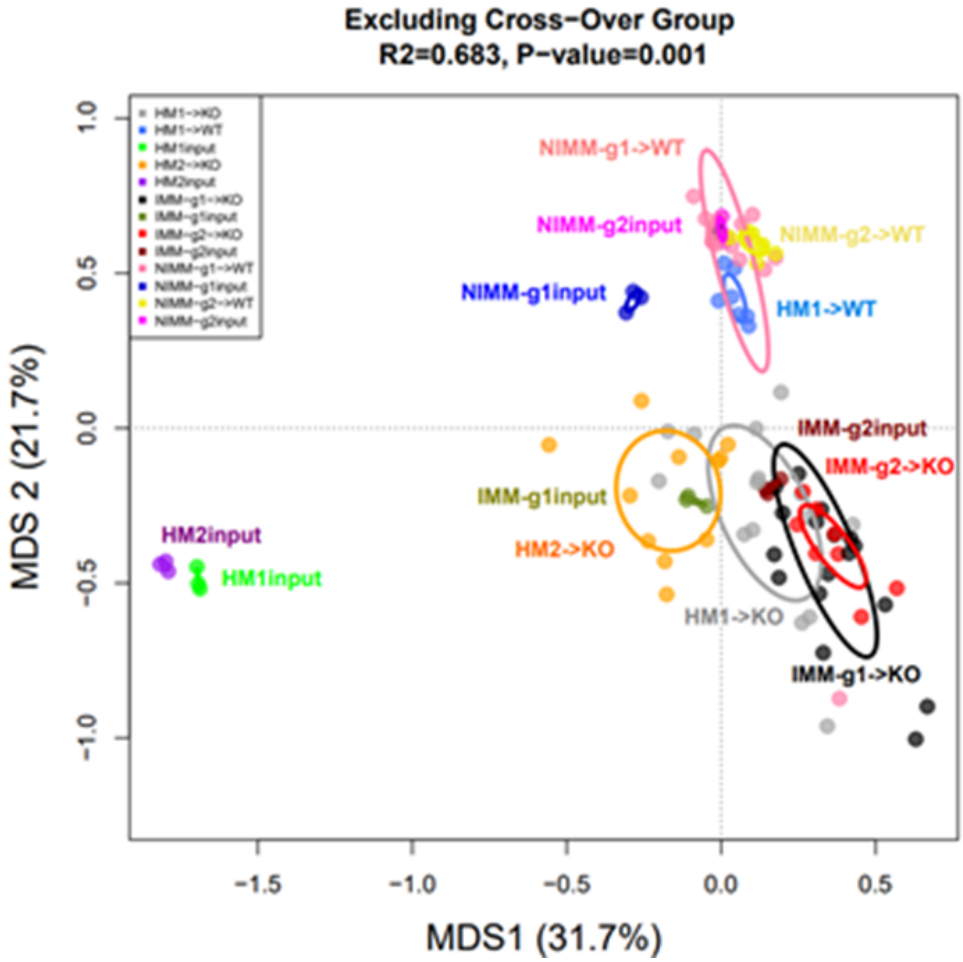
Contents

Aim: Replicate Figure 3A



Main Python Methods

In [1]:
```
import numpy as np
import pandas as pd
```

```
import matplotlib.pyplot as plt
import os
import re
import skbio
from scipy import stats
from scipy.spatial import distance
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

# Create dictionary from metadata table
def get_dict_from_metadata(input_df):
    mydict = {}
    for row in input_df.iterrows():
        obj = row[1]
        sample_id = obj['SampleID']
        key = obj['FMTGroupFMTsourcegtRecipientbackground']
        if key not in mydict:
            mydict[key] = [sample_id]
        else:
            mydict[key].append(sample_id)
    return mydict

def write_dataframe(input_df, file_path):
    input_df.to_csv(file_path, sep='\t', header=True, index=False)

# Reference:  Function "get_cov_ellipse" extracted from the following URL:
# https://scipython.com/book/chapter-7-matplotlib/examples/bmi-data-with-confi
dence-ellipses/
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
feature_tbl_file_path = os.path.join(current_working_dir, 'feature-table_balfo
```

```
ur.txt')
# original metadata table
metadata_file_path = os.path.join(current_working_dir, 'mappingMetadata.txt')

# cross reference dictionary that maps original sample group nomenclature to r
evised nomenclature
cross_ref_dict = { '1gKOgtKO':'IMM-g1->KO',
                   '2gKOgtKO':'IMM-g2->KO',
                   '1gWTgtKO':'NIMM-g1->KO',
                   '1gWTgtWT':'NIMM-g1->WT',
                   '2gWTgtWT':'NIMM-g2->WT',
                   'hFMT.1.2.3.gtKO':'HM1->KO',
                   'hFMT.3.4.5.gtKO':'HM2->KO',
                   'hFMT.1.2.3.gtWT':'HM1->WT',
                   'hFMT.1.2.3.input':'HM1input',
                   'hFMT.3.4.5.input': 'HM2input',
                   '1gKOinput':'IMM-g1input',
                   '2gKOinput':'IMM-g2input',
                   '1gWTinput':'NIMM-g1input',
                   '2gWTinput':'NIMM-g2input'
                 }

# target sample group names for Figure 3A
dict_keys_ordered = ['hFMT.1.2.3.gtKO',
                     'hFMT.1.2.3.gtWT',
                     'hFMT.1.2.3.input',
                     'hFMT.3.4.5.gtKO',
                     'hFMT.3.4.5.input',
                     '1gKOgtKO',
                     '1gKOinput',
                     '2gKOgtKO',
                     '2gKOinput',
                     '1gWTgtWT',
                     '1gWTinput',
                     '2gWTgtWT',
                     '2gWTinput']
```

Data Processing Steps

Step 1: Retrieve original metadata table

In [2]:
```
df_metadata = read_csv_file(metadata_file_path)
df_metadata
```

Out[2]:

	SampleID	UniversalCageNumber	Background	FMTGroupFMTsourcegtRecipientbackground	Passage
0	F8-1	F8-cage-1	129.IL10KO	1gKOgtKO	8
1	F8-2	F8-cage-1	129.IL10KO	1gKOgtKO	8
2	F8-3	F8-cage-2	129.IL10KO	1gKOgtKO	8
3	F8-4	F8-cage-2	129.IL10KO	1gKOgtKO	8
4	F8-5	F8-cage-3	129.IL10KO	1gKOgtKO	8
...	...	...	...	...	...
105	1gWT.2	NaN	NaN	1gWTinput	1gWT
106	1gWT.3	NaN	NaN	1gWTinput	1gWT

107	2gWT.1	NaN	NaN	2gWTinput	2gWT
108	2gWT.2	NaN	NaN	2gWTinput	2gWT
109	2gWT.3	NaN	NaN	2gWTinput	2gWT

110 rows × 8 columns

Step 2: Create dictionary from metadata table

In [3]:
```
dict_metadata = get_dict_from_metadata(df_metadata)
dict_keys_filtered = dict_keys_ordered
filtered_columns = [col for sublist in [dict_metadata[key] for key in dict_key
s_filtered] for col in sublist]
dict_keys_filtered
```

Out[3]:
```
['hFMT.1.2.3.gtKO',
 'hFMT.1.2.3.gtWT',
 'hFMT.1.2.3.input',
 'hFMT.3.4.5.gtKO',
 'hFMT.3.4.5.input',
 '1gKOgtKO',
 '1gKOinput',
 '2gKOgtKO',
 '2gKOinput',
 '1gWTgtWT',
 '1gWTinput',
 '2gWTgtWT',
 '2gWTinput']
```

Step 3: Retrieve feature counts table with filtered sample columns

In [4]:
```
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[4]:

	#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

In [5]:
```
df_feature_counts_filtered = df_feature_counts[filtered_columns]
df_feature_counts_filtered
```

Out[5]:

	F3-7	F3-8	F3-9	F3-10	F3-11	F3-12	F1-7	F1-8	F1-9	F1-10	...	F8-35	F8-36	F8-37
0	8445	319	10551	15735	14058	10445	8869	8035	5953	11460	...	20081	21965	11552
1	34435	43961	31682	20382	17233	27241	19991	23011	12533	27427	...	374	113	200
2	978	13	1555	34	27	51	24450	10453	39194	12486	...	64	55	14
3	28326	10652	3300	3168	900	23043	886	5723	480	6491	...	8678	3211	2961
4	3647	10380	3639	3886	3693	3755	655	2478	205	5444	...	6252	5577	7567
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
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 × 103 columns

Step 4: Calculate total read counts per sample

In [6]:
```
sample_count_sums = df_feature_counts_filtered.sum(axis=0)
sample_count_sums
```

Out[6]:
```
F3-7       99046
F3-8      112057
F3-9      121801
F3-10      98907
F3-11      93879
           ...
F8-40      63669
F8-41      72906
2gWT.1     86473
2gWT.2     76015
2gWT.3     78810
Length: 103, dtype: int64
```

Step 5: Generate normalized counts feature table

In [7]:
```
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[7]:

	F3-7	F3-8	F3-9	F3-10	F3-11	F3-12	F1-7	F1-8	F1-9	F1-1
0	3.904009	2.429163	3.910888	4.174860	4.148578	3.988447	3.906037	3.851763	3.766695	4.00565

1	4.514368	4.566832	4.388371	4.287234	4.237009	4.404734	4.258967	4.308672	4.089975	4.38462
2	2.968161	1.075794	3.079633	1.522678	1.447759	1.686105	4.346407	3.966004	4.585116	4.04289
3	4.429556	3.951232	3.406225	3.478894	2.955347	4.332053	2.906082	3.704425	2.674047	3.75881
4	3.539416	3.940000	3.448677	3.567585	3.568120	3.544228	2.775080	3.341015	2.305792	3.68243
...	...	...	...	...	...	...	...	...	...	.
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 × 103 columns

Step 6: Generate Bray-Curtis dissimilarity distance matrix

In [8]:
```
col_name_list = df_feature_counts_filtered_norm.columns
num_of_cols = len(col_name_list)
dist_mat = np.zeros((num_of_cols, num_of_cols))
tup_list = [(col_name_list.get_loc(col2), col_name_list.get_loc(col1))  for co
l1 in col_name_list for col2 in col_name_list if col_name_list.get_loc(col2) >
 col_name_list.get_loc(col1)]

for tup in tup_list:
    col_1 = tup[0]
    col_2 = tup[1]
    dist = distance.braycurtis(df_feature_counts_filtered_norm.iloc[:, col_1],
 df_feature_counts_filtered_norm.iloc[:, col_2])
    dist_mat[col_1, col_2] = dist

dist_mat_sym = dist_mat + dist_mat.T
print('Dimensions of matrix: {}'.format(dist_mat_sym.shape))
dist_mat_sym
```

```
Dimensions of matrix: (103, 103)
```

Out[8]:
```
array([[0.        , 0.17459747, 0.34079072, ..., 0.32584445, 0.34141039,
        0.35352788],
       [0.17459747, 0.        , 0.38958066, ..., 0.34970683, 0.36941559,
        0.37073948],
       [0.34079072, 0.38958066, 0.        , ..., 0.30164799, 0.30201668,
        0.31426109],
       ...,
       [0.32584445, 0.34970683, 0.30164799, ..., 0.        , 0.03575365,
        0.04063024],
       [0.34141039, 0.36941559, 0.30201668, ..., 0.03575365, 0.        ,
        0.03552113],
       [0.35352788, 0.37073948, 0.31426109, ..., 0.04063024, 0.03552113,
        0.        ]])
```

Step 7: Perform Principal Coordinate Analysis (PCoA)

In [9]:
```
my_pcoa = skbio.stats.ordination.pcoa(dist_mat_sym)
df_pcoa = my_pcoa.samples[['PC1', 'PC2']]
my_pcoa.proportion_explained
```

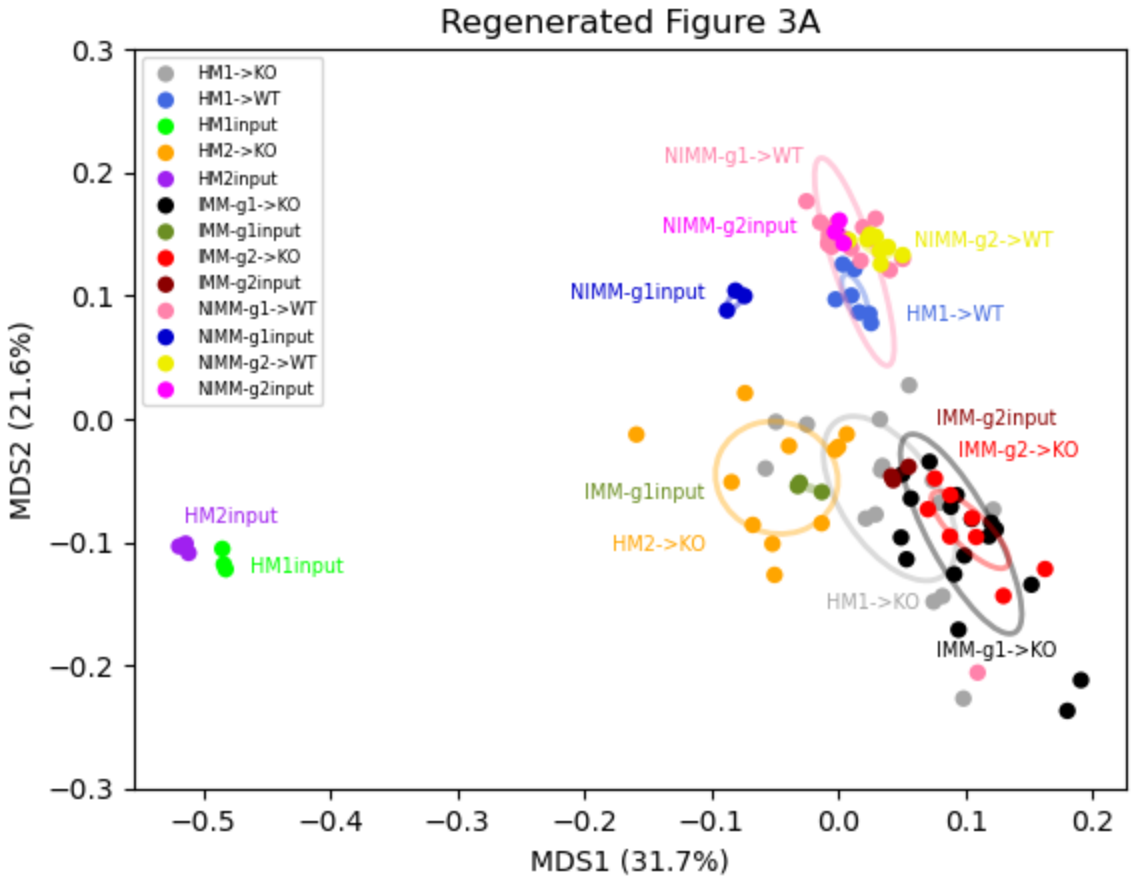
```
C:\ProgramData\Anaconda3\lib\site-packages\skbio\stats\ordination\_principal
_coordinate_analysis.py:143: RuntimeWarning: The result contains negative ei
genvalues. Please compare their magnitude with the magnitude of some of the
largest positive eigenvalues. If the negative ones are smaller, it's probabl
y safe to ignore them, but if they are large in magnitude, the results won't
be useful. See the Notes section for more details. The smallest eigenvalue i
s -0.03986325960166524 and the largest is 1.9638240904410669.
  warn(
```

Out[9]:
```
PC1      0.317344
PC2      0.216448
PC3      0.100538
PC4      0.059908
PC5      0.035616
           ...
PC99     0.000000
PC100    0.000000
PC101    0.000000
PC102    0.000000
PC103    0.000000
Length: 103, dtype: float64
```

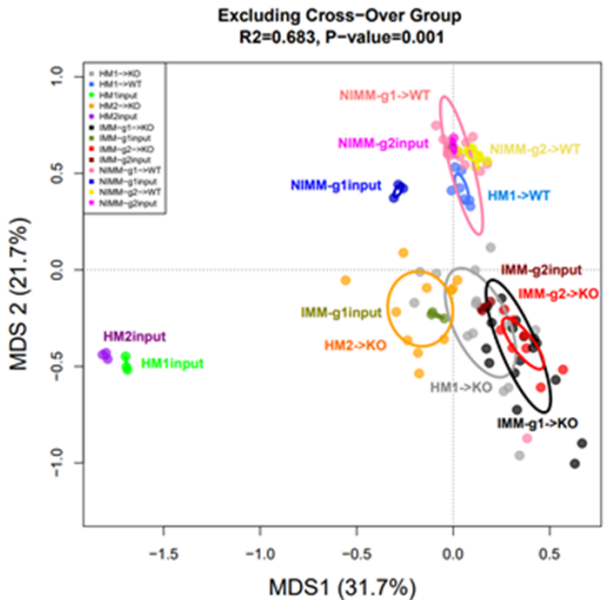
Step 8: Plot PCoA Results

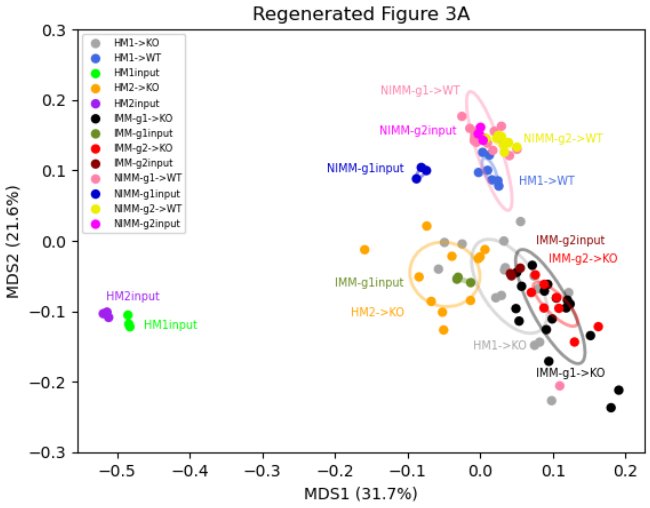
In [10]:
```
group_counts = [len(dict_metadata[key]) for key in dict_keys_filtered]
group_names = [cross_ref_dict[key] for key in dict_keys_filtered]
colors = ["#A6A6A6", "#4169E1", "#00FF00", "#FFA500", "#A020F0",
                    "#000000", "#6B8E23", "#FF0000", "#8B0000",
                     "#FF82AB", "#0000CD", "#EEEE00", "#FF00FF"]
text_tup = [(-0.05, -0.09),
            (0.04, -0.02),
            (0.02, -0.01),
            (-0.13,-0.06),
            (0.0, 0.02),
            (-0.02, -0.1),
            (-0.175, -0.01),
            (-0.01, 0.06),
            (0.03, 0.04),
            (-0.15, 0.08),
            (-0.13, 0.0),
            (0.03, 0.0),
            (-0.14, 0.0)]

fig, ax = plt.subplots()
ax.set_ylim((-0.3, 0.3))
ax.set_xlabel('MDS1 ({}%)'.format(np.round(my_pcoa.proportion_explained.PC1 *
100, 1)))
ax.set_ylabel('MDS2 ({}%)'.format(np.round(my_pcoa.proportion_explained.PC2 *
100, 1)))
ax.set_title('Regenerated Figure 3A')

end = 0
for idx, group_count in enumerate(group_counts):
    start = end
```

```
    end = start + group_count

    # multiply by -1 to rotate vector 180 degrees in order to match figure 3A
    pc1 = df_pcoa.iloc[start:end, 0] * -1.0
    # multiply by -1 to rotate vector 180 degrees in order to match figure 3A
    pc2 = df_pcoa.iloc[start:end, 1] * -1.0

    group_name = group_names[idx]
    ax.scatter(pc1, pc2, c=colors[idx],  label=group_name,
               alpha=1.0, 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)

ax.legend(fontsize=6, loc='upper left')
plt.show()
```



Step 9: Compare Original and Regenerated Figures


Excluding Cross−Over Group
R2=0.683, P−value=0.001
NIMM-g1->WT
NIMM-g2input
NIMM-g2->WT
NIMM-g1input
HM1->WT
IMM-g2input
IMM-g2->KO
IMM-g1input
HM2->KO
HM2input
HM1input
HM1->KO
IMM-g1->KO
MDS 2 (21.7%)
MDS1 (31.7%)


Regenerated Figure 3A
HM1->KO
HM1->WT
HM1input
HM2->KO
HM2input
IMM-g1->KO
IMM-g1input
IMM-g2->KO
IMM-g2input
NIMM-g1->WT
NIMM-g1input
NIMM-g2->WT
NIMM-g2input
MDS2 (21.6%)
MDS1 (31.7%)