project

May 4, 2019

1 36610 - Python and Unix for Bioinformaticians

1.1 Project 6: Resistance to antibiotics

1.2 May 4, 2019

1.2.1 Peter's discussion

- res genes can look alike/similar
- he uses .find()
- check only part of the read, if suffienctly enough kmers match, then go through each kmer of that read
- do NOT loop over the res gene dict (-> slow)

Set of RES kmer's to quickly check if sequencing read needs further investigation

In [ ]:
```
# Pseudocode
#
# for line in resistance_genes_file:
#    get header and sequence and store in a list
#    generate a set of kmers and a dictionary of kmer and a value of 0
#
# for line in sequencing_read_1 and 2:
#     isolate read sequence
#     check if at least 2 out 3 read kmers match the resistance gene set
#     if so:
#         for each kmer in the read sequence:
#             if kmer in resistance gene dictionary:
#                 increase its value by 1
#     else:
#         check if the reverse complement has at least 2 out of 3 matches
#         if that is the case:
#             generate kmers for the reverse complement, and increase matching ones by
#     otherwise discard this read (and do not generate kmers for it step by step)
#
# for resistance_gene in list_of_resistance_gene_header_and_sequence:
#     get the header
#     get the sequence
```

```
#     get the length of the sequence
#     for each kmer in that sequence:
#         add the depth of each kmer to a temporary list
#         check if that addition would result in a coverage <95%
#         if it passes:
#             continue collecting kmer depths for that sequence
#         if not:
#             discard this sequence
#     since >95% coverage, and check for >=10 depth:
#         we have a winner: store gene name, coverage and minimum depth in a final lis
#
# sort the list according to coverage and depth, and print it.
```

In [17]:
```
# Libraries
import gzip # Provides support for *.gzip files
import time # Used for runtime measurement
time_start = time.time() # start timer

# User defined kmer length
kmer_length = 19

# Functions
def reverse_complement(DNA_sequence):
    """Reverse complement a given DNA sequence"""
    DNA_translation_table = str.maketrans("ACGT", "TGCA")
    rev_compliment = DNA_sequence.translate(DNA_translation_table)
    return rev_compliment[::-1]

def read_in_seq_reads(read, kmer_length):
    """Go through sequencing reads and check if at least 2 out of 3 kmer match
    the resistance gene kmer set. If so, generate all kmers for that read and
    increase the count for matching kmers. Also consider the reverse complement."""
    raw_reads = gzip.open("data/Unknown3_raw_" + read + ".txt.gz", "rt")

    # Variables
    read_seq = ""
    rev_read_seq = ""
    read_kmer = ""
    line_count = 3

    for read_line in raw_reads:
        if (line_count % 4 == 0):
            read_seq = read_line.rstrip() # arrived at read sequence

            # Generate 3 read kmers and check if at least 2 match to the
            # resistance gene dictionary. If so, generate all kmers for
            # that read, otherwise try the reverse complement. If the
            # reverse complement does not have at least 2 matches, ignore
```

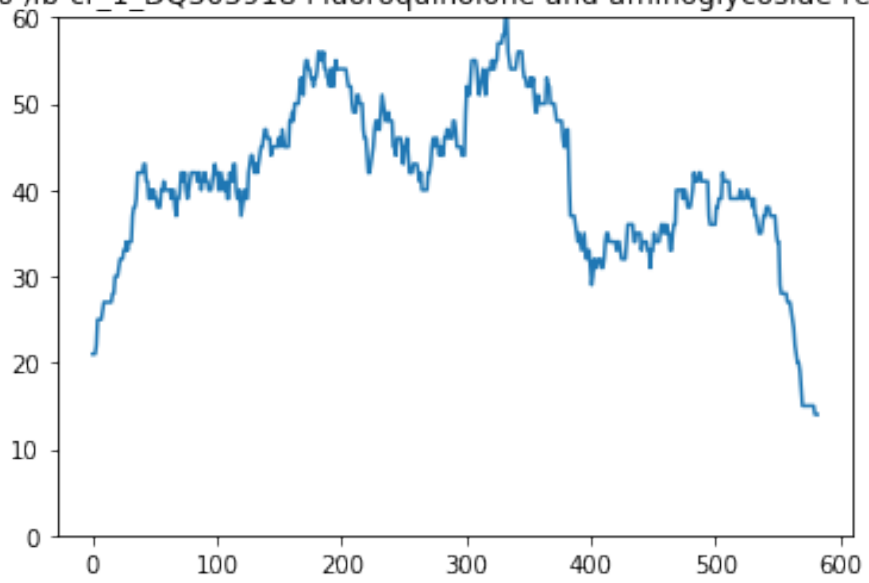
```
                # this read.
                read_kmer_set = set()
                read_kmer_set.add(read_seq[1:1+kmer_length])
                read_kmer_set.add(read_seq[41:41+kmer_length])
                read_kmer_set.add(read_seq[81:81+kmer_length])

                if len(read_kmer_set.intersection(ResKmerSet)) >= 2:
                    for j in range(0, len(read_seq), 1):
                        if (j < len(read_seq) - kmer_length + 1):
                            read_kmer = read_seq[j:j+kmer_length]
                            if read_kmer in ResKmerDict.keys():
                                ResKmerDict[read_kmer] += 1
                else:
                    rev_read_seq = reverse_complement(read_seq) # reverse complement

                    read_kmer_set = set()
                    read_kmer_set.add(rev_read_seq[1:1+kmer_length])
                    read_kmer_set.add(rev_read_seq[41:41+kmer_length])
                    read_kmer_set.add(rev_read_seq[81:81+kmer_length])

                    if len(read_kmer_set.intersection(ResKmerSet)) >= 2:
                        for j in range(0, len(rev_read_seq), 1):
                            if (j < len(rev_read_seq) - kmer_length + 1):
                                read_kmer = rev_read_seq[j:j+kmer_length]
                                if read_kmer in ResKmerDict.keys():
                                    ResKmerDict[read_kmer] += 1
            line_count += 1
        raw_reads.close()

    print("Processing resistance genes...")
    fasta = []            # Will contain resistance gene sequence temporarily
    store_fasta_seqs = [] # List of header and sequence of resistance file
    ResKmerSet  = set()   # Resistance genes kmer set
    ResKmerDict = dict()  # Unique resistance genes kmer dictionary {kmer1:0, kmer2:0, ..

    ResFile = open("data/resistance_genes.fsa", "r")
    for line in ResFile:
        line = line.rstrip()
        if line.startswith(">"):
            if fasta:
                full_seq = "".join(fasta) # Store entire resistance gene sequence
                store_fasta_seqs.append(header)
                store_fasta_seqs.append(full_seq)

                # Generate kmers of the resistance gene,
                # add them to a kmer set, and
                # add them to a kmer dictionary with value 0
                for i in range(0, len(full_seq), 1):
```

```
                kmer = full_seq[i:i+kmer_length]
                if (i < len(full_seq) - kmer_length + 1):
                    ResKmerSet.add(kmer)
                if kmer not in ResKmerDict.keys():
                    ResKmerDict[kmer] = 0
            fasta = []
        header = line
    else:
        sequence = line
        fasta.append(sequence)

# Process last resistance gene
if fasta:
    full_seq = "".join(fasta) # Store entire resistance gene sequence
    store_fasta_seqs.append(header)
    store_fasta_seqs.append(full_seq)

    # Generate kmers of the resistance gene,
    # add them to a kmer set, and
    # add them to a kmer dictionary with value 0
    for i in range(0, len(full_seq), 1):
        kmer = full_seq[i:i+kmer_length]
        if (i < len(full_seq) - kmer_length + 1):
            ResKmerSet.add(kmer)
        if kmer not in ResKmerDict.keys():
            ResKmerDict[kmer] = 0

ResFile.close()

print("Finished processing resistance gene file.")

print("Processing read file 1...")
read_in_seq_reads("reads_1", kmer_length)

print("Processing read file 2...")
read_in_seq_reads("reads_2", kmer_length)

print("Filtering resistance genes that do not match the requirements (>95% coverage, >

skip_sequence = True
counter = 0
sequence = ""
final_result = list()
from matplotlib import pyplot as plt
import numpy as np
for i in range(len(store_fasta_seqs)):
    if (counter % 2 == 0):
        # arrived at gene id
```

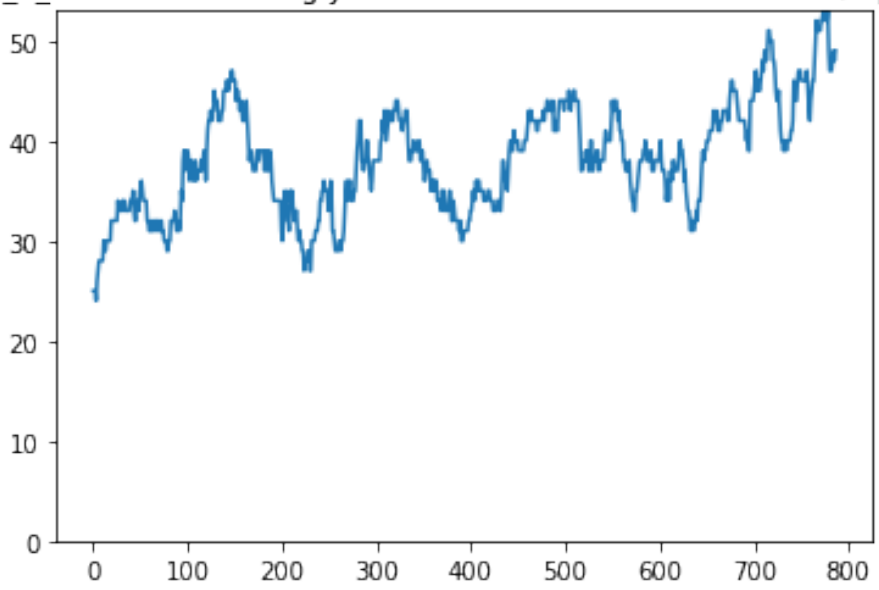
```
                skip_sequence = True
                header = store_fasta_seqs[i]
            else:
                # arrived at gene sequence
                temp = list()
                sequence = store_fasta_seqs[i]
                len_of_sequence = len(sequence)

                # Generate kmers of the sequence.
                # Add the depth to a kmer depth list, and check for coverage.
                # As soon as the coverage drops below 95%, ignore the sequence.
                # (= does not fulfill the requirements).
                for j in range(0, len(sequence), 1):
                    if (j < len(sequence) - kmer_length + 1):
                        read_kmer = sequence[j:j+kmer_length]
                        depth_of_kmer = ResKmerDict[read_kmer]
                        temp.append(depth_of_kmer)
                        temp_coverage = 1 - temp.count(0) / len_of_sequence
                        if (temp_coverage < 0.95): # check coverage, stop once below 95%
                            skip_sequence = False
                            break
                # Once we processed every kmer of the sequence and the coverage
                # is still >95%, we check for minimum depth (>=10)
                if skip_sequence and (min(temp) >= 10):
                    x = np.linspace(0, len(temp), len(temp))
                    plt.plot(x, temp)
                    plt.ylim(0, max(temp))
                    plt.title(header)
                    plt.show()
                    #print(temp)
                    final_result.append([header, temp_coverage, min(temp)])
            counter += 1

        # Sort results according to coverage and then minimum depth. Print results.
        final_result.sort(key=lambda x: (x[1], x[2]), reverse=True)
        print("Coverage [%]\tMinimum depth\tGene")
        for i in range(len(final_result)):
            print("{:.2f}\t\t{:d}\t\t{}".format(final_result[i][1] * 100, final_result[i][2],

        time_end = time.time()
        time_elapsed = time_end - time_start
        time_elapsed
```

```
Processing resistance genes...
Finished processing resistance gene file.
Processing read file 1...
Processing read file 2...
Filtering resistance genes that do not match the requirements (>95% coverage, >=10 depth)...
```

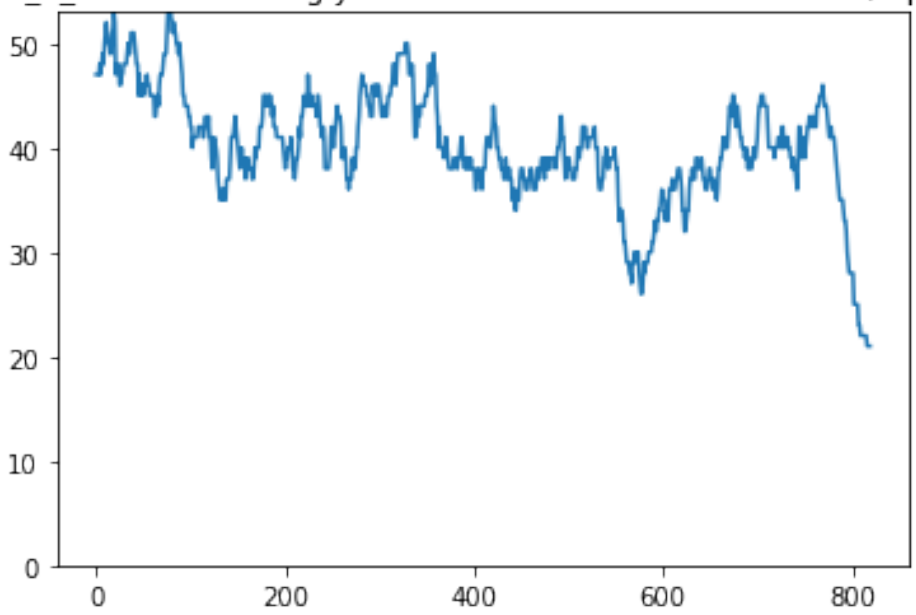
>aac(6')Ib-cr_1_DQ303918 Fluoroquinolone and aminoglycoside resistance:



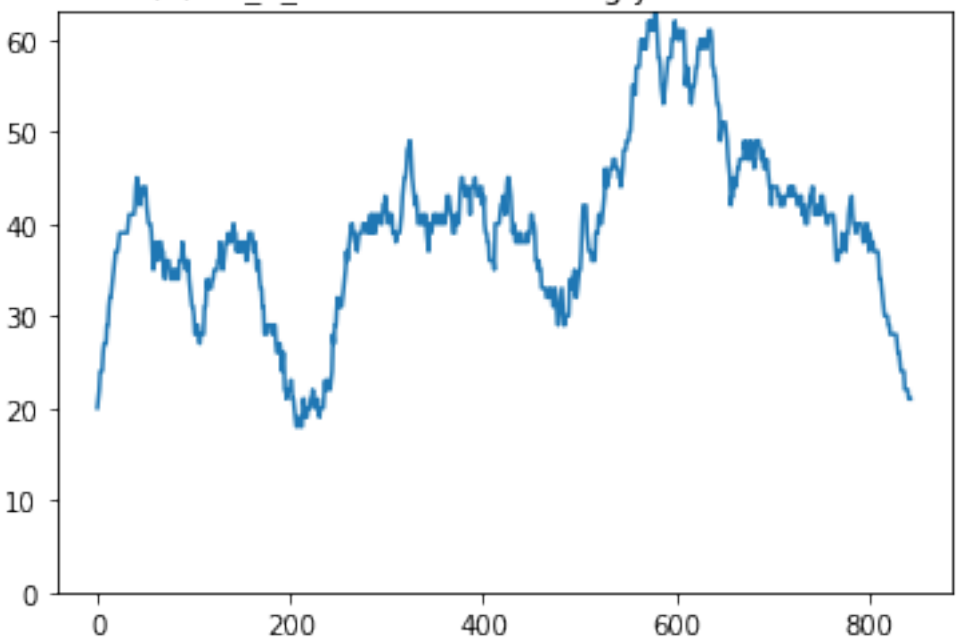
>strA_4_AF321551 Aminoglycoside resistance:Alternate name; aph(3'')-Ib



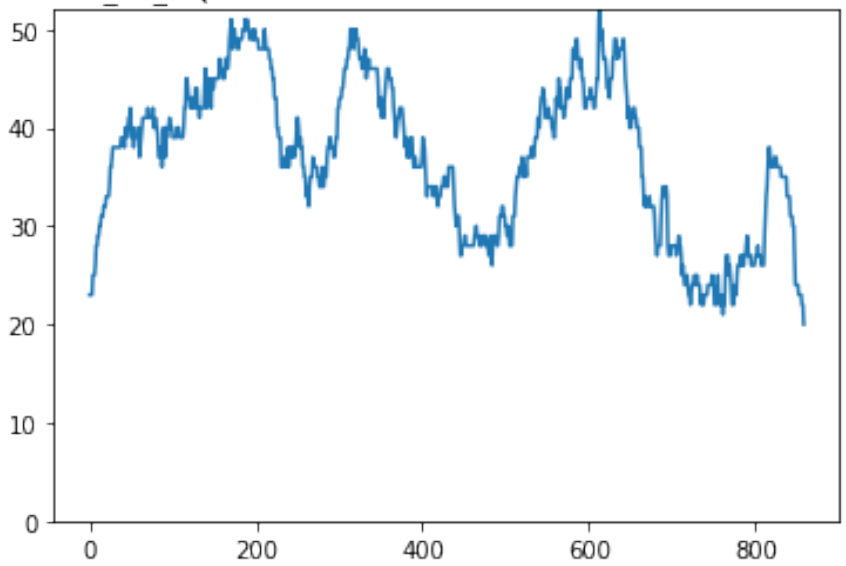
>strB_1_M96392 Aminoglycoside resistance:Alternate name; aph(6)-Id



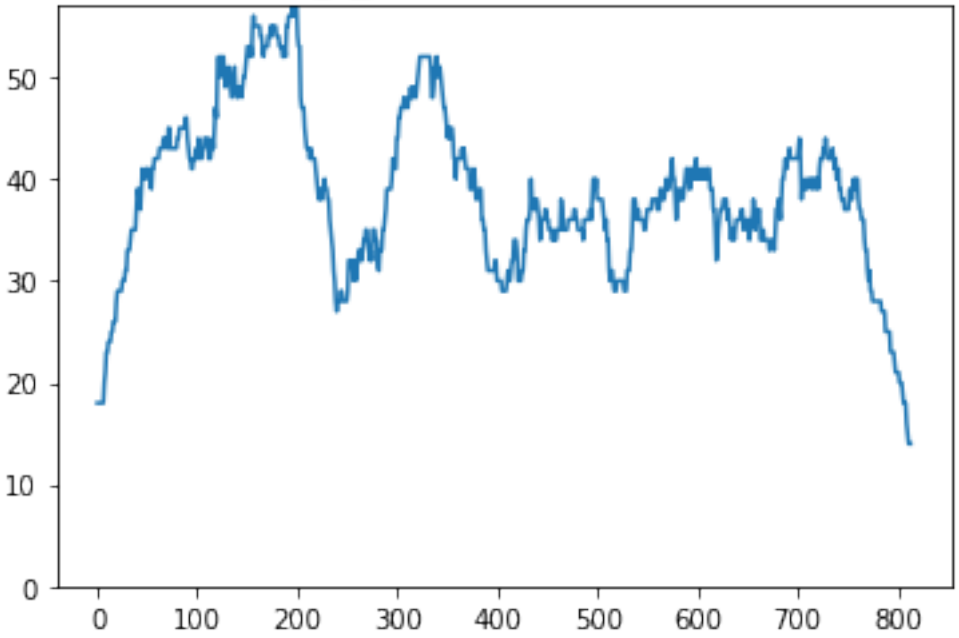
>aac(3)-IIa_1_CP023555.1 Aminoglycoside resistance:

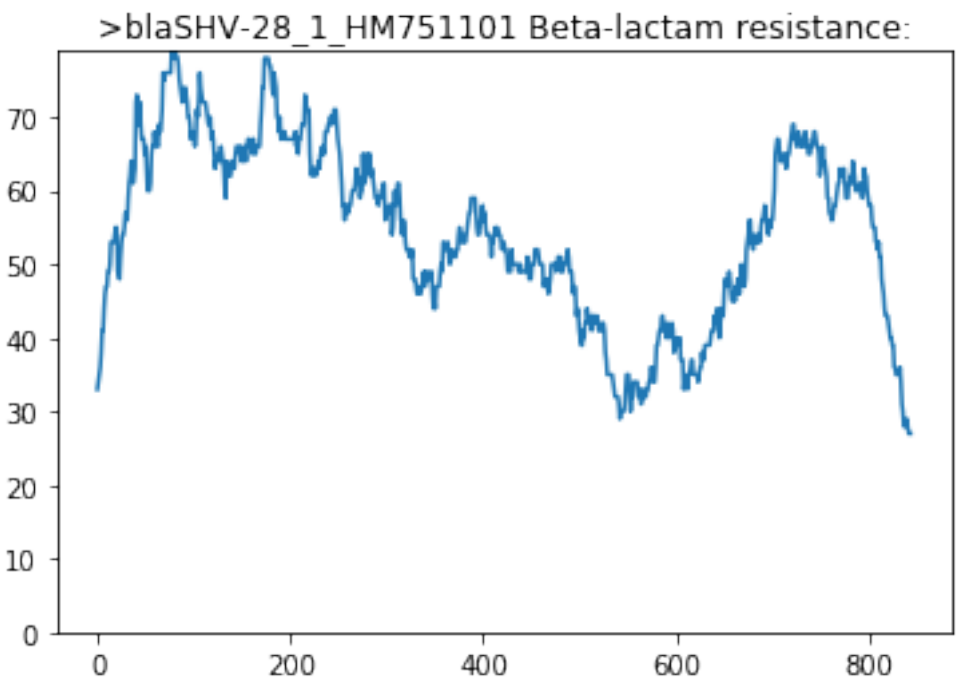


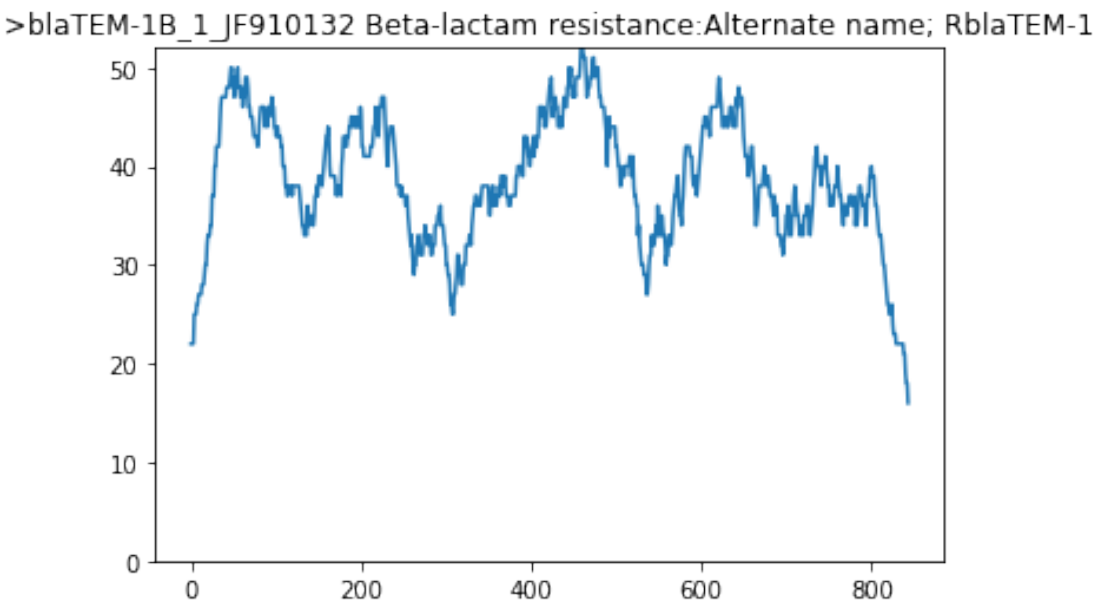
>blaCTX-M-15_23_DQ302097 Beta-lactam resistance:Alternate name; UOE-1

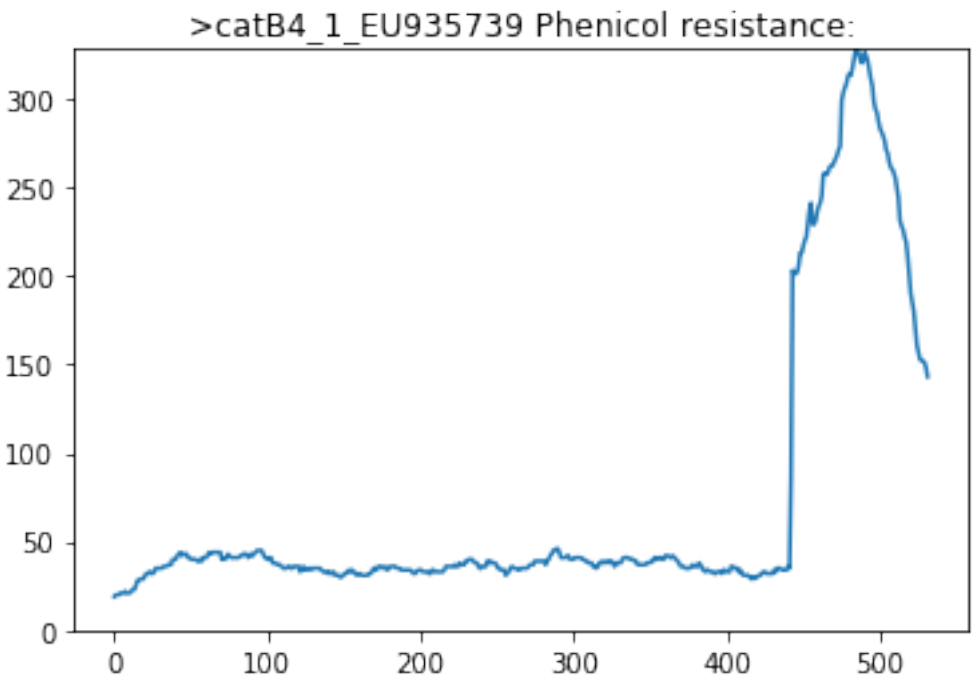


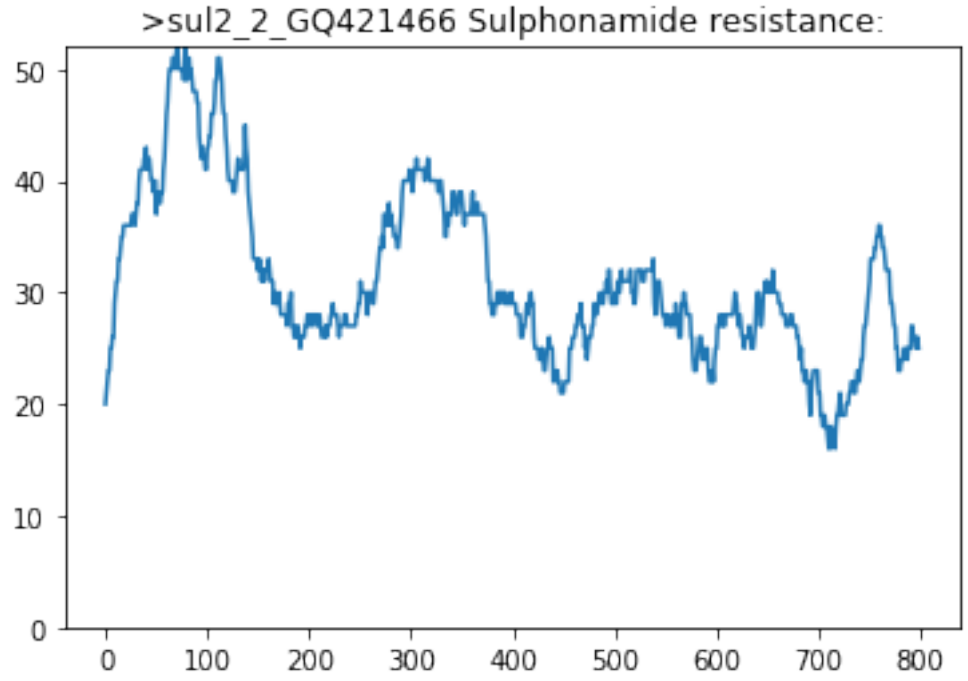
>blaOXA-1_1_J02967 Beta-lactam resistance:

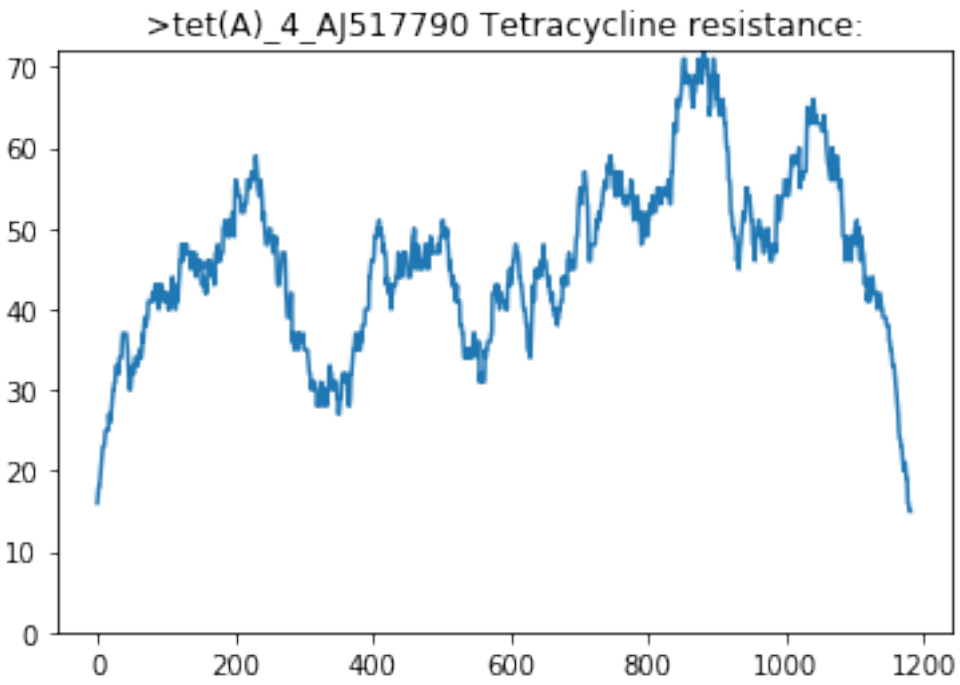



>blaSHV-28_1_HM751101 Beta-lactam resistance:
70
60
50
40
30
20
10
0
0
200
400
600
800


>blaTEM-1B_1_JF910132 Beta-lactam resistance:Alternate name; RblaTEM-1
50
40
30
20
10
0
0
200
400
600
800


>catB4_1_EU935739 Phenicol resistance:
300
250
200
150
100
50
0
0
100
200
300
400
500


>sul2_2_GQ421466 Sulphonamide resistance:
50
40
30
20
10
0
0
100
200
300
400
500
600
700
800



Coverage [%]	Minimum depth	Gene
100.00	27	>blaSHV-28_1_HM751101 Beta-lactam resistance:
100.00	24	>strA_4_AF321551 Aminoglycoside resistance:Alternate na
100.00	21	>strB_1_M96392 Aminoglycoside resistance:Alternate name
100.00	20	>blaCTX-M-15_23_DQ302097 Beta-lactam resistance:Alterna
100.00	19	>catB4_1_EU935739 Phenicol resistance:
100.00	18	>aac(3)-IIa_1_CP023555.1 Aminoglycoside resistance:
100.00	16	>blaTEM-1B_1_JF910132 Beta-lactam resistance:Alternate
100.00	16	>sul2_2_GQ421466 Sulphonamide resistance:
100.00	15	>tet(A)_4_AJ517790 Tetracycline resistance:
100.00	14	>aac(6')Ib-cr_1_DQ303918 Fluoroquinolone and aminoglyc
100.00	14	>blaOXA-1_1_J02967 Beta-lactam resistance:

Out[17]: 54.684276819229126