A *Quickstart Notebook* for ClimSim:

Go to ClimSim Github Repository

Step 1: Import data_utils

```
from data_utils import *
```

Step 2: Instantiate class

Go to original grid_info

Go to original input_mean input_max input_min output_scale

```
#Change the path to your own
grid_info = xr.open_dataset('ClimSim_low-res_grid-info.nc')
input_mean = xr.open_dataset('input_mean.nc')
input_max = xr.open_dataset('input_max.nc')
input_min = xr.open_dataset('input_min.nc')
output_scale = xr.open_dataset('output_scale.nc')

data = data_utils(grid_info = grid_info,
                  input_mean = input_mean,
                  input_max = input_max,
                  input_min = input_min,
                  output_scale = output_scale)

# set variables to V1 subset
data.set_to_v1_vars()
```

Step 3: Load training and validation data

Go to Original Dataset

```
data.input_train = data.load_npy_file('train_input_small.npy')
data.target_train = data.load_npy_file('train_target_small.npy')
data.input_val = data.load_npy_file('val_input_small.npy')
data.target_val = data.load_npy_file('val_target_small.npy')
```

Step 4: Train models

Train constant prediction model

$$\hat{y} = E[y_{train}]$$

```
const_model = data.target_train.mean(axis = 0)
```

Train multiple linear regression model

$$\beta = (X_{train}^T X_{train})^{-1} X_{train}^T y_{train}$$
$$\hat{y} = X_{input}^T \beta$$

where X_{train} and X_{input} correspond to the training data and the input data you would like to inference on, respectively. X_{train} and X_{input} both have a column of ones concatenated to the feature space for the bias.

```
adding bias unit
```

```
X = data.input_train
bias_vector = np.ones((X.shape[0], 1))
X = np.concatenate((X, bias_vector), axis=1)
```

create model

```
mlr_weights = np.linalg.inv(X.transpose()@X)@X.transpose()@data.target_train
```

Train your models here

```
###
# train your model here
###
```

Step 5: Evaluate on validation data

Set pressure grid

```
data.set_pressure_grid(data_split = 'val')
```

Load predictions

```
# Constant Prediction
const_pred_val = np.repeat(const_model[np.newaxis, :], data.target_val.shape[0], a
print(const_pred_val.shape)

# Multiple Linear Regression
X_val = data.input_val
bias_vector_val = np.ones((X_val.shape[0], 1))
X_val = np.concatenate((X_val, bias_vector_val), axis=1)
mlr_pred_val = X_val@mlr_weights
print(mlr_pred_val.shape)

# Load your prediction here

# Load predictions into data_utils object
data.model_names = ['const', 'mlr'] # add names of your models here
preds = [const_pred_val, mlr_pred_val] # add your custom predictions here
data.preds_val = dict(zip(data.model_names, preds))
```

Weight predictions and target

1.Undo output scaling
2.Weight vertical levels by dp/g
3.Weight horizontal area of each grid cell by a[x]/mean(a[x])
4.Convert units to a common energy unit

```
data.reweight_target(data_split = 'val')
data.reweight_preds(data_split = 'val')
```

Set and calculate metrics

```
data.metrics_names = ['MAE', 'RMSE', 'R2', 'bias']
data.create_metrics_df(data_split = 'val')
```

Create plots

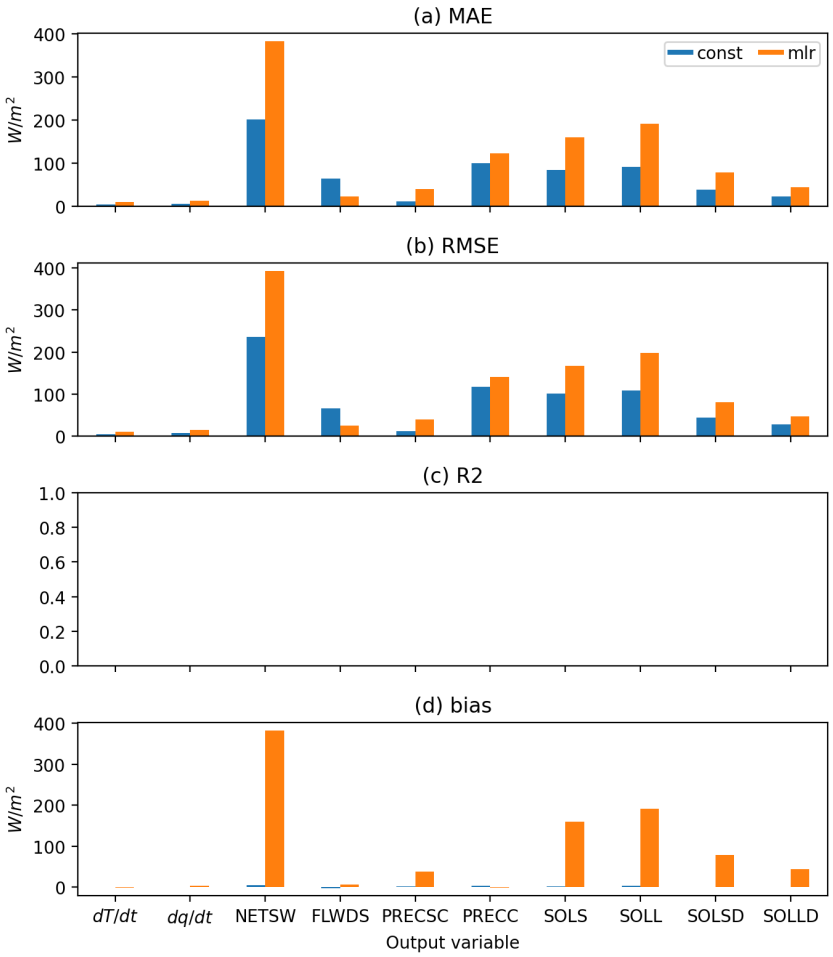
```
# set plotting settings
%config InlineBackend.figure_format = 'retina'
letters = string.ascii_lowercase

# create custom dictionary for plotting
dict_var = data.metrics_var_val
plot_df_byvar = {}
for metric in data.metrics_names:
    plot_df_byvar[metric] = pd.DataFrame([dict_var[model][metric] for model in dat
                                          index=data.model_names)
    plot_df_byvar[metric] = plot_df_byvar[metric].rename(columns = data.var_short_
```

```
# plot figure
fig, axes = plt.subplots(nrows  = len(data.metrics_names), sharex = True)
for i in range(len(data.metrics_names)):
    plot_df_byvar[data.metrics_names[i]].plot.bar(
        legend = False,
        ax = axes[i])
    if data.metrics_names[i] != 'R2':
        axes[i].set_ylabel('$W/m^2$')
    else:
        axes[i].set_ylim(0,1)

    axes[i].set_title(f'({letters[i]}) {data.metrics_names[i]}')
axes[i].set_xlabel('Output variable')
axes[i].set_xticklabels(plot_df_byvar[data.metrics_names[i]].index,     rotation=0

axes[0].legend(columnspacing = .9,
               labelspacing = .3,
               handleheight = .07,
               handlelength = 1.5,
               handletextpad = .2,
               borderpad = .2,
               ncol = 3,
               loc = 'upper right')
fig.set_size_inches(7,8)
fig.tight_layout()
```




```
If you trained models with different hyperparameters, use the ones that performed th
```

Step 6: Evaluate on scoring data

Do this at the VERY END (when you have finished tuned the hyperparameters for your model and are seeking a final evaluation)

Load scoring data

```
data.input_scoring = np.load('scoring_input_small.npy')
data.target_scoring = np.load('scoring_target_small.npy')
```

Set pressure grid

```
data.set_pressure_grid(data_split = 'scoring')
```

Load predictions

```
# constant prediction
const_pred_scoring = np.repeat(const_model[np.newaxis, :], data.target_scoring.sha
print(const_pred_scoring.shape)

# multiple linear regression
X_scoring = data.input_scoring
bias_vector_scoring = np.ones((X_scoring.shape[0], 1))
X_scoring = np.concatenate((X_scoring, bias_vector_scoring), axis=1)
mlr_pred_scoring = X_scoring@mlr_weights
print(mlr_pred_scoring.shape)

# Your model prediction here

# Load predictions into object
data.model_names = ['const', 'mlr'] # model name here
preds = [const_pred_scoring, mlr_pred_scoring] # add prediction here
data.preds_scoring = dict(zip(data.model_names, preds))
```

Weight predictions and target

1.Undo output scaling
2.Weight vertical levels by dp/g
3.Weight horizontal area of each grid cell by a[x]/mean(a[x])
4.Convert units to a common energy unit

```
# weight predictions and target
data.reweight_target(data_split = 'scoring')
data.reweight_preds(data_split = 'scoring')

# set and calculate metrics
data.metrics_names = ['MAE', 'RMSE', 'R2', 'bias']
data.create_metrics_df(data_split = 'scoring')
```

Create plots

```
# set plotting settings
%config InlineBackend.figure_format = 'retina'
letters = string.ascii_lowercase

# create custom dictionary for plotting
dict_var = data.metrics_var_scoring
plot_df_byvar = {}
for metric in data.metrics_names:
    plot_df_byvar[metric] = pd.DataFrame([dict_var[model][metric] for model in dat
                                          index=data.model_names)
    plot_df_byvar[metric] = plot_df_byvar[metric].rename(columns = data.var_short_

# plot figure
fig, axes = plt.subplots(nrows  = len(data.metrics_names), sharex = True)
for i in range(len(data.metrics_names)):
    plot_df_byvar[data.metrics_names[i]].plot.bar(
        legend = False,
        ax = axes[i])
    if data.metrics_names[i] != 'R2':
        axes[i].set_ylabel('$W/m^2$')
    else:
        axes[i].set_ylim(0,1)

    axes[i].set_title(f'({letters[i]}) {data.metrics_names[i]}')
axes[i].set_xlabel('Output variable')
axes[i].set_xticklabels(plot_df_byvar[data.metrics_names[i]].index,     rotation=0
```

```
axes[0].legend(columnspacing = .9,
               labelspacing = .3,
               handleheight = .07,
               handlelength = 1.5,
               handletextpad = .2,
               borderpad = .2,
               ncol = 3,
               loc = 'upper right')
fig.set_size_inches(7,8)
fig.tight_layout()
```