```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.utils import resample
import matplotlib.patches as mpatches
from tqdm import tqdm


def score_model(model, x_train, y_train, x_test, y_test):
    '''
    returns a tuple of (train, test) scores
    '''

    train_score = model.score(x_train, y_train)
    test_score = model.score(x_test, y_test)

    return np.array([train_score, test_score])


def results_df(model, x_train, y_train, x_test, y_test, coef_names=None):
    '''
    Gets dataframe containing scores and optionally coefficients
    '''

    # get train and test scores
    scores_df = pd.DataFrame(score_model(model, x_train, y_train, x_test, y_test)).transpose()
    scores_df.columns = ['Train R2','Test R2']

    # get coefficient matrix
    if coef_names is not None:
        coeffs_df = pd.DataFrame(model.coef_).transpose()
        coeffs_df.columns = coef_names
        # join dataframes
        return pd.concat([scores_df, coeffs_df], axis=1)

    else:
        return scores_df


def run_experiment(model_gen, n_iters, x_train, y_train, x_test, y_test, coeff_names=None):
    '''
    Bootstraps a set of models to sample scores and optionally coefficients

    Inputs:
    --model_gen, a function that returns a dict of named and fitted models
    --n_iters, number of iterations to run
    --coef_names, [optional] names of coefficients to sample
    '''
```

```python
    sample_results = dict()

    for n in tqdm(range(n_iters)):
        # get new sample
        xb, yb = resample(x_train, y_train)

        # make and fit models
        model_dict = model_gen(xb, yb)

        # get sample of results for each model
        for key in model_dict:

            # initialize empty dictionary
            if key not in sample_results:
                sample_results[key] = []

            # get model results
            sample_results[key].append(results_df(model_dict[key], x_train, y_train, x_test, y_test, coeff_names))

    # concatenate results dfs into single df
    for key in sample_results:
        sample_results[key] = pd.concat(sample_results[key])

    return sample_results


def get_coefdata(experiment):
    '''
    Input:
    --experiment, the results of run_experiment()
    '''
    exp = experiment

    coef_dict = dict()

    # iterate over all models
    for key in exp:

        # iterate over results of this model
        for c in exp[key].columns:

            # initialize dict for result names
            if c not in coef_dict:
                coef_dict[c] = dict()

            # add this coeff to the dict
```

```python
            coef_dict[c][key] = exp[key][c]

    # convert dict of dicts into dict of dataframes
    coef_dfs = {key: pd.DataFrame(coef_dict[key]) for key in coef_dict}

    return coef_dfs


def violin_plots(experiment, column_names, experiment_name=None, center_zero=True, cmap=None):
    '''
    Makes violin plots from the results of a run_experiment() for any subset of result variables.
    Can handle single experiment or a list of experiments.

    Inputs:
    --experiment, result or list of results of run_experiment()
    --column_names, names of columns to be plotted
    --experiment_name, name or list of names of the experiments
    --center_zero, if true will center plot on 0 and scale, if false will use [0,1]
    '''

    # listify the experiment and name if they aren't already
    if type(experiment) is not list:
        experiment = [experiment]
        experiment_name = [experiment_name]

    # grab the sub-data
    coef_dfs = [get_coefdata(e) for e in experiment]

    # colors for plotting
    if cmap is None:
        colors = plt.cm.Dark2.colors
    else:
        colors = cmap

    # make a separate plot for each column in column_names,
    # which is now spelled correctly without typos
    for key in column_names:

        # make a new figure
        plt.figure(figsize=(20,10))

        # make plots for violin and 5-95 percentile
        patches = []
        for i,(c,n) in enumerate(zip(coef_dfs, experiment_name)):

            # plot violins
            data = c[key].values
```

```
        violins = plt.violinplot(data)

        # set up colors and labels for violins
        for v in violins['bodies']:
            v.set_color(colors[i])
            v.set_alpha(0.5)
        violins['cmaxes'].set_edgecolor(colors[i])
        violins['cmins'].set_edgecolor(colors[i])
        patches.append(mpatches.Patch(label=n, color=colors[i]))

    # center everything on 0 but allow scale adjustment
    if center_zero:
        lims = [np.abs(c[key].values).max()*1.2 for c in coef_dfs]
        lim = max(lims)
        plt.ylim((-lim,lim))
        plt.axhline(c='black')
    else:
        plt.ylim((0,1))

    # labels and title
    x_labels = list(coef_dfs[0][key].columns)
    plt.xticks(range(len(x_labels)+1), ['']+x_labels, rotation='horizontal')
    plt.title(key)
    plt.legend(handles=patches);
```