Here are the function written for the analysis for far, with an example:

import random  
random.seed(8675309)

Firstly, we’ll need a way to get create random DNA:

"""  
 generates random DNA.  
  
"""  
def random\_dna(n, nucleotides = ["A", "C", "G", "T"]):  
 return "".join([random.choice(nucleotides) for i in range(n)])  
  
ex\_dna = random\_dna(10)  
  
ex\_dna

'TCGCGAATGT'

Next, we will want to get the kmers in that DNA:

"""  
 gets the kmers from a DNA strand.  
  
"""  
def get\_kmers(seq, k):  
 kmers = []  
 end = len(seq)+1  
 for i in range(0, end-k):  
 kmers.append(seq[i:i+k])  
 return kmers  
  
ex\_kmers = get\_kmers(ex\_dna, 3)  
  
ex\_kmers

['TCG', 'CGC', 'GCG', 'CGA', 'GAA', 'AAT', 'ATG', 'TGT']

Next, we may want to count the kmers, so we will add it while we’re at it.

"""  
 counts the kmers in a DNA strand.  
  
"""  
def get\_kmer\_counts(kmers):  
 # this will store the kmers and their counts  
 kmer\_counts\_dict = {}  
 # go through each kmer  
 for kmer in kmers:  
 # if we don't have it in the dictionary  
 if kmer not in kmer\_counts\_dict.keys():  
 # give it a count of one  
 kmer\_counts\_dict[kmer] = 1  
 else:  
 # if we do, increase the number of times we've seen it by   
 kmer\_counts\_dict[kmer] += 1  
 return kmer\_counts\_dict  
  
ex\_kmer\_counts = get\_kmer\_counts(ex\_kmers)  
  
ex\_kmer\_counts

{'TCG': 1,  
 'CGC': 1,  
 'GCG': 1,  
 'CGA': 1,  
 'GAA': 1,  
 'AAT': 1,  
 'ATG': 1,  
 'TGT': 1}

If we’re going to hide progressively more divergent sequences in real DNA, we will need a function to do so withouth changing the overall sequence length. We can do so with the following function:

"""  
 takes a sequence and hides it in another sequence, preserving the overall length (writing over sequences)  
 to do so.  
  
"""  
def insert\_sequence(seq, dna, insert\_index = None):  
  
 # if we don't specify were, make it random  
 if not insert\_index:  
 # we need to pick a random spot to insert it that doesn't go over the edge  
 insert\_location = random.choice(range(0, len(dna) - (len(seq))+1))  
 else:  
 insert\_location = insert\_index  
  
 # the left/right boundaries are 0 and as close to the end as   
 # we can get without going over  
 left\_boundary = 0  
 right\_boundary = len(dna) - len(seq)  
  
  
 # if it's at the start or the end  
 if insert\_index == left\_boundary:  
 # insert the sequence at the start  
 left = seq  
 right = dna[len(seq):len(dna)]  
 result = "".join([left, right])  
  
  
 elif insert\_index == right\_boundary:  
 # insert the sequence at the end  
 left = dna[0:right\_boundary]  
 right = seq  
 result = "".join([left, right])  
   
  
 # if it's in the middle somewhere  
 else:   
 # get the sequence before that insertion point   
 before\_insertion = dna[0:insert\_location]  
 # get the sequence after  
 after\_insertion = dna[insert\_location+len(seq):len(dna)]  
 # join them into a strand  
 result = "".join([before\_insertion, seq, after\_insertion])  
  
 # make sure we didn't madify the length  
 assert len(result) == len(dna)  
  
  
 return result

This is a little more involved and I don’t want to just check it by eye, so we will set up some tests and edge cases:

# a starting sequence  
before = "GCAAGAACTATATACACACTTAGCAAGTTACGTTTCTTAT"  
# something to kide in it  
to\_hide = "(THADRYAN)"  
  
# left-most position  
after = insert\_sequence(to\_hide, before, 0)  
print("before: ", before)  
print("after: ", after)  
  
# right-most position  
after = insert\_sequence(to\_hide, before, 30)  
print("before: ", before)  
print("after: ", after)  
  
# random position  
after = insert\_sequence(to\_hide, before)  
print("before: ", before)  
print("after: ", after)  
  
# test the edge-cases  
edge\_case1 = "(THADRYAN)TATACACACTTAGCAAGTTACGTTTCTTAT"  
edge\_case2 = "GCAAGAACTATATACACACTTAGCAAGTTA(THADRYAN)"  
  
# run it a bunch to see if they come up and are handled ok  
test\_cases = [insert\_sequence(to\_hide, before) for i in range(10000)]  
  
print(edge\_case1 in test\_cases)  
print(edge\_case2 in test\_cases)

before: GCAAGAACTATATACACACTTAGCAAGTTACGTTTCTTAT  
after: (THADRYAN)TATACACACTTAGCAAGTTACGTTTCTTAT  
before: GCAAGAACTATATACACACTTAGCAAGTTACGTTTCTTAT  
after: GCAAGAACTATATACACACTTAGCAAGTTA(THADRYAN)  
before: GCAAGAACTATATACACACTTAGCAAGTTACGTTTCTTAT  
after: GCAAGAACTATATACACACTTAGCAAGTTA(THADRYAN)  
True  
True