Abelardo Riojas
ISC 5317 Final Project Report
Fall 2021, Professor Beerli

Using Dendropy for Phylogenetic Analysis for Two Families of Pelecaniformes: Fregatidae and Phaethonidae

1. What is Dendropy?

DendroPy is a Python library for phylogenetic computing. It provides classes and functions for the simulation, processing, and manipulation of phylogenetic trees and character matrices, and supports the reading and writing of phylogenetic data in a range of formats.

- NEXUS
- NEWICK
- NeXML
- Phylip
- FASTA

Useful phylogenetic operations, such as data conversion and tree posterior distribution summarization, are also distributed and installed as part of the library.

2. What are we doing in Dendropy?

After doing some searching for phylogenetic data on Treebase, I found a particular Phylogenetic study that I really liked!

Kennedy M., & Spencer H. 2004. Phylogenies of the Frigatebirds (Fregatidae) and Tropicbirds (Phaethonidae), two divergent groups of the traditional order Pelecaniformes, inferred from mitochondrial DNA sequences. Molecular Phylogenetics and Evolution, 31: 31-38.

From the abstract: “The abstract: The frigatebirds (Fregatidae) and Tropicbirds (Phaethonidae) represent the most morphologically and behaviorally distinct members of the traditional Order Pelecaniformes. Using 1756 bp of mitochondrial DNA sequence consisting of the 12S, ATPase-6, ATPase-8, and COI genes obtained from all extant species, we derive a completely resolved phylogeny for both groups.”

3. First Jupyter cell: reading in data

In [1]:
```
import dendropy
taxa = dendropy.TaxonNamespace()
chars = dendropy.DnaCharacterMatrix.get(
    path="S1144.nex",
    schema="nexus",
    taxon_namespace=taxa)
treestr1 = '(Megadyptes_antipodes,(Diomedea_epomophora,((((Phaethon_aethereus,(Phaethon_lepturus,Phaethon_rubricauda)),
treestr2 = '(Megadyptes_antipodes,(((Diomedea_epomophora,(Phaethon_aethereus,(Phaethon_lepturus,Phaethon_rubricauda))),
treestr3 = '(((((Fregata_andrewsi,Fregata_minor),(Fregata_aquila,Fregata_magnificens)),Fregata_ariel),(Phaethon_aethere

tree1 = dendropy.Tree.get(data=treestr1,
        schema="newick", taxon_namespace=taxa)
tree2 = dendropy.Tree.get(data=treestr2,
        schema="newick", taxon_namespace=taxa)
tree3 = dendropy.Tree.get(data=treestr3,
        schema="newick", taxon_namespace=taxa)
```

Here we establish a common Taxon namespace, and read in the DNA sequence data from the Nexus file. The three associated trees from the file would not read in as a TreeList, so I manually extracted them from the .nex file and read them in as trees in Newick format.

4. Second Jupyter Cell: Population Genetics Summary Statistics

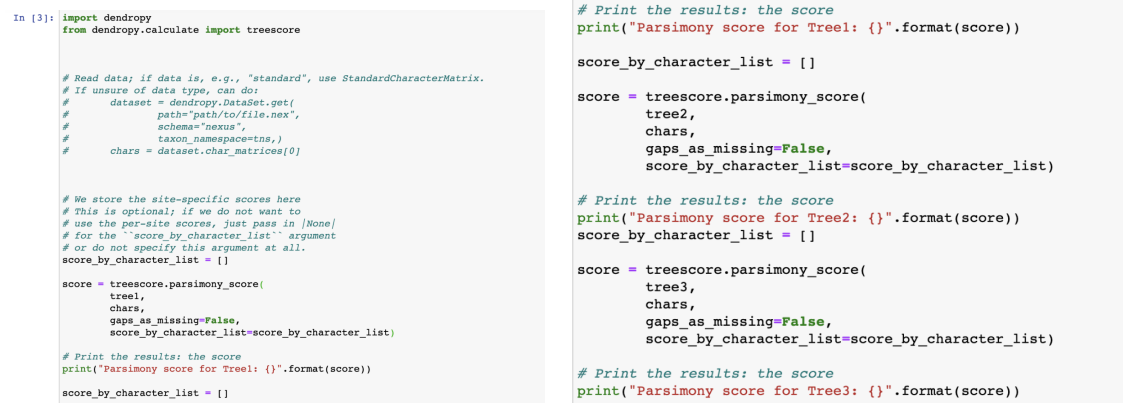
In [2]:
```
from dendropy.calculate import popgenstat

p1 = []
p2 = []
for idx, t in enumerate(chars.taxon_namespace):
    if t.label.startswith('Fregata'):
        p1.append(chars[t])
    else:
        p2.append(chars[t])
pp = popgenstat.PopulationPairSummaryStatistics(p1, p2)

print('Average number of pairwise differences (total): %s' \
    % pp.average_number_of_pairwise_differences)
print('Average number of pairwise differences (between populations): %s' \
    % pp.average_number_of_pairwise_differences_between)
print('Average number of pairwise differences (within populations): %s' \
    % pp.average_number_of_pairwise_differences_within)
print('Average number of pairwise differences (net): %s' \
    % pp.average_number_of_pairwise_differences_net)
print('Number of segregating sites: %s' \
    % pp.num_segregating_sites)
print("Watterson's theta: %s" \
    % pp.wattersons_theta)
print("Wakeley's Psi: %s" \
    % pp.wakeleys_psi)
print("Tajima's D: %s" \
    % pp.tajimas_d)
```

Here we use the DNA character matrix we loaded in to do some population summary statistics between the two families of Pelicans. We separate the populations by type, Fregata vs non-fregata. We then run the popgenstat module command on both populations.

```
Average number of pairwise differences (total): 238.24175824175825
Average number of pairwise differences (between populations): 261.1777777777778
Average number of pairwise differences (within populations): 303.7
Average number of pairwise differences (net): -42.52222222222218
Number of segregating sites: 700
Watterson's theta: 220.11652776238597
Wakeley's Psi: 0.145992214810748
Tajima's D: 0.3729589691673767
```

Here are our results and interpretations of what we calculated.
238.21 = The average number of pairwise differences between every sequence across both populations.
261.17 = The average number of pairwise differences between every sequence between both populations.
303.7 = The average number of pairwise differences between every sequence within each population.
-42.52 = The net number of pairwise differences.
700 = Number of positions which show differences (polymorphisms) between related genes in a sequence alignment.
220.1 = Watterson's Theta uses the number of segregating sites and population size to describe genetic diversity. High for our two populations.
.145 = Wakely's Psi, couldn't find any information on this one, but hey! We calculated it!
.372 = Tajima's D is computed as the difference between two measures of genetic diversity: the mean number of pairwise differences and the number of segregating sites, each scaled so that they are expected to be the same in a neutrally evolving population of constant size.

5. Third Jupyter Cell: Parsimony Score of Trees



```
In [3]: import dendropy
from dendropy.calculate import treescore



# Read data; if data is, e.g., "standard", use StandardCharacterMatrix.
# If unsure of data type, can do:
#       dataset = dendropy.DataSet.get(
#               path="path/to/file.nex",
#               schema="nexus",
#               taxon_namespace=tns,)
#       chars = dataset.char_matrices[0]



# We store the site-specific scores here
# This is optional; if we do not want to
# use the per-site scores, just pass in |None|
# for the ``score_by_character_list`` argument
# or do not specify this argument at all.
score_by_character_list = []

score = treescore.parsimony_score(
        tree1,
        chars,
        gaps_as_missing=False,
        score_by_character_list=score_by_character_list)

# Print the results: the score
print("Parsimony score for Tree1: {}".format(score))

score_by_character_list = []
```

```
# Print the results: the score
print("Parsimony score for Tree1: {}".format(score))

score_by_character_list = []

score = treescore.parsimony_score(
        tree2,
        chars,
        gaps_as_missing=False,
        score_by_character_list=score_by_character_list)

# Print the results: the score
print("Parsimony score for Tree2: {}".format(score))
score_by_character_list = []

score = treescore.parsimony_score(
        tree3,
        chars,
        gaps_as_missing=False,
        score_by_character_list=score_by_character_list)

# Print the results: the score
print("Parsimony score for Tree3: {}".format(score))
```

In this very fat Jupyter cell, we calculate the Parsimony score of each of our three trees.

```
Parsimony score for Tree1: 1535
Parsimony score for Tree2: 1542
Parsimony score for Tree3: 1539
```

The tree with the lowest parsimony score is the most parsimonious tree. Therefore Tree1 is the tree with the lowest number of nucleotide substitutions. Not necessarily an indicator of the best tree though!

6. Fourth Jupyter Cell: Unary Tree Statistics and Metrics for Tree1

```
In [6]: print('B1 statistic for tree: '+ str(dendropy.calculate.treemeasure.B1(tree)))

        print('N_bar statistic for tree: '+ str(dendropy.calculate.treemeasure.N_bar(tree)))

        print('Colless tree imabalance for tree: '+ str(dendropy.calculate.treemeasure.colless_tree_imbalance(tree, normalize='

        B1 statistic for tree: 6.783333333333333
        N_bar statistic for tree: 5.214285714285714
        Colless tree imabalance for tree: 0.4230769230769231
```

In this Jupyter cell, we calculate some tree statistics for our most parsimonious tree, Tree1

6.78 = the reciprocal of the sum of the maximum number of nodes between each interior node and tip over all internal nodes excluding root.
5.21 = the average number of nodes above a terminal node.
.423 = the sum of differences of numbers of children in left and right subtrees over all internal nodes

7. Fifth Jupyter Cell: Comparing the three trees using the Robinsons-Foulds distance measure.

```
In [11]: from dendropy.calculate import treecompare
         tree1.encode_bipartitions()
         tree2.encode_bipartitions()
         tree3.encode_bipartitions()
         print('Unweighted Robinson-Foulds distance between Tree 1 and Tree 2: '+ str(treecompare.symmetric_difference(tree1, tr
         print('Unweighted Robinson-Foulds distance between Tree 1 and Tree 3: '+ str(treecompare.symmetric_difference(tree1, tr
         print('Unweighted Robinson-Foulds distance between Tree 2 and Tree 3: '+ str(treecompare.symmetric_difference(tree3, tr

         Unweighted Robinson-Foulds distance between Tree 1 and Tree 2: 6
         Unweighted Robinson-Foulds distance between Tree 1 and Tree 3: 3
         Unweighted Robinson-Foulds distance between Tree 2 and Tree 3: 3
```

Here we use the treecompare command from the dendropy.calculate module to compare the Robinsons-Foulds distance between the three trees. The most parsimonious tree, tree one has the highest total distance among the three trees.

8. Conclusion:

I really enjoyed figuring out how to work with this API. It forced me into some uncomfortable territory where in the beginning I was really unsure what I was doing with this data. Working on a project like this with no supervision really heightened my sense of what I could do and understand in a field that I had no prior background in. While I cannot really give an informed analysis on my results, beyond what I could glean off of Wikipedia, computing them was the entire battle.