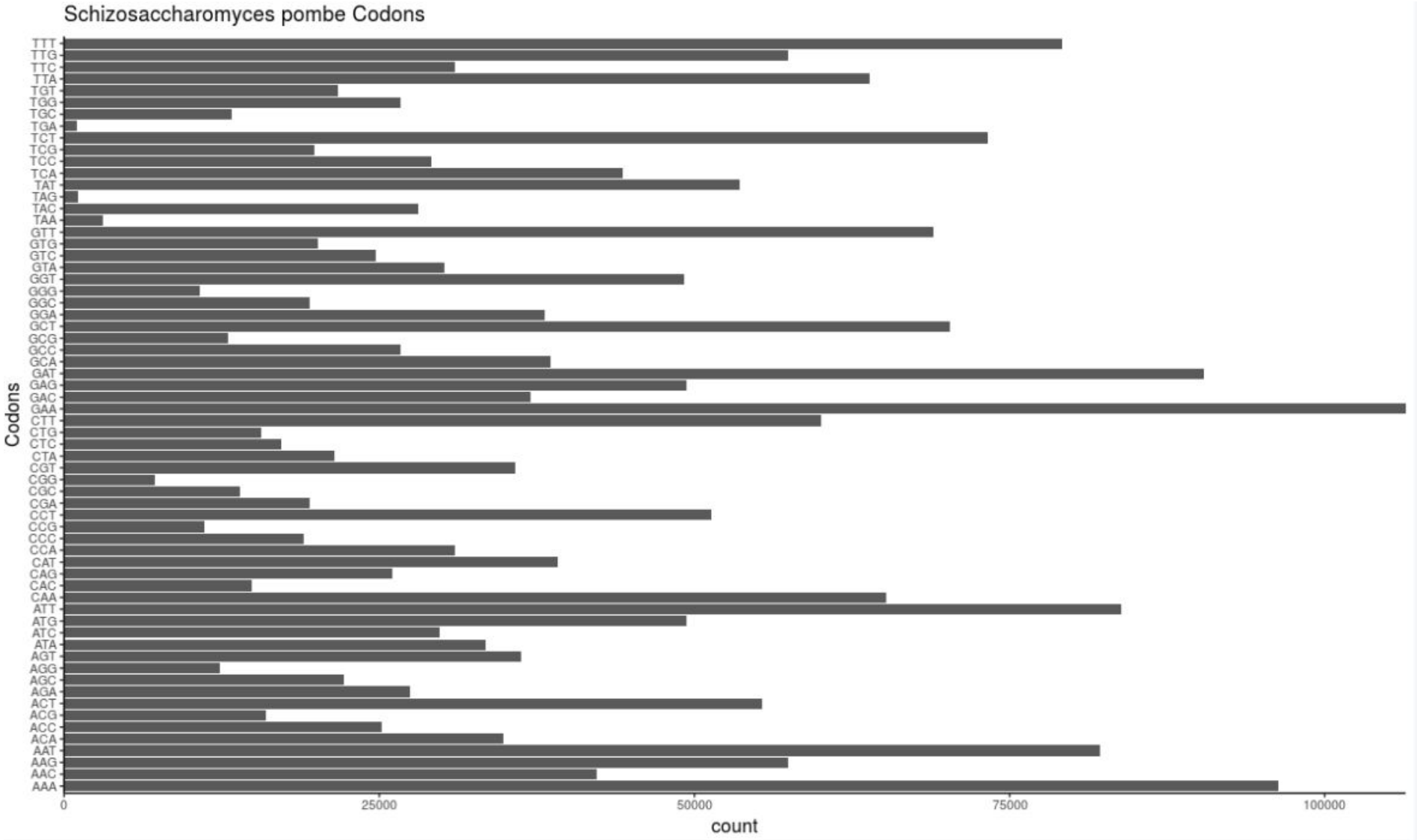
```
333  celegans_possible_codons_count <- tibble(
334    "Possible Codons" = possible_codons,
335    "Count" = celegans_codon_count,
336    "Amino Acid" = corresponding_amino_acid
337  )
```

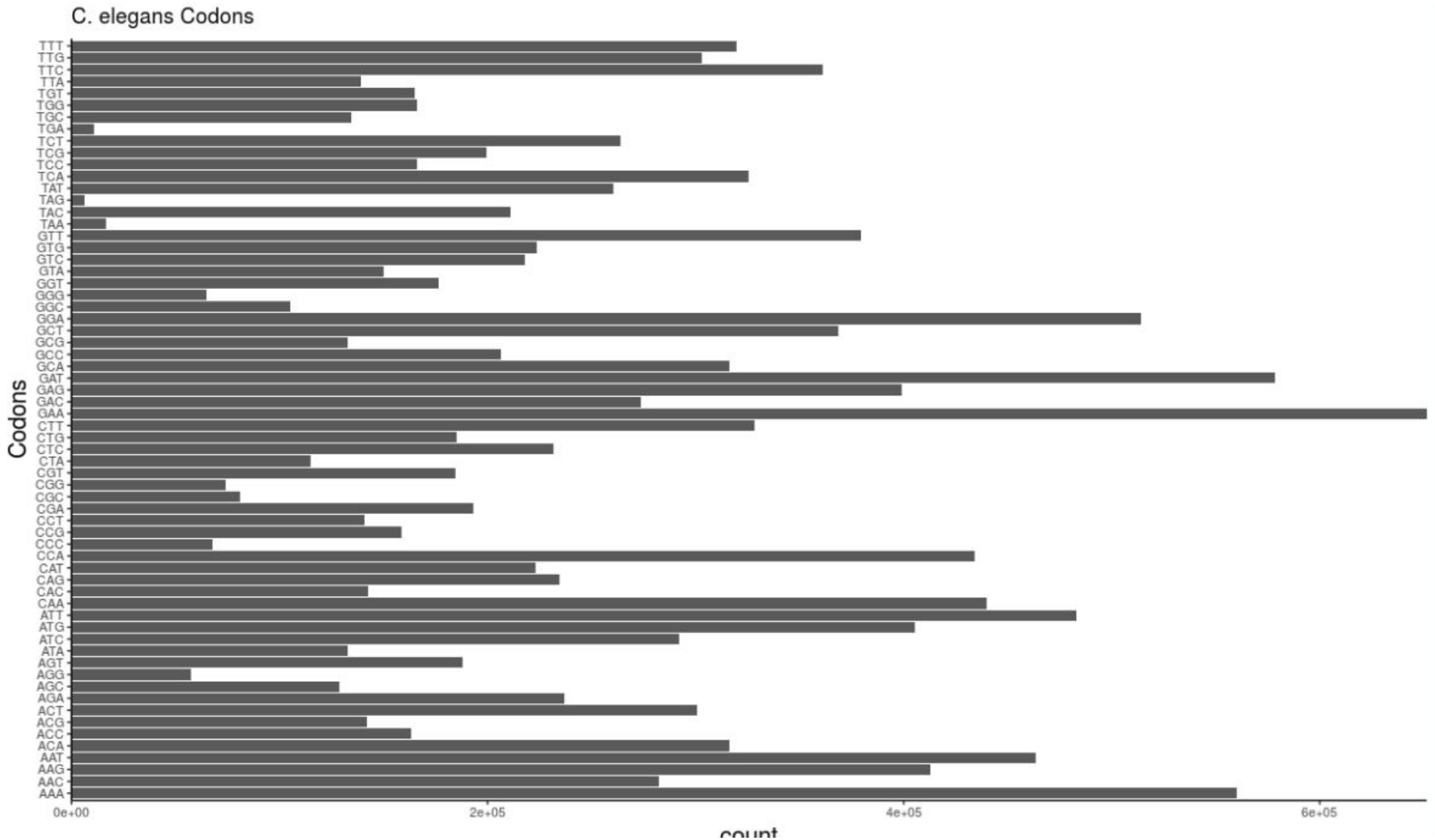
	Possible Codons	Count	Amino Acid
1	ATT	483230	Isoleucine
2	ATC	292020	Isoleucine
3	ATA	132854	Isoleucine
4	CTT	328405	Leucine
5	CTC	231679	Leucine
6	CTA	114953	Leucine
7	CTG	184936	Leucine

STEP 6: Combine bar graph with data frame of counts then filter to analyze specific relationships.

This is the final deliverable of the project. This includes 3 bar graphs and 3 data frames containing counts. Following these images is a discussion of conclusions and observations.


E. coli K-12 Codons
Codons
TTT
TTG
TTC
TTA
TGT
TGG
TGC
TGA
TCT
TCG
TCC
TCA
TAT
TAG
TAC
TAA
GTT
GTG
GTC
GTA
GGT
GGG
GGC
GGA
GCT
GCG
GCC
GCA
GAT
GAG
GAC
GAA
CTT
CTG
CTC
CTA
CGT
CGG
CGC
CGA
CCT
CCG
CCC
CCA
CAT
CAG
CAC
CAA
ATT
ATG
ATC
ATA
AGT
AGG
AGC
AGA
ACT
ACG
ACC
ACA
AAT
AAG
AAC
AAA
0
20000
40000
60000
count


Schizosaccharomyces pombe Codons
Codons
TTT
TTG
TTC
TTA
TGT
TGG
TGC
TGA
TCT
TCG
TCC
TCA
TAT
TAG
TAC
TAA
GTT
GTG
GTC
GTA
GGT
GGG
GGC
GGA
GCT
GCG
GCC
GCA
GAT
GAG
GAC
GAA
CTT
CTG
CTC
CTA
CGT
CGG
CGC
CGA
CCT
CCG
CCC
CCA
CAT
CAG
CAC
CAA
ATT
ATG
ATC
ATA
AGT
AGG
AGC
AGA
ACT
ACG
ACC
ACA
AAT
AAG
AAC
AAA
0
25000
50000
75000
100000
count


C. elegans Codons
Codons
TTT
TTG
TTC
TTA
TGT
TGG
TGC
TGA
TCT
TCG
TCC
TCA
TAT
TAG
TAC
TAA
GTT
GTG
GTC
GTA
GGT
GGG
GGC
GGA
GCT
GCG
GCC
GCA
GAT
GAG
GAC
GAA
CTT
CTG
CTC
CTA
CGT
CGG
CGC
CGA
CCT
CCG
CCC
CCA
CAT
CAG
CAC
CAA
ATT
ATG
ATC
ATA
AGT
AGG
AGC
AGA
ACT
ACG
ACC
ACA
AAT
AAG
AAC
AAA
0e+00
2e+05
4e+05
6e+05
count

Conclusions and Observations:

Have completed some background work on this subject before deciding how to approach the project, I knew that there is indeed a codon frequency bias present in human DNA. Now, after looking at the results from the counts of other organisms, it reaffirms this notion of a codon frequency bias when redundancy is involved. It understood that these biases are heavily influenced by the specificity of the amino acid and how likely it is to have another amino acid evolutionarily accepted in place of it (i.e. how conserved it is). For example, Tryptophan is highly conserved and has no redundant codons so there is no context for bias. The following table is the reference used when matching amino acids to possible codons:

Amino Acid	single letter code	3-letter code	DNA codons
Isoleucine	I	Ile	ATT, ATC, ATA
Leucine	L	Leu	CTT, CTC, CTA, CTG, TTA, TTG
Valine	V	Val	GTT, GTC, GTA, GTG
Phenylalanine	F	Phe	TTT, TTC
Methionine	M	Met (start)	ATG
Cysteine	C	Cys	TGT, TGC
Alanine	A	Ala	GCT, GCC, GCA, GCG
Glycine	G	Gly	GGT, GGC, GGA, GGG
Proline	P	Pro	CCT, CCC, CCA, CCG
Threonine	T	Thr	ACT, ACC, ACA, ACG
Serine	S	Ser	TCT, TCC, TCA, TCG, AGT, AGC
Tyrosine	Y	Tyr	TAT, TAC
Tryptophan	W	Trp	TGG
Glutamine	Q	Gln	CAA, CAG
Asparagine	N	Asn	AAT, AAC
Histidine	H	His	CAT, CAC
Glutamic acid	E	Glu	GAA, GAG
Aspartic acid	D	Asp	GAT, GAC
Lysine	K	Lys	AAA, AAG
Arginine	R	Arg	CGT, CGC, CGA, CGG, AGA, AGG
Stop codons	Stop	termination	TAA, TAG, TGA

On the other hand, as the table shows, Isoleucine falls in the middle in terms of the scale of codon redundancy and as the following tables show, there is a clear bias for the “ATT” codon.

E. coli K-12: Ratio of codon “ATT” to others = 40171 / 78847 = 0.50948

Possible Codons <chr>	Count <int>	Amino Acid <chr>
ATT	40171	Isoleucine
ATC	33331	Isoleucine
ATA	5345	Isoleucine

Schizosaccharomyces pombe: Ratio of codon “ATT” to others = 83822 / 147067 = 0.56996

Possible Codons <chr>	Count <int>	Amino Acid <chr>
ATT	83822	Isoleucine
ATC	29798	Isoleucine
ATA	33447	Isoleucine

C. elegans: Ratio of codon “ATT” to others = 483230 / 908104 = 0.53213

Possible Codons <chr>	Count <int>	Amino Acid <chr>
ATT	483230	Isoleucine
ATC	292020	Isoleucine
ATA	132854	Isoleucine

Another good example of bias would be to look at Arginine which can be encoded by 6 different codons, which makes it one of three amino acids with the highest observed redundancy. The following tables show that each organism will even favor 2 or 3 codon possibilities over the others:

E. coli K-12 : Ratio of “CGT” and “CGC” to others = 57144 / 72502 = 0.78817

Possible Codons <chr>	Count <int>	Amino Acid <chr>
CGT	27843	Arginine
CGC	29301	Arginine
CGA	4523	Arginine
CGG	6983	Arginine
AGA	2489	Arginine
AGG	1363	Arginine

Schizosaccharomyces pombe: Ratio of “CGT” and “AGA” to others = 63214 / 116223 = 0.54390

Possible Codons <chr>	Count <int>	Amino Acid <chr>
CGT	35798	Arginine
CGC	13981	Arginine
CGA	19478	Arginine
CGG	7163	Arginine
AGA	27416	Arginine
AGG	12387	Arginine

C. elegans: Ratio of “CGT”, “CGA”, and “AGA” to others = 614639 / 826560 = 0.74361

Possible Codons <chr>	Count <int>	Amino Acid <chr>
CGT	184369	Arginine
CGC	80770	Arginine
CGA	193322	Arginine
CGG	73957	Arginine
AGA	236948	Arginine
AGG	57194	Arginine

For this particular amino acid, it appears that there are at least 2 codons per organism that have a bias.

Although I have only scratched the surface of this topic of research but even at this depth, the clear underlying principles can be observed when applying the proper tools and programming capabilities.

Issues Encountered:

The first major issue was picking organisms that would both make the project well rounded but not require hours of computation. Most of the interesting organisms have very large genomes that would require more time than necessary but I am very pleased with the three I selected.

The next issue was shortly after the first. I was having trouble finding the best way to collect the data I wanted to use. I tried many different attempts to use the “rvest” package coupled with a CSS selector but couldn’t manage to isolate exactly what I needed.

The last major hurdle was developing the function that counts all the occurrences of the possible codons for the larger organisms. At first I struggled with passing the right number of iterations. The last part to get over was realizing I need to return the vector at the end or else the function would complete but return a NULL vector.

Points of Learning:

I learned quickly that using the simple method does not diminish the quality of the outcome if used properly when I had to resort to using a simple “curl_download()” function rather than a whole script of “rvest” commands.

Genomic data is a perfect example of a “big data” problem. I briefly worked with genomes that would be considered on the smaller side of the size spectrum and they still required longer than expected computation time.

There are so many well formed packages created for R by its community. When I came the “seqinr” package, I know I would be using these functions again as they had many applications to my desired field of Genomic Data Science.

Potential Expansion on Work:

Given more time, a better understanding of evolutionary effects on codon conservation, and perhaps a more powerful computer I would be happy to expand on this project in the future.

- I would like to add more organism’s CDSs to the database
- Compare the bias ratio within CDS to full genome sequence to see if the bias persists

References:

- http://m.ensembl.org/info/data/ftp/index.html
- https://www.rdocumentation.org/packages/seqinr
- http://fourier.eng.hmc.edu/bioinformatics/intro/node7.html
- http://www.biology-pages.info/G/GenomeSizes.html
- https://en.wikipedia.org