In [1]:
```
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as st
from scipy.stats import ttest_ind, sem
import os
%matplotlib inline

os.chdir("/Users/yolandatiao/GSuite Scripps/MLC_Screen")
import fit_nbinom
```

0. Define Screen Data Class and Transformations

In [2]:
```
class screenData():
    def __init__(self, raw_data, col_data, row_data):
        self.raw_df = pd.read_csv(raw_data, index_col="name")
        self.col_df = pd.read_csv(col_data)
        self.row_df = pd.read_csv(row_data, index_col='name')
        self.qcflag = False
        conditions = list(set(list(self.col_df['condition'])))
        self.cond_dict = {}
        for cond in conditions:
            self.cond_dict[cond] = [str(x) for x in list(self.col_df[self.c
        raw_cond_df = pd.DataFrame({'name': list(self.raw_df.index.values)}
        for i in self.cond_dict.keys():
            raw_cond_df[i] = list(self.raw_df[self.cond_dict[i]].sum(axis=1
        raw_cond_df = raw_cond_df.set_index('name')
        self.raw_df = raw_cond_df

    def qc(self, input_name, count_cutoff, input_vs_output_cutoff):
        self.raw_df['target'] = list(self.row_df['target'])
        self.raw_df['sum'] = list(self.raw_df.sum(axis=1))
        self.raw_df['out_sum'] = list(self.raw_df['sum'] - self.raw_df[inpu
        self.raw_df['input_vs_output'] = list(self.raw_df[input_name] / sel
        self.raw_df = self.raw_df[self.raw_df['sum'] > count_cutoff]
        self.raw_df = self.raw_df[self.raw_df['input_vs_output'] < input_vs
        # update row metadata
        self.row_df = pd.DataFrame({'name': list(self.raw_df.index.values),
        self.row_df.set_index('name')

        del self.raw_df['sum']
        del self.raw_df['out_sum']
        del self.raw_df['input_vs_output']
        del self.raw_df['target']
        self.qcflag = True

    def norm_count(self):
        if (not self.qcflag):
            print("Proceed without quality control...")
        print("Perform normalization for each sample...")
        self.norm_df = self.raw_df / self.raw_df.sum() * 1000000
        self.norm_df = self.norm_df.astype('int32')
        self.all_normalized_counts = []
        for col in self.norm_df.columns:
            self.all_normalized_counts += list(self.norm_df[col])
        self.all_normalized_counts.sort()

    def nb_pctl(self):
        if not hasattr(self, "norm_df"):
            self.norm_count()
        self.nbinom_parameters = fit_nbinom.fit_nbinom(np.array(self.all_no
        self.nbpctl_df = pd.DataFrame(st.nbinom.cdf(self.norm_df, self.nbin
        self.nbpctl_df.columns = list(self.raw_df.columns)
        self.nbpctl_df.index = list(self.raw_df.index.values)

    def nb_pctl_shift(self):
        if not hasattr(self, "nbpctl_df"):
            self.nb_pctl()
        self.nbpctl_shift_df = pd.DataFrame({"name": list(self.nbpctl_df.in
```

```
        conditions = list(self.nbpctl_df.columns)
        for i in range(0, len(conditions)):
            for j in range((i+1), len(conditions)):
                i_name = conditions[i]
                j_name = conditions[j]
                comp = "__vs__".join([i_name, j_name])
                self.nbpctl_shift_df[comp] = list(self.nbpctl_df[i_name] -
        self.nbpctl_shift_df = self.nbpctl_shift_df.set_index('name')

    def target_eff(self):
        if not hasattr(self, "nbpctl_shift"):
            self.nb_pctl_shift()

        self.nbpctl_shift_df['target'] = list(self.row_df['target'])
        self.target_shift_df = self.nbpctl_shift_df.groupby('target').mean(

        self.target_shift_z_df = pd.DataFrame({'target': list(self.target_s
        for col in self.target_shift_df.columns:
            self.target_shift_z_df[col] = st.zscore(np.array(self.target_sh
        self.target_shift_z_df = self.target_shift_z_df.set_index('target')

        del self.nbpctl_shift_df['target']

    def target_ttest(self):
        if not hasattr(self, "nbpctl_shift_df"):
            self.nb_pctl_shift()
        comparisons = list(self.nbpctl_shift_df.columns)
        targets = list(set(list(self.row_df['target'])))
        self.target_shift_p_df = pd.DataFrame({'target': targets})
        self.target_shift_st_df = pd.DataFrame({'target': targets})
        self.nbpctl_shift_df['target'] = list(self.row_df['target'])
        for cp in comparisons:
            cp_p = []
            cp_st = []
            cp_nums = list(self.nbpctl_shift_df[cp])
            for tg in targets:
                tg_nums = list(self.nbpctl_shift_df[self.nbpctl_shift_df['t
                cp_tg_tt = st.ttest_ind(tg_nums, cp_nums)
                cp_p.append(cp_tg_tt.pvalue)
                cp_st.append(cp_tg_tt.statistic)
            self.target_shift_p_df[cp] = cp_p
            self.target_shift_st_df[cp] = cp_st
        self.target_shift_p_df = self.target_shift_p_df.set_index('target')
        self.target_shift_st_df = self.target_shift_st_df.set_index('target
        self.target_shift_p_df = self.target_shift_p_df.fillna(1)
        self.target_shift_st_df = self.target_shift_st_df.fillna(0)
        del self.nbpctl_shift_df['target']

    def target_sem(self):
        if not hasattr(self, "nbpctl_shift_df"):
            self.nb_pctl_shift()
        comparisons = list(self.nbpctl_shift_df.columns)
        targets = list(set(list(self.row_df['target'])))
        self.target_shift_sem_df = pd.DataFrame({'target': targets})
        self.nbpctl_shift_df['target'] = list(self.row_df['target'])
        for cp in comparisons:
            cp_sem = []
```

```
            for tg in targets:
                tg_nums = list(self.nbpctl_shift_df[self.nbpctl_shift_df['t
                cp_sem.append(sem(np.array(tg_nums)))
            self.target_shift_sem_df[cp] = cp_sem
        self.target_shift_sem_df = self.target_shift_sem_df.set_index('targ
        self.target_shift_sem_df = self.target_shift_sem_df.fillna(100)
        del self.nbpctl_shift_df['target']

    def adj_z_by_p(self):
        if hasattr(self, 'target_shift_z_adj_df'):
            print("Adjust Z Score already exist. Exiting...")
            return(False)
        if not hasattr(self, 'target_shift_p_df'):
            self.target_ttest()
        if not hasattr(self, 'target_shift_z_df'):
            self.target_eff()
        self.target_shift_z_df = self.target_shift_z_df.sort_index()
        self.target_shift_p_df = self.target_shift_p_df.sort_index()
        self.target_shift_z_adj_df = pd.DataFrame({'target': list(self.targ
        comparisons = list(self.target_shift_z_df.columns)
        for cp in comparisons:
            cp_z = list(self.target_shift_z_df[cp])
            cp_p = [(x + 0.01) for x in list(self.target_shift_p_df[cp])]
            cp_z_adj = cp_z / (np.sqrt(cp_p))
            self.target_shift_z_adj_df[cp] = cp_z_adj
        self.target_shift_z_adj_df = self.target_shift_z_adj_df.set_index('

    def adj_z_by_sem(self):
        if hasattr(self, 'target_shift_z_adj_df'):
            print("Adjust Z Score already exist. Exiting...")
            return(False)
        if not hasattr(self, 'target_shift_sem_df'):
            self.target_sem()
        if not hasattr(self, 'target_shift_z_df'):
            self.target_eff()
        self.target_shift_z_df = self.target_shift_z_df.sort_index()
        self.target_shift_sem_df = self.target_shift_sem_df.sort_index()
        self.target_shift_z_adj_df = pd.DataFrame({'target': list(self.targ
        comparisons = list(self.target_shift_z_df.columns)
        for cp in comparisons:
            cp_z = list(self.target_shift_z_df[cp])
            cp_sem = [(x + 0.01) for x in list(self.target_shift_sem_df[cp]
            cp_z_adj = cp_z / (np.sqrt(cp_sem))
            self.target_shift_z_adj_df[cp] = cp_z_adj
        self.target_shift_z_adj_df = self.target_shift_z_adj_df.set_index('

    def to_csv(self, out_dir):
        not_write = ['col_df', 'row_df']
        items = list(vars(mlc_data).keys())
        items = [x for x in items if ('df' in x and x not in not_write)]
        if (not os.path.exists(out_dir)):
            os.mkdir(out_dir)
        for i in items:
            i_name = "%s/%s.csv"%(out_dir, i)
            i_name = i_name.replace("_df", "")
            vars(mlc_data)[i].to_csv(i_name)
```

1. Read data and apply transformations

In [3]:
```
mlc_raw = "/Users/yolandatiao/GSuite Scripps/MLC_Screen/input/sundrud_repor
mlc_col = "/Users/yolandatiao/GSuite Scripps/MLC_Screen/input/sundrud_repor
mlc_row = "/Users/yolandatiao/GSuite Scripps/MLC_Screen/input/sundrud_repor

# Read input data
mlc_data = screenData(mlc_raw, mlc_col, mlc_row)

# Apply quality control, filter: low reads and drop outs
mlc_data.qc('pre_inj', 100, 3)

# Calculate adjusted z score
mlc_data.adj_z_by_sem()

# Write output
mlc_data.to_csv('mlc_screen_semAdj')
```

```
Perform normalization for each sample...

/Users/yolandatiao/anaconda3/envs/stats/lib/python3.8/site-packages/scip
y/optimize/optimize.py:697: RuntimeWarning: invalid value encountered in
double_scalars
  df = (f(*((xk + d,) + args)) - f0) / d[k]
/Users/yolandatiao/anaconda3/envs/stats/lib/python3.8/site-packages/nump
y/core/_methods.py:216: RuntimeWarning: Degrees of freedom <= 0 for slice
  ret = _var(a, axis=axis, dtype=dtype, out=out, ddof=ddof,
/Users/yolandatiao/anaconda3/envs/stats/lib/python3.8/site-packages/nump
y/core/_methods.py:209: RuntimeWarning: invalid value encountered in doub
le_scalars
  ret = ret.dtype.type(ret / rcount)
```

2. Check output

- Focus on small intestine Mdr1 positive (**si_Mp**) versus small intestine Mdr1 negative (**si_Mn**)

In [4]:
```
# Select and sort data
shift_sorted_df = pd.DataFrame(mlc_data.target_shift_df.sort_values(by='si_
shift_z_sorted_df = pd.DataFrame(mlc_data.target_shift_z_df.sort_values(by=
shift_z_adj_sorted_df = pd.DataFrame(mlc_data.target_shift_z_adj_df.sort_va

#shift_p_df = pd.DataFrame(mlc_data.target_shift_p_df.sort_values(by='si_Mp
#shift_st_df = pd.DataFrame(mlc_data.target_shift_st_df.sort_values(by='si_
shift_sem_df = pd.DataFrame(mlc_data.target_shift_sem_df.sort_values(by='si
```

2.1 Adjusted Z Score (of percentile shift)

In [5]:
```
shift_z_adj_sorted_df.head(n=10)
```

Out[5]:

target	si_Mp__vs__si_Mn
Abcb1b	-1.102015
Abcb1a	-1.072482
Nr1i3	-0.924523
Nr2f6	-0.797840
Scand1	-0.648201
Nr0b2	-0.625456
Thra	-0.605021
Nr1d2	-0.590592
Hnf4g	-0.560093
Ppargc1b	-0.495772

In [6]:
```
shift_z_adj_sorted_df.tail(n=10)
```

Out[6]:

target	si_Mp__vs__si_Mn
Nr4a1	0.838000
Rxra	0.872658
Cd19	0.882607
Rxrb	0.919613
Ncoa1	0.982140
Ar	1.025874
Nr5a2	1.052669
negCtrl	1.105642
Esr2	1.256860
Nr1d1	1.305904

2.1 Z Score (of percentile shift)

In [7]:
```
shift_z_sorted_df.head(n=10)
```

Out[7]:

target	si_Mp__vs__si_Mn
Abcb1a	-2.701878
Scand1	-2.062608
Nr2f6	-1.882859
Thra	-1.726952
Ppargc1b	-1.540401
Nr1i3	-1.444528
Hnf4g	-1.317952
Nr2f1	-1.258917
Abcb1b	-1.239469
Esr1	-1.188065

In [8]:
```
shift_z_sorted_df.tail(n=10)
```

Out[8]:

target	si_Mp__vs__si_Mn
Thrb	0.843061
Nr4a1	0.954861
Hnf4a	1.227635
Ar	1.231297
Nr2e3	1.333887
Nr5a2	1.335143
Esr2	1.483393
Nr1d1	1.817280
Ncoa1	2.327917
Rxra	2.890425

2.2 Percentile shift Standard Error of Measurement (SEM)

In [9]:
```
shift_sem_df.head(n=10)
```

Out[9]:

target	si_Mp__vs__si_Mn
negCtrl	0.358437
Rxrb	0.435297
Cd19	0.482369
Eomes	0.501814
Rxrg	0.539177
Ncoa5	0.653375
Vdr	0.722210
Esrrb	0.770201
Creb	0.912086
Nr2e1	1.090655

In [10]:
```
shift_sem_df.tail(n=10)
```

Out[10]:

target	si_Mp__vs__si_Mn
Esrra	6.343448
Nr4a3	6.542613
Rorb	6.544823
Esr1	7.501264
Nr2f1	7.841656
Thra	8.137419
Ppargc1b	9.643933
Scand1	10.115437
Rxra	10.960725
Blimp	100.000000