In [1]:

```
from charcoal import utils
import sourmash
from sourmash.lca import taxlist, LineagePair
import collections

import plotly.graph_objects as go
```

In [2]:

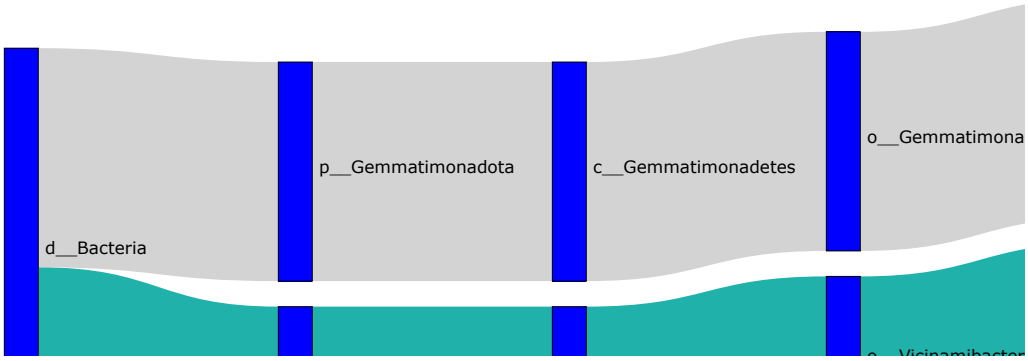
```
class GenomeSankeyFlow:
    def __init__(self):
        self.next_index = 0
        self.index_d = {}
        self.links_d = {}
        self.genus_lins = set()

        taxlist_pairs = []
        for rank in taxlist():
            if rank == 'superkingdom':
                last_rank = rank
                continue
            if rank == 'species':
                break
            taxlist_pairs.append((last_rank, rank))
            last_rank = rank
        self.taxlist_pairs = tuple(taxlist_pairs)

        unassigned_lin = []
        for rank in taxlist():
            if rank == 'species':
                break
            unassigned_lin.append(LineagePair(rank, 'unassigned'))

        self.unassigned_lin = tuple(unassigned_lin)

    def get_index(self, lin, rank=None):
        if rank:
            lin = utils.pop_to_rank(lineage, rank)

        lin = tuple(lin)
        if lin not in self.index_d:
            self.index_d[lin] = self.next_index
            self.next_index += 1
        return self.index_d[lin]

    def make_labels(self):
        linlist = list(self.index_d.items())
        linlist.sort(key = lambda x: x[1])
        return [ lin[-1].name for lin, idx in linlist ]

    def add_link(self, lin, count, src_rank, dest_rank, color):
        src_lin = utils.pop_to_rank(lin, src_rank)
        dest_lin = utils.pop_to_rank(lin, dest_rank)

        dest = self.get_index(dest_lin)
        src = self.get_index(src_lin)
        d1 = self.links_d.get(src, {})
        (prev_color, total_count) = d1.get(dest, (color, 0))
        total_count += count

        assert color == prev_color, (color, prev_color)

        d1[dest] = (color, total_count)
        self.links_d[src] = d1

    def process_contigs(self, contigs_info):
        counts = collections.Counter()
        for contig_name, gather_info in contigs_info.items():
            contig_taxlist = gather_info.gather_tax

            # note: contig_taxlist may be empty here. handle?

            # iterate over each contig match and summarize counts.
```

```
            # note - here we can stop at first one, or track them all.
            # note - b/c gather counts each hash only once, these are non-overlapp
ing
            total_hashcount = 0
            for lin, hashcount in contig_taxlist:
                self.genus_lins.add(lin)

                counts[lin] += hashcount
                total_hashcount += hashcount

            unident = gather_info.num_hashes - total_hashcount
            counts[self.unassigned_lin] += unident

        return counts

    def make_links(self, genome_lineage, counts, show_unassigned=False):
        # collect the set of lineages to display - by default, all.
        # note: could add a filter function to focus in on a specific 'un
        genus_lins = set(self.genus_lins)
        if show_unassigned:
            genus_lins.add(self.unassigned_lin)

        for lin in genus_lins:
            count = counts[lin]
            for last_rank, rank in self.taxlist_pairs:
                rank_lin = utils.pop_to_rank(lin, rank)

                color = "lightgrey"
                if utils.is_lineage_match(genome_lineage, lin, rank):
                    color = "lightseagreen"
                self.add_link(lin, count, last_rank, rank, color)
                last_rank = rank

    def make_lists(self):
        src_l = []
        dest_l = []
        cnt_l = []
        color_l = []
        label_l = []

        sum_counts = 0
        for k in sorted(self.links_d):
            for j in sorted(self.links_d[k]):
                sum_counts += self.links_d[k][j][1]

        for k in sorted(self.links_d):
            for j in sorted(self.links_d[k]):
                src_l.append(k)
                dest_l.append(j)

                color, counts = self.links_d[k][j]
                color_l.append(color)
                cnt_l.append(counts)

                pcnt = counts / sum_counts * 100
                label_l.append(f'{pcnt:.1f}% of total k-mers')

        return src_l, dest_l, cnt_l, color_l, label_l

    def make_plotly_fig(self, genome_lineage, contigs_info, title=None):
        counts = self.process_contigs(contigs_info)
        self.make_links(genome_lineage, counts)
        labels = self.make_labels()
        src_l, dest_l, cnt_l, color_l, label_l = self.make_lists()
```

```
        fig = go.Figure(data=[go.Sankey(
            node = dict(
              pad = 15,
              thickness = 20,
              line = dict(color = "black", width = 0.5),
              label = labels,
              color = "blue"
            ),
            link = dict(
              source = src_l, # indices correspond to labels, eg A1, A2, A2, B1,
...
              target = dest_l,
              value = cnt_l,
              color = color_l,
              label = label_l,
          ))])

        if title:
            fig.update_layout(title_text=title, font_size=10)
        else:
            fig.update_layout(title_text=f"genome lin: {sourmash.lca.display_linea
ge(genome_lineage)}", font_size=10)
        return fig

def load_and_make_fig(dirname, genome_name):
    contigs_filename = f'{dirname}/{genome_name}.contigs-tax.json'
    contigs_info = utils.load_contigs_gather_json(contigs_filename)

    hit_list_filename = f'{dirname}/hit_list_for_filtering.csv'
    hit_list = utils.HitList(hit_list_filename)

    row = hit_list[genome_name]
    genome_lineage = utils.make_lineage(row['lineage'])

    ####

    obj = GenomeSankeyFlow()
    fig = obj.make_plotly_fig(genome_lineage, contigs_info)

    return fig
```

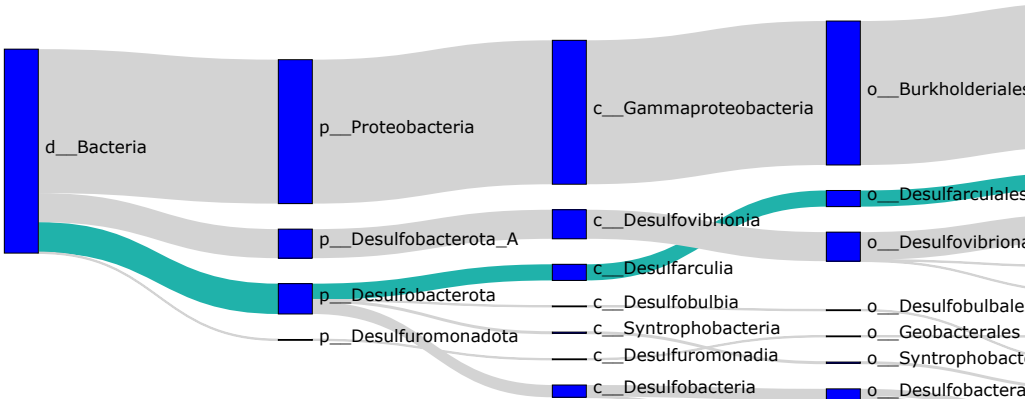
In [3]:
```
fig = load_and_make_fig('output.gtdb-contam-dna', 'GCA_003222535.1_genomic.fna.gz
')
fig.show()
```

genome lin: d__Bacteria;p__Acidobacteriota;c__Vicinamibacteria;o__Vicinamibacterales;



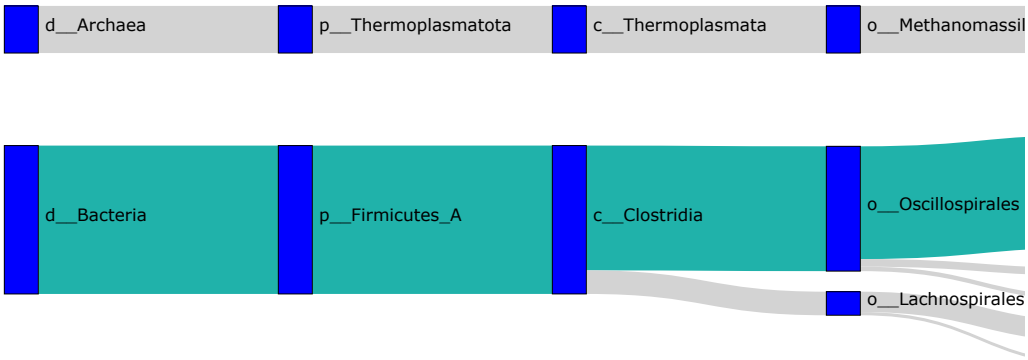
In [4]:
```
fig = load_and_make_fig('output.gtdb-contam-dna', 'GCF_001184205.1_genomic.fna.gz
')
fig.show()
```

genome lin: d__Bacteria;p__Desulfobacterota;c__Desulfarculia;o__Desulfarculales;f__De



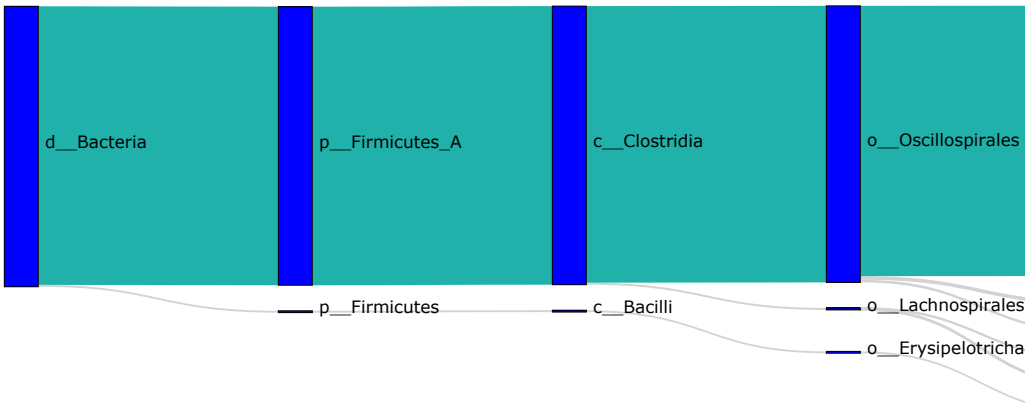
In [5]:
```
fig = load_and_make_fig('output.gtdb-contam-dna', 'GCF_000492175.1_genomic.fna.gz
')
fig.show()
```

genome lin: d__Bacteria;p__Firmicutes_A;c__Clostridia;o__Oscillospirales;f__Oscillospira



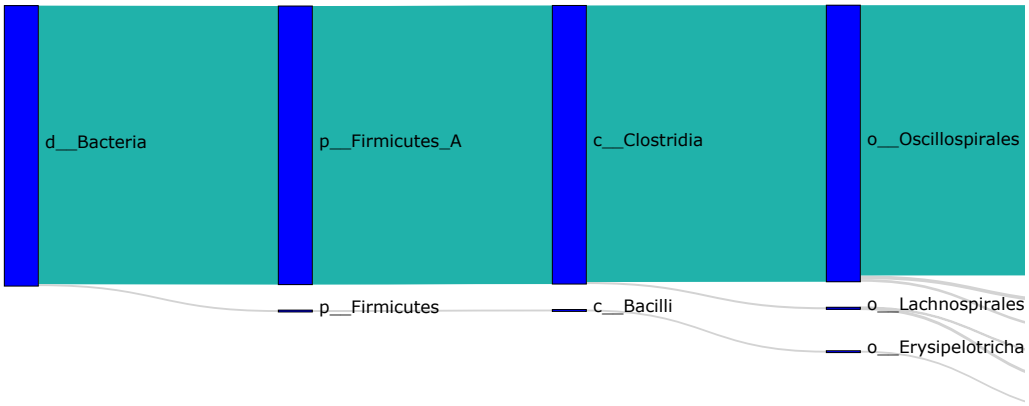
In [6]:
```
fig = load_and_make_fig('output.demo', 'LoombaR_2017__SID1050_bax__bin.11.fa.gz')
fig.show()
```

genome lin: d__Bacteria;p__Firmicutes_A;c__Clostridia;o__Oscillospirales;f__Acutalibact



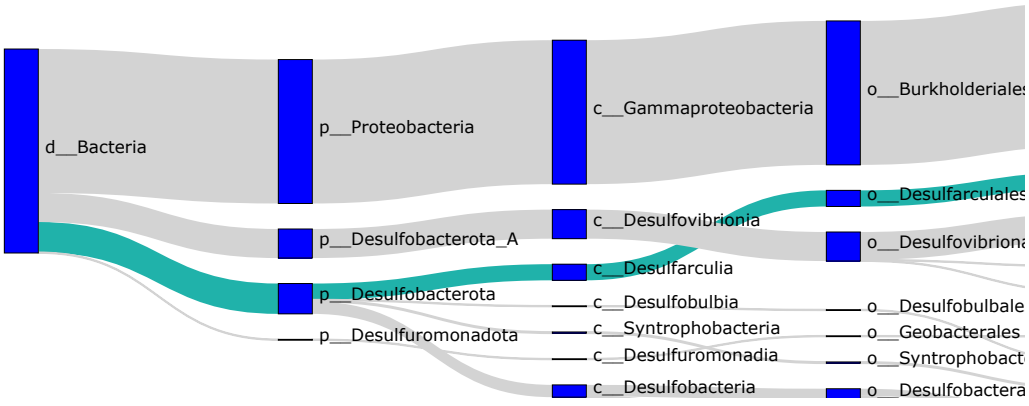
In [7]:
```
def load_all(dirname):
    hit_list_filename = f'{dirname}/hit_list_for_filtering.csv'
    hit_list = utils.HitList(hit_list_filename)

    for genome_name in hit_list.rows:
        if int(hit_list[genome_name]['total_bad_bp']):
            contigs_filename = f'{dirname}/{genome_name}.contigs-tax.json'
            contigs_info = utils.load_contigs_gather_json(contigs_filename)

            row = hit_list[genome_name]
            genome_lineage = utils.make_lineage(row['lineage'])

            obj = GenomeSankeyFlow()
            fig = obj.make_plotly_fig(genome_lineage, contigs_info, title=genome_n
ame)
            fig.show()

load_all('output.demo')
```

LoombaR_2017__SID1050_bax__bin.11.fa.gz

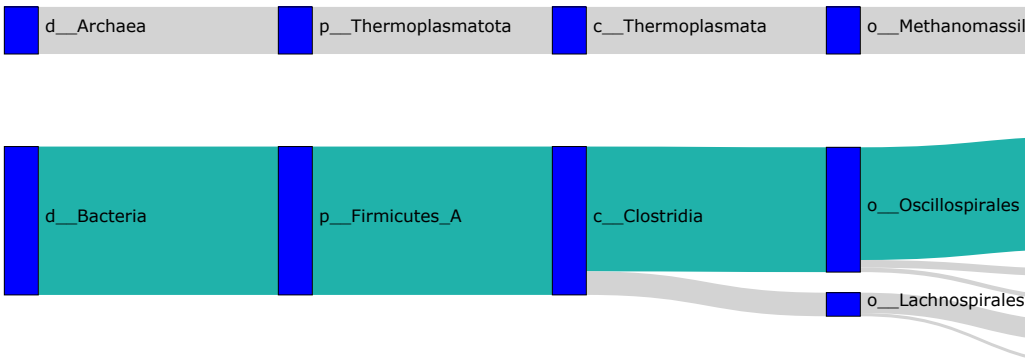


In [8]:
```
load_all('output.gtdb-contam-dna')
```

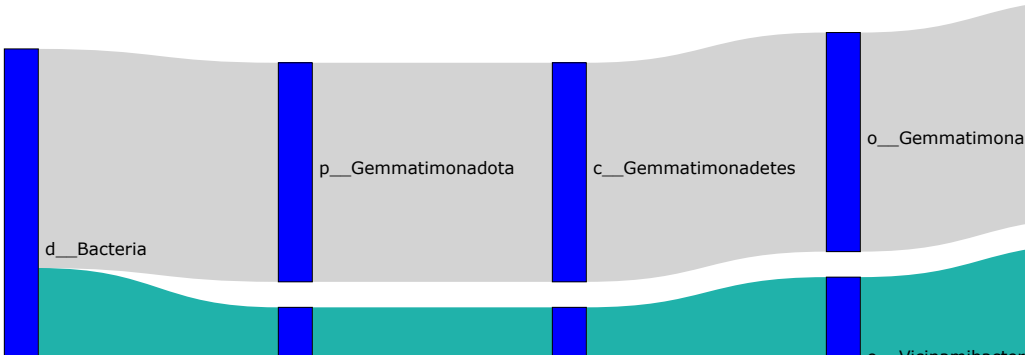
GCF_001184205.1_genomic.fna.gz



GCF_000492175.1_genomic.fna.gz

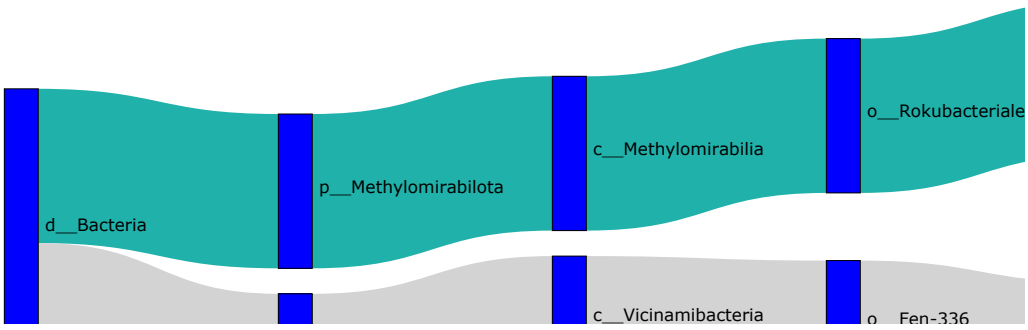


GCA_003222535.1_genomic.fna.gz



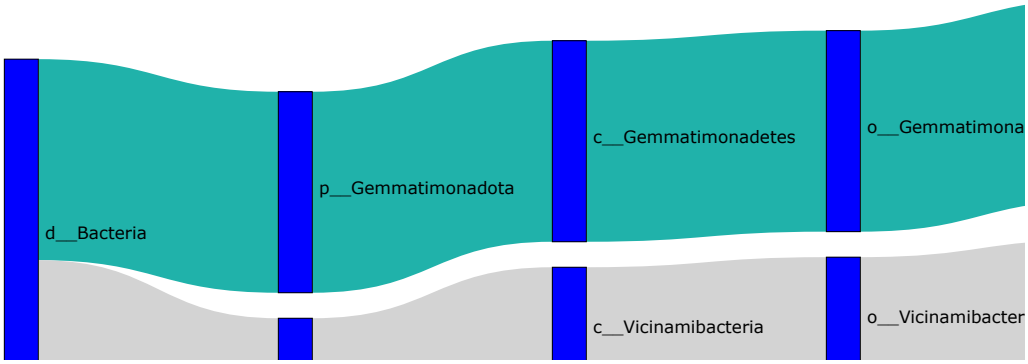
GCF_001672295.1_genomic.fna.gz

GCA_003220225.1_genomic.fna.gz



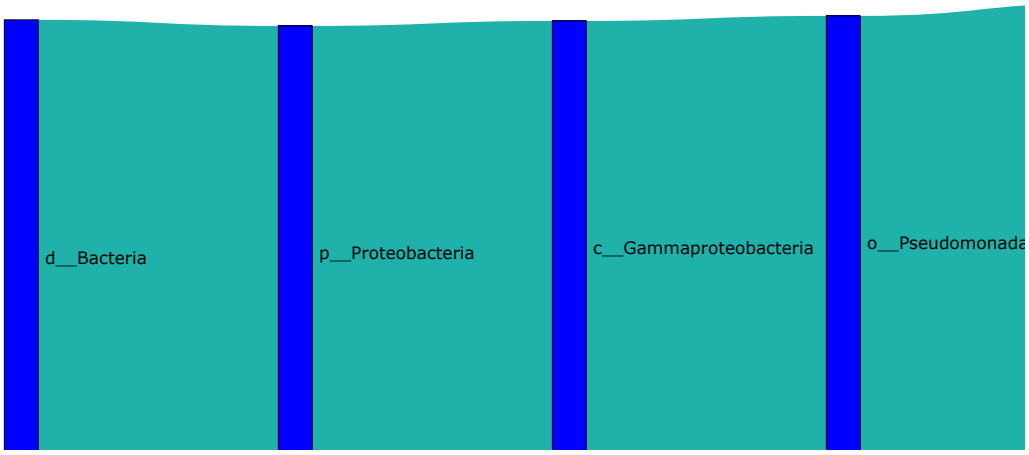
GCF_001683825.1_genomic.fna.gz

GCA_003221985.1_genomic.fna.gz



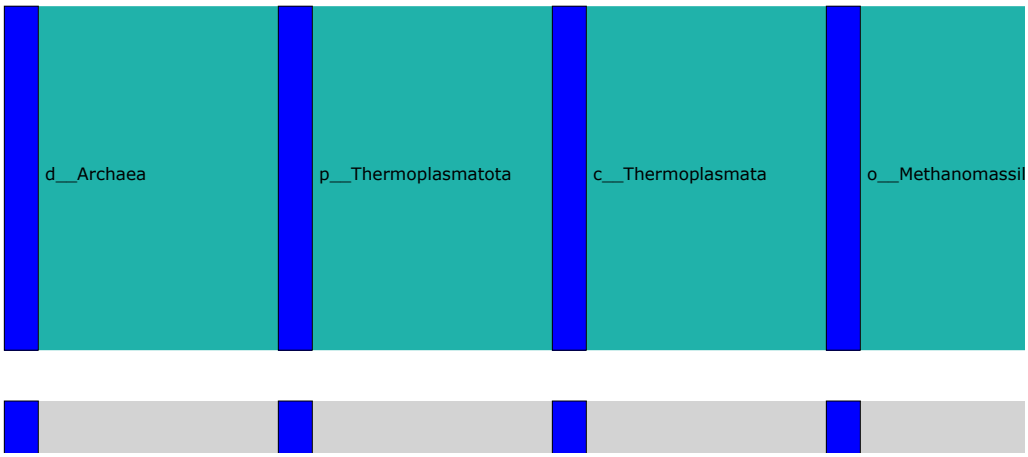
GCF_001408335.1_genomic.fna.gz

GCF_900016235.2_genomic.fna.gz



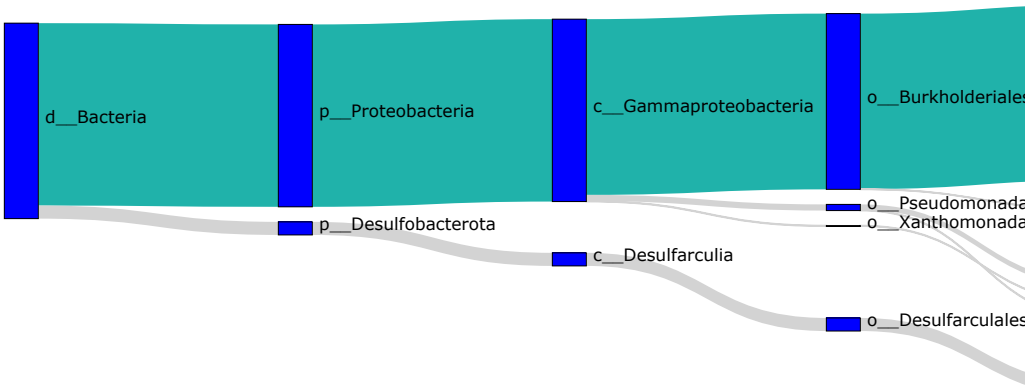
GCF_001749745.1_genomic.fna.gz

GCA_001421185.1_genomic.fna.gz

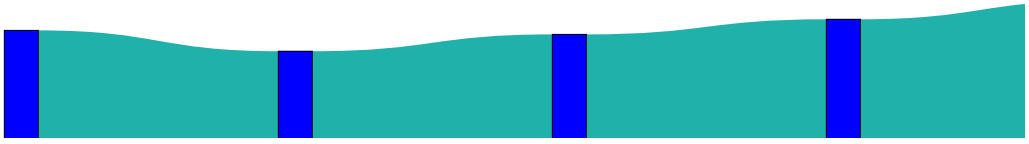


GCF_000763125.1_genomic.fna.gz

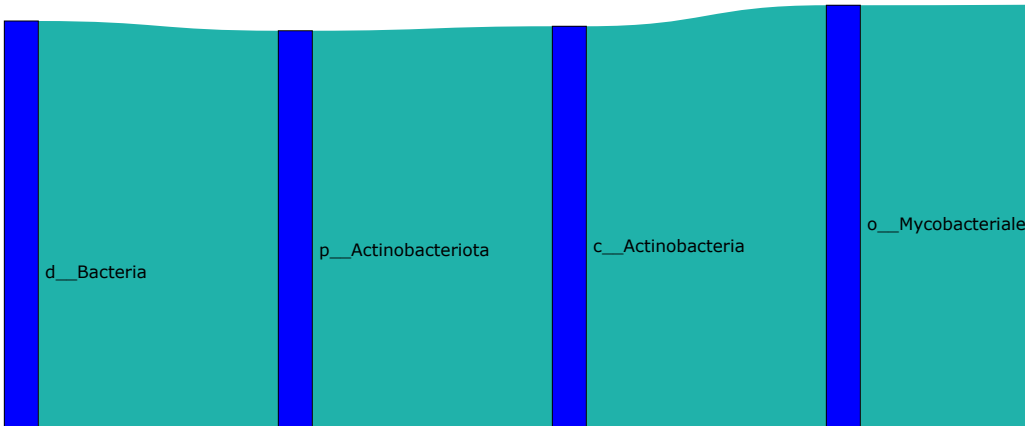
GCF_001078575.1_genomic.fna.gz



GCF_002901805.1_genomic.fna.gz



GCF_002154655.1_genomic.fna.gz



In [ ]: