

# Diagnostics

## Diagnostics

`diag.diag` is a class to unify various diagnostics methods and provide a consistent interface for diagnostics.

Compute the widely applicable information criterion.

Estimates the expected log pointwise predictive density (elpd) using WAIC. Also calculates the WAIC's standard error and the effective number of parameters. Read more theory here <https://arxiv.org/abs/1507.04544> and here <https://arxiv.org/abs/1004.2316>

## Parameters

`pointwise`: bool If True the pointwise predictive accuracy will be returned. Defaults to `stats.ic_pointwise` rcParam. `var_name`: str, optional The name of the variable in `log_likelihood` groups storing the pointwise log likelihood data to use for waic computation. `scale`: str Output scale for WAIC. Available options are:

- `log`: (default) log-score
- `negative_log`:  $-1 * \text{log-score}$
- `deviance`:  $-2 * \text{log-score}$

A higher log-score (or a lower deviance or negative log\_score) indicates a model with better predictive accuracy. `dask_kwargs`: dict, optional Dask related kwargs passed to `:func:~arviz.wrap_xarray_ufunc`.

## Returns

ELPDData object (inherits from :class:pandas.Series) with the following row/attributes:  
elpd\_waic: approximated expected log pointwise predictive density (elpd) se: standard error of the elpd p\_waic: effective number parameters n\_samples: number of samples n\_data\_points: number of data points warning: bool True if posterior variance of the log predictive densities exceeds 0.4 waic\_i: :class:xarray.DataArray with the pointwise predictive accuracy, only if pointwise=True scale: scale of the elpd

The returned object has a custom print method that overrides pd.Series method.

```
bi.dist.WAIC(  
self,  
pointwise=None,  
var_name=None,  
scale=None,  
dask_kwargs=None,  
)
```

---

Plot autocorrelation of the MCMC chains.

Args: \*args, \*\*kwargs: Additional arguments for arviz.plot\_autocorr

Returns: fig: Autocorrelation plot

```
bi.dist.autocor(  
self,  
*args,  
**kwargs,  
)
```

---

Compare models based on their expected log pointwise predictive density (ELPD).

The ELPD is estimated either by Pareto smoothed importance sampling leave-one-out cross-validation (LOO) or using the widely applicable information criterion (WAIC). We recommend loo. Read more theory here - in a paper by some of the leading authorities on model comparison [dx.doi.org/10.1111/1467-9868.00353](https://doi.org/10.1111/1467-9868.00353)

## Parameters

`compare_dict`: dict of {str: InferenceData or ELPDData} A dictionary of model names and :class:arviz.InferenceData or ELPDData. `ic`: str, optional Method to estimate the ELPD, available options are “loo” or “waic”. Defaults to `rcParams["stats.information_criterion"]`. `method`: str, optional Method used to estimate the weights for each model. Available options are:

- ‘stacking’ : stacking of predictive distributions.
- ‘BB-pseudo-BMA’ : pseudo-Bayesian Model averaging using Akaike-type weighting. The weights are stabilized using the Bayesian bootstrap.
- ‘pseudo-BMA’: pseudo-Bayesian Model averaging using Akaike-type weighting, without Bootstrap stabilization (not recommended).

For more information read <https://arxiv.org/abs/1704.02030> `b_samples`: int, optional default = 1000 Number of samples taken by the Bayesian bootstrap estimation. Only useful when `method = ‘BB-pseudo-BMA’`. Defaults to `rcParams["stats.ic_compare_method"]`. `alpha`: float, optional The shape parameter in the Dirichlet distribution used for the Bayesian bootstrap. Only useful when `method = ‘BB-pseudo-BMA’`. When `alpha=1` (default), the distribution is uniform on the simplex. A smaller `alpha` will keeps the final weights more away from 0 and 1. `seed`: int or `np.random.RandomState` instance, optional If int or `RandomState`, use it for seeding Bayesian bootstrap. Only useful when `method = ‘BB-pseudo-BMA’`. Default None the global `mod:numpy.random` state is used. `scale`: str, optional Output scale for IC. Available options are:

- `log` : (default) log-score (after Vehtari et al. (2017))
- `negative_log` :  $-1 * (\text{log-score})$
- `deviance` :  $-2 * (\text{log-score})$

A higher log-score (or a lower deviance) indicates a model with better predictive accuracy. `var_name`: str, optional If there is more than a single observed variable in the `InferenceData`, which should be used as the basis for comparison.

## Returns

A `DataFrame`, ordered from best to worst model (measured by the ELPD). The index reflects the key with which the models are passed to this function. The columns are: `rank`: The rank-order of the models. 0 is the best. `elpd`: ELPD estimated either using (PSIS-LOO-CV `elpd_loo` or WAIC `elpd_waic`). Higher ELPD indicates higher out-of-sample predictive fit (“better” model). If `scale` is `deviance` or `negative_log` smaller values indicates higher out-of-sample predictive fit (“better” model). `pIC`: Estimated effective number of parameters. `elpd_diff`: The difference in ELPD between two models. If more than two models are compared, the difference is computed relative to the top-ranked model, that always has a `elpd_diff` of 0.

weight: Relative weight for each model. This can be loosely interpreted as the probability of each model (among the compared model) given the data. By default the uncertainty in the weights estimation is considered using Bayesian bootstrap. SE: Standard error of the ELPD estimate. If method = BB-pseudo-BMA these values are estimated using Bayesian bootstrap. dSE: Standard error of the difference in ELPD between each model and the top-ranked model. It's always 0 for the top-ranked model. warning: A value of 1 indicates that the computation of the ELPD may not be reliable. This could be indication of WAIC/LOO starting to fail see <http://arxiv.org/abs/1507.04544> for details. scale: Scale used for the ELPD.

## References

.. [1] Vehtari, A., Gelman, A. & Gabry, J. Practical Bayesian model evaluation using leave-one-out cross-validation and WAIC. Stat Comput 27, 1413–1432 (2017) see <https://doi.org/10.1007/s11222-016-9696-4>

```
bi.dist.compare(  
  compare_dict,  
  ic=None,  
  method='stacking',  
  b_samples=1000,  
  alpha=1,  
  seed=None,  
  scale=None,  
  var_name=None,  
)
```

---

Plot density plots of the posterior distribution.

Args: var\_names (list): Variables to include shade (float): Transparency of the filled area  
\*args, \*\*kwargs: Additional arguments for arviz.plot\_density

Returns: fig: Density plots

```
bi.dist.density(  
  self,  
  var_names=None,  
  shade=0.2,  
  *args,  
  **kwargs,  
)
```

---

Calculate effective sample size (ESS).

Args: \*args, \*\*kwargs: Additional arguments for arviz.ess

Returns: ess: Effective sample sizes

```
bi.dist.ess(  
self,  
*args,  
**kwargs,  
)
```

---

Create a forest plot of estimated values.

Args: list: Data to plot (default: self.trace) kind (str): Type of plot (default: “ridgeplot”) ess (bool): Include effective sample size var\_names (list): Variables to include \*args, \*\*kwargs: Additional arguments for arviz.plot\_forest

Returns: fig: Forest plot

```
bi.dist.forest(  
self,  
list=None,  
kind='ridgeplot',  
ess=True,  
var_names=None,  
*args,  
**kwargs,  
)
```

---

Compute Pareto-smoothed importance sampling leave-one-out cross-validation (PSIS-LOO-CV).

Estimates the expected log pointwise predictive density (elpd) using Pareto-smoothed importance sampling leave-one-out cross-validation (PSIS-LOO-CV). Also calculates LOO's standard error and the effective number of parameters. Read more theory here <https://arxiv.org/abs/1507.04544> and here <https://arxiv.org/abs/1507.02646>

## Parameters

pointwise: bool, optional If True the pointwise predictive accuracy will be returned. Defaults to `stats.ic_pointwise` rcParam. `var_name` : str, optional The name of the variable in `log_likelihood` groups storing the pointwise log likelihood data to use for loo computation. `reff`: float, optional Relative MCMC efficiency, `ess / n` i.e. number of effective samples divided by the number of actual samples. Computed from trace by default. `scale`: str Output scale for loo. Available options are:

- `log` : (default) log-score
- `negative_log` :  $-1 * \text{log-score}$
- `deviance` :  $-2 * \text{log-score}$

A higher log-score (or a lower deviance or negative log\_score) indicates a model with better predictive accuracy.

## Returns

ELPDData object (inherits from `:class:pandas.Series`) with the following row/attributes: `elpd_loo`: approximated expected log pointwise predictive density (elpd) `se`: standard error of the elpd `p_loo`: effective number of parameters `n_samples`: number of samples `n_data_points`: number of data points `warning`: bool True if the estimated shape parameter of Pareto distribution is greater than `good_k`. `loo_i`: `:class:~xarray.DataArray` with the pointwise predictive accuracy, only if `pointwise=True` `pareto_k`: array of Pareto shape values, only if `pointwise=True` `scale`: scale of the elpd `good_k`: For a sample size  $S$ , the threshold is compute as  $\min(1 - 1/\log_{10}(S), 0.7)$

The returned object has a custom print method that overrides `pd.Series` method.

```
bi.dist.loo(  
self,  
pointwise=None,  
var_name=None,  
reff=None,  
scale=None,  
)
```

---

Perform comprehensive model diagnostics.

Creates various diagnostic plots including: - Posterior distributions - Autocorrelation plots - Trace plots - Rank plots - Forest plots

Stores plots in instance variables: `self.plot_posterior`, `self.autocor`, `self.traces`, `self.rank`, `self.forest`

```
bi.dist.model_checks(  
self,  
)
```

---

Create pairwise plots of the posterior distribution.

Args: `var_names` (list): Variables to include `kind` (list): Type of plots (“scatter” and/or “kde”) `kde_kwargs` (dict): Additional arguments for KDE plots `marginals` (bool): Include marginal distributions `point_estimate` (str): Point estimate to plot `figsize` (tuple): Size of the figure `*args, **kwargs`: Additional arguments for `arviz.plot_pair`

Returns: `fig`: Pair plot

```
bi.dist.pair(  
self,  
var_names=None,  
kind=['scatter', 'kde'],  
kde_kwargs={'fill_last': False},  
marginals=True,  
point_estimate='median',  
figsize=(11.5, 5),  
*args,  
**kwargs,  
)
```

---

Summary plot for model comparison.

Models are compared based on their expected log pointwise predictive density (ELPD). This plot is in the style of the one used in [2]\_. Chapter 6 in the first edition or 7 in the second.

## Notes

The ELPD is estimated either by Pareto smoothed importance sampling leave-one-out cross-validation (LOO) or using the widely applicable information criterion (WAIC). We recommend LOO in line with the work presented by [1]\_.

## Parameters

`comp_df` : pandas.DataFrame Result of the `:func:arviz.compare` method. `insample_dev` : bool, default False Plot in-sample ELPD, that is the value of the information criteria without the penalization given by the effective number of parameters (`p_loo` or `p_waic`). `plot_standard_error` : bool, default True Plot the standard error of the ELPD. `plot_ic_diff` : bool, default False Plot standard error of the difference in ELPD between each model and the top-ranked model. `order_by_rank` : bool, default True If True ensure the best model is used as reference. `legend` : bool, default False Add legend to figure. `figsize` : (float, float), optional If `None`, size is (6, num of models) inches. `title` : bool, default True Show a title with a description of how to interpret the plot. `textsize` : float, optional Text size scaling factor for labels, titles and lines. If `None` it will be autoscaled based on `figsize`. `labeller` : Labeller, optional Class providing the method `make_label_vert` to generate the labels in the plot titles. Read the `:ref:label_guide` for more details and usage examples. `plot_kwargs` : dict, optional Optional arguments for plot elements. Currently accepts `'color_ic'`, `'marker_ic'`, `'color_insample_dev'`, `'marker_insample_dev'`, `'color_dse'`, `'marker_dse'`, `'ls_min_ic'`, `'color_ls_min_ic'`, `'fontsize'` `ax` : matplotlib\_axes or bokeh\_figure, optional Matplotlib axes or bokeh figure. `backend` : {"matplotlib", "bokeh"}, default "matplotlib" Select plotting backend. `backend_kwargs` : bool, optional These are kwargs specific to the backend being used, passed to `:func:matplotlib.pyplot.subplots` or `:class:bokeh.plotting.figure`. For additional documentation check the plotting method of the backend. `show` : bool, optional Call backend show function.

## Returns

`axes` : matplotlib\_axes or bokeh\_figure

## See Also

`plot_elpd` : Plot pointwise elpd differences between two or more models. `compare` : Compare models based on PSIS-LOO `loo` or WAIC `waic` cross-validation. `loo` : Compute Pareto-smoothed importance sampling leave-one-out cross-validation (PSIS-LOO-CV). `waic` : Compute the widely applicable information criterion.

## References

- .. [1] Vehtari et al. (2016). Practical Bayesian model evaluation using leave-one-out cross-validation and WAIC <https://arxiv.org/abs/1507.04544>
- .. [2] McElreath R. (2022). Statistical Rethinking A Bayesian Course with Examples in R and Stan, Second edition, CRC Press.

```

bi.dist.plot_compare(
    comp_df,
    insample_dev=False,
    plot_standard_error=True,
    plot_ic_diff=False,
    order_by_rank=True,
    legend=False,
    title=True,
    figsize=None,
    textsize=None,
    labeller=None,
    plot_kwargs=None,
    ax=None,
    backend=None,
    backend_kwargs=None,
    show=None,
)

```

---

Plot evolution of effective sample size across iterations.

Returns: fig: ESS evolution plot

```

bi.dist.plot_ess(
    self,
)

```

---

Create a trace plot for visualizing MCMC diagnostics.

Args: var\_names (list): List of variable names to include kind (str): Type of plot (default: "rank\_bars") \*args, \*\*kwargs: Additional arguments for arviz.plot\_trace

Returns: plot: The trace plot object

```

bi.dist.plot_trace(
    self,
    var_names=None,
    kind='rank_bars',
    *args,

```

```
**kwargs,  
)
```

---

Create posterior distribution plots.

Args: figsize (tuple): Size of the figure (width, height)

Returns: fig: Matplotlib figure containing posterior plots

```
bi.dist.posterior(  
self,  
figsize=(8, 4),  
)
```

---

Visualize prior distributions compared to log probability.

Args: N (int): Number of samples to draw from priors

Returns: fig: Matplotlib figure containing the prior distribution plots

```
bi.dist.prior_dist(  
self,  
N=100,  
)
```

---

Create rank plots for MCMC chains.

Args: \*args, \*\*kwargs: Additional arguments for `arviz.plot_rank`

Returns: fig: Rank plots

```
bi.dist.rank(  
self,  
*args,  
**kwargs,  
)
```

---

Calculate R-hat statistics for convergence.

Args: *\*args*, *\*\*kwargs*: Additional arguments for `arviz.rhat`

Returns: `rhat`: R-hat values

```
bi.dist.rhat(  
self,  
*args,  
**kwargs,  
)
```

---

Calculate summary statistics for the posterior distribution.

Args: `round_to` (int): Number of decimal places to round results `kind` (str): Type of statistics to compute (default: “stats”) `hdi_prob` (float): Probability for highest posterior density interval *\*args*, *\*\*kwargs*: Additional arguments for `arviz.summary`

Returns: `pd.DataFrame`: Summary statistics of the posterior distribution

```
bi.dist.summary(  
self,  
round_to=2,  
kind='stats',  
hdi_prob=0.89,  
*args,  
**kwargs,  
)
```

---

Convert the sampler output to an arviz trace object.

This method prepares the trace for use with arviz diagnostic tools.

Returns: `self.trace`: The arviz trace object containing the diagnostic data

```
bi.dist.to_az(  
self,  
backend='numpyro',  
sample_stats_name=['target_log_prob', 'log_accept_ratio', 'has_divergence', 'energy'],  
)
```

---

Create trace plots for MCMC chains.

Args: \*args, \*\*kwargs: Additional arguments for `arviz.plot_trace`

Returns: fig: Trace plots

```
bi.dist.traces(  
self,  
*args,  
**kwargs,  
)
```