multi_shrna_screening_ap009

March 26, 2025

1 Multiplexed shRNA screening AP009 analytics

Biao Li

This notebook includes analysis scripts of processing count table of multiplexed shRNA screening experiments on AP009 cell line received from Cellecta, and evaluating * whether multi-shRNA system works as expected * prognostic effect of genotype X, and * predictive effect of genetype X on response to treatment

Note - bash commands to convert notebook to pdf:

> `jupyter nbconvert --to latex multi_shrna_screening_ap009.ipynb`

> `xelatex multi_shrna_screening_ap009.tex`

1.1 Setup path to project directory

This should be absolute path to where project is saved at as a git repo.

Subfolders of the project directory should be `code`, `data`, `app`, etc.

```
[303]: proj_dir = "/Users/bli/Documents/Projects/Multi_shRNA_screening_AP009/"
```

1.2 (ToDos)

- (Done) Add correlation plots between baseline condition and plasmid counts per target
- (Done) Normalize raw counts to library size of each sample
- (Done) Normalize representations to dox- group
- (Done) log2 transform ratio of ratios such that under the alternative hypothesis Score != 0, a positive sign suggests that cells with a particular shRNA are more resistant than cells of NT shRNAs, while a negative sign suggests cells are more sensitive
- (Done) Use nonparametric bootstrapping to derive null distribution of medians (among resampling 10 NT shRNAs from NT shRNA pool) and assess statistical significance / adjusted empirical p-values of each target gene
  - Prognostic effect (combine three timepoints)
  - Predictive effect (combine three timepoints and two dosages, combine three timepoints and keep dosages separate)
  - instead of boxplot, show median +/- bootstrapped CI of median, highlight medians of statistical significance with a bright color, and others with gray
  - Volcano plot of adjusted p-values vs observed median effects
- (flip) Reannotate RCC (relative cell counts) as RTN (relative tumor-cell numbers)

- (+/- Dox) Add analytics/figures similar to Fig.2 D of Ian's NatComm 2023 paper
    - check Prognostic / predictive effects between T10 and T13
- Add analytics/figures similar to Fig.7 of Ian's NatComm 2023 paper
- Assess reproducibility between two technical replicates (D18-A vs D18-B)
- Annotated gene list
    - update control annotations
    - oncokb annotations
    - pathway / protein complex annotations

[4]:
```
!pip install matplotlib_venn --trusted-host pypi.python.org --trusted-host pypi.
↪org --trusted-host files.pythonhosted.org
```

```
Requirement already satisfied: matplotlib_venn in
/opt/anaconda3/lib/python3.11/site-packages (1.1.2)
Requirement already satisfied: matplotlib in /opt/anaconda3/lib/python3.11/site-
packages (from matplotlib_venn) (3.8.0)
Requirement already satisfied: numpy in /opt/anaconda3/lib/python3.11/site-
packages (from matplotlib_venn) (1.23.4)
Requirement already satisfied: scipy in /opt/anaconda3/lib/python3.11/site-
packages (from matplotlib_venn) (1.9.3)
Requirement already satisfied: contourpy>=1.0.1 in
/opt/anaconda3/lib/python3.11/site-packages (from matplotlib->matplotlib_venn)
(1.2.0)
Requirement already satisfied: cycler>=0.10 in
/opt/anaconda3/lib/python3.11/site-packages (from matplotlib->matplotlib_venn)
(0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in
/opt/anaconda3/lib/python3.11/site-packages (from matplotlib->matplotlib_venn)
(4.25.0)
Requirement already satisfied: kiwisolver>=1.0.1 in
/opt/anaconda3/lib/python3.11/site-packages (from matplotlib->matplotlib_venn)
(1.4.4)
Requirement already satisfied: packaging>=20.0 in
/opt/anaconda3/lib/python3.11/site-packages (from matplotlib->matplotlib_venn)
(23.1)
Requirement already satisfied: pillow>=6.2.0 in
/opt/anaconda3/lib/python3.11/site-packages (from matplotlib->matplotlib_venn)
(10.2.0)
Requirement already satisfied: pyparsing>=2.3.1 in
/opt/anaconda3/lib/python3.11/site-packages (from matplotlib->matplotlib_venn)
(3.0.9)
Requirement already satisfied: python-dateutil>=2.7 in
/opt/anaconda3/lib/python3.11/site-packages (from matplotlib->matplotlib_venn)
(2.8.2)
Requirement already satisfied: six>=1.5 in /opt/anaconda3/lib/python3.11/site-
packages (from python-dateutil>=2.7->matplotlib->matplotlib_venn) (1.16.0)
```

1.3 Dependency pkgs

[312]:
```
import pandas as pd
import numpy as np
from prettytable import PrettyTable
from IPython.display import display
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import os
from scipy.stats import pearsonr

from matplotlib_venn import venn3
import matplotlib.patches as patches
from adjustText import adjust_text
from statsmodels.stats.multitest import multipletests

# import utility functions
from utility_functions import *

# seed of random number generator
rng_seed = 1234
```

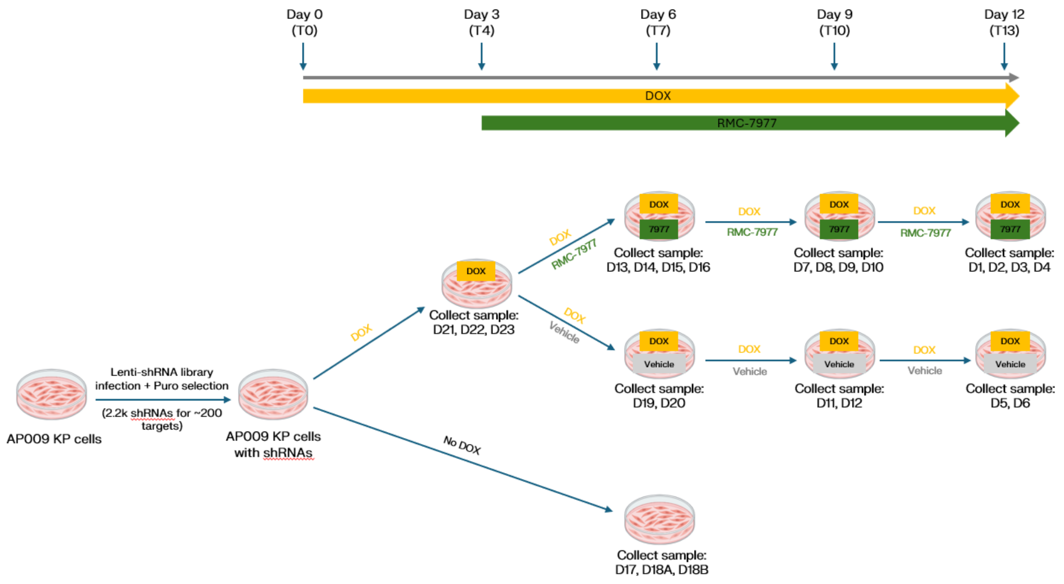
1.4 Experimental design and parameters

Load experimental design xlsx file (from Ian) * Master list tab of target genes, vector types, and annotations * Experimental design tab of groups and samples

Load sample description xlsx file (from Cellecta) * Actual Sample ID and description on flowcell, and inferred information of Group, Day_Tx, Replicate, Dox, etc.

Also check out schematic workflow of experimental design (from Zheng)

Study Design


Day 0
(T0)
Day 3
(T4)
Day 6
(T7)
Day 9
(T10)
Day 12
(T13)
DOX
RMC-7977
AP009 KP cells
Lenti-shRNA library infection + Puro selection
(2.2k shRNAs for ~200 targets)
AP009 KP cells with shRNAs
DOX
Collect sample: D21, D22, D23
DOX
RMC-7977
DOX
7977
Collect sample: D13, D14, D15, D16
DOX
RMC-7977
Collect sample: D7, D8, D9, D10
DOX
RMC-7977
Collect sample: D1, D2, D3, D4
DOX
Vehicle
Collect sample: D19, D20
DOX
Vehicle
Collect sample: D11, D12
DOX
Vehicle
Collect sample: D5, D6
No DOX
Collect sample: D17, D18A, D18B

[6]:
```
# #### experimental design xlsx file
# file_path = "~/Documents/Projects/Multi_shRNA_screening_AP009/data/
↪Multiplexed shRNA screen in 2D AP009 - Cellecta.xlsx"

# xls = pd.ExcelFile(file_path)

# ## Gene targets
# gene_targets_df = pd.read_excel(xls, sheet_name="Master list")
# # Filter out "Individual dual shRNA vector" from gene targets
# filtered_gene_targets_df = gene_targets_df[gene_targets_df["Vector type"] !=␣
↪"Individual dual shRNA vector"]
# filtered_gene_targets = filtered_gene_targets_df["Mouse gene symbol"].
↪dropna().unique()

# # ## Define experimental parameters
# # # Num clonal barcodes per gene
# # num_clonal_barcodes = 12000
# # # Num shRNAs per gene
# # num_shRNAs_per_gene = 10
# # # N_reps per condition per timepoint
# # num_replicates = 2

# ## Define experimental conditions and timepoints
# # available timepoints
# time_points = ["0d", "3d", "6d", "9d"]
# # experimental conditions based on design
# conditions = [
#     "Baseline_NoDox_Vehicle",
#     "Baseline_Dox_PreTx",  # Only at 0d
#     "Prognostic_Dox_Vehicle",
#     "Predictive_Dox_7977_LowDose",  # IC30 early, IC50 later
#     "Predictive_Dox_7977_HighDose"  # IC90
# ]
```

[298]:
```
#### sample description xlsx file
# file_path_cellecta = "~/Documents/Projects/Multi_shRNA_screening_AP009/data/
 ↪sample_description_rectified.xlsx"
file_path_cellecta = os.path.join(proj_dir, "data/sample_description_rectified.
 ↪xlsx")

df_sd = pd.read_excel(file_path_cellecta, sheet_name='Sheet1')

# Set the first row as column headers and remove it from the data
df_sd.columns = df_sd.iloc[0]
df_sd = df_sd[1:].reset_index(drop=True)

# Rename columns to remove any unintended whitespace
df_sd.columns = df_sd.columns.str.strip()
```

Utility function of table viewing

[299]:
```
display(df_sd)
```

```
0  Sample_ID Sample_Description         Library  \
0         D1      T13_Dox_0.6nM  2.2K-REVMED-ZZ
1         D2      T13_Dox_0.6nM  2.2K-REVMED-ZZ
2         D3      T13_Dox_3.5nM  2.2K-REVMED-ZZ
3         D4      T13_Dox_3.5nM  2.2K-REVMED-ZZ
4         D5    T13_Dox_Vehicle  2.2K-REVMED-ZZ
5         D6    T13_Dox_Vehicle  2.2K-REVMED-ZZ
6         D7      T10_Dox_0.6nM  2.2K-REVMED-ZZ
7         D8      T10_Dox_0.6nM  2.2K-REVMED-ZZ
8         D9      T10_Dox_3.5nM  2.2K-REVMED-ZZ
9        D10      T10_Dox_3.5nM  2.2K-REVMED-ZZ
10       D11    T10_Dox_Vehicle  2.2K-REVMED-ZZ
11       D12    T10_Dox_Vehicle  2.2K-REVMED-ZZ
12       D13       T7_Dox_0.6nM  2.2K-REVMED-ZZ
13       D14       T7_Dox_0.6nM  2.2K-REVMED-ZZ
14       D15       T7_Dox_3.5nM  2.2K-REVMED-ZZ
15       D16       T7_Dox_3.5nM  2.2K-REVMED-ZZ
16       D17      T7_NoDox_NoTx  2.2K-REVMED-ZZ
17     D18-A      T7_NoDox_NoTx  2.2K-REVMED-ZZ
18     D18-B      T7_NoDox_NoTx  2.2K-REVMED-ZZ
19       D19     T7_Dox_Vehicle  2.2K-REVMED-ZZ
20       D20     T7_Dox_Vehicle  2.2K-REVMED-ZZ
21       D21        T4_Dox_NoTx  2.2K-REVMED-ZZ
```

```
22       D22        T4_Dox_NoTx  2.2K-REVMED-ZZ
23       D23        T4_Dox_NoTx  2.2K-REVMED-ZZ
24   plasmid            plasmid  2.2K-REVMED-ZZ

0                                           Vector          Flowcell  \
0    pRSIT16cb-U6tet-sh-CMV-tetR-2A-TagRFP-2A-Puro  25-03-11  102190
1    pRSIT16cb-U6tet-sh-CMV-tetR-2A-TagRFP-2A-Puro  25-03-11  102190
2    pRSIT16cb-U6tet-sh-CMV-tetR-2A-TagRFP-2A-Puro  25-03-11  102190
3    pRSIT16cb-U6tet-sh-CMV-tetR-2A-TagRFP-2A-Puro  25-03-11  102190
4    pRSIT16cb-U6tet-sh-CMV-tetR-2A-TagRFP-2A-Puro  25-03-11  102190
5    pRSIT16cb-U6tet-sh-CMV-tetR-2A-TagRFP-2A-Puro  25-03-11  102190
6    pRSIT16cb-U6tet-sh-CMV-tetR-2A-TagRFP-2A-Puro  25-03-11  102190
7    pRSIT16cb-U6tet-sh-CMV-tetR-2A-TagRFP-2A-Puro  25-03-11  102190
8    pRSIT16cb-U6tet-sh-CMV-tetR-2A-TagRFP-2A-Puro  25-03-11  102190
9    pRSIT16cb-U6tet-sh-CMV-tetR-2A-TagRFP-2A-Puro  25-03-11  102190
10   pRSIT16cb-U6tet-sh-CMV-tetR-2A-TagRFP-2A-Puro  25-03-11  102190
11   pRSIT16cb-U6tet-sh-CMV-tetR-2A-TagRFP-2A-Puro  25-03-11  102190
12   pRSIT16cb-U6tet-sh-CMV-tetR-2A-TagRFP-2A-Puro  25-03-11  102190
13   pRSIT16cb-U6tet-sh-CMV-tetR-2A-TagRFP-2A-Puro  25-03-11  102190
14   pRSIT16cb-U6tet-sh-CMV-tetR-2A-TagRFP-2A-Puro  25-03-11  102190
15   pRSIT16cb-U6tet-sh-CMV-tetR-2A-TagRFP-2A-Puro  25-03-11  102190
16   pRSIT16cb-U6tet-sh-CMV-tetR-2A-TagRFP-2A-Puro  25-03-11  102190
17   pRSIT16cb-U6tet-sh-CMV-tetR-2A-TagRFP-2A-Puro  25-03-11  102190
18   pRSIT16cb-U6tet-sh-CMV-tetR-2A-TagRFP-2A-Puro  25-03-11  102190
19   pRSIT16cb-U6tet-sh-CMV-tetR-2A-TagRFP-2A-Puro  25-03-11  102190
20   pRSIT16cb-U6tet-sh-CMV-tetR-2A-TagRFP-2A-Puro  25-03-11  102190
21   pRSIT16cb-U6tet-sh-CMV-tetR-2A-TagRFP-2A-Puro  25-03-11  102190
22   pRSIT16cb-U6tet-sh-CMV-tetR-2A-TagRFP-2A-Puro  25-03-11  102190
23   pRSIT16cb-U6tet-sh-CMV-tetR-2A-TagRFP-2A-Puro  25-03-11  102190
24   pRSIT16cb-U6tet-sh-CMV-tetR-2A-TagRFP-2A-Puro  25-03-11  102190

0         Tx   Group Day_Tx Replicate  Dox                             Note
0        Low       5      9         2    Y                              NaN
1        Low       5      9         1    Y                              NaN
2       High       4      9         2    Y                              NaN
3       High       4      9         1    Y                              NaN
4    Vehicle       2      9         2    Y                              NaN
5    Vehicle       2      9         1    Y                              NaN
6        Low       5      6         2    Y                              NaN
7        Low       5      6         1    Y                              NaN
8       High       4      6         2    Y                              NaN
9       High       4      6         1    Y                              NaN
10   Vehicle       2      6         2    Y                              NaN
11   Vehicle       2      6         1    Y                              NaN
12       Low       5      3         2    Y                              NaN
13       Low       5      3         1    Y                              NaN
14      High       4      3         2    Y                              NaN
15      High       4      3         1    Y                              NaN
```

```
16      None       1      3          2    N                             NaN
17      None       1      3          1    N                             NaN
18      None       1      3          3    N  should tech rep be in group 2?
19   Vehicle       2      3          2    Y                             NaN
20   Vehicle       2      3          1    Y                             NaN
21      None  pre-Tx      0          3    Y     should this be replicate 3?
22      None  pre-Tx      0          2    Y                             NaN
23      None  pre-Tx      0          1    Y                             NaN
24       NaN     NaN    NaN        NaN  NaN                             NaN
```

1.5 Barcodes (shRNA) count table

Load count table (from Cellecta) and extract fields of sample IDs and target gene

```
[161]: # file_path_count_table = "~/Documents/Projects/Multi_shRNA_screening_AP009/
        ↪data/Count_Table.csv"
       file_path_count_table = os.path.join(proj_dir, "data/Count_Table.csv")

       df_counts = pd.read_csv(file_path_count_table)

       # Extract columns that start with 'D' plus the 'Gene Symbol / Target Name'␣
        ↪column
       selected_columns = [col for col in df_counts.columns if col.startswith("D")] +␣
        ↪["Gene Symbol / Target Name", "plasmid"]

       # Create a new dataframe with selected columns
       df_counts = df_counts[selected_columns].copy()

       # Rename 'Gene Symbol / Target Name' to 'target_gene'
       df_counts = df_counts.rename(columns={"Gene Symbol / Target Name":␣
        ↪"Target_Gene"})

       # Rename 'Non-Targeting-Mouse' to 'NT' in the target_gene column
       df_counts["Target_Gene"] = df_counts["Target_Gene"].
        ↪replace("Non-Targeting-Mouse", "NT")

       # Ensure there are no NaN values in the target_gene column before counting␣
        ↪repeats
       df_counts["Target_Gene"] = df_counts["Target_Gene"].fillna("Unknown")

       # Generate a sequential ID for each occurrence of target_gene
       df_counts["target_gene_repeat_ID"] = df_counts.groupby("Target_Gene").
        ↪cumcount() + 1

       # Format as "01", "02", "03", etc.
       df_counts["target_gene_repeat_ID"] = df_counts["target_gene_repeat_ID"].
        ↪astype(int).apply(lambda x: f"{x:02d}")
```

```
# Combine with target_gene to create a unique ID
df_counts["shRNA_ID"] = df_counts["Target_Gene"] + "_" +␣
  ↪df_counts["target_gene_repeat_ID"]

# Drop the temporary repeat ID column
df_counts = df_counts.drop(columns=["target_gene_repeat_ID"])

# Update the target_gene_ID column accordingly
df_counts["shRNA_ID"] = df_counts["Target_Gene"] + "_" + df_counts["shRNA_ID"].
  ↪str.split("_").str[-1]

# melt the dataframe to long format
df_long = df_counts.melt(id_vars=["Target_Gene", "shRNA_ID"],␣
  ↪var_name="Sample_ID", value_name="Read_Counts")
```

```
[162]: display(df_counts)
```

```
        D1    D2    D3    D4    D5    D6    D7    D8    D9   D10  …  D18-A  \
0     1399  2777  1312  2281  2056  1579  1072  3116  2532  3019  …   1411
1     1785  3277  1776  3414  3139  2482  1289  3877  3373  3953  …   2000
2      569  1270   539  1025  1075   892   484  1728  1201  1301  …    726
3      955  1703   912  1607  1714  1235   817  2199  1727  2160  …   1153
4     1150  2403  1177  2281  1981  1467  1055  2641  2071  2504  …   1415
…      …     …     …     …     …     …     …     …     …     …  …      …
2182   460   889   548   870   810   571   332  1141   935  1101  …    679
2183   725  1451   709  1416  1151  1017   483  1693  1601  1726  …    782
2184   547  1133   575  1052  1093   843   477  1365  1246  1497  …    771
2185  1354  2584  1371  2570  2230  1773  1047  3223  2530  3248  …   1610
2186   479   960   497   907   832   562   368  1276   813  1025  …    534

      D18-B   D19   D20   D21   D22   D23  Target_Gene  plasmid  shRNA_ID
0      1717  1682  2105  1482  1257  1965           NT     3776     NT_01
1      2510  2559  2959  2321  1782  2763           NT     4949     NT_02
2       804   774  1094   768   601   945           NT     1881     NT_03
3      1372  1268  1481  1010   855  1385           NT     2748     NT_04
4      1599  1556  2025  1541  1126  1868           NT     3560     NT_05
…        …     …     …     …     …     …          …        …        …
2182    758   709   892   658   475   859         Zeb1     1552   Zeb1_06
2183    970   933  1242   931   757  1071         Zeb1     2043   Zeb1_07
2184    868   820  1157   698   637   846         Zeb1     2101   Zeb1_08
2185   1877  1745  2160  1681  1329  1973         Zeb1     4534   Zeb1_09
2186    597   583   787   609   474   691         Zeb1     1383   Zeb1_10

[2187 rows x 27 columns]
```

```
[163]: display(df_long)
```

```
      Target_Gene shRNA_ID Sample_ID  Read_Counts
0              NT    NT_01        D1         1399
1              NT    NT_02        D1         1785
2              NT    NT_03        D1          569
3              NT    NT_04        D1          955
4              NT    NT_05        D1         1150
…             …        …       …          …
54670        Zeb1  Zeb1_06   plasmid         1552
54671        Zeb1  Zeb1_07   plasmid         2043
54672        Zeb1  Zeb1_08   plasmid         2101
54673        Zeb1  Zeb1_09   plasmid         4534
54674        Zeb1  Zeb1_10   plasmid         1383

[54675 rows x 4 columns]
```

1.5.1 Total number of barcodes per sample / condition

[165]:
```
df_full = df_long.merge(df_sd[["Sample_ID", "Sample_Description", "Day_Tx",␣
↪"Tx", "Dox", "Replicate"]],
                        left_on = "Sample_ID", right_on = "Sample_ID", how =␣
↪"left")
# df_full.to_csv("~/Documents/Projects/Multi_shRNA_screening_AP009/data/
↪long_format_joint_count_data.csv", index = False)
df_full.to_csv(os.path.join(proj_dir, "data/long_format_joint_count_data.csv"),␣
↪index = False)
```

[166]:
```
display(df_full)
print(df_full["Target_Gene"].value_counts())
print(df_full["Sample_Description"].value_counts())
```

```
      Target_Gene shRNA_ID Sample_ID  Read_Counts Sample_Description Day_Tx  \
0              NT    NT_01        D1         1399       T13_Dox_0.6nM      9
1              NT    NT_02        D1         1785       T13_Dox_0.6nM      9
2              NT    NT_03        D1          569       T13_Dox_0.6nM      9
3              NT    NT_04        D1          955       T13_Dox_0.6nM      9
4              NT    NT_05        D1         1150       T13_Dox_0.6nM      9
…             …        …       …          …                  …     …
54670        Zeb1  Zeb1_06   plasmid         1552             plasmid    NaN
54671        Zeb1  Zeb1_07   plasmid         2043             plasmid    NaN
54672        Zeb1  Zeb1_08   plasmid         2101             plasmid    NaN
54673        Zeb1  Zeb1_09   plasmid         4534             plasmid    NaN
54674        Zeb1  Zeb1_10   plasmid         1383             plasmid    NaN

        Tx  Dox Replicate
0      Low    Y         2
1      Low    Y         2
2      Low    Y         2
3      Low    Y         2
```

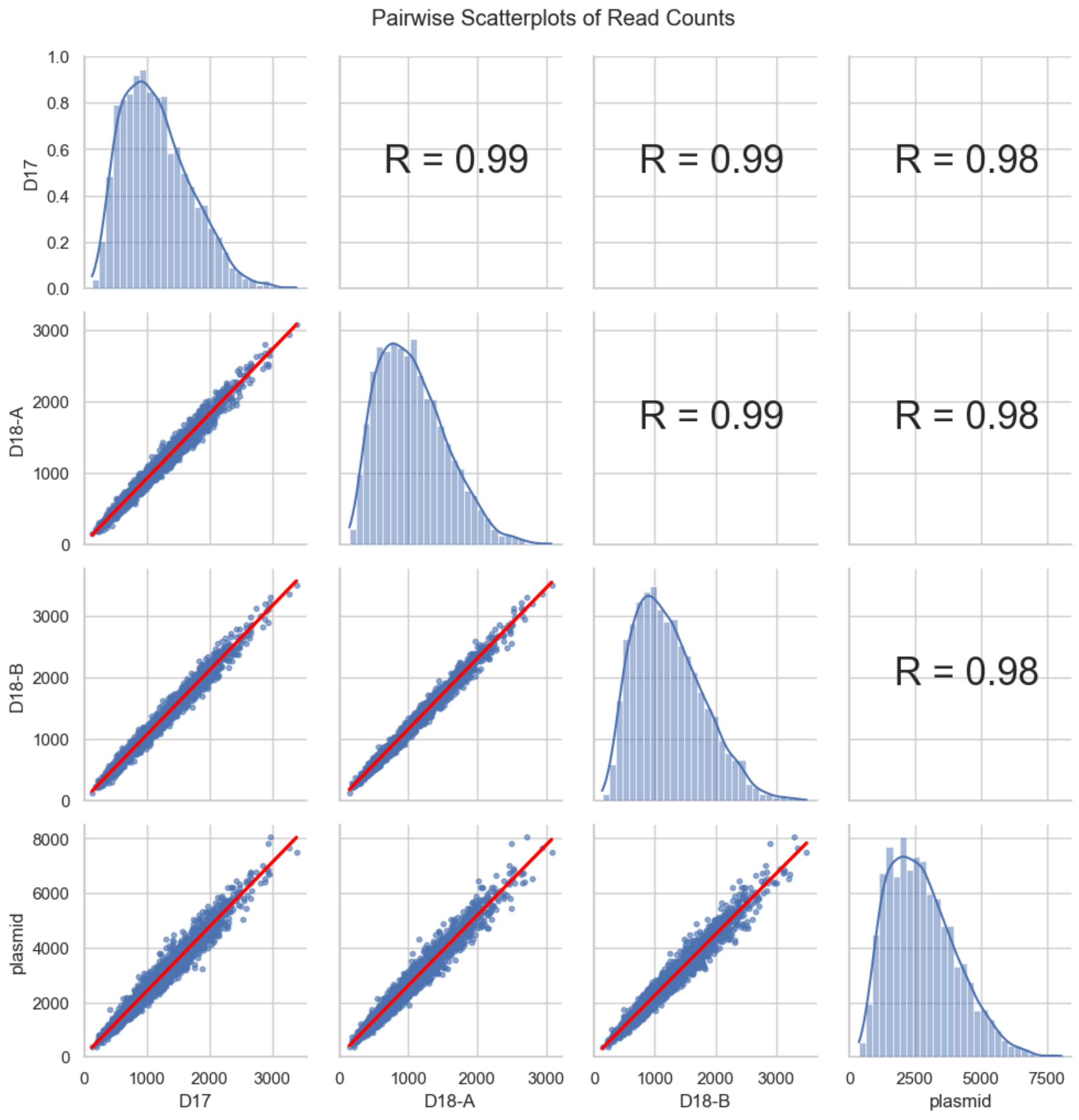
```
4      Low    Y          2
…    …  …        …
54670  NaN  NaN       NaN
54671  NaN  NaN       NaN
54672  NaN  NaN       NaN
54673  NaN  NaN       NaN
54674  NaN  NaN       NaN

[54675 rows x 9 columns]
```

```
NT                 5000
Pdgfra              250
Nf1                 250
Nf2                 250
Nfe2l2              250
                   …
Erbb2               250
Erbb3               250
Ern1                250
Zeb1                250
Cdkn2a(Ink4a)       175
Name: Target_Gene, Length: 200, dtype: int64
T7_NoDox_NoTx      6561
T4_Dox_NoTx        6561
T13_Dox_0.6nM      4374
T13_Dox_3.5nM      4374
T13_Dox_Vehicle    4374
T10_Dox_0.6nM      4374
T10_Dox_3.5nM      4374
T10_Dox_Vehicle    4374
T7_Dox_0.6nM       4374
T7_Dox_3.5nM       4374
T7_Dox_Vehicle     4374
plasmid            2187
Name: Sample_Description, dtype: int64
```

1.5.2 Count correlation between T7_NoDox_NoTx (D17, D18-A, D18-B) and plasmid

[178]:
```
df_full = pd.read_csv(os.path.join(proj_dir, "data/long_format_joint_count_data.
 ↪csv"))

# Pivot the data to wide format for pairwise comparisons using correct column␣
 ↪name
df_pivot = df_full.pivot_table(index="shRNA_ID", columns="Sample_ID",␣
 ↪values="Read_Counts", aggfunc="sum")

# Select the sample IDs of interest
```

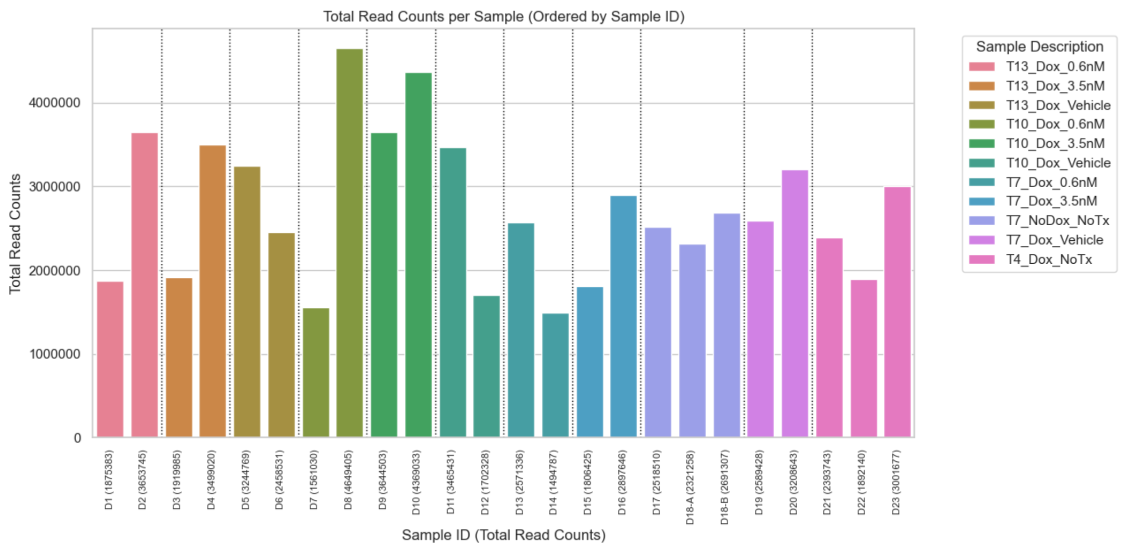
```python
sample_ids = ["D17", "D18-A", "D18-B", "plasmid"]

# Keep only the relevant columns and drop any missing data
df_selected = df_pivot[sample_ids].dropna()

def corrfunc(x, y, **kws):
    r = np.corrcoef(x, y)[0, 1]
    ax = plt.gca()
    ax.annotate(f"R = {r:.2f}", xy=(0.2, 0.5), xycoords=ax.transAxes,␣
 ↪fontsize=25)

sns.set(style="whitegrid")
g = sns.PairGrid(df_selected)
g.map_lower(sns.regplot, scatter_kws={'s': 10, 'alpha': 0.6}, line_kws={"color":
 ↪ "red"})
g.map_diag(sns.histplot, bins=30, kde=True)
g.map_upper(corrfunc)

for i in range(len(sample_ids)):
    for j in range(len(sample_ids)):
        ax = g.axes[i, j]
        if ax is not None:
            ax.set_xlim(left=0)
            ax.set_ylim(bottom=0)

plt.suptitle("Pairwise Scatterplots of Read Counts", y=1.02)

# Show the plot
plt.show()
```



1.5.3 Total read counts per sample

[179]:
```
df_full = pd.read_csv(os.path.join(proj_dir, "data/long_format_joint_count_data.
 ↪csv"))

df_counts = df_full[df_full["Sample_ID"] != "plasmid"]

# Aggregate total read counts per sample
df_sample_counts_simple = df_counts.groupby(["Sample_ID",␣
 ↪"Sample_Description"])["Read_Counts"].sum().reset_index()
```

```
# Ensure Sample_ID is sorted numerically rather than lexicographically
df_sample_counts_simple["Sample_ID_Sort"] =␣
 ↪df_sample_counts_simple["Sample_ID"].str.extract('(\d+)').astype(int)
df_sample_counts_simple = df_sample_counts_simple.
 ↪sort_values(by="Sample_ID_Sort").reset_index()

# Modify Sample_ID labels to include total counts in parentheses
df_sample_counts_simple["Sample_ID_Label"] = df_sample_counts_simple.apply(
    lambda row: f"{row['Sample_ID']} ({int(row['Read_Counts'])})", axis=1
)

# Plot the bar chart with modified x-axis labels
plt.figure(figsize=(12, 6))
ax = sns.barplot(data=df_sample_counts_simple, x="Sample_ID_Label",␣
 ↪y="Read_Counts", hue="Sample_Description", dodge=False)

# Identify group transitions for adding vertical dotted lines
prev_desc = None
for index, row in df_sample_counts_simple.iterrows():
    current_desc = row["Sample_Description"]
    if prev_desc is not None and prev_desc != current_desc:
        plt.axvline(x=index - 0.5, color="black", linestyle="dotted",␣
 ↪linewidth=1)  # Add vertical separator
    prev_desc = current_desc

# Set y-axis to exact number format
plt.ticklabel_format(style='plain', axis='y')  # Disable scientific notation

plt.xticks(rotation=90, fontsize=8)
plt.xlabel("Sample ID (Total Read Counts)")
plt.ylabel("Total Read Counts")
plt.title("Total Read Counts per Sample (Ordered by Sample ID)")
plt.legend(title="Sample Description", bbox_to_anchor=(1.05, 1), loc='upper␣
 ↪left')  # Move legend outside
plt.show()
```



1.6 Non-targeting shRNAs quantification and selection

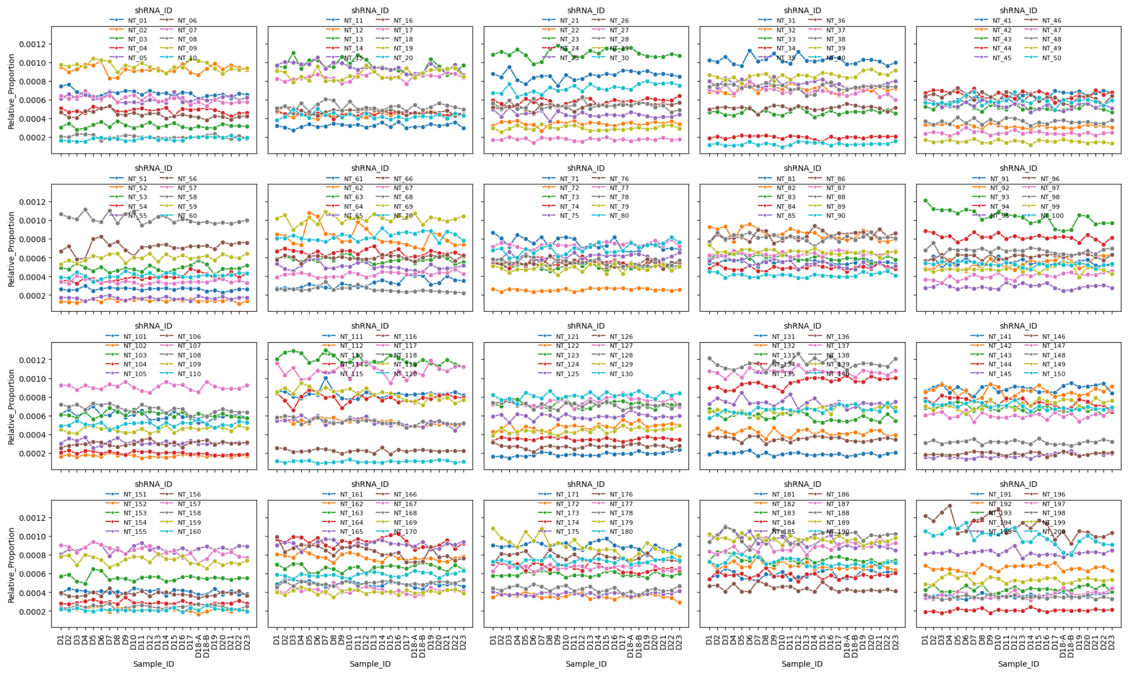
Assuming that for each non-targeting shRNA, its reads proportion should be minimally variable among replicates of each experimental condition (as denoted by `Sample_Description` field of `df_full`

[16]:
```python
# Reload full count table
# df_full_path = "~/Documents/Projects/Multi_shRNA_screening_AP009/data/
↪long_format_joint_count_data.csv"
df_full_path = os.path.join(proj_dir, "data/long_format_joint_count_data.csv")

df_full = pd.read_csv(df_full_path)

# Filter for only NT (Non-Targeting) shRNAs
df_nt = df_full[df_full["Target_Gene"] == "NT"]

# Compute total counts per sample
df_total_counts = df_full.groupby("Sample_ID")["Read_Counts"].sum().
↪reset_index()
df_total_counts = df_total_counts.rename(columns={"Read_Counts":␣
↪"Total_Read_Counts"})

# Merge total counts back to the NT dataset
df_nt = df_nt.merge(df_total_counts, on="Sample_ID", how="left")

# Compute relative proportion for each shRNA within each sample
df_nt["Relative_Proportion"] = df_nt["Read_Counts"] / df_nt["Total_Read_Counts"]
```

```
[17]: display(df_nt)
      print(df_nt["Target_Gene"].value_counts())
```

```
     Target_Gene shrNA_ID Sample_ID  Read_Counts Sample_Description  Day_Tx  \
0             NT    NT_01        D1         1399       T13_Dox_0.6nM       9
1             NT    NT_02        D1         1785       T13_Dox_0.6nM       9
2             NT    NT_03        D1          569       T13_Dox_0.6nM       9
3             NT    NT_04        D1          955       T13_Dox_0.6nM       9
4             NT    NT_05        D1         1150       T13_Dox_0.6nM       9
…           …      …         …           …                …      …
4795          NT   NT_196       D23         3103         T4_Dox_NoTx       0
4796          NT   NT_197       D23         1325         T4_Dox_NoTx       0
4797          NT   NT_198       D23          991         T4_Dox_NoTx       0
4798          NT   NT_199       D23         1613         T4_Dox_NoTx       0
4799          NT   NT_200       D23         2721         T4_Dox_NoTx       0

        Tx Dox  Replicate  Total_Read_Counts  Relative_Proportion
0      Low   Y          2            1875383             0.000746
1      Low   Y          2            1875383             0.000952
2      Low   Y          2            1875383             0.000303
3      Low   Y          2            1875383             0.000509
4      Low   Y          2            1875383             0.000613
…     …  ..        …                …                   …
4795  None   Y          1            3001677             0.001034
4796  None   Y          1            3001677             0.000441
4797  None   Y          1            3001677             0.000330
4798  None   Y          1            3001677             0.000537
4799  None   Y          1            3001677             0.000906

[4800 rows x 11 columns]

NT    4800
Name: Target_Gene, dtype: int64
```

```
[18]: # Get unique shRNA_IDs
      unique_shRNAs = df_nt["shRNA_ID"].unique()

      # Create a 4x5 subplot grid
      fig, axes = plt.subplots(4, 5, figsize=(20, 12), sharex=True, sharey=True)
      axes = axes.flatten()  # Flatten 2D array of subplots

      # Plot in batches of 10 shRNAs per subplot
      batch_size = 10
      for i in range(0, len(unique_shRNAs), batch_size):
          shRNA_subset = unique_shRNAs[i:i + batch_size]
          ax = axes[i // batch_size]

          sns.lineplot(data=df_nt[df_nt["shRNA_ID"].isin(shRNA_subset)],
                       x="Sample_ID", y="Relative_Proportion", hue="shRNA_ID",␣
      ↪marker="o", ax=ax)

          ax.tick_params(axis='x', rotation=90)

          # Move legend on top of each subplot and shrink marker size to avoid clutter
          legend = ax.legend(title="shRNA_ID", bbox_to_anchor=(0.5, 1.2), loc="upper␣
      ↪center", fontsize=8, ncol=2, frameon=False)
          for line in legend.get_lines():
              line.set_markersize(4)  # Reduce marker size

      # Adjust layout
      plt.tight_layout()
      plt.show()
```



Quantifying CV of reads proportion for each NT shRNA among samples within each experimental condition of `Sample_Description`

[19]:
```
# Group by shRNA within each Sample_Description
df_nt_variation = df_nt.groupby(["shRNA_ID",␣
↪"Sample_Description"])["Relative_Proportion"].agg(
    Mean_Proportion="mean",
    Std_Proportion="std"
).reset_index()
```

```
# Compute Coefficient of Variation (CV)
df_nt_variation["CV"] = df_nt_variation["Std_Proportion"] /␣
 ↪df_nt_variation["Mean_Proportion"]
```

[20]:
```
display(df_nt_variation)
# df_nt_variation.to_csv("~/Documents/Projects/Multi_shRNA_screening_AP009/data/
 ↪non_targeting_shrna_variation_table.csv", index = False)
df_nt_variation.to_csv(os.path.join(proj_dir, "data/
 ↪non_targeting_shrna_variation_table.csv"), index = False)
```

```
     shRNA_ID Sample_Description  Mean_Proportion  Std_Proportion        CV
0       NT_01       T10_Dox_0.6nM         0.000678        0.000012  0.017231
1       NT_01       T10_Dox_3.5nM         0.000693        0.000003  0.003822
2       NT_01     T10_Dox_Vehicle         0.000673        0.000028  0.041265
3       NT_01       T13_Dox_0.6nM         0.000753        0.000010  0.013204
4       NT_01       T13_Dox_3.5nM         0.000668        0.000022  0.033302
…       …                 …                 …               …        …
2195    NT_99         T4_Dox_NoTx         0.000457        0.000028  0.061696
2196    NT_99        T7_Dox_0.6nM         0.000470        0.000032  0.068284
2197    NT_99        T7_Dox_3.5nM         0.000461        0.000037  0.080397
2198    NT_99      T7_Dox_Vehicle         0.000444        0.000075  0.167892
2199    NT_99       T7_NoDox_NoTx         0.000531        0.000020  0.037855

[2200 rows x 5 columns]
```

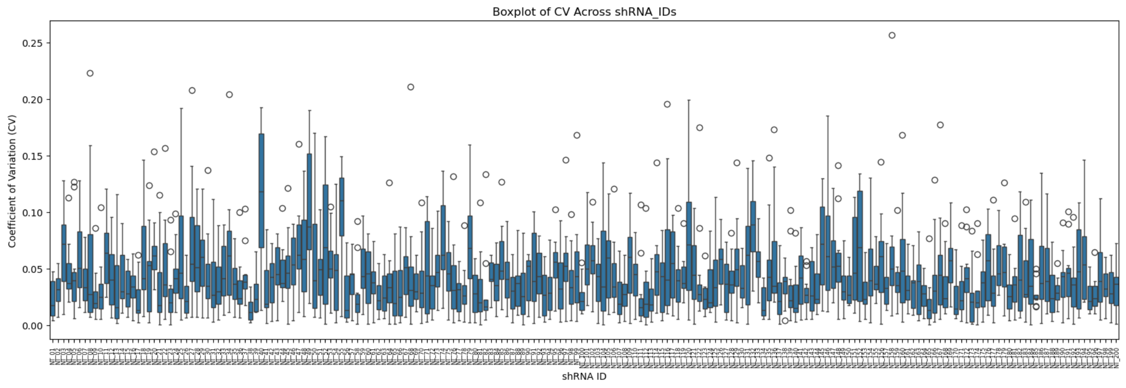
[21]:
```
display(df_nt_variation[df_nt_variation["Sample_Description"] == "T10_Dox_0.
 ↪6nM"])
```

```
     shRNA_ID Sample_Description  Mean_Proportion  Std_Proportion        CV
0       NT_01       T10_Dox_0.6nM         0.000678        0.000012  0.017231
11      NT_02       T10_Dox_0.6nM         0.000830        0.000006  0.006931
22      NT_03       T10_Dox_0.6nM         0.000341        0.000044  0.127808
33      NT_04       T10_Dox_0.6nM         0.000498        0.000036  0.071551
44      NT_05       T10_Dox_0.6nM         0.000622        0.000076  0.122570
…       …                 …                 …               …        …
2145    NT_95       T10_Dox_0.6nM         0.000277        0.000010  0.035703
2156    NT_96       T10_Dox_0.6nM         0.000595        0.000042  0.070011
2167    NT_97       T10_Dox_0.6nM         0.000403        0.000026  0.063755
2178    NT_98       T10_Dox_0.6nM         0.000689        0.000017  0.025127
2189    NT_99       T10_Dox_0.6nM         0.000474        0.000003  0.007054

[200 rows x 5 columns]
```

Boxplot of CV of `df_nt_variation` across NT shRNAs

[22]:
```
# Extract numeric part of shRNA_ID and sort sequentially from 1 to 200
df_nt_variation["shRNA_Seq"] = df_nt_variation["shRNA_ID"].str.
 ↪extract(r'(\d+)').astype(int)
```

```
df_nt_variation = df_nt_variation.sort_values(by="shRNA_Seq")

# Create a wider boxplot with ordered shRNA_IDs
plt.figure(figsize=(20, 6))
sns.boxplot(data=df_nt_variation, x="shRNA_ID", y="CV")

# Customize the plot
plt.xticks(rotation=90, fontsize=6)  # Smaller font size for x-axis labels
plt.xlabel("shRNA ID")
plt.ylabel("Coefficient of Variation (CV)")
plt.title("Boxplot of CV Across shRNA_IDs")

# Show the plot
plt.show()
```

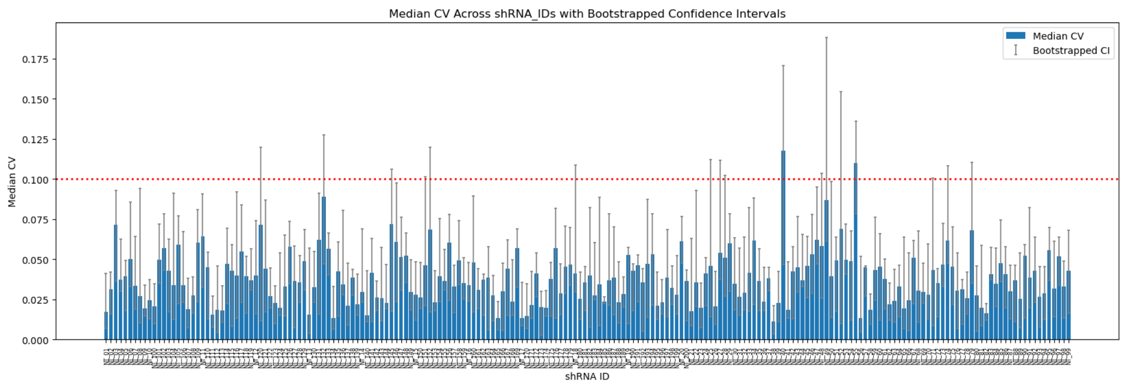


Barchart of Median CV of `df_nt_variation` across NT shRNAs with bootstrapped confience internal

[23]:
```
# Compute median and bootstrapped confidence intervals for each shRNA_ID
df_cv_stats = df_nt_variation.groupby("shRNA_ID")["CV"].agg(
    Median_CV="median"
).reset_index()

# Apply bootstrapping for confidence intervals
df_cv_stats["Lower_CI"], df_cv_stats["Upper_CI"] = zip(*df_nt_variation.
 ↪groupby("shRNA_ID")["CV"].apply(lambda x: bootstrap_median_ci(x)))

# Calculate error bars (difference between median and lower/upper bounds)
df_cv_stats["Error_Lower"] = df_cv_stats["Median_CV"] - df_cv_stats["Lower_CI"]
df_cv_stats["Error_Upper"] = df_cv_stats["Upper_CI"] - df_cv_stats["Median_CV"]
```

[24]:
```
# Create a bar chart with bootstrapped confidence intervals
plt.figure(figsize=(20, 6))
```

```
# Plot bars for Median_CV
plt.bar(df_cv_stats["shRNA_ID"], df_cv_stats["Median_CV"], label="Median CV")

# Add error bars with small caps at both ends
plt.errorbar(df_cv_stats["shRNA_ID"], df_cv_stats["Median_CV"],
             yerr=[df_cv_stats["Error_Lower"], df_cv_stats["Error_Upper"]],
             fmt='none', ecolor="gray", elinewidth=1.2, capsize=1.5, capthick=1.
 ↪2, label="Bootstrapped CI")
plt.axhline(y=0.1, color='red', linestyle='dotted', linewidth=2)
# Customize the plot
plt.xticks(rotation=90, fontsize=6)
plt.xlabel("shRNA ID")
plt.ylabel("Median CV")
plt.title("Median CV Across shRNA_IDs with Bootstrapped Confidence Intervals")
plt.legend()

# Show the plot
plt.show()
```



There are 183 NT shRNA whose bootstrapped confidence internals of median CVs < 0.1

```
[25]: df_selected_nt_boot_median = df_cv_stats[(df_cv_stats["Upper_CI"] < 0.1)]
display(df_selected_nt_boot_median)
# df_selected_nt_boot_median.to_csv("~/Documents/Projects/
 ↪Multi_shRNA_screening_AP009/data/
 ↪selected_non_targeting_shRNA_table_median_cv_bootstrapped.csv", index =␣
 ↪False)
df_selected_nt_boot_median.to_csv(os.path.join(proj_dir, "data/
 ↪selected_non_targeting_shRNA_table_median_cv_bootstrapped.csv"), index =␣
 ↪False)
```

```
    shRNA_ID  Median_CV  Lower_CI  Upper_CI  Error_Lower  Error_Upper
0      NT_01   0.017231  0.007015  0.041265     0.010216     0.024034
```

```
1      NT_02   0.031408  0.016015  0.042011      0.015393      0.010603
2      NT_03   0.071599  0.036633  0.092930      0.034966      0.021331
3      NT_04   0.037188  0.029977  0.062541      0.007211      0.025353
4      NT_05   0.039316  0.017949  0.049489      0.021367      0.010173
..       …        …        …        …          …          …
195    NT_95   0.055613  0.036926  0.069820      0.018688      0.014206
196    NT_96   0.031795  0.014230  0.061475      0.017565      0.029680
197    NT_97   0.051950  0.033848  0.063755      0.018101      0.011805
198    NT_98   0.033246  0.013237  0.048943      0.020009      0.015697
199    NT_99   0.043054  0.016347  0.068284      0.026707      0.025230

[183 rows x 6 columns]
```

Quantifying overall variability for each NT shRNA by summarizing mean, median, inter-quartile-range, and max CVs, where * mean is the average across all conditions * median is more robust to outliers * IQR measures spread * max captures extreme values

[26]:
```
df_nt_cv_summary = df_nt_variation.groupby("shRNA_ID")["CV"].agg(
    Mean_CV="mean",
    Median_CV="median",
    IQR_CV=lambda x: x.quantile(0.75) - x.quantile(0.25),  # Interquartile Range
    Max_CV="max"
).reset_index()
```

[27]:
```
display(df_nt_cv_summary)
```

```
    shRNA_ID   Mean_CV  Median_CV    IQR_CV    Max_CV
0      NT_01  0.023505   0.017231  0.030772  0.047327
1      NT_02  0.031011   0.031408  0.019975  0.054954
2      NT_03  0.066902   0.071599  0.050576  0.127808
3      NT_04  0.046917   0.037188  0.022093  0.112538
4      NT_05  0.047722   0.039316  0.025738  0.126396
..       …        …         …        …        …
195    NT_95  0.054365   0.055613  0.021983  0.102406
196    NT_96  0.037031   0.031795  0.041091  0.092345
197    NT_97  0.056563   0.051950  0.024747  0.146140
198    NT_98  0.037734   0.033246  0.029870  0.097928
199    NT_99  0.051712   0.043054  0.045255  0.167892

[200 rows x 5 columns]
```

From a total of 200 non-targeting shRNAs Selecting 49 that have * low Mean_CV (ensuring overall low variability) * low IQR_CV (ensuring tight distribution of variability) * filtered out cases of extreme outliers (in case Max_CV >= 3 x Median_CV)

[28]:
```
mean_cv_threshold = df_nt_cv_summary["Mean_CV"].quantile(0.5)
iqr_cv_threshold = df_nt_cv_summary["IQR_CV"].quantile(0.5)
max_cv_threshold = df_nt_cv_summary["Max_CV"].quantile(0.8)
```

```
df_selected_nt = df_nt_cv_summary[
    (df_nt_cv_summary["Mean_CV"] < mean_cv_threshold) &
    (df_nt_cv_summary["IQR_CV"] < iqr_cv_threshold) &
    (df_nt_cv_summary["Max_CV"] < 3 * df_nt_cv_summary["Median_CV"])
]

print(df_selected_nt.shape)
```

```
(49, 5)
```

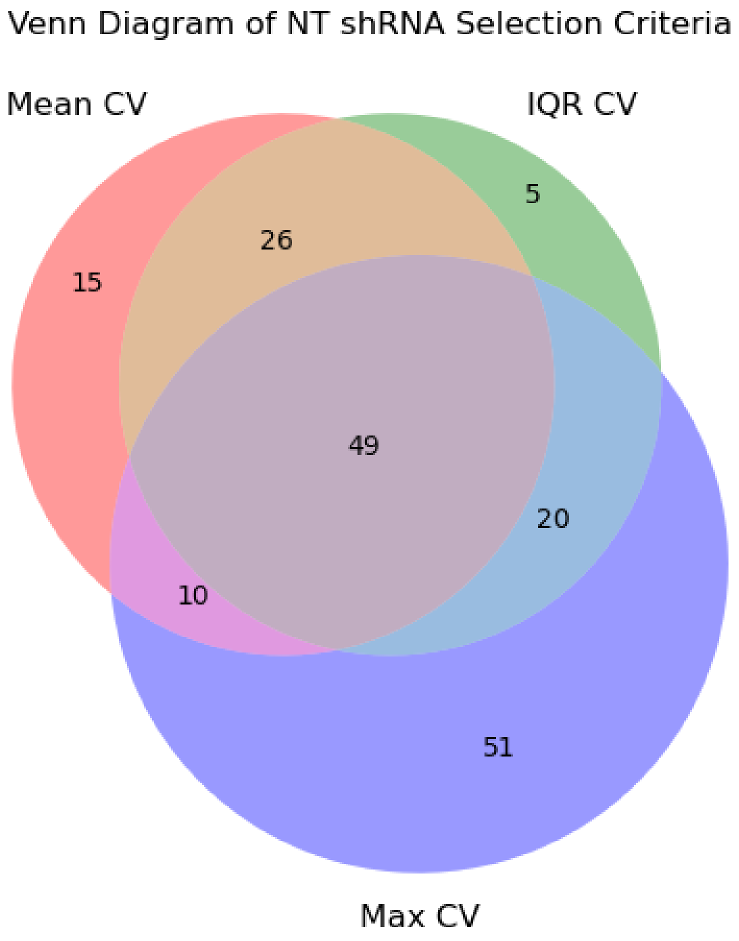
[29]:
```
display(df_nt_variation)
```

```
     shRNA_ID Sample_Description  Mean_Proportion  Std_Proportion        CV  \
0       NT_01        T10_Dox_0.6nM         0.000678        0.000012  0.017231
1       NT_01        T10_Dox_3.5nM         0.000693        0.000003  0.003822
2       NT_01      T10_Dox_Vehicle         0.000673        0.000028  0.041265
3       NT_01        T13_Dox_0.6nM         0.000753        0.000010  0.013204
4       NT_01        T13_Dox_3.5nM         0.000668        0.000022  0.033302
…         …                  …              …               …         …
1322   NT_200      T10_Dox_Vehicle         0.001081        0.000013  0.011939
1321   NT_200        T10_Dox_3.5nM         0.001017        0.000002  0.001932
1320   NT_200        T10_Dox_0.6nM         0.000998        0.000055  0.054743
1324   NT_200        T13_Dox_3.5nM         0.001059        0.000046  0.043319
1329   NT_200       T7_Dox_Vehicle         0.000975        0.000041  0.041774

      shRNA_Seq
0             1
1             1
2             1
3             1
4             1
…           …
1322        200
1321        200
1320        200
1324        200
1329        200

[2200 rows x 6 columns]
```

[180]:
```
# df_selected_nt.to_csv("~/Documents/Projects/Multi_shRNA_screening_AP009/data/
↪selected_non_targeting_shRNA_table.csv", index = False)
df_selected_nt.to_csv(os.path.join(proj_dir, "data/
↪selected_non_targeting_shRNA_table.csv"), index = False)
```

[31]:
```
# Define sets for each selection criterion
set_mean_cv = set(df_nt_cv_summary[df_nt_cv_summary["Mean_CV"] <␣
↪mean_cv_threshold]["shRNA_ID"])
```

```
set_iqr_cv = set(df_nt_cv_summary[df_nt_cv_summary["IQR_CV"] <␣
↪iqr_cv_threshold]["shRNA_ID"])
set_max_cv = set(df_nt_cv_summary[df_nt_cv_summary["Max_CV"] < 3 *␣
↪df_nt_cv_summary["Median_CV"]]["shRNA_ID"])

# Create Venn diagram
plt.figure(figsize=(6, 6))
venn = venn3([set_mean_cv, set_iqr_cv, set_max_cv], ('Mean CV', 'IQR CV', 'Max␣
↪CV'))

# Customize colors and labels
plt.title("Venn Diagram of NT shRNA Selection Criteria")
plt.show()
```



```
[ ]:
```

1.7 RTN (relative tumor-cell number)

- For each shRNA (`shRNA_ID`) of each target gene (`Target_Gene`) of each condition (`Sample_Description`), RTN is defined and calculated as ratio of total read counts of the shRNA of the target gene divided by total read counts of (selected) non-target genes
  - Remove sample `D18-B`, technical replicate of `D18-A`, from downstream analytics
  - Within each sample for relative abundance divide each read count (per shRNA per target gene) by total read counts of the sample
  - Using bootstrapped confidence internals of median CVs < 0.1 as acceptance criteria of selecting 183 NT shRNAs
  - To harmonize replicates for each condition, compute mean of `Read_Counts` for each `shRNA_ID` of each `Target_Gene`
  - Normalizing cell counts of each condition to the reference condition of `T7_NoDox_NoTx`

1.7.1 Remove plasmid and D18-B

```
[221]: df_full = pd.read_csv(os.path.join(proj_dir, "data/long_format_joint_count_data.
        ↪csv"))

       # Remove sample `D18-B`, technical replicate of `D18-A`, from downstream␣
        ↪analytics
       df_counts = df_full[~df_full["Sample_ID"].isin(["plasmid", "D18-B"])]
```

1.7.2 Normalize total read counts per sample

```
[222]: # Normalize total read counts per sample
       df_total_counts = df_counts.groupby("Sample_ID")["Read_Counts"].sum().
        ↪reset_index()
       df_total_counts = df_total_counts.rename(columns={"Read_Counts":␣
        ↪"Total_Read_Counts"})

       # merge total counts back to df_counts
       df_counts = df_counts.merge(df_total_counts, on = "Sample_ID", how = "left")

       # Calculate relative read counts
       df_counts["Relative_Read_Counts"] = df_counts["Read_Counts"] /␣
        ↪df_counts["Total_Read_Counts"]

       df_counts.drop(["Read_Counts", "Total_Read_Counts"], axis = 1, inplace = True)
       df_counts.rename(columns = {"Relative_Read_Counts": "Read_Counts"}, inplace =␣
        ↪True)

       display(df_counts)
```

```
      Target_Gene shRNA_ID Sample_ID Sample_Description  Day_Tx   Tx Dox  \
0              NT    NT_01        D1       T13_Dox_0.6nM     9.0  Low   Y
1              NT    NT_02        D1       T13_Dox_0.6nM     9.0  Low   Y
2              NT    NT_03        D1       T13_Dox_0.6nM     9.0  Low   Y
```

```
3               NT    NT_04         D1      T13_Dox_0.6nM     9.0   Low    Y
4               NT    NT_05         D1      T13_Dox_0.6nM     9.0   Low    Y
…            …      …       …              …     …    …   ..
50296         Zeb1  Zeb1_06       D23        T4_Dox_NoTx     0.0  None    Y
50297         Zeb1  Zeb1_07       D23        T4_Dox_NoTx     0.0  None    Y
50298         Zeb1  Zeb1_08       D23        T4_Dox_NoTx     0.0  None    Y
50299         Zeb1  Zeb1_09       D23        T4_Dox_NoTx     0.0  None    Y
50300         Zeb1  Zeb1_10       D23        T4_Dox_NoTx     0.0  None    Y

       Replicate  Read_Counts
0            2.0     0.000746
1            2.0     0.000952
2            2.0     0.000303
3            2.0     0.000509
4            2.0     0.000613
…          …          …
50296        1.0     0.000286
50297        1.0     0.000357
50298        1.0     0.000282
50299        1.0     0.000657
50300        1.0     0.000230

[50301 rows x 9 columns]
```

1.7.3 Select NT shRNAs

[223]:
```
df_full_nt_sel = df_counts.copy()

# Using bootstrapped confidence internals of median CVs < 0.1 as acceptance␣
 ↪criteria of selecting 183 NT shRNAs
df_nt_filtering = df_selected_nt_boot_median

df_full_nt_sel = df_full_nt_sel[
    (df_full_nt_sel["Target_Gene"] != "NT") |  # Keep all non-NT genes
    ((df_full_nt_sel["Target_Gene"] == "NT") & df_full_nt_sel["shRNA_ID"].
 ↪isin(df_nt_filtering["shRNA_ID"]))  # Keep only selected NT shRNAs
]

display(df_full_nt_sel)
```

```
      Target_Gene shRNA_ID Sample_ID Sample_Description  Day_Tx    Tx Dox  \
0               NT    NT_01         D1      T13_Dox_0.6nM     9.0   Low    Y
1               NT    NT_02         D1      T13_Dox_0.6nM     9.0   Low    Y
2               NT    NT_03         D1      T13_Dox_0.6nM     9.0   Low    Y
3               NT    NT_04         D1      T13_Dox_0.6nM     9.0   Low    Y
4               NT    NT_05         D1      T13_Dox_0.6nM     9.0   Low    Y
…            …      …       …              …     …    …   ..
50296         Zeb1  Zeb1_06       D23        T4_Dox_NoTx     0.0  None    Y
```

```
50297         Zeb1   Zeb1_07       D23        T4_Dox_NoTx     0.0   None    Y
50298         Zeb1   Zeb1_08       D23        T4_Dox_NoTx     0.0   None    Y
50299         Zeb1   Zeb1_09       D23        T4_Dox_NoTx     0.0   None    Y
50300         Zeb1   Zeb1_10       D23        T4_Dox_NoTx     0.0   None    Y

       Replicate  Read_Counts
0            2.0     0.000746
1            2.0     0.000952
2            2.0     0.000303
3            2.0     0.000509
4            2.0     0.000613
…             …           …
50296        1.0     0.000286
50297        1.0     0.000357
50298        1.0     0.000282
50299        1.0     0.000657
50300        1.0     0.000230

[49910 rows x 9 columns]
```

1.7.4 Mean read counts of each shRNA/target/condition

```
[224]: ## Compute mean of Read_Counts for each shRNA_ID of each Target_Gene within␣
        ↪each Sample_Description
       df_mean_counts = df_full_nt_sel.groupby(["Sample_Description", "Target_Gene",␣
        ↪"shRNA_ID", "Day_Tx", "Tx", "Dox"])\
           ["Read_Counts"].mean().reset_index()
       display(df_mean_counts)
```

```
      Sample_Description Target_Gene  shRNA_ID  Day_Tx    Tx Dox  Read_Counts
0           T10_Dox_0.6nM       Abcb1  Abcb1_01     6.0   Low   Y     0.000319
1           T10_Dox_0.6nM       Abcb1  Abcb1_02     6.0   Low   Y     0.000312
2           T10_Dox_0.6nM       Abcb1  Abcb1_03     6.0   Low   Y     0.000277
3           T10_Dox_0.6nM       Abcb1  Abcb1_04     6.0   Low   Y     0.000668
4           T10_Dox_0.6nM       Abcb1  Abcb1_05     6.0   Low   Y     0.000679
…                   …           …         …       …     …   ..           …
23865       T7_NoDox_NoTx        Zeb1   Zeb1_06     3.0  None   N     0.000290
23866       T7_NoDox_NoTx        Zeb1   Zeb1_07     3.0  None   N     0.000354
23867       T7_NoDox_NoTx        Zeb1   Zeb1_08     3.0  None   N     0.000320
23868       T7_NoDox_NoTx        Zeb1   Zeb1_09     3.0  None   N     0.000692
23869       T7_NoDox_NoTx        Zeb1   Zeb1_10     3.0  None   N     0.000233

[23870 rows x 7 columns]
```

1.7.5 Normalize shRNA read counts to reference condition of T7_NoDox_NoTx

[244]:
```
# reference condition
ref_condition = "T7_NoDox_NoTx"

df_ref = df_mean_counts[df_mean_counts["Sample_Description"] == ref_condition]
df_ref_selected = df_ref[["shRNA_ID", "Read_Counts"]]
df_ref_selected = df_ref_selected.rename(columns = {"Read_Counts":␣
  ↪"Ref_Read_Counts"})
# display(df_ref_selected)

# merge reference values back
df_normalized_mean_counts = df_mean_counts.merge(df_ref_selected, on =␣
  ↪"shRNA_ID", how = "left")
display(df_normalized_mean_counts)

# normalize read_counts
df_normalized_mean_counts["Read_Counts"] =␣
  ↪df_normalized_mean_counts["Read_Counts"] /␣
  ↪df_normalized_mean_counts["Ref_Read_Counts"]
df_normalized_mean_counts.drop("Ref_Read_Counts", axis = 1, inplace = True)
display(df_normalized_mean_counts)
```

```
      Sample_Description Target_Gene  shRNA_ID  Day_Tx    Tx Dox  Read_Counts  \
0           T10_Dox_0.6nM       Abcb1  Abcb1_01     6.0   Low   Y     0.000319
1           T10_Dox_0.6nM       Abcb1  Abcb1_02     6.0   Low   Y     0.000312
2           T10_Dox_0.6nM       Abcb1  Abcb1_03     6.0   Low   Y     0.000277
3           T10_Dox_0.6nM       Abcb1  Abcb1_04     6.0   Low   Y     0.000668
4           T10_Dox_0.6nM       Abcb1  Abcb1_05     6.0   Low   Y     0.000679
…                     …         …        …       …   …  ..         …
23865       T7_NoDox_NoTx        Zeb1   Zeb1_06     3.0  None   N     0.000290
23866       T7_NoDox_NoTx        Zeb1   Zeb1_07     3.0  None   N     0.000354
23867       T7_NoDox_NoTx        Zeb1   Zeb1_08     3.0  None   N     0.000320
23868       T7_NoDox_NoTx        Zeb1   Zeb1_09     3.0  None   N     0.000692
23869       T7_NoDox_NoTx        Zeb1   Zeb1_10     3.0  None   N     0.000233

       Ref_Read_Counts
0             0.000341
1             0.000336
2             0.000279
3             0.000653
4             0.000647
…                  …
23865         0.000290
23866         0.000354
23867         0.000320
23868         0.000692
23869         0.000233
```

```
[23870 rows x 8 columns]

      Sample_Description Target_Gene  shRNA_ID  Day_Tx    Tx Dox  Read_Counts
0            T10_Dox_0.6nM       Abcb1  Abcb1_01     6.0   Low   Y     0.933098
1            T10_Dox_0.6nM       Abcb1  Abcb1_02     6.0   Low   Y     0.929893
2            T10_Dox_0.6nM       Abcb1  Abcb1_03     6.0   Low   Y     0.991018
3            T10_Dox_0.6nM       Abcb1  Abcb1_04     6.0   Low   Y     1.023020
4            T10_Dox_0.6nM       Abcb1  Abcb1_05     6.0   Low   Y     1.049045
…                      …           …         …       …     …  ..           …
23865        T7_NoDox_NoTx        Zeb1   Zeb1_06     3.0  None   N     1.000000
23866        T7_NoDox_NoTx        Zeb1   Zeb1_07     3.0  None   N     1.000000
23867        T7_NoDox_NoTx        Zeb1   Zeb1_08     3.0  None   N     1.000000
23868        T7_NoDox_NoTx        Zeb1   Zeb1_09     3.0  None   N     1.000000
23869        T7_NoDox_NoTx        Zeb1   Zeb1_10     3.0  None   N     1.000000

[23870 rows x 7 columns]
```

1.7.6 Calculate RTNs (relative tumor-cell numbers)

[228]:
```
# Compute summed Read_Counts across all NT shRNA_IDs for each Sample_Description
df_nt_summed =␣
↪df_normalized_mean_counts[df_normalized_mean_counts["Target_Gene"] == "NT"]\
    .groupby("Sample_Description")["Read_Counts"].sum().reset_index()
display(df_nt_summed)
```

```
   Sample_Description  Read_Counts
0       T10_Dox_0.6nM   187.368448
1       T10_Dox_3.5nM   186.568467
2     T10_Dox_Vehicle   187.688389
3       T13_Dox_0.6nM   187.687578
4       T13_Dox_3.5nM   187.318488
5     T13_Dox_Vehicle   188.971069
6         T4_Dox_NoTx   185.723554
7        T7_Dox_0.6nM   185.639371
8        T7_Dox_3.5nM   186.266811
9      T7_Dox_Vehicle   186.601081
10      T7_NoDox_NoTx   183.000000
```

[237]:
```
df_rtn = df_normalized_mean_counts.merge(df_nt_summed, on =␣
↪"Sample_Description", suffixes = ("", "_NT"))

# Compute RTN
df_rtn["RTN"] = df_rtn["Read_Counts"] / df_rtn["Read_Counts_NT"]

# Drop the redundant NT read count column
df_rtn = df_rtn.drop(columns=["Read_Counts_NT"])
```

```
display(df_rtn)
# df_rtn.to_csv("~/Documents/Projects/Multi_shRNA_screening_AP009/data/
↪shrna_target_gene_relative_tumor_number_table.csv", index = False)
df_rtn.to_csv(os.path.join(proj_dir, "data/
↪shrna_target_gene_relative_tumor_number_table.csv"), index = False)
```

```
      Sample_Description Target_Gene  shRNA_ID  Day_Tx    Tx Dox  Read_Counts  \
0            T10_Dox_0.6nM       Abcb1  Abcb1_01     6.0   Low   Y     0.933098
1            T10_Dox_0.6nM       Abcb1  Abcb1_02     6.0   Low   Y     0.929893
2            T10_Dox_0.6nM       Abcb1  Abcb1_03     6.0   Low   Y     0.991018
3            T10_Dox_0.6nM       Abcb1  Abcb1_04     6.0   Low   Y     1.023020
4            T10_Dox_0.6nM       Abcb1  Abcb1_05     6.0   Low   Y     1.049045
…                      …          …         …       …    … ..           …
23865        T7_NoDox_NoTx        Zeb1   Zeb1_06     3.0  None   N     1.000000
23866        T7_NoDox_NoTx        Zeb1   Zeb1_07     3.0  None   N     1.000000
23867        T7_NoDox_NoTx        Zeb1   Zeb1_08     3.0  None   N     1.000000
23868        T7_NoDox_NoTx        Zeb1   Zeb1_09     3.0  None   N     1.000000
23869        T7_NoDox_NoTx        Zeb1   Zeb1_10     3.0  None   N     1.000000

            RTN
0      0.004980
1      0.004963
2      0.005289
3      0.005460
4      0.005599
…           …
23865  0.005464
23866  0.005464
23867  0.005464
23868  0.005464
23869  0.005464

[23870 rows x 8 columns]
```

[232]:
```
# Split shRNA_ID into two parts: the main ID and the repetition number
df_rtn["shRNA_Rep"] = df_rtn["shRNA_ID"].str.split("_").str[1]
```

1.8 Prognostic effects

A vs. B <-> Dox (target gene vs NT) vs. NoDox (target gene vs NT) * ratio of ratios - T13_Dox_Vehicle vs. T7_NoDox_NoTx

[233]:
```
# Define conditions for A and B
# condition_A = "T7_Dox_Vehicle"
condition_A = "T13_Dox_Vehicle"
condition_B = "T7_NoDox_NoTx"

# Include the Target_Gene field in both condition datasets before merging
```

```
df_A = df_rtn[df_rtn["Sample_Description"] == condition_A][["shRNA_ID",␣
↪"Target_Gene", "RTN"]].rename(columns={"RTN": "RTN_A"})
df_B = df_rtn[df_rtn["Sample_Description"] == condition_B][["shRNA_ID",␣
↪"Target_Gene", "RTN"]].rename(columns={"RTN": "RTN_B"})

# Merge the two datasets on shRNA_ID and Target_Gene
df_prognostic = df_A.merge(df_B, on=["shRNA_ID", "Target_Gene"], how="inner")

# Compute the prognostic effect as log2(RTN_A / RTN_B)
df_prognostic["Prognostic_Effect"] = np.log2(df_prognostic["RTN_A"] /␣
↪df_prognostic["RTN_B"])

# Define control gene lists
loss_of_representation_target_genes = ["Rpa1", "Rpa3", "Rps6", "Pcna", "Psmc5",␣
↪"Rbx1", "Ran", "Snrpd1", "Rpl7", "Kif11"]
neutral_control_target_genes = ["NT", "Trp53"]
gain_of_representation_target_genes = ["Pten"]

# Assign categories for sorting
df_prognostic["Gene_Category"] = "Other"  # Default category
df_prognostic.loc[df_prognostic["Target_Gene"].
↪isin(neutral_control_target_genes), "Gene_Category"] = "Neutral Control"
df_prognostic.loc[df_prognostic["Target_Gene"].
↪isin(loss_of_representation_target_genes), "Gene_Category"] = "Loss of␣
↪Representation"
df_prognostic.loc[df_prognostic["Target_Gene"].
↪isin(gain_of_representation_target_genes), "Gene_Category"] = "Gain of␣
↪Representation"

# Sort Target_Gene first by category, then alphabetically within each category
df_prognostic["Sort_Order"] = df_prognostic["Gene_Category"].map({"Neutral␣
↪Control": 1,
                                                                   "Loss of␣
↪Representation": 2,
                                                                   "Gain of␣
↪Representation": 3,
                                                                   "Other": 4})
df_prognostic = df_prognostic.sort_values(by=["Sort_Order", "Target_Gene"])
```

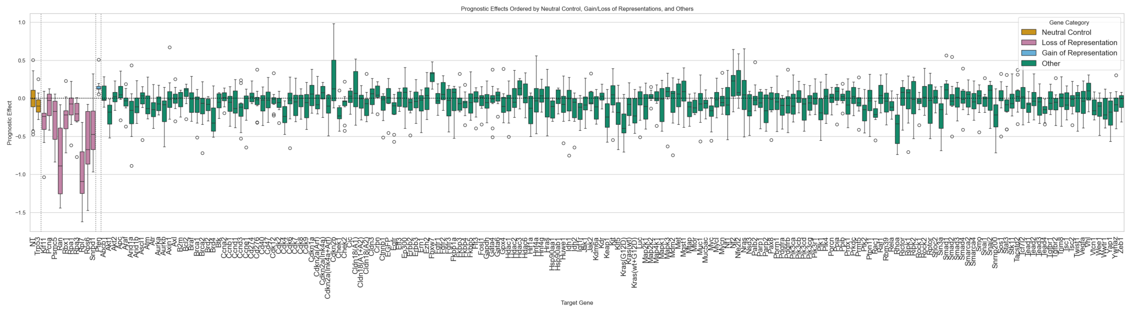
[234]:
```
display(df_prognostic)
df_prognostic.to_csv(os.path.join(proj_dir, "data/prognostic_effect.csv"),␣
↪index = False)
```

```
     shRNA_ID Target_Gene     RTN_A     RTN_B  Prognostic_Effect  \
1147    NT_01          NT  0.005291  0.005464          -0.046575
1148    NT_02          NT  0.005976  0.005464           0.129089
1149    NT_03          NT  0.006107  0.005464           0.160482
```

```
1150    NT_04          NT  0.005527  0.005464          0.016455
1151    NT_05          NT  0.005368  0.005464         -0.025699
…         …       …       …       …                …
2165  Zeb1_06        Zeb1  0.004400  0.005464         -0.312661
2166  Zeb1_07        Zeb1  0.005739  0.005464          0.070672
2167  Zeb1_08        Zeb1  0.005615  0.005464          0.039090
2168  Zeb1_09        Zeb1  0.005383  0.005464         -0.021580
2169  Zeb1_10        Zeb1  0.005513  0.005464          0.012882

        Gene_Category  Sort_Order
1147  Neutral Control           1
1148  Neutral Control           1
1149  Neutral Control           1
1150  Neutral Control           1
1151  Neutral Control           1
…                 …         …
2165            Other           4
2166            Other           4
2167            Other           4
2168            Other           4
2169            Other           4

[2170 rows x 7 columns]
```

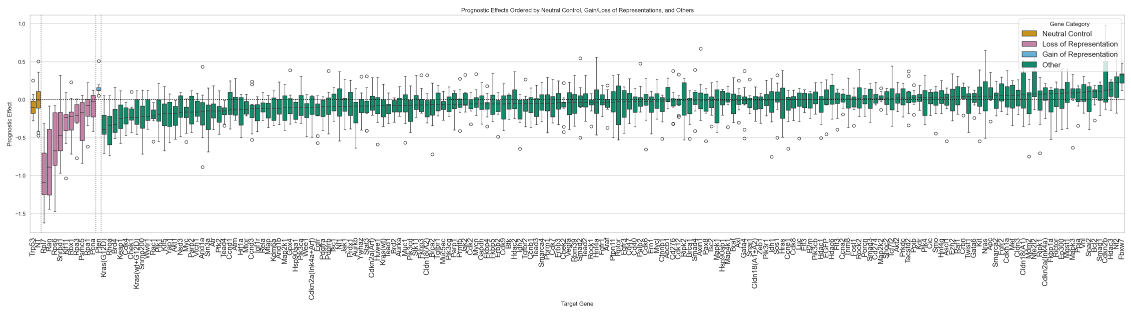
[235]:
```python
plt.figure(figsize=(40, 8))

# Use a distinct color palette for better differentiation
palette = {"Neutral Control": "#E69F00",
           "Gain of Representation": "#56B4E9",
           "Loss of Representation": "#CC79A7",
           "Other": "#009E73"}

# Create the boxplot
ax = sns.boxplot(data=df_prognostic, x="Target_Gene", y="Prognostic_Effect",␣
 ↪hue="Gene_Category", dodge=False, palette=palette)

# Add a horizontal reference line at 0
plt.axhline(y=0, color="black", linestyle="dotted")

# Compute correct category boundaries
neutral_control_count = df_prognostic[df_prognostic["Gene_Category"] ==␣
 ↪"Neutral Control"]["Target_Gene"].nunique()
loss_of_representation_count = df_prognostic[df_prognostic["Gene_Category"] ==␣
 ↪"Loss of Representation"]["Target_Gene"].nunique()
gain_of_representation_count = df_prognostic[df_prognostic["Gene_Category"] ==␣
 ↪"Gain of Representation"]["Target_Gene"].nunique()
```

```
# Add vertical dotted lines at the correct positions
plt.axvline(x=neutral_control_count - 0.5, color="gray", linestyle="dotted")  #␣
↪End of neutral controls
plt.axvline(x=neutral_control_count + loss_of_representation_count - 0.5,␣
↪color="gray", linestyle="dotted")  # End of loss of representation
plt.axvline(x=neutral_control_count + loss_of_representation_count +␣
↪gain_of_representation_count - 0.5, color="gray", linestyle="dotted")

# Customize the plot
plt.xticks(rotation=90, fontsize=10)  # Smaller font size for x-axis labels
plt.xlabel("Target Gene")
plt.ylabel("Prognostic Effect")
plt.title("Prognostic Effects Ordered by Neutral Control, Gain/Loss of␣
↪Representations, and Others")
plt.xticks(rotation = 90, fontsize = 15)

# Move legend inside the plot at the top-right corner
plt.legend(title="Gene Category", loc="upper right", fontsize=15, frameon=True)

# Show the plot
plt.show()
```



[236]:
```
## 2nd plot of prognostic effect ordered by median of each gene_category
df_prognostic2 = df_prognostic.copy()

# Compute median Prognostic_Effect for each Target_Gene within each␣
↪Gene_Category
gene_order = (
    df_prognostic2.groupby(["Sort_Order", "Target_Gene"])["Prognostic_Effect"]
    .median()
    .reset_index()
    .sort_values(["Sort_Order", "Prognostic_Effect"], ascending=[True, True])
)

# Update Target_Gene with the new categorical order
```

```
df_prognostic2["Target_Gene"] = pd.Categorical(
    df_prognostic2["Target_Gene"],
    categories=gene_order["Target_Gene"],
    ordered=True
)

# Now, re-plot with ordered Target_Gene
plt.figure(figsize=(40, 8))

# Define custom palette
palette = {
    "Neutral Control": "#E69F00",
    "Gain of Representation": "#56B4E9",
    "Loss of Representation": "#CC79A7",
    "Other": "#009E73"
}

# Create the boxplot with the updated Target_Gene order
ax = sns.boxplot(
    data=df_prognostic2,
    x="Target_Gene",
    y="Prognostic_Effect",
    hue="Gene_Category",
    dodge=False,
    palette=palette
)

# Add a horizontal reference line
plt.axhline(y=0, color="black", linestyle="dotted")

# Compute correct category boundaries
neutral_control_count = df_prognostic2[df_prognostic2["Gene_Category"] ==␣
 ↪"Neutral Control"]["Target_Gene"].nunique()
loss_of_representation_count = df_prognostic2[df_prognostic2["Gene_Category"]␣
 ↪== "Loss of Representation"]["Target_Gene"].nunique()
gain_of_representation_count = df_prognostic2[df_prognostic2["Gene_Category"]␣
 ↪== "Gain of Representation"]["Target_Gene"].nunique()

# Add vertical dotted lines to separate categories
plt.axvline(x=neutral_control_count - 0.5, color="gray", linestyle="dotted")
plt.axvline(x=neutral_control_count + loss_of_representation_count - 0.5,␣
 ↪color="gray", linestyle="dotted")
plt.axvline(x=neutral_control_count + loss_of_representation_count +␣
 ↪gain_of_representation_count - 0.5, color="gray", linestyle="dotted")

# Customize the plot
plt.xticks(rotation=90, fontsize=15)
plt.xlabel("Target Gene")
plt.ylabel("Prognostic Effect")
plt.title("Prognostic Effects Ordered by Neutral Control, Gain/Loss of␣
  ↪Representations, and Others")
plt.legend(title="Gene Category", loc="upper right", fontsize=15, frameon=True)

# Show the plot
plt.show()
```



[238]:
```
print(gene_order)
```

```
     Sort_Order Target_Gene  Prognostic_Effect
1             1       Trp53          -0.101733
0             1          NT          -0.003778
9             2        Rpl7          -1.090820
5             2         Ran          -0.889429
10            2        Rps6          -0.675664
..          …          …                …
167           4       Smad1           0.114286
55            4      Cdkn2b           0.126070
89            4       Hdac3           0.130950
128           4         Nf2           0.227321
73            4       Fbxw7           0.325220

[200 rows x 3 columns]
```

1.8.1 Nonparametric null distribution of prognostic effect

- Taking median of 10 shRNAs to derive bootstrapped empirical distribution from selected NT shRNAs

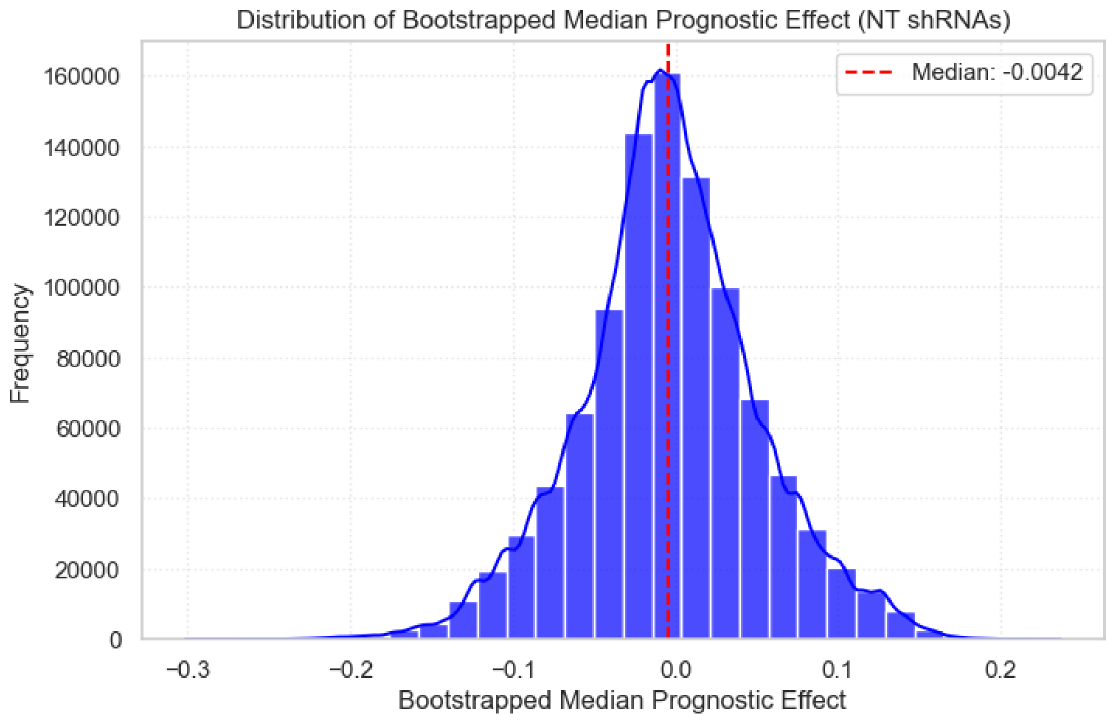
[239]:
```
df_nt_prognostic = df_prognostic[df_prognostic["Target_Gene"] == "NT"]
display(df_nt_prognostic)
```

```
     shRNA_ID Target_Gene     RTN_A     RTN_B  Prognostic_Effect  \
1147    NT_01          NT  0.005291  0.005464          -0.046575
1148    NT_02          NT  0.005976  0.005464           0.129089
```

```
1149    NT_03          NT  0.006107  0.005464          0.160482
1150    NT_04          NT  0.005527  0.005464          0.016455
1151    NT_05          NT  0.005368  0.005464         -0.025699
…       …        …       …       …                …
1325    NT_95          NT  0.005542  0.005464          0.020312
1326    NT_96          NT  0.005375  0.005464         -0.023874
1327    NT_97          NT  0.004655  0.005464         -0.231205
1328    NT_98          NT  0.005645  0.005464          0.047014
1329    NT_99          NT  0.004736  0.005464         -0.206335

        Gene_Category  Sort_Order
1147  Neutral Control           1
1148  Neutral Control           1
1149  Neutral Control           1
1150  Neutral Control           1
1151  Neutral Control           1
…                 …       …
1325  Neutral Control           1
1326  Neutral Control           1
1327  Neutral Control           1
1328  Neutral Control           1
1329  Neutral Control           1

[183 rows x 7 columns]
```

[307]:
```python
# Generate bootstrapped null distribution (median prognostic effect of 10 NT␣
 ↪shRNAs per replicate)
file_bootstrapped_nt_medians = os.path.join(proj_dir, "data/
 ↪bootstrapped_nt_medians_prognostic.npy")
rerun_bootstrapping_nt_medians_prognostic = False

if os.path.exists(file_bootstrapped_nt_medians) and not␣
 ↪rerun_bootstrapping_nt_medians_prognostic:

    bootstrapped_nt_medians = np.load(file_bootstrapped_nt_medians)

else:

    n_bootstraps = 1000000

    bootstrapped_nt_medians = np.array([
        df_nt_prognostic.sample(n=10, replace=True)["Prognostic_Effect"].
 ↪median()
        for _ in range(n_bootstraps)
    ])

    np.save(file_bootstrapped_nt_medians, bootstrapped_nt_medians)
```

[308]:
```
# Plot the distribution of bootstrapped median prognostic effects
plt.figure(figsize=(8, 5))
sns.histplot(bootstrapped_nt_medians, bins=30, kde=True, color="blue", alpha=0.
 ↪7)
plt.axvline(np.mean(bootstrapped_nt_medians), color='red', linestyle='dashed',␣
 ↪label=f"Median: {np.mean(bootstrapped_nt_medians):.4f}")
plt.xlabel("Bootstrapped Median Prognostic Effect")
plt.ylabel("Frequency")
plt.title("Distribution of Bootstrapped Median Prognostic Effect (NT shRNAs)")
plt.legend()
plt.grid(True, linestyle="dotted", alpha=0.5)

# Show plot
plt.show()
```



[309]:
```
# Compute descriptive statistics for the bootstrapped null distribution
bootstrapped_stats = {
    "Mean": np.mean(bootstrapped_nt_medians),
    "Median": np.median(bootstrapped_nt_medians),
    "Standard Deviation": np.std(bootstrapped_nt_medians),
    "IQR (Interquartile Range)": np.percentile(bootstrapped_nt_medians, 75) -␣
 ↪np.percentile(bootstrapped_nt_medians, 25),
    "5th Percentile": np.percentile(bootstrapped_nt_medians, 5),
    "95th Percentile": np.percentile(bootstrapped_nt_medians, 95),
    "Minimum": np.min(bootstrapped_nt_medians),
    "Maximum": np.max(bootstrapped_nt_medians),
}
print(bootstrapped_stats)
```

```
{'Mean': -0.004166216790693322, 'Median': -0.004976092851850339, 'Standard
Deviation': 0.05473564517171997, 'IQR (Interquartile Range)':
0.06409262101583624, '5th Percentile': -0.09698770887823943, '95th Percentile':
0.08973522212381643, 'Minimum': -0.3011943105673096, 'Maximum':
0.236535196845922}
```

1.8.2 Median prognostic effect and its bootstrapped CI of each target gene

[310]:
```
file_path_summary_prognostic_effect = os.path.join(proj_dir, "data/
 ↪prognostic_effect_bootstrapped_summary.csv")
rerun_prognostic_effect_bootstrapping = False

if os.path.exists(file_path_summary_prognostic_effect) and not␣
 ↪rerun_prognostic_effect_bootstrapping:

    df_summary_prognostic_effect = pd.
 ↪read_csv(file_path_summary_prognostic_effect)

else:
    df_prognostic_effect = df_prognostic.copy()

    # df_summary_prognostic_effect = df_prognostic_effect.
 ↪groupby(["Target_Gene", "Timepoint", "Dosage"]).apply(
    df_summary_prognostic_effect = df_prognostic_effect.
 ↪groupby(["Target_Gene"]).apply(
        lambda df: pd.Series({
            "RTN_A_median": df["RTN_A"].median(),
            "RTN_A_CI_lower": bootstrap_median_ci(df["RTN_A"])[0],
            "RTN_A_CI_upper": bootstrap_median_ci(df["RTN_A"])[1],
            "RTN_B_median": df["RTN_B"].median(),
            "RTN_B_CI_lower": bootstrap_median_ci(df["RTN_B"])[0],
            "RTN_B_CI_upper": bootstrap_median_ci(df["RTN_B"])[1],
            "Prognostic_Effect_median": df["Prognostic_Effect"].median(),
            "Prognostic_Effect_CI_lower":␣
 ↪bootstrap_median_ci(df["Prognostic_Effect"])[0],
            "Prognostic_Effect_CI_upper":␣
 ↪bootstrap_median_ci(df["Prognostic_Effect"])[1],
        })
    ).reset_index()
```

1.8.3 Calculate adjusted empirical p-values

```
[316]: null_medians = bootstrapped_nt_medians.copy()

# Function to compute two-sided empirical p-value
def empirical_p(observed, null_dist):
    p = (np.sum(np.abs(null_dist - np.median(null_dist)) >= np.abs(observed -␣
 ↪np.median(null_dist))) + 1) / (len(null_dist) + 1)
    return p

# Apply empirical p-value calculation on Prognostic_Effect_median and its lower/
 ↪upper CIs
df_summary_prognostic_effect["p_value"] =␣
 ↪df_summary_prognostic_effect["Prognostic_Effect_median"].apply(lambda x:␣
 ↪empirical_p(x, null_medians))
df_summary_prognostic_effect["p_value_CI_lower"] =␣
 ↪df_summary_prognostic_effect["Prognostic_Effect_CI_lower"].apply(lambda x:␣
 ↪empirical_p(x, null_medians))
df_summary_prognostic_effect["p_value_CI_upper"] =␣
 ↪df_summary_prognostic_effect["Prognostic_Effect_CI_upper"].apply(lambda x:␣
 ↪empirical_p(x, null_medians))


# Apply FDR correction for multi-testing
df_summary_prognostic_effect["adjusted_p_value"] =␣
 ↪multipletests(df_summary_prognostic_effect["p_value"], method='fdr_bh')[1]
df_summary_prognostic_effect["adjusted_p_value_CI_lower"] =␣
 ↪multipletests(df_summary_prognostic_effect["p_value_CI_lower"],␣
 ↪method='fdr_bh')[1]
df_summary_prognostic_effect["adjusted_p_value_CI_upper"] =␣
 ↪multipletests(df_summary_prognostic_effect["p_value_CI_upper"],␣
 ↪method='fdr_bh')[1]
```

```
[317]: display(df_summary_prognostic_effect)
df_summary_prognostic_effect.to_csv(os.path.join(proj_dir, "data/
 ↪prognostic_effect_bootstrapped_summary.csv"), index = False)
```

```
    Target_Gene  RTN_A_median  RTN_A_CI_lower  RTN_A_CI_upper  RTN_B_median  \
0         Abcb1      0.005398        0.005229        0.006241      0.005464
1          Akt1      0.004826        0.004227        0.005182      0.005464
2          Akt2      0.005548        0.005220        0.005829      0.005464
3           Apc      0.005654        0.005452        0.006212      0.005464
4          Araf      0.005351        0.005068        0.005671      0.005464
..          …             …               …               …             …
195        Wee1      0.005089        0.004621        0.005404      0.005464
196       Wwtr1      0.004707        0.004370        0.005595      0.005464
197        Yap1      0.004820        0.004148        0.005433      0.005464
198       Ywhaz      0.005161        0.004806        0.005570      0.005464
```

```
199        Zeb1     0.005448        0.004779        0.005615       0.005464

     RTN_B_CI_lower  RTN_B_CI_upper  Prognostic_Effect_median  \
0          0.005464        0.005464                 -0.017580
1          0.005464        0.005464                 -0.179909
2          0.005464        0.005464                  0.021745
3          0.005464        0.005464                  0.048979
4          0.005464        0.005464                 -0.030266
..              …               …                     …
195        0.005464        0.005464                 -0.103053
196        0.005464        0.005464                 -0.216411
197        0.005464        0.005464                 -0.181341
198        0.005464        0.005464                 -0.082765
199        0.005464        0.005464                 -0.004349

     Prognostic_Effect_CI_lower  Prognostic_Effect_CI_upper   p_value  \
0                     -0.032643                    0.191148  0.780867
1                     -0.357014                   -0.076574  0.002444
2                     -0.066057                    0.103250  0.566323
3                     -0.003302                    0.180114  0.297116
4                     -0.108810                    0.062957  0.585909
..                          …                           …         …
195                   -0.242021                   -0.040025  0.086727
196                   -0.321652                    0.034058  0.000263
197                   -0.397553                   -0.007331  0.002297
198                   -0.188809                    0.026265  0.160111
199                   -0.126748                    0.052610  0.988277

     adjusted_p_value  p_value_CI_lower  p_value_CI_upper  \
0            0.913295      5.538794e-01          0.000796
1            0.021252      9.999990e-07          0.188479
2            0.782258      2.474438e-01          0.060158
3            0.582475      9.690840e-01          0.001462
4            0.790548      7.120793e-02          0.204499
..                …               …                 …
195          0.309739      1.999998e-05          0.467562
196          0.003094      9.999990e-07          0.423989
197          0.020882      9.999990e-07          0.958131
198          0.421929      1.566998e-03          0.508596
199          0.990396      3.616696e-02          0.271311

     adjusted_p_value_CI_lower  adjusted_p_value_CI_upper
0                     0.579978                   0.007581
1                     0.000006                   0.301566
2                     0.277623                   0.167327
3                     0.969084                   0.010443
4                     0.093082                   0.319529
..                         …                          …
```

```
195                     0.000071                    0.553650
196                     0.000006                    0.526694
197                     0.000006                    0.967809
198                     0.003482                    0.574685
199                     0.051301                    0.384838

[200 rows x 16 columns]
```

1.9 Genotype-specific drug (predictive) effects

A vs. B <-> Treated (target gene vs NT) vs. Vehicle (target gene vs NT) * ratio of ratios * two dosages - 0.6nM and 3.5nM * three time point - T7, T10, and T13

[245]:
```
print(df_rtn["Sample_Description"].value_counts())
```

```
T10_Dox_0.6nM      2170
T10_Dox_3.5nM      2170
T10_Dox_Vehicle    2170
T13_Dox_0.6nM      2170
T13_Dox_3.5nM      2170
T13_Dox_Vehicle    2170
T4_Dox_NoTx        2170
T7_Dox_0.6nM       2170
T7_Dox_3.5nM       2170
T7_Dox_Vehicle     2170
T7_NoDox_NoTx      2170
Name: Sample_Description, dtype: int64
```

[246]:
```
# Define all possible treatment vs vehicle comparisons for predictive effect␣
 ↪calculation
timepoints = ["T7", "T10", "T13"]
dosages = ["0.6nM", "3.5nM"]

# Prepare an empty list to store results
predictive_effect_results = []

# Iterate over each combination of timepoint and dosage
for timepoint in timepoints:
    for dosage in dosages:
        # Define condition labels for treated (A) and vehicle (B)
        condition_A = f"{timepoint}_Dox_{dosage}"
        condition_B = f"{timepoint}_Dox_Vehicle"

        # Filter data for both conditions
        df_A = df_rtn[df_rtn["Sample_Description"] == condition_A][["shRNA_ID",␣
 ↪"Target_Gene", "RTN"]].rename(columns={"RTN": "RTN_A"})
        df_B = df_rtn[df_rtn["Sample_Description"] == condition_B][["shRNA_ID",␣
 ↪"Target_Gene", "RTN"]].rename(columns={"RTN": "RTN_B"})
```

```
        # Merge the two datasets on shRNA_ID and Target_Gene
        df_predictive = df_A.merge(df_B, on=["shRNA_ID", "Target_Gene"],␣
 ↪how="inner")

        # Compute the predictive effect as log2(RTN_A / RTN_B)
        df_predictive["Predictive_Effect"] = np.log2(df_predictive["RTN_A"] /␣
 ↪df_predictive["RTN_B"])

        # Add timepoint and dosage for reference
        df_predictive["Timepoint"] = timepoint
        df_predictive["Dosage"] = dosage

        # Append results
        predictive_effect_results.append(df_predictive)

# Concatenate all results into a single dataframe
df_predictive_effect = pd.concat(predictive_effect_results, ignore_index=True)

# Assign categories for sorting
df_predictive_effect["Gene_Category"] = "Other"  # Default category
df_predictive_effect.loc[df_predictive_effect["Target_Gene"].
 ↪isin(neutral_control_target_genes), "Gene_Category"] = "Neutral Control"
df_predictive_effect.loc[df_predictive_effect["Target_Gene"].
 ↪isin(loss_of_representation_target_genes), "Gene_Category"] = "Loss of␣
 ↪Representation"
df_predictive_effect.loc[df_predictive_effect["Target_Gene"].
 ↪isin(gain_of_representation_target_genes), "Gene_Category"] = "Gain of␣
 ↪Representation"


# Sort Target_Gene first by category, then alphabetically within each category
df_predictive_effect["Sort_Order"] = df_predictive_effect["Gene_Category"].
 ↪map({"Neutral Control": 1,
                                                                    "Loss of␣
 ↪Representation": 2,
                                                                    "Gain of␣
 ↪Representation": 3,
                                                                              ␣
 ↪"Other": 4})
df_predictive_effect = df_predictive_effect.sort_values(by=["Sort_Order",␣
 ↪"Target_Gene"])

# Get unique timepoint-dosage combinations
```

```
timepoint_dosage_combinations = df_predictive_effect[["Timepoint", "Dosage"]].
 ↪drop_duplicates().sort_values(by=["Timepoint", "Dosage"])

print(timepoint_dosage_combinations)
```

```
      Timepoint Dosage
5487        T10  0.6nM
7657        T10  3.5nM
9827        T13  0.6nM
11997       T13  3.5nM
1147         T7  0.6nM
3317         T7  3.5nM
```

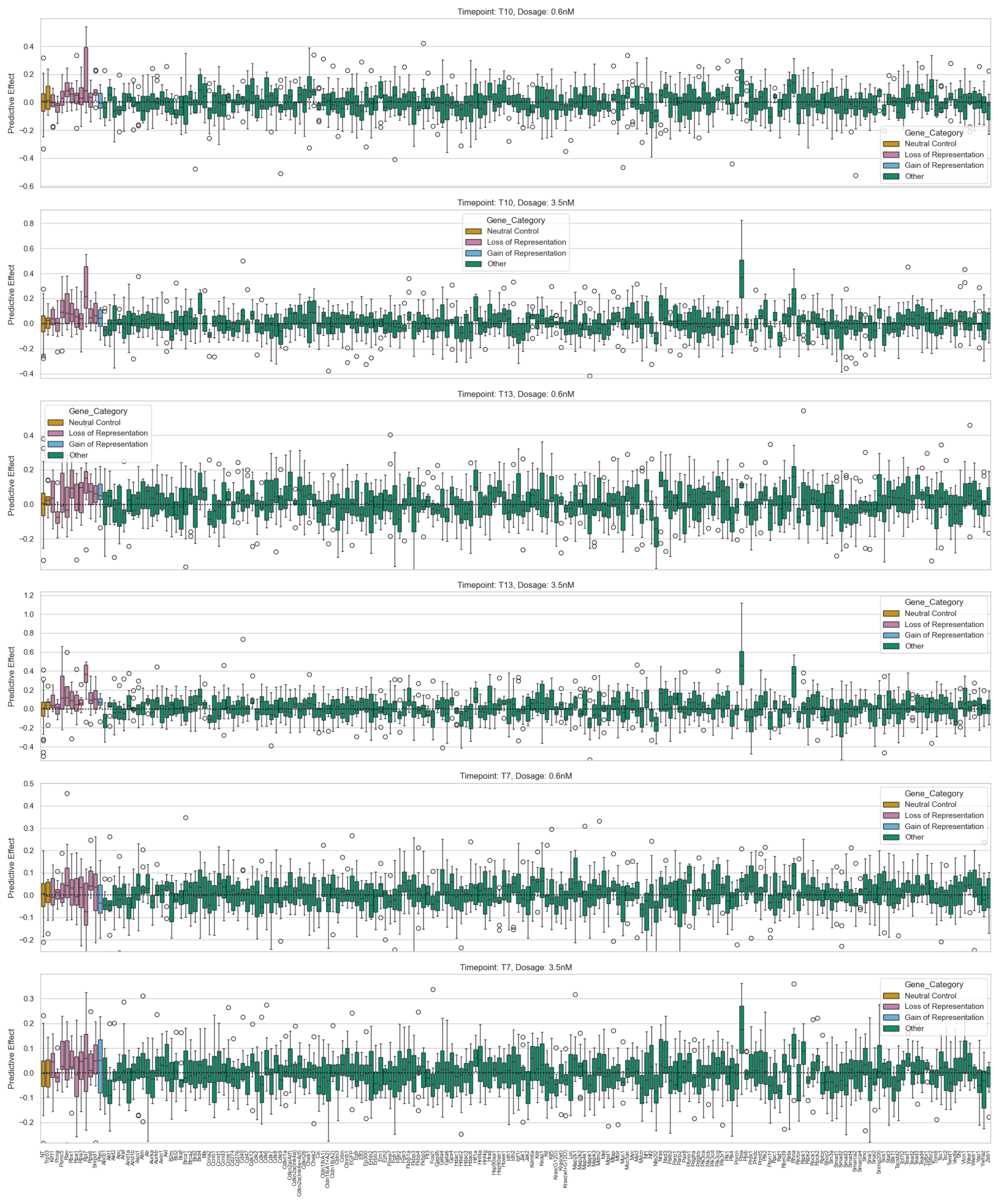
[247]:
```
display(df_predictive_effect)
df_predictive_effect.to_csv(os.path.join(proj_dir, "data/predictive_effect.
 ↪csv"), index = False)
```

```
      shRNA_ID Target_Gene     RTN_A     RTN_B  Predictive_Effect Timepoint  \
1147     NT_01          NT  0.005643  0.005483           0.041638        T7
1148     NT_02          NT  0.005596  0.005848          -0.063652        T7
1149     NT_03          NT  0.005607  0.005702          -0.024107        T7
1150     NT_04          NT  0.005454  0.005167           0.078223        T7
1151     NT_05          NT  0.005208  0.005548          -0.091238        T7
…          …           …         …         …                  …         …
13015  Zeb1_06        Zeb1  0.004919  0.004400           0.160985       T13
13016  Zeb1_07        Zeb1  0.005831  0.005739           0.023096       T13
13017  Zeb1_08        Zeb1  0.005001  0.005615          -0.167014       T13
13018  Zeb1_09        Zeb1  0.005586  0.005383           0.053211       T13
13019  Zeb1_10        Zeb1  0.005941  0.005513           0.107826       T13

      Dosage    Gene_Category  Sort_Order
1147   0.6nM  Neutral Control           1
1148   0.6nM  Neutral Control           1
1149   0.6nM  Neutral Control           1
1150   0.6nM  Neutral Control           1
1151   0.6nM  Neutral Control           1
…        …               …          …
13015  3.5nM            Other           4
13016  3.5nM            Other           4
13017  3.5nM            Other           4
13018  3.5nM            Other           4
13019  3.5nM            Other           4

[13020 rows x 9 columns]
```

[248]:
```
# # Compute correct category boundaries
# neutral_control_count = df_prognostic[df_prognostic["Gene_Category"] ==␣
 ↪"Neutral Control"]["Target_Gene"].nunique()
```

```
# loss_of_representation_count = df_prognostic[df_prognostic["Gene_Category"]␣
 ↪== "Loss of Representation"]["Target_Gene"].nunique()
# gain_of_representation_count = df_prognostic[df_prognostic["Gene_Category"]␣
 ↪== "Gain of Representation"]["Target_Gene"].nunique()

# # Add vertical dotted lines at the correct positions
# plt.axvline(x=neutral_control_count - 0.5, color="gray", linestyle="dotted") ␣
 ↪# End of neutral controls
# plt.axvline(x=neutral_control_count + loss_of_representation_count - 0.5,␣
 ↪color="gray", linestyle="dotted")  # End of loss of representation
# plt.axvline(x=neutral_control_count + loss_of_representation_count +␣
 ↪gain_of_representation_count - 0.5, color="gray", linestyle="dotted")

# Define grid size (rows = number of combinations, 1 column)
num_rows = len(timepoint_dosage_combinations)
num_cols = 1

# Create the grid plot with individually scaled y-axes for each subplot
fig, axes = plt.subplots(num_rows, num_cols, figsize=(20, num_rows * 4),␣
 ↪sharex=True)

# Ensure axes is always a list for iteration
if num_rows == 1:
    axes = [axes]

# Plot each timepoint-dosage combination in a separate row with individual␣
 ↪y-axis scaling
for ax, (timepoint, dosage) in zip(axes, timepoint_dosage_combinations.
 ↪itertuples(index=False)):
    subset = df_predictive_effect[(df_predictive_effect["Timepoint"] ==␣
 ↪timepoint) & (df_predictive_effect["Dosage"] == dosage)]

    sns.boxplot(data=subset, x="Target_Gene", y="Predictive_Effect",␣
 ↪hue="Gene_Category", dodge=False, ax=ax,
                palette=palette)

    ax.axhline(y=0, color="black", linestyle="dotted")  # Add reference line at␣
 ↪1.0
    ax.set_title(f"Timepoint: {timepoint}, Dosage: {dosage}", fontsize=12)
    ax.set_xlabel("")
    ax.set_ylabel("Predictive Effect")
    ax.tick_params(axis='x', rotation=90, labelsize=8)

    # Adjust y-axis range dynamically based on the subset
    ax.set_ylim(subset["Predictive_Effect"].min() * 0.9,␣
 ↪subset["Predictive_Effect"].max() * 1.1)
```

```
# Adjust layout for better spacing
plt.tight_layout()
plt.show()
```



1.9.1 Generate median and its bootstrapped 95% CI

For RTN_A, RTN_B, and Predictive_Effect of each Target_Gene

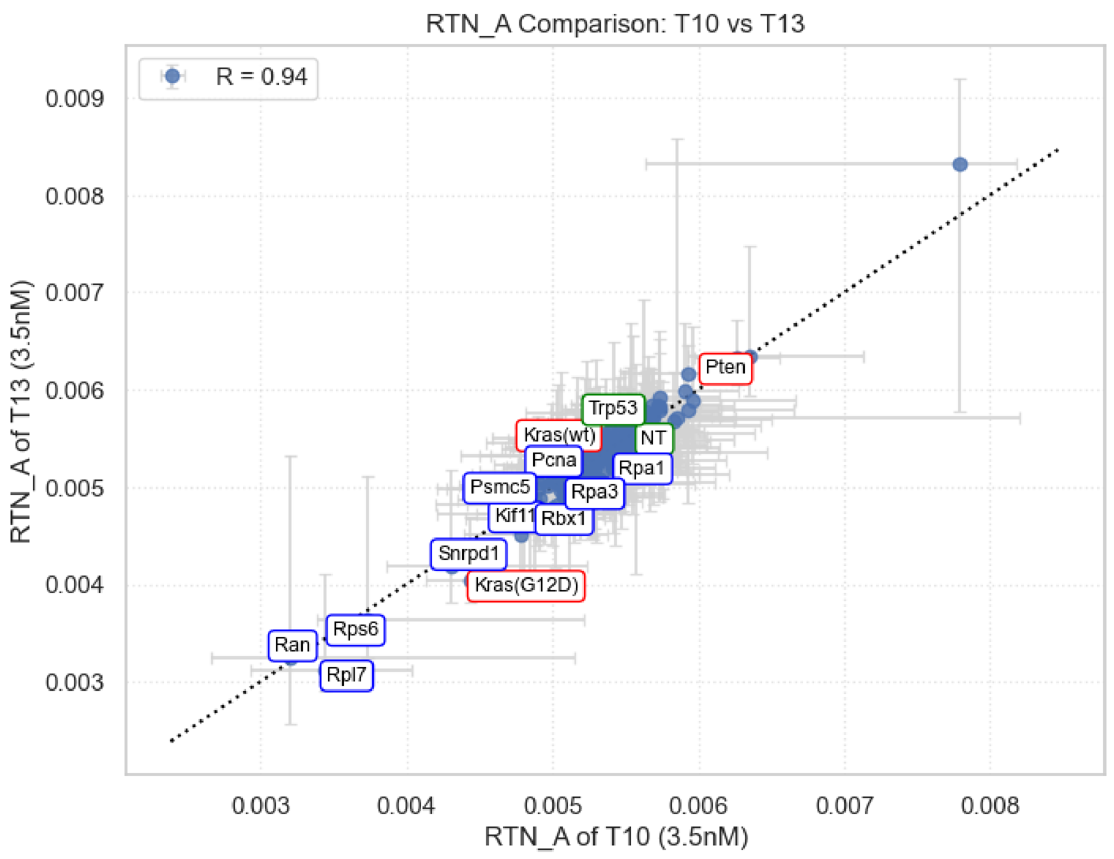
[254]:
```
file_path_summary_predictive_effect = os.path.join(proj_dir, "data/
↪predictive_effect_bootstrapped_summary.csv")
rerun_predictive_effect_bootstrapping = False

if os.path.exists(file_path_summary_predictive_effect) and not␣
↪rerun_predictive_effect_bootstrapping:

    df_summary_predictive_effect = pd.
↪read_csv(file_path_summary_predictive_effect)

else:
    df_summary_predictive_effect = df_predictive_effect.groupby(["Target_Gene",␣
↪"Timepoint", "Dosage"]).apply(
        lambda df: pd.Series({
            "RTN_A_median": df["RTN_A"].median(),
            "RTN_A_CI_lower": bootstrap_median_ci(df["RTN_A"])[0],
            "RTN_A_CI_upper": bootstrap_median_ci(df["RTN_A"])[1],
            "RTN_B_median": df["RTN_B"].median(),
            "RTN_B_CI_lower": bootstrap_median_ci(df["RTN_B"])[0],
            "RTN_B_CI_upper": bootstrap_median_ci(df["RTN_B"])[1],
            "Predictive_Effect_median": df["Predictive_Effect"].median(),
            "Predictive_Effect_CI_lower":␣
↪bootstrap_median_ci(df["Predictive_Effect"])[0],
            "Predictive_Effect_CI_upper":␣
↪bootstrap_median_ci(df["Predictive_Effect"])[1],
        })
    ).reset_index()
    df_summary_predictive_effect.to_csv(os.path.join(proj_dir, "data/
↪predictive_effect_bootstrapped_summary.csv"), index = False)
```

[255]:
```
display(df_summary_predictive_effect)
```

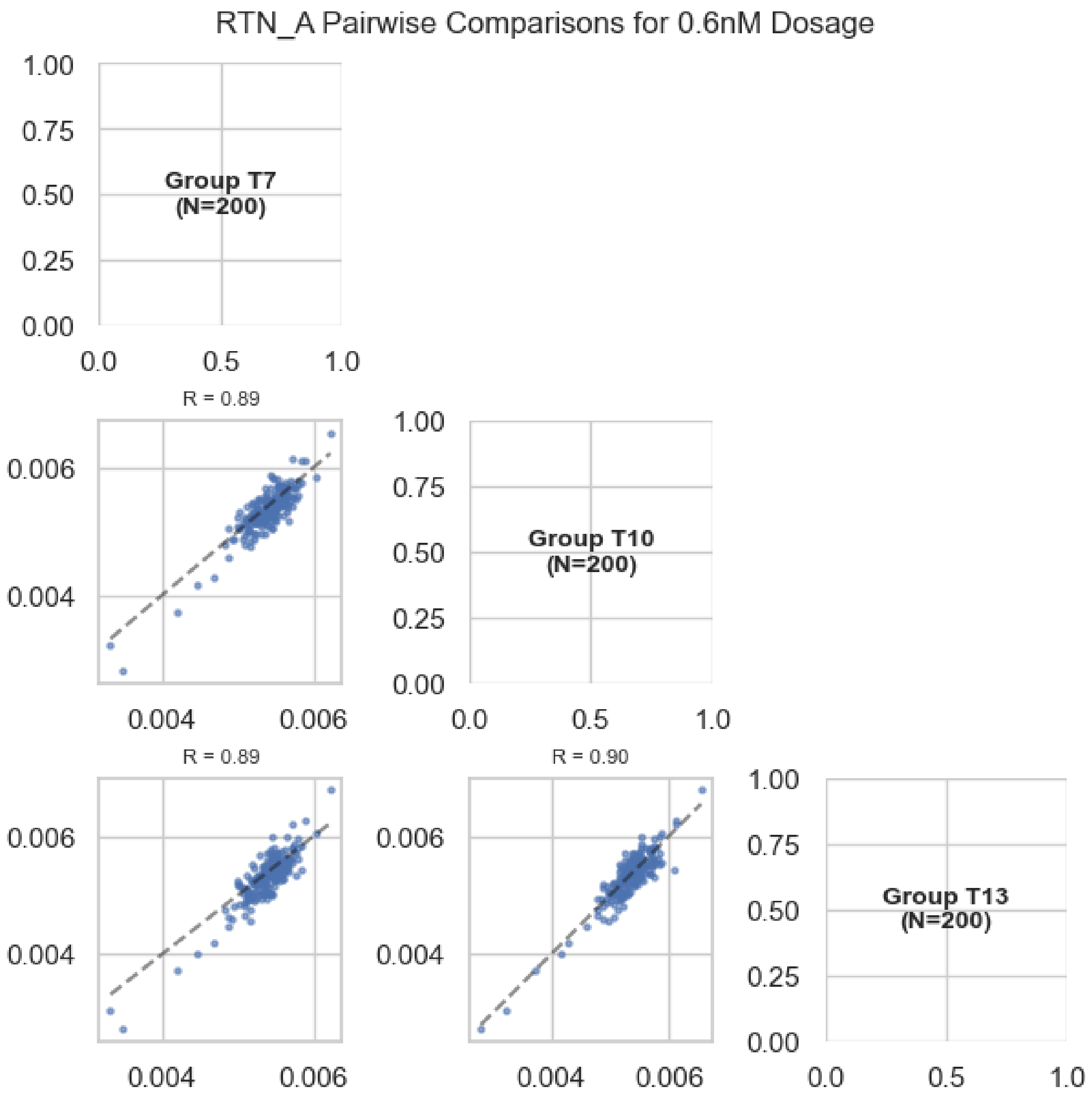
```
     Target_Gene Timepoint Dosage  RTN_A_median  RTN_A_CI_lower  \
0          Abcb1       T10  0.6nM      0.005560        0.005220
1          Abcb1       T10  3.5nM      0.005469        0.005260
2          Abcb1       T13  0.6nM      0.005441        0.005304
3          Abcb1       T13  3.5nM      0.005162        0.004821
4          Abcb1        T7  0.6nM      0.005493        0.005157
…              …       …     …             …              …
1195        Zeb1       T10  3.5nM      0.005522        0.005142
1196        Zeb1       T13  0.6nM      0.005510        0.004962
1197        Zeb1       T13  3.5nM      0.005204        0.005001
1198        Zeb1        T7  0.6nM      0.005498        0.005145
1199        Zeb1        T7  3.5nM      0.005403        0.005223
```

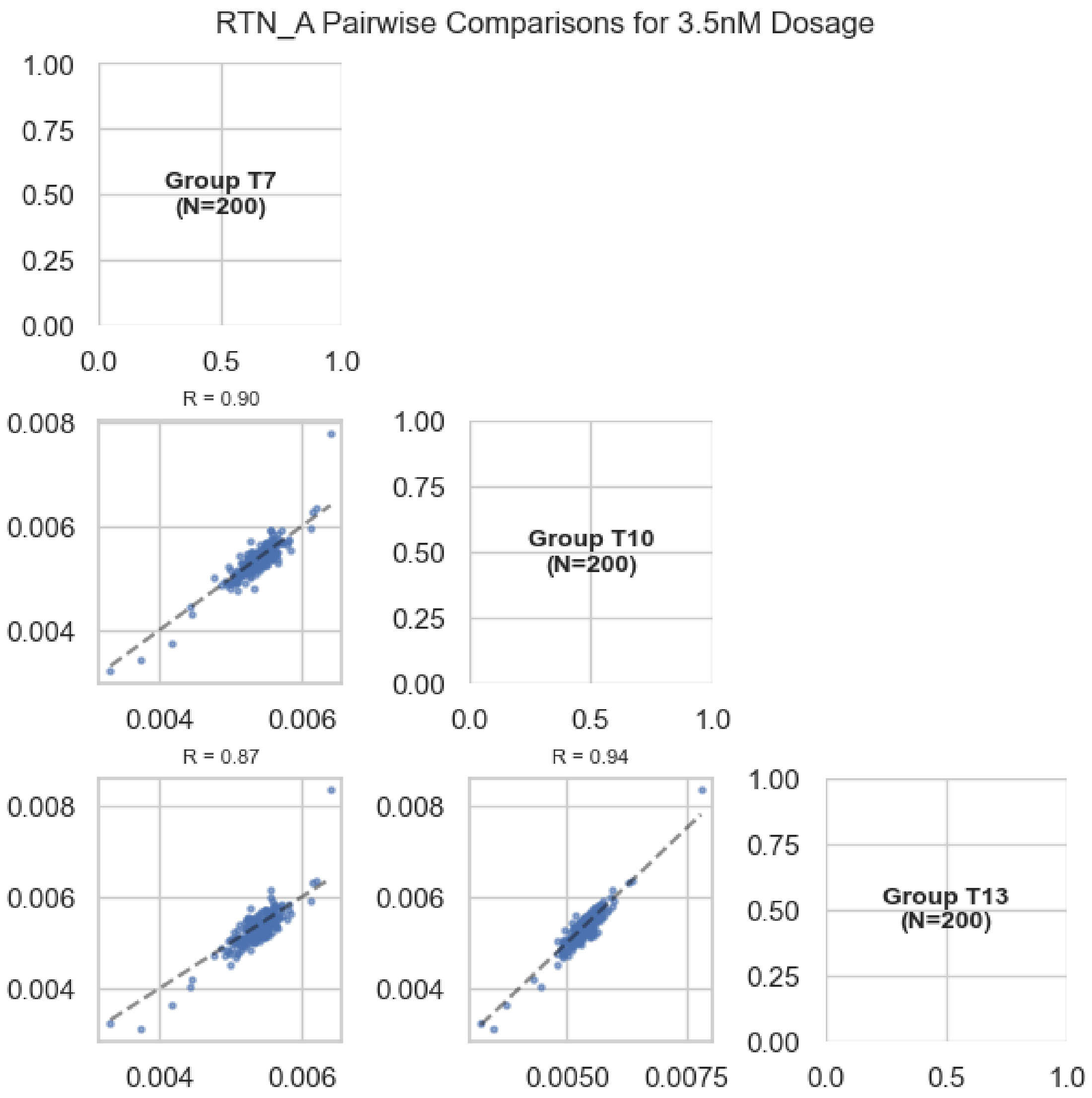
```
      RTN_A_CI_upper  RTN_B_median  RTN_B_CI_lower  RTN_B_CI_upper  \
0           0.005681      0.005670        0.005326        0.005951
1           0.005958      0.005670        0.005287        0.005944
2           0.005877      0.005398        0.005229        0.006241
3           0.005652      0.005398        0.005205        0.006241
4           0.005806      0.005622        0.005289        0.005870
…                …             …               …               …
1195        0.005823      0.005352        0.005005        0.005661
1196        0.005893      0.005448        0.004921        0.005668
1197        0.005739      0.005448        0.004985        0.005627
1198        0.005730      0.005534        0.005197        0.005872
1199        0.005727      0.005534        0.005158        0.005872

      Predictive_Effect_median  Predictive_Effect_CI_lower  \
0                    -0.038917                   -0.131243
1                    -0.050333                   -0.126803
2                    -0.014494                   -0.130830
3                    -0.099522                   -0.207631
4                    -0.043688                   -0.069247
…                          …                          …
1195                  0.077716                   -0.059963
1196                  0.034647                   -0.038528
1197                  0.036092                   -0.075872
1198                 -0.012914                   -0.091544
1199                 -0.003222                   -0.061372

      Predictive_Effect_CI_upper
0                       0.015642
1                       0.037177
2                       0.080142
3                      -0.048330
4                      -0.006350
…                            …
1195                    0.090652
1196                    0.108480
1197                    0.107826
1198                    0.088947
1199                    0.030045

[1200 rows x 12 columns]
```

[324]:
```
plot_bootstrapped_scatter(df_summary_predictive_effect, "T10", "3.5nM", "T13",␣
↪"3.5nM", "RTN_A")
```

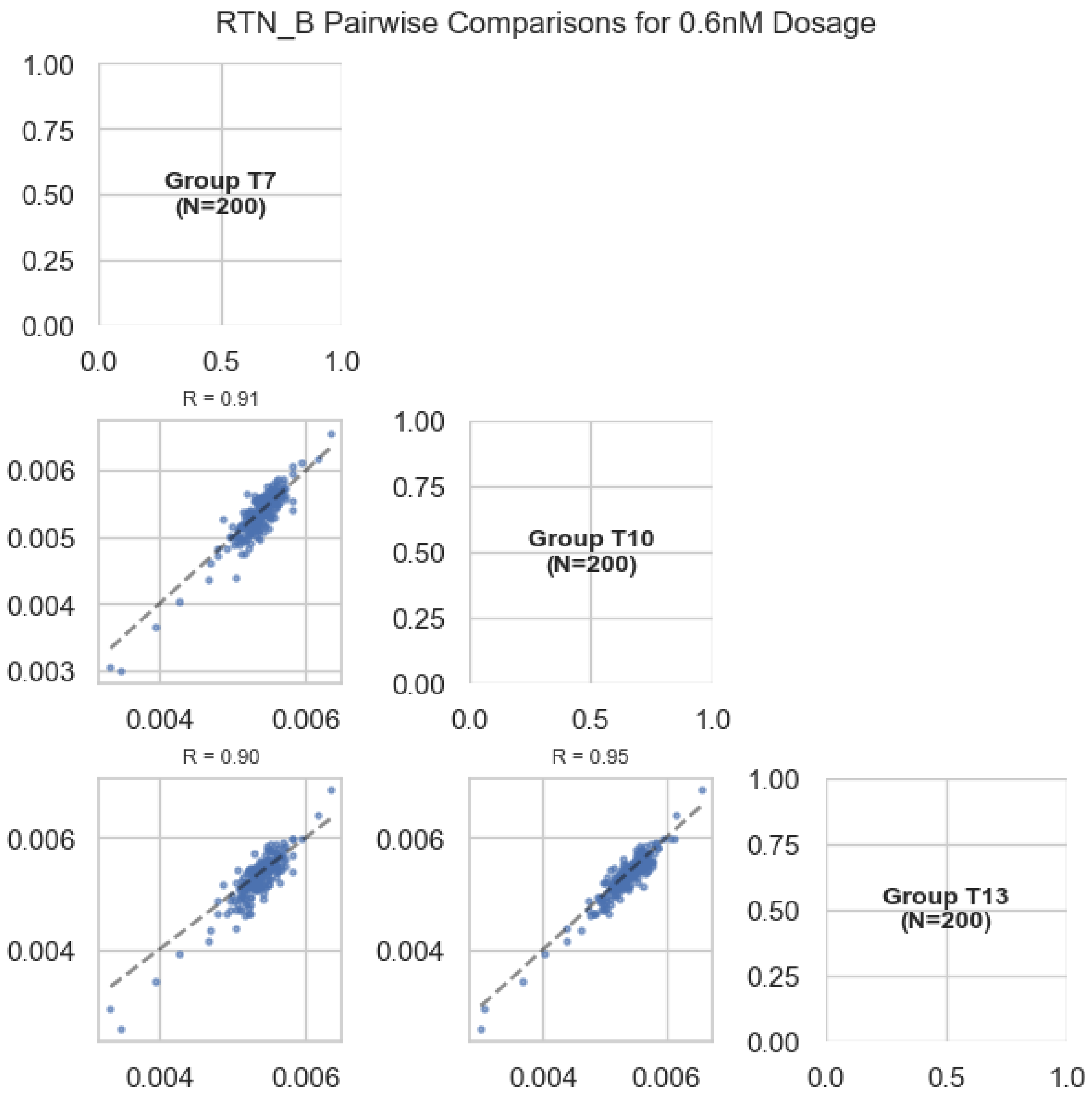


[271]:
```
# Generate optimized grid plot for RTN_A at 0.6nM and 3.5nM
plot_optimized_grid(df_summary_predictive_effect, "0.6nM", "RTN_A")
plot_optimized_grid(df_summary_predictive_effect, "3.5nM", "RTN_A")
```

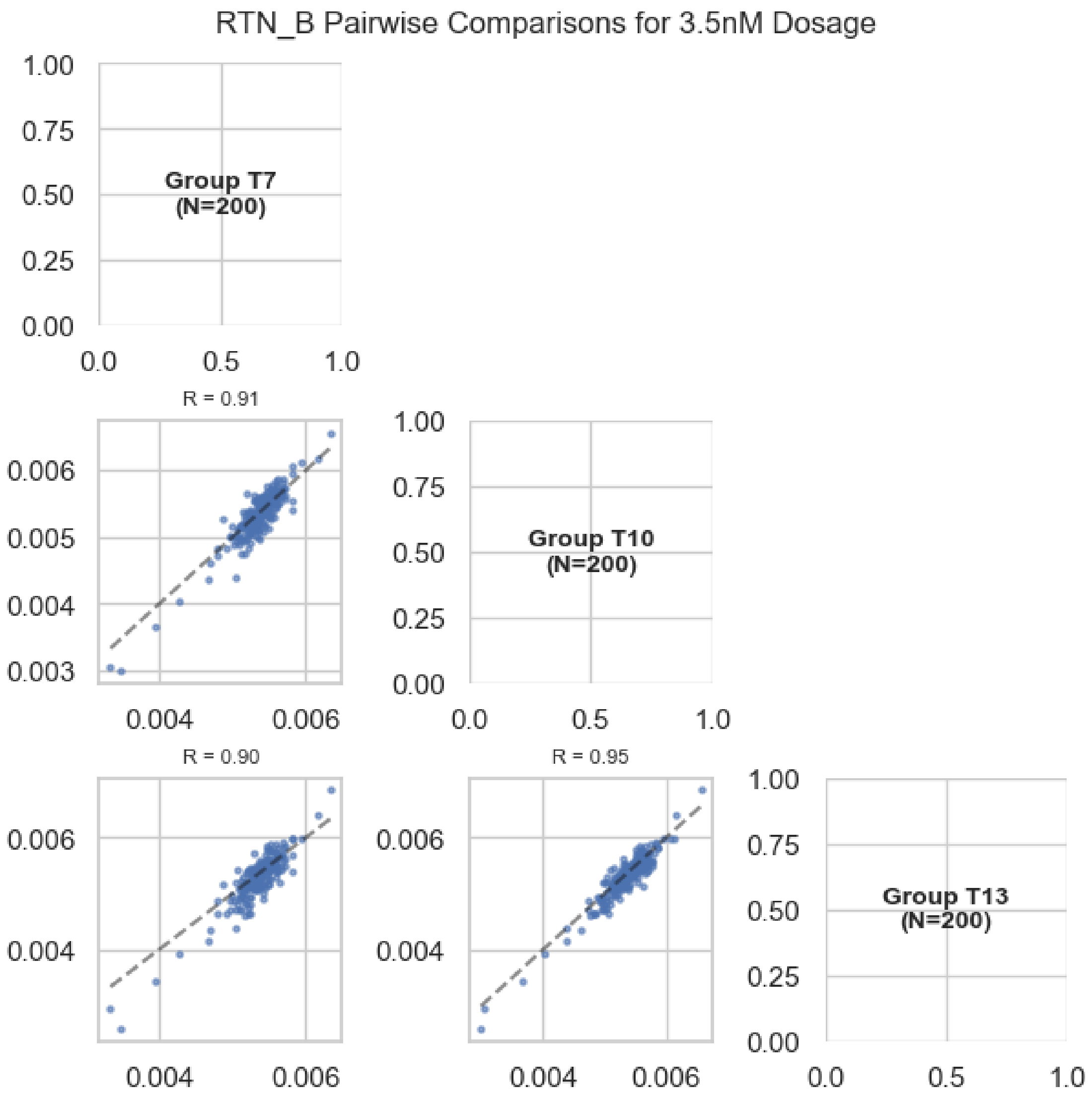

RTN_A Pairwise Comparisons for 0.6nM Dosage
Group T7 (N=200)
Group T10 (N=200)
Group T13 (N=200)
R = 0.89
R = 0.89
R = 0.90
0.004
0.006
0.0
0.5
1.0
0.00
0.25
0.50
0.75
1.00



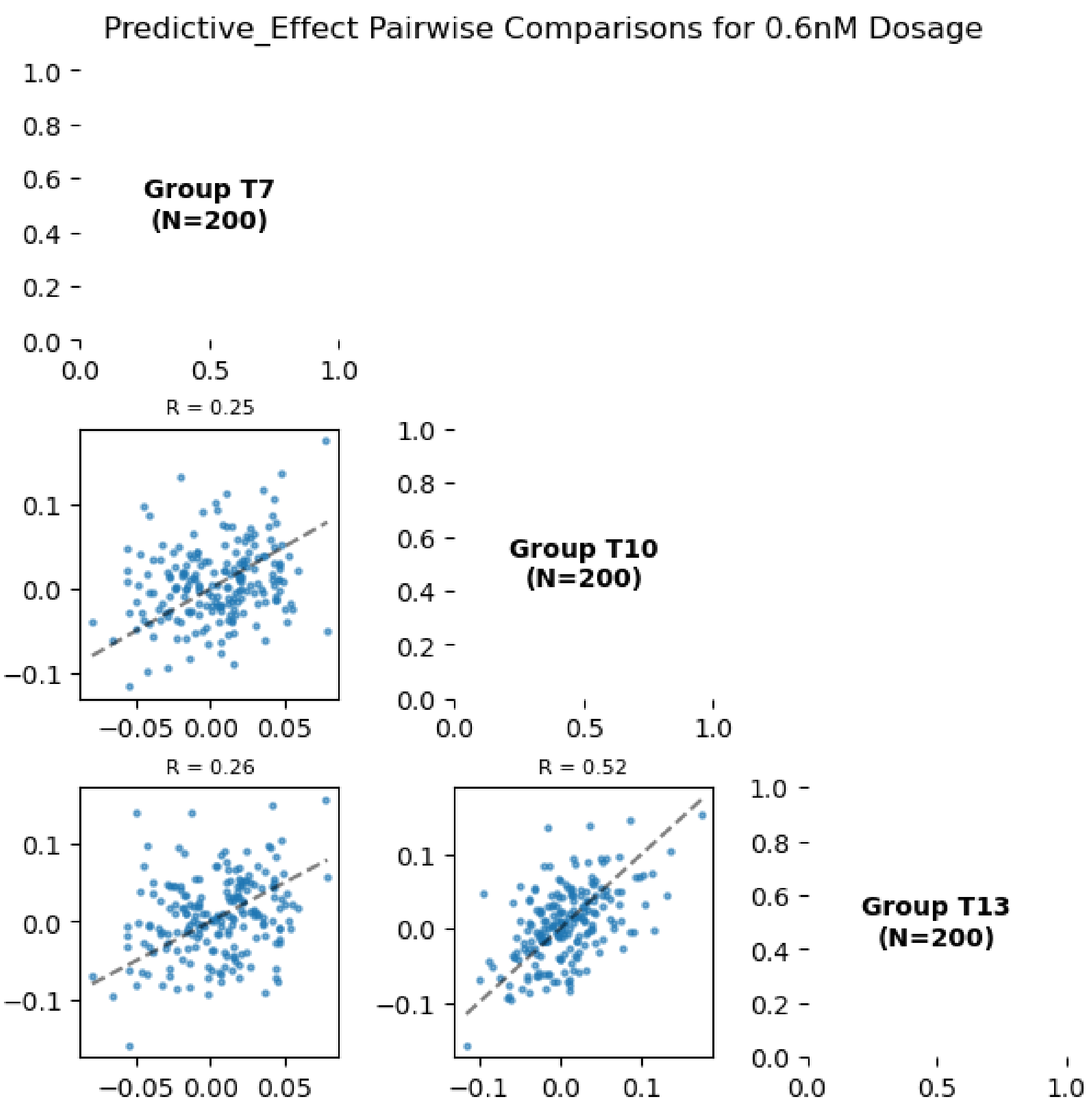
[272]:
```
# Generate optimized grid plot for RTN_B at 0.6nM and 3.5nM
plot_optimized_grid(df_summary_predictive_effect, "0.6nM", "RTN_B")
plot_optimized_grid(df_summary_predictive_effect, "3.5nM", "RTN_B")
```



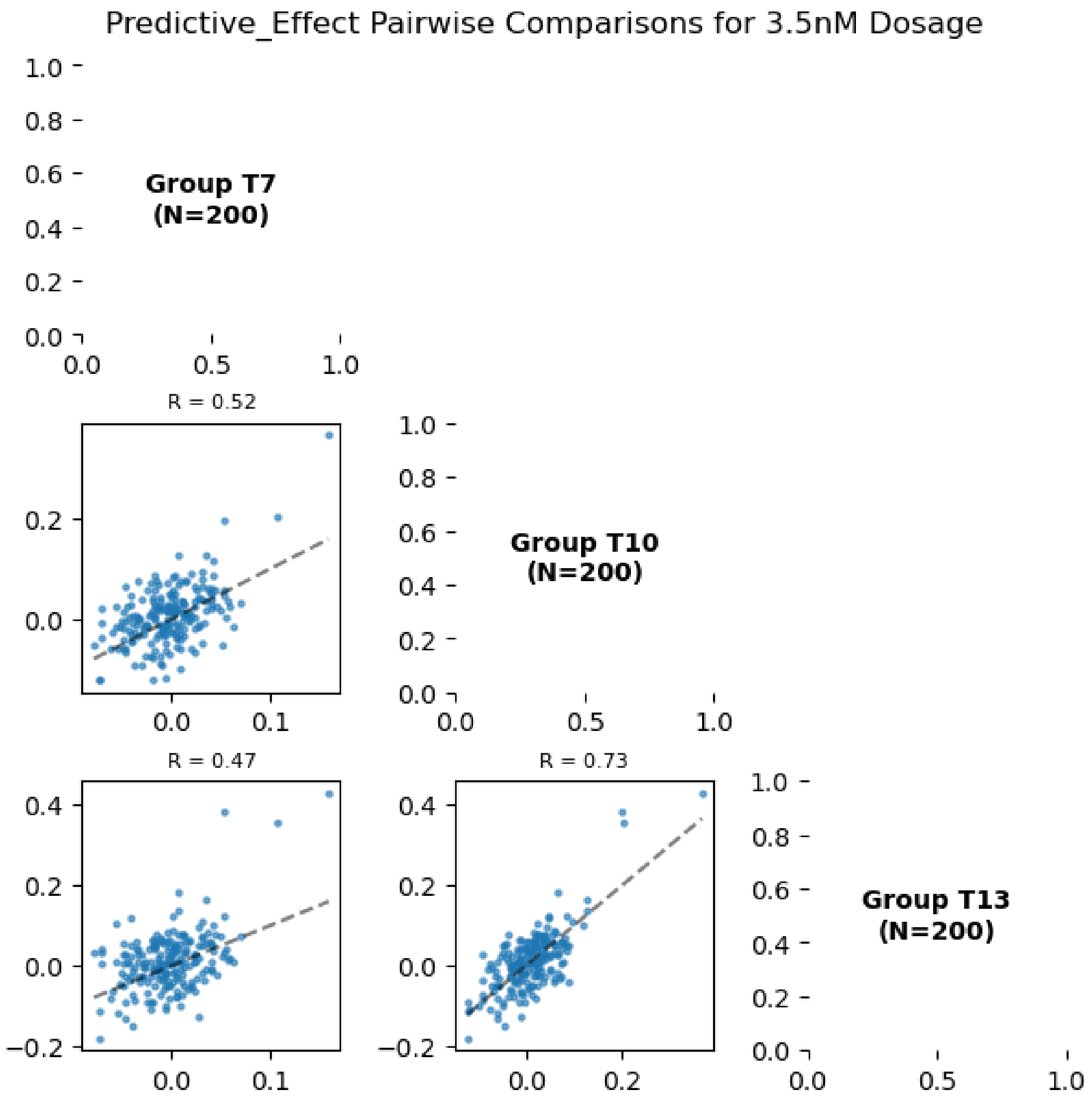
RTN_B Pairwise Comparisons for 0.6nM Dosage
Group T7 (N=200)
Group T10 (N=200)
Group T13 (N=200)
R = 0.91
R = 0.90
R = 0.95



[133]:
```
# Generate optimized grid plot for Predictive effect at 0.6nM and 3.5nM
plot_optimized_grid(df_summary_predictive_effect, "0.6nM", "Predictive_Effect")
plot_optimized_grid(df_summary_predictive_effect, "3.5nM", "Predictive_Effect")
```



Predictive_Effect Pairwise Comparisons for 0.6nM Dosage
Group T7
(N=200)
Group T10
(N=200)
Group T13
(N=200)
R = 0.25
R = 0.26
R = 0.52


Predictive_Effect Pairwise Comparisons for 3.5nM Dosage
Group T7 (N=200)
Group T10 (N=200)
Group T13 (N=200)
R = 0.52
R = 0.47
R = 0.73

```
[ ]:
```