Lab 3_20011141

Maria Bassem

2024-03-06

ID: “20011141”

Introduction:

In this lab, I was introduced to preprocessing and sequence alignment techniques, such as Principal Component Analysis (PCA), statistical testing, and sequence alignment using BLAST, rentrez, … etc.

Note:

For the easiness and readability of your code. Make sure to include and run this cell at the beginning of your markdown

```
# install.packages("rentrez")
# install.packages("seqinr")
library(tidyverse)
library(rentrez)
```

```
## Warning: package 'rentrez' was built under R version 4.3.3
```

```
library(seqinr)
```

```
## Warning: package 'seqinr' was built under R version 4.3.3
```

```
library(Biostrings)
library(ggplot2)
library(data.table)
```

Part 1: Principal Component Analysis (PCA) (30 points)

Principal Component Analysis (PCA) is a dimensionality reduction technique used in statistics and machine learning to transform high-dimensional data into a new set of uncorrelated variables, called principal components. These components capture the maximum variance in the original data, enabling a simplified representation of complex datasets while maintaining essential information about the relationships among variables.

For this task, use the minified version of brain cancer dataset from this link

```
brain_min <- fread("D:\\Third Year Computer\\Term 2\\Bio\\Labs\\Lab 3\\BrainCancerMin.csv")
brain_min <- as.data.frame(brain_min)
```

Task 1.1: Perform PCA (30 points)

1. Perform PCA using the princomp function

```
gene_info <- select(brain_min, -samples, -type)
pca <- princomp(cor(gene_info)) # cpr --> correlation matrix of the gene_info
```

2. Calculate the variation explained by each principal component

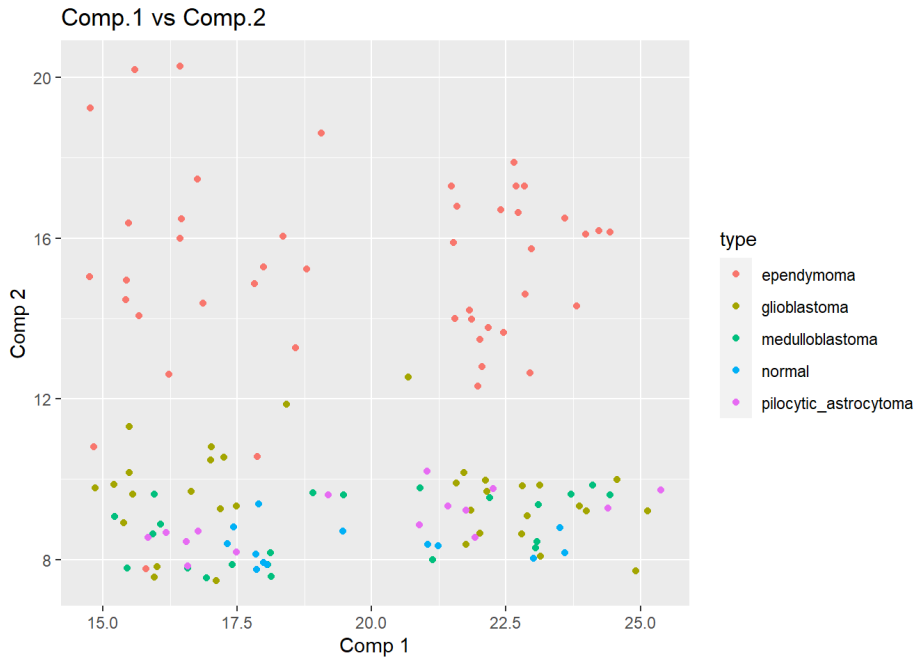
```
explained_variance = pca$sdev^2 / sum(pca$sdev^2)
print(head(explained_variance))
```

```
##     Comp.1     Comp.2     Comp.3     Comp.4     Comp.5     Comp.6
## 0.66460207 0.08383510 0.07322121 0.06408607 0.02617023 0.01567058
```

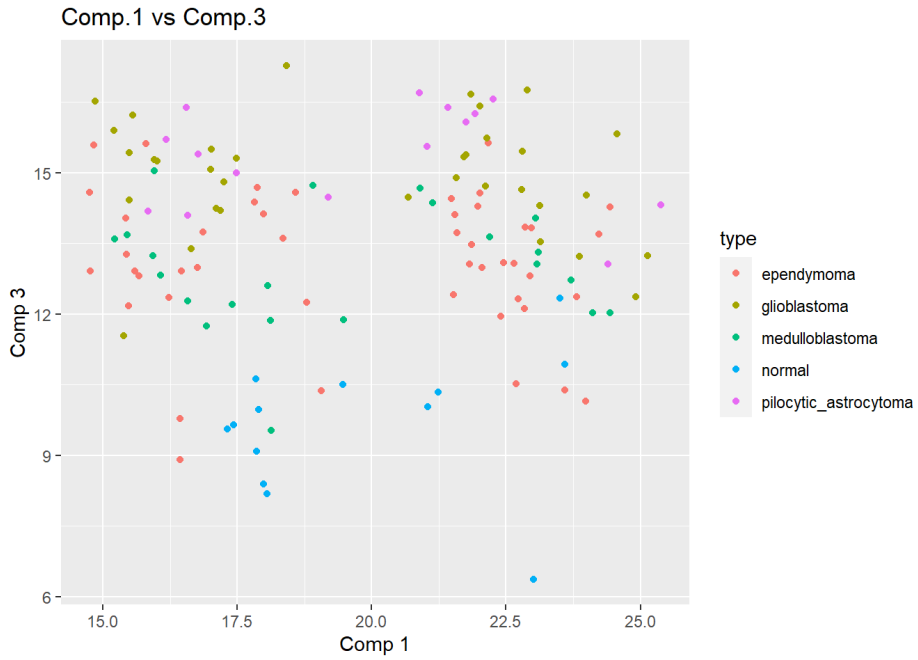
3. Plot 3 scatter plots. (Comp.1 vs Comp.2, Comp.1 vs Comp.3, Comp.2 vs Comp.3). Which plot do you think is best?

Since we know that the primary objective of PCA is to maximize variance and reduce dimensions to minimize reconstruction error, we can evaluate the scatter plots based on how well they capture and represent the variance in the data. The best plot in my opinion is the one showing comp 1 vs comp 2, because it shows big variance of data which minimizes the reconstruction error

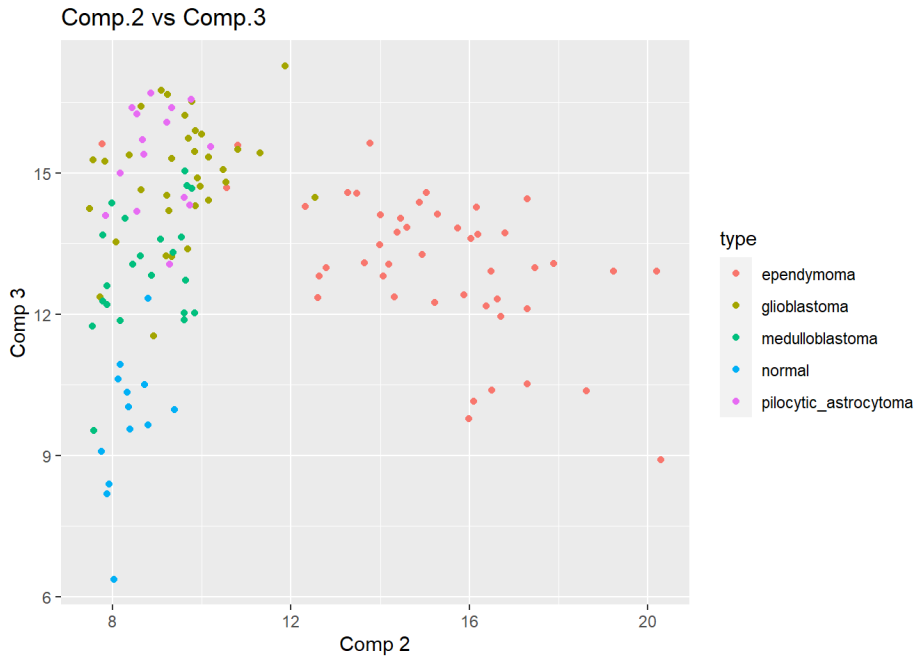
```
transformed_dataset <- as.data.frame.matrix(predict(pca, brain_min))

transformed_dataset$type <- c(brain_min$type)

ggplot(transformed_dataset, aes(x = Comp.1, y = Comp.2, color = type)) +
  geom_point() +
  labs(title = "Comp.1 vs Comp.2", x = "Comp 1", y = "Comp 2")
```




```
ggplot(transformed_dataset, aes(x = Comp.1, y = Comp.3, color = type)) +
  geom_point() +
  labs(title = "Comp.1 vs Comp.3", x = "Comp 1", y = "Comp 3")
```




```
ggplot(transformed_dataset, aes(x = Comp.2, y = Comp.3, color = type)) +
  geom_point() +
  labs(title = "Comp.2 vs Comp.3", x = "Comp 2", y = "Comp 3")
```

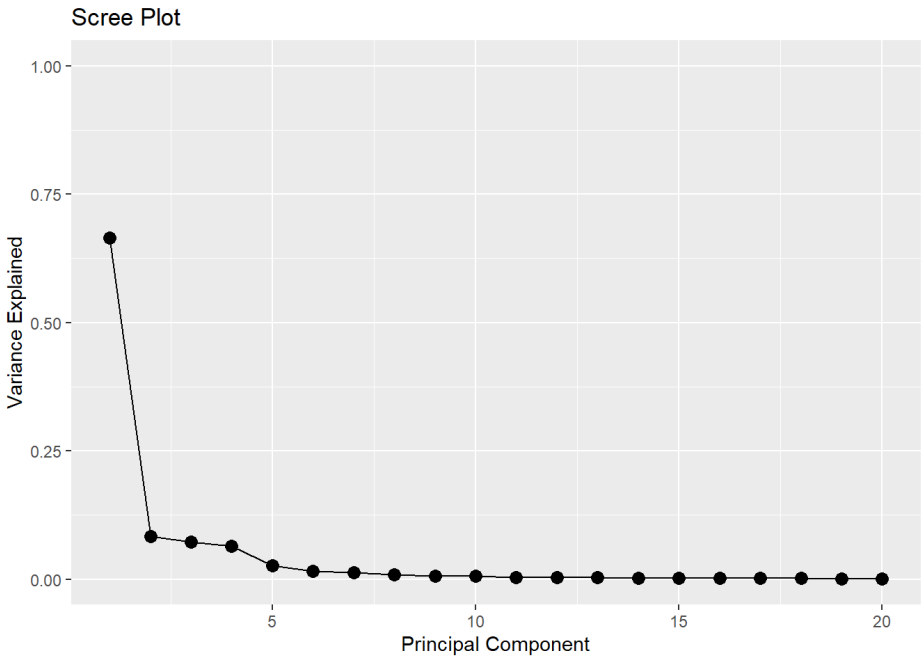


4. Draw a scree plot using the ggplot2 for the first 20 principal components

Reference from Geeks for Geeks: Scree Plot (https://www.geeksforgeeks.org/how-to-make-scree-plot-in-r-with-ggplot2/)

```
# Extracting variance for the first 20 components
variance_20 <- explained_variance[1:20]

# Scree plot
# instead of geom_line() we could use geom_col() according to geeks for geeks
qplot(c(1:20), variance_20) +
  geom_line() +
  geom_point(size=3)+
  xlab("Principal Component") +
  ylab("Variance Explained") +
  ggtitle("Scree Plot") +
  ylim(0, 1)
```

```
## Warning: `qplot()` was deprecated in ggplot2 3.4.0.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```



Part 2: Statistical Testing (25)

For this task, use the diabetes prediction dataset from this link

```
diabetes_data <- fread("D:\\Third Year Computer\\Term 2\\Bio\\Labs\\Lab 3\\diabetes_prediction_dataset.csv")
diabetes_data <- as.data.frame(diabetes_data)
```

Task 2.1: Fisher's Test (15)

Fisher's Test is a statistical method used for contingency table analysis, specifically when dealing with categorical data. This test calculates the probability of observing a particular distribution of categorical variables in a contingency table, assuming the independence of the variables. It is used to assess the significance of associations between categorical variables.

1. Count the alleles with respect to diabetes (hint 1: Use tables) (hint 2: Split the alleles into two columns and create two counts with each column then add them)

```
# separate is used to split the allele_column into two columns
alleles_T2D <- separate(diabetes_data, alleles, into = c("allele1", "allele2"), sep = 1)

# Count the alleles using table equivalent to hashmap
allele1_counts <- table(
  alleles_T2D$allele1,
  alleles_T2D$diabetes
)

print("allele1_counts")
```

```
## [1] "allele1_counts"
```

```
print(allele1_counts)
```

```
##
##         0     1
##   A 61088  5681
##   C 30412  2819
```

```
allele2_counts <- table(
  alleles_T2D$allele2,
  alleles_T2D$diabetes
)
cat("\n")
```

```
print("allele2_counts")
```

```
## [1] "allele2_counts"
```

```
print(allele2_counts)
```

```
##
##         0     1
##   A 30572  2818
##   C 60928  5682
```

```
# adding the results of the 2 tables together to get the final table
total_alleles_count <- allele1_counts + allele2_counts
cat("\n")
```

```
print("total_alleles_count")
```

```
## [1] "total_alleles_count"
```

```
print(total_alleles_count)
```

```
##
##         0     1
##   A 91660  8499
##   C 91340  8501
```

2. Run fisher test and report the p-value. Is the treatment significant?

```
fisher_result <- fisher.test(total_alleles_count)
p_value <- fisher_result$p.value
cat("p_value = ", p_value, "\n")
```

```
## p_value =  0.816139
```

```
# Check for significance (using 0.05)
if (p_value < 0.05) { # reject the null hypothesis --> accept the significance
  print("The association between alleles and diabetes is significant.")
} else {
  print("The association between alleles and diabetes is most probably by chance or random.")
}
```

```
## [1] "The association between alleles and diabetes is most probably by chance or random."
```

According to the results, there is no association between the alleles and diabetes, so the treatment is not significant.

Task 2.2: T Test (10)

T Test is a widely used statistical method for comparing the means of two independent samples to determine if they are significantly different from each other. T Test is good for small sample sizes and assumes that the data is normally distributed. This test provides a way to evaluate whether the observed difference in means between groups are statistically significant.

1. Extract two BMI samples from two of the alleles family (AA, AC, CC)

```
# Subsetting data for the AA allele family
AA_samples <- diabetes_data[diabetes_data$alleles == "AA", "bmi"]
print("BMI samples for AA allele family:")
```

```
## [1] "BMI samples for AA allele family:"
```

```
print(head(AA_samples))
```

```
## [1] 27.32 23.45 33.64 27.32 27.32 25.69
```

```
AC_samples <- diabetes_data[diabetes_data$alleles == "AC", "bmi"]
print("BMI samples for AC allele family:")
```

```
## [1] "BMI samples for AC allele family:"
```

```
print(head(AC_samples))
```

```
## [1] 20.14 27.32 19.31 23.86 54.70 36.05
```

2. Perform T-Test and report the p-value. What is this test measuring and what is its significance?

The t-test is a statistical test used to compare the means of two groups and determine whether there is a significant difference between them or no.

In our examples, the t-test is evaluating whether there is a **statistically significant difference** in the mean BMI between these two allele groups.

- The null hypothesis (H_0) is that the true difference between these group means is zero.
- The alternate hypothesis (H_a) is that the true difference is different from zero.

Useful link: T-test Explanation (https://www.scribbr.com/statistics/t-test/)

```
# Perform t-test
t_test_result <- t.test(AA_samples, AC_samples)

# Extract the p-value
p_value <- t_test_result$p.value
cat("p_value = ", p_value, "\n")
```

```
## p_value =  0.5125191
```

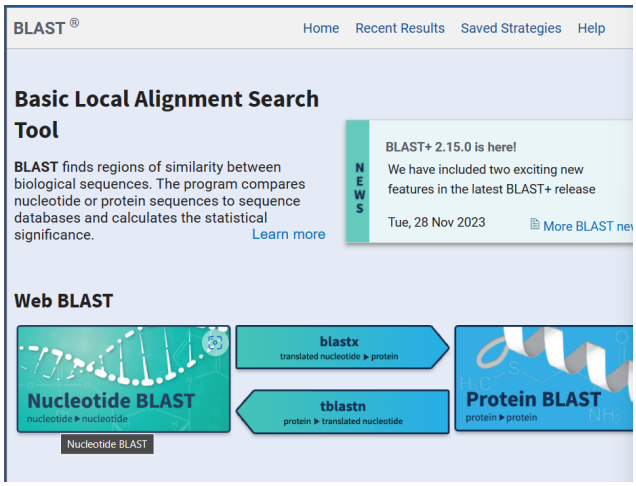
```
# Check for significance (using 0.05)
if (p_value < 0.05) { # reject the null hypothesis --> accept the significance
  print("Reject the null hypothesis of no difference and say with a high degree of confidence that the true difference in me
ans is not equal to zero.")
} else {
  print("Accept the null hypothesis of no difference")
}
```

```
## [1] "Accept the null hypothesis of no difference"
```

Part 3: Sequence Alignment (40)

Task 3.1: BLAST (10)

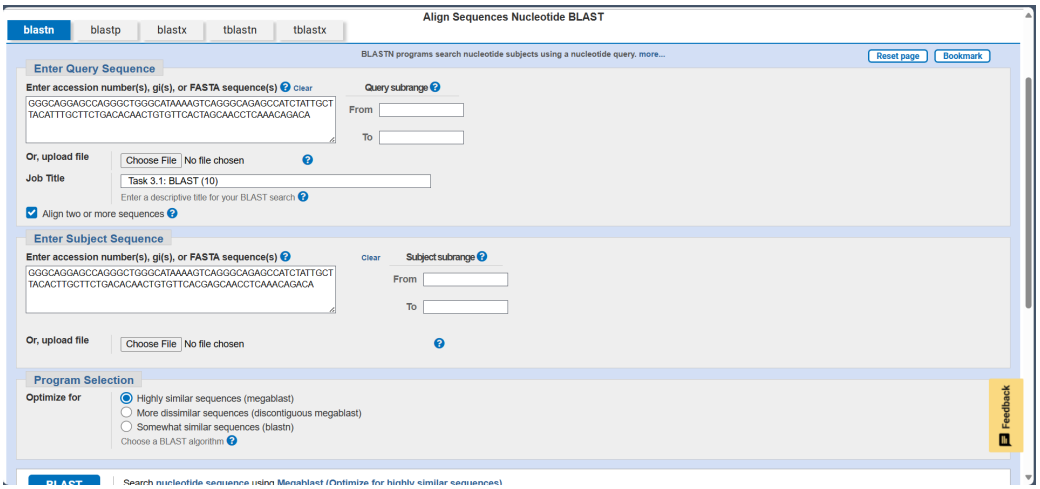
1. Visit BLAST from this link

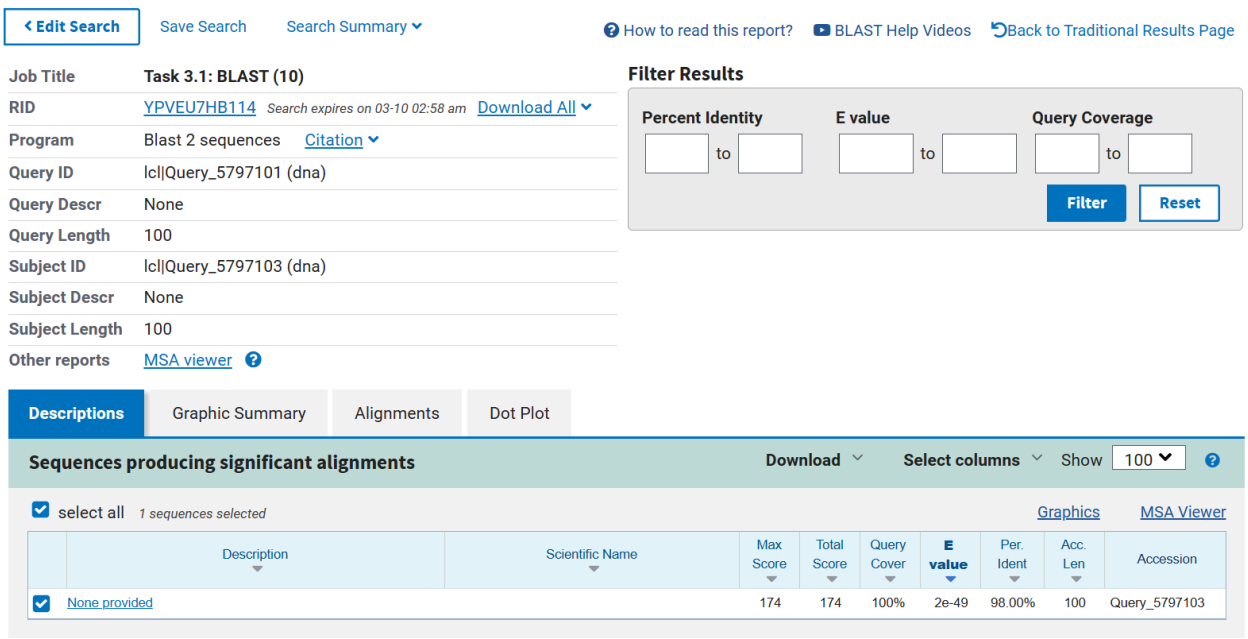


2. Use the following sequences for your experiment

- **SEQ.A**
  GGGCAGGAGCCAGGGCTGGGCATAAAAGTCAGGGCAGAGCCATCTATTGCTTACATTTGCTTCTGACACAACTGTGTTCACTAGCAACCTCAAACAG

- **SEQ.B**
  GGGCAGGAGCCAGGGCTGGGCATAAAAGTCAGGGCAGAGCCATCTATTGCTTACACTTGCTTCTGACACAACTGTGTTCACGAGCAACCTCAAACA

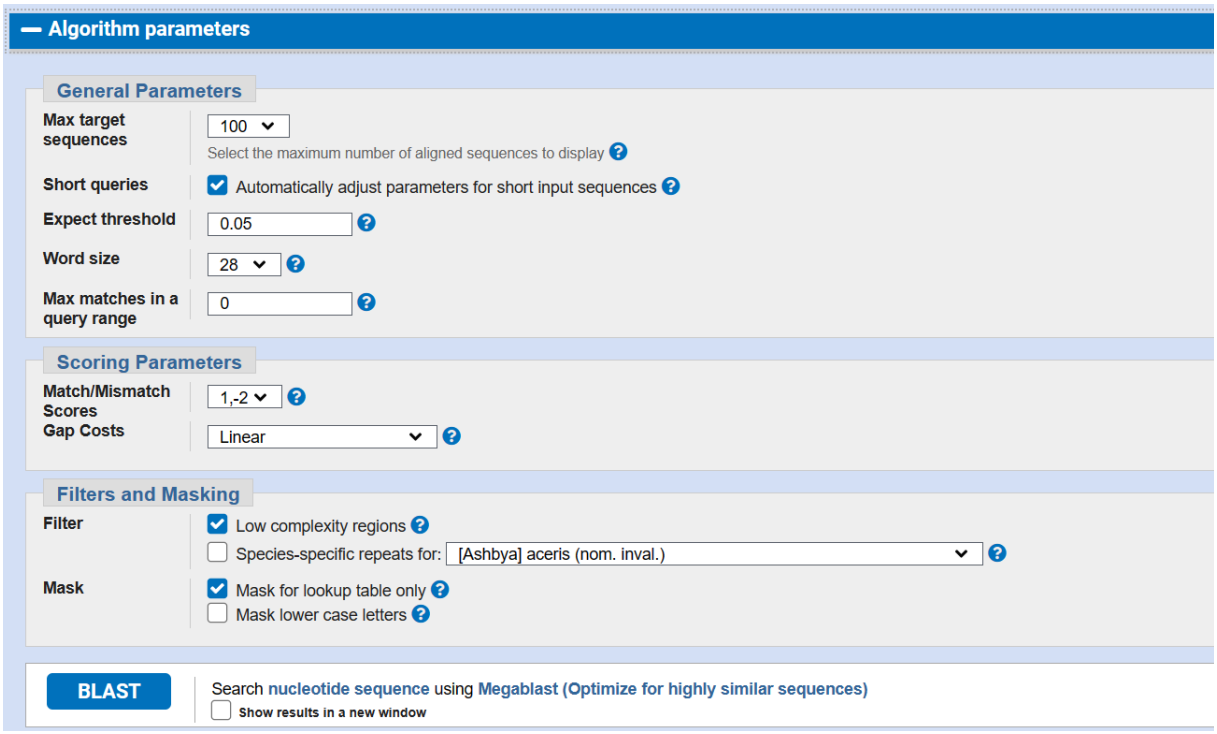




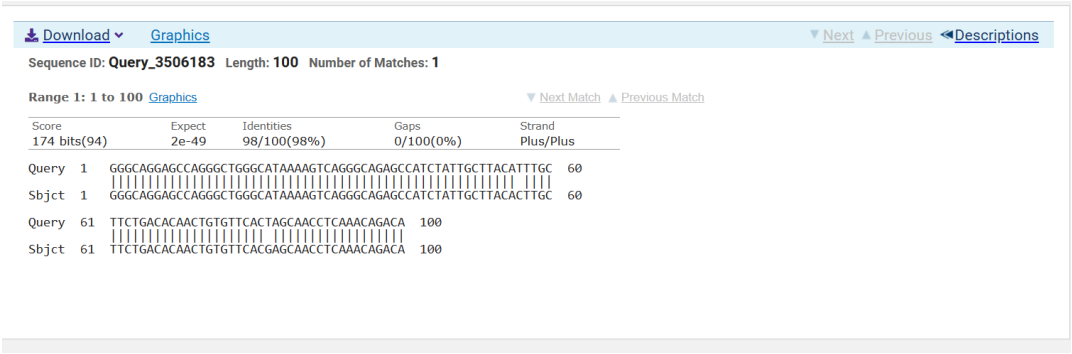
3. Report the algorithm parameters, score, percent identity, and e-value

- Algorithm Parameters:

  nt for nucleotides

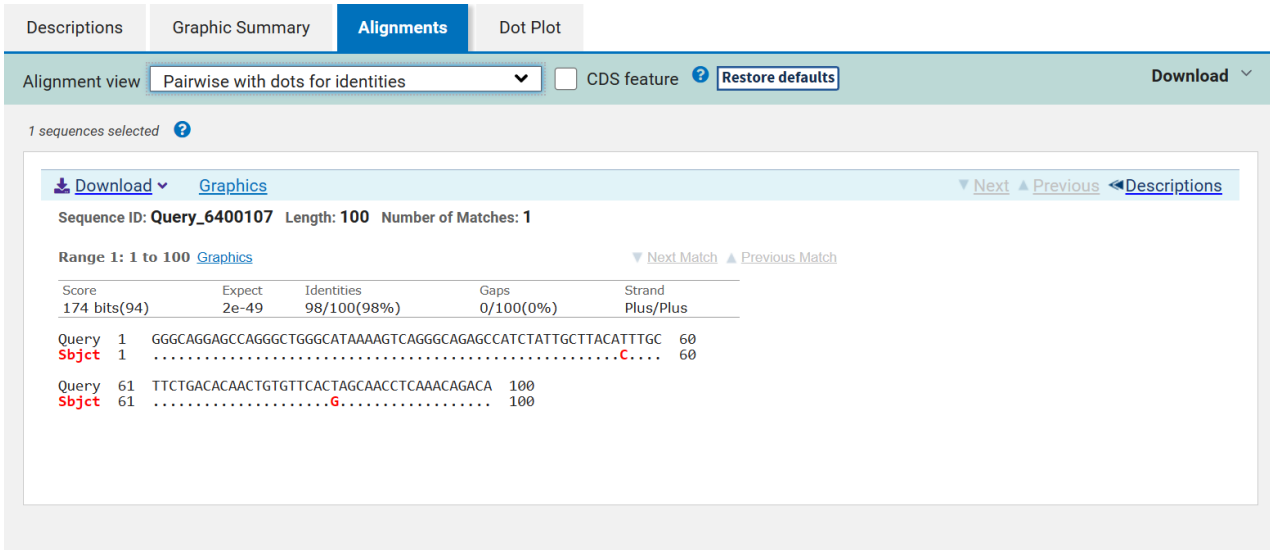


- Score: 174 bits
- Percent Identity: 98%
- e-value: 2e-49



4. How many base pairs are mismatched?

- 2 basepairs are mismatched
    - At location 56 and at 82



Task 3.2: Retrieve Sequences (10)

In this task, the focus was on retrieving sequences from GenBank using the “BioStrings” and “rentrez” libraries. We accessed the sequences with specified accession numbers. we delt with the format (FASTA) and then stored it in a data structure (DNAString), to facilitate further sequence manipulation.

Reference: Rentrez Tutorial (https://cran.r-project.org/web/packages/rentrez/vignettes/rentrez_tutorial.html#fetch-dna-sequences-in-fasta-format)

1. Install and load the necessary packages: “BioStrings” and “rentrez”
2. Fetch two sequences from GenBank (using accession numbers NG_050578.1 and X03562.1) : Hint: The function you’ll use from the rentrez package has parameters for specifying the database, the ID of the sequence, and the format you want the sequence in. Think about how you specify the database for nucleotide sequences and the format for FASTA.

- Nucleotide database (referred to as `nuccore` in EUtils)

```
accession_nums <- c("NG_050578.1", "X03562.1")
two_seq <- entrez_fetch(db="nuccore", id=accession_nums, rettype="fasta")
# Rather than printing all those bases, we can take a peak at the top of the file
cat(strwrap(substr(two_seq, 1, 500)), sep="\n")
```

```
## >NG_050578.1 Homo sapiens INS-IGF2 readthrough (INS-IGF2), RefSeqGene
## on chromosome 11
## GAGGTGCGGATCCTGGGCGGCCAGGGAAGGTCTCTGCCGCCAGGGAAGTGTCCCAGAGACCCCTGGAGGG
## GCTGCTGACACCCCCGGTGCCCCCACCTCGAGCATGACCCAGGGCTGCCTCTCCCCATCCTTCATCCTCC
## CTGCTCCACAGGACATTGGCCTGGCGTCCCTGGGGGCCTCGGATGAGGAAATTGAGAAGCTGTCCACGGT
## GGGTTGACCCCTCCCTGCAGGGCCTGGGGTGTGGGTTTGGGGGTCTGAATCCAGGCCTCACCCTCTTGCC
## GTCCAGGCTGAGGCCTCTCCTTCCACCCACGAATTGTGACCCTCACCCTGGCCTGCCTGCATCCTGGCCT
## GGCCTCCCTGGGGGTGGTATCCTGGTCACGGGTGACCAGGGGCTGCCCGGTGGGCGGC
```

3. Store the sequence in an appropriate R data structure:

Hint: When you fetch data, it might not be in a format directly readable by the Biostrings functions. Consider how you can create a connection to read text data directly into R functions. You need to convert FASTA sequences to DNAStringSet. You can use getSequence function

```
temp <- tempfile()
write(two_seq, temp)
parsed_recs <- readDNAStringSet(temp)
print(parsed_recs)
```

```
## DNAStringSet object of length 2:
##     width seq                                               names
## [1] 39098 GAGGTGCGGATCCTGGGCGGCCA...ACTCATCCATCCACTCATCCCTC NG_050578.1 Homo ...
## [2]  8837 CCCAACCCCGCGCACAGCGGGCA...CACGGGAATTTTCAGGGTAAACT X03562.1 Human ge...
```

Task 3.3: Sequence Processing (20)

1. Identify sequences with gaps or ambiguous bases: Hint: Use the alphabetFrequency function. Try to understand what is the format of the returned value

```
sequences <- parsed_recs

freq_seq <- alphabetFrequency(sequences)

# printing
freq_seq
```

```
##        A     C     G    T M R W S Y K V H D B  N - + .
## [1,] 7355 12386 11660 7697 0 0 0 0 0 0 0 0 0 0  0 0 0 0
## [2,] 1388  3037  2697 1685 0 0 0 0 0 0 0 0 0 0 30 0 0 0
```

We have 30 ambiguous bases (N) in sequence 2 and no gaps (-) in any of the sequences.

2. Remove gaps and ambiguous bases from sequences, and determine the length of the sequence before and after the removal

```
# before removing
seq1 <- sequences[1]
seq2 <- sequences[2]
len1_before <- seq1@ranges@width
len2_before <- seq2@ranges@width
cat("Length of sequence 1 before removal:", len1_before,
    "\nLength of sequence 2 before removal:", len2_before,"\n\n")
```

```
## Length of sequence 1 before removal: 39098
## Length of sequence 2 before removal: 8837
```

```
# removing
cleaned_seq1 <- DNAString(gsub("[-N]", "", seq1))
cleaned_seq2 <- DNAString(gsub("[-N]", "", seq2))
print("Sequences after removing gaps and ambiguous bases")
```

```
## [1] "Sequences after removing gaps and ambiguous bases"
```

```
print(cleaned_seq1)
```

```
## 39098-letter DNAString object
## seq: GAGGTGCGGATCCTGGGCGGCCAGGGAAGGTCTCTG...CACCCATCCCTCCACTCATCCATCCACTCATCCCTC
```

```
print(cleaned_seq2)
```

```
## 8807-letter DNAString object
## seq: CCCAACCCCGCGCACAGCGGGCACTGGTTTCGGGCC...CCATCTCCCTTCTCACGGGAATTTTCAGGGTAAACT
```

```
# after removing
len1_after <- cleaned_seq1@length
len2_after <- cleaned_seq2@length
cat("Length of sequence 1 after removal:", len1_after,
    "\nLength of sequence 2 after removal:", len2_after,"\n")
```

```
## Length of sequence 1 after removal: 39098
## Length of sequence 2 after removal: 8807
```

3. Run Pairwise local alignment, report the score and width of each sequence before and after the alignment.

```
# width of each sequence before
len1_after <- cleaned_seq1@length
len2_after <- cleaned_seq2@length
cat("Length of sequence 1 after removal:", len1_after,
    "\nLength of sequence 2 after removal:", len2_after,"\n\n")
```

```
## Length of sequence 1 after removal: 39098
## Length of sequence 2 after removal: 8807
```

```
# Local Alignment
mat <- nucleotideSubstitutionMatrix(match = 1, mismatch = -3, baseOnly = TRUE)

localAlign <- pairwiseAlignment(
 pattern = cleaned_seq1,
 subject = cleaned_seq2,
 type = "local",
  substitutionMatrix = mat
)
# Printing Results
localAlign
```

```
## Local PairwiseAlignmentsSingleSubject (1 of 1)
## pattern: [24984] CCCAACCCCGCGCACAGCGGGCACTGGTTT...CCCTTCTCACGGGAATTTTCAGGGTAAACT
## subject:     [1] CCCAACCCCGCGCACAGCGGGCACTGGTTT...CCCTTCTCACGGGAATTTTCAGGGTAAACT
## score: 7355
```

```
# Extract information
alignment_score <- localAlign@score
pattern_width <- localAlign@pattern@range@width
subject_width <- localAlign@subject@range@width

# Print the required info
cat("\nAlignment Score: ", alignment_score, "\n")
```

```
##
## Alignment Score:  7355
```

```
cat("Pattern Width: ", pattern_width, "\n")
```

```
## Pattern Width:  8895
```

```
cat("Subject Width: ", subject_width, "\n")
```

```
## Subject Width:  8807
```

4. Create a function that returns all the positions of mismatching pairs

```
# Two sequences to compare and find the positions of mismatching pairs from
seq1 <- as.character(alignedPattern(localAlign))
seq2 <- as.character(alignedSubject(localAlign))


arg_mismatch <- function(seq1, seq2) {

  seq1 <- strsplit(seq1, "")[[1]]
  seq2 <- strsplit(seq2, "")[[1]]

  # Find positions where both seq1 and seq2 are not gaps
  non_gap_positions <- which(seq1 != "-" & seq2 != "-")

  # Use which to find mismatch positions
  mismatch_positions <- non_gap_positions[
    which(seq1[non_gap_positions] != seq2[non_gap_positions])
  ]

  return(mismatch_positions)
}

# Function call
result_value <- arg_mismatch(seq1, seq2)
print(result_value)
```

```
##   [1]   58   59  466  467  469  526  765  766  767  768  776  797  811 1215 1279
##  [16] 1280 1550 1651 1655 1659 1660 1663 1772 3055 3197 3198 3200 3201 3202 3204
##  [31] 3206 3207 3266 3400 3445 3502 3527 3528 3529 3530 3532 3647 3648 3649 3650
##  [46] 3655 3659 3720 3854 3881 3955 4373 4374 4495 4542 4543 4679 4694 4760 4866
##  [61] 5231 5243 5245 5246 5317 5429 5886 5939 5944 6124 6125 6126 6261 6621 6921
##  [76] 6926 6927 6928 6929 6930 6931 6935 7171 7201 7359 7361 7378 7431 7509 7510
##  [91] 7511 7512 7513 7517 7538 8036 8037 8038 8039 8040 8041 8045 8843 8872 8873
```

The overall objective of Part 3 tasks was to walk through a comprehensive sequence alignment workflow, starting from sequence retrieval, processing, to local alignment.