Installation

You can run BI on python or R. For R users you need to have installed python and the R **reticulate** package.

Python

```
pip install BayesInference
```

R

```
package.install(BayesInference)
```

Import BI class

Before anything else, you need to import the BI class. This will allow you to create a BI object that will be used to : 1) import data, 2) define the model, 3) fit the model, 4) summarize the results, and 5) plot the results.

Python

```
from BI import bi
m = bi()
```

R

```
library(BayesInference)
m=importBI(platform='cpu')
```

Import data

The BI class can import data from a csv file or from a dictionary for data that can't be stored in a tabular format.

Import tabular data from a csv file

Python

```
m.data(data_path, sep=';')
```

R

```
m$data(data_path,  sep=';')
```

Import non tabular data

First you need to create our own dictionary with the data.

Python

```
m.data_on_model = dict(
    ID1 = Value1,
    ID2 = Value2,
)
```

R

```
keys <- c("ID1","ID2")
values <- list(Value1,Value2)
data = py_dict(keys, values, convert = TRUE)
m$data_on_model=data
```

Define model

A function that declare a mathematical model need to be defined first and sent it to the BI class.

Python

```python
def model(ID1, ID2):
    # Define model here
    pass
```

R

```r
model <- function(ID1, ID2){
  # Define model here
  pass
}
```

Fit model

To fit your model, you need to use the `bi.fit` function. This function takes the model function as an argument and returns a `bi` object that contains the results of the model fitting. The `bi` object can be used to access the posterior distributions of the parameters, as well as the summary statistics of the model.

Python

```python
m.fit(model)
```

R

```r
m$fit(model)
```

Summary

The `bi` object also provides a summary of the model results. This summary includes the posterior distributions of the parameters, as well as the summary statistics of the model.

Python

```
m.summary()
```

R

```
m$summary()
```

Diagnostics

Bayesian models can be subject to various types of model misspecification, such as model over-fitting, model under-fitting, or model mis-specification. To assess the model's performance, you can use the `bi` object's `bi.diag` function. This function provides a range of diagnostic tools, including posterior predictive checks, posterior predictive distributions, and model comparisons.

Python

```
m.diag.pp_check(model, num_samples=100)
```

R

```
m$diag.pp_check(model, num_samples=100)
```