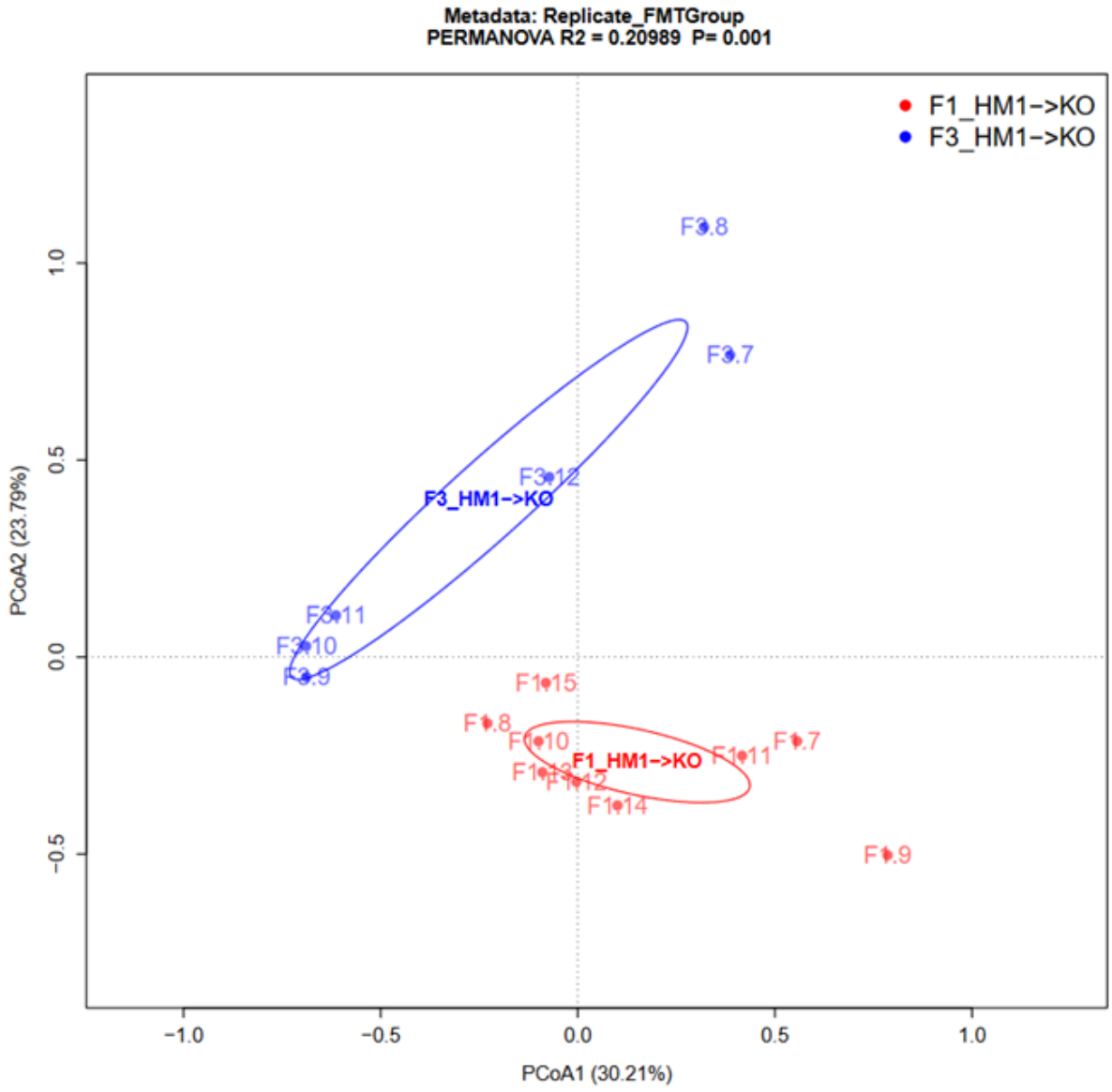
Contents

Aim: Replicate figure



Main Python Methods

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import os
import skbio
from scipy import stats
from scipy.spatial.distance import pdist, squareform
from matplotlib.patches import Ellipse
import matplotlib.transforms as transforms

# ****************************** 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 = {'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']}
# cross reference for sample grouping names
cross_ref_dict = {'hFMT.1.2.3.gtKO_1':'F1_HM1->KO', 'hFMT.1.2.3.gtKO_2':'F3_HM
2->KO'}
# criss referebce for sample names
```

```
sample_cross_ref_dict = {'F1-7':'F1.7', 'F1-8':'F1.8', 'F1-9':'F1.9', 'F1-10':
'F1.10',
                         'F1-11':'F1.11', 'F1-12':'F1.12', 'F1-13':'F1.13','F1
-14':'F1.14', 'F1-15':'F1.15',
                        'F3-7':'F3.7', 'F3-8':'F3.8', 'F3-9':'F3.9', 'F3-10':
'F3.10', 'F3-11':'F3.11', 'F3-12':'F3.12'}
```

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 target sample 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]:

	F1-7	F1-8	F1-9	F1-10	F1-11	F1-12	F1-13	F1-14	F1-15	F3-7	F3-8	F3-9	F3-10	F3
0	8869	8035	5953	11460	10562	6795	6439	6312	13082	8445	319	10551	15735	14
1	19991	23011	12533	27427	4496	12442	22199	7442	4318	34435	43961	31682	20382	17
2	24450	10453	39194	12486	18793	13233	14613	32605	675	978	13	1555	34	
3	886	5723	480	6491	986	8890	10948	3644	2698	28326	10652	3300	3168	

4	655	2478	205	5444	384	3389	2020	1481	8495	3647	10380	3639	3886	3
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
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 × 15 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]:
```
F1-7     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
dtype: int64
```

Step 4: Generate normalized counts feature table

In [5]:
```
df_feature_counts_filtered_norm = np.log10((df_feature_counts_filtered/sample_
count_sums)*sample_count_sums.mean() + 1)
df_feature_counts_filtered_norm
```

Out[5]:

	F1-7	F1-8	F1-9	F1-10	F1-11	F1-12	F1-13	F1-14	F1-15	F3-
0	3.932144	3.877869	3.792801	4.031765	4.237153	3.821558	3.781116	3.791088	4.175130	3.93011
1	4.285076	4.334781	4.116083	4.410735	3.866267	4.084229	4.318580	3.862600	3.693797	4.54047
2	4.372516	3.992111	4.611225	4.069000	4.487392	4.110995	4.136996	4.504148	2.888293	2.99424
3	2.932160	3.730530	2.700104	3.784923	3.207528	3.938255	4.011602	3.552551	3.489609	4.45566
4	2.801147	3.367113	2.331777	3.708543	2.798404	3.519506	3.277806	3.161705	3.987635	3.56551
...	...	...	...	...	...	...	...	...	...	.
191	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.00000
192	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.00000
193	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.00000

194	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.00000
195	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.00000

196 rows × 15 columns

Step 5: Generate Bray-Curtis dissimilarity distance matrix

In [6]:
```
condensed_arr = pdist(df_feature_counts_filtered_norm.T, metric='braycurtis')
dist_mat_sym = squareform(condensed_arr)
print('Dimensions of symmetrical Bray-Curtris distance matrix: {}'.format(dist
_mat_sym.shape))
dist_mat_sym
```

```
Dimensions of symmetrical Bray-Curtris distance matrix: (15, 15)
```

Out[6]:
```
array([[0.        , 0.2354373 , 0.22808031, 0.22417016, 0.12012291,
        0.23853699, 0.24808563, 0.22766537, 0.28355288, 0.2810975 ,
        0.31556637, 0.36293577, 0.32842803, 0.31651299, 0.30864063],
       [0.2354373 , 0.        , 0.33151514, 0.14967141, 0.24991627,
        0.24001849, 0.21987228, 0.26156663, 0.27932507, 0.30678854,
        0.34797223, 0.23344469, 0.21036296, 0.20676359, 0.30008727],
       [0.22808031, 0.33151514, 0.        , 0.27522759, 0.23980917,
        0.30038263, 0.31849063, 0.27440435, 0.35510262, 0.33107932,
        0.40318452, 0.39587151, 0.40440186, 0.39791723, 0.37538382],
       [0.22417016, 0.14967141, 0.27522759, 0.        , 0.19583741,
        0.20769077, 0.19516   , 0.19688723, 0.26357676, 0.28888256,
        0.33578638, 0.22325478, 0.19780983, 0.21319542, 0.23524972],
       [0.12012291, 0.24991627, 0.23980917, 0.19583741, 0.        ,
        0.25426323, 0.26428184, 0.24114472, 0.28099155, 0.27165053,
        0.34444904, 0.3306451 , 0.30140723, 0.29320604, 0.31426941],
       [0.23853699, 0.24001849, 0.30038263, 0.20769077, 0.25426323,
        0.        , 0.07849586, 0.11983502, 0.17587685, 0.30023051,
        0.34731476, 0.27896171, 0.25896355, 0.26408819, 0.26365952],
       [0.24808563, 0.21987228, 0.31849063, 0.19516   , 0.26428184,
        0.07849586, 0.        , 0.12215787, 0.16324204, 0.2992072 ,
        0.34827216, 0.25906799, 0.24790667, 0.26258638, 0.25885452],
       [0.22766537, 0.26156663, 0.27440435, 0.19688723, 0.24114472,
        0.11983502, 0.12215787, 0.        , 0.19308055, 0.29340743,
        0.35498576, 0.28113353, 0.27029515, 0.28299124, 0.28161598],
       [0.28355288, 0.27932507, 0.35510262, 0.26357676, 0.28099155,
        0.17587685, 0.16324204, 0.19308055, 0.        , 0.30740146,
        0.32011341, 0.26906954, 0.28062787, 0.28590995, 0.29277325],
       [0.2810975 , 0.30678854, 0.33107932, 0.28888256, 0.27165053,
        0.30023051, 0.2992072 , 0.29340743, 0.30740146, 0.        ,
        0.17442066, 0.3409591 , 0.33767882, 0.32261034, 0.24950042],
       [0.31556637, 0.34797223, 0.40318452, 0.33578638, 0.34444904,
        0.34731476, 0.34827216, 0.35498576, 0.32011341, 0.17442066,
        0.        , 0.38938968, 0.3599006 , 0.33861661, 0.30290218],
       [0.36293577, 0.23344469, 0.39587151, 0.22325478, 0.3306451 ,
        0.27896171, 0.25906799, 0.28113353, 0.26906954, 0.3409591 ,
        0.38938968, 0.        , 0.17416425, 0.19202294, 0.29804967],
       [0.32842803, 0.21036296, 0.40440186, 0.19780983, 0.30140723,
        0.25896355, 0.24790667, 0.27029515, 0.28062787, 0.33767882,
        0.3599006 , 0.17416425, 0.        , 0.06813747, 0.27586168],
       [0.31651299, 0.20676359, 0.39791723, 0.21319542, 0.29320604,
        0.26408819, 0.26258638, 0.28299124, 0.28590995, 0.32261034,

        0.33861661, 0.19202294, 0.06813747, 0.        , 0.2958373 ],
```

```
[0.30864063, 0.30008727, 0.37538382, 0.23524972, 0.31426941,
 0.26365952, 0.25885452, 0.28161598, 0.29277325, 0.24950042,
 0.30290218, 0.29804967, 0.27586168, 0.2958373 , 0.        ]])
```

Step 6: Perform principal coordinate analysis (PCoA)

In [7]:
```
import warnings
warnings.filterwarnings('ignore')
my_pcoa = skbio.stats.ordination.pcoa(dist_mat_sym)
df_pcoa = my_pcoa.samples[['PC1', 'PC2']]
# Normalize PC1 and PC2 into unit vectors
df_pcoa = pd.DataFrame(df_pcoa.to_numpy()/np.linalg.norm(df_pcoa.to_numpy(), a
xis=0))
print('PCoA proportion explained:')
my_pcoa.proportion_explained
```

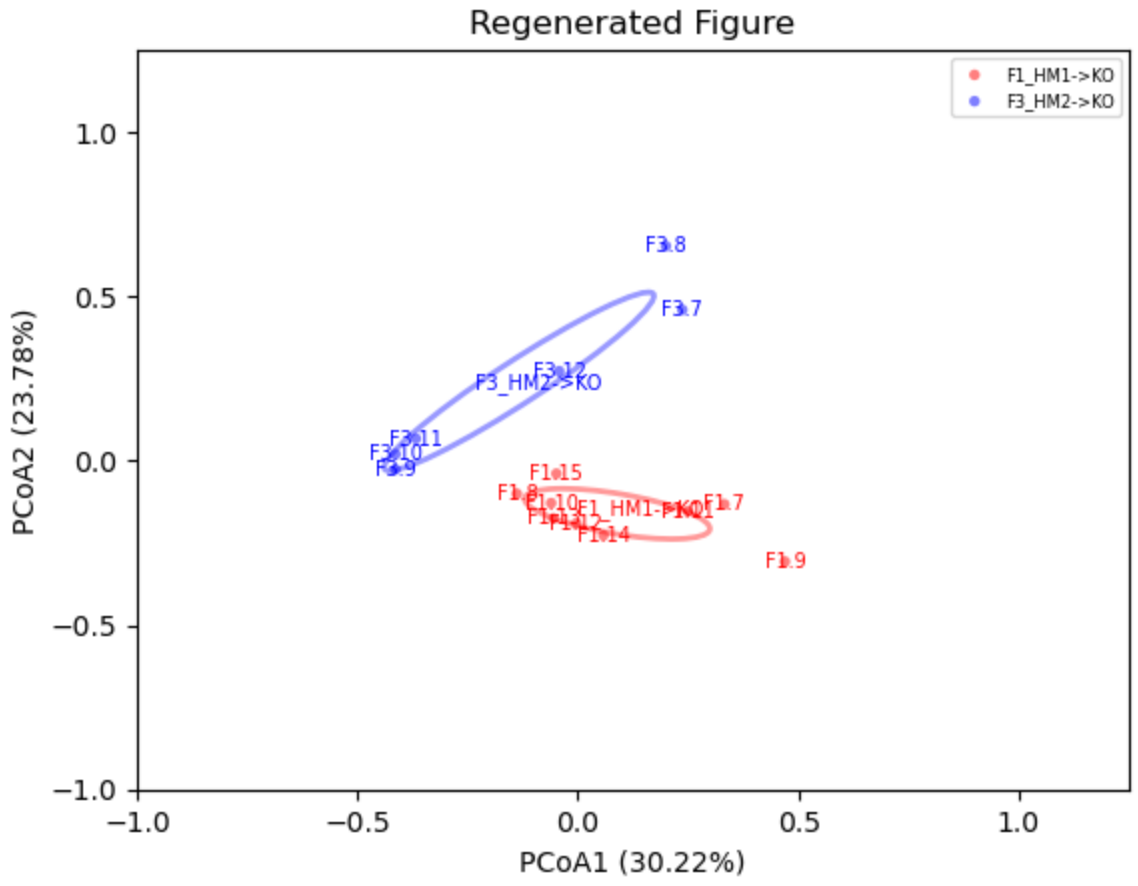
```
PCoA proportion explained:
```

Out[7]:
```
PC1     0.302200
PC2     0.237754
PC3     0.153340
PC4     0.084033
PC5     0.065666
PC6     0.045122
PC7     0.039731
PC8     0.028952
PC9     0.018222
PC10    0.012539
PC11    0.007886
PC12    0.003585
PC13    0.000970
PC14    0.000000
PC15    0.000000
dtype: float64
```

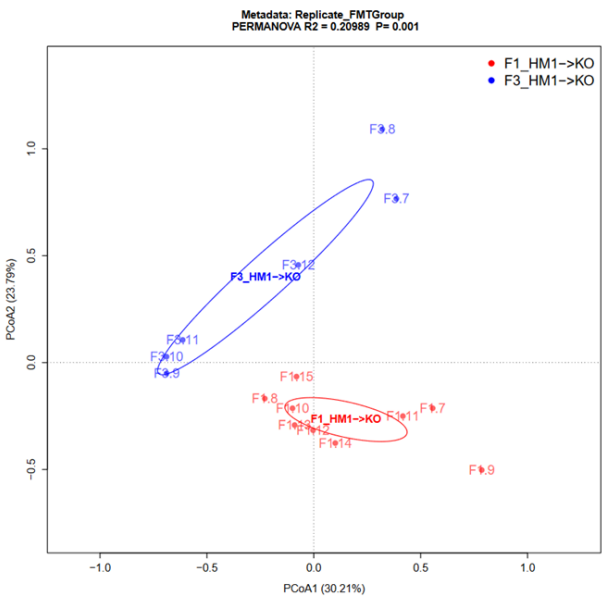
Step 7: Plot PCoA results

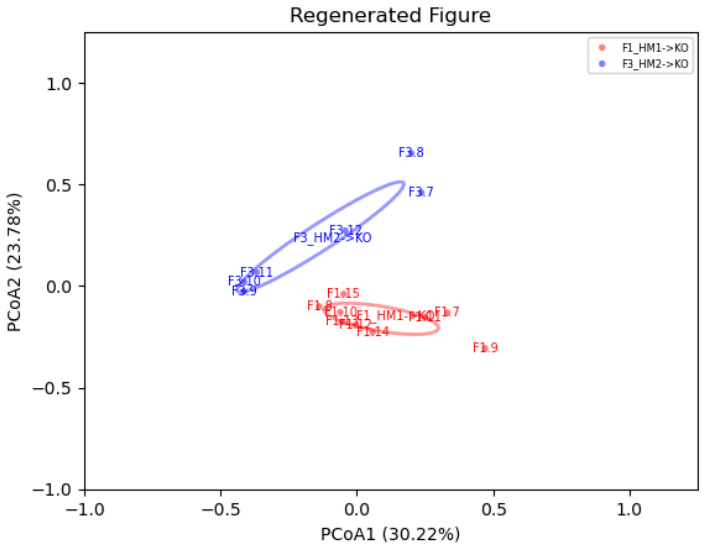
In [8]:
```
colors = ['red', 'blue']
text_tup = [(-0.09, 0.00),
            (-0.1, -0.02)]

fig, ax = plt.subplots()
ax.set_xlim((-1.0, 1.25))
ax.set_ylim((-1.0, 1.25))
ax.set_yticks([-1.0, -0.5, 0.0, 0.5, 1.0])
ax.set_xticks([-1.0, -0.5, 0.0, 0.5, 1.0])
ax.set_xlabel('PCoA1 ({}%)'.format(np.round(my_pcoa.proportion_explained.PC1 *
 100, 2)))
ax.set_ylabel('PCoA2 ({}%)'.format(np.round(my_pcoa.proportion_explained.PC2 *
 100, 2)))
ax.set_title('Regenerated Figure')

end = 0
for idx, group_name in enumerate(dict_metadata):
    sample_names = dict_metadata[group_name]
    group_name = cross_ref_dict[group_name]
```

```
    group_count = len(sample_names)
    start = end
    end = start + group_count

    # multiply by -1 to rotate vector 180 degrees in order to match figure
    pc1 = df_pcoa.iloc[start:end, 0] * -1.0
    pc2 = df_pcoa.iloc[start:end, 1]
    # plot points
    ax.scatter(pc1, pc2, s=15, c=colors[idx],  label=group_name,
               alpha=0.5, edgecolors='none')
    cov = np.cov(pc1,pc2)
    x_mean = pc1.mean()
    y_mean = pc2.mean()
    e = get_cov_ellipse(cov, (x_mean, y_mean), 1,
                        ec=colors[idx], linewidth=2.0, alpha=0.4)
    ax.text(x_mean + text_tup[idx][0], y_mean + text_tup[idx][1], group_name,
            fontsize='x-small', c=colors[idx])
    ax.add_artist(e)

    for i, sample_name in enumerate(sample_names):
        ax.text(pc1.iloc[i], pc2.iloc[i], sample_cross_ref_dict[sample_name],
                fontsize='x-small', c=colors[idx], horizontalalignment='cente
r',
                verticalalignment='center')
ax.legend(fontsize=6, loc='upper right')
plt.show()
```



Step 8: Compare original and regenerated figures


Metadata: Replicate_FMTGroup
PERMANOVA R2 = 0.20989 P= 0.001
F1_HM1->KO
F3_HM1->KO
F3_HM1->KO
F1_HM1->KO
PCoA1 (30.21%)
PCoA2 (23.79%)


Regenerated Figure
F1_HM1->KO
F3_HM2->KO
F3_HM2->KO
PCoA1 (30.22%)
PCoA2 (23.78%)