```
#!/usr/bin/env python
# coding: utf-8

# In[1]:


#!/usr/bin/python3
import sys
import numpy as np
import pandas as pd
import csv
import urllib.request
import os
from Bio import Entrez
from Bio.SeqRecord import SeqRecord
from Bio import SeqIO
from itertools import islice
from os import listdir
from os.path import isfile, join
from Bio.Align.Applications import ClustalwCommandline
import matplotlib.pyplot as plt
get_ipython().run_line_magic('matplotlib', 'inline')
from Bio import AlignIO
from Bio.Align import AlignInfo
from multiprocessing import Pool
import multiprocessing as mp
from collections import defaultdict
from collections import Counter
import seaborn as sns
import matplotlib.gridspec as gridspec
import matplotlib.patches as patches
import random
from mpl_toolkits.axes_grid1.anchored_artists import AnchoredDrawingArea
from matplotlib.patches import Ellipse
import matplotlib.transforms as transforms
from numpy import exp
import scipy as sp
import math
from matplotlib.pyplot import figure


# In[2]:


Burk_inFile = "../INPUT/WspH_clusterblast_output.txt"
#Burk_lastgene = "AKM42514"
Burk_lastgene="AMO78240"
Pseu_inFile = "../INPUT/WspR_clusterblast_output.txt"
#Pseu_lastgene = "AIO56285"
Pseu_lastgene="AGZ30312"
inFolder = "../gene_genbank_files"
geneAccession = ['WspA', 'WspB', 'WspC', 'WspD', 'WspE', 'WspF']
BCT_Assembly = "../BCT_Assemblies/"
run_summary = []
```

```
#Rscript directory and script location
outTitle = "Wsp_Pseu_Burk_Aug2020"
R_dir = "/home/kim_lab_kvm/Documents/Species_Phylogeny/bacterial/data/" +
outTitle
Rscript =
"/home/kim_lab_kvm/Documents/Species_Phylogeny/Phyton_handoff_scripts/"


# In[3]:


##what is your email address?
Entrez.email = "kesslerc1@duq.edu"
api_key = "a4a8839a243498db2c50029dcd8986c7ed08"


# In[4]:


##color library
sel_annotations = [" " ,"HAMP", "Extracellular Sensory","Dock",
"Signaling", "CH3", "Methyltransferase", "HATPase", "HPT",
"Dimerization", "REC", "GGDEF(WspR)"]
sel_colors = ["#f7f7f5", "#e74c3c", "#c39bd3",  "#5499c7",  "#48c9b0",
"#c3edb7",  "#d4ac0d",  "#eb984e",  "#d2b4de",  "#f5b7b1",  "#a9cce3",
"#a2d9ce",  "#f9e79f", "#edbb99", "#abb2b9"]


# In[ ]:




# In[5]:


if not os.path.exists('../GeneFasta'):
    os.makedirs('../GeneFasta')
if not os.path.exists('../GeneFasta/WspR'):
    os.makedirs('../GeneFasta/WspR')
if not os.path.exists('../GeneFasta/WspH'):
    os.makedirs('../GeneFasta/WspH')
if not os.path.exists("../gene_genbank_files"):
    os.makedirs("../gene_genbank_files")
if not os.path.exists(BCT_Assembly):
    os.makedirs(BCT_Assembly)
if not os.path.exists(R_dir):
    os.makedirs(R_dir)
if not os.path.exists(R_dir + "/assembly_stats/"):
    os.makedirs(R_dir + "/assembly_stats/")
if not os.path.exists('../FIMO'):
    os.makedirs('../FIMO')
```

```
if not os.path.exists('../RAxML'):
    os.makedirs('../RAxML')
if not os.path.exists('../ShannonEntropyData_H_vs_R/'):
    os.makedirs('../ShannonEntropyData_H_vs_R/')
if not os.path.exists("../GeneShannon/WspR/"):
    os.makedirs("../GeneShannon/WspR/")
if not os.path.exists("../GeneShannon/WspH/"):
    os.makedirs("../GeneShannon/WspH/")
if not os.path.exists("../ShannonEntropyData_H_and_R/"):
    os.makedirs("../ShannonEntropyData_H_and_R/")



# In[6]:


## Interpret MGB Data
iterin = (Burk_inFile, Pseu_inFile)
iterlastgene = (Burk_lastgene, Pseu_lastgene)
itersave = ("Burk", "Pseu")
hits_to_skip = []

for zz in range(len(iterin)):
    operons = []
    with open(iterin[zz]) as f:
        copy = False
        for line in f:
            if '>>' in line:
                name = (''.join(islice(f, 2)))
                name = name.split(' ')
                name = name[1]
                name = name.replace('\n','')

            if line.strip() == "Table of Blast hits (query gene, subject
gene, %identity, blast score, %coverage, e-value):":
                copy = True
                continue
                #write to file "operon - Letter"
            elif line.strip() == ">>":
                copy = False
                continue
            elif copy:
                gene = line.rstrip().split()
                gene.insert(0, name)
                operons.append(gene)

    opData = pd.DataFrame(operons, columns=["Organism","Query",
"Subject", "% Identity", "Blast Score", "% Coverage", "e-value"])
    opData = opData[opData.Subject.notnull()]

    value = len(opData)
    list = range(value)
    opData.index = list
```

```
    count = [i for i, s in enumerate(opData.Subject) if iterlastgene[zz]
in s]
    opData = opData.iloc[:count[0]+1, :]


    data = np.asarray(opData)
    gene_accession = (data[0:,2])



    download_gbks = []
    gbks_filename = []


    Entrez.email = "cok25@pitt.edu"
    existingFiles = set(os.listdir("../gene_genbank_files/"))

    L = len(gene_accession)

    for line in set(gene_accession):
        AA_filename = (line + ".aa.gbk")
        if AA_filename not in existingFiles:
            records = Entrez.read(Entrez.esearch(db='protein', term=line,
api_key=api_key))
            records_list = records['IdList']
            result =
Entrez.efetch(db="protein",id=records_list,rettype="gbwithparts",retmode=
'text', api_key=api_key)
            sequences = SeqIO.parse(result,'gb')
            SeqIO.write(sequences, "../gene_genbank_files/" +
AA_filename, "gb")

    for line in set(gene_accession):
        AA_filename = (line + ".aa.gbk")
        filesize = os.path.getsize("../gene_genbank_files/" +
AA_filename)
        if filesize == 0:
            hits_to_skip.append(line)
            print("The file is empty: " + str(AA_filename), "this value
will be removed")

    if "Burk" in itersave:
        labeled_opData = []
        for index, row in opData.iterrows():
            row["Query"] = row["Query"].replace("ABK10523.1","WspA")
            row["Query"] = row["Query"].replace("ABK10524.1","WspB")
            row["Query"] = row["Query"].replace("ABK10525.1","WspC")
            row["Query"] =
row["Query"].replace("ABK10526.1","hypothetical")
            row["Query"] = row["Query"].replace("ABK10527.1","WspD")
            row["Query"] = row["Query"].replace("ABK10528.1","WspE")
            row["Query"] = row["Query"].replace("ABK10529.1","WspF")
            row["Query"] = row["Query"].replace("ABK10522.1","WspR_H")
```

```
    if "Pseu" in itersave:
        labeled_opData = []
        for index, row in opData.iterrows():
            row["Query"] = row["Query"].replace("ABA72795.1","WspA")
            row["Query"] = row["Query"].replace("ABA72796.1","WspB")
            row["Query"] = row["Query"].replace("ABA72797.1","WspC")
            row["Query"] = row["Query"].replace("ABA72798.1","WspD")
            row["Query"] = row["Query"].replace("ABA72799.1","WspE")
            row["Query"] = row["Query"].replace("ABA72800.1","WspF")
            row["Query"] = row["Query"].replace("ABA72801.1","WspR_H")

    savename = "../"+itersave[zz]+"_operons.txt"
    np.savetxt(savename, opData.values, fmt='%s', delimiter=",")
print()
print("Output files sucessfully generated")


# In[7]:


## Concatonate MGB Runs and Rooting Data
Burki = "../Burk_operons.txt"
Pseui = "../Pseu_operons.txt"
Rooti = "../INPUT/Caulobacter_for_root.txt"
OutCat = "../operons_rooting.txt"

get_ipython().system('cat $Burki $Pseui $Rooti > $OutCat')


# In[8]:


##Export genome accession number data
operons = []
with open("../operons_rooting.txt") as f:
    for line in f:
        operons.append(line.rstrip().split(','))
operons = np.asarray(operons[1:])

organismAccessionsRaw = (np.unique(operons[0:,0]))

organismAccessions = []
for a in organismAccessionsRaw:
    b = a.split('_')
    organismAccessions.append(b[0])

np.savetxt(r'../organismAccessions.txt', organismAccessions, fmt='%s',
delimiter=",")


# In[9]:


## Convert data to Pandas dataframe for indexing
```

```
data_frame=[]
org_accession_dup_search_1 = []
with open("../operons_rooting.txt") as f:
    for line in f:
        a = (line.replace("\n","")).rstrip().rsplit(",")
        data_frame.append(a)
header = "Organism","Query", "Wsp_locus", "% Identity", "Blast Score", "%
Coverage", "e-value"
MGB_Summarydf_1 = pd.DataFrame(data=data_frame, columns=header)
print(len(MGB_Summarydf_1))
for index, row in MGB_Summarydf_1.iterrows():
    if row["Organism"] not in org_accession_dup_search_1:
        org_accession_dup_search_1.append(row["Organism"])

org_accession_dup_search_2 = []
for line in org_accession_dup_search_1:
    a = line.rstrip().rsplit("_")
    org_accession_dup_search_2.append(a)
org_accession_dup_search_df =
pd.DataFrame(data=org_accession_dup_search_2,columns=["ACC", "VERSION"])
org_accession_dup_search_df_sorted =
org_accession_dup_search_df.sort_values(by='VERSION', ascending=False)
b = len(org_accession_dup_search_df_sorted)
org_accession_dup_search_df_sorted.drop_duplicates(keep="first",inplace=T
rue, subset="ACC")
c = len(org_accession_dup_search_df_sorted)
d = b-c
print(d, "genomes in this search have been dropped because we identifed
hits in more than one genome version. The most recent genome hits have
been retained...")

updated_genomes = []
new_dataset = []
for index, row in org_accession_dup_search_df_sorted.iterrows():
    a = row["ACC"]
    b = row["VERSION"]
    c = a + "_" + b
    updated_genomes.append(c)

updated_genomes = set(updated_genomes)
for index, row in MGB_Summarydf_1.iterrows():
    if row["Organism"] in updated_genomes:
        new_row = row["Organism"],row["Query"],row["Wsp_locus"],row["%
Identity"],row["Blast Score"],row["% Coverage"],row["e-value"]
        new_dataset.append(new_row)
MGB_Summarydf = pd.DataFrame(data=new_dataset, columns=header)
MGB_Summarydf.drop_duplicates(keep="first",inplace=True,
subset="Wsp_locus")
print(len(MGB_Summarydf))


# In[10]:
```

```
## Test currated dataset for incomplete wsp operons/ gene clusters
operon_dict = {}
key_set = []
for index, row in MGB_Summarydf.iterrows():
    key = row["Organism"]
    key_set.append(key)
    value = str(row["Query"]+ "," + row["Wsp_locus"])
    if key in operon_dict:
        current_value = operon_dict[key]
        new_value = str(current_value + ";" + value)
        operon_dict.update({key:new_value})
    if key not in operon_dict:
        operon_dict.update({key:value})
key_set = set(key_set)
organisms_to_omit =[]
locus_array = ["WspA", "WspB", "WspC", "WspD", "WspE", "WspF", "WspR_H"]
print("Do any hits have an IRREGULAR locus set...(no news is good news)")

for i in key_set:
    locus_count = [operon_dict[i].count(element) for element in
locus_array]
    for k in locus_count:
        if k > 1:
            print(i,"contains multiple wsp loci: operon has been removed
from dataset for seperate evalutation")
            organisms_to_omit.append(i)

    if "WspA" and "WspB" and "WspC" and "WspD" and "WspE" and "WspF" and
"WspR_H" not in operon_dict[i]:
        if "CP001340" not in i:
            print(i, "is an INCOMPLETE operon")
            organisms_to_omit.append(i)
print()
print("Do any hits have an INCOMPLETE locus set...(no news is good
news)")
hits_to_skip = set(hits_to_skip)
for i in key_set:
    for j in hits_to_skip:
        if j in operon_dict[i]:
            organisms_to_omit.append(i)
            print("The dataset for", i, "is incomplete, the file", j,
"could not be downloaded")


# In[11]:


## Retreive exisiting data
existingFiles = (os.listdir(BCT_Assembly))
existingFiles = set(existingFiles)
assembly_metadata = []
no_assembly_data = []
completed_analysis = []
```

```
print("RuntimeError is expected if no record exists")
print("HTTPError is expected if no API key is provided or if there is a
server error for the file which we recommend to skip")
print("Processing...")

if 'efetch_assemblies.txt' in existingFiles:
    with open(BCT_Assembly + "efetch_assemblies.txt") as f:
        for line in f:
            if "None" not in line:
                z = line.rstrip().rsplit("##")
                y = z[0] + "##" + z[1] + "##" + z[2] + "\n"
                y = y.replace(" ","")
                assembly_metadata.append(y)
                completed_analysis.append(z[0])

completed_analysis = set(completed_analysis)


# In[12]:


## Fetch URL data for genome assemblies
def Efetch_Url(term):
    successes = []
    if term not in completed_analysis:
        term = term.replace("\n","")
        try:
            handle = Entrez.esearch(db="assembly", term=term,
api_key=api_key, source="refseq")
            record = Entrez.read(handle)
            esummary_handle = Entrez.esummary(db="assembly",
id=record['IdList'], report="full",api_key=api_key)
            esummary_record = Entrez.read(esummary_handle)
            esummary_record_list =
str(esummary_record).rstrip().rsplit(",")
            for line in esummary_record_list:
                if "FtpPath_Stats_rpt" in line:
                    a = line.rstrip().rsplit("':")
                    if len(a)>=2:
                        b = a[1].replace("'","")
                        d = b.rstrip().rsplit("/")
                        e = d[-1]
                        c = term + "##" + e + "##" + b + "\n"
                        c = c.replace(" ","")
                        successes.append(c)
            return successes

        except:
            no_assembly_data.append(term)
            print(term, "ERROR TYPE:", sys.exc_info()[0])
            pass

if __name__ == '__main__':
    pool = mp.Pool(processes=3)
```

```
    total_successes = pool.map(Efetch_Url, organismAccessions)
    pool.terminate()


# In[13]:


multisearch_list = []
for item in total_successes:
    if "None" not in str(item):
        item = str(item).replace("']","")
        item = item.replace("['","")
        item = item.replace("\\n","\n")
        multisearch_list.append(item)

TOTAL_SUCCESSES = assembly_metadata + multisearch_list
print(len(TOTAL_SUCCESSES))


# In[14]:


with open(BCT_Assembly + "efetch_assemblies.txt", 'w') as f:
    for line in TOTAL_SUCCESSES:
        f.write(line)


# In[15]:


#prep for assembly download
existingFiles = (os.listdir(BCT_Assembly))
URL_LIST =[]
GCA_URLs = []
if 'efetch_assemblies.txt' in existingFiles:
    with open(BCT_Assembly + 'efetch_assemblies.txt') as f:
        for line in f:
            if "None" not in line:
                z = line.rstrip().rsplit("##")
                term = z[0]
                file_name = z[1]
                URL = z[2]
                if "GCA" not in URL:
                    a = term, file_name, URL
                    URL_LIST.append(a)
                if "GCA" in URL:
                    a = term
                    GCA_URLs.append(a)

URL_LIST = np.asarray(URL_LIST)
print(len(GCA_URLs), "are not compatible with the R scripts for genome
assembly")
```

```
run_summary.append(str((str(len(GCA_URLs))) +  " of the " +
str(len(assembly_metadata)) + " are not compatible with the R scripts for
genome assembly"))


# In[16]:


#multithread assembly download
existingFiles = (os.listdir(BCT_Assembly))
existingFiles = set(existingFiles)

def MultiprocessWget(tupple_of_3):
    URL = tupple_of_3[2]
    file_name = tupple_of_3[1]
    out_file = BCT_Assembly + file_name
    if file_name not in existingFiles:
        get_ipython().system('wget -O $out_file $URL')

if __name__ == '__main__':
    pool = mp.Pool(processes=16)
    pool.map(MultiprocessWget, URL_LIST)
    pool.terminate()


# In[17]:


## Export new information
assembly_dic={}
with open(BCT_Assembly + 'efetch_assemblies.txt') as f:
    for line in f:
        z = line.rstrip().rsplit("##")
        key = z[0]
        value = z[1]
        if "GCF" in value:
            assembly_dic.update({key:value})

new_dataframe = []
cannot_continue = []
for index, row in MGB_Summarydf.iterrows():
    search_name = (row["Organism"]).rstrip().rsplit("_")
    if search_name[0] in assembly_dic:
        return_assem = assembly_dic[search_name[0]]
        new_row = row["Organism"],row["Query"],row["Wsp_locus"],row["%
Identity"],row["Blast Score"],row["% Coverage"],row["e-
value"],return_assem
        new_dataframe.append(new_row)
    if search_name[0] not in assembly_dic:
        cannot_continue.append(search_name[0])

header = "Organism","Query", "Wsp_locus", "% Identity", "Blast Score", "%
Coverage", "e-value", "Assembly"
MGB_Summarydf_assembly = pd.DataFrame(data=new_dataframe, columns=header)
```

```
print(str(len(cannot_continue))+" hits do not have a refseq assembly and
must be omited")


# In[18]:


## Prep assemblies for R and see if any have multiple operons
assemblies_for_R = []
for index, row in MGB_Summarydf_assembly.iterrows():
    if row["Organism"] not in set(organisms_to_omit):
        assemblies_for_R.append(row["Assembly"])

set_assemblies_for_R = set(assemblies_for_R)

#compile list of dups
multi_copy_wsps = []
access_set = []
category = Counter(assemblies_for_R)
for key, value in category.items():
    if value > 7:
        for index, row in MGB_Summarydf_assembly.iterrows():
            if key in row["Assembly"]:
                export = row["Organism"], key
                multi_copy_wsps.append(export)
                access_set.append(key)
#clean list
access_set = set(access_set)
def removeDuplicates(lst):
    return [t for t in (set(tuple(i) for i in lst))]
multi_copy_wsps_clean =(removeDuplicates(multi_copy_wsps))

#choose most recent if a duplicate
mini_list =[]
mini_set = set()
for refseq in access_set:
    for typle in multi_copy_wsps_clean:
        if refseq in typle:
            h = typle[0].rstrip().rsplit("_")
            if h[0] in mini_set:
                print(h[0], "is a duplicate, choosing the most recent")
            g = h[0], int(h[1]), refseq
            mini_list.append(g)
            if h[0] not in mini_set:
                mini_set.add(h[0])


mini_listdf = pd.DataFrame(data=mini_list,columns=["acc", "version",
"refseq"])
mini_listdf_sorted = mini_listdf.sort_values(by='version',
ascending=False)
mini_listdf_sorted.drop_duplicates(keep="first",inplace=True,
subset="acc")
out_list = []
```

```
for index, row in mini_listdf_sorted.iterrows():
    c = row["acc"]+"_"+str(row["version"]), row["refseq"]
    out_list.append(c)
out_list = np.asarray(out_list)


# #execute handshake with Rscipts for species alignemnt - modify these
scripts to reflect your working environment
# #this step will take a while!
# existingFiles = os.listdir("../")
# existingAssemblies = os.listdir(BCT_Assembly)
#
# if "species_120.fasta" not in existingFiles:
#     stats_dir = R_dir + "/assembly_stats/"
#     !rm -fr $stats_dir
#     !mkdir $stats_dir
#     for ref_file in assemblies_for_R:
#         current_file = BCT_Assembly + ref_file
#         !cp $current_file $stats_dir
#
#     script1 = Rscript + "getGenomes_python.R " + outTitle
#     script2 = Rscript + "getHousekeepingFastaFromHMM_python.R " +
outTitle
#     script3 = Rscript + "getHousekeepingTreeFromFastaFromHMM_python.R "
+ outTitle
#
#     !Rscript $script1
#     !Rscript $script2
#     !Rscript $script3
#
# print("R script complete")

# In[19]:


existingFiles = os.listdir("../")
if "species_120.fasta" not in existingFiles:
    species_from =
R_dir+"/hmmsearch_bac120_output_from_translated_cds_concatenated/bac120.f
aa.aln.trimal.renamed"
    species_to = "../species_120.fasta"
    get_ipython().system('cp $species_from $species_to')
bianary_data = []

seqs = AlignIO.read("../species_120.fasta",'fasta')
seq_length = range((len(seqs)))
for entry in seq_length:
        a = seqs[entry].id
        b = str(seqs[entry].seq)
        c = a, b
        bianary_data.append(c)
bianary_datadf = pd.DataFrame(data=bianary_data,columns=["ID", "SEQ"])
d = len(bianary_datadf)
bianary_datadf.drop_duplicates(keep="first",inplace=True, subset="SEQ")
```

```
e = len(bianary_datadf)
f = d-e
print(f, "duplicate species sequences removed")
prefasta = []
filtered_assemblies = []
for index, row in bianary_datadf.iterrows():
    g = ">" + row["ID"] + "\n"
    h = row["SEQ"] + "\n"
    i = row["ID"] + "_assembly_stats.txt"
    filtered_assemblies.append(i)
    prefasta.append(g)
    prefasta.append(h)
with open("../species_120_reduced.fasta", 'w') as f:
    f.writelines(prefasta)


# In[20]:


bianary_species = []
trimmed_assebmly_dataframe = []
for index, row_species in bianary_datadf.iterrows():
    bianary_species_name = row_species["ID"]+"_assembly_stats.txt"
    bianary_species.append(bianary_species_name)
bianary_species = set(bianary_species)
organisms_to_omit = set(organisms_to_omit)

## Transfer independent gene data into Wsp specific files
filtered_assemblies = set(filtered_assemblies)

presentFiles = [f for f in listdir('../GeneFasta') if
isfile(join('../GeneFasta', f))]
L = []
for i in geneAccession:
    if (str(i) + ".fasta") in presentFiles:
        L.append(str(i) + ".fasta")
for i in geneAccession:
    if (str(i) + ".fasta") in presentFiles:
        os.remove('../GeneFasta/' + str(i) + ".fasta")
        print(str(i) + ".fasta replaced")
HR_file = "../GeneFasta/WspR_H.fasta"
Hyp_file = "../GeneFasta/hypothetical.fasta"
get_ipython().system('rm -fr $HR_file $Hyp_file')
print("WspR_H.fasta replaced")
print("hypothetical.fasta replaced")

for index, row_proteins in MGB_Summarydf_assembly.iterrows():
    if row_proteins["Assembly"] in bianary_species:
        if row_proteins["Organism"] not in organisms_to_omit:
            AA_filename = (inFolder + "/" + row_proteins["Wsp_locus"] +
".aa.gbk")
            with open('../GeneFasta/' + str(row_proteins["Query"]) +
".fasta", 'a', encoding='utf-8') as f:
                for record in SeqIO.parse(AA_filename, "gb"):
```

```
                    f.write(">" + row_proteins["Query"] + "_" +
row_proteins["Wsp_locus"] + "_" + row_proteins["Organism"] + "\n" +
str(record.seq) + "\n")


# In[21]:


## Search H and R for DGC domain
ggdef_file = "../INPUT/ggeef.meme"
WspHR = "../GeneFasta/WspR_H.fasta"
get_ipython().system('fimo $ggdef_file $WspHR')

fimo_out = "./fimo_out"
fimo_final = "../FIMO/"
get_ipython().system('cp -fr $fimo_out $fimo_final')
get_ipython().system('rm -fr $fimo_out')


# In[22]:


new_dataset = []
ggdef_positives = []
with open("../FIMO/fimo_out/fimo.tsv") as f:
    for line in f:
        a = line.rstrip().rsplit("\t")
        if "motif_id" not in a:
            if len(a) > 1:
                b = a[2]
                c = b.rstrip().rsplit("_")
                ggdef_positives.append(c[2])

ggdef_positives_set = set(ggdef_positives)
for index, row in MGB_Summarydf_assembly.iterrows():
    if row["Assembly"] in bianary_species:
        if row["Organism"] not in organisms_to_omit:
            if "WspR_H" not in row["Query"]:
                new_row =
row["Organism"],row["Query"],row["Wsp_locus"],row["%
Identity"],row["Blast Score"],row["% Coverage"],row["e-
value"],row["Assembly"]
                new_dataset.append(new_row)
            if "WspR_H" in row["Query"]:
                if row["Wsp_locus"] in ggdef_positives_set:
                    new_row =
row["Organism"],"WspR",row["Wsp_locus"],row["% Identity"],row["Blast
Score"],row["% Coverage"],row["e-value"],row["Assembly"]
                    new_dataset.append(new_row)
            if "WspR_H" in row["Query"]:
                if row["Wsp_locus"] not in ggdef_positives_set:
                    new_row =
row["Organism"],"WspH",row["Wsp_locus"],row["% Identity"],row["Blast
Score"],row["% Coverage"],row["e-value"],row["Assembly"]
```

```
                    new_dataset.append(new_row)
header = "Organism","Query", "Wsp_locus", "% Identity", "Blast Score", "%
Coverage", "e-value", "Assembly"
MGB_assembly_DGC_pd = pd.DataFrame(data=new_dataset, columns=header)


# In[23]:


operon_dict_refined = {}
key_set = []
for index, row in MGB_assembly_DGC_pd.iterrows():
    key = row["Organism"]
    key_set.append(key)
    value = str(row["Query"]+ "," + row["Wsp_locus"])
    if key in operon_dict_refined:
        current_value = operon_dict_refined[key]
        new_value = str(current_value + ";" + value)
        operon_dict_refined.update({key:new_value})
    if key not in operon_dict_refined:
        operon_dict_refined.update({key:value})


# In[24]:


## align gene_fasta
## stich alignments
## build trees
## evaluate conservation


# In[25]:


existingFiles = os.listdir("../GeneFasta/")
for i in existingFiles:
    if ".fasta" in i:
        align_in = "../GeneFasta/"+i
        j = i.replace(".fasta", ".aln")
        k = j + ".trimal"
        if k not in existingFiles:
            align_out = "../GeneFasta/"+j
            trimal_out = "../GeneFasta/"+k
            get_ipython().system('mafft --maxiterate 1000 --thread 16 --
globalpair $align_in > $align_out')
            get_ipython().system('trimal -in $align_out -out $trimal_out
-gapthreshold 0.3 -fasta')


# In[26]:


new_dataset = []
```

```
rename_dict = {}
for index, row in MGB_assembly_DGC_pd.iterrows():
    current_file = BCT_Assembly + row["Assembly"]
    with open (current_file) as f:
        for line in f:
            if "# Organism name:  " in line:
                EZ_TAX = ((((line.replace("# Organism name:  ","")).
                           replace("\n","").replace(" (b-
proteobacteria)","")).
                          replace(" (g-proteobacteria)","")).replace("
(a-proteobacteria)",""))
                key = row["Organism"]
                value = (row["Organism"].replace("_",".")) + "_" +EZ_TAX
                rename_dict.update({key:value})
                new_row = (row["Organism"],row["Query"],
                           row["Wsp_locus"],row["% Identity"],
                           row["Blast Score"],row["% Coverage"],
                           row["e-value"],row["Assembly"],EZ_TAX)
                new_dataset.append(new_row)
header = "Organism","Query", "Wsp_locus", "% Identity", "Blast Score", "%
Coverage", "e-value", "Assembly","EZ_Tax"
MGB_assembly_DGC_ezTax_pd = pd.DataFrame(data=new_dataset,
columns=header)


# In[27]:


#Read in the individual alignment files and create the continuous
sequence used for the gene tree

existingFiles = os.listdir("../GeneFasta/")
geneAccession = ['WspA', 'WspB', 'WspC', 'WspD', 'WspE', 'WspF']
prefasta = []
if "Wsp_operon.aln" not in existingFiles:
    for i in operon_dict_refined:
        wspa = []
        wspb = []
        wspc = []
        wspd = []
        wspe = []
        wspf = []
        gene_list = wspa, wspb, wspc, wspd, wspe, wspf
        genome = i
        operon = operon_dict_refined[i]
        operon_list = operon.rstrip().rsplit(";")
        for hit_num in range(len(gene_list)):
            for entry in operon_list:
                if geneAccession[hit_num] in entry:
                    prot = (entry.rstrip().rsplit(",")[1])
                    in_fasta =
"../GeneFasta/"+geneAccession[hit_num]+".aln"
                    seqs = AlignIO.read(in_fasta,'fasta');
                    for record in seqs:
```

```
                        if record.id ==
(geneAccession[hit_num]+"_"+prot+"_"+i):
                            sequence = str(record.seq)
                            gene_list[hit_num].append(sequence)
        final_operon = (((str(gene_list)).replace("'],
['","").replace("'])","")).replace("(['","",))+"\n"
        header = ">"+rename_dict[i]+"\n"
        prefasta.append(header)
        prefasta.append(final_operon)

    with open("../GeneFasta/Wsp_operon.aln", 'w') as f:
        f.writelines(prefasta)

prefasta = []
if "Wsp_operon.aln.trimal" not in existingFiles:
    for i in operon_dict_refined:
        wspa = []
        wspb = []
        wspc = []
        wspd = []
        wspe = []
        wspf = []
        gene_list = wspa, wspb, wspc, wspd, wspe, wspf
        genome = i
        operon = operon_dict_refined[i]
        operon_list = operon.rstrip().rsplit(";")
        for hit_num in range(len(gene_list)):
            for entry in operon_list:
                if geneAccession[hit_num] in entry:
                    prot = (entry.rstrip().rsplit(",")[1])
                    in_fasta =
"../GeneFasta/"+geneAccession[hit_num]+".aln.trimal"
                    seqs = AlignIO.read(in_fasta,'fasta');
                    for record in seqs:
                        if record.id ==
(geneAccession[hit_num]+"_"+prot+"_"+i):
                            sequence = str(record.seq)
                            gene_list[hit_num].append(sequence)
        final_operon = (((str(gene_list)).replace("'],
['","").replace("'])","")).replace("(['","",))+"\n"
        header = ">"+rename_dict[i]+"\n"
        prefasta.append(header)
        prefasta.append(final_operon)

    with open("../GeneFasta/Wsp_operon.aln.trimal", 'w') as f:
        f.writelines(prefasta)


# In[28]:


# modify the wsp operon database that ties each wsp gene to its specified
organism
existingFiles = os.listdir("../GeneFasta/")
```

```
geneAccession = ['WspA', 'WspB', 'WspC', 'WspD', 'WspE', 'WspF']
prefasta = []
H_dict = {}
R_dict = {}
for i in operon_dict_refined:
    operon = operon_dict_refined[i]
    operon_list = operon.rstrip().rsplit(";")
    for hit_vals in operon_list:
        if "WspH" in hit_vals:
            H_dict.update({i:operon_dict_refined[i]})
        if "WspR" in hit_vals:
            R_dict.update({i:operon_dict_refined[i]})


# In[29]:


#remove the grouped .fasta files to be used for shannon entropy
comparisons
h_dir = "../GeneFasta/WspH/*"
r_dir = "../GeneFasta/WspR/*"
get_ipython().system('rm -fr $h_dir $r_dir')


# In[30]:


#CREATE the fasta files to be used for shannon entropy analysis of the H-
and R-systems

existingFiles = os.listdir("../GeneFasta/")
new_dir = "../GeneFasta/For_Shannon"
get_ipython().system('rm -rf $new_dir')
get_ipython().system('mkdir $new_dir')
geneAccession = ['WspA', 'WspB', 'WspC', 'WspD', 'WspE', 'WspF',
'WspR_H']
geneAccession_iter = ['WspA', 'WspB', 'WspC', 'WspD', 'WspE', 'WspF']


#make the wspR specific files
for i in R_dict:
    genome = i
    operon = R_dict[i]
    operon_list = operon.rstrip().rsplit(";")
    for entry in operon_list:
        if "WspR" in entry:
            prot = (entry.rstrip().rsplit(",")[1])
            in_fasta = "../GeneFasta/WspR_H.aln.trimal"
            seqs = AlignIO.read(in_fasta,'fasta');
            for record in seqs:
                if record.id == ("WspR_H_"+prot+"_"+i):
                    with
open('../GeneFasta/For_Shannon/WspR_(R)_for_shannon.fasta', 'a') as
output_handle:
```

```
                        SeqIO.write(record, output_handle, "fasta")
        for wsp_protein in geneAccession_iter:
            if wsp_protein in entry:
                prot = (entry.rstrip().rsplit(",")[1])
                in_fasta = "../GeneFasta/" + wsp_protein + ".aln.trimal"
                seqs = AlignIO.read(in_fasta,'fasta');
                for record in seqs:
                    if record.id == (wsp_protein +"_"+prot+"_"+i):
                        with open('../GeneFasta/For_Shannon/' +
wsp_protein + '_(R)_for_shannon.fasta', 'a') as output_handle:
                            SeqIO.write(record, output_handle, "fasta")

#make the wspH specific files
for i in H_dict:
    genome = i
    operon = H_dict[i]
    operon_list = operon.rstrip().rsplit(";")
    for entry in operon_list:
        if "WspH" in entry:
            prot = (entry.rstrip().rsplit(",")[1])
            in_fasta = "../GeneFasta/WspR_H.aln.trimal"
            seqs = AlignIO.read(in_fasta,'fasta');
            for record in seqs:
                if record.id == ("WspR_H_"+prot+"_"+i):
                    with
open('../GeneFasta/For_Shannon/WspH_(H)_for_shannon.fasta', 'a') as
output_handle:
                        SeqIO.write(record, output_handle, "fasta")
        for wsp_protein in geneAccession_iter:
            if wsp_protein in entry:
                prot = (entry.rstrip().rsplit(",")[1])
                in_fasta = "../GeneFasta/" + wsp_protein + ".aln.trimal"
                seqs = AlignIO.read(in_fasta,'fasta');
                for record in seqs:
                    if record.id == (wsp_protein +"_"+prot+"_"+i):
                        with open('../GeneFasta/For_Shannon/' +
wsp_protein + '_(H)_for_shannon.fasta', 'a') as output_handle:
                            SeqIO.write(record, output_handle, "fasta")
#make all other files
for hit_num in range(len(geneAccession)):
    in_fasta = "../GeneFasta/"+geneAccession[hit_num]+".aln.trimal"
    seqs = AlignIO.read(in_fasta,'fasta');
    for record in seqs:
        if "CP001340" not in record.id:
            with open('../GeneFasta/For_Shannon/' +
geneAccession[hit_num] + "_for_shannon.fasta", 'a') as output_handle:
                SeqIO.write(record, output_handle, "fasta")


#make consensus of 794 SEQ alignments
for a in geneAccession:
    b = '../GeneFasta/For_Shannon/' + a + '_for_shannon.fasta'
    c = '../GeneFasta/For_Shannon/' + a + '_consensus.fasta'
    seqs = AlignIO.read(b,'fasta')
```

```
    summary_align = AlignInfo.SummaryInfo(seqs)
    consensus = summary_align.gap_consensus()

    #export consensus
    new_record = SeqRecord(
    consensus,
    id=a,
    name="",
    description="",
    )
    with open(c, "w") as d:
        SeqIO.write(new_record, d, "fasta")

#make consensus of alignment subsets
geneAccession_H = ['WspA', 'WspB', 'WspC', 'WspD', 'WspE', 'WspF',
'WspH']
for a in geneAccession_H:
    b = '../GeneFasta/For_Shannon/' + a + '_(H)_for_shannon.fasta'
    c = '../GeneFasta/For_Shannon/' + a + '_(H)_consensus.fasta'
    seqs = AlignIO.read(b,'fasta')
    summary_align = AlignInfo.SummaryInfo(seqs)
    consensus = summary_align.gap_consensus()

    #export consensus
    new_record = SeqRecord(
    consensus,
    id=a,
    name="",
    description="",
    )
    with open(c, "w") as d:
        SeqIO.write(new_record, d, "fasta")

geneAccession_R = ['WspA', 'WspB', 'WspC', 'WspD', 'WspE', 'WspF',
'WspR']
for a in geneAccession_R:
    b = '../GeneFasta/For_Shannon/' + a + '_(R)_for_shannon.fasta'
    c = '../GeneFasta/For_Shannon/' + a + '_(R)_consensus.fasta'
    seqs = AlignIO.read(b,'fasta')
    summary_align = AlignInfo.SummaryInfo(seqs)
    consensus = summary_align.gap_consensus()

    #export consensus
    new_record = SeqRecord(
    consensus,
    id=a,
    name="",
    description="",
    )
    with open(c, "w") as d:
        SeqIO.write(new_record, d, "fasta")


# In[31]:
```

```
#print warning if shannon entropy calculations/data cannot be found
existing_shannon_Files = os.listdir("../GeneShannon/")

if "WspA.shannon" not in existing_shannon_Files:
    print("One or more Shannon Entropy inputs cannot be found. Take the
fasta files from GeneFasta/WspR/Wsp*.fasta, run them on
https://compbio.cs.princeton.edu/conservation/ with select settings, and
save the file as GeneShannon/WspR/Wsp*.shannon in the respective folder.
Press ENTER when ready...")
    input()


# In[32]:


#Evaluate differences between R-system and H-system alignment subsets
summary_statistics = []
comparison_lookup_table_H = []
comparison_lookup_table_R = []
comparison_metadata = []
comparison_metadata_non_un = []

## main analysis
wspa = []
wspb = []
wspc = []
wspd = []
wspe = []
wspf = []
wsphr = []
geneAccession = ['WspA', 'WspB', 'WspC', 'WspD', 'WspE', 'WspF',
'WspH_R']
gene_list_wsp = [wspa, wspb, wspc, wspd, wspe, wspf, wsphr]
for i in range(len(geneAccession)):
    with open('../GeneShannon/' + geneAccession[i] + '.shannon') as f:
        count=0
        for line in f:
            a = line.rstrip("\r\n").rsplit("\t")
            if len(a) > 1:
                if "score" not in a:
                    b = float(a[1])
                    if b > -1000:
                        count += 1
                        c = count,b
                        gene_list_wsp[i].append(c)
    gene_list_wsp[i] = np.asarray(gene_list_wsp[i])
    count=0
    for ii in gene_list_wsp[i]:
        if ii[1] >= .8:
            count += 1
    percent_conservation = (count/(len(gene_list_wsp[i])+1))
```

```
    hh = (geneAccession[i] + "% >.8:  "
"{:.0%}".format(float(percent_conservation)))
    summary_statistics.append(hh)

## H subset analysis
wspa_H = []
wspb_H = []
wspc_H = []
wspd_H = []
wspe_H = []
wspf_H = []
wsph_H = []
geneAccession = ['WspA', 'WspB', 'WspC', 'WspD', 'WspE', 'WspF', 'WspH']
gene_list_wsp = [wspa_H, wspb_H, wspc_H, wspd_H, wspe_H, wspf_H, wsph_H]
for i in range(len(geneAccession)):
    with open('../GeneShannon/WspH/' + geneAccession[i] +
'_(H)_for_shannon.txt') as f:
        count=0
        for line in f:
            a = line.rstrip("\r\n").rsplit("\t")
            if len(a) > 1:
                if "score" not in a:
                    b = float(a[1])
                    if b > -1000:
                        count += 1
                        c = count,b
                        gene_list_wsp[i].append(c)
    gene_list_wsp[i] = np.asarray(gene_list_wsp[i])
    count=0
    for ii in gene_list_wsp[i]:
        if ii[1] >= .8:
            count += 1
            gg = "H-system", geneAccession[i], ii[0], ii[1]
            comparison_lookup_table_H.append(gg)
    percent_conservation = (count/(len(gene_list_wsp[i])+1))
    hh = (geneAccession[i] + "(H)% >.8:  "
"{:.0%}".format(float(percent_conservation)))
    summary_statistics.append(hh)

## R subset analysis
wspa_R = []
wspb_R = []
wspc_R = []
wspd_R = []
wspe_R = []
wspf_R = []
wspr_R = []
geneAccession = ['WspA', 'WspB', 'WspC', 'WspD', 'WspE', 'WspF', 'WspR']
gene_list_wsp = [wspa_R, wspb_R, wspc_R, wspd_R, wspe_R, wspf_R, wspr_R]
for i in range(len(geneAccession)):
    with open('../GeneShannon/WspR/' + geneAccession[i] +
'_(R)_for_shannon.txt') as f:
        count=0
        for line in f:
```

```
            a = line.rstrip("\r\n").rsplit("\t")
            if len(a) > 1:
                if "score" not in a:
                    b = float(a[1])
                    if b > -1000:
                        count += 1
                        c = count,b
                        gene_list_wsp[i].append(c)
    gene_list_wsp[i] = np.asarray(gene_list_wsp[i])
    count=0
    for ii in gene_list_wsp[i]:
        if ii[1] >= .8:
            count += 1
            gg = "R-system", geneAccession[i], ii[0], ii[1]
            comparison_lookup_table_R.append(gg)
    percent_conservation = (count/(len(gene_list_wsp[i])+1))
    hh = (geneAccession[i] + "(R)% >.8:  "
"{:.0%}".format(float(percent_conservation)))
    summary_statistics.append(hh)

## Evaluating the conservation differences for high conservation
locations in the R-system
geneAccession = ['WspA', 'WspB', 'WspC', 'WspD', 'WspE', 'WspF']
for R_entry in comparison_lookup_table_R: ### identify high cons. sites
(loop)
    for H_entry in comparison_lookup_table_H:
        for ff in geneAccession:
            if ff in R_entry:
                if ff in H_entry:
                    if R_entry[2] == H_entry[2]: ### determine residue(s)
at high cons. sites
                        R_cons_seq = str()
                        with open('../GeneFasta/For_Shannon/' + ff +
'_(R)_consensus.fasta') as R_f:
                            for line in R_f:
                                if ">" not in line:
                                    R_cons_seq += line
                        R_cons_seq = R_cons_seq.replace("\n","")
                        H_cons_seq = str()
                        with open('../GeneFasta/For_Shannon/' + ff +
'_(H)_consensus.fasta') as H_f:
                            for line in H_f:
                                if ">" not in line:
                                    H_cons_seq += line
                        H_cons_seq = H_cons_seq.replace("\n","")
                        if R_cons_seq[int(R_entry[2]-1)] !=
H_cons_seq[int(H_entry[2]-1)]:
                            new_entry =
"R_system",ff,int(R_entry[2]),R_cons_seq[int(R_entry[2]-
1)],R_entry[3],"H_system",ff,int(H_entry[2]),H_cons_seq[int(H_entry[2]-
1)],H_entry[3]
                            comparison_metadata.append(new_entry)
```

```
                        if R_cons_seq[int(R_entry[2]-1)] ==
H_cons_seq[int(H_entry[2]-1)]:
                            new_entry =
"R_system",ff,int(R_entry[2]),R_cons_seq[int(R_entry[2]-
1)],R_entry[3],"H_system",ff,int(H_entry[2]),H_cons_seq[int(H_entry[2]-
1)],H_entry[3]
                            comparison_metadata_non_un.append(new_entry)

for R_entry in comparison_lookup_table_R: ### identify high cons. sites
(static)
    for H_entry in comparison_lookup_table_H:
        if 'WspR' in R_entry:
            if 'WspH' in H_entry:
                if R_entry[2] == H_entry[2]: ### determine residue(s) at
high cons. sites
                    R_cons_seq = str()
                    with
open('../GeneFasta/For_Shannon/WspR_(R)_consensus.fasta') as R_f:
                        for line in R_f:
                            if ">" not in line:
                                R_cons_seq += line
                    R_cons_seq = R_cons_seq.replace("\n","")
                    H_cons_seq = str()
                    with
open('../GeneFasta/For_Shannon/WspH_(H)_consensus.fasta') as H_f:
                        for line in H_f:
                            if ">" not in line:
                                H_cons_seq += line
                    H_cons_seq = H_cons_seq.replace("\n","")
                    if R_cons_seq[int(R_entry[2]-1)] !=
H_cons_seq[int(H_entry[2]-1)]:
                        new_entry =
"R_system","WspR",int(R_entry[2]),R_cons_seq[int(R_entry[2]-
1)],R_entry[3],"H_system","WspH",int(H_entry[2]),H_cons_seq[int(H_entry[2
]-1)],H_entry[3]
                        comparison_metadata.append(new_entry)
comparison_metadata = np.asarray(comparison_metadata)
summary_statistics = np.asarray(summary_statistics)


np.savetxt(r'../ShannonEntropyData_H_and_R/Comparison_Report.txt',
comparison_metadata, fmt='%s', delimiter=",")
np.savetxt(r'../ShannonEntropyData_H_and_R/Percent_high_conservation_repo
rt.txt', summary_statistics, fmt='%s', delimiter=",")


# In[33]:


## Evaluate the residues identified above assuming R-system -> H-system
for cons. or non-cons substitutions
## will use the following dataset:
##        Pechmann, Sebastian; Frydman, Judith (2015): Conservative and
non-conservative amino acid
```

```
##        substitutions.. PLOS Computational Biology. Dataset.
https://doi.org/10.1371/journal.pcbi.1003674.t001
sub_df = []
nc_sub_df = []
conservative_subs_dict = {}
nonconservative_subs_dict = {}

with open('../INPUT/Plos_comp_bio_subs.txt') as f:
    for line in f:
        entry = line.rstrip().rsplit("\t")
        original = entry[0]
        conservative_substitutions = (entry[1]).rstrip().rsplit(",")
        nonconservative_substitutions = (entry[2]).rstrip().rsplit(",")

conservative_subs_dict.update({original:conservative_substitutions})

nonconservative_subs_dict.update({original:nonconservative_substitutions}
)

#print(conservative_subs_dict['Y'])
#print(nonconservative_subs_dict['F'])
count = 0
for aa in comparison_metadata:
    outcome = []
    key = aa[3]
    value_to_test = aa[8]
    if 'X' not in key:
        if 'X' not in value_to_test:
              count+=1
        cons_rep = conservative_subs_dict[key]
        non_cons_rep = nonconservative_subs_dict[key]

        if value_to_test in cons_rep:
            outcome = "conservative"
        if value_to_test in non_cons_rep:
            outcome = "non-conservative"
        if outcome == []:
            if 'X' not in value_to_test:
                outcome = "non-conservative"
        new_row = aa[0], aa[1], aa[2], aa[3], aa[4], aa[5], aa[6], aa[7],
aa[8], aa[9], outcome
        sub_df.append(new_row)
        if outcome == "non-conservative":
            nc_sub_df.append(new_row)

print(len(comparison_metadata)+len(comparison_metadata_non_un), "total
conserved pairs")
print(count, "of", len(comparison_metadata), "have meaningful data")
print(count - len(nc_sub_df), "mutations are conservative")
print(len(nc_sub_df), "mutations are non-conservative")
nc_sub_df = np.asarray(nc_sub_df)
np.savetxt(r'../ShannonEntropyData_H_and_R/Non-
conservative_mutations.txt', nc_sub_df, fmt='%s', delimiter=",")
```

```
# In[40]:


#Evaluate Conservation of BOTH wspH/R systems to identify fundamental
residues needed for all wsp signalling

existing_shannon_Files = os.listdir("../GeneShannon/")
distribution_of_all_data = []
geneAccession = ['WspA', 'WspB', 'WspC', 'WspD', 'WspE', 'WspF',
'WspH_R', 'WspH', 'WspR']

high_shannon_dir = "../GeneShannon/Reports/"
get_ipython().system('rm -fr $high_shannon_dir')
get_ipython().system('mkdir $high_shannon_dir')

for i in range(len(geneAccession)):
    ## Initialize data storage
    wsp_type = "../GeneShannon/"
    wspa = []
    wspb = []
    wspc = []
    wspd = []
    wspe = []
    wspf = []
    wsphr = []
    wsph = []
    wspr = []
    high_score_residues = []

    ## Evaluate Shannon Entropy files and record data
    gene_list_wsp = wspa, wspb, wspc, wspd, wspe, wspf, wsphr, wsph, wspr
    if geneAccession[i]+".shannon" in existing_shannon_Files:
        print("opening:", "../GeneShannon/" + geneAccession[i] +
".shannon")
        with open("../GeneShannon/" + geneAccession[i] + ".shannon") as
f:
            count=0
            for line in f:
                a = line.rstrip("\r\n").rsplit("\t")
                if len(a) > 1:
                    if "score" not in a:
                        b = float(a[1])
                        if b > -1000:
                            count += 1
                            c = count,b
                            gene_list_wsp[i].append(c)

    wsp_array = np.asarray(gene_list_wsp[i])


    ## Prepair to plot the Data
    Title = geneAccession[i]
    gridkw = dict(height_ratios=[3, 1, 4], wspace=0, hspace=0.01)
```

```
    fig, (ax1, ax2, ax3) = plt.subplots(3, 1, gridspec_kw=gridkw,
sharex=True, figsize=(28,4))


    ## PLOT DOMAIN INFORMATION (updated Sept 2020)
    Ann_data = []
    Ann_array = []
    Annotation_File = ("../INPUT/Annotation_trimal/" + geneAccession[i] +
"_annotation.txt")
    with open(Annotation_File) as Ann:
        for line in Ann:
            d = line.rstrip("\r\n").rsplit("\t")
            Ann_data.append(d)
            Ann_array = np.asarray(Ann_data)

        xstart = min(wsp_array[0:,0].astype(int))
        xstop = max(wsp_array[0:,0].astype(int))

        ax2.add_patch(
        patches.Rectangle(
        (xstart, 0.4), # (x,y)
        xstop, # width
        0.4, # height
        # You can add rotation as well with 'angle'
        alpha=0.2, facecolor="black", edgecolor="black", linewidth=0,
linestyle='solid',zorder=1
        )
        )

        count = 0

        if len(Ann_array) > 0:
            for line in Ann_array:

                domain = line[0]
                color_index = sel_annotations.index(domain)
                start = int(line[1])
                stop = int(line[2])
                midpoint = ((stop-start)/2) + start

                ax2.add_patch(
                patches.Rectangle(
                (start, 0.2), # (x,y)
                stop-start, # width
                0.8, # height
                # You can add rotation as well with 'angle'
                alpha=1, facecolor=sel_colors[color_index],
edgecolor="black", linewidth=0, linestyle='solid',zorder=2
                )
                )
                ax2.annotate("", xy=(start, 0.1), xycoords='data',
xytext=(stop, 0.1), textcoords='data')
```

```
                ax2.text(midpoint, 0.6, domain,
horizontalalignment='center',verticalalignment='center',fontsize=20,
wrap=True)
                count += 1

    ## PLOT MUTATION DATA
    R_muts = []
    with open("../INPUT/Lit_Mutations_trimal/" + geneAccession[i] +
"_R_operon.txt") as R_muts_to_plot:
        for line in R_muts_to_plot:
            AA = (line.replace("\n",""))
            BB = AA.rstrip().rsplit(",")
            if BB[2] == 'OFF':
                on_off_state = 0
            if BB[2] == 'ON':
                on_off_state = 1
            CC = int(BB[0]), int(BB[1]), on_off_state
            R_muts.append(CC)
    R_muts = np.asarray(R_muts)

    H_muts = []
    with open("../INPUT/Lit_Mutations_trimal/" + geneAccession[i] +
"_H_operon.txt") as H_muts_to_plot:
        for line in H_muts_to_plot:
            AA = (line.replace("\n",""))
            BB = AA.rstrip().rsplit(",")
            if BB[2] == 'OFF':
                on_off_state = 0
            if BB[2] == 'ON':
                on_off_state = 1
            CC = int(BB[0]), int(BB[1]), on_off_state
            H_muts.append(CC)
    H_muts = np.asarray(H_muts)

    synth_muts = []
    with open("../INPUT/Lit_Mutations_trimal/" + geneAccession[i] +
"_synth_operon.txt") as synth_muts_to_plot:
        for line in synth_muts_to_plot:
            AA = (line.replace("\n",""))
            BB = AA.rstrip().rsplit(",")
            if BB[2] == 'OFF':
                on_off_state = 0
            if BB[2] == 'ON':
                on_off_state = 1
            CC = int(BB[0]), int(BB[1]), on_off_state
            synth_muts.append(CC)
    synth_muts = np.asarray(synth_muts)

    if len(H_muts) > 0:

        for ii in H_muts:
            x_coord = ii[0]
            if ii[1] == 1:
                height = 0.15
```

```
            if ii[1] == 2:
                height = 0.44
            if ii[1] == 3:
                height = 0.72

ax1.axvline(x=x_coord,ymin=0,ymax=height,c="black",linewidth=2,zorder=0,
clip_on=False)
            ax2.axvline(x=x_coord,ymin=0.8,ymax=1,c="black",linewidth=2,
zorder=0,clip_on=False)

        ON_muts = []
        OFF_muts = []
        for ii in H_muts:
            if ii[2] == 1:
                plot_pair = ii[0], ii[1]
                ON_muts.append(plot_pair)
            if ii[2] == 0:
                plot_pair = ii[0], ii[1]
                OFF_muts.append(plot_pair)
        ON_muts = np.asarray(ON_muts)
        OFF_muts = np.asarray(OFF_muts)

        if len(ON_muts) > 0:
            ax1.scatter(ON_muts[0:,0], ON_muts[0:,1], s=400,zorder=1,
label="H (On)", facecolors = '#1f77b4', alpha = 0.5)
        if len(OFF_muts) > 0:
            ax1.scatter(OFF_muts[0:,0], OFF_muts[0:,1], marker='^',
s=400,zorder=1, label="H (Off)", facecolors = '#1f77b4', alpha = 0.5)

    if len(R_muts) > 0:
        for ii in R_muts:
            x_coord = ii[0]
            if ii[1] == 1:
                height = 0.15
            if ii[1] == 2:
                height = 0.44
            if ii[1] == 3:
                height = 0.72

ax1.axvline(x=x_coord,ymin=0,ymax=height,c="black",linewidth=2,zorder=0,
clip_on=False)
            ax2.axvline(x=x_coord,ymin=0.8,ymax=1,c="black",linewidth=2,
zorder=0,clip_on=False)

        ON_muts = []
        OFF_muts = []
        for ii in R_muts:
            if ii[2] == 1:
                plot_pair = ii[0], ii[1]
                ON_muts.append(plot_pair)
            if ii[2] == 0:
                plot_pair = ii[0], ii[1]
                OFF_muts.append(plot_pair)
        ON_muts = np.asarray(ON_muts)
```

```
        OFF_muts = np.asarray(OFF_muts)

        if len(ON_muts) > 0:
            ax1.scatter(ON_muts[0:,0], ON_muts[0:,1], s=400,zorder=1,
label="R (On)", facecolors = '#ff7f0e', alpha = 0.5)
        if len(OFF_muts) > 0:
            ax1.scatter(OFF_muts[0:,0], OFF_muts[0:,1], marker='^',
s=400,zorder=1, label="R (Off)", facecolors = '#ff7f0e', alpha = 0.5)

        #ax1.scatter(R_muts[0:,0], R_muts[0:,1], s=400,zorder=1,
label="R-system", facecolors = '#ff7f0e', alpha = 0.5)

    if len(synth_muts) > 0:
        for ii in synth_muts:
            x_coord = ii[0]
            if ii[1] == 1:
                height = 0.15
            if ii[1] == 2:
                height = 0.44
            if ii[1] == 3:
                height = 0.72

ax1.axvline(x=x_coord,ymin=0,ymax=height,c="black",linewidth=2,zorder=0,
clip_on=False)
            ax2.axvline(x=x_coord,ymin=0.8,ymax=1,c="black",linewidth=2,
zorder=0,clip_on=False)

        ON_muts = []
        OFF_muts = []
        for ii in synth_muts:
            if ii[2] == 1:
                plot_pair = ii[0], ii[1]
                ON_muts.append(plot_pair)
            if ii[2] == 0:
                plot_pair = ii[0], ii[1]
                OFF_muts.append(plot_pair)
        ON_muts = np.asarray(ON_muts)
        OFF_muts = np.asarray(OFF_muts)

        if len(ON_muts) > 0:
            ax1.scatter(ON_muts[0:,0], ON_muts[0:,1], s=400,zorder=1,
label="Clone (On)", facecolors = 'none', edgecolors='black')
        if len(OFF_muts) > 0:
            ax1.scatter(OFF_muts[0:,0], OFF_muts[0:,1], marker='^',
s=400,zorder=1, label="Clone (Off)", facecolors = 'none',
edgecolors='black')

        #ax1.scatter(synth_muts[0:,0], synth_muts[0:,1], s=400,zorder=1,
label="engineered", facecolors='none', edgecolors='black')



    ## PLOT ENTOPY HIGHLIGHTS
    conservation_value = 0.8
```

```
    wsp_array_cons = []
    export_cons_array = []
    for line in wsp_array:
        if line[1] > conservation_value:
            new_score_value = conservation_value
        if line[1] >= conservation_value:
            export_cons_array.append(line)
        if line[1] <= conservation_value:
            new_score_value = line[1]
        tupp = line[0], new_score_value
        wsp_array_cons.append(tupp)
    wsp_array_cons = np.asarray(wsp_array_cons)
    export_cons_array = np.asarray(export_cons_array)
    ## Plot main entropy graph
    sns.lineplot(x=wsp_array[0:,0], y=wsp_array[0:,1],ax=ax3,
color="black", zorder=1)
    ax3.fill_between(wsp_array[0:,0], wsp_array[0:,1], y2=0, color =
"red")
    ax3.fill_between(wsp_array_cons[0:,0], wsp_array_cons[0:,1], y2=0,
color = "white")

    #sns.lineplot(x=wsp_array_cons[0:,0], y=wsp_array_cons[0:,1],ax=ax3,
color="black", zorder=1)


    ## ASTHETICS
    if "WspH_R" in Title:
        Title = "WspH/R"

    fig.suptitle(Title, fontsize=30)

    ax1.set_ylim(0, 3.5)
    ax1.set_yticks([0,1,2,3])
    ax1.set_yticklabels([0,'','',3], fontsize=20)
    ax1.set_ylabel("Mutations", fontsize=15)
    #ax1.legend(frameon=False, fontsize=15,  ncol=1, borderaxespad=1,
bbox_to_anchor=(0., 1.5, 1., .102))


    ax2.set_yticks([])
    ax2.set_xticks([])
    ax2.tick_params(bottom=False, left=False)

    xrange = (range(min(wsp_array[0:,0].astype(int))-1,
max(wsp_array[0:,0].astype(int))+1, 50))


    ax3.set_ylim(0, 1.1)
    ax3.set_yticks([0,0.5,1])
    ax3.set_yticklabels([0,'',1], fontsize=20)
```

```
    sns.despine(ax=ax1,top=True, right=True, left=False, bottom=True,
trim=True)
    sns.despine(ax=ax2,top=True, right=True, left=True, bottom=True,
trim=True)
    sns.despine(ax=ax3,top=True, right=True, left=False, bottom=False,
trim=True)

    ax3.set_xticks(xrange)
    ax3.set_xticklabels(xrange,fontsize=20, rotation=0)
    ax3.set_xlabel("Amino Acid Alignment Position", fontsize=20)
    ax3.set_ylabel("Cons. Score", fontsize=15)

    fig.align_ylabels()
    savename = str("../ShannonEntropyData_H_and_R/" + geneAccession[i] +
".svg")
    plt.savefig(savename, bbox_inches='tight')

    np.savetxt(high_shannon_dir + geneAccession[i] +
"shannon_summary.txt", export_cons_array, delimiter=',', fmt='%f')
    #plt.close()


# In[39]:


#Graph differences in conservation between H- and R- systems

R_color = '#ff7f0e'
H_color = '#1f77b4'

H_shannon_Files = os.listdir("../GeneShannon/WspH/")
R_shannon_Files = os.listdir("../GeneShannon/WspR/")


wspa_R = []
wspb_R = []
wspc_R = []
wspd_R = []
wspe_R = []
wspf_R = []
wsph_r_R = []
wspa_H = []
wspb_H = []
wspc_H = []
wspd_H = []
wspe_H = []
wspf_H = []
wsph_r_H = []


distribution_of_all_data = []
geneAccession = ['WspA', 'WspB', 'WspC', 'WspD', 'WspE', 'WspF',
'WspH_R']
```

```
gene_list_wsp_H = (wspa_H, wspb_H, wspc_H, wspd_H, wspe_H, wspf_H,
wsph_r_H)
gene_list_wsp_R = (wspa_R, wspb_R, wspc_R, wspd_R, wspe_R, wspf_R,
wsph_r_R)

for i in range(len(geneAccession)):
    R_shannon_set = []
    H_shannon_set = []
    for entry in H_shannon_Files:
        if geneAccession[i] in entry:
            #print(entry)
            print("opening:", "../GeneShannon/WspH/" + entry)
            with open("../GeneShannon/WspH/" + entry) as f:
                count=0
                for line in f:
                    a = line.rstrip("\r\n").rsplit("\t")
                    if len(a) > 1:
                        if "score" not in a:
                            b = float(a[1])
                            if b > -1000:
                                count += 1
                                c = count,b
                                gene_list_wsp_H[i].append(c)

    for entry in R_shannon_Files:
        if geneAccession[i] in entry:
            #print(entry)
            print("opening:", "../GeneShannon/WspR/" + entry)
            with open("../GeneShannon/WspR/" + entry) as f:
                count=0
                for line in f:
                    a = line.rstrip("\r\n").rsplit("\t")
                    if len(a) > 1:
                        if "score" not in a:
                            b = float(a[1])
                            if b > -1000:
                                count += 1
                                c = count,b
                                gene_list_wsp_R[i].append(c)

    wsp_H_array = np.asarray(gene_list_wsp_H[i])
    wsp_R_array = np.asarray(gene_list_wsp_R[i])
    #print(geneAccession[i], len(wsp_H_array))
    #print(geneAccession[i], len(wsp_R_array))

    ## Prepair to plot the Data
    Title = geneAccession[i]
    gridkw = dict(height_ratios=[3, 1, 4], wspace=0, hspace=0.01)
    fig, (ax1, ax2, ax3) = plt.subplots(3, 1, gridspec_kw=gridkw,
sharex=True, figsize=(28,4))

    ## PLOT ENTOPY DATA
    ## Plot main entropy graph
```

```
    ax3.axhline(0.8, color = 'red', alpha = 0.7, linestyle = '--',
zorder=1)
    sns.lineplot(x=wsp_H_array[0:,0], y=wsp_H_array[0:,1],ax=ax3,
color=H_color, zorder=3)
    sns.lineplot(x=wsp_R_array[0:,0], y=wsp_R_array[0:,1],ax=ax3,
color=R_color, zorder=3)


    ## PLOT DOMAIN INFORMATION (updated March 2021)
    Ann_data = []
    Ann_array = []
    Annotation_File = ("../INPUT/Annotation_trimal/" + geneAccession[i] +
"_annotation.txt")
    with open(Annotation_File) as Ann:
        for line in Ann:
            d = line.rstrip("\r\n").rsplit("\t")
            Ann_data.append(d)
            Ann_array = np.asarray(Ann_data)

        xstart = min(wsp_H_array[0:,0].astype(int))
        xstop = max(wsp_H_array[0:,0].astype(int))

        ax2.add_patch(
        patches.Rectangle(
        (xstart, 0.4), # (x,y)
        xstop, # width
        0.4, # height
        # You can add rotation as well with 'angle'
        alpha=0.2, facecolor="black", edgecolor="black", linewidth=0,
linestyle='solid',zorder=1
        )
        )

        count = 0
        if len(Ann_array) > 0:
            for line in Ann_array:

                domain = line[0]
                color_index = sel_annotations.index(domain)
                start = int(line[1])
                stop = int(line[2])
                midpoint = ((stop-start)/2) + start

                ax2.add_patch(
                patches.Rectangle(
                (start, 0.2), # (x,y)
                stop-start, # width
                0.8, # height
                # You can add rotation as well with 'angle'
                alpha=1, facecolor=sel_colors[color_index],
edgecolor="black", linewidth=0, linestyle='solid',zorder=2
                )
                )
```

```
                ax2.annotate("", xy=(start, 0.1), xycoords='data',
xytext=(stop, 0.1), textcoords='data')
                ax2.text(midpoint, 0.6, domain,
horizontalalignment='center',verticalalignment='center',fontsize=20,
wrap=True)
                count += 1


    ## PLOT MUTATION DATA
    R_muts = []
    with open("../INPUT/Lit_Mutations_trimal/" + geneAccession[i] +
"_R_operon.txt") as R_muts_to_plot:
        for line in R_muts_to_plot:
            AA = (line.replace("\n",""))
            BB = AA.rstrip().rsplit(",")
            if BB[2] == 'OFF':
                on_off_state = 0
            if BB[2] == 'ON':
                on_off_state = 1
            CC = int(BB[0]), int(BB[1]), on_off_state
            R_muts.append(CC)
    R_muts = np.asarray(R_muts)

    H_muts = []
    with open("../INPUT/Lit_Mutations_trimal/" + geneAccession[i] +
"_H_operon.txt") as H_muts_to_plot:
        for line in H_muts_to_plot:
            AA = (line.replace("\n",""))
            BB = AA.rstrip().rsplit(",")
            if BB[2] == 'OFF':
                on_off_state = 0
            if BB[2] == 'ON':
                on_off_state = 1
            CC = int(BB[0]), int(BB[1]), on_off_state
            H_muts.append(CC)
    H_muts = np.asarray(H_muts)

    synth_muts = []
    with open("../INPUT/Lit_Mutations_trimal/" + geneAccession[i] +
"_synth_operon.txt") as synth_muts_to_plot:
        for line in synth_muts_to_plot:
            AA = (line.replace("\n",""))
            BB = AA.rstrip().rsplit(",")
            if BB[2] == 'OFF':
                on_off_state = 0
            if BB[2] == 'ON':
                on_off_state = 1
            CC = int(BB[0]), int(BB[1]), on_off_state
            synth_muts.append(CC)
    synth_muts = np.asarray(synth_muts)

    if len(H_muts) > 0:

        for ii in H_muts:
```

```
            x_coord = ii[0]
            if ii[1] == 1:
                height = 0.15
            if ii[1] == 2:
                height = 0.44
            if ii[1] == 3:
                height = 0.72

ax1.axvline(x=x_coord,ymin=0,ymax=height,c="black",linewidth=2,zorder=0,
clip_on=False)
            ax2.axvline(x=x_coord,ymin=0.8,ymax=1,c="black",linewidth=2,
zorder=0,clip_on=False)

        ON_muts = []
        OFF_muts = []
        for ii in H_muts:
            if ii[2] == 1:
                plot_pair = ii[0], ii[1]
                ON_muts.append(plot_pair)
            if ii[2] == 0:
                plot_pair = ii[0], ii[1]
                OFF_muts.append(plot_pair)
        ON_muts = np.asarray(ON_muts)
        OFF_muts = np.asarray(OFF_muts)

        if len(ON_muts) > 0:
            ax1.scatter(ON_muts[0:,0], ON_muts[0:,1], s=400,zorder=1,
label="H (On)", facecolors = '#1f77b4', alpha = 0.5)
        if len(OFF_muts) > 0:
            ax1.scatter(OFF_muts[0:,0], OFF_muts[0:,1], marker='^',
s=400,zorder=1, label="H (Off)", facecolors = '#1f77b4', alpha = 0.5)

    if len(R_muts) > 0:
        for ii in R_muts:
            x_coord = ii[0]
            if ii[1] == 1:
                height = 0.15
            if ii[1] == 2:
                height = 0.44
            if ii[1] == 3:
                height = 0.72

ax1.axvline(x=x_coord,ymin=0,ymax=height,c="black",linewidth=2,zorder=0,
clip_on=False)
            ax2.axvline(x=x_coord,ymin=0.8,ymax=1,c="black",linewidth=2,
zorder=0,clip_on=False)

        ON_muts = []
        OFF_muts = []
        for ii in R_muts:
            if ii[2] == 1:
                plot_pair = ii[0], ii[1]
                ON_muts.append(plot_pair)
            if ii[2] == 0:
```

```
                plot_pair = ii[0], ii[1]
                OFF_muts.append(plot_pair)
        ON_muts = np.asarray(ON_muts)
        OFF_muts = np.asarray(OFF_muts)

        if len(ON_muts) > 0:
            ax1.scatter(ON_muts[0:,0], ON_muts[0:,1], s=400,zorder=1,
label="R (On)", facecolors = '#ff7f0e', alpha = 0.5)
        if len(OFF_muts) > 0:
            ax1.scatter(OFF_muts[0:,0], OFF_muts[0:,1], marker='^',
s=400,zorder=1, label="R (Off)", facecolors = '#ff7f0e', alpha = 0.5)

        #ax1.scatter(R_muts[0:,0], R_muts[0:,1], s=400,zorder=1,
label="R-system", facecolors = '#ff7f0e', alpha = 0.5)

    if len(synth_muts) > 0:
        for ii in synth_muts:
            x_coord = ii[0]
            if ii[1] == 1:
                height = 0.15
            if ii[1] == 2:
                height = 0.44
            if ii[1] == 3:
                height = 0.72

ax1.axvline(x=x_coord,ymin=0,ymax=height,c="black",linewidth=2,zorder=0,
clip_on=False)
            ax2.axvline(x=x_coord,ymin=0.8,ymax=1,c="black",linewidth=2,
zorder=0,clip_on=False)

        ON_muts = []
        OFF_muts = []
        for ii in synth_muts:
            if ii[2] == 1:
                plot_pair = ii[0], ii[1]
                ON_muts.append(plot_pair)
            if ii[2] == 0:
                plot_pair = ii[0], ii[1]
                OFF_muts.append(plot_pair)
        ON_muts = np.asarray(ON_muts)
        OFF_muts = np.asarray(OFF_muts)

        if len(ON_muts) > 0:
            ax1.scatter(ON_muts[0:,0], ON_muts[0:,1], s=400,zorder=1,
label="Clone (On)", facecolors = 'none', edgecolors='black')
        if len(OFF_muts) > 0:
            ax1.scatter(OFF_muts[0:,0], OFF_muts[0:,1], marker='^',
s=400,zorder=1, label="Clone (Off)", facecolors = 'none',
edgecolors='black')

        #ax1.scatter(synth_muts[0:,0], synth_muts[0:,1], s=400,zorder=1,
label="engineered", facecolors='none', edgecolors='black')

    ## ASTHETICS
```

```
    if "WspH_R" in Title:
        Title = "WspH/R"

    fig.suptitle(Title, fontsize=30)

    ax1.set_ylim(0, 3.5)
    ax1.set_yticks([0,1,2,3])
    ax1.set_yticklabels([0,'','',3], fontsize=20)
    ax1.set_ylabel("Mutations", fontsize=15)
    #ax1.legend(frameon=False, fontsize=15,  ncol=1, borderaxespad=1,
bbox_to_anchor=(0., 1.5, 1., .102))


    ax2.set_yticks([])
    ax2.set_xticks([])
    ax2.tick_params(bottom=False, left=False)

    xrange = (range(min(wsp_H_array[0:,0].astype(int))-1,
max(wsp_H_array[0:,0].astype(int))+1, 50))


    ax3.set_ylim(0, 1.1)
    ax3.set_yticks([0,0.5,1])
    ax3.set_yticklabels([0,'',1], fontsize=20)



    sns.despine(ax=ax1,top=True, right=True, left=False, bottom=True,
trim=True)
    sns.despine(ax=ax2,top=True, right=True, left=True, bottom=True,
trim=True)
    sns.despine(ax=ax3,top=True, right=True, left=False, bottom=False,
trim=True)


    ax3.set_xticks(xrange)
    ax3.set_xticklabels(xrange,fontsize=20, rotation=0)
    ax3.set_xlabel("Amino Acid Alignment Position", fontsize=20)
    ax3.set_ylabel("Cons. Score", fontsize=15)

    fig.align_ylabels()
    savename = str("../ShannonEntropyData_H_vs_R/" + geneAccession[i] +
".svg")
    plt.savefig(savename, bbox_inches='tight')



    #plt.close()


# In[36]:


MGB_assembly_DGC_ezTax_pd.to_csv(r'../Run_Summary_Data.csv')
```

```
# #### Rename Species Fasta
# old_iter_file = "../species_120_reduced_renamed.fasta"
# !rm -fr $old_iter_file
#
#
# species_rename_dict = {}
# for index, row in MGB_assembly_DGC_ezTax_pd.iterrows():
#     if row["Assembly"] in bianary_species:
#         if row["Organism"] not in organisms_to_omit:
#             if "WspH" in row["Query"]:
#                 key =
(row["Assembly"].replace("_assembly_stats.txt",""))
#                 value = "WspH_" + (row["Organism"].replace("_",".")) +
"_" + row["EZ_Tax"]
#                 species_rename_dict.update({key:value})
#             if "WspR" in row["Query"]:
#                 key =
(row["Assembly"].replace("_assembly_stats.txt",""))
#                 value = "WspR_" + (row["Organism"].replace("_",".")) +
"_" + row["EZ_Tax"]
#                 species_rename_dict.update({key:value})
#             if "Caulobacter vibrioides NA1000" in row["EZ_Tax"]:
#                 key =
(row["Assembly"].replace("_assembly_stats.txt",""))
#                 value = "Root_" + (row["Organism"].replace("_",".")) +
"_" + row["EZ_Tax"]
#                 species_rename_dict.update({key:value})
# in_fasta = "../species_120_reduced.fasta"
# seqs = AlignIO.read(in_fasta,'fasta');
# for record in seqs:
#     if record.id in species_rename_dict:
#         new_record = SeqRecord(record.seq,
id=species_rename_dict[record.id], description='')
#         with open("../species_120_reduced_renamed.fasta", 'a') as
output_handle:
#             SeqIO.write(new_record, output_handle, "fasta")

# In[37]:


## 1000 bootstrap gene tree


# TreeInFile = "../GeneFasta/Wsp_operon.aln"
#
# !raxmlHPC-PTHREADS-AVX2 -s $TreeInFile -n wsp_operon -m PROTCATBLOSUM62
-T 8  -p 101 -x 123 -#1000

# In[ ]:
```