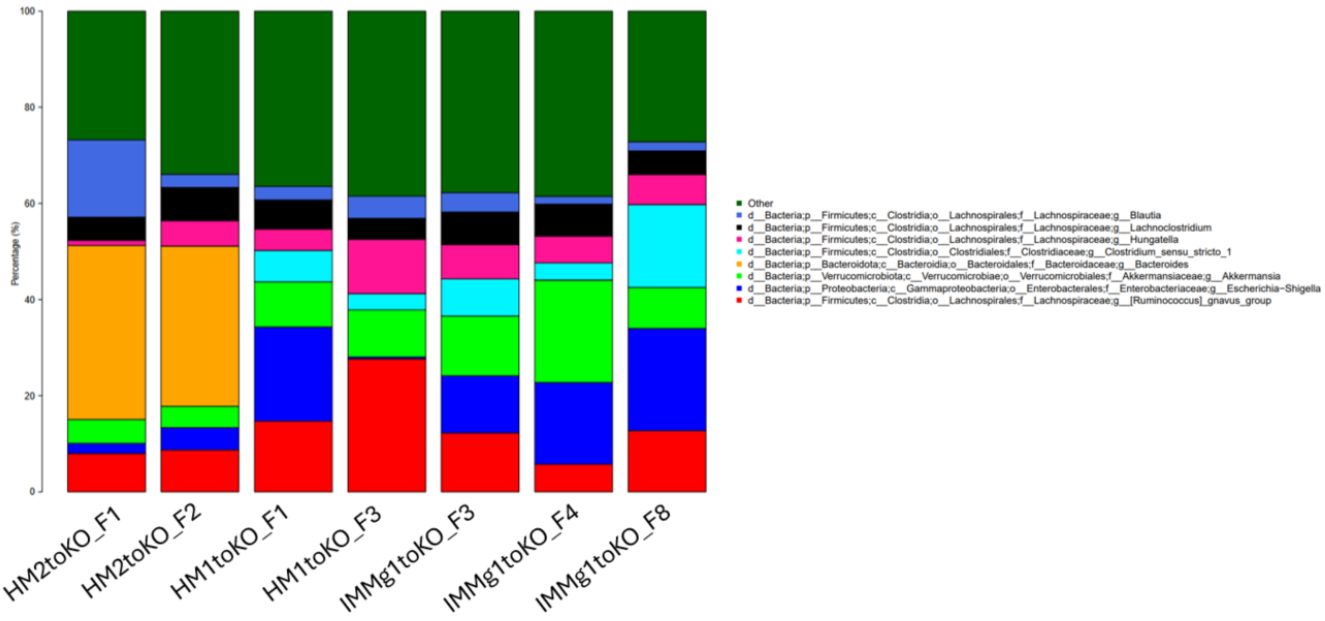
Contents

Aim: Replicate figure



Main Python Methods

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import os

# ****************************** MAIN METHODS ******************************
# Retrieve tab delimited file
def read_csv_file(file_path, skiprows=None, header = 0, sep = '\t', index_col=
False):
    df = pd.read_csv(file_path, skiprows=skiprows, sep=sep, header=header, ind
ex_col=index_col)

    return df
```

```
# Reference:  Function "get_cov_ellipse" was extracted from the following URL:

# https://scipython.com/book/chapter-7-matplotlib/examples/bmi-data-with-confi
dence-ellipses/
# (Learning Scientific Programming with Python by Christian Hill)
def get_cov_ellipse(cov, centre, nstd, **kwargs):
    """
    Return a matplotlib Ellipse patch representing the covariance matrix
    cov centred at centre and scaled by the factor nstd.
    """
    # Find and sort eigenvalues and eigenvectors into descending order
    eigvals, eigvecs = np.linalg.eigh(cov)
    order = eigvals.argsort()[::-1]
    eigvals, eigvecs = eigvals[order], eigvecs[:, order]

    # The anti-clockwise angle to rotate our ellipse by
    vx, vy = eigvecs[:,0][0], eigvecs[:,0][1]
    theta = np.arctan2(vy, vx)

    # Width and height of ellipse to draw
    width, height = 2 * nstd * np.sqrt(eigvals)
    return Ellipse(xy=centre, width=width, height=height,
                   angle=np.degrees(theta), fill=False, **kwargs)

# ****************************** INPUT FILE PATHS ************************
****
# Get current working directory
current_working_dir = os.getcwd()
# original counts table file path
feature_tbl_file_path = os.path.join(current_working_dir, 'feature-table_balfo
ur.txt')
# dictionary for two sample groupings
dict_metadata = {'129.IL10KO_1':['F1-1', 'F1-2', 'F1-3', 'F1-4', 'F1-5', 'F1-
6'],
                 '129.IL10KO_2':['F2-1' ,'F2-2', 'F2-3', 'F2-4', 'F2-5'],
                'hFMT.1.2.3.gtKO_1':['F1-7', 'F1-8', 'F1-9', 'F1-10', 'F1-11',
 'F1-12', 'F1-13', 'F1-14', 'F1-15'],
                 'hFMT.1.2.3.gtKO_2':['F3-7', 'F3-8', 'F3-9', 'F3-10', 'F3-11'
, 'F3-12'],
                '1gKOgtKO_1':['F3-1', 'F3-2', 'F3-4', 'F3-5', 'F3-6'],
                 '1gKOgtKO_2':['F4-1', 'F4-2', 'F4-3', 'F4-4', 'F4-5'],
                '1gKOgtKO_3':['F8-1', 'F8-2', 'F8-3', 'F8-4', 'F8-5', 'F8-6',
'F8-7', 'F8-8']}
# cross reference for sample grouping names
cross_ref_dict = {'129.IL10KO_1':'HM2toKO_F1',
                  '129.IL10KO_2':'HM2toKO_F2',
                  'hFMT.1.2.3.gtKO_1':'HM1toKO_F1',
                  'hFMT.1.2.3.gtKO_2':'HM1toKO_F3',
                 '1gKOgtKO_1':'IMMg1toKO_F3',
                  '1gKOgtKO_2':'IMMg1toKO_F4',
                  '1gKOgtKO_3':'IMMg1toKO_F8'}
# stacked bar plot colors
colors = ["red","blue","lightgreen","orange","turquoise","deeppink","black","r
oyalblue","darkgreen"]
# Set parameter for top N most abundant taxa across all sample groups
top_most_abundant_count = 8
```

Data Processing Steps

Step 1: Retrieve source feature counts table

In [2]:
```
df_feature_counts = read_csv_file(feature_tbl_file_path, 1)
df_feature_counts = df_feature_counts.astype({col:'int32' for col in df_featur
e_counts.columns[1:]})
df_feature_counts
```

Out[2]:

	#OTU ID	1gKO.1	1gKO.2	1gKO.3	1gWT.1	1gWT.2	1gWT.3
0	d__Bacteria;p__Verrucomicrobiota;c__Verrucomic...	11370	14120	14648	16632	19817	21706
1	d__Bacteria;p__Firmicutes;c__Clostridia;o__Lac...	22545	25836	17048	7547	9272	5996
2	d__Bacteria;p__Proteobacteria;c__Gammaproteoba...	5919	6481	6714	31	31	30
3	d__Bacteria;p__Firmicutes;c__Clostridia;o__Lac...	8134	9827	7090	8264	10100	6324
4	d__Bacteria;p__Firmicutes;c__Clostridia;o__Lac...	6280	7649	5719	9715	11851	8047
...	...	...	...	...	...	...	...
191	d__Bacteria;p__Firmicutes;c__Clostridia;o__Pep...	0	0	0	0	0	0
192	d__Bacteria;p__Firmicutes;c__Clostridia;o__Clo...	0	0	0	0	0	0
193	d__Bacteria;p__Firmicutes;c__Clostridia;o__Chr...	0	0	0	0	0	0
194	d__Bacteria;p__Firmicutes;c__Incertae_Sedis;o_...	0	0	0	0	0	0
195	d__Bacteria;p__Firmicutes;c__Bacilli;o__Lactob...	0	0	0	0	0	0

196 rows × 111 columns

Step 2: Filter feature counts table with seven target sample group columns

In [3]:
```
target_samples = [ sample for sample_group in list(dict_metadata.values()) for
 sample in sample_group ]
df_feature_counts_filtered = df_feature_counts[target_samples]
df_feature_counts_filtered
```

Out[3]:

	F1-1	F1-2	F1-3	F1-4	F1-5	F1-6	F2-1	F2-2	F2-3	F2-4	...	F4-4	F4-5	F8-1	
0	10401	2753	7914	2398	6595	2313	8957	10961	344	460	...	22240	22298	2627	1
1	7133	7051	3523	26054	524	2105	10749	5732	14621	319	...	8085	4026	10750	
2	2132	5867	4116	54	87	284	2194	1425	3296	1071	...	19808	23320	30563	2
3	1965	394	779	828	929	1973	4084	39	2827	17547	...	3631	7010	1440	
4	4785	1982	3119	2274	41521	49604	3306	3350	3278	2247	...	1346	1347	56	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
191	0	0	0	0	0	0	0	0	0	0	...	0	0	0	
192	0	0	0	0	0	0	0	0	0	0	...	0	0	0	
193	0	0	0	0	0	0	0	0	0	0	...	0	0	0	
194	0	0	0	0	0	0	0	0	0	0	...	0	0	0	
195	0	0	0	0	0	0	0	0	0	0	...	0	0	0	

196 rows × 44 columns

Step 3: Calculate total read counts per sample

In [4]:
```
sample_count_sums = df_feature_counts_filtered.sum(axis=0)
sample_count_sums
```

Out[4]:
```
F1-1     120620
F1-2      89533
F1-3     103243
F1-4      92643
F1-5     120934
F1-6      98057
F2-1      98214
F2-2      90013
F2-3      95932
F2-4     111659
F2-5      87600
F1-7     103534
F1-8     106286
F1-9      95787
F1-10    106356
F1-11     61083
F1-12    102329
F1-13    106433
F1-14    101965
F1-15     87272
F3-7      99046
F3-8     112057
F3-9     121801
F3-10     98907
F3-11     93879
F3-12    100855
F3-1      89907
F3-2      95182
F3-4     110663
F3-5     122538
F3-6     107733
F4-1      93069
F4-2     102268
F4-3     106452
F4-4     102778
F4-5     108196
F8-1     101846
F8-2     103946
F8-3      99383
F8-4     108840
F8-5      19323
F8-6      96062
F8-7     102664
F8-8      84853
dtype: int64
```

Step 4: Generate normalized counts feature table (without log10)

In [5]:
```
df_feature_counts_filtered_norm = (df_feature_counts_filtered/sample_count_sum
s)*sample_count_sums.mean()
```

```
df_feature_counts_filtered_norm
```

Out[5]:

	F1-1	F1-2	F1-3	F1-4	F1-5	F1-6	F2-1	
0	8547.969608	3048.107446	7598.758242	2565.923864	5405.969352	2338.321527	9040.581637	1
1	5862.192790	7806.830948	3382.666829	27878.473871	429.526602	2128.044450	10849.303564	
2	1752.165292	6495.912235	3952.045606	57.781438	71.314531	287.109085	2214.473162	
3	1614.917823	436.234774	747.969759	885.982051	761.508041	1994.599383	4122.109569	
4	3932.509814	2194.460210	2994.759534	2433.240561	34035.064968	50147.038912	3336.849715	
...	...	...	...	...	...	...	...	
191	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
192	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
193	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
194	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
195	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	

196 rows × 44 columns

Step 5: Identify top eight most abundant taxa

In [6]:
```
df_feature_counts_filtered_norm.index = df_feature_counts['#OTU ID']
df_top_eight_genera = pd.DataFrame(df_feature_counts_filtered_norm.mean(axis=1
).\
                                   sort_values(ascending=False)[:top_most_abun
dant_count])
df_top_eight_genera.columns = ['norm_mean_abundance']
df_top_eight_genera
```

Out[6]:

	#(
d__Bacteria;p__Firmicutes;c__Clostridia;o__Lachnospirales;f__Lachnospiraceae;g__[Ruminococcus]_gnavus_	
d__Bacteria;p__Proteobacteria;c__Gammaproteobacteria;o__Enterobacterales;f__Enterobacteriaceae;g__Esche S	
d__Bacteria;p__Verrucomicrobiota;c__Verrucomicrobiae;o__Verrucomicrobiales;f__Akkermansiaceae;g__Akkerr	
d__Bacteria;p__Bacteroidota;c__Bacteroidia;o__Bacteroidales;f__Bacteroidaceae;g__Bacte	
d__Bacteria;p__Firmicutes;c__Clostridia;o__Clostridiales;f__Clostridiaceae;g__Clostridium_sensu_st	
d__Bacteria;p__Firmicutes;c__Clostridia;o__Lachnospirales;f__Lachnospiraceae;g__Hun	
d__Bacteria;p__Firmicutes;c__Clostridia;o__Lachnospirales;f__Lachnospiraceae;g__Lachnoclost	
d__Bacteria;p__Firmicutes;c__Clostridia;o__Lachnospirales;f__Lachnospiraceae;g__	

Step 6: Calculate proportional taxa abundance for each sample group

In [7]:
```
# Initialize dictionary to store proportional abundances for each sample group
taxa_dict = {}
```

```
for taxa in df_top_eight_genera.index:
    taxa_dict[taxa] = []
taxa_dict['Other'] = []
sample_group_list = []

# Populate dictionary for sample groups
for sample_group in dict_metadata:
    cross_ref_dict[sample_group]
    sample_group_list.append(cross_ref_dict[sample_group])
    # For a given sample group, calculate the sum of the normalized counts for
 each taxa
    ser = df_feature_counts_filtered_norm[dict_metadata[sample_group]].sum(axi
s=1)
    # For a given sample group, calculate the proportion for each taxa
    ser_prop = ser/ser.sum()
    # For a given sample group, verify that the sum of all taxa proportions is
 equal to zero
    assert np.isclose(ser_prop.sum(), 1.0)
    # Filter out top eight most abundant taxa previously identified in step 5
    ser_prop_top_eight =  ser_prop[df_top_eight_genera.index]
    # Calculate and store the sum of proportions for all other taxa
    # that are not in the set of the top eight taxa
    other = 1.0 - ser_prop_top_eight.sum()
    taxa_dict['Other'].append(other)
    # Store the proportions for the top eight most abundant taxa
    for item in ser_prop_top_eight.items():
        taxa_dict[item[0]].append(item[1])

# Save data to a dataframe to plot during the following step
df_taxa = pd.DataFrame(taxa_dict)
df_taxa.index = sample_group_list
# Multiply by 100 to convert to percentage
df_taxa = df_taxa * 100
df_taxa
```

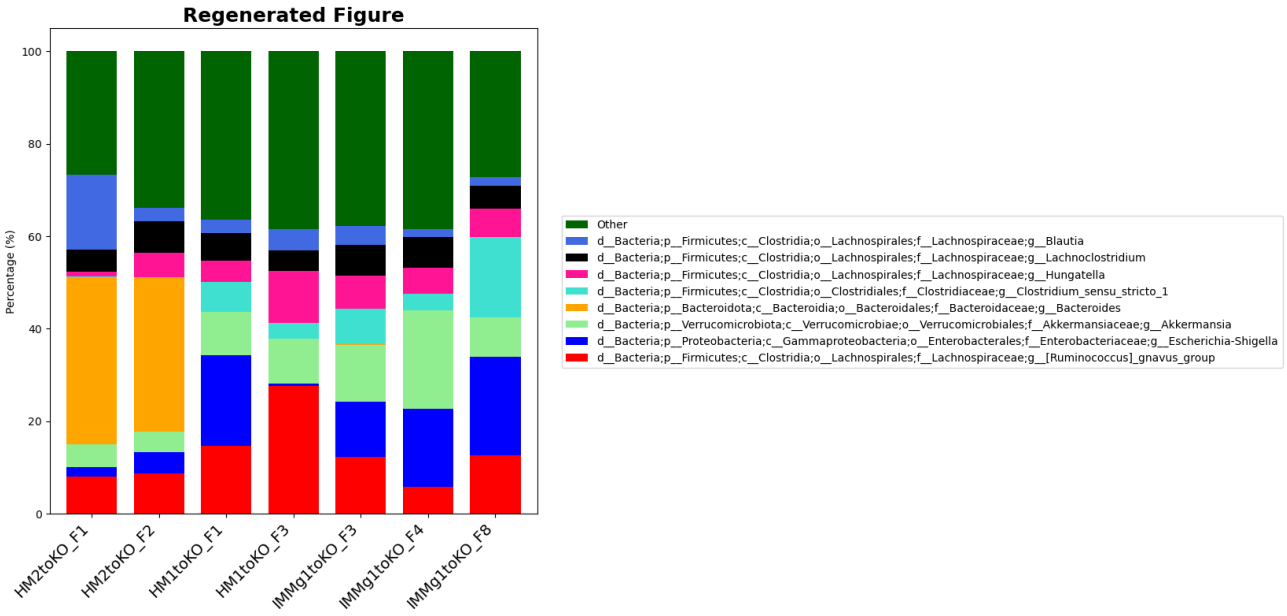
Out[7]:

	d__Bacteria;p__Firmicutes;c__Clostridia;o__Lachnospirales;f__Lachnospiraceae;g__[Ruminoc
HM2toKO_F1	
HM2toKO_F2	
HM1toKO_F1	
HM1toKO_F3	
IMMg1toKO_F3	
IMMg1toKO_F4	
IMMg1toKO_F8	

Step 7: Plot stacked bar plot results for each sample group

In [8]:
```
df_taxa.plot(kind='bar', stacked=True, color=colors, width=0.75, figsize=(8,8
))
plt.title('Regenerated Figure', fontsize=18, fontweight='bold')
plt.xticks(rotation=45, ha='right', fontsize=14)
plt.legend(loc=(1.05, 0.3), prop={'size': 10}, reverse=True)
```

```
plt.ylabel('Percentage (%)')
plt.show()
```



Step 8: Compare original and regenerated figures