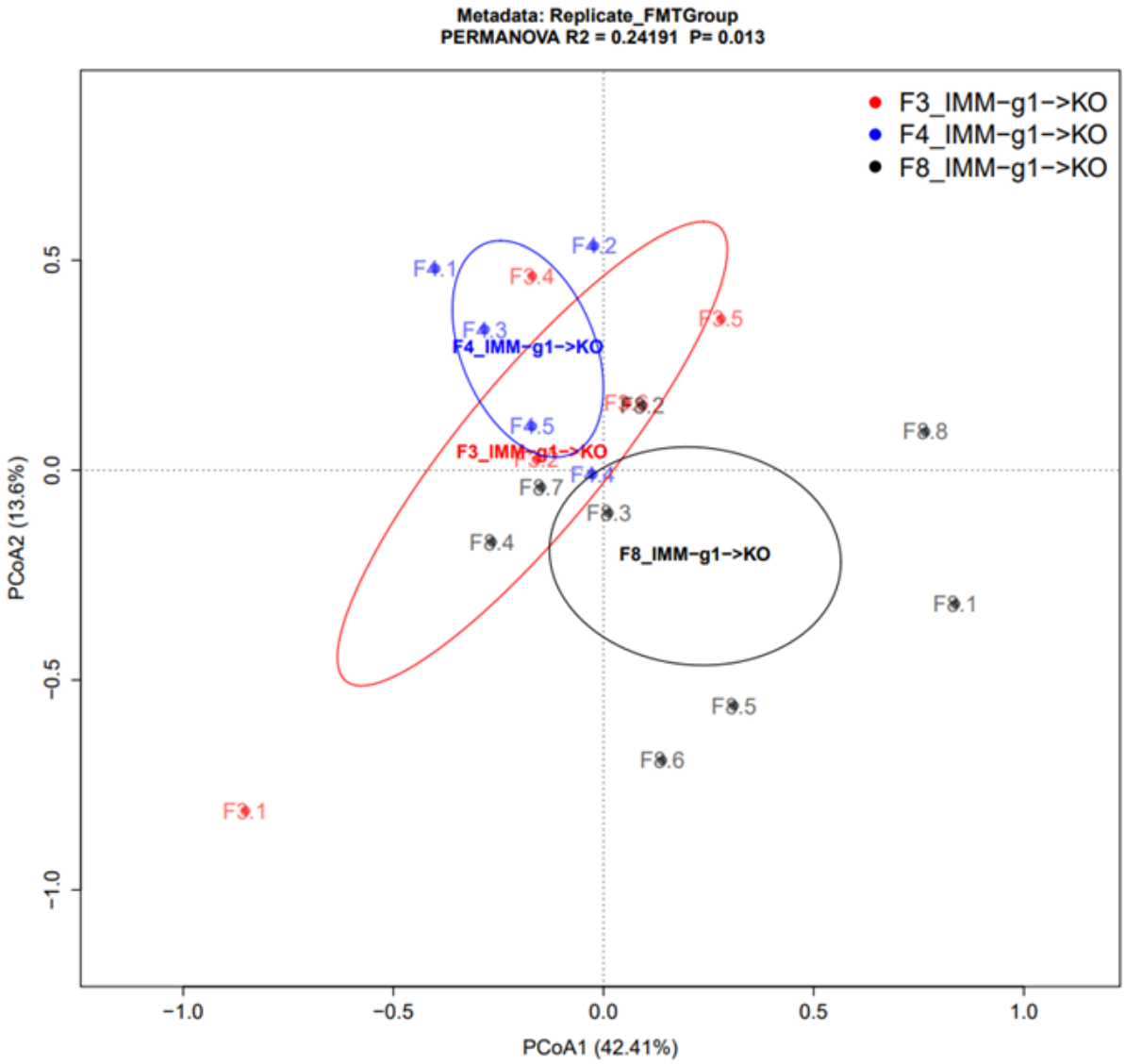
Contents

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 three sample groupings
dict_metadata = {'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 = {'1gKOgtKO_1':'F3_IMM-g1->KO', '1gKOgtKO_2':'F4_IMM-g1->KO',
'1gKOgtKO_3':'F8_IMM-g1->KO'}
# criss referebce for sample names
sample_cross_ref_dict = {'F3-1':'F3.1', 'F3-2':'F3.2', 'F3-4':'F3.4', 'F3-5':
'F3.5', 'F3-6':'F3.6',
```

```
                          'F4-1':'F4.1', 'F4-2':'F4.2', 'F4-3':'F4.3', 'F4-4':'F
4.4', 'F4-5':'F4.5',
                          'F8-1':'F8.1', 'F8-2':'F8.2', 'F8-3':'F8.3', 'F8-4':'F
8.4',
                           'F8-5':'F8.5', 'F8-6':'F8.6', 'F8-7':'F8.7', 'F8-8':
'F8.8'}
```

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 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]:

	F3-1	F3-2	F3-4	F3-5	F3-6	F4-1	F4-2	F4-3	F4-4	F4-5	F8-1	F8-2	F8-3	F
0	17605	8824	16088	9737	11518	24109	16317	23844	22240	22298	2627	12243	9359	15
1	292	25296	9605	2068	25952	3552	6985	6857	8085	4026	10750	7020	8879	13
2	40	2810	20375	38880	6836	1158	26515	18156	19808	23320	30563	28879	38922	8
3	1952	20829	4524	1738	6522	7172	2984	7713	3631	7010	1440	5601	5679	10

4	10538	2268	2403	1978	2505	2330	1026	2151	1346	1347	56	739	1149	2
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
191	0	0	2	0	0	0	0	0	0	0	0	0	0	
192	0	0	0	2	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 × 18 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]:
```
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

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]:

	F3-1	F3-2	F3-4	F3-5	F3-6	F4-1	F4-2	F4-3	F4-4	F4-
0	4.281044	3.956337	4.151709	3.889389	4.018249	4.402570	4.192107	4.339424	4.324434	4.30325
1	2.502137	4.413692	3.927725	3.216722	4.371019	3.570963	3.823670	3.798228	3.885015	3.55996
2	1.647339	3.459481	4.254298	4.490648	3.791702	3.084440	4.402948	4.221071	4.274142	4.32271
3	3.326070	4.329312	3.600808	3.141271	3.771284	3.876072	3.454390	3.849309	3.537431	3.80075
4	4.058181	3.366454	3.326135	3.197410	3.355829	3.387907	2.991030	3.294885	3.106663	3.08469
...	...	...	...	...	...	...	...	...	...	.
191	0.000000	0.000000	0.441351	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.00000

192	0.000000	0.000000	0.000000	0.413632	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 × 18 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: (18, 18)
```

Out[6]:
```
array([[0.        , 0.28454686, 0.29267021, 0.36808693, 0.32967506,
        0.2605009 , 0.29730312, 0.24727062, 0.28361299, 0.25890973,
        0.44438252, 0.29948291, 0.27046481, 0.21882052, 0.36697641,
        0.30325325, 0.25152437, 0.45398825],
       [0.28454686, 0.        , 0.15342963, 0.21946027, 0.1471099 ,
        0.20831996, 0.18695832, 0.18255075, 0.21447577, 0.19886628,
        0.33151212, 0.21464976, 0.17580187, 0.18108425, 0.2389772 ,
        0.19459264, 0.17578575, 0.29076401],
       [0.29267021, 0.15342963, 0.        , 0.14439185, 0.1778078 ,
        0.1458174 , 0.11984601, 0.11024685, 0.145016  , 0.12118633,
        0.32270016, 0.17400607, 0.16635111, 0.14562586, 0.22162812,
        0.20358452, 0.14600089, 0.28600799],
       [0.36808693, 0.21946027, 0.14439185, 0.        , 0.24023838,
        0.24230403, 0.18393303, 0.20344634, 0.19103171, 0.19224901,
        0.24405912, 0.18131433, 0.19226307, 0.2595383 , 0.19738032,
        0.21403131, 0.2074863 , 0.23233857],
       [0.32967506, 0.1471099 , 0.1778078 , 0.24023838, 0.        ,
        0.22602656, 0.18825317, 0.20670047, 0.22431224, 0.2196271 ,
        0.28836239, 0.19908431, 0.18851052, 0.19963634, 0.2392397 ,
        0.22310475, 0.20029594, 0.24800114],
       [0.2605009 , 0.20831996, 0.1458174 , 0.24230403, 0.22602656,
        0.        , 0.13228615, 0.08405766, 0.16189807, 0.15142551,
        0.36028806, 0.20185117, 0.17992468, 0.12857937, 0.28657616,
        0.2400629 , 0.14094868, 0.32119404],
       [0.29730312, 0.18695832, 0.11984601, 0.18393303, 0.18825317,
        0.13228615, 0.        , 0.0956604 , 0.141664  , 0.1371242 ,
        0.25628948, 0.13370686, 0.1419186 , 0.15473474, 0.26008824,
        0.22197027, 0.14452128, 0.22554503],
       [0.24727062, 0.18255075, 0.11024685, 0.20344634, 0.20670047,
        0.08405766, 0.0956604 , 0.        , 0.11693841, 0.10950776,
        0.3218014 , 0.17444426, 0.16195855, 0.125435  , 0.2403541 ,
        0.21527653, 0.13417896, 0.28984225],
       [0.28361299, 0.21447577, 0.145016  , 0.19103171, 0.22431224,
        0.16189807, 0.141664  , 0.11693841, 0.        , 0.09469814,
        0.26741369, 0.14436788, 0.13571197, 0.13232554, 0.19829445,
        0.15854515, 0.12310822, 0.23427458],
       [0.25890973, 0.19886628, 0.12118633, 0.19224901, 0.2196271 ,
        0.15142551, 0.1371242 , 0.10950776, 0.09469814, 0.        ,

        0.30136331, 0.13609655, 0.11308482, 0.11049238, 0.21923821,
```

```
 0.19349934, 0.11755898, 0.26633493],
[0.44438252, 0.33151212, 0.32270016, 0.24405912, 0.28836239,
 0.36028806, 0.25628948, 0.3218014 , 0.26741369, 0.30136331,
 0.        , 0.2350507 , 0.25292598, 0.3100213 , 0.25768962,
 0.24297859, 0.29501488, 0.11959038],
[0.29948291, 0.21464976, 0.17400607, 0.18131433, 0.19908431,
 0.20185117, 0.13370686, 0.17444426, 0.14436788, 0.13609655,
 0.2350507 , 0.        , 0.12171157, 0.15559515, 0.24067187,
 0.21444788, 0.13243287, 0.20929897],
[0.27046481, 0.17580187, 0.16635111, 0.19226307, 0.18851052,
 0.17992468, 0.1419186 , 0.16195855, 0.13571197, 0.11308482,
 0.25292598, 0.12171157, 0.        , 0.11402347, 0.20420303,
 0.18106997, 0.11758264, 0.20746761],
[0.21882052, 0.18108425, 0.14562586, 0.2595383 , 0.19963634,
 0.12857937, 0.15473474, 0.125435  , 0.13232554, 0.11049238,
 0.3100213 , 0.15559515, 0.11402347, 0.        , 0.2390292 ,
 0.17429227, 0.10710297, 0.26585477],
[0.36697641, 0.2389772 , 0.22162812, 0.19738032, 0.2392397 ,
 0.28657616, 0.26008824, 0.2403541 , 0.19829445, 0.21923821,
 0.25768962, 0.24067187, 0.20420303, 0.2390292 , 0.        ,
 0.15346262, 0.20293655, 0.24230826],
[0.30325325, 0.19459264, 0.20358452, 0.21403131, 0.22310475,
 0.2400629 , 0.22197027, 0.21527653, 0.15854515, 0.19349934,
 0.24297859, 0.21444788, 0.18106997, 0.17429227, 0.15346262,
 0.        , 0.1867873 , 0.23780244],
[0.25152437, 0.17578575, 0.14600089, 0.2074863 , 0.20029594,
 0.14094868, 0.14452128, 0.13417896, 0.12310822, 0.11755898,
 0.29501488, 0.13243287, 0.11758264, 0.10710297, 0.20293655,
 0.1867873 , 0.        , 0.2521906 ],
[0.45398825, 0.29076401, 0.28600799, 0.23233857, 0.24800114,
 0.32119404, 0.22554503, 0.28984225, 0.23427458, 0.26633493,
 0.11959038, 0.20929897, 0.20746761, 0.26585477, 0.24230826,
 0.23780244, 0.2521906 , 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     0.423829
PC2     0.136141
PC3     0.125652
PC4     0.102058
PC5     0.065312
PC6     0.044789
PC7     0.028576
PC8     0.022913
PC9     0.017731
PC10    0.013240
```

```
PC11    0.009806
PC12    0.005152
PC13    0.003683
PC14    0.001117
PC15    0.000000
PC16    0.000000
PC17    0.000000
PC18    0.000000
dtype: float64
```

Step 7: Plot PCoA results

In [8]:

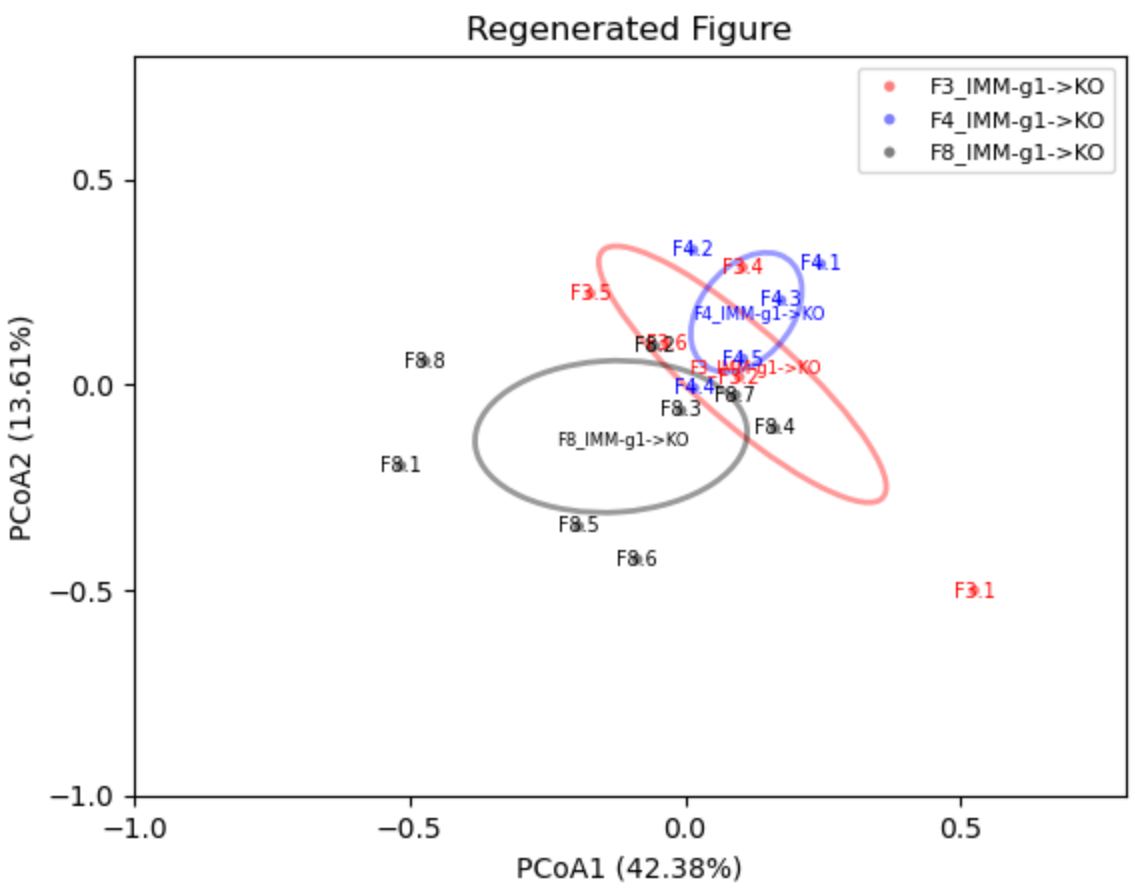
```
colors = ['red', 'blue', 'black']
text_tup = [(-0.1, 0.00),
            (-0.1, -0.02),
            (-0.1, -0.02)]

fig, ax = plt.subplots()
ax.set_xlim((-1.0, 0.8))
ax.set_ylim((-1.0, 0.8))
ax.set_yticks([-1.0, -0.5, 0.0, 0.5])
ax.set_xticks([-1.0, -0.5, 0.0, 0.5])
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
    # multiply by -1 to rotate vector 180 degrees in order to match figure
    pc2 = df_pcoa.iloc[start:end, 1] *-1.0
    # 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='6', 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=8, loc='upper right')
plt.show()
```



Step 8: Compare original and regenerated figures