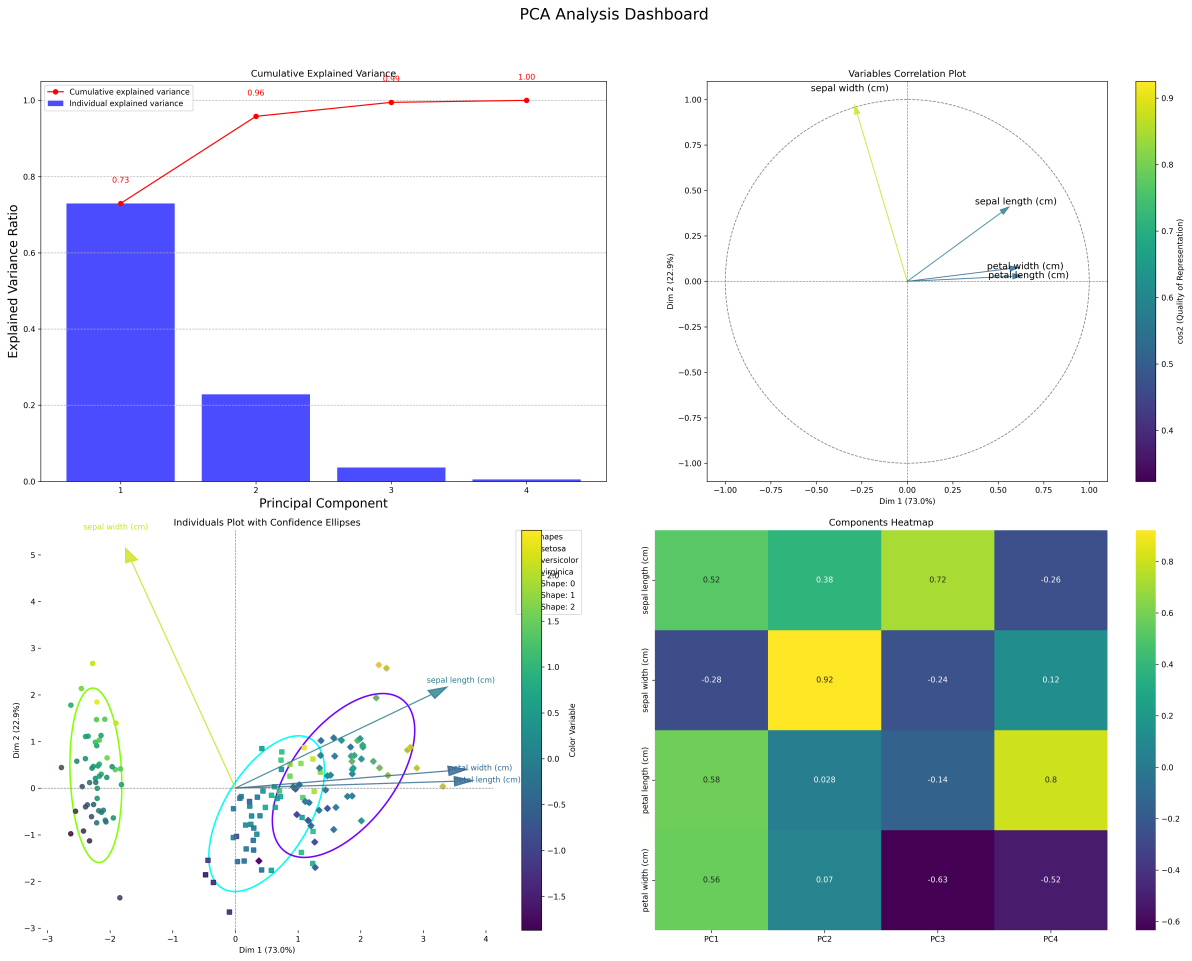
Build in models

PCA

```
from BI import bi, jnp

m=bi()
m.data('iris.csv', sep=',') # Data is already scaled
m.data_on_model = dict(
    X=jnp.array(m.df.iloc[:,0:-2].values)
)
m.fit(m.models.pca(type="ARD"), progress_bar=False) # or robust, sparse, classic, sparse_robu

m.models.pca.plot(
    X=m.df.iloc[:,0:-2].values,
    y=m.df.iloc[:,-2].values,
    feature_names=m.df.columns[0:-2],
    target_names=m.df.iloc[:,-1].unique(),
    color_var=m.df.iloc[:,0].values,
    shape_var=m.df.iloc[:,-2].values
)
```

```
jax.local_device_count 16
```

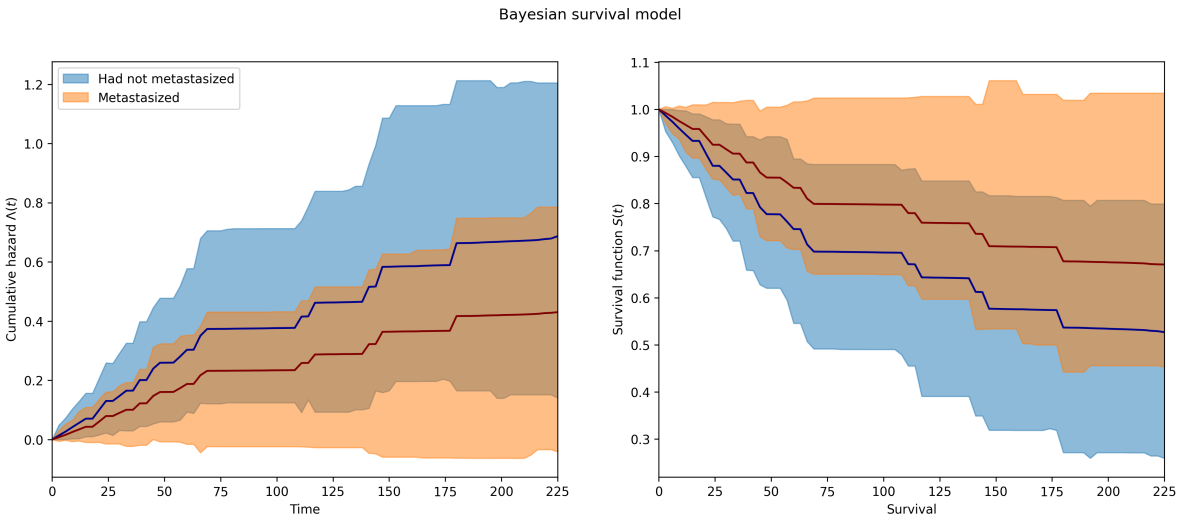


Survival analysis

```
from BI import bi, jnp

m = bi()
m.data('mastectomy.csv', sep=',').head()

m.df.metastasized = (m.df.metastasized.values == "yes").astype(jnp.int64)
m.models.survival.import_time_even(
    m.df.time.values,
    m.df.event.values, interval_length=3
) # Import time-steps and events
```

```
m.models.survival.import_covF(
    m.df.metastasized.values, ['metastasized']
) # Import time-fixed covariates
# m.models.survival.import_covV # To import time-varying covariates

m.fit(m.models.survival.model, num_samples=500)
m.summary()

m.models.survival.plot_surv( beta = 'Hazard_rate_metastasized')
```

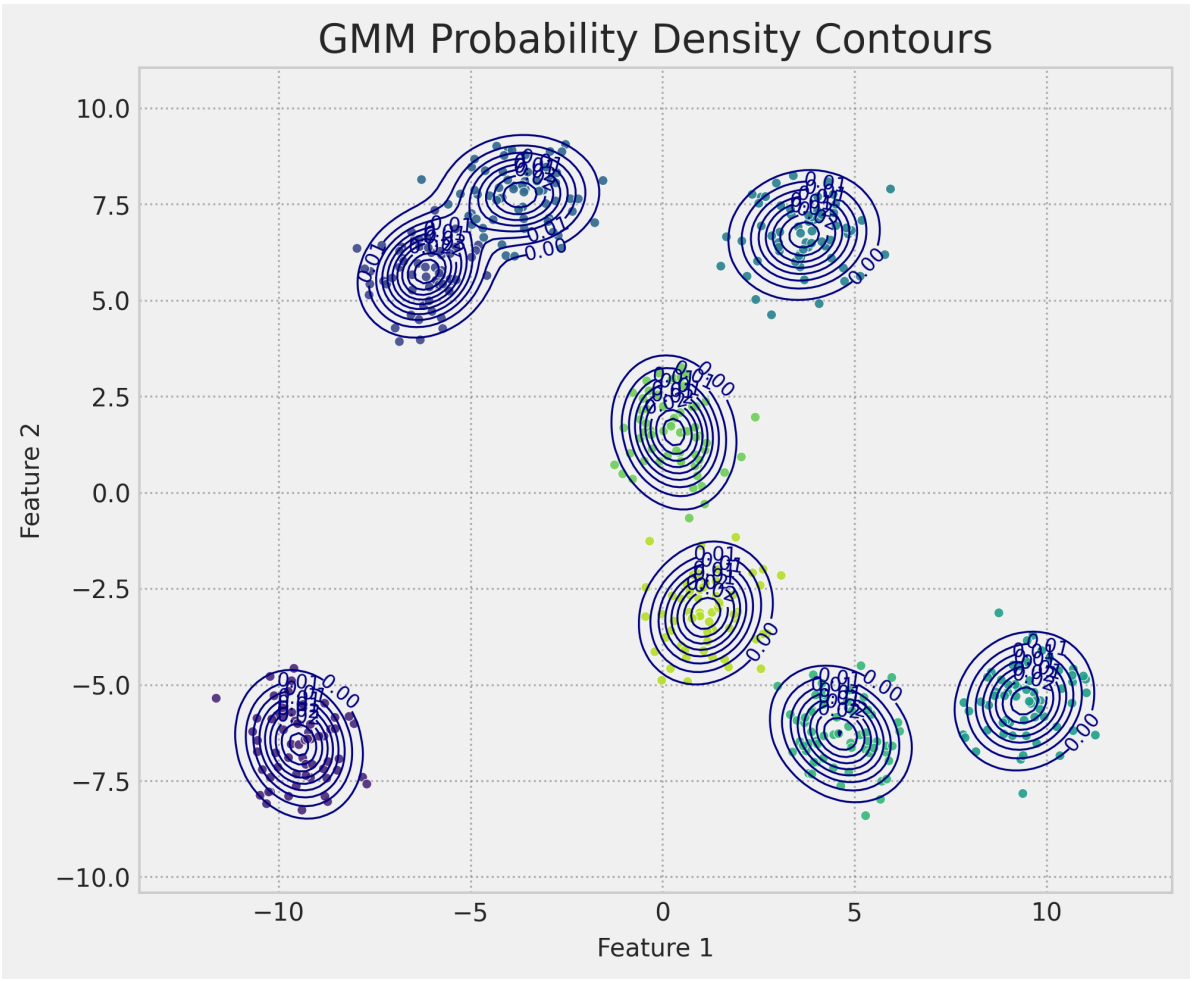
```
jax.local_device_count 16
--------------------------------------------------------------------------------
Survival concern 44 individuals in 76 intervals.
18.0 individuals experienced the event.
--------------------------------------------------------------------------------
Covariates imported: ['metastasized']
Surv object now has 1 covariates: ['metastasized']

  0%|          | 0/1000 [00:00<?, ?it/s]warmup:   0%|          | 1/1000 [00:01<29:12,  1.75s/
```



Gaussian Mixture Models

```
from BI import bi
from sklearn.datasets import make_blobs

# Generate synthetic data
data, true_labels = make_blobs(
    n_samples=500, centers=8, cluster_std=0.8,
    center_box=(-10,10), random_state=101
)


m.data_on_model = {"data": data,"K": 8 }
m.fit(m.models.gmm) # Optimize model parameters through MCMC sampling
m.plot(X=data,sampler=m.sampler) # Prebuild plot function for GMM
```

```
  0%|          | 0/1000 [00:00<?, ?it/s]warmup:   0%|          | 1/1000 [00:03<1:06:20,  3.98
```




```
Model not supported
```

Dirichlet Process Mixture Models

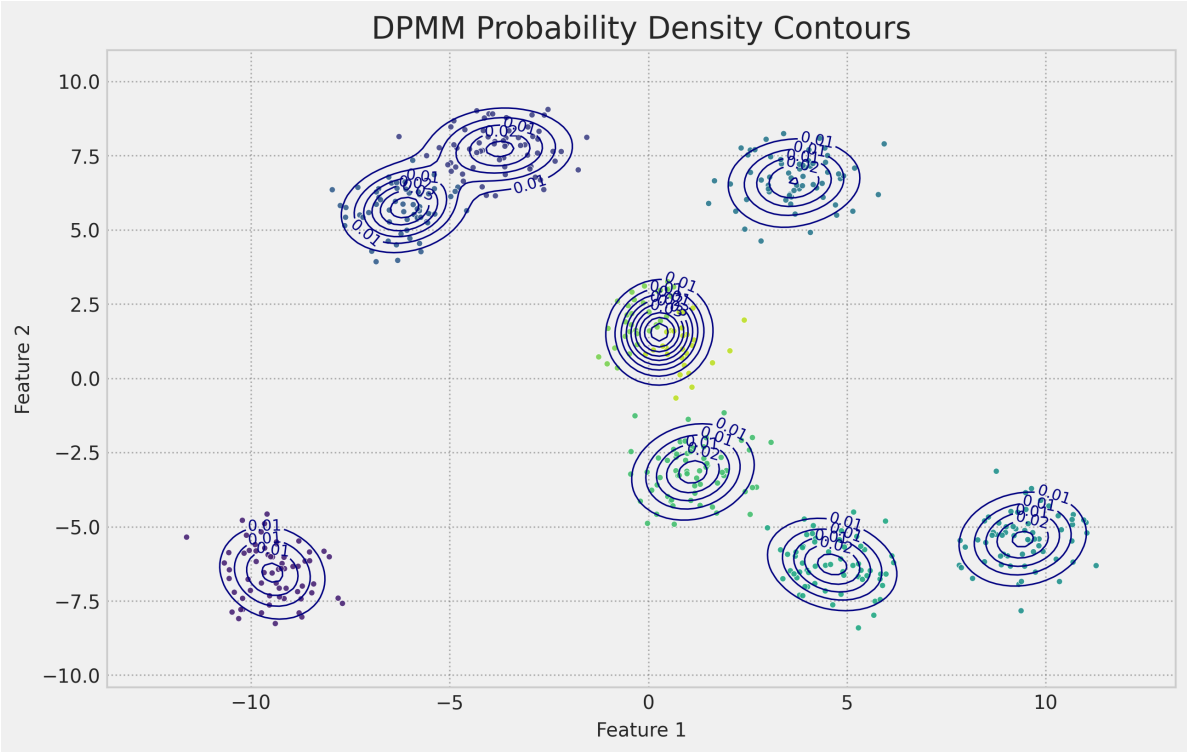
Python

```python
from BI import bi
from sklearn.datasets import make_blobs

# Generate synthetic data
data, true_labels = make_blobs(
    n_samples=500, centers=8, cluster_std=0.8,
```

```
    center_box=(-10,10), random_state=101
)

m.data_on_model = {"data": data, "T": 10 } # T = Number of maximum cluster to test for
m.fit(m.models.dpmm)
m.plot(X=data,sampler=m.sampler) # Prebuild plot function for GMM
```

```
  0%|          | 0/1000 [00:00<?, ?it/s]warmup:   0%|          | 1/1000 [00:03<1:04:07,  3.85
```

```
Model found 9 clusters.
```



Network Models

Python

```
from BI import bi

# Setup device-------------------------------------------------
from BI import bi
# Setup device-------------------------------------------------
m = bi(platform='cpu')

m.data_on_model = dict(
    idx = idx,
    Any = Any-1,
    Merica = Merica-1,
    Quantum = Quantum-1,
    result_outcomes = m.net.mat_to_edgl(data['outcomes']),
    kinship = m.net.mat_to_edgl(kinship),
    focal_individual_predictors = data['individual_predictors'],
    target_individual_predictors = data['individual_predictors'],
    exposure_mat = data['exposure']
)


def model(idx, result_outcomes,
    exposure,
    kinship,
    focal_individual_predictors, target_individual_predictors,
    Any, Merica, Quantum):
      # Block ---------------------------------------
      B_any = m.net.block_model(Any,1)
      B_Merica = m.net.block_model(Merica,3)
      B_Quantum = m.net.block_model(Quantum,2)

      ## SR shape =  N individuals---------------------------------------
      sr =  m.net.sender_receiver(focal_individual_predictors,target_individual_predictors)

      # Dyadic shape = N dyads-----------------------------------
      dr = m.net.dyadic_effect(dyadic_predictors)

      m.dist.poisson(jnp.exp(B_any + B_Merica + B_Quantum + sender_receiver + dr), obs = resu



m.fit(model)
summary = m.summary()
```

```
summary.loc[['focal_effects[0]', 'target_effects[0]', 'dyad_effects[0]']]

m.fit(model2)
summary = m.summary()
summary
```

R

```
library(BI)
# Setup platform-------------------------------------------------
m=importbi(platform='cpu')

# Import data ---------------------------------------------------

load(paste(system.file(package = "BI"),'/data/STRAND sim sr only.Rdata', sep = ''))

ids = 0:(model_dat$N_id-1)
idx = m$net$vec_node_to_edgle(jnp$stack(jnp$array(list(ids, ids)), axis = -as.integer(1)))

keys <- c("idx",
          'idxShape',
          "result_outcomes",
          'focal_individual_predictors',
          'target_individual_predictors')

values <- list(
  idx,
  idx$shape[[1]],
  m$net$mat_to_edgl(model_dat$outcomes[,,1]),
  jnp$array(model_dat$individual_predictors)$reshape(as.integer(1),as.integer(50)),
  jnp$array(model_dat$individual_predictors)$reshape(as.integer(1),as.integer(50))
)
data = py_dict(keys, values, convert = TRUE)
m$data_on_model=data

# Define model ---------------------------------------------------
model <- function(idx, idxShape, result_outcomes,focal_individual_predictors, target_individu
  N_id = 50
  x=0.1/jnp$sqrt(N_id)
  tmp=jnp$log(x / (1 - x))
```

```
  ## Block ---------------------------------------
  B = bi.dist.normal(tmp, 2.5, shape=c(1), name = 'block')

  #SR ---------------------------------------
  sr =  m$net$sender_receiver(focal_individual_predictors,target_individual_predictors)

  ### Dyadic---------------------------------------
  #dr, dr_raw, dr_sigma, dr_L = m.net.dyadic_random_effects(idx.shape[0], cholesky_density =
  dr = m$net$dyadic_effect(shape = c(idxShape))

  ## SR ---------------------------------------
  m$poisson(jnp$exp(B + sr + dr), obs=result_outcomes)
}

# Run MCMC ------------------------------------------------
m$fit(model) # Optimize model parameters through MCMC sampling

# Summary ------------------------------------------------
summary =m$summary()

summary[rownames(summary) %in% c('focal_effects[0]', 'target_effects[0]', 'block[0]'),]
```