Appendix A. API Documentation

This section describes the methods available in the *Climate Econometrics Toolkit API*.

Appendix A.1. Methods for Step 1: Gridded Climate Data Aggregation

`extract_raster_data(gridded_climate_filepath, shape_filepath, weight_file=None)`: Wrapper for exact_extract
gridded_climate_filepath: a string representing the path to a NetCDF file of gridded climate data
shape_filepath: a string representing the path to a Shape file
weight_file (optional): a string representing the path to a netCDF file of weights sampled to the same size as the gridded climate file
Returns: a dictionary containing extracted results

`aggregate_raster_data(extracted_climate_data, shape_filepath, climate_var_name, aggregation_func, geo_identifier, subperiods_per_time_unit, months_to_use=None)`: Aggregates output of *extract_raster_data* into a DataFrame based on the desired temporal level. Takes as input the output from the *extract_raster_data* method, a string representing the path to the same Shape file used in the call to *extract_raster_data*, a string representing the name of the climate variable to aggregate (this becomes the name of the data column in the DataFrame), a string representing the function to use for aggregation ("mean" or "sum"), a string representing the geographical identifier column in the shape file (used to appropriately label the data in the DataFrame), an integer representing the number of subperiods to aggregate together (e.g. if converting monthly data to yearly data, use 12), and optionally a dictionary representing which months should be used in the aggregation. Returns a Pandas DataFrame as output.

Appendix A.2. Methods for Step 2: Econometric Model Development

`evaluate_model()`: Run out-of-sample evaluation on the current model. Returns: a string representing the model ID assigned to the current model

`get_best_model(metric='r2')`: Get the best model from the current session, based on the supplied metric. `metric` should be one of "out_sample_mse_reduction","out_sample_mse","out_sample_pred_int_cov", "rmse","r2"

Returns: a ClimateEconometricsModel

`get_all_model_ids()`: Get IDs of all models in the current session.
Returns: a list of strings representing the model IDs

`get_model_by_id(model_id)`: Get the model object corresponding to the given model ID
Returns: a ClimateEconometricsModel

`load_dataset_from_file(datafile)`: Loads the given filepath as a CSV into a Pandas DataFrame and adds it to the current model
Returns: None

`view_current_model()`: Prints details of the current model
Returns: None

`set_target_variable(var, existence_check=True)`: Sets the dependent (target) variable in the current model
var: a string representing a column in the loaded dataset
existence_check: Boolean indicating whether to check to see if a dataset is loaded and if *var* exists in the loaded dataset before attempting to add to model
Returns: None

`set_time_column(var)`: Sets the time column in the current model
var: a string representing a column in the loaded dataset
Returns: None

`set_panel_column(var)`: Sets the panel column in the current model
var: a string representing a column in the loaded dataset
Returns: None

`add_transformation(var, transformations, keep_original_var=True)`: add a transformation or transformations to a single model variable
var: a string representing a column in the loaded dataset
transformations: a list of transformations to be applied to the specified model variable. All list items should be one of "fd","sq","cu","ln","lag1","lag2","lag3".

keep_original_var: Boolean indicating whether the non-transformed variable should be kept or removed from the model after the transformation is applied to it
Returns: None

`add_covariates(vars, existence_check=True)`: add a covariate or covariates to the current model
vars: a list of strings representing columns in the loaded dataset
existence_check: Boolean indicating whether to check to see if a dataset is loaded and if *var* exists in the loaded dataset before attempting to add to model
Returns: None

`add_fixed_effect(vars)`: add a fixed effect or fixed effects to the current model
`vars`: a list of strings representing columns in the loaded dataset
Returns: None

`add_time_trend(vars, exp)`: add a time trend or time trends to the current model
`vars`: a list of strings representing columns in the loaded dataset
`exp`: integer representing the power of the time trend. For instance, 1 means that the time trend will be linear, 2 will be quadratic, 3 will be cubic, etc.
Returns: None

`remove_covariates(vars)`: remove a covariate or covariates from the current model
vars: a list of strings representing covariates in the current model
Returns: None

`remove_time_trend(var, exp)`: remove a time trend from the current model
var: a string representing a variable with a time trend
exp: an integer representing the power of the time trend to remove
Returns: None

`remove_transformation(var, transformations)`: remove a transformed variable from the current model
var: a transformed variable in the current model
transformations: the list of transformations applied to the transformed vari-

able to remove
Returns: None

`run_bayesian_inference(model, num_samples=1000`: Run Bayesian Inference against the current model and dataset. Results will be saved to the subdirectory `bayes_samples` in the CET home directory. This command can be long running.
model: a ClimateEconometricsModel
num_samples: the number of posterior samples to generate for each sampling chain
Returns: None

`run_block_bootstrap(model, num_samples)`: Run bootstrapping against the current model and dataset. Results will be saved to the subdirectory `bootstrap_samples` in the CET home directory. This command can be long running.
model: a ClimateEconometricsModel
num_samples: the number of bootstrap samples to generate
Returns: None

Appendix A.3. Methods for Step 3: Computation of Impacts

`predict_out_of_sample(model, data, transform_data=False, var_map=None)`: use the fitted model to generate predictions on out-of-sample data
model: a ClimateEconometricsModel
data: a Pandas DataFrame with columns for all variables in the model
transform_data: a Boolean indicating whether to apply the data transformations (e.g. square, log, first difference) to the out-of-sample data
var_map (optional): A dictionary of variable names, in the case that the out-of-sample data has column names that do not match the variable names in the model
Returns: a Pandas DataFrame containing the predictions

`geotemporal_cumulative_sum(model, predictions, geo_weights=None, prediction_columns=None)`: apply the yearly cumulative sum to predictions for all countries.
Called by invoking the API with `call_user_prediction_function(`

`'geotemporal_cumulative_sum', [predictions, geo_weights, prediction_columns])`
model: a ClimateEconometricsModel
prediction: a Pandas DataFrame containing predictions generated by the model
geo_weights (optional): a dictionary containing weights for each geolocation, which will be multiplied with the cumulative sum of the impacts for each country
prediction_columns (optional): a list of columns in the predict DataFrame to include. This is useful if you want to restrict the number of prediction samples generated by bootstrapping or Bayesian inference post-hoc.