BioPython Fragaria x ananassa

Vs

Fragaria vesca

Comparative Genomic Analysis

Link to video:

https://youtu.be/wSam4itsuF4

Python and numpy basic lectures referenced in the video: https://github.com/jrjohansson/scientific-python-lectures

BioPython's official manual:

http://biopython.org/DIST/docs/tutorial/Tutorial.pdf

By Eliza Doering



http://strawberry-garden.kazusa.or.jp/images/strawberry1.jpg

INTRODUCTION/BACKGROUND

The strawberry (genus Fragaria) is a diverse plant with over 22 described species. Two of these are *Fragaria x ananassa* and *Fragaria vesca*. The diversity of the Fragaria genus can be shown via the comparison of these two organisms using the BioPython library.

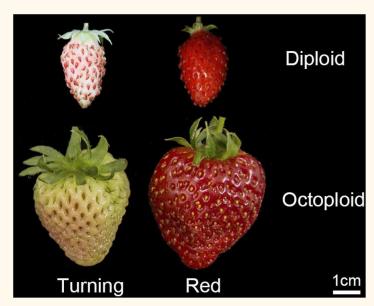
BioPython

Biopython is a great tool for bioinformaticians. It includes functionalities for the reading and parsing of multiple types of files, analysis of sequences (such as GC content and length), plotting, visualizations, and more! It is an open source technology, meaning that anyone can look at its source code. It also includes a wiki with user uploaded functions and tools currently in development. This project includes all three of these types of functions.

Datasets: Cultivated vs Wild

As the image displayed below demonstrates these two organisms' respective sizes differ greatly. For each organism: fasta files containing the annotated proteins and transcripts. These particular files are useful as they are the same format and they are all from two resources that interact with each other (GDR sources Strawberry Garden and vice versa). The larger

counterpart is more commonly grown than its wild counterpart and they can both grow under similar conditions. Anecdotally I have found that the wild counterpart is more difficult to plant in a garden, but it has a tendency of maintaining its sweetness through extreme weather than its cultivated relative. I wanted to see if they GC content and transcript length could account for any of this.



https://www.biorxiv.org/content/biorxiv/early/2020/01/03/2020.01.02.893453/F6.large.jpg

1. Fragaria x ananassa (cultivated)

As stated above, Fragaria x ananassa is a much larger species than its wild counterpart. Fasta files downloaded from GDR (www.rosaceae.org) and Strawberry Garden (http://strawberry-garden.kazusa.or.jp/).

File Name + Link	Description	Size
Fxa_v1.2_makerStandard_p roteins_woTposases.fasta	Annotated proteins from GDR	52.6 MB
FAN_r2.3_transcripts.fa	Transcripts from Strawberry Garden	222 MB

2. Fragaria vesca (wild/alpine)

Files were also accessed from GDR and Strawberry Garden for the smaller species.

File Name + Link	Description	Size
Fragaria_vesca_v4.0.a1_ma kerStandard_proteins.fasta	Annotated proteins from GDR	13.5 MB
Fragaria_vesca_v4.0.a2.tra nscripts.fa	Transcripts from GDR	134 MB

The file size differences echo the different organism sizes.

BioPython installation

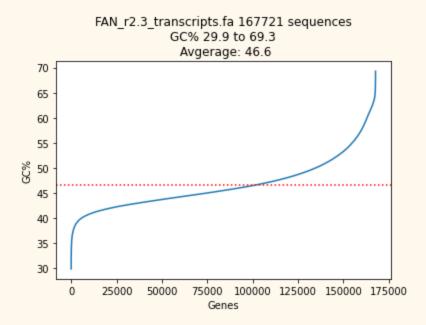
pip install biopython

Installing BioPython is easy! It can be done in a single command if you have the python library installer 'pip' installed. That option is shown in the image above. Python3 and JupyterLab should also be installed. The directions for how to install these are in the link on the first page of this manual.

Alternatively biopython can be installed directly into a google colab notebook. This option is shown above. Once installation is complete it is time to begin! Before any programming can take place the correct libraries must be imported. All imports needed to complete this tutorial are shown in the image above.

The rest of the tutorial will be in the last pages of the manual.

BioPython Use with the Organisms Output

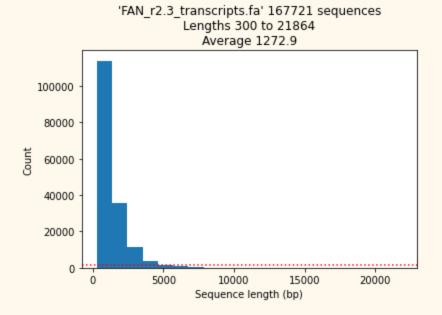


Here is the GC content of *Fragaria x ananassa* transcripts.

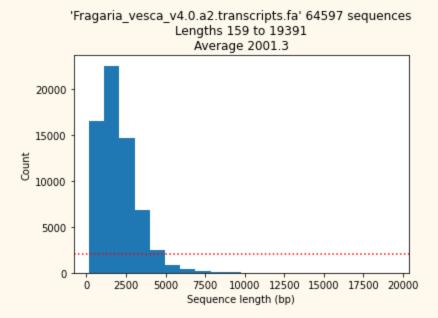
Fragaria_vesca_v4.0.a2.transcripts.fa 64597 sequences GC% 28.5 to 79.5 Avgerage: 43.6 Ò

Genes

Here is the GC content of *Fragaria vesca* transcripts.



Here is the average sequence length of *Fragaria x ananassa* transcripts.



Here is the average sequence length of *Fragaria vesca* transcripts.

Before any programming can take place the correct libraries must be imported. These are all of the imports necessary for this tutorial. Imprting 'Seq', 'SeqIO', 'GC', and 'StringIO' in this way allows for direct calling. This will be more clear in a later cell. In [1]: **from Bio.Seq import** Seq from Bio import SeqIO from Bio.SeqUtils import GC from io import StringIO import gzip import pylab import os import numpy as np Now that imports are done it is time to load in and parse some fasta files. What 'direct calling' means will become more obvious. Calling SeqIO's parsing function is simple now. Since this is a fasta file the second argument (filetype) written is fasta. This can be done for different types of files, but fasta will be the only one used in this tutorial. In [2]: FAN r2 = 'FAN r2.3 transcripts.fa' anan Trs = SeqIO.parse(FAN r2, "fasta") #ananassa transcripts To view it's contents I'll make a function that loops through the contents of the parsed file. I'll stop at 10 lines to reduce computational power. Since I'll only be passing fasta objects through it I can grab BioPython's fasta parameters. In this case I'm grabbing 'id' and 'seq'. In [3]: def printFastaContents(fasta): i=0 for record in fasta: i+=1print("ID:", record.id) print((repr(record.seq))) print("Length:", len(record)) if i>10: break Now I'll run it through with the ananassa transcript variable I made in cell 4. In [4]: | printFastaContents(anan Trs) ID: FAN r2.3ch1Ava g000010.1 ${\tt Seq('ATGTCTCACAGATCTCATCAACCTCGACCTCTCTGACTCACTGACAAG...GAG')}\\$ Length: 666 ID: FAN r2.3ch1Ava g000020.1 Seq('CCCCTCTTGCACAAGTACCCCTCTAGACGAATTCTTATCCTAGGCGGGATCTAC...CTC') Length: 342 ID: FAN r2.3ch1Ava g000030.1 Seq('ATGGAGGAAGAGCCTGTAATGCTCAAGATCGCTCAAATCCTGGACGAGGCT...TAG') Length: 2178 ID: FAN_r2.3ch1Ava_g000040.1 Seq('ATGGTTACCTGTATGCAGAAGTGTATATCAAAGGCTTTCATTCGGGTCATGCGT...TAG') Length: 873 ID: FAN r2.3ch1Ava g000050.1 ${\tt Seq('CCCCTGCTGAATCTCTTGTATCCATAACCATGGTAGCTGTGGTATCATCTCAA...TGA')}$ Length: 609 ID: FAN r2.3ch1Ava g000060.1 Seq('ATGTCGTTCATGAAAGGAGATTTACTTACGAAAACCAGAAAGCTGGTCAAGGGC...GTC') Length: 675 ID: FAN_r2.3ch1Ava_g000070.1 Seq('ATGGTGGCAATCATCACCAGCAGCTCTCTGCTACTTGTCTCCTTCTCCACA...TAA') Length: 885 ID: FAN_r2.3ch1Ava_g000080.1 Seq('ATGAGATGCCCCTCTTTGTTTGCATATATAAAGTCCTCCTCCGCCTTCCAAAGT...TGA') Length: 588 ID: FAN_r2.3ch1Ava g000090.1 ${\tt Seq('ATGATTTGGGAAACCCAATTAGTCAGCGAGGTAATCATAACATTGAATAAGCCG...TAA')}$ Length: 360 ID: FAN r2.3ch1Ava g000100.1 Seq('ATGCCGCGAGGAGCTGGCGAGGCGAGACCAGGGAATGCCGCGAGGAGCTGGCGA...TAG') Length: 462 ID: FAN_r2.3ch1Ava_g000110.1 Length: 318 You'll notice it prints out a shortened version of the transcript. The Python function 'repr' shortens outputs and makes them 'printable'. I've decided to use this because iPython files can become too large and 'break' if the outputs are too long. I'll now load in ananassa's proteins and fragaria vesca's transcripts and proteins in the same way I did ananassa's transcripts . In [6]: type(anan Trs) Out[6]: Bio.SeqIO.FastaIO.FastaIterator In [7]: FXAv = 'Fxa_v1.2_makerStandard_proteins_woTposases.fasta' anan_Prot = SeqIO.parse(FXAv, "fasta") In [8]: vesca v4 = 'Fragaria vesca v4.0.a2.transcripts.fa' vesca_Trs = SeqIO.parse(vesca_v4, "fasta") #vesca transcript | vesca v4 prot = 'Fragaria vesca v4.0.al makerStandard proteins.fasta' vesca_Prot = SeqIO.parse(vesca_v4_prot,'fasta') #vesca_proteins Now lets look through the contents of our newly loaded files! In [10]: print('Vesca Transcripts') printFastaContents(vesca_Trs) Vesca Transcripts ID: FvH4_1g00010.t1 ${\tt Seq('TCTCTTTGTCTCTACACTTTATTCTTCGTCAAGATATATTATTCAGTTTATA}...{\tt AAG')}$ Length: 1436 ID: FvH4_1g00020.t2 Length: 1625 ID: FvH4 1g00020.t11 Length: 2898 ID: FvH4_1g00020.t12 Length: 2830 ID: FvH4_1g00020.t14 Length: 1721 ID: FvH4 1g00020.t3 Length: 1653 ID: FvH4 1g00020.t7 Length: 1611 ID: FvH4 1q00020.t9 Length: 1742 ID: FvH4 1g00020.t6 Seq('AACGAACTGAACACAGATTTACAGTCGATCGATAAAAACCTAAACTTGACAATG...AAA') Length: 1744 ID: FvH4 1g00020.t4 Seq('ACGAACTGAACACAGATTTACAGTCGATCGATAAAAACCTAAACTTGACAATGG...GCA') Length: 1701 ID: FvH4 1g00020.t10 Seq('CGAACTGAACACAGATTTACAGTCGATCGATAAAAACCTAAACTTGACAATGGT...AGG') Length: 931 In [11]: | print('Ananassa Proteins') printFastaContents(anan Prot) Ananassa Proteins ID: maker-Fvb4-2-snap-gene-0.67-mRNA-1 Seq('MKNWEVDGERTEMENLKLGMLDGFLTKNHQKTLKSLFRQSSSTESSQDDSPSSA...KKS') Length: 743 ID: maker-Fvb4-2-augustus-gene-0.46-mRNA-1 Seq('MLLPTFFVVSSAINAHQFFTEETWDSFKQIFDTYMVEASKEWTETNEGSSLLDT...SRS') Length: 199 ID: maker-Fvb4-2-snap-gene-0.65-mRNA-1 Seq('MDLYQAITSSSLIIARPEFNVQSKILKVMSPFKILSPDALNDLGCLHVRPYPDT...EYL') Length: 233 ID: augustus masked-Fvb4-2-processed-gene-0.7-mRNA-1 Seq('MGNYISCTLATPLIKSTMAARVIFPTGEVRQFREPIKAAELMLESPNTFLANSK...FSR') Length: 181 ID: maker-Fvb4-2-snap-gene-0.70-mRNA-1 Seq('MGVVKDGTISGFLPRTQVFGVHYPGYPSSMSRAIDTLGGTQAIHKAHSSASNNN...HHV') Length: 1274 ID: maker-Fvb4-2-augustus-gene-0.60-mRNA-1 Seq('MVQSSSSAVDSSNIGFQGRLEPGQAYLKQSKRGLGADAPNKRPLQVSENESSGM...DNV') Length: 187 ID: maker-Fvb4-2-augustus-gene-0.51-mRNA-1 Seq('MAFSCGSHYYDGEEPEVPLINETKRCAAMDVQVHNELHEGQVELFDFEFWPVEH...YLL') Length: 140 ID: maker-Fvb4-2-augustus-gene-0.56-mRNA-1 Seq('MQLGVGDSTFYFTVYTLVVSKARARSCEQNRIMANIDIEGILKELPNDGRIPKT...IVK') Length: 542 ID: maker-Fvb4-2-augustus-gene-0.55-mRNA-1 Seq('MRWNSGGDAAVTKRMIVLTVLMVMLMGAYASPVRRLMMADIKEQQRRNLLANGL...PIS') Length: 445 ID: maker-Fvb4-2-snap-gene-0.75-mRNA-1 Seq('MISRACCGSIPIVKPAGGVRCEAVVGPFTGRDPNVKKAGWLRQKAAQGIKYEEV...AIR') Length: 333 ID: maker-Fvb4-2-snap-gene-0.73-mRNA-1 Seq('MASALATACCSQLLPPRKGLPFLGPTVSPAFVSLPVGVRAGRPLLAAAAAKSNS...DSD') Length: 319 In [12]: print('Vesca Proteins') printFastaContents(vesca Prot) Vesca Proteins ID: FvH4 c10g00030.1 $\verb|Seq('MTVVHVLATKRSESDVEGANTPSPPNVPYPRQQAATCKPSQGAPRKARTMRGEQ...PLQ')| \\$ Length: 80 ID: FvH4 c10g00020.1 Seq('MTVVHVLATKRSESDVEGANTPSPPNVPYPRQQAATCKPSQGAPRKARTMRGEQ...PLQ') Length: 80 ID: FvH4 c10g00010.1 $\verb|Seq('MTVVHVLATKRSESDVEGANTPSPPNVPYPRQQAATCKPSQGAPRKARTMRGEQ...PLQ')| \\$ Length: 80 ID: FvH4 c1g00020.1 Seq('MELAYKRRPHKLRSSPDFFHCLEGAPSASLSLVTDAATDAPIRVIPAIGKRRQE...LDT') Length: 80 ID: FvH4 c1g00310.1 Seq('MGTCFSVLIRMELARPGDQILGGNHQLYNVLITAHAFLMIFFMVMPAMIGGSGN...YVK') Length: 476 ID: FvH4 c1g00060.1 Seq('MAFRECPDKDPAVPLLRLYVKRKVEKVMRDSIATDYLDRRLPLGFPEGLTLIFP...LEP') Length: 68 ID: FvH4 c1g00130.1 Seq('MHASFIRPGGVAQDLPLGLCRDIDSFTQQFASRIDELEEMLTGNRIWKQRLVDI...VDR') Length: 242 ID: FvH4 c1g00080.1 Seq('MGQSFASLVPTVAAAESAIGLAIFVITFRVRGTIAVESINSIQGCERIRFESLP...QKS') Length: 196 ID: FvH4 c1g00690.1 Seq('MKQETGLLGEHSGKFDYYVQYSKPPEPDPALIPPLPSWDDYPPVPTYTPGPITG...LIA') Length: 273 ID: FvH4 c1g00250.1 Seq('MAFVQRRKGPDVVGAFGLLQPLADGLKLILKEPISPSSANFSLFRMAPVTTFML...SSN') Length: 71 ID: FvH4 c1g00460.1 Seq('MDNAGMPGSEVSDGREEINSNTSGSGPSNSSDRRWRSFDEGVLLEPFSASRDRG...SSA') Length: 204 Now that we have all of our files loaded we can start to do some analysis! We'll start with the transcripts. Since both of these are strawberries it would be interesting to see what transcripts they may have in common. These fasta files aren't in a state where we can perform set theory computations so we'll need to convert them to Python's set type. We'll also need to isolate sequences because the IDs are going to differ. I've made a function to accomplish this. In [13]: def getSeqs(fasta): seqs = []for record in fasta: seqs.append(record.seq) return set(seqs) In [14]: path='/Users/elizadoering/projects/Bioinfo/strawberry/' temp1 = 'FAN_r2.3_transcripts.fa' anan_Trs_Fa = SeqIO.parse(path+temp1, 'fasta') temp2 = 'Fragaria_vesca_v4.0.a2.transcripts.fa' vesca_Trs_Fa = SeqIO.parse(path+temp2, "fasta") s1 = getSeqs(anan_Trs_Fa) s2 = getSeqs(vesca_Trs_Fa) Now that we've isolated just the sequences and converted them to set type we can so some set theory! In [15]: |s3| = s2 & s1s3 Out[15]: {Seq('ATGAAACTCTGTTTCGTTTCTCTAATTCGGCTGCTCAACAGCTCAAGTGCTGAT...TGA'), Seq('ATGAAATCCTCCAGGCAGCACAAGAAGCAACCGCAGCTGAATCGTAGAGCTCCG...TGA'), Seq('ATGAAATTACGGTTCACCCACTGCGTCATAAGCTATAAGAACAGGCTGGACCAA...TAG'), Seq('ATGAACACCAAAACGATGCGTTTGCCTCCACGTCGGATTTCGACGACTATTGCG...TGA'), Seq('ATGAAGAAATTGTGATAAAAGTCGGCATAAATTGCCAGATTTGCAAGACAGAT...TGA'), Seq('ATGAAGAAATCAAGCTCGGCCACACCATCCAACCCCAAATGGGTCCTACCCTAC...TAG'), Seq('ATGAAGAAGATCAGTTTGCTATTAAGGAAGTGCAAGAGCTTGTCAAGCCAGCTA...TAA'), Seq('ATGAAGACTCTGGAAATTCACAGGGGACATAAAGGACATGCGTCTATTGAAGAA...TAG'), Seq('ATGAAGCCTATTGATCAAAGAGAGCGCACTGGGGGGTCGTAGTGACATTCACACT...TGA'), Seq('ATGAAGTTCTACTGCAGCGTCGAAAATCGTGGGGGATGAACCAACGAACTGTATT...TAG'), Seq('ATGAATGACTACGAGTTCATAATCCGTAACACCATCAAGTCACTTGTAGCCTTG...TGA'), Seq('ATGAATGCCATCATCAAGAGTCTTATTCACCCTGCCGGCCCTGGGCTTTCCGGC...TAG'), Seq('ATGACAGTCTCCACAACTAGAACGTCTAGGGATCCAGATATGATTTTCAGGGAT...TGA'), Seq('ATGACGAAGGACAGAAACTTAGGGTTTGCGGTGGATTTCTCGAAGGGAAGCAAG...TAA'), Seq('ATGACGACCCGATTCAAGAAGAACCGCAAGAAGAGAGGCCCACGTGAGCGCCGGT...TAG'), Seq('ATGACGTTGGTGGGAGATCTGCACAGAGCAGCCACGTCTCCGCCGTCGAGAGGT...TGA'), Seq('ATGACTAGTTTTGTTACTTACGCCTACTTCATCATCTATCAACCTCCTTCACA...TAG'), Seq('ATGACTATCTCTGGCGATGATCATCCTCCTCTGGAGGAGGGATTTGTAAAGCTT...TAG'), Seq('ATGAGAACCACAGTAACGGCTACTGTTTCACTATCAAAACCATCGACGGAAGCA...TGA'), Seq('ATGAGCAAGTTCCGGTGGAAGAGCAGTGAATCTGTCCATACAAACTACCTTATC...TAG'), Seq('ATGAGCAGCGGTTGTGGTGATTCAATGGAGGCTCAGTACATAAGGAGACACCAT...TAG'), Seq('ATGAGCGCCTCCAGGTTCATAAAGTGCGTGACGGTCGGCGACGGCGCCGTCGGG...TGA'), Seq('ATGAGCTATCCTTACTCTCCTCCACCACCTTCACCACCTCTCCCTTGTGCT...TAA'), Seq('ATGAGCTCGGTGGCATCTTCTGTTGGTACTTGCAGTCCCAGAAAGCTCAGTACT...TGA'), Seq('ATGAGGACTCTGCCAGCAAAAGAGAAGGTGGATGACCAGCTAGATTTCAAAGAG...TGA'), Seq('ATGAGGGATGACAGGTTGGGATCTTTGCCAGCCCTCCCGGCAGCGCCAAGTCA...TAA'), Seq('ATGATAACGCGATCGAATCTGGCGGATCAGTTGAGAGAGTATCAGATTCGATCG...TAA'), Seq('ATGATGAGAACAAGCTCCTTGTGCTTTTCTAACTTCCAATTACCTAAGTTTTGC...TGA'), Seq('ATGATGCACATGACGTTCTACTGGAGCAGGCAGGTGACGCTCCTGTTCGACTCG...TGA'), Seq('ATGATGGCTGCTGTTCTGATGATGCTCATCTTCTCTCACATTCCTTTCTTCC...TGA'), Seq('ATGCAAAGTCTTCAAGCCAAGGCCTCCCAGTGGAGCGGAGTCGACTCCGCCGAC...TAA'), Seq('ATGCACGATTTCTGCTTCACGATCCCGTACGGGCTGTTGCTGTTGGGTGGAGGA...TAA'), Seq('ATGCAGCACAAAAGGCAAAAACAATATCATCATAATCGTGAAGAAGATGAGACC...TAG'), Seq('ATGCAGTCCGCCAAGGGTGTCAATGCTGCCAAGCATTTCACCAACATGTTTGCT...TAA'), Seq('ATGCATCGTTCAACAGCTACTAGCTTCCAACGCTACCTCTTCAGCCATCAAAAC...TAG'), Seq('ATGCCCCAGAAGTGCCTCATTATCGATCTTCTGTAACTCCTCTAGTACGGTTC...TAA'), Seq('ATGCCGAAGAACAAAGGAAAGGGAGGCAAGAACCGCAAGAGGGGAAAGAACGAA...TGA'), Seq('ATGCCGGCGCCGGCGAATCATTCCTGGCCGGCGGTCGACGAGCTTGTCCCGGA...TGA'), Seq('ATGCCGGCGGAACTACCCCGAATCAGCAGCATACCAACTTATGGAAATCTCAAG...TAA'), Seq('ATGCCTAAATGGAATTTGATTCTGGGTGGGAGTAGTGGAACGGGTAGTCTTCGA...TGA'), Seq('ATGCCTCAAGGAACCCTAGATGTTCATCTTATGAATGCCAAGGGTCTAAAAAAC...TAG'), Seg('ATGCCTCGTCTCTCTCTCTCCCTTCTCTCCTTCTCGTCCTCGTTGTAGCC...TAG'), Seq('ATGCGTGTCTTGGAGGAGCTGGATTTGAGTTACTGTTGCCAGCTAACTGATGTT...TGA'), ${\tt Seq('ATGCTGCGAGAGCAGGAGCGAGGGTCGCTCAGGCGGAGCTTGTCGAGGAGGCGG...TAA'),}$ Seq('ATGCTTCCTATTCTTTCATCATCATCAAAGCTTCATGCTCGACAAACCAAA...TGA'), Seq('ATGGAAGAGCAGCGGAGAAGCAAGTAAGAGAGGAGGTGCAGCGGAAAGAGAAG...TAG'), Seq('ATGGAAGCCAAACTCCTCCATTTCCTCGCTTCAAAACCGGTAGCACCACCCAAC...TGA'), Seq('ATGGAAGTAACTCACTCACTAAATCGCTTCTACTTTGATGTCACTCCCCTTGAT...TGA'), Seq('ATGGAAGTTGTGATCTCAATCTGTGCTTCTGTAGTGGGAGAAACTCTCCAATAT...TGA'), Seq('ATGGAAGTTTCGTCATGGGATGGAACCCTTTCGTCACGCGACTTCCAGGTTGCA...TGA'), Seq('ATGGAATGGGATGGTAGAAACTTGGCGAGGACGGATTGGGGGAGACTAGTAAAG...TAG'), ${\tt Seq('ATGGACAATTCAGTTCTTATACATCCCACGAGTTCTGAAATCCCGGTCGATCTC...TGA'),}$ Seq('ATGGACAGAGCAGAGCTCCGGAGGGTTTTCCAGATGTTCGACCGGAACGGGGAC...TAA'), Seq('ATGGACCCTTATGCACCTCAGGCTTGGGTGTTAGAAATGATCAGAAATCCCATA...TGA'), Seq('ATGGACTACAGCCAAGCAGCGGTGAAGCTGCTCTGCGCCCATCTAAAGCACGCA...TGA'), Seq('ATGGAGAAGGCTCTAACCAAAGTGAGCAGCTTGAAGATTGGGAACTCATGGATC...TGA'), Seq('ATGGAGATCGACTTGTCAAGCATCACTCTCCGTCCATTCCGGCTATCGGATGTC...TAG'), Seq('ATGGAGATCGCTCCTAGTTCGTACTACACCAACGTGAAGAGGTACTGGAGGAGG...TGA'), Seq('ATGGAGATCTCATCTCTAACTCTATCCACTCCAAGATCTCCACCTTCAGTTCCT...TGA'), Seq('ATGGAGCTACCAAACCAAAACTTTCAGCTTTACCAGTATGTCTTCCCCTCTCAT...TGA'), Seq('ATGGAGCTCAAGGTCGCTCAAGTGTTCATCTCCTCACTTAGAATCAGTGTCTTC...TAA'), Seq('ATGGAGCTTTCCATCCATGGAGCAGTTCCGGAATTAGAGAACACATGCAGTTTG...TGA'), Seq('ATGGAGGAAGTGAAGGTAATAGGATTCTGGGCAAGTCCATATGTTTACAGGGTC...TGA'), Seq('ATGGAGGCACTTGCTTTTAGCGCTTGTGAGTTGGTTCTTCATTTACGGTTTGTG...TGA'), Seq('ATGGAGGCGGTGACAAAGATGGTGTCCGAGAGACCGGTGGTGATCTTCAGCAAG...TAA'), Seq('ATGGAGGGTCTGATCCCATTGGTGTTCAAGGCGATTAAGAGGAACAGAAGTCGG...TAG'), Seq('ATGGAGTCGAATCGCAAGCGCAAAGGGTTCATGAAGGGGAAGCTGATGCCATTC...TAA'), Seq('ATGGAGTCTAATGAAGCTAAGCACCACTGTGGTGAAAGGTTAATTTTGTATATA...TGA'), Seq('ATGGAGTCTTATAAACACTTGTTGAGCACAGAAGGATGTAGCAGCAGTGAGTCA...TGA'), Seq('ATGGATATGAACATGACGATGAACAATATGACGATGGATATGACTATGCAGATG...TGA'), Seq('ATGGATATGGTGAAGGGAAACTCAAGAAAGAAGAATCAGACTAAGAAACCGTCG...TAA'), Seq('ATGGATCCTCACAATATAGCAGCATTGCAAACTACTGTCGTATTCGATGAAAGG...TGA'), Seq('ATGGATCCTCTGTTGTTTGTGTTCGTCCTTGTTGCTCTTCCCCATATCAGAA...TAA'), Seq('ATGGATGACTTGGAGGAGTCAATAGGCCACGGAAGAACCAGCTCGCGCTGCTGC...TGA'), Seq('ATGGATGATGTTATCCCACAATCTGAGAGCCTGAGCCTGAGGATTCCGTCCAAT...TAA'), Seq('ATGGATGCAGAAAAGCGAAAGCGCGACGATGAGGGCAGAATTGACGGGAAGAGA...TAA'), Seq('ATGGATGTGCTAGGGGTGAACAACCGGAGCCCGAAGCAGAGCAAGAAGCAGA...TAA'), Seq('ATGGCAACAGTACCATGCCTAATACTCTTCTTTCTGGTCTCCTTTTCCATCTCA...TAA'), Seq('ATGGCAAGGTTTAGTGTAGCTCTGTGGCAATGGGTGATGGTACTTCTGTTGATT...TGA'), Seq('ATGGCAATGAAAAGGGAAGCATTGTCTTCGTTGCTGCTAGTGACATTACTGACC...TAG'), Seq('ATGGCAATGGGGATAACTTGTTGGATGTTTTTATCGTAACCCTAAGCGCAAGT...TAG'), Seq('ATGGCACGACGCACTGGTGGTCTGTTTCTCGTTGCCTATTTCGTTTTTCCCAGC...TAA'), Seq('ATGGCAGCCAAAGAACCAGAGATAATCAGAGACCAGAGGACCAAATGAGGAACTGG...TGA'), Seq('ATGGCAGCGATTTACAGCCTCTACATCATCAACAAATCAGGCGGCTTGATCTTC...TGA'), Seq('ATGGCAGCTCTCTCCGGCACCATGGTCAGCACCTCCTTCATCCGCGGGCAGCCC...TAA'), Seq('ATGGCAGCTGAGAAGGCAATGAAGAAGAGTCACTCAAGGAGCAACTATAAGGCA...TGA'), Seq('ATGGCAGCTTCTTCCTCCTCCACCTTGTTCCTCTCACCCTCCATCACCAGGCCA...TGA'), Seq('ATGGCATTCAACGTGACCCCAAAGCATAAGAACCTCAAGCTTGCACTTCTCGTT...TGA'), Seq('ATGGCATTGAGTATGTTTTGCGCAACCAGGGCTTCTCTGTATCTGATGAACATG...TAG'), Seq('ATGGCCAAGAGTTTCAAGCCTCTTTGCTCCTTGTTTGTTCTGGTTCTTGTAATA...TAA'), Seq('ATGGCCACCACTGGTCTAGACTTGTGCCAAGACCTTCCATCTCTCATGCGATCA...TAA'), Seq('ATGGCCATTAGAGCAGGTCTAGACCTACTGCACTCTATGCAGACAAGAGATGTT...TGA'), Seq('ATGGCCGCGTCGTCGAATCTGCCCCGTTGCCCAAGAGATCATTCTGCCCCAC...TAA'), Seq('ATGGCCGGATACAACAATATTACTGCTGCATTTTGCCGTGATAGTTATAGCTGTT...TAA'), Seq('ATGGCCGGGTACAGAGCCGACGACGAGTACGATTACCTCTTCAAGCTGGTCCTC...TAG'), Seq('ATGGCCTCTTCCCAGCTCTTCTTCATCAGCCTTTCTAGCAATTGTTGCC...TAG'), Seq('ATGGCCTCTTTTGTTTCAACTTCCCCAGTCCCTGTGGCCACTCTTTCCTACACT...TGA'), ${\tt Seq('ATGGCCTGCTGCCGATTTCCATGTTCAGAAAGAGGTGCACGGGG...TAG'),}$ Seq('ATGGCGAAGCATCATCCTGACCTGATTATGTGCCGGAAACAACCAGGAATAGCT...TAG'), Seq('ATGGCGACGGCGCGCAAGCAGAGGTTTCGAGGAGGATGATGGGATGCAAAGGA...TAG'), Seq('ATGGCGACTTCAAAGCTCCAAGCGCTGTGGAACCACCCCGCCGGCCCTAAAACC...TGA'), Seq('ATGGCGGAACAGACGGAGAAGGCTTTCCTCAAACAGCCCAAGGTTTTCCTCAGC...TAA'), Seq('ATGGCGGGTCGAGGCAAGGCTTTGGGATCTGGGGCGGCCAAGAAGGCCACGTCA...TGA'), Seq('ATGGCGGTTGCTAGCGCTCATAGGATAAGAGGAGGGTCTCTGACGGCGGCCACC...TGA'), Seq('ATGGCGTCCGATACGGCGTCGTCTGTGGACGAACACAGGCACAGCTTGTCACT...TGA'), Seq('ATGGCGTCCTCGGAAGTGGGTTTTTGGGGGCAGATCCGTATTTGGGATCCG...TAG'), Seq('ATGGCTAACTCACCTTCTATTCCATCCCTACCCTTGGAGTGTCATCTAAACTA...TGA'), Seq('ATGGCTACAGCTGGAGATACTCACCTCGGAGGTGAAGATATTGACAACAGATTG...TAG'), Seq('ATGGCTAGAGTCCAGAACTTGTCACTTGTAAGCCACCTGAAGAGCAGTGAAG...TAG'), Seq('ATGGCTAGTGCTAAAACTACCTACCGTCTTCTTCTTCTTCTTCTCTAT...TGA'), Seq('ATGGCTATGGCCAGTGACTACTGGAAAGTGGTGGGCGATGCTTTCTGCAACGTA...TGA'), Seq('ATGGCTCAGTCGAGTTCTCTATCTGCAGAAACCGAGTCACTGAGTCAGGTCCTG...TAA'), Seq('ATGGCTCGGCAGCTTCTTCTGTCACACCCCCTACCAAGCATACTTCAAGTCGGC...TGA'), Seq('ATGGCTCTCAGACTGAATTCCACGGTGAACACCTTCCCATCTCACCAGCTCATG...TAA'), Seq('ATGGCTGCTATTTCCAATTCGCTGACTTCTCCCCACCTATCTTCTGGATCTTCT...TGA'), Seq('ATGGCTGTGCAAGAACACAGCTTTCAGTGGCCTCCAATTGGAGCTCCACTAAAT...TGA'), Seq('ATGGCTTCACCAACGCTTGCATGCTTCACTCTTCTTATATCAGTTATTTCCACT...TAA'), Seq('ATGGCTTCCATGACCATGACAGCCTCATTCCTAGCCTCAACCATCACCAAACCT...TGA'), Seq('ATGGCTTCCGCTTGTTGCCGCCTTATCTCAACTACCACCTCAGCCACTGTAACC...TGA'), Seq('ATGGCTTCTTCTAGTGGAACATCCTCTGTTTCCTCCATGATTCAAAACTCTGGC...TGA'), Seq('ATGGGAGAAAAGAAGAAGAAGAACAAGAACAAGAACACAC...TAG'), Seq('ATGGGAGAGGTTCTAAGTGAAGAACAGATTGCTGAGTTTCAGGAAGCCTTCTGT...TGA'), Seq('ATGGGGAACTGCGGTTTGAGGCGTGATTCCGCGGCGGTTTGGGCCGACAACAAC...TAG'), Seq('ATGGGGAAGCTTAGTTTTGGGAAGGTGCTAGACACTCTTTGTCTTCTTCTGGC...TGA'), Seq('ATGGGGACCGAGATTTTAAGACCGCAGGATTGTCTGGTCGAGAGAATCAGAGTC...TGA'), Seq('ATGGGGACGAACTATGTTCTCGCTCTGCTCTTGGTTCTGATTCTCAACTTGGGA...TAA'), Seq('ATGGGGAGGAACTTATGGTGTGTCAGTACTCTCATATCTGGCCTAACAATGAGA...TAA'), Seq('ATGGGGAGGATTTACGTGGTGGAATTGGAAGGCAGAAGTTACAGATGCCGGTTC...TGA'), Seq('ATGGGGCTTCTTTTCCGAAGTCTCGTGATACTAGTGGTTCTGATATGGTTGTCA...TGA'), Seq('ATGGGGCTTGTGAATACCGCTTTTGGCTGTGCTGCTTCTTTTGTGTTCTACAT...TGA'), Seq('ATGGGGGCTTCATCTAGTGCTTCTTCCTCCAAATCAAAGAAATCCACCAAAAAC...TAG'), Seq('ATGGGGTACCTTTCTTGCAAAGCTGAATCTGCTATCTCCGTCGCCAATTCCCAG...TAA'), Seq('ATGGGGTTACAGGGCCAGATGAGCGACTTGTCGTCCGACTCCATCCCAATCCTC...TGA'), $\verb"Seq" ('ATGGGTAGGCCATTGGTTTTCGAGATCTTGGAGAAACCAGTCACTAGTTGCTTT...TAA')",$ Seq('ATGGGTCCTCATCAGCGTCCTCTTTTTTTTTCCCTTGATTTGAAGGGTCTGCTG...TAG'), Seq('ATGGGTGTTCTTCATCATCTCAACATTGCAGTCACTGTAATCTCATTCTTT...TGA'), Seq('ATGGGTTCAAGAGGAGGCGCGACGGTTTTGCAGCAGCTAAATCGGGCGAGTCGA...TAG'), Seq('ATGGGTTCGAGTAAATGCTTAGATGCCTCTGCAGGGAACAAGGAACGGTTGCGG...TAA'), Seq('ATGGGTTTTTCCAAGAAGTCACAAGTGGTGGACGGAGGCCTGGAATCCGAGGGG...TGA'), Seq('ATGGTAGATAACAGAGGCAACAGCAACAGTATTATTAGTTCAGACTCCAGA...TGA'), Seq('ATGGTGAACTTAGTAGCACTACAGAAGCCCTTGCTGCATGGCCTGATGAAAATG...TAA'), Seq('ATGGTGAAGGCACATAGAAGCCGAAAAGAGATGAAAAGGGATGAACCGAAACTC...TGA'), Seq('ATGGTGAAGTTCCTCAAGCCTAACAAAGCCGTGCTCCTACTCCAGGGGAGGTTC...TGA'), Seq('ATGGTGAAGTTCTTGAAACCCAACAAGGCCGTGATCCTTCTCCAAGGTCGCTAC...TGA'), Seq('ATGGTGATAATTACATACGTCTTTTTCAGGGTGTTATGTGTCAAGTACAGCTGT...TGA'), Seq('ATGGTGGGATTAGGGCCCTCGGCCGGCCTCCCAATCGGCCCCATTTGCATGATG...TAG'), Seq('ATGGTTCAGAAGCTTGAAGCTATAAAGGGTGGTGGAGGATCCATCAGGGTAGGG...TAA'), Seq('ATGGTTCCGCGACCGGTCTCGTCTCGAGAAGAAGAAGAAGAAGAAGAAGAGCTTTCT...TAA'), Seq('ATGGTTCGTAGGTACATTCGCTATCGAAGGTTCATGAAACATAAACAAAGTTC...TGA'), Seq('ATGGTTCTCTTGTTACAAAAGATTGACAAGCTCTACTCAAAGCTGGAGAATTAC...TAG'), Seq('ATGGTTGATCATGAGGAAGAACAAGATCGGTTGCTGCCAATTGCAAACGTAGGG...TGA'), Seq('ATGGTTGTGCAAGAACACAGCTTTCAGTGGCCTCCAATTGGAGCTCCACTTAAT...TGA'), Seq('ATGGTTTCTGCTTATTCTTCTTCTTCCAATGTTTCGGCCTTGTTCATTTTC...TAG'), Seq('ATGTACTTGAAGAAGCCCCAATGGACGGAGGGAATCACACCCAAGCCTTCGCCG...TGA'), Seq('ATGTCAAGGCAATCGAGCAGGCGTGTGAGTTTTAGTCCTGATGTGTATGATCAT...TAG'), Seq('ATGTCAGATGATGATATTCAGTATCACAGTCATCAGGCATTGCCAATCCTCAT...TAG'), Seq('ATGTCAGGGCGCGCAAGGGAGGCAAGGGATTGGGAAAGGGAGGAGCCAAGCGT...TGA'), ${\tt Seq('ATGTCAGGGCGCGAAGGGAGGCAAGGGATTGGGCAAAGGAGGAGCCAAGCGT...TAA'),}$ ${\tt Seq('ATGTCCAAGACTTCTCTCACGACATGCCGCCGGAGAAGAAGCTTTATGCA...TAG'),}$ Seq('ATGTCCAGGCCATATGGTGAGCAACCTCCACCACCACCACAAGCCTTTATAATA...TAA'), Seq('ATGTCCGACGAGGAGCACCACTTCGAGTCTAAGGCCGACGCCGGAGCCTCCAAG...TAG'), Seq('ATGTCCTCCAAAGCCCTTCTGGTTTTCTGTTACTGGGTGTGCTGCTTCTGATC...TAA'), Seq('ATGTCCTCCCCAGCTCTTCATAGTTCTTGCCATCCTTGCCATTGTCTCCCCT...TAG'), Seq('ATGTCGATCATCCCCAGTTTCCGACGGAACAGCATCATCTCGACCCCTTC...TAA'), Seq('ATGTCGTCGAGGAATTTGAGAGGCGGCGGCGTTGGATCTTCGGCGGAGGAGTTG...TAA'), Seq('ATGTCGTGGCAGACTTATGTAGATGAACACTTGATGTGTGACATTGATGGCAAT...TAG'), Seq('ATGTCTCACCTTAACTTCCTCCTCCTCCTTCAGTTTTCTCTGTTCTTCATCAAC...TGA'), Seq('ATGTCTCACGTCTCCAAAGCCCTAATGGAGGTCGAGGTCAAGCTGCGGCTGCCG...TGA'), Seq('ATGTCTCTGAGAATCAAAGTGGTGGTGGACAAGTTCGTACAGGAGCTCAAGGAA...TGA'), Seq('ATGTCTCTTATAACCGATGAGGTGAGAGCCCAAGGCTGATGAGCTCTACCATGGT...TAG'), Seq('ATGTCTTCGGAGCTTCATGACCGTGCCGGATCGGAAGTTGCGATAGATCAGCCA...TAA'), Seq('ATGTGTAGAGGTGTAGAAGAAGAGGCCATTTCAGTTAAAGCTTCAAAAGGGTTT...TAA'), Seq('ATGTGTCACCCAGCGGGAGTGTCTTTGGTTGCACACAGAGACTTGGCGGTGCAA...TGA'), ${\tt Seq('ATGTTCGACATTCCCTCTGCCACACTAAACGCAGCCTATCTCGACATGGGGTAT...TGA'),}$ Seq('ATGTTGCTTCTTACACCAGCATCCACTTCTCTTTTGTCACACCCAGCAAATGGT...TGA'), Seq('ATGTTGTCTCATCCATTTTTCCGTTGTGCCCATGCCAAGTTGTCCTTGCTTATC...TGA'), Seq('ATGTTGTTAATAGAGGATCGGCTTCAGAGAAACAAGAACATGGCCAAGTCCACC...TGA'), Seq('ATGTTTCCGAACTCCATCACCCTCGTCCTTGCTGGTAATTGGAATAGCAGCCCC...TAG')} In [16]: print(len(s3)) 183 In [17]: print ('s1 length: ', len(s1), 's2 length: ', len(s2)) s1 length: 161908 s2 length: 64475 That got a little messy but we can see some similar sequences! Not many, but still something. Now we can look into other things like length and GC content. I've referenced the BioPython cookbook and found two functions that will print out some plots. Pylab is a great resource for these types of visualizations. In [18]: def getGCContentPlot(file): gc values = sorted(GC(rec.seq) for rec in SeqIO.parse(path+file, "fasta")) avg = np.average(gc_values) pylab.plot(gc_values) pylab.axhline(y=avg, color='r', linestyle=':') pylab.title("%s %i sequences\nGC%% %0.1f to %0.1f\n Avgerage: %0.1f" % (file, len(gc_values), min(gc_values), max(gc_values), avg) pylab.xlabel("Genes") pylab.ylabel("GC%") pylab.show() I've run into some parsing errors with the fasta object so we'll parse that again within the plot functions. I'll run it with ananassa first! In [19]: getGCContentPlot(FAN_r2) FAN_r2.3_transcripts.fa 167721 sequences GC% 29.9 to 69.3 Avgerage: 46.6 70 65 60 55 50 45 40 35 30 25000 50000 75000 100000 125000 150000 175000 Genes Now I'll run it with vesca! In [20]: getGCContentPlot(vesca v4) Fragaria_vesca_v4.0.a2.transcripts.fa 64597 sequences GC% 28.5 to 79.5 Avgerage: 43.6 80 70 60 40 30 10000 20000 30000 40000 50000 60000 I've also found a plot that looks for average length. It also utilizes Pylab and had the same parsing errors so I'll reparse the files again to generate the plots. In [21]: def getSequenceLenPlot(file): sizes = [len(rec) for rec in SeqIO.parse(path+file, "fasta")] pylab.hist(sizes, bins=20) avg = np.average(sizes) pylab.axhline(y=avg, color='r', linestyle=':') pylab.title("%r %i sequences\nLengths %i to %i\nAverage %0.1f" % (file,len(sizes), min(sizes), max(sizes),avg) pylab.xlabel("Sequence length (bp)") pylab.ylabel("Count") pylab.show() Lets run it with the same two files! In [22]: getSequenceLenPlot(FAN r2) 'FAN_r2.3_transcripts.fa' 167721 sequences Lengths 300 to 21864 Average 1272.9 100000 80000 60000 40000 20000 0 0 5000 10000 15000 20000 Sequence length (bp) In [23]: getSequenceLenPlot(vesca_v4) 'Fragaria_vesca_v4.0.a2.transcripts.fa' 64597 sequences Lengths 159 to 19391 Average 2001.3 20000 15000 10000 5000 0 2500 5000 7500 10000 12500 15000 17500 20000 Sequence length (bp) BioPython also has some pretty cool translation capabilities. Within their cookbook I found a way to print the proteins associated with DNA strands. I've built a function to do this below. I've also stopped the output at 10 due to computational limitations. In [24]: def DNAtoProtein(fasta): i=0for record in fasta: i+=1 coding DNA = record.seg print(repr(record.seq),'->',repr(coding_DNA.translate())) if i>10: break We'll run it with the same two files we've been working with! We have to reparse the files again because of the BioPython limitations. In [25]: anan Trs Fa = SeqIO.parse(path+temp1, 'fasta') DNAtoProtein(anan_Trs_Fa) Seq('ATGTCTCTGCTCACAGATCTCATCAACCTCGACCTCTGACTCCACTGACAAG...GAG') -> Seq('MSLLTDLINLDLSDSTDKIIAEYIW IGGSGMDLRSKARTLPGPVSDPSKLPKWN...ILE') Seq('CCCCTCTTGCACAAGTACCCCTCTAGACGAATTCTTATCCTAGGCGGGATCTAC...CTC') -> Seq('PLLHKYPSRRILILGGIYSWIIPVK FDPNHRCFINPLGDPTRAHCPSYFHTGRS...ILL') Seq('ATGGAGGAAGAGCCTGTAATGCTCAAGATCGCTCAAATCCTGGACGAGGCT...TAG') -> Seq('MEEEKPVMLKIAQILDEARASNAIH HRKCKELYALRSKTASSSVFFTSLCKTLL...QR*') Seq('ATGGTTACCTGTATGCAGAAGTGTATATCAAAGGCTTTCATTCGGGTCATGCGT...TAG') -> Seq('MVTCMQKCISKAFIRVMRSMWPGIN GNAGGSASVVSNMRKRAVQVSRFMLQMMQ...SD*') Seq('CCCCCTGCTGAATCTCTTGTATCCATAACCATGGTAGCTGTGGTATCATCTCAA...TGA') -> Seq('PPAESLVSITMVAVVSSQENMVSRT GRHLQRYSNGRRQVVGCIPYRYKKAKQSL...SL*') Seq('ATGTCGTTCATGAAAGGAGATTTACTTACGAAAACCAGAAAGCTGGTCAAGGGC...GTC') -> Seq('MSFMKGDLLTKTRKLVKGLAKAEPV WYKAMEQAPPPTFPRADAIQRITLPEDVY...GGV') Seq('ATGGTGGCAATCATCACCAGCAGCTCTCTGCTACTTGTCTCCTCCACA...TAA') -> Seq('MVAIIITSSSLLLVSFSTIFIVSLI SSLCCIPSLSAAPSTDTFVFGGCTQEKYA...GK*') Seq('ATGAGATGCCCCTCTTTGTTTGCATATATAAAGTCCTCCTCCGCCTTCCAAAGT...TGA') -> Seq('MRCPSLFAYIKSSSAFQSYTLPFCS LSLSLSLSLSDFLANARPPPSMADQLTDD...AK*') Seq('ATGATTTGGGAAACCCAATTAGTCAGCGAGGTAATCATAACATTGAATAAGCCG...TAA') -> Seq('MIWETQLVSEVIITLNKPHSPNLAP TKPRSPPTRSTAGIQPSPANPCDLPLLTT...LT*') Seq('ATGCCGCGAGGAGCTGGCGAGGCGAGACCAGGGAATGCCGCGAGGAGCTGGCGA...TAG') -> Seq('MPRGAGEARPGNAARSWRGETRGVC EVSGVRKWMAEPAESVRTGLTIGVREWMA...IH*') RCLGESVCAWLGTVARELGVARRLWRDQW...AW*') Now lets take a look at vesca! In [26]: vesca Trs Fa = SeqIO.parse(path+temp2, "fasta") DNAtoProtein(vesca Trs Fa) Seq('TCTCTTTGTCTCTACACTTTATTCTTCGTCAAGATATATTATTCTTCAGTTTATA...AAG') -> Seq('SLCLYTLFFVKIYIIQFITIYMFIS GYPKDTIMR*SNI*SDLSDSFLIQFVPIS...VYE') GSVFLWIYIRQ**HCFLGR*LQVNIYASG...KWE') GSVFLWIYIRQ**HCFLGR*LQVNIYASG...CFA') GSVFLWIYIRQ**HCFLGR*LQVNIYASG...LFR') GSVFLWIYIRQ**HCFLGR*LQVNIYASG...FVS') GSVFLWIYIRQ**HCFLGR*LQVNIYASG...CFA') GSVFLWIYIRQ**HCFLGR*LQVNIYASG...CFA') GSVFLWIYIRQ**HCFLGR*LQVNIYASG...FVS') Seq('AACGAACTGAACACAGATTTACAGTCGATCAATAAAAACCTAAACTTGACAATG...AAA') -> Seq('NELNTDLQSIDKNLNLTMVLSFFGF TSDSDNIVFWVVNSR*TFMQVE*EGIEKL...KQK') Seq('ACGAACTGAACACAGATTTACAGTCGATCGATAAAAACCTAAACTTGACAATGG...GCA') -> Seq('TN*TQIYSRSIKT*T*QWFCLSLDL HPTVITLFSGSLTPGEHLCKWSRRVLKNS...CFA') Seq('CGAACTGAACACAGATTTACAGTCGATCGATAAAAACCTAAACTTGACAATGGT...AGG') -> Seq('RTEHRFTVDR*KPKLDNGSVFLWIY IRQ**HCFLGR*LQVNIYASGVGGY*KTR...PVK') /usr/local/lib/python3.8/site-packages/Bio/Seq.py:2334: BiopythonWarning: Partial codon, len(sequenc e) not a multiple of three. Explicitly trim the sequence or add trailing N before translation. This m ay become an error in future. warnings.warn(You'll remember we also uploaded some proteins - lets take a look at the length of those! We'll use the same method for length plots that we did for the transcripts! In [27]: getSequenceLenPlot(FXAv) 'Fxa_v1.2_makerStandard_proteins_woTposases.fasta' 108087 sequences Lengths 1 to 5775 Average 380.4 50000 40000 30000 20000 10000 0 0 1000 3000 4000 5000 6000 Sequence length (bp) In [28]: getSequenceLenPlot(vesca_v4_prot) 'Fragaria_vesca_v4.0.a1_makerStandard_proteins.fasta' 28588 sequences Lengths 4 to 5392 Average 391.6 12000 10000 8000 6000 4000 2000 0 0 1000 2000 3000 4000 5000 Sequence length (bp) Since BioPython is open-source I decided to include a function that a user submitted, but hasn't been fully incorporated into BioPython's source code. Its purpose is to clean sequences. Take a look at it here: https://biopython.org/wiki/Sequence_Cleaner In [29]: **def** sequence cleaner(fasta file, min length=0, por n=100): # Create our hash table to add the sequences sequences = {} # Using the Biopython fasta parse we can read our fasta input for seq record in SeqIO.parse(fasta file, "fasta"): # Take the current sequence sequence = str(seq_record.seq).upper() # Check if the current sequence is according to the user parameters len(sequence) >= min length and (float(sequence.count("N")) / float(len(sequence))) * 100 <= por n</pre> # If the sequence passed in the test "is it clean?" and it isn't in the # hash table, the sequence and its id are going to be in the hash if sequence not in sequences: sequences[sequence] = seq_record.id If it is already in the hash table, we're just gonna concatenate the ID of the current sequence to another one that is already in the hash table else: sequences[sequence] += "_" + seq_record.id # Write the clean sequences # Create a file in the same directory where you ran this script with open ("clear " + fasta file, "w+") as output file: # Just read the hash table and write on the file as a fasta format for sequence in sequences: output_file.write(">" + sequences[sequence] + "\n" + sequence + "\n") print("CLEAN!!!\nPlease check clear " + fasta file) We'll run it with ananassa first. In [30]: sequence cleaner(FAN r2) CLEAN!!! Please check clear FAN r2.3 transcripts.fa Now we can look at how much of it was cleaned. In [31]: b = os.path.getsize("clear FAN r2.3 transcripts.fa") og = os.path.getsize(FAN r2) print('Old size: %i\nNew size %i'%(og,b)) Old size: 221989226 New size 212615072 We'll also run it with vesca! In [32]: sequence cleaner(vesca v4) CLEAN!!! Please check clear_Fragaria_vesca_v4.0.a2.transcripts.fa In [33]: b = os.path.getsize("clear Fragaria vesca v4.0.a2.transcripts.fa") og = os.path.getsize(vesca v4) print('Old size: %i\nNew size %i'%(og,b)) Old size: 134199917 New size 130193754 In []: