Index Hopping Project

Alexa Dowdell

Thu Sep 14 21:50:18 2017

Part 1-1 Plots: Generate per-nucleotide call distribution of quality scores

The following Python script `part1Hist1.py` was used to generate the data for the per-nucleotide distribution plots. Two very similar scripts were used ran seperately to accomodate the R1/R4 read lengths (100 bp) versus the R2/R3 index read lengths (8 bp)

```
#!usr/bin/env python

import argparse

parser = argparse.ArgumentParser(description="Quality Index Swapping")
parser.add_argument("-f", "--filepath", help="File", required=True, type=str)
args = parser.parse_args() #sets arguments as call-able

f = args.filepath

mean_scores = [0.0]*101 # create an empty array with values at ewach index
initialized to a float 0.0. Current script adapter for seq reads (101 bp),
the index read script would be adapted by changing 101 to 8.

def convert_phred(letter): #define a function to calculate phred score when
passed an ASCII character.
    n = ord(letter)-33
    return n

i = 0
with open(f, "r") as fh:
    for line in fh:
        line = line.strip("\n")
        i+=1
    if i%4 == 0: #if the current line being read is a quality score line
            j = 0
            for char in line:
                score = convert_phred(char) #convert each character in the line
                # to a phred score and add it to the appropriate index position
                # in the array mean_scores.
                mean_scores[j] = mean_scores[j] + score
                j+=1

for i in range(101): #for each index in the length of the mean_scores array
    mean_scores[i] = mean_scores[i]/363246735 # calculate mean by dividing by
    # total number of quality score lines.

with open("R1_Part1_dist_output.tsv", "w") as output:
    #write the distribution to a tsv file
    output.write("Base Position" + "\t" + "Mean Q_score" + "\n") # write header
    for i in range(101):
        output.write(str(i) + "\t" + str(mean_scores[i]) + "\n") #write new
        # line with contents for each index position.
```

The following shell script was submitted for each of the four files. The appropriate python script for each file, in this case R1, `part1Hist1.py` with the respective argparse file argument.

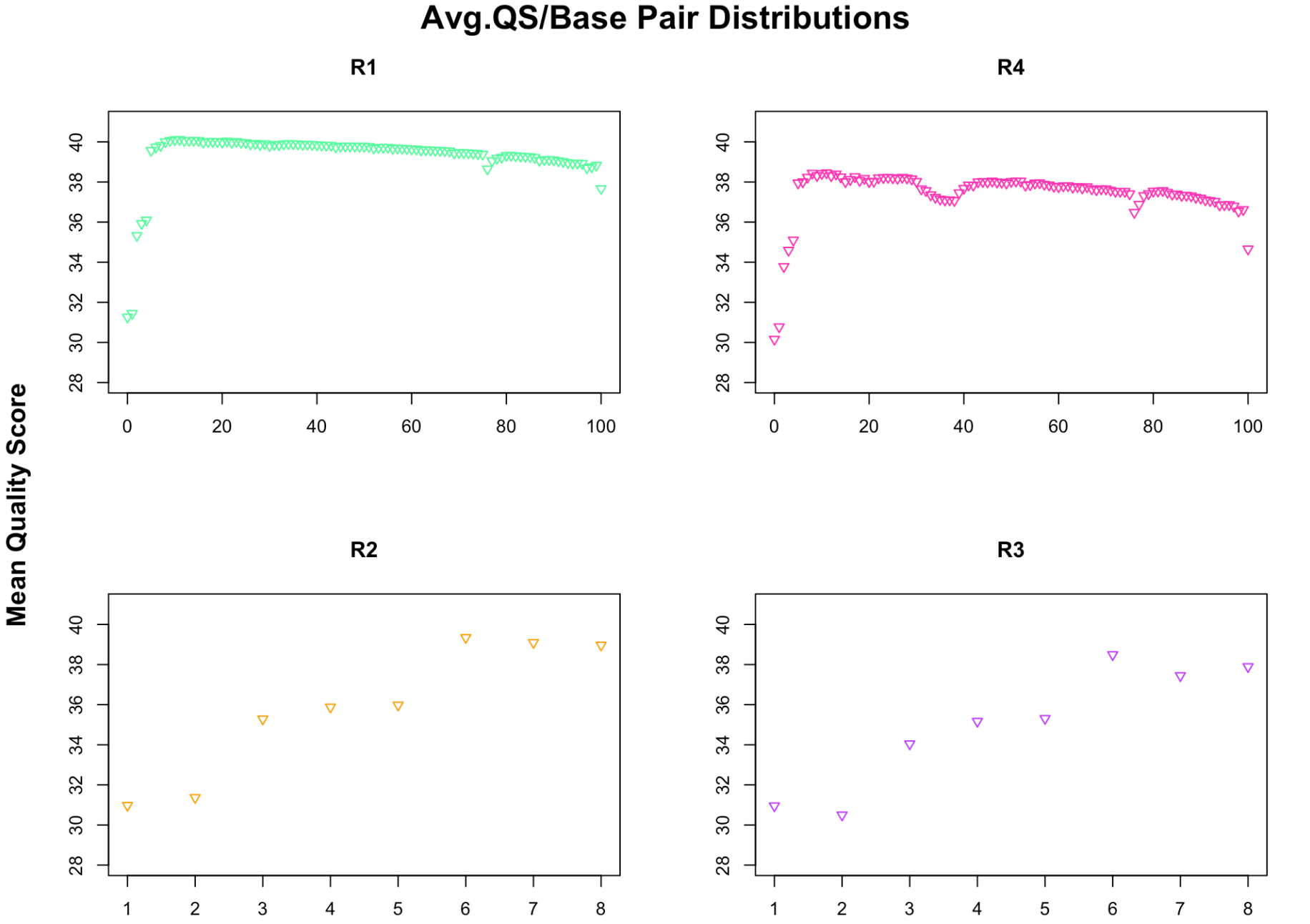
```
#!/bin/bash

#SBATCH --partition=long        ### Indicate want long node
#SBATCH --job-name=AD_R1     ### Job Name
#SBATCH --output=AD_R1.out         ### File in which to store job output
#SBATCH --error=AD_R1.err          ### File in which to store job error messages
#SBATCH --time=0-12:00:00   ### Wall clock time limit in Days-HH:MM:SS
#SBATCH --nodes=1                ### Node count required for the job
#SBATCH --ntasks-per-node=28     ### Nuber of tasks to be launched per Node

ml easybuild GCC/6.3.0-2.27 OpenMPI/2.0.2 Python/3.6.1

python part1Hist1.py -f 1294_S1_L008_R1_001.fastq
```

The files generated from the previous script `R1_Part1_dist_output.tsv`, `R2_Part1_dist_output.tsv`, `R3_Part1_dist_output.tsv`, `R4_Part1_dist_output.tsv` were imported into R as data frames from the appropriate directory. Then, four per base pair position distributions of mean quality score were generated for the R1, R2, R3, and R4 files.

```
getwd()
```

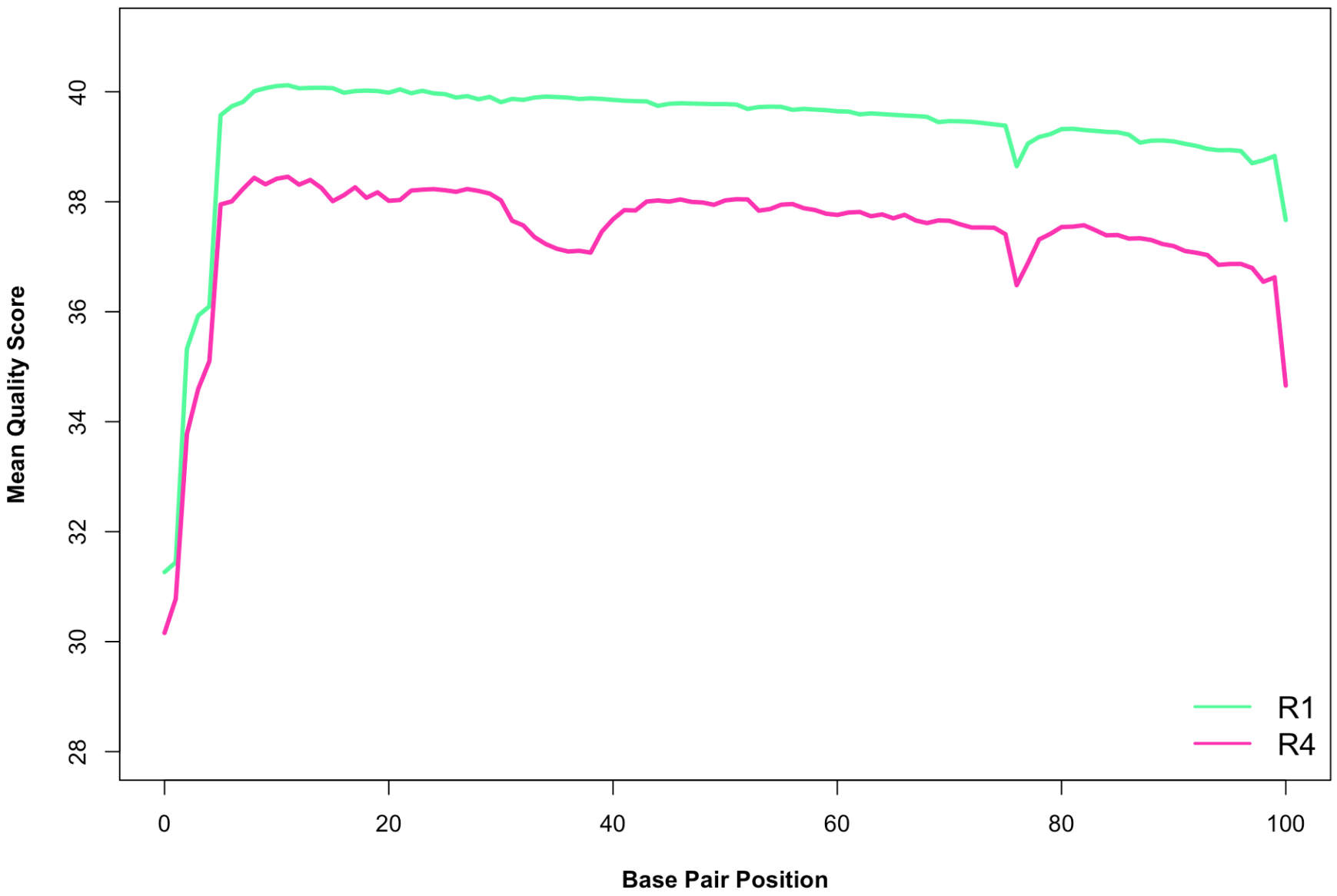
```
## [1] "/Users/adowdell/QualityIndexHopping/IndexHoppingPart1Plots"
```

```
setwd("/Users/adowdell/QualityIndexHopping/IndexHoppingPart1Plots")

R1_basepair <- read.table("R1_Part1_dist_output.tsv", sep = "\t", header = TRUE)
R2_basepair <- read.table("R2_Part1_dist_output.tsv", sep = "\t", header = TRUE)
R3_basepair <- read.table("R3_Part1_dist_output.tsv", sep = "\t", header = TRUE)
R4_basepair <- read.table("R4_Part1_dist_output.tsv", sep = "\t", header = TRUE)

par(mfrow = c(2, 2), oma = c(0, 1, 1, 0))
plot(R1_basepair$Mean.Q_score ~ R1_basepair$Base.Position, main = "R1", pch = 6,
    cex = 0.9, col = "seagreen1", xlab = "", ylab = "", ylim = c(28, 41))

plot(R4_basepair$Mean.Q_score ~ R4_basepair$Base.Position, main = "R4", pch = 6,
    cex = 0.9, col = "maroon1", xlab = "", ylab = "", ylim = c(28, 41))

basenames <- c(1, 2, 3, 4, 5, 6, 7, 8)  #for proper x-axis labels

plot(R2_basepair$Mean.Q_score[0:8] ~ R2_basepair$Base.Position[0:8], main = "R2",
    pch = 6, cex = 0.9, col = "orange", xlab = "", ylab = "", xaxt = "n", ylim = c(28
,
        41))
axis(side = 1, at = seq(0, 7, by = 1), labels = basenames)

plot(R3_basepair$Mean.Q_score[0:8] ~ R3_basepair$Base.Position[0:8], main = "R3",
    pch = 6, cex = 0.9, col = "darkorchid1", xlab = "", ylab = "", xaxt = "n", ylim =
c(28,
        41))
axis(side = 1, at = seq(0, 7, by = 1), labels = basenames)


mtext("Avg.QS/Base Pair Distributions", side = 3, font = 2, outer = TRUE, line = -0.5
,
    cex = 1.5)
mtext("Mean Quality Score", side = 2, font = 2, outer = TRUE, line = -0.5, cex = 1.2)
mtext("Base Pair Position", side = 1, font = 2, outer = TRUE, line = -1, cex = 1.2)
```



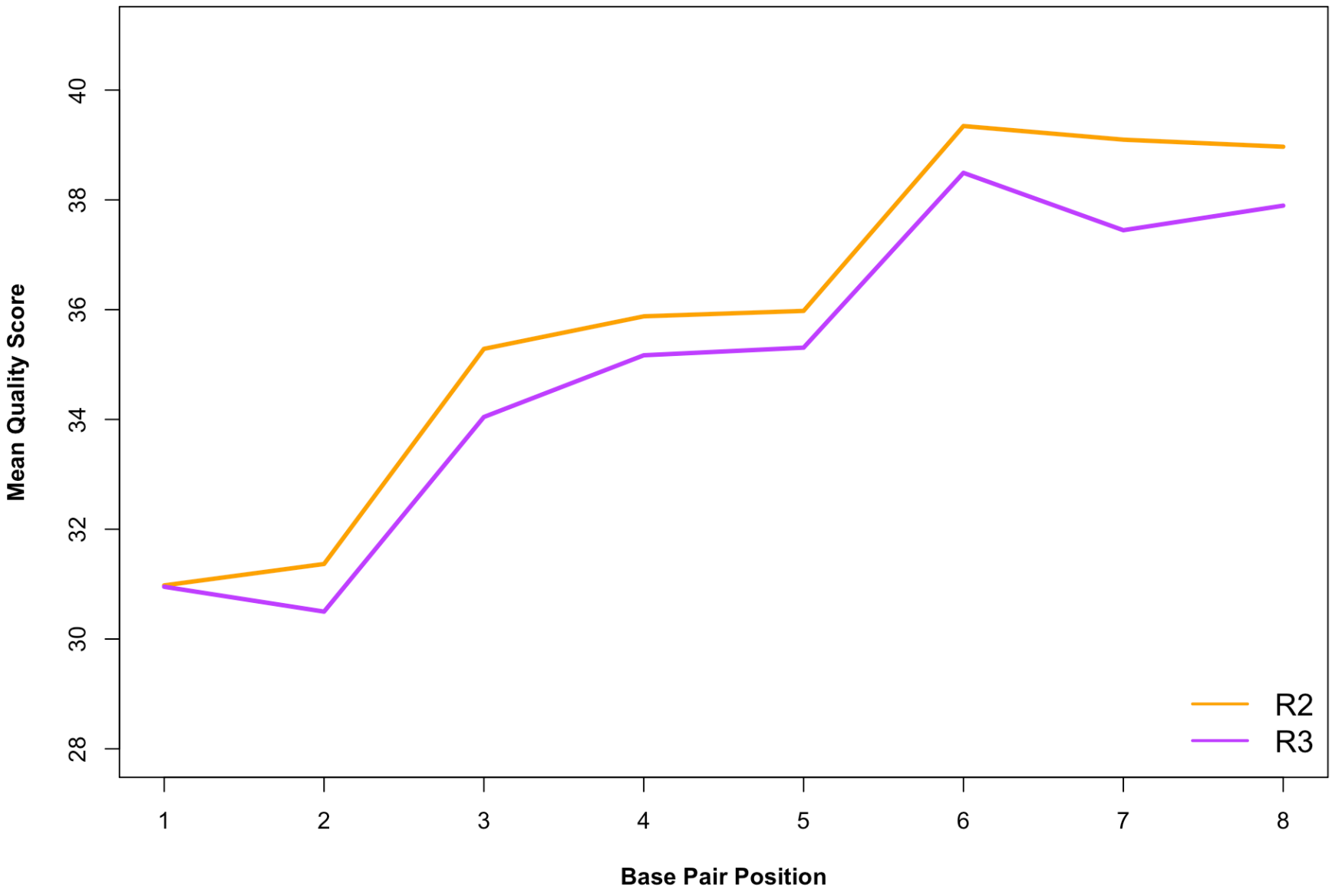
Overlayed the R1,R4 sequence distributions for comparison of average quality score per base position for sequence reads.

```
par(mfrow = c(1, 1), oma = c(0, 1, 1, 0))
plot(R1_basepair$Mean.Q_score ~ R1_basepair$Base.Position, main = "R1, R4 Mean QS/Bas
e Pair Position",
    pch = 6, cex = 0.9, col = "seagreen1", xlab = expression(bold("Base Pair Position
")),
    ylab = expression(bold("Mean Quality Score")), type = "l", lwd = 3, ylim = c(28,
        41))
points(R4_basepair$Mean.Q_score ~ R4_basepair$Base.Position, pch = 6, cex = 0.9,
    col = "maroon1", xlab = "", ylab = "", type = "l", lwd = 3)
legend("bottomright", c("R1", "R4"), col = c("seagreen1", "maroon1"), bty = "n",
    lty = c(1, 1), lwd = c(2, 2), cex = 1.3)
```



Overlayed the R2, R3 sequence distributions for comparison of average quality score per base position for index reads.

```
par(mfrow = c(1, 1), oma = c(0, 1, 1, 0))
plot(R2_basepair$Mean.Q_score[0:8] ~ R2_basepair$Base.Position[0:8], main = "R2, R3 M
ean QS/Base Pair Position",
    pch = 6, cex = 0.9, col = "orange", xlab = expression(bold("Base Pair Position"))
,
    ylab = expression(bold("Mean Quality Score")), type = "l", lwd = 3, xaxt = "n",
    ylim = c(28, 41))
axis(side = 1, at = seq(0, 7, by = 1), labels = basenames)
points(R3_basepair$Mean.Q_score[0:8] ~ R3_basepair$Base.Position[0:8], main = "R3",
    pch = 6, cex = 0.9, col = "darkorchid1", xlab = "", ylab = "", type = "l", lwd =
3)
legend("bottomright", c("R2", "R3"), col = c("orange", "darkorchid1"), bty = "n",
    lty = c(1, 1), lwd = c(2, 2), cex = 1.3)
```



b. What is a good quality score cutoff for index reads and pairs to utilize for sample identification and downstream analysis, respectively?

The lowest mean quality score for index reads (R2, R3) is 30.499 at the second position of the R3 index file. A good quality cutoff in terms of a point in which the average quality score of each read does not meet the cutoff, the entire read is discarded should be below the lowest average quality score of the index reads. Therefore, considering 30.499 is the lowest mean quality score, 30 seems like an acceptable cutoff for sample identification and downstream analysis.

c. How many indexes have Undetermined (N) base calls? (Utilize your command line tool knowledge. Summit the command you used. CHALLENGE: use a one line command)

```
cat 1294_S1_L008_R2_001.fastq | awk 'NR %4 == 2' | grep -c "N"
3976613

cat 1294_S1_L008_R3_001.fastq | awk 'NR %4 == 2' | grep -c "N"
3328051
```

R2 has 3976613 indexes that contain Undetermined (N) base calls.

R3 has 3328051 indexes that contain Undetermined (N) base calls.

Part 1-2: Plot the frequency distribution of avg. quality scores

A new python script called `part2FreqDistFixed.py` was written to calculate frequency distributions of average quality scores for each of the four files as shown below.

```
#!usr/bin/env python

import argparse

parser = argparse.ArgumentParser(description="Quality Index Swapping")
parser.add_argument("-f1", "--filepath1", help="R1 file", required=True, type=str)
parser.add_argument("-f2", "--filepath2", help="R2 file", required=True, type=str)
parser.add_argument("-f3", "--filepath3", help="R3 file", required=True, type=str)
parser.add_argument("-f4", "--filepath4", help="R4 file", required=True, type=str)

args = parser.parse_args() #sets arguments as call-able

f1 = args.filepath1
f2 = args.filepath2
f3 = args.filepath3
f4 = args.filepath4

# f1 = "./1294_S1_L008_R1_001_sample.fastq"
# f2 = "./1294_S1_L008_R2_001_sample.fastq"
# f3 = "./1294_S1_L008_R3_001_sample.fastq"
# f4 = "./1294_S1_L008_R4_001_sample.fastq"

def convert_phred(char):
    n = ord(char) - 33
    return n

def read_frequency(file, index, n):
    phredscore_dict = {}

    if index == True:
        length = 8
    else:
        length = 101

    # add up quality scores
    with open(file) as fh:
        i = 0  # line counter
        for line in fh:
            i += 1
            line = line.strip("\n")
            if i % 4 == 0:  # picks out only quality score lines
                sum1 = 0
```

```
                for char in line:
                    sum1 += ord(char) - 33
                mean = int(sum1 / length)
                if mean in phredscore_dict:
                    phredscore_dict[mean] += 1
                else:
                    phredscore_dict[mean] = 1
    with open("R" + str(n) + "_part2Plot.txt", "w") as output:
        for key in phredscore_dict:
            output.write(str(key) + "\t" + str(phredscore_dict[key])+"\n")

read_frequency(f1, False,1)
read_frequency(f2, True,2)
read_frequency(f3, True,3)
read_frequency(f4, False,4)
```

The following shell script was submitted which calls the script and passes four file arguments for R1, R2, R3, and R4.

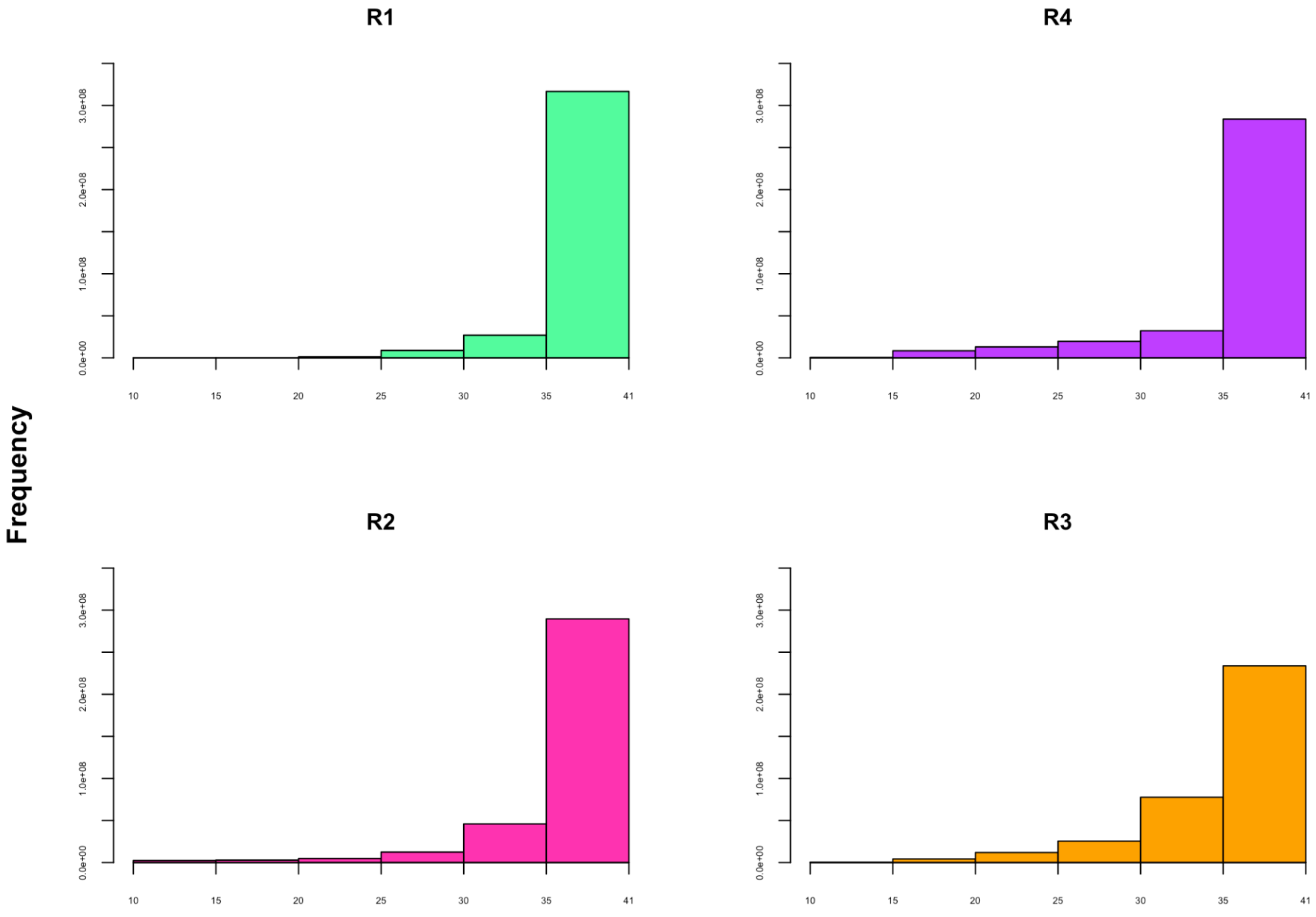
```
#!/bin/bash

#SBATCH --partition=long       ### Indicate want long node
#SBATCH --job-name=AD_freq     ### Job Name
#SBATCH --output=AD_freq.out         ### File in which to store job output
#SBATCH --error=AD_freq.err          ### File in which to store job error messages
#SBATCH --time=2-00:00:00   ### Wall clock time limit in Days-HH:MM:SS
#SBATCH --nodes=1                ### Node count required for the job
#SBATCH --ntasks-per-node=28     ### Nuber of tasks to be launched per Node

ml easybuild GCC/6.3.0-2.27 OpenMPI/2.0.2 Python/3.6.1
pip list installed | grep numpy

python part2FreqDistFixed.py -f1 1294_S1_L008_R1_001.fastq -f2 \
1294_S1_L008_R2_001.fastq \
-f3 1294_S1_L008_R3_001.fastq -f4 1294_S1_L008_R4_001.fastq
```

The four files output from `part2FreqDistFixed.py`: `R1_part2Plot.txt`, `R2_part2Plot.txt`, `R3_part2Plot.txt` and `R4_part2Plot.txt` were read into R as data frames.

```
R1_counts <- read.table("R1_part2Plot.txt", sep = "\t", header = TRUE)
R2_counts <- read.table("R2_part2Plot.txt", sep = "\t", header = TRUE)
R3_counts <- read.table("R3_part2Plot.txt", sep = "\t", header = TRUE)
R4_counts <- read.table("R4_part2Plot.txt", sep = "\t", header = TRUE)

headers <- c("QS", "Counts")  #assign column headers
colnames(R1_counts) <- headers
colnames(R2_counts) <- headers
colnames(R3_counts) <- headers
colnames(R4_counts) <- headers
```

The four files were utilized to generate four mean quality score frequency distributions.

```
par(mfrow = c(2, 2), oma = c(0, 1, 1, 0))

bins <- c(10, 15, 20, 25, 30, 35, 41)

R1_counts$buckets = cut(R1_counts$QS, breaks = bins)
R1_new = aggregate(Counts ~ buckets, data = R1_counts, sum)
barplot(R1_new$Counts, space = 0, width = 1, xlim = c(0, 6), ylim = c(0, 3.5e+08),
    ylab = "", xlab = "", col = "seagreen1", main = "R1", cex.axis = 0.5)
axis(1, at = 0:6, labels = bins, cex.axis = 0.5)


R4_counts$buckets = cut(R4_counts$QS, breaks = bins)
R4_new = aggregate(Counts ~ buckets, data = R4_counts, sum)
barplot(R4_new$Counts, space = 0, width = 1, xlim = c(0, 6), ylim = c(0, 3.5e+08),
    ylab = "", xlab = "", col = "darkorchid1", main = "R4", cex.axis = 0.5)
axis(1, at = 0:6, labels = bins, cex.axis = 0.5)


R2_counts$buckets = cut(R2_counts$QS, breaks = bins)
R2_new = aggregate(Counts ~ buckets, data = R2_counts, sum)
barplot(R2_new$Counts, space = 0, width = 1, xlim = c(0, 6), ylim = c(0, 3.5e+08),
    ylab = "", xlab = "", col = "maroon1", main = "R2", cex.axis = 0.5)
axis(1, at = 0:6, labels = bins, cex.axis = 0.5)


R3_counts$buckets = cut(R3_counts$QS, breaks = bins)
R3_new = aggregate(Counts ~ buckets, data = R3_counts, sum)
barplot(R3_new$Counts, space = 0, width = 1, xlim = c(0, 6), ylim = c(0, 3.5e+08),
    ylab = "", xlab = "", col = "orange", main = "R3", cex.axis = 0.5)
axis(1, at = 0:6, labels = bins, cex.axis = 0.5)

mtext("Avg QS Frequency Distributions", side = 3, font = 2, outer = TRUE, line = -0.5
,
    cex = 1.5)
mtext("Frequency", side = 2, font = 2, outer = TRUE, line = -0.5, cex = 1.2)
mtext("Avg Quality Score", side = 1, font = 2, outer = TRUE, line = -1, cex = 1.2)
```



d. What do the averaged Quality Scores across the reads tell you? Interpret your data specifically.

Among the four frequency distributions, majority of the reads have a mean quality score over 35. The R1 file contains the highest proportion of mean quality score reads with average scores over 35. On the other end of the spectrum, the R3 file contains the lowest proportion of mean quality score reads over 35. Overall, majority of mean quality score reads fall in the category of a mean quality score of 25 or greater. Note: there do seem to be a number of reads in each of the four files with mean quality scores less than 25, and in R2, R3, and R4, a mean quality score less than 20.

Part 2: De-multiplex the samples and document index swapping and number of reads retained per sample

Cutoff -30

Write a program to de-multiplex the samples and document index swapping and number of reads retained per sample.

The following Python script `outputTableFixed30-2.py` was written to demultiplex samples and track index hopping as well as relavent statistics.

```
#!usr/bin/env python

############################## Format argparse ################################
# 1. Write script parameters to take in the four files as well as the index.tsv
# file and a min quality score cut off.

import argparse
import numpy as np

parser = argparse.ArgumentParser(description="Quality Index Swapping")
parser.add_argument("-f1", "--filepath1", help="R1 file", required=True, type=str)
parser.add_argument("-f2", "--filepath2", help="R2 file", required=True, type=str)
parser.add_argument("-f3", "--filepath3", help="R3 file", required=True, type=str)
parser.add_argument("-f4", "--filepath4", help="R4 file", required=True, type=str)
parser.add_argument("-f5", "--filepath5", help="Indexes.txt file", required =True,
type=str)
parser.add_argument("-c", "--cutoff", help="Min quality score cutoff",
required=True, type=int)



args = parser.parse_args() #sets arguments as call-able
f1 = args.filepath1
f2 = args.filepath2
f3 = args.filepath3
f4 = args.filepath4
f5 = args.filepath5
c = args.cutoff

# f1 = "./1294_S1_L008_R1_001_sample.fastq"
# f2 = "./1294_S1_L008_R2_001_sample.fastq"
# f3 = "./1294_S1_L008_R3_001_sample.fastq"
# f4 = "./1294_S1_L008_R4_001_sample.fastq"
# f5 = "./indexes2.txt"
# c = 0

# when passed a ACSCII character, convert to phred score

def convert_phred(letter):
    n = ord(letter) - 33
    return n


###### Build dictionary containing every 4 nucleotide seq combination ########

# Every possible combo of 2, 8 base long indexes as key and value = counter
# (increases by 1 every time seq is uncovered in index reads R2, R3)
nIndexDict = {}
```

```
indexComboDict = {}
unmatchedIndexDict = {}
list_index_reads = [] #list holding index reads from said index file

with open(f5) as fh: #open tsv file, set to fh
    for line in fh:
        col_values = line.strip().split("\t") # strip and split by tab for each
        #value in both columns
        index_reads = (col_values[4]) #grab index reads
        list_index_reads.append(index_reads) # put into list_index_reads array
    #print(list_index_reads)

# Generate all possible combinations of 2 index values and store in dict with
#value of 0
for currentLine in list_index_reads:
    for index in list_index_reads: #call all indexes in array
        if currentLine == index: #if the indexes match
            indexComboDict[currentLine + "_" + index]=0 # set indexes to 0
            #[index1, index2:0] in expected index dictionary
        else:
            unmatchedIndexDict[currentLine + "_" + index]=0 # add mismatched
            #indexes to index swapping dictionary


######## Read in four files and assign corresponding lines to variables #########

# Initialize variables for lines being read in

headerR1 = ""
headerR2 = ""
headerR3 = ""
headerR4 = ""

seqR1 = ""
seqR2 = ""
seqR3 = ""
seqR4 = ""

plusR1 = ""
plusR2 = ""
plusR3 = ""
plusR4 = ""

phredR1 = ""
phredR2 = ""
phredR3 = ""
phredR4 = ""

#initialize counters
N_cntr = 0
```

```
matched_cntr = 0
indexHopped_cntr = 0
sequencing_error_cntr = 0
retained_cntr = 0
discarded_cntr = 0
total_cntr = 0

#reverse complement function for R3 indexes
def rc(dna):
    seq_dict = {'A':'T','T':'A','G':'C','C':'G', 'N':'N'}
    return "".join([seq_dict[base] for base in reversed(dna)])

with open(f1, "rt") as fh_R1, open(f2, "rt") as fh_R2, open(f3) as fh_R3, open(f4) as
fh_R4:
# open each of the four files
    for line in fh_R1:
    # for each line, strip the line and assign appropriate variable
        line = line.strip("\n")
        headerR1 = line.strip("\n")
        seqR1 = fh_R1.readline().strip("\n")
        plusR1 = fh_R1.readline().strip("\n")
        phredR1 = fh_R1.readline().strip("\n")

        headerR2 = fh_R2.readline().strip("\n")
        seqR2 = fh_R2.readline().strip("\n")
        plusR2 = fh_R2.readline().strip("\n")
        phredR2 = fh_R2.readline().strip("\n")

        headerR3 = fh_R3.readline().strip("\n")
        seqR3 = rc(fh_R3.readline().strip("\n"))
        plusR3 = fh_R3.readline().strip("\n")
        phredR3 = fh_R3.readline().strip("\n")

        headerR4 = fh_R4.readline().strip("\n")
        seqR4 = fh_R4.readline().strip("\n")
        plusR4 = fh_R4.readline().strip("\n")
        phredR4 = fh_R4.readline().strip("\n")

        val = seqR2 + "_" + seqR3

        total_cntr+=1 #increment counter for total lines read

        if total_cntr%500000000: #for debugging on Talapas, print progress
            print("On line: ", total_cntr, flush = True)

        score = []
        for base in range(len(phredR2)):   #for each character in the length of
        # the index read
            score.append(convert_phred(phredR2[base])) #calculate phred scores
            # for each index position
```

```
        score.append(convert_phred(phredR3[base]))

    if all(items >= c for items in score): #if all phred scores are greater
    # than cutoff retain read
        retained_cntr+=1  #increment reads retained counter
        if val in indexComboDict: # if the index pair in expected index
        #dictionary
            indexComboDict[val] += 1 #increment value of associated key
            matched_cntr += 1 #increment matched counter for expected

        if val in unmatchedIndexDict:  # if index pair in index swapped
        #dictionary
            unmatchedIndexDict[val] += 1 # increment value of the associated key
            indexHopped_cntr += 1 #increment index hopped counter

        if "N" in val: #if the index pair contains even 1 "N" character
            if val in nIndexDict: #if the index pair is already in the
            # undetermined index pair dictionary
                nIndexDict[val]+=1 #increment value of associated key
                N_cntr += 1 # increment undetermined N counter
            else:
                nIndexDict[val]=1 #Otherwise add the index pair as a new key
                # to the dictionary with a value 1
                N_cntr += 1 # increment undetermined N counter

            if val not in indexComboDict and val not in unmatchedIndexDict and
            val not in nIndexDict:
                # if the index pair isnt in any of the three dictionaries checked
                above, increment sequencing error counter
                sequencing_error_cntr += 1

        else:
            # if the read is not retained increment the discarded read counter
            discarded_cntr += 1


with open("1294_S1_L008_R"+ "_Output_table_Fixed_30-2", 'w') as output1:
# open a file to write to with a name specified
    j = 0
    k = 0
    m = 0
    #output all counters
    output1.write("\n" + "Number of Undetermined N Reads: " + str(N_cntr))
    output1.write("\n" + "Number of Expected Paired Index Reads : " +
    str(matched_cntr))
    output1.write("\n" + "Number of Index Swapped Reads: " + str(indexHopped_cntr))
    output1.write("\n" + "Number of Sequencing Error Reads: " +
    str(sequencing_error_cntr))
    output1.write("\n" + "Number of Retained Reads: " + str(retained_cntr))
    output1.write("\n" + "Number of Discarded Reads: " + str(discarded_cntr))
```

```
output1.write("\n" + "Total Number of Reads: " + str(total_cntr))
output1.write("\n" + "Percentage of Reads Index Swapped: " +
str((indexHopped_cntr/total_cntr)*100))
output1.write("\n" + "Percentage of Expected Paired Index Reads: " +
str((matched_cntr/total_cntr)*100))
output1.write("\n" + "Cutoff: " + str(c) + "\n")
output1.write("\n")
output1.write("Dual Indexed Pairs" + "\t" + "Counts" + "\n")
for i in indexComboDict: #output contents for each of the three dictionaries
    output1.write(str(i) + "\t" + str(indexComboDict[i]) + "\n")
    j+=1
output1.write("\n")
output1.write(("Index Hopped Pairs" + "\t" + "Counts" + "\n"))
for i in unmatchedIndexDict:
    output1.write(str(i) + "\t" + str(unmatchedIndexDict[i]) + "\n")
    k+=1
output1.write("\n")
output1.write(("N Indexed Pairs" + "\t" + "Counts" + "\n"))
for i in nIndexDict:
    output1.write(str(i) + "\t" + str(nIndexDict[i]) + "\n")
    m+=1
```

The following shell script `outputTableFixed30.srun` was created to call `outputTableFixed30-2.py` and the associated argparse parameters were passed. Note: `indexes2.txt` is a copy of `indexes.txt` with the header removed so the header didn't mistakenly get passed as an index and incorporated into the index dictionaries as additional keys.

```
#!/bin/bash

#SBATCH --partition=gpu       ### Indicate want long node
#SBATCH --job-name=AD_0Fixed     ### Job Name
#SBATCH --output=AD_0Fixed.out         ### File in which to store job output
#SBATCH --error=AD_0Fixed.err     ### File in which to store job error messages
#SBATCH --time=0-24:00:00   ### Wall clock time limit in Days-HH:MM:SS
#SBATCH --nodes=1               ### Node count required for the job
#SBATCH --ntasks-per-node=28     ### Nuber of tasks to be launched per Node

ml easybuild GCC/6.3.0-2.27 OpenMPI/2.0.2 Python/3.6.1
pip list installed | grep numpy

python outputTableFixed30.py -f1 1294_S1_L008_R1_001.fastq -f2 \
1294_S1_L008_R2_001.fastq -f3 1294_S1_L008_R3_001.fastq -f4 \
1294_S1_L008_R4_001.fastq -f5 indexes2.txt -c 30
```

The output file `1294_S1_L008_R_Output_table_Fixed_30-2` contained statistics for the four files at a cutoff of 30 as well as the contents of the expected index pairs, index swapped pairs, and undetermined (N) index pairs dictionaries.

The first 50 lines of `1294_S1_L008_R_Output_table_Fixed_30-2` are displayed below:

Number of Undetermined N Reads: 0
Number of Expected Paired Index Reads : 226715602
Number of Index Swapped Reads: 330975
Number of Sequencing Error Reads: 5181567
Number of Retained Reads: 232228144
Number of Discarded Reads: 131018591
Total Number of Reads: 363246735
Percentage of Reads Index Swapped: 0.09111575359376596
Percentage of Expected Paired Index Reads: 62.41366546625671
Cutoff: 30

Dual Indexed Pairs	Counts
GTAGCGTA_GTAGCGTA	5774439
CGATCGAT_CGATCGAT	4237854
GATCAAGG_GATCAAGG	4628196
AACAGCGA_AACAGCGA	6368144
TAGCCATG_TAGCCATG	7148153
CGGTAATC_CGGTAATC	2393021
CTCTGGAT_CTCTGGAT	24515042
TACCGGAT_TACCGGAT	49686878
CTAGCTCA_CTAGCTCA	13034311
CACTTCAC_CACTTCAC	2577666
GCTACTCT_GCTACTCT	4301318
ACGATCAG_ACGATCAG	5933528
TATGGCAC_TATGGCAC	7651472
TGTTCCGT_TGTTCCGT	11450554
GTCCTAAG_GTCCTAAG	6200133
TCGACAAG_TCGACAAG	2644260
TCTTCGAC_TCTTCGAC	30089661
ATCATGCG_ATCATGCG	6927867
ATCGTGGT_ATCGTGGT	4730009
TCGAGAGT_TCGAGAGT	7448072
TCGGATTC_TCGGATTC	2874320
GATCTTGC_GATCTTGC	2636332
AGAGTCCA_AGAGTCCA	7602663
AGGATAGC_AGGATAGC	5861709

Index Hopped Pairs	Counts
GTAGCGTA_CGATCGAT	71
GTAGCGTA_GATCAAGG	94
GTAGCGTA_AACAGCGA	146
GTAGCGTA_TAGCCATG	101
GTAGCGTA_CGGTAATC	58
GTAGCGTA_CTCTGGAT	346
GTAGCGTA_TACCGGAT	743
GTAGCGTA_CTAGCTCA	333
GTAGCGTA_CACTTCAC	45
GTAGCGTA_GCTACTCT	902
GTAGCGTA_ACGATCAG	429

a. How many reads are retained for each expected index pair? What is the percentage?

Let's clarify a retained read qualifies as a read which passes the designated cutoff (30, in this case). There were 232228144 total retained reads, 226715602 of which were retained reads for expected index pairs. 62.41% of the total reads were accounted for with expected index pairs that passed a cutoff of 30 and were retained. Below is the distribution of retained reads in the expected index pairs dictionary with the associated percentages out of total retained reads as well as total reads.

```
expected_Indices <- read.table("expectedOutputTableFixed30.tsv", sep = "\t", header =
TRUE)

expected_Indices$percent_retained <- expected_Indices$Counts/232228144 * 100
expected_Indices$percent_total <- expected_Indices$Counts/363246735 * 100
headersExp <- c("Expected Index Pairs", "Counts", "% Retained", "% Total")
colnames(expected_Indices) <- headersExp

library(pander)
pander(expected_Indices)
```

Expected Index Pairs	Counts	% Retained	% Total
GTAGCGTA_GTAGCGTA	5774439	2.487	1.59
CGATCGAT_CGATCGAT	4237854	1.825	1.167
GATCAAGG_GATCAAGG	4628196	1.993	1.274
AACAGCGA_AACAGCGA	6368144	2.742	1.753
TAGCCATG_TAGCCATG	7148153	3.078	1.968
CGGTAATC_CGGTAATC	2393021	1.03	0.6588
CTCTGGAT_CTCTGGAT	24515042	10.56	6.749
TACCGGAT_TACCGGAT	49686878	21.4	13.68
CTAGCTCA_CTAGCTCA	13034311	5.613	3.588
CACTTCAC_CACTTCAC	2577666	1.11	0.7096
GCTACTCT_GCTACTCT	4301318	1.852	1.184
ACGATCAG_ACGATCAG	5933528	2.555	1.633
TATGGCAC_TATGGCAC	7651472	3.295	2.106
TGTTCCGT_TGTTCCGT	11450554	4.931	3.152
GTCCTAAG_GTCCTAAG	6200133	2.67	1.707
TCGACAAG_TCGACAAG	2644260	1.139	0.728
TCTTCGAC_TCTTCGAC	30089661	12.96	8.284

ATCATGCG_ATCATGCG	6927867	2.983	1.907
ATCGTGGT_ATCGTGGT	4730009	2.037	1.302
TCGAGAGT_TCGAGAGT	7448072	3.207	2.05
TCGGATTC_TCGGATTC	2874320	1.238	0.7913
GATCTTGC_GATCTTGC	2636332	1.135	0.7258
AGAGTCCA_AGAGTCCA	7602663	3.274	2.093
AGGATAGC_AGGATAGC	5861709	2.524	1.614

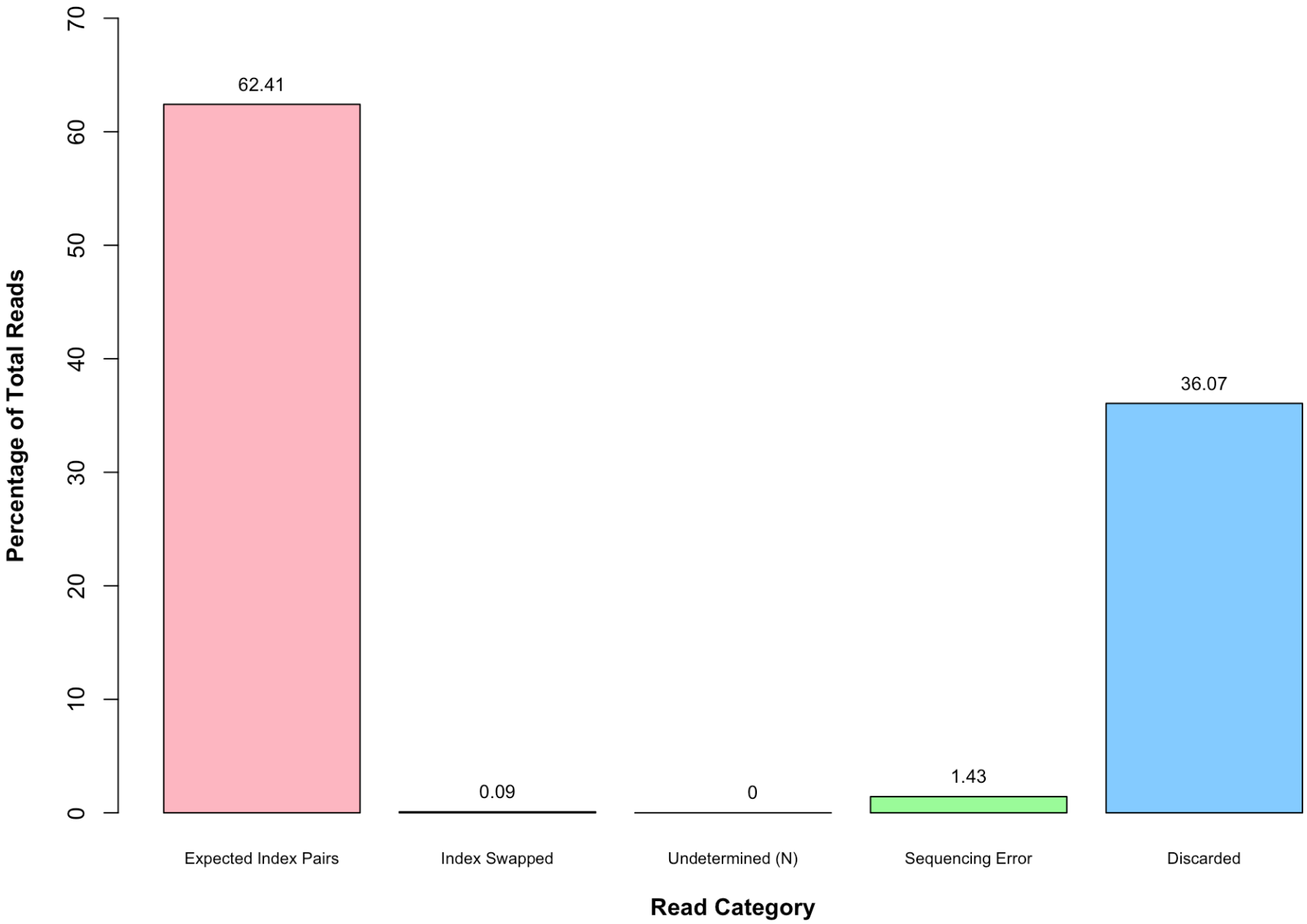
b. How many reads are indicative of index swapping?

330975 reads are indicative of index swapping which correlates to 0.0911% of total reads. The distribution of swapped index pairs with their associated counts of retained reads can be view under the `outputfiles` directory as `swappedOutputTableFixed.tsv` .

c. Create a distribution of swapped indexes. What does this histogram tell you/what is your interpretation of this data?

The plot below displays the general distributions of total reads, first if they were discarded, then if they were retained whether the read was an expected index pair, index swapped pair, undetermined (N) index pair, or sequencing error. Out of the 363246735 total reads, 36.07% were discarded for not meeting a cutoff threshold of 30. Of the remaining 63.93% of retained reads, 62.41% of the retained reads (226715602 reads) were expected index pair reads. 0.09% of retained reads (330975 reads) were index swapped reads, and the cutoff of 30 removed all undetermined (N) index pair reads.

```
overall_data = data.frame(c("Expected Index Pairs", "Index Swapped", "Undetermined (N
)",
    "Sequencing Error", "Discarded"), c(226715602, 330975, 0, 5181567, 131018591))

overall_data$Percentage <- overall_data[, 2]/sum(overall_data[, 2]) * 100

colnames(overall_data) = c("Category", "Counts", "Percentage")

barplot(overall_data$Percentage, names.arg = overall_data$Category, ylim = c(0, 70),
    xlab = expression(bold("Read Category")), ylab = expression(bold("Percentage of T
otal Reads")),
    cex.names = 0.7, main = "Cutoff 30 - Read Distribution After Demultiplexing",
    col = c("lightpink", "mediumpurple1", "coral", "palegreen1", "skyblue1"))
text(x = c(0.7, 1.9, 3.2, 4.3, 5.5), y = overall_data$Percentage, label = round(overa
ll_data$Percentage,
    2), pos = 3, cex = 0.8)
```

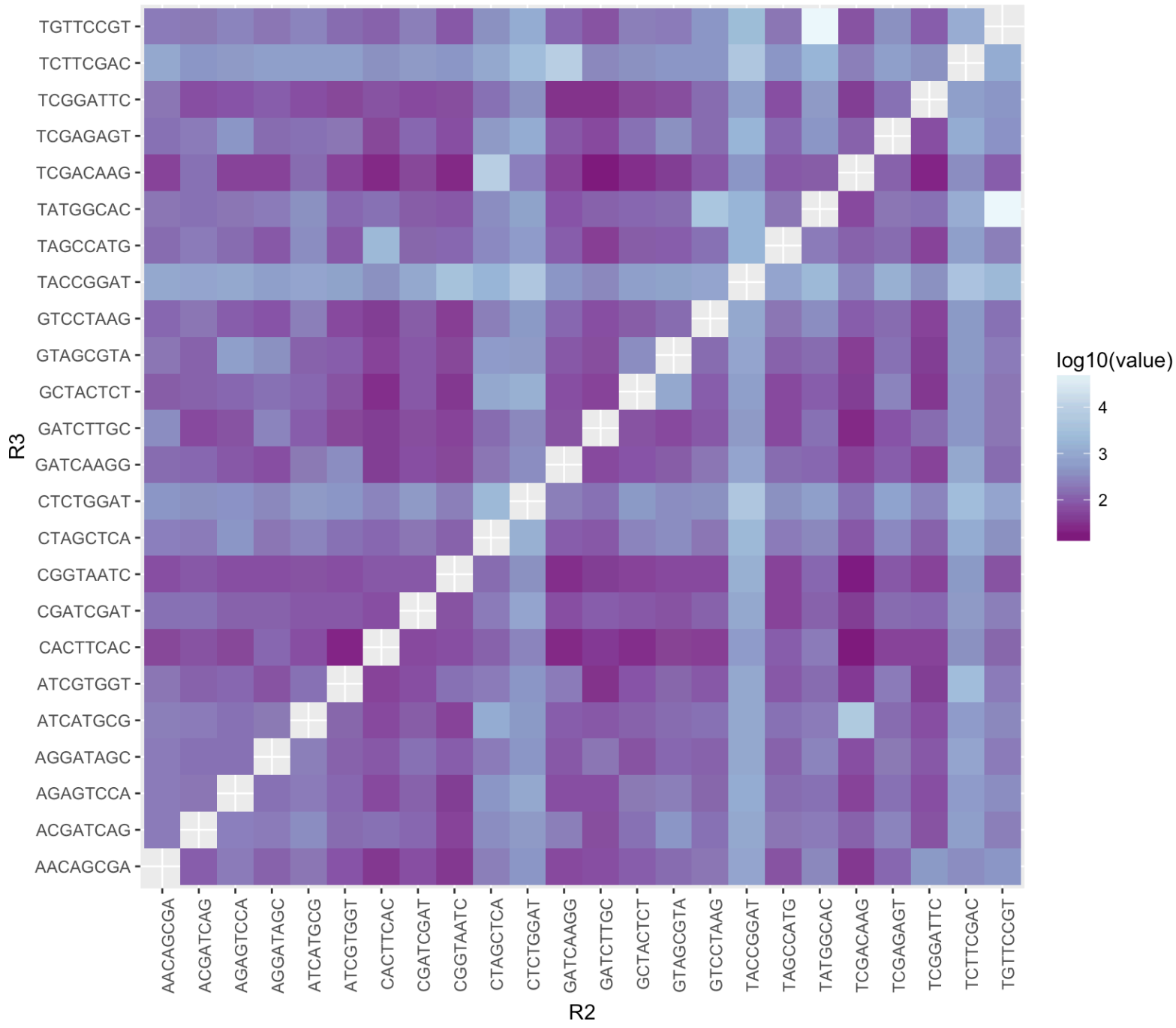


A better visualization of the dictionary of swapped index pairs was constructed below utilizing a heatmap. The counts associated with each index swapped pair were plotted with the log10(counts). Lighter shades of blue approaching white, signify a perfect match of high counts. The darker the shade of blue, the lower the count frequency for that particular swapped pair of indices.

```
swapped_reformat <- read.table("swappedOutputTableFixedHeatmapReformatted.tsv", sep =
"\t",
    header = FALSE)
colnames(swapped_reformat) = c("R2", "R3", "Counts")

library(ggplot2)
library(reshape2)

swapped_reformat.m <- melt(swapped_reformat)  #melt so each row is a unique id-variab
le combination
```

```
## Using R2, R3 as id variables
```

```
ggplot(data = swapped_reformat.m, aes(x = R2, y = R3, fill = log10(value))) + geom_ti
le() +
    coord_equal() + scale_fill_distiller(palette = "BuPu") + theme(axis.text.x = elem
ent_text(angle = 90,
    hjust = 1))
```



The heatmap evidently shows lack of even coverage across reads which could be attributed to the inconsistency in library prep leading to vastly different sample concentrations being adjusted before sequencing. TACCGGAT-TACCGGAT index pair experienced the most index swapping across all other indices, which can be seen in the higher frequency (lighter shade of blue) of swapped indices across the whole TACCGGAT line and column.

Cutoff -0

The `outputTableFixed30-2.py` script was adapted for a second run with a cutoff of 0 and saved as `outputTableFixed0-2.py`. Since just over half of the reads were retained (62.41%) using a cutoff of 30, the demultiplexing script was rerun on the raw reads setting the cutoff parameter to not discard any reads out of curiosity of the category distribution of raw reads.

The output file `1294_S1_L008_R_Output_table_Fixed_Raw-2` contained statistics for the four files at a cutoff of 0 as well as the contents of the expected index pairs, index swapped pairs, and undetermined (N) index pairs dictionaries.

The first 50 lines of `1294_S1_L008_R_Output_table_Fixed_Raw-2` are shown below:

```
Number of Undetermined N Reads: 4205183
Number of Expected Paired Index Reads : 331755033
Number of Index Swapped Reads: 707740
Number of Sequencing Error Reads: 26578779
Number of Retained Reads: 363246735
Number of Discarded Reads: 0
Total Number of Reads: 363246735
Percentage of Reads Index Swapped: 0.19483726398807136
Percentage of Expected Paired Index Reads: 91.33049275721639
Cutoff: 0

Dual Indexed Pairs  Counts
GTAGCGTA_GTAGCGTA   8119243
CGATCGAT_CGATCGAT   5604966
GATCAAGG_GATCAAGG   6587100
AACAGCGA_AACAGCGA   8872034
TAGCCATG_TAGCCATG   10629633
CGGTAATC_CGGTAATC   5064906
CTCTGGAT_CTCTGGAT   34976387
TACCGGAT_TACCGGAT   76363857
CTAGCTCA_CTAGCTCA   17332036
CACTTCAC_CACTTCAC   4191388
GCTACTCT_GCTACTCT   7416557
ACGATCAG_ACGATCAG   7942853
TATGGCAC_TATGGCAC   11184304
TGTTCCGT_TGTTCCGT   15733007
GTCCTAAG_GTCCTAAG   8830276
TCGACAAG_TCGACAAG   3853350
TCTTCGAC_TCTTCGAC   42094112
ATCATGCG_ATCATGCG   10087503
ATCGTGGT_ATCGTGGT   6887592
TCGAGAGT_TCGAGAGT   11741547
TCGGATTC_TCGGATTC   4611350
GATCTTGC_GATCTTGC   3641072
AGAGTCCA_AGAGTCCA   11316780
AGGATAGC_AGGATAGC   8673180

Index Hopped Pairs  Counts
GTAGCGTA_CGATCGAT   213
GTAGCGTA_GATCAAGG   258
GTAGCGTA_AACAGCGA   348
GTAGCGTA_TAGCCATG   332
GTAGCGTA_CGGTAATC   149
GTAGCGTA_CTCTGGAT   965
```

```
GTAGCGTA_TACCGGAT   1975
GTAGCGTA_CTAGCTCA   1014
GTAGCGTA_CACTTCAC   101
GTAGCGTA_GCTACTCT   1532
GTAGCGTA_ACGATCAG   826
```

In this case, all reads are retained, and the demultiplexing script took extra statistics on retained reads that were undetermined or contained “N” base pairs. 91.33% of the total reads were associated with expected index pairs, which is much better than the 62.41% resulting from a cutoff of 30. However, 19.48% of total retained reads were index swapped, which is a concerning statistic. The expected index pairs and respective frequency counts were imported into R and the percentage of total reads per expected index pair was calculated.

```
expected_IndicesRaw <- read.table("expectedOutputTableFixed0-2.tsv", sep = "\t",
    header = TRUE)
expected_IndicesRaw$Counts <- as.integer(expected_IndicesRaw$Counts)
expected_IndicesRaw$percent_total <- expected_IndicesRaw$Counts/363246735 * 100
headersExp0 <- c("Expected Index Pairs", "Counts", "% Total")
colnames(expected_IndicesRaw) <- headersExp0

library(pander)
pander(x = expected_IndicesRaw)
```

Expected Index Pairs	Counts	% Total
GTAGCGTA_GTAGCGTA	8119243	2.235
CGATCGAT_CGATCGAT	5604966	1.543
GATCAAGG_GATCAAGG	6587100	1.813
AACAGCGA_AACAGCGA	8872034	2.442
TAGCCATG_TAGCCATG	10629633	2.926
CGGTAATC_CGGTAATC	5064906	1.394
CTCTGGAT_CTCTGGAT	34976387	9.629
TACCGGAT_TACCGGAT	76363857	21.02
CTAGCTCA_CTAGCTCA	17332036	4.771
CACTTCAC_CACTTCAC	4191388	1.154
GCTACTCT_GCTACTCT	7416557	2.042
ACGATCAG_ACGATCAG	7942853	2.187
TATGGCAC_TATGGCAC	11184304	3.079
TGTTCCGT_TGTTCCGT	15733007	4.331
GTCCTAAG_GTCCTAAG	8830276	2.431

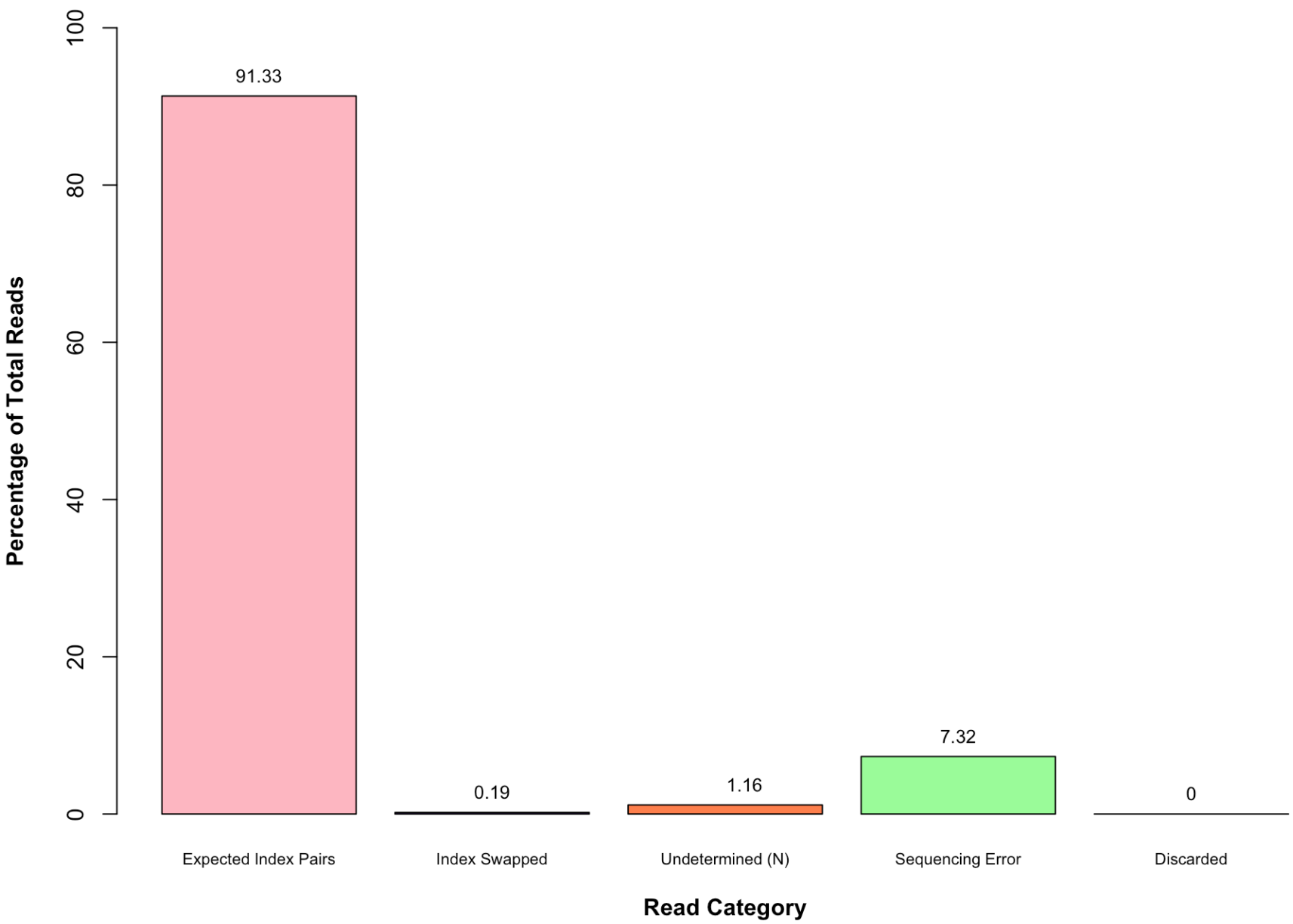
TCGACAAG_TCGACAAG	3853350	1.061
TCTTCGAC_TCTTCGAC	42094112	11.59
ATCATGCG_ATCATGCG	10087503	2.777
ATCGTGGT_ATCGTGGT	6887592	1.896
TCGAGAGT_TCGAGAGT	11741547	3.232
TCGGATTC_TCGGATTC	4611350	1.269
GATCTTGC_GATCTTGC	3641072	1.002
AGAGTCCA_AGAGTCCA	11316780	3.115
AGGATAGC_AGGATAGC	8673180	2.388

A distribution of the general categories the retained raw reads fell into was constructed. Since no reads were discarded, every raw read fell into one of four categories: undetermined (N) read, expected index pair read, swapped index pair read, or sequencing error. Below are the frequency counts associated with each retained read category.

```
Number of Undetermined N Reads: 4205183
Number of Expected Paired Index Reads : 331755033
Number of Index Swapped Reads: 707740
Number of Sequencing Error Reads: 26578779
```

The following is a visual distribution of the raw reads:

```
raw_data = data.frame(c("Expected Index Pairs", "Index Swapped", "Undetermined (N)",
    "Sequencing Error", "Discarded"), c(331755033, 707740, 4205183, 26578779, 0))

raw_data$Percentage <- raw_data[, 2]/sum(raw_data[, 2]) * 100

colnames(raw_data) = c("Category", "Counts", "Percentage")

barplot(raw_data$Percentage, names.arg = raw_data$Category, ylim = c(0, 100), xlab =
expression(bold("Read Category")),
    ylab = expression(bold("Percentage of Total Reads")), cex.names = 0.7, main = "Cu
toff 0 - Read Distribution After Demultiplexing",
    col = c("lightpink", "mediumpurple1", "coral", "palegreen1", "skyblue1"))
text(x = c(0.7, 1.9, 3.2, 4.3, 5.5), y = raw_data$Percentage, label = round(raw_data$
Percentage,
    2), pos = 3, cex = 0.8)
```



With the raw data, index swapping is much less of a problem, however would still be interesting to look at using a heatmap.

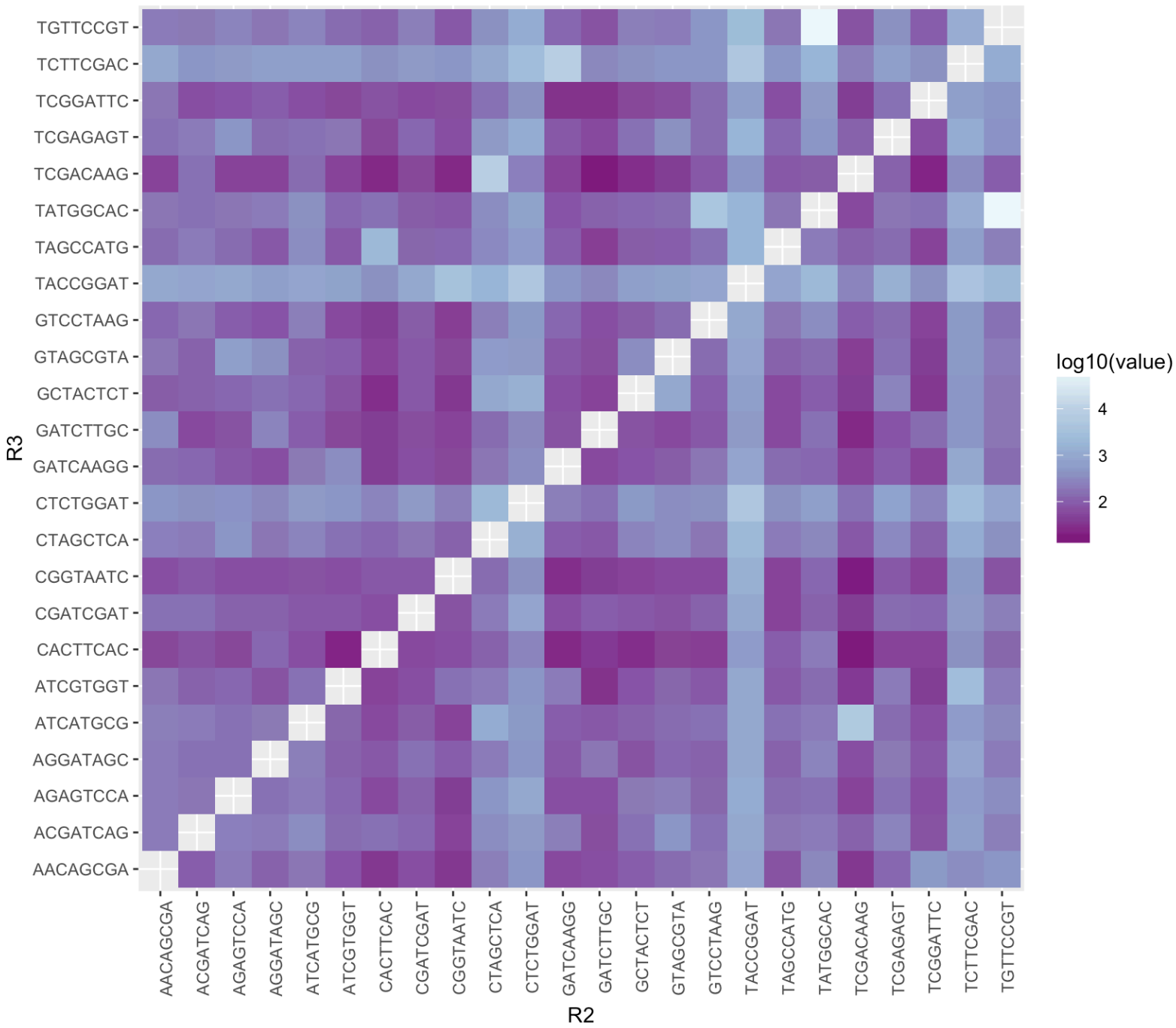
```
# Raw reads:cutoff 0
swapped_reformatRaw <- read.table("swappedIndexesRaw2.tsv", sep = "\t", header = FALS
E)
colnames(swapped_reformatRaw) = c("R2", "R3", "Counts")
swapped_reformatRaw.m <- melt(swapped_reformatRaw)  #melt so each row is a unique id-
variable combination
```

```
## Using R2, R3 as id variables
```

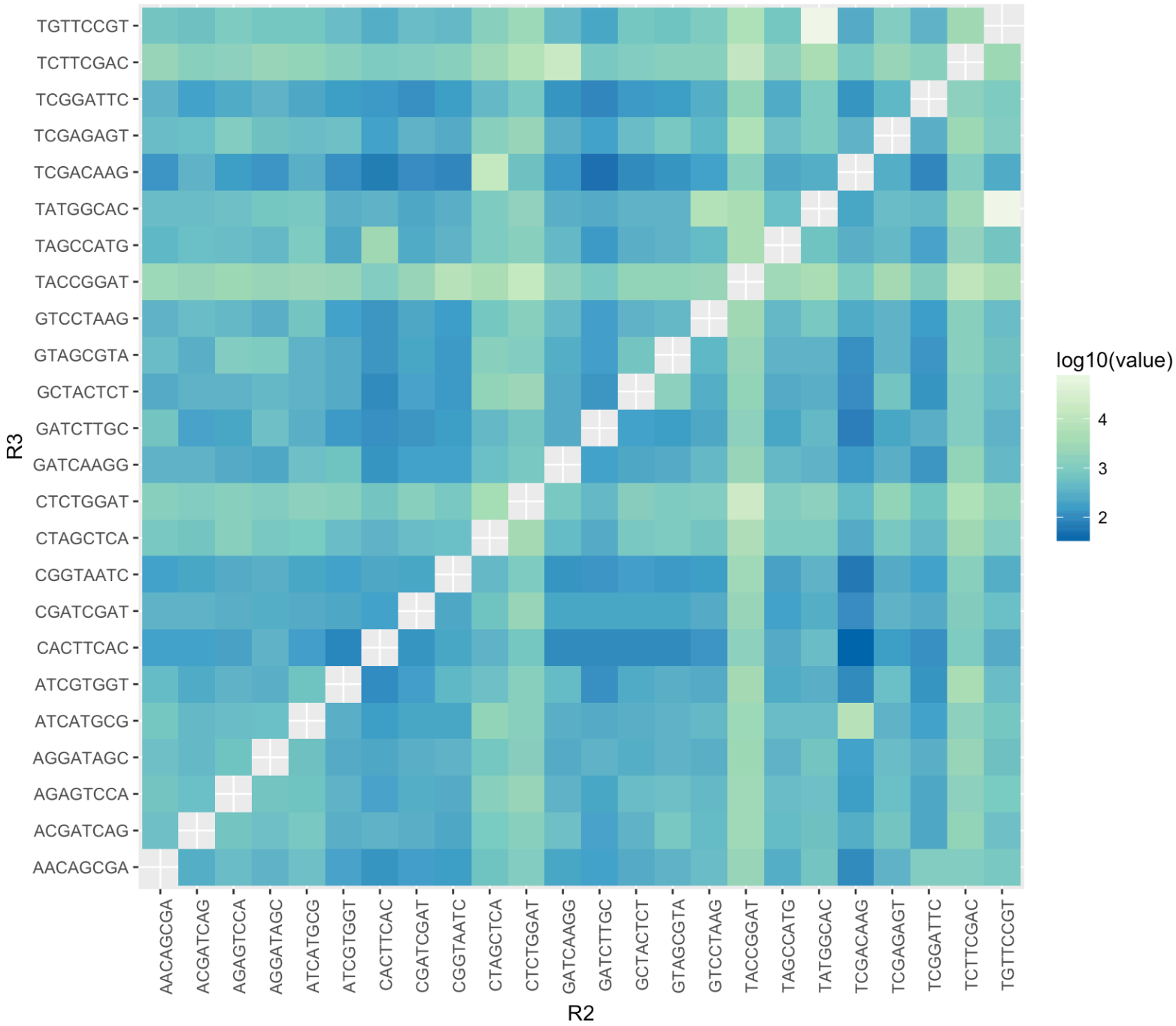
```
# Cutoff 30 reads
swapped_reformat <- read.table("swappedOutputTableFixedHeatmapReformatted.tsv", sep =
"\t",
    header = FALSE)
colnames(swapped_reformat) = c("R2", "R3", "Counts")
swapped_reformat.m <- melt(swapped_reformat)
```

```
## Using R2, R3 as id variables
```

```
library(ggplot2)
library(reshape2)

par(mfrow = c(1, 2), oma = c(0, 1, 1, 0))

ggplot(data = swapped_reformat.m, aes(x = R2, y = R3, fill = log10(value))) + geom_ti
le() +
    coord_equal() + scale_fill_distiller(palette = "BuPu") + theme(axis.text.x = elem
ent_text(angle = 90,
    hjust = 1))
```




```
ggplot(data = swapped_reformatRaw.m, aes(x = R2, y = R3, fill = log10(value))) +
    geom_tile() + coord_equal() + scale_fill_distiller(palette = "GnBu") + theme(axis
.text.x = element_text(angle = 90,
    hjust = 1))
```



0.19% of total raw reads are made up of swapped pairs of indices, most commonly associated with the notion of indexing hopping. The distribution of swapped indices in the total raw reads is observed to be similar to the subset of swapped index reads that were retained at a cutoff of 30 once viewed side by side. Note: the blue to green gradient heatmap corresponds to the raw reads index swapped while the blue to purple gradient of the cutoff 30 index reads was carried down from the previous.