

**Predictive Analytics - Regression and Classification**  
**Prof. Sourish Das**  
**Department of Mathematics**  
**Chennai Mathematical Institute**

**Lecture - 60**

**Hands on with Julia\_Bayesian Logistic Regression with Horse Shoe Prior\_Genetic Data Analysis**

Hello all, welcome to the hands on exercise for this weeks. In this hands on exercise, we are going to do some logistic regression analysis using Julia from genetics data set that are available in R.

(Refer Slide Time: 00:39)

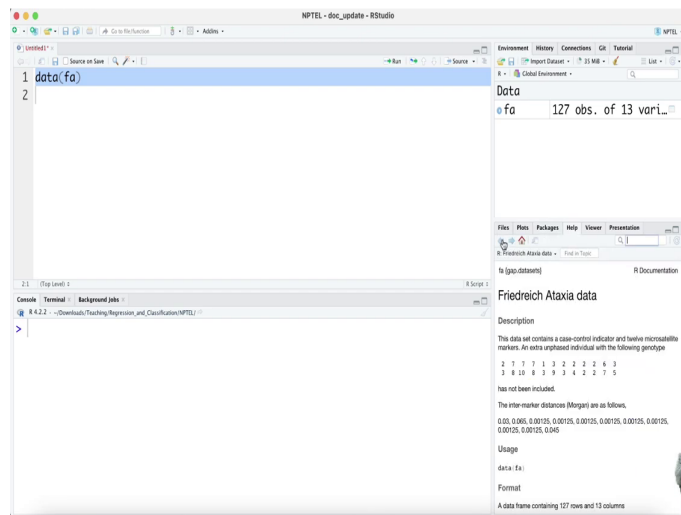
The screenshot shows the RStudio interface. The main window displays a data frame 'fa' with 127 observations and 13 variables (y, loc1, loc2, loc3, loc4, loc5, loc6, loc7, loc8, loc9, loc10, loc11, loc12). The console shows the output of a function call, listing values for 'y' and 'x' variables. The output is as follows:

```
70 14
71 10
72 9
73 11
74 9
75 11
76 9
```

The console also shows the function call: `[reached 'max' /getOption("max.print") -- omitted 51 rows]` and the command `> View(fa)`.



(Refer Slide Time: 00:41)



The screenshot shows the RStudio interface. The source editor contains the following code:

```
1 data(fa)
2
```

The Environment pane on the right shows a data frame named 'fa' with 127 observations and 13 variables.

The console shows the command `data(fa)` being executed. The help window for 'fa' (Friedreich Ataxia data) is open, displaying the following information:

**Friedreich Ataxia data**

Description

This data set contains a case-control indicator and twelve microsatellite markers for each affected individual with the following genotype:

```
2 7 7 1 3 2 2 2 6 3
3 8 10 8 3 3 4 2 2 7 5
```

has not been included.

The inter-marker distances (Morgans) are as follows:


```
0.03, 0.0465, 0.00115, 0.00115, 0.00115, 0.00115, 0.00115, 0.00115,
0.00115, 0.00115, 0.0465
```

Usage

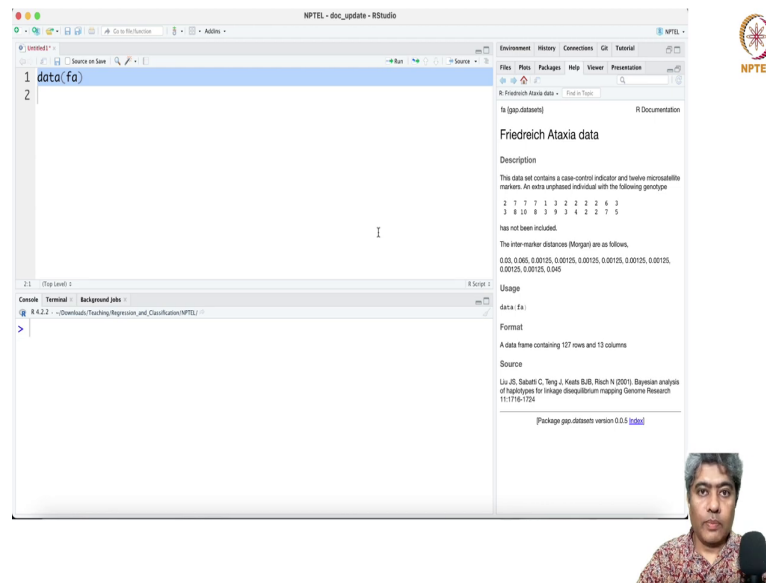
```
data(fa)
```

Format

A data frame containing 127 rows and 13 columns



(Refer Slide Time: 00:55)



The screenshot shows the RStudio interface. The source editor on the left contains the R code `data(fa)`. The console at the bottom shows the R version `R 4.2.2` and the file path `~/Downloads/Teaching/Regression_and_Classification/NPTEL/`. The right-hand pane displays the documentation for the `fa` data set from the `gap` package. The documentation includes a description of the data set, a list of inter-marker distances (Morgan), and the source information.

**Friedreich Ataxia data**

**Description**

This data set contains a case-control indicator and twelve microsatellite markers for each individual with the following genotype:

2	7	7	1	3	2	2	2	6	3	
3	8	10	8	3	3	4	2	2	7	5

has not been included.

The inter-marker distances (Morgan) are as follows:

0.03	0.040	0.00135	0.00125	0.00125	0.00125	0.00125	0.00125	0.00125	0.00125	0.00125	0.00125	0.00125
------	-------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------	---------

Usage

`data(fa)`

Format

A data frame containing 127 rows and 13 columns

Source

Liu J.S, Sabatti C, Teng J, Kessler B.B, Reich N (2001) Bayesian analysis of haplotypes for linkage disequilibrium mapping Genome Research 11:1716-1724

Package: gap, datasets version 0.0-9 [help](#)

So, first I am going to start R. In R, you have this genetics in the package there is a called gap package Genetic Analysis Package. And in the genetic analysis package, there is a data called fa data, fa data, Friedreich Ataxia, data. We are going to use this data was first appeared in the genome research in 2001.

We are going to use this data to implement logistic regression using Julia. So, one of the advantage of Julia is many of the packages that are available in R are also available in Julia or from Julia you can call those data set using R data set from a package.

(Refer Slide Time: 01:37)

**Logistic Regression Analysis of Friedrich Ataxia data**

**Source**

Liu JS, Sabatti C, Tang J, Keats B, Risch N (2001), Bayesian analysis of haplotypes for linkage disequilibrium mapping, *Genome Research* 11:1716-1724

```
In [1]: using CRRAo, RDatasets, StableRNG, StatModels


In [2]: fa = dataset("gap", "fa");
        first(fa,10)

In [3]: using Random
        randomeverything!(1234);
        function splitdf(df, pct)
            R.seed(0)
            pct = pct * 100
            ids = collect(keys(df, 1))
            shuffle!(ids)
            sel = ids .* rand{Bool}(df) .* pct
            train = df[sel, :]
            test = df[!sel, :]
            return train, test
        end

In [4]: train, test = splitdf(fa, 0.7);

In [5]: first(train,10)
```

**Maximum Likelihood Method**



So, let me go to Julia my Jupyter Notebook. I have prepared this Jupyter Notebook to an extent to some extent and I will use this Jupyter Notebook to an extent this thing. So, the here is the in R data set you can see I have first what I am going to do let me just open this. So, maybe I will just do, ok. So, like here it is a Friedreich Ataxia data. So, I am going to give a name of you know.

Ah So, maybe Logistic Regression Analysis of Ataxia data, ok. So, here is the source of the data I have given and so, maybe I will just turn it like this and this is the first thing I am going to call CRRAo RDatasets StableRNG and StatModels, ok. So, this is first I am going to call this packages then I am going to from the RDatasets I am just calling RDatasets this is the name of the package gap package from R and fa is the name of the data set. So, I am let me just call this.

(Refer Slide Time: 03:09)

The screenshot shows a Jupyter Notebook interface with the following code and output:

```
Source
Liu JS, Sabatti C, Teng J, Keetha B, R. Rich N (2001) Bayesian analysis of haplotypes for linkage disequilibrium mapping. Genome Research 11:1716-1724

In [1]: using CR Rao, R Datasets, StableRNGs, StatsModels

In [2]: fa = dataset("gsp", "fa");
        df = fa[1:10]

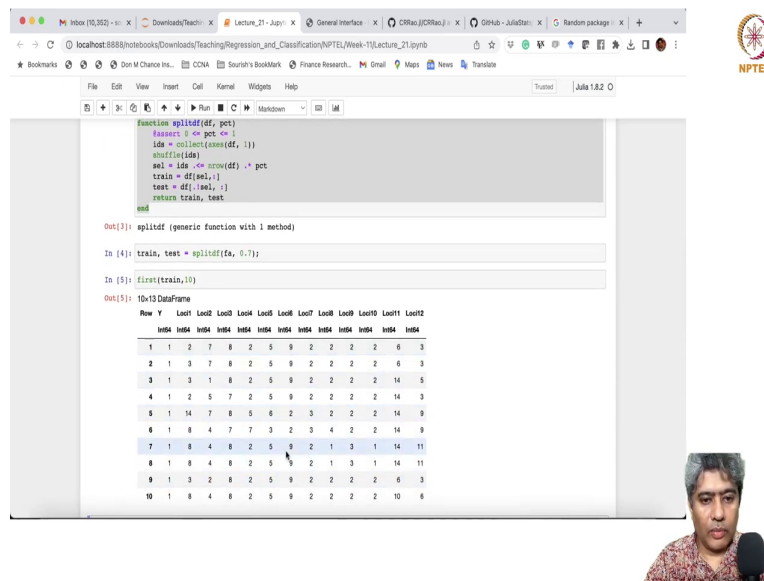
Out[2]: 10x13 DataFrame
         Y  Lect1 Lect2 Lect3 Lect4 Lect5 Lect6 Lect7 Lect8 Lect9 Lect10 Lect11 Lect12
Row 1  0     1     2     7     8     2     5     9     2     2     2     2     6     3
      2  1     3     7     8     2     5     9     2     2     2     2     2     6     3
      3  1     3     1     8     2     5     9     2     2     2     2     2     14     5
      4  1     2     5     7     2     5     9     2     2     2     2     2     14     3
      5  1    14     7     8     5     6     2     3     2     2     2     2     14     9
      6  1     8     4     7     7     3     2     3     4     2     2     2     14     9
      7  1     8     4     8     2     5     9     2     1     3     1     14    11
      8  1     8     4     8     2     5     9     2     1     3     1     14    11
      9  1     3     2     8     2     5     9     2     2     2     2     2     6     3
     10  1     8     4     8     2     5     9     2     2     2     2     2     10     6

In [ ]: using Random
        @reproducible(1234)
        function splitdf(df, pct)
            assert 0 <= pct <= 1
            ids = collect(randi(100000, 100000))
```

The output shows a 10x13 DataFrame with columns Y, Lect1, Lect2, Lect3, Lect4, Lect5, Lect6, Lect7, Lect8, Lect9, Lect10, Lect11, and Lect12. The first column (Y) contains 0s and 1s, and the subsequent columns (Lect1-Lect12) contain numerical values representing features.

So, here we have the first column is the target column Y it has 0 1s and then there are 12 columns, ok. There are 12 columns which are we will be using as a predictor variable.

(Refer Slide Time: 03:41)



The screenshot shows a Jupyter Notebook interface with the following code and output:

```
function splitdf(df, pct)
    @assert 0 <= pct <= 1
    ids = collect(keys(df, :))
    shuffle(ids)
    sel = ids <= sort(ids) .* pct
    train = df[sel,:]
    test = df[!sel,:]
    return train, test
end

Out[3]: splitdf (generic function with 1 method)

In [4]: train, test = splitdf(fe, 0.7);

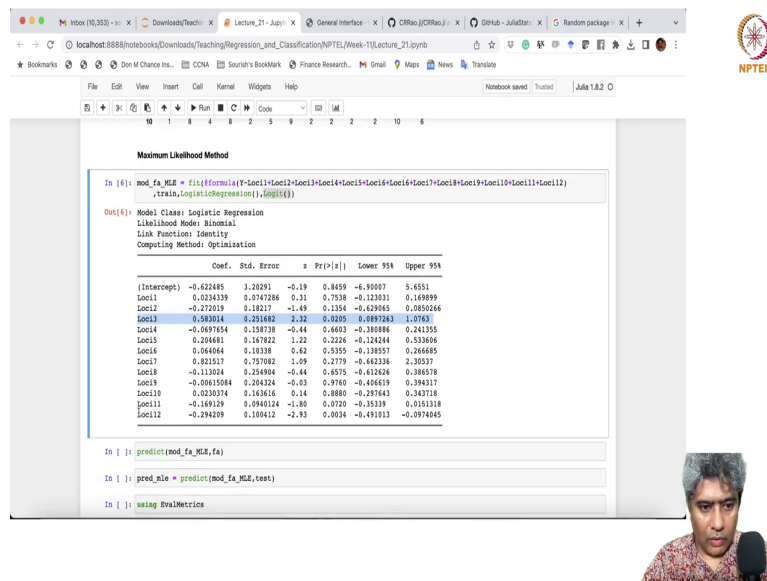
In [5]: first(train,10)

Out[5]: 10x13 DataFrame
  Row Y  Loc1  Loc2  Loc3  Loc4  Loc5  Loc6  Loc7  Loc8  Loc9  Loc10  Loc11  Loc12
  Int64 Int64 Int64 Int64 Int64 Int64 Int64 Int64 Int64 Int64 Int64 Int64 Int64
1  1  2  7  8  2  5  9  2  2  2  2  6  3
2  1  3  7  8  2  5  9  2  2  2  2  6  3
3  1  3  1  8  2  5  9  2  2  2  2  14  5
4  1  2  5  7  2  5  9  2  2  2  2  14  3
5  1  14  7  8  5  8  2  3  2  2  2  14  9
6  1  8  4  7  7  3  2  3  4  2  2  14  9
7  1  8  4  8  2  5  9  2  1  3  1  14  11
8  1  8  4  8  2  5  9  2  1  3  1  14  11
9  1  3  2  8  2  5  9  2  2  2  2  6  3
10 1  8  4  8  2  5  9  2  2  2  2  10  6
```

Now, what I am going to do in this if you just instead of actually run the first thing. So, here I am going to run this piece of code in this piece of code I am going to split the data frame into train and test. You can see that first time taking the percentage between 0 and 1 and then I am collecting the IDs of the data frame and then kind of shuffling them.

And 70 percent of that I am keeping it in the train and rest of them I am keeping in the test and then I am returning train and test. So, I am calling this split data frame function and splitting it into train and test. And so, this is the first 10 rows of the training data set.

(Refer Slide Time: 04:34)



Maximum Likelihood Method

```
In [6]: mod_fa_MLE = fit(formula=(Y~Loci1+Loci2+Loci3+Loci4+Loci5+Loci6+Loci7+Loci8+Loci9+Loci10+Loci11+Loci12),
                        train,logisticRegression(),logit())
```

```
Out[6]: Model Class: Logistic Regression
Likelihood Model: Binomial
Link Function: Identity
Computing Method: Optimization
```

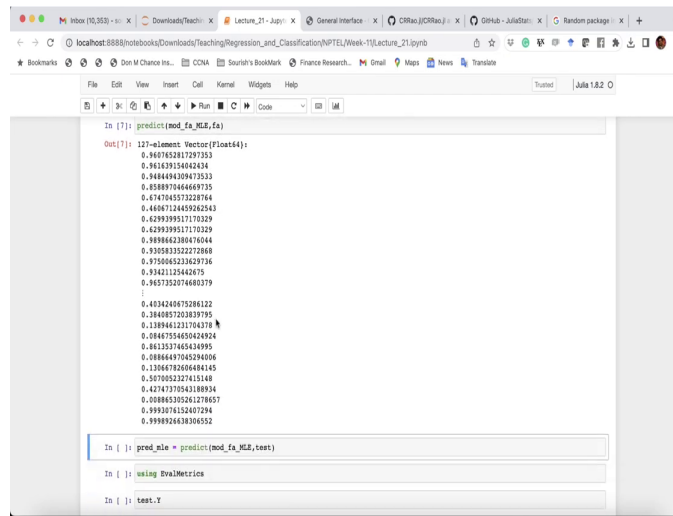
	Coef.	Std. Error	z	Pr(> z )	Lower 95%	Upper 95%
(Intercept)	-0.422485	1.26291	-0.33	0.8459	-6.90007	5.6551
Loci1	0.0234339	0.0747286	0.31	0.7538	-0.123031	0.169899
Loci2	-0.272019	0.180217	-1.49	0.1354	-0.629065	0.0850266
Loci3	0.583034	0.251682	2.32	0.0205	0.0697828	1.07638
Loci4	-0.0697454	0.158738	-0.44	0.6603	-0.380886	0.241355
Loci5	0.204681	0.187822	1.22	0.2226	-0.124204	0.533606
Loci6	0.044064	0.10328	0.42	0.3355	-0.138557	0.266685
Loci7	0.821517	0.757082	1.09	0.2779	-0.662336	2.38537
Loci8	-0.113024	0.254904	-0.44	0.6575	-0.612026	0.386578
Loci9	-0.08615084	0.204234	-0.43	0.3760	-0.406159	0.394317
Loci10	0.0230374	0.143616	0.14	0.8880	-0.297643	0.343718
Loci11	-0.169129	0.0940124	-1.80	0.0720	-0.35339	0.0151318
Loci12	-0.294209	0.109412	-2.63	0.0074	-0.491013	-0.0974045

```
In [7]: predict(mod_fa_MLE,fa)
In [8]: pred_mle = predict(mod_fa_MLE,test)
In [9]: using EvalMetrics
```

Then first I am going to implement the maximum likelihood of a method of logistic regression in that I am going to call fit from the CRRao. First, I have to provide the formula Y is the target and then all the predictors that I want to fit all the predictors that I have given here another data frame is train the I want to fit a logistic regression with Logit link. So, if I just run this. So, here is the output.

Now, if you see the P value for most of these Loci are not great except the third Loci third predictor Loci3 and then these are also not effective and Loci11 and 12 have effective here. So, these are the out of 12 Loci only three have effective kind of effect on the target variable.

(Refer Slide Time: 05:52)



```
In [7]: predict(mod_fa_MSE, fa)

Out[7]: 127-element Vector{Float64}:
 0.967165311739733
 0.961639154042434
 0.9484494309473533
 0.858875464646735
 0.6747045573228764
 0.44087124459262543
 0.6289399511710329
 0.6299399511710329
 0.998862380476044
 0.93583322272868
 0.9750665233629736
 0.93421125462675
 0.963732074680279
 0.4574242617928122
 0.3848857205839795
 0.1389461231704378
 0.08487554850424924
 0.861353165834895
 0.08866497045294006
 0.1206878206484145
 0.507805327151148
 0.42747370543188934
 0.00896305261278657
 0.9930761526407294
 0.998926638306552

In [8]: pred_mse = predict(mod_fa_MSE, test)

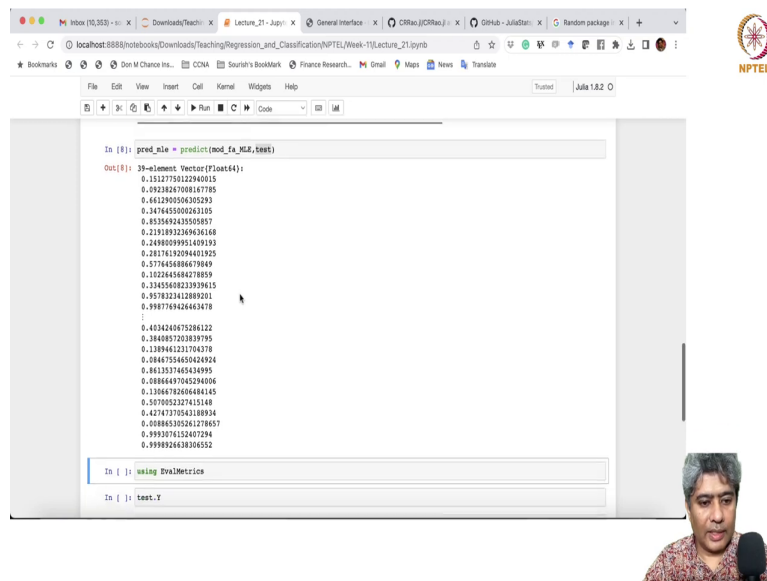
In [9]: using EvalMetrics

In [10]: test_Y
```





(Refer Slide Time: 06:00)



The screenshot shows a Jupyter Notebook interface with a browser window in the background. The notebook code cell contains the following text:

```
In [8]: pred_nle = predict(mod_fa_MLR, test)
```

The output of the cell is a 39-element vector of probability scores, displayed as follows:

```
Out[8]: 39-element Vector{Float64}:  
 0.151277591222940015  
 0.09328267008167785  
 0.661290506305293  
 0.347455006203185  
 0.8536464335050657  
 0.22918932369636168  
 0.2498009951409193  
 0.2817632096401395  
 0.577454886679849  
 0.102264584278859  
 0.334546823939615  
 0.957823412889201  
 0.998769426663478  
  
 0.403424675286122  
 0.3848897203839795  
 0.1389463211764378  
 0.0846754565044924  
 0.861357145434995  
 0.0886467045294086  
 0.1366678266484145  
 0.5070952327415148  
 0.4278270563188934  
 0.00886335261278657  
 0.9993076152407294  
 0.99892648306552
```

Below the output, there are two more code cells:

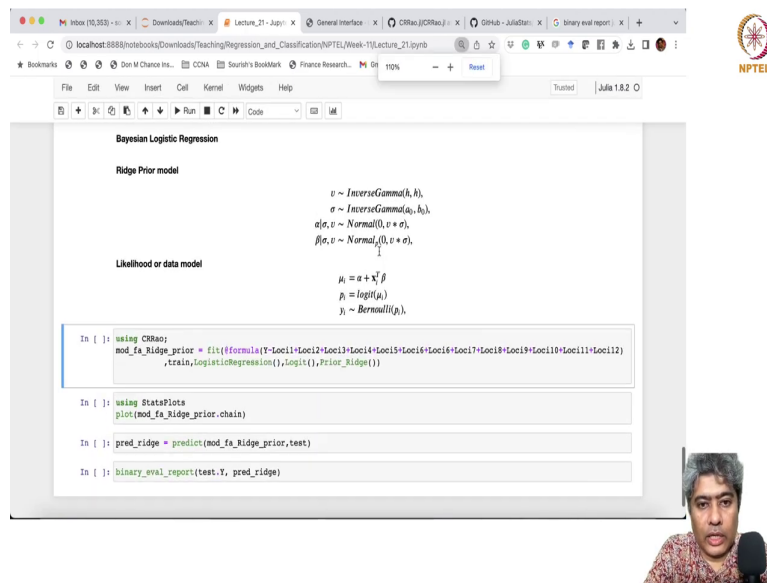
```
In [ ]: using EvalMetrics  
In [ ]: test_T
```

In the bottom right corner, there is a small video inset showing a man with grey hair and a patterned shirt, likely the instructor.

Now, using this model we can have a predictive we can predict the probability score on the you know on the train data set on the entire data set. Or similarly, if you just say predict you could call the predict function from CRRao and provide the model name and the test data set you can. So, you can I can I do not need this actually I need this one I need to calculate the predict on the test data set.



(Refer Slide Time: 06:52)



The screenshot shows a Jupyter Notebook interface with the following content:

**Bayesian Logistic Regression**

**Ridge Prior model**

$$\begin{aligned} v &\sim \text{InverseGamma}(h, \theta), \\ \sigma &\sim \text{InverseGamma}(a_0, b_0), \\ \alpha | \sigma, v &\sim \text{Normal}(\mu, \tau + \sigma), \\ \beta | \sigma, v &\sim \text{Normal}(\mathbf{0}, v + \sigma). \end{aligned}$$

**Likelihood or data model**

$$\begin{aligned} \mu_i &= \alpha + \mathbf{x}_i^T \beta \\ p_i &= \text{logit}(\mu_i) \\ y_i &\sim \text{Bernoulli}(p_i). \end{aligned}$$

```
In [ ]: using CR Rao;
mod_fa_Ridge_prior = fit((formula(Y~Loc1+Loc2+Loc3+Loc4+Loc5+Loc6+Loc7+Loc8+Loc9+Loc10+Loc11+Loc12)
,train,logisticRegression()),logit(),Prior_Ridge())

In [ ]: using StatsPlots
plot(mod_fa_Ridge_prior.chain)

In [ ]: pred_ridge = predict(mod_fa_Ridge_prior,test)

In [ ]: binary_eval_report(test.Y, pred_ridge)
```

NPTEL logo is visible in the top right corner of the notebook interface.

So, in the test from the test data set I am just taking the Y and binary eval report if I just give the fine test dot Y the actual Y values and the predicted score, ok. So, predict mle contains the all the probability scores, right. So, now if I just run this gives me the confusion matrix. Similarly, now if I have to if I want to implement the Bayesian Logistic Regression say Logistic Regression so, with say ridge prior model.

So, here if you see that Y follow Bernoulli  $\mu_i$  where  $\mu_i$  follow  $\alpha + \mathbf{x}_i^T \beta$ . And here is that that is my likelihood model and then well actually I should not say this actually  $\mu_i$  that should be like  $p_i$   $p_i$  it should be  $p_i$  actually. And then  $p_i$  is (Refer Time: 08:07) equal to this logit  $\mu_i$  think now that is a right way of writing it and then  $\mu_i$  equal to  $\mathbf{x}_i^T \beta$ , which  $\alpha + \mathbf{x}_i^T \beta$  and then.

So, alpha follows say normal 0 and beta follow normal 0 v sigma follow inverse gamma and v follow some another inverse gamma then this model this is typically we will implement as ridge regression model. Now, what we can do if you see the way we have implemented the regression simple Logistic Regression model almost similar way we can implement it here except that at the end I am just using prior ridge.

(Refer Slide Time: 09:17)

```

In [17]: using CR Rao;
mod_fa_Ridge_prior = fit((formula(Y~Loc1+Loc2+Loc3+Loc4+Loc5+Loc6+Loc7+Loc8+Loc9
,train,LogisticRegression()),Logit(),Prior_Ridge())

Info: Found initial step size
L
ε = 0.025
Sampling: 100% Time: 0:00:03

Out[17]: Formula: Y ~ 1 + Loc1 + Loc2 + Loc3 + Loc4 + Loc5 + Loc6 + Loc7 + Loc8 + Loc9 + Loc1
0 + Loc11 + Loc12
Link: Logit(CR Rao.Logit_Link)
Chain: Chains MCMC chain (1000×26×1 Array{Float64, 3}):

Iterations = 501:1:1500
Number of chains = 1
Samples per chain = 1000
Wall duration = 9.21 seconds
Compute duration = 9.21 seconds
parameters = λ, β[1], β[2], β[3], β[4], β[5], β[6], β[7], β[8], β[9], β[10], β[11], β[1
2], β[13]
internals = lp, n_steps, is_accept, acceptance_rate, log_density, hamiltonian_energy,
hamiltonian_energy_error, max_hamiltonian_energy_error, tree_depth, numerical_error, step_siz
e, nom_step_size

Summary Statistics
parameters mean std mcmc ess_bulk ess_tail rhat -
Symbol Float64 Float64 Float64 Float64 Float64 Float64 -

```

I am just calling ridge prior here and if I just run it. So, it will take few seconds maybe ok it is taking about few second it took about, alright.

(Refer Slide Time: 09:35)

The screenshot shows a Julia 1.8.2 terminal window displaying the output of a regression analysis. The output is divided into two sections: 'Summary Statistics' and 'Quantiles'. The 'Summary Statistics' section lists parameters  $\lambda$ ,  $\beta[1]$  through  $\beta[13]$  with columns for mean, std, mse, ess\_bulk, ess\_tail, and rhat. The 'Quantiles' section lists the same parameters with columns for 2.5th, 25th, 50th, 75th, and 97.5th percentiles. The rhat values are consistently near 1.0, indicating convergence. A small video inset of a person is visible in the bottom right corner of the terminal window.

parameters	mean	std	mse	ess_bulk	ess_tail	rhat
$\lambda$	0.1952	0.0713	0.0043	235.1529	304.9297	1.0084
$\beta[1]$	0.0228	0.2028	0.0063	1108.8722	625.0235	0.9995
$\beta[2]$	0.0441	0.0627	0.0019	1061.1883	720.8847	1.0004
$\beta[3]$	-0.1462	0.1165	0.0042	768.3125	625.3696	0.9995
$\beta[4]$	0.3837	0.1444	0.0080	315.5710	488.5369	1.0022
$\beta[5]$	-0.0217	0.0926	0.0027	1131.3778	701.4886	1.0015
$\beta[6]$	0.0870	0.1088	0.0040	765.8883	653.6702	0.9991
$\beta[7]$	0.0672	0.0672	0.0021	987.7819	646.5443	1.0010
$\beta[8]$	0.1850	0.1769	0.0071	724.1068	698.9634	1.0030
$\beta[9]$	-0.0528	0.1391	0.0041	1165.3020	601.7201	1.0002
$\beta[10]$	-0.0254	0.1154	0.0035	1065.3179	737.6944	0.9999
$\beta[11]$	0.0131	0.1053	0.0030	1194.4092	709.8982	1.0004
$\beta[12]$	-0.0833	0.0708	0.0034	437.7141	517.8896	1.0026
$\beta[13]$	-0.2036	0.0788	0.0036	470.5635	758.8753	1.0004

Quantiles	2.5%	25.0%	50.0%	75.0%	97.5%
$\lambda$	0.0872	0.1416	0.1833	0.2369	0.3524
$\beta[1]$	-0.3863	-0.1038	0.0222	0.1381	0.4155
$\beta[2]$	-0.0634	0.0005	0.0429	0.0881	0.1600
$\beta[3]$	-0.4000	-0.2230	-0.1344	-0.0667	0.0698

So, it took most of the and if you see that all of them are kind of near one so; that means, if that is near one; that means, the convergence has taken place.

(Refer Slide Time: 09:51)

The screenshot shows a Julia REPL window with the following output:

```
β[7] 0.0672 0.0672 0.0021 987.7819 646.5443 1.0010 --
β[8] 0.1850 0.1769 0.0071 724.1068 698.9634 1.0030 --
β[9] -0.0528 0.1391 0.0041 1165.3020 601.7201 1.0002 --
β[10] -0.0254 0.1154 0.0035 1065.3179 737.6944 0.9999 --
β[11] 0.0131 0.1053 0.0030 1194.4092 709.8982 1.0004 --
β[12] -0.0833 0.0708 0.0034 437.7141 517.8896 1.0026 --
β[13] -0.2036 0.0788 0.0036 470.5635 758.8753 1.0004 --
1 column omitted

Quantiles
parameters 2.5% 25.0% 50.0% 75.0% 97.5%
Symbol Float64 Float64 Float64 Float64 Float64

λ 0.0872 0.1416 0.1833 0.2369 0.3524
β[1] -0.3863 -0.1038 0.0222 0.1381 0.4155
β[2] -0.0634 0.0005 0.0429 0.0881 0.1600
β[3] -0.4000 -0.2230 -0.1344 -0.0667 0.0699
β[4] 0.1251 0.2787 0.3806 0.4759 0.6932
β[5] -0.2090 -0.0813 -0.0192 0.0367 0.1640
β[6] -0.1055 0.0123 0.0832 0.1520 0.3040
β[7] -0.0656 0.0200 0.0681 0.1123 0.1995
β[8] -0.0779 0.0543 0.1573 0.2917 0.5929
β[9] -0.2854 -0.1823 -0.0468 0.0380 0.2040
β[10] -0.2450 -0.1024 -0.0258 0.0510 0.2019
β[11] -0.1942 -0.0543 0.0149 0.0782 0.2216
β[12] -0.2342 -0.1263 -0.0787 -0.0329 0.0420
β[13] [-0.3764 -0.2537 -0.2007 -0.1470 -0.0627
```

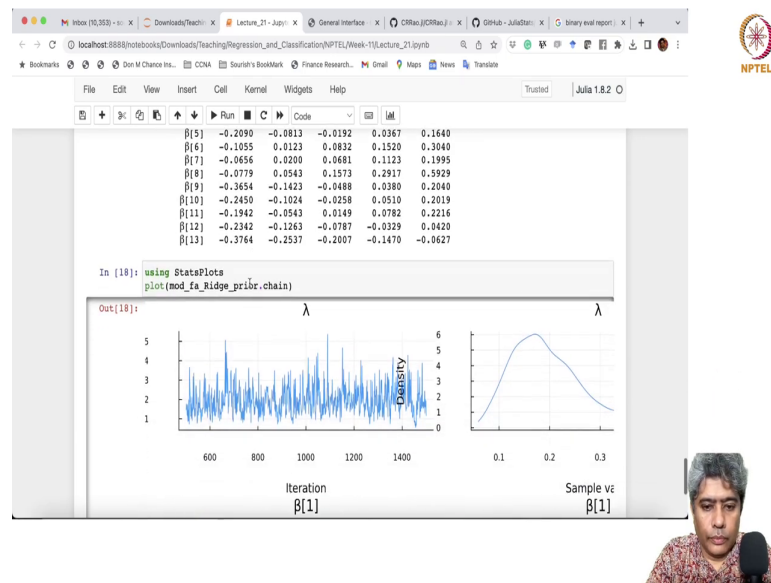
And if you see 13 entirely negative, but 12 is not negativity it includes the confidence interval if I see the 2.5 percentile point and 90. So, let me just explain you this Bayesian statistics a little bit. So, this is first the coefficient estimates and these are the standard deviations, if I have to compare it with the likelihood estimates, right.

So, these are the my coefficient estimates and these are the standard error. So, that is how you will see it here these are my coefficient estimates. So, beta 1 you will see including beta 1 it is there are 13 betas whereas; here I have actually 12 betas and alpha. So, total 13 betas I have. So, the first one is a intercept beta 1 this is intercept and then or here it is intercept and then you have 12 coefficient corresponds to each of the predictor.

Now, these are the some Monte Carlo standard error and if smaller Monte Carlo standard error indicates that the most likely it has converged nicely. And another statistics is called that

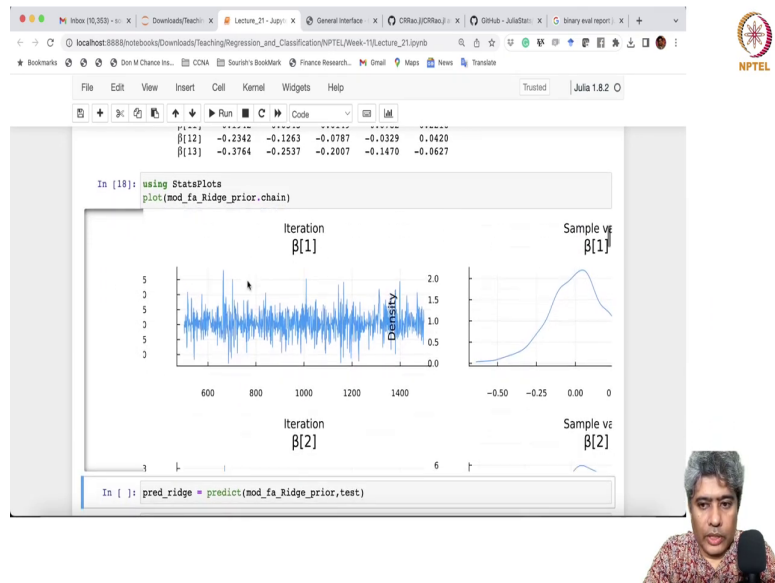
if that is close to one; that means, the mcmc convergence has taken place. So, Bayesian statistics in this case are being implemented using Markov chain Monte Carlo algorithm and not gradient descent algorithm.

(Refer Slide Time: 11:35)



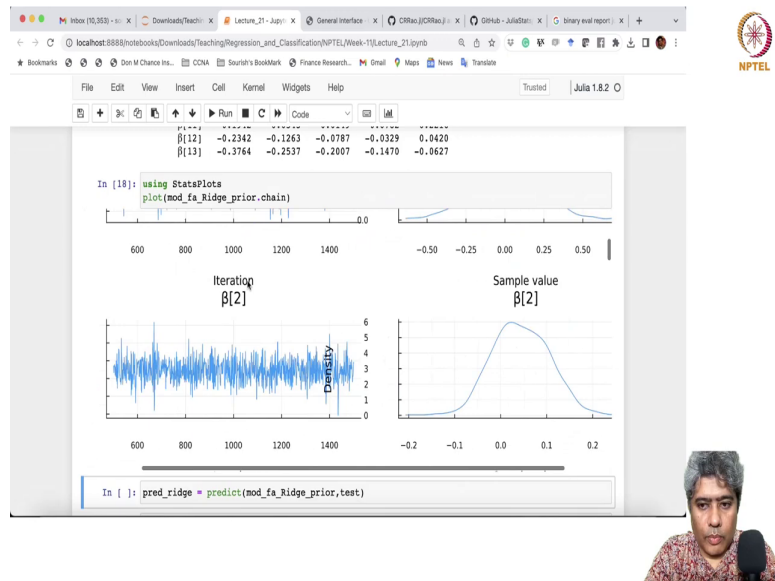
So, it is a Monte Carlo simulation based technique that they have increased and here for inference what they do they give a 95 percent confidence interval. And if you look into the 95 percent confidence that the last coefficient does have a effect sort of. And the third coefficient also does have a effect it is a positive and this is a negative coefficient and rest all of the coefficient includes 0 some way or other. So, there none of them have actually any effect on that.

(Refer Slide Time: 12:08)

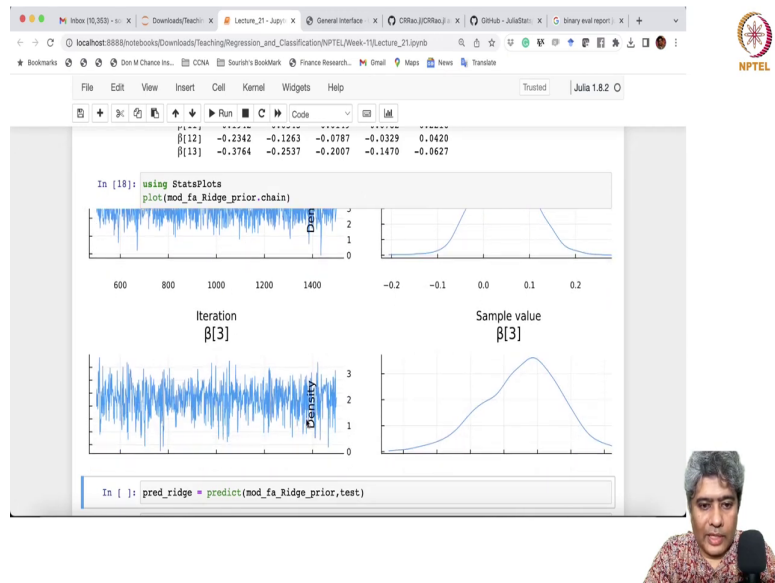




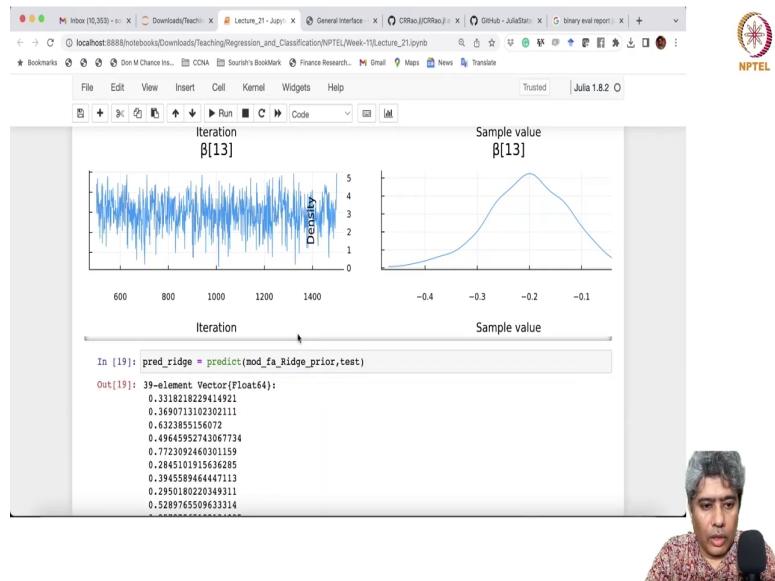
(Refer Slide Time: 12:10)



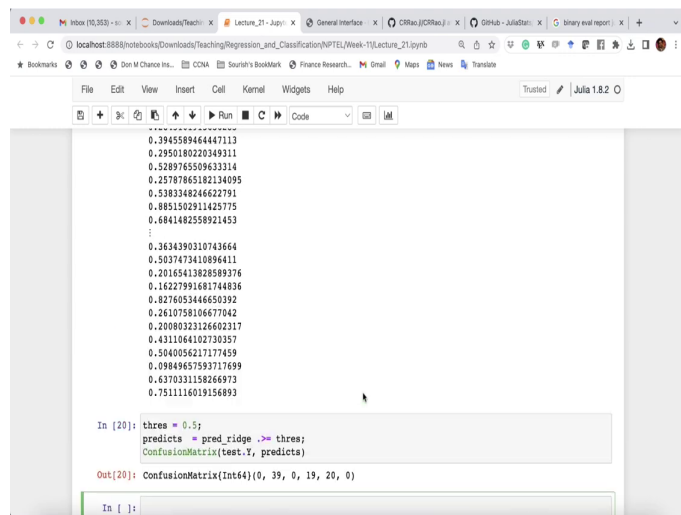
(Refer Slide Time: 12:13)



(Refer Slide Time: 12:24)



(Refer Slide Time: 12:29)





The screenshot shows a Julia REPL window with the following content:

```
0.3945589464447113
0.2950180220349311
0.5289765509633314
0.25787865182134095
0.5383348244622791
0.985150291425775
:
0.3634390310743664
0.5037473410896411
0.20165413828589376
0.16227991681744836
0.8276053446650392
0.2610758106677042
0.20080323126602317
0.6311064102739357
0.50405621177459
0.0984657593717699
0.637031158266973
0.751116019156893

In [20]: thres = 0.5;
         predicts = pred_ridge .>= thres;
         ConfusionMatrix(test.Y, predicts)

Out[20]: ConfusionMatrix{Int64}(0, 39, 0, 19, 20, 0)

In [ ]:
```



So, if I just do a plot simple stat plot. So, here is the plots you see they have decently converged nicely converged, ok. All of them pretty much decently converged here I am doing the predictive error. And here I have to do the confusion matrix I have to compute for some reason binary evaluation of the report is not working I will come back to you on the same.

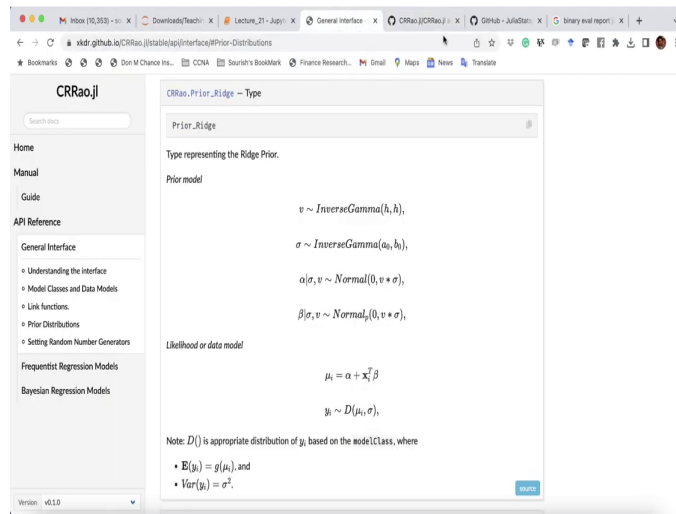
So, I will say take predictive ridge as that and then if I just use that. So, these are the confusion matrix that have we have found. So, I am not sure if it is correctly done. So, I will check with this.

(Refer Slide Time: 13:11)

```
129 ----
130 """
131 Prior_Ridge
132 """
133 Type representing the Ridge Prior.
134
135 @Prior_model
136 """math
137 v \sim \text{InversGamma}(h, s)
138 """
139 """math
140 \sigma \sim \text{InversGamma}(\beta, \beta)
141 """
142 """math
143 \alpha_j \sim \text{Normal}(0, v \sigma)
144 """
145 """math
146 \beta \sim \text{Normal}(\mu, v \sigma)
147 """
148 # Likelihood or data model
149
150 """math
151 y_{m,j} \sim \text{Normal}(x_j^T \beta)
152 """
153 """math
154 y_j \sim \text{Dir}(\mu_j, \sigma)
155 """
156 #Notes: "Dir" is appropriate distribution of "y_j" based on the "modelClass", where
157
158 # \text{Normal}(y_j, \mu_j) = \text{Normal}(y_j, \sigma)
159 # \text{Dir}(y_j, \mu_j) = \text{Dir}(y_j, \sigma)
160 """
161 struct Prior_Ridge end
162
163 """
```



(Refer Slide Time: 13:23)



The screenshot shows a web browser displaying the CRRao.jl website. The page title is "CRRao.jl - Type". The main content area is titled "Ridge Prior" and describes the model. The prior model is defined by the following distributions:

$$\begin{aligned}v &\sim \text{InverseGamma}(h, h), \\ \sigma &\sim \text{InverseGamma}(a_0, h), \\ \alpha | \sigma, v &\sim \text{Normal}(0, v * \sigma), \\ \beta | \sigma, v &\sim \text{Normal}_p(0, v * \sigma),\end{aligned}$$

The likelihood or data model is defined as:

$$\begin{aligned}\mu_i &= \alpha + \mathbf{x}_i^T \beta \\ y_i &\sim D(\mu_i, \sigma),\end{aligned}$$

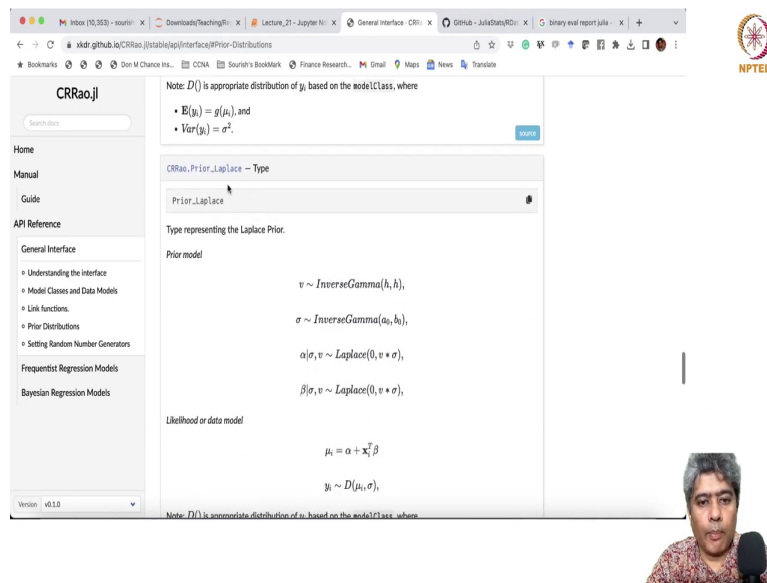
Note:  $D()$  is appropriate distribution of  $y_i$ , based on the model class, where

- $\mathbb{E}(y_i) = g(\mu_i)$ , and
- $\text{Var}(y_i) = \sigma^2$ .

The left sidebar contains navigation links: Home, Manual, Guide, API Reference, General Interface, Understanding the Interface, Model Classes and Data Models, Link functions, Prior Distributions, Setting Random Number Generators, Frequentist Regression Models, and Bayesian Regression Models. The version is v0.1.0.



(Refer Slide Time: 13:28)



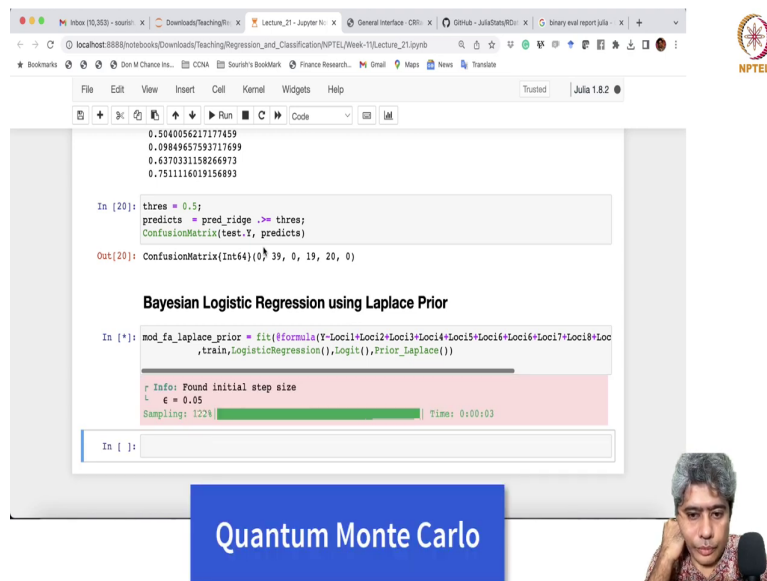
The screenshot shows the CRRao.jl API documentation for the `Prior_Laplace` type. The page includes a search bar, a navigation menu with sections like Home, Manual, Guide, and API Reference, and a main content area. The main content area displays the following information:

- Note:  $D()$  is appropriate distribution of  $y_i$ , based on the model Class, where
  - $E(y_i) = g(\mu_i)$ , and
  - $\text{Var}(y_i) = \sigma^2$ .
- CR Rao, Prior\_Laplace - Type
- Prior\_Laplace
- Type representing the Laplace Prior.
- Prior model
  - $v \sim \text{InverseGamma}(h, b)$ ,
  - $\sigma \sim \text{InverseGamma}(a_0, b_0)$ ,
  - $\alpha | \sigma, v \sim \text{Laplace}(0, v * \sigma)$ ,
  - $\beta | \sigma, v \sim \text{Laplace}(0, v * \sigma)$ ,
- Likelihood or data model
  - $\mu_i = \alpha + x_i^T \beta$
  - $y_i \sim D(\mu_i, \sigma)$ ,

At the bottom right of the screenshot, there is a small video feed of a man with grey hair, wearing a patterned shirt, speaking into a microphone.

Now, if you look back if you go to the say if I am now interested in implementing say Laplace prior how do I do implement.

(Refer Slide Time: 13:33)



```
0.504005621717459
0.09849657593717699
0.6370331156266973
0.7511116019156993

In [20]: thres = 0.5;
         predicts = pred_ridge .>= thres;
         ConfusionMatrix(test.Y, predicts)

Out[20]: ConfusionMatrix{Int64}(0, 39, 0, 19, 20, 0)

Bayesian Logistic Regression using Laplace Prior

In [ ]: mod_fa_laplace_prior = fit(@formula(Y~Loc1+Loc2+Loc3+Loc4+Loc5+Loc6+Loc7+Loc8+Loc
,train,logisticRegression(),logit(),Prior_laplace())

Info: Found initial step size
     g = 0.05
Sampling: 122% | Time: 0:00:03

In [ ]:
```

Quantum Monte Carlo

So, if I am interested in implementing say Bayesian Logistic Regression Bayesian Logistic Regression using Laplace prior, ok. So, if I have to do that all I have to do is just copy this guy from here and instead of I will just name it as Laplace and instead of prior ridge. I will just say Prior underscore Laplace.

So, Prior underscore Laplace ok and rest of the thing will be exactly same and now if you just run sorry, I think Laplace there was a spelling mistake yeah now I think it is done, yeah.



(Refer Slide Time: 14:58)

```
Out[22]: Formula: Y ~ 1 + Loc11 + Loc12 + Loc13 + Loc14 + Loc15 + Loc16 + Loc17 + Loc18 + Loc19 + Loc1
10 + Loc111 + Loc112
Link: Logit(CRao.Logit_Link)
Chain: Chains MCMC chain (1000*26*1 Array{Float64, 3}):

Iterations      = 501:1:1500
Number of chains = 1
Samples per chain = 1000
Wall duration    = 6.65seconds
Compute duration = 6.65 seconds
parameters      = λ, β[1], β[2], β[3], β[4], β[5], β[6], β[7], β[8], β[9], β[10], β[11], β[1
2], β[13]
internals       = lp, n_steps, is_accept, acceptance_rate, log_density, hamiltonian_energy,
hamiltonian_energy_error, max_hamiltonian_energy_error, tree_depth, numerical_error, step_size,
non_step_size

Summary Statistics
parameters      mean      std      mcmc      ess_bulk      ess_tail      rhat      -
Symbol          Float64  Float64  Float64  Float64      Float64      Float64
λ              0.1516   0.0595   0.0031   283.5977     254.1596     1.0040   --
β[1]           0.0161   0.2080   0.0071   1030.9581    342.7859     1.0000   --
β[2]           0.0315   0.0621   0.0025   654.4157     429.4716     1.0007   --
β[3]          -0.1279   0.1119   0.0055   528.1824     281.6956     1.0041   --
β[4]           0.4643   0.1522   0.0063   584.2328     531.1155     1.0015   --
β[5]          -0.0344   0.0894   0.0033   814.6973     555.9581     1.0018   --
β[6]           0.0530   0.0927   0.0042   528.2093     365.0896     1.0047   --
β[7]           0.0389   0.0694   0.0024   835.1357     716.3752     1.0060   --
```



(Refer Slide Time: 15:02)

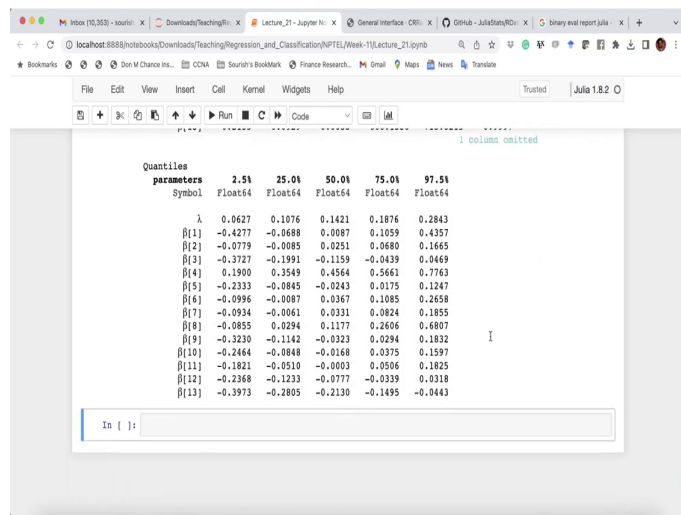
```
hamiltonian_energy_error, max_hamiltonian_energy_error, tree_depth, numerical_error, step_size, non_step_size

Summary Statistics
parameters      mean      std      mse      ess_bulk  ess_tail  rhat  -
Symbol Float64  Float64  Float64  Float64   Float64  Float64  -
λ              0.1516  0.0595  0.0031  283.5977  254.1596  1.0040  -
β[1]           0.0161  0.2080  0.0071  1030.9581  342.7859  1.0000  -
β[2]           0.0315  0.0621  0.0025  654.4137  429.4716  1.0007  -
β[3]          -0.1279  0.1119  0.0055  520.1824  281.6956  1.0041  -
β[4]           0.4643  0.1522  0.0063  584.2328  531.1155  1.0015  -
β[5]          -0.0344  0.0894  0.0033  814.6973  555.9581  1.0018  -
β[6]           0.0530  0.0927  0.0042  528.2093  365.0896  1.0047  -
β[7]           0.0389  0.0694  0.0024  835.1357  716.3752  1.0060  -
β[8]           0.1704  0.1999  0.0112  446.9434  515.5575  1.0038  -
β[9]          -0.0484  0.1309  0.0045  941.9574  659.1198  0.9991  -
β[10]         -0.0259  0.0986  0.0040  677.4022  639.0650  0.9994  -
β[11]         0.0008  0.0885  0.0033  703.7122  593.8553  0.9992  -
β[12]        -0.0832  0.0679  0.0027  651.6228  678.0120  1.0071  -
β[13]        -0.2155  0.0929  0.0033  806.1336  715.8215  0.9997  -
| column omitted

Quantiles
parameters      2.5%      25.0%      50.0%      75.0%      97.5%
Symbol Float64  Float64  Float64  Float64   Float64
λ              0.0627  0.1076  0.1421  0.1076  0.2843
β[1]         -0.4277  -0.0688  0.0087  0.1059  0.4357
```





(Refer Slide Time: 15:04)



The screenshot shows a Julia REPL window with the following output:

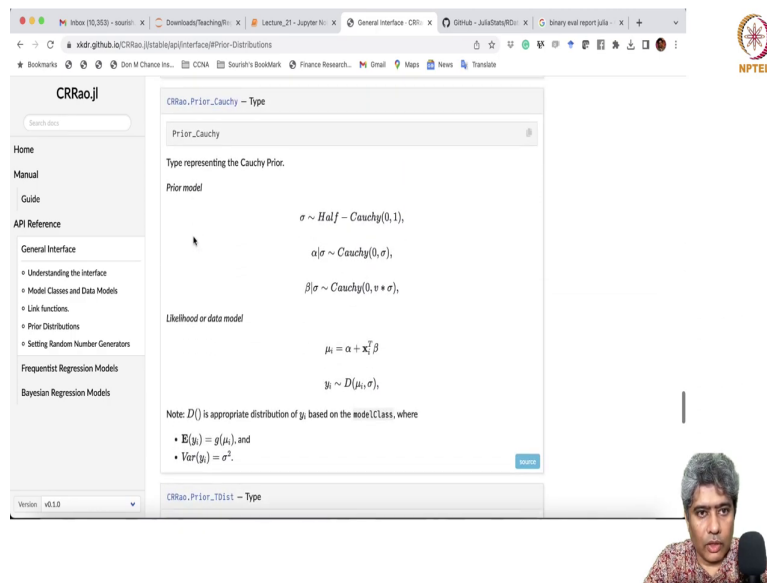
Quantiles	2.5%	25.0%	50.0%	75.0%	97.5%
parameters	Float64	Float64	Float64	Float64	Float64
$\lambda$	0.0627	0.1076	0.1421	0.1876	0.2843
$\beta[1]$	-0.4277	-0.0688	0.0087	0.1059	0.4357
$\beta[2]$	-0.0779	-0.0085	0.0251	0.0680	0.1665
$\beta[3]$	-0.3727	-0.1991	-0.1159	-0.0439	0.0469
$\beta[4]$	0.1900	0.3549	0.4564	0.5661	0.7763
$\beta[5]$	-0.2333	-0.0845	-0.0243	0.0175	0.1247
$\beta[6]$	-0.0996	-0.0087	0.0367	0.1085	0.2658
$\beta[7]$	-0.0934	-0.0061	0.0331	0.0824	0.1855
$\beta[8]$	-0.0855	0.0294	0.1177	0.2606	0.6807
$\beta[9]$	-0.3230	-0.1142	-0.0323	0.0294	0.1832
$\beta[10]$	-0.2464	-0.0848	-0.0168	0.0375	0.1597
$\beta[11]$	-0.1821	-0.0510	-0.0003	0.0506	0.1825
$\beta[12]$	-0.2368	-0.1233	-0.0777	-0.0339	0.0318
$\beta[13]$	-0.3973	-0.2805	-0.2130	-0.1495	-0.0443

The output is displayed in a REPL window with the prompt "In [ ]:" at the bottom.



So, ok looks like Laplace also has been implemented very fast and it is kind of 1500 simulation samples have been created from 1500 samples first 500 is being considered as a burning and from 501 to 1500, 1000 samples based on 1000 samples it is being created. And similar kind of inference you can see you can do based on the credible interval.

(Refer Slide Time: 15:40)



The screenshot shows a web browser displaying the documentation for the `Prior_Cauchy` type in the CRRao.jl package. The page is titled "CRRao.jl" and has a sidebar with navigation links: Home, Manual, Guide, API Reference, General Interface, Understanding the Interface, Model Classes and Data Models, Link functions, Prior Distributions, Setting Random Number Generators, Frequentist Regression Models, and Bayesian Regression Models. The main content area is titled "CRRao.Prior\_Cauchy - Type" and contains the following information:

`Prior_Cauchy`

Type representing the Cauchy Prior.

Prior model

$$\sigma \sim \text{Half} - \text{Cauchy}(0,1),$$
$$\alpha | \sigma \sim \text{Cauchy}(0, \sigma),$$
$$\beta | \sigma \sim \text{Cauchy}(0, \nu * \sigma),$$

Likelihood or data model

$$\mu_i = \alpha + \vec{x}_i^T \beta$$
$$y_i \sim D(\mu_i, \sigma),$$

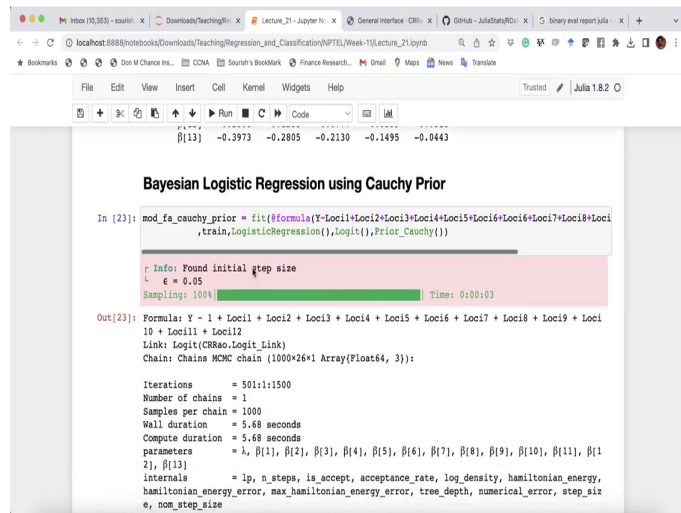
Note:  $D()$  is appropriate distribution of  $y_i$  based on the `model` class, where

- $E(y_i) = g(\mu_i)$ , and
- $\text{Var}(y_i) = \sigma^2$ .

NPTEL logo is visible in the top right corner of the browser window.

Now, if you interested in say doing Cauchy interval Cauchy prior. So, all you have to do is just take this information.

(Refer Slide Time: 15:51)



NPTEL

```
β[13] -0.3973 -0.2805 -0.2130 -0.1495 -0.0443
```


### Bayesian Logistic Regression using Cauchy Prior

```
In [23]: mod_fa_cauchy_prior = fit(formula(Y~Loc1+Loc2+Loc3+Loc4+Loc5+Loc6+Loc7+Loc8+Loc9+Loc10+Loc11+Loc12),train,LogisticRegression(),Logit(),Prior_Cauchy())
```

```
Info: Found initial step size
ε = 0.05
Sampling: 100% Time: 0:00:03
```

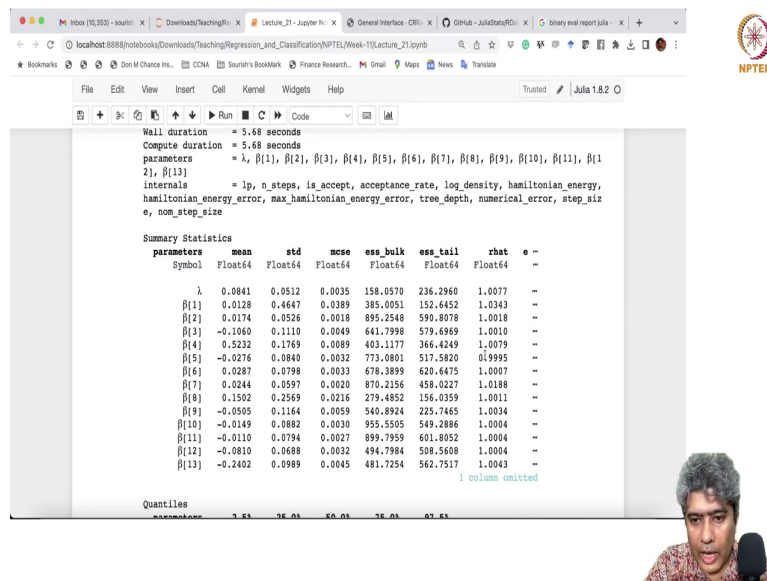
```
Out[23]: Formula: Y = 1 + Loc1 + Loc2 + Loc3 + Loc4 + Loc5 + Loc6 + Loc7 + Loc8 + Loc9 + Loc10 + Loc11 + Loc12
Link: Logit(CRao.Logit_Link)
Chain: Chains MCMC chain (1000*26*1 Array{Float64, 3}):

Iterations = 501:1:1500
Number of chains = 1
Samples per chain = 1000
Wall duration = 5.68 seconds
Compute duration = 5.68 seconds
parameters = λ, β[1], β[2], β[3], β[4], β[5], β[6], β[7], β[8], β[9], β[10], β[11], β[12], β[13]
Internals = lp, n_steps, is_accept, acceptance_rate, log_density, hamiltonian_energy, hamiltonian_energy_error, max_hamiltonian_energy_error, tree_depth, numerical_error, step_size, non_step_size
```



And now if I want to introduce say all I have to do just take this and instead of Laplace Prior I want Cauchy prior, ok C a u c h y Cauchy Prior, but not Laplace. So, I will just instead of Laplace I will say Cauchy Prior. So, I will just effectively if I just write the C and then tab will fill up the rest of the thing if I just write the C and then tab if you just press tab it will fill up the rest.

(Refer Slide Time: 16:56)



```
Wall duration = 5.68 seconds
Compute duration = 5.68 seconds
parameters = λ, β[1], β[2], β[3], β[4], β[5], β[6], β[7], β[8], β[9], β[10], β[11], β[12], β[13]
internals = lp, n_steps, is_accept, acceptance_rate, log_density, hamiltonian_energy, hamiltonian_energy_error, max_hamiltonian_energy_error, tree_depth, numerical_error, step_size, nom_step_size

Summary Statistics
parameters mean std mcmc ess_bulk ess_tail rhat e -
Symbol Float64 Float64 Float64 Float64 Float64 Float64
λ 0.0841 0.0512 0.0035 158.0570 236.2960 1.0077 -
β[1] 0.0128 0.4647 0.0389 385.0051 152.6452 1.0343 --
β[2] 0.0174 0.0526 0.0018 895.2548 590.8078 1.0018 --
β[3] -0.1060 0.1110 0.0049 641.7998 579.6969 1.0010 --
β[4] 0.5232 0.1769 0.0089 403.1177 366.4249 1.0079 --
β[5] -0.0276 0.0840 0.0032 773.0801 517.5820 0.9995 --
β[6] 0.0287 0.0798 0.0033 678.3899 620.6475 1.0007 --
β[7] 0.0244 0.0597 0.0020 870.2156 458.0227 1.0188 --
β[8] 0.1502 0.2569 0.0216 279.4852 156.0359 1.0011 --
β[9] -0.0505 0.1164 0.0059 540.8924 225.7465 1.0034 --
β[10] -0.0149 0.0882 0.0030 955.5505 549.2886 1.0004 --
β[11] -0.0110 0.0794 0.0027 899.7959 601.8052 1.0004 --
β[12] -0.0810 0.0688 0.0032 494.7984 508.5608 1.0004 --
β[13] -0.2402 0.0989 0.0045 481.7254 562.7517 1.0043 --
1 column omitted

Quantiles
parameters 2.5% 25.0% 50.0% 75.0% 97.5%
```

So, instead of Laplace I want to name it as a say Cauchy prior. So, for some reason the evaluation methods are not working properly I will correct those and I will share the correct Notebook with you guys. For some reason it is not working, but as you see here also pretty much all rhats are first thing you should check in the Bayesian statistics with all rhats are very close to one or not if it is indeed, one close to one then you are in good shape.

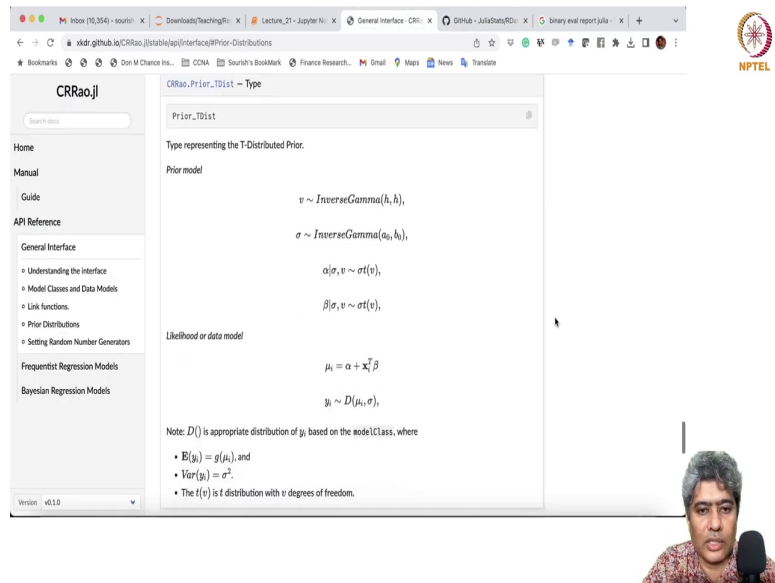
(Refer Slide Time: 17:09)

Quantiles

parameters	2.5%	25.0%	50.0%	75.0%	97.5%
Symbol	Float64	Float64	Float64	Float64	Float64
$\lambda$	0.0197	0.0481	0.0742	0.1073	0.2098
$\beta[1]$	-0.7013	-0.0633	0.0061	0.0752	1.0800
$\beta[2]$	-0.0755	-0.0141	0.0113	0.0463	0.1345
$\beta[3]$	-0.3609	-0.1702	-0.0866	-0.0241	0.0560
$\beta[4]$	0.1719	0.4010	0.5229	0.6406	0.8702
$\beta[5]$	-0.2276	-0.0709	-0.0183	0.0195	0.1305
$\beta[6]$	0.1093	-0.0183	0.0163	0.0662	0.2205
$\beta[7]$	-0.0926	-0.0135	0.0176	0.0617	0.1513
$\beta[8]$	-0.0928	0.0052	0.0660	0.1920	0.8906
$\beta[9]$	-0.3629	-0.0925	-0.0259	0.0168	0.1364
$\beta[10]$	-0.2055	-0.0533	-0.0092	0.0296	0.1616
$\beta[11]$	-0.1603	-0.0550	-0.0096	0.0296	0.1597
$\beta[12]$	-0.2275	-0.1255	-0.0730	-0.0293	0.0343
$\beta[13]$	-0.4377	-0.3087	-0.2359	-0.1687	-0.0607

And here also you can see the third predictor is all positive and the last predictor is all negative rest of the all predictors are essentially either or including 0 then 95 percent confidence interval is including 0. So, similar consistent inference that we are getting from all of the Cauchy Priors.

(Refer Slide Time: 17:46)



The screenshot shows a web browser displaying the CRRao.jl documentation page for the `Prior_TDist` type. The page is titled "CRRao.jl" and "Prior\_TDist - Type". The main content area contains the following text:

Type representing the T-Distributed Prior.

Prior model

$$v \sim \text{InverseGamma}(h, h)$$
$$\sigma \sim \text{InverseGamma}(a_0, b_0)$$
$$\alpha | \sigma, v \sim \sigma t(v)$$
$$\beta | \sigma, v \sim \sigma t(v)$$

Likelihood or data model

$$\mu_i = \alpha + \mathbf{x}_i^T \beta$$
$$y_i \sim D(\mu_i, \sigma)$$

Note:  $D()$  is appropriate distribution of  $y_i$  based on the model class, where

- $\mathbb{E}(y_i) = g(\mu_i)$ , and
- $\text{Var}(y_i) = \sigma^2$ .
- The  $t(v)$  is  $t$  distribution with  $v$  degrees of freedom.

On the right side of the browser window, there is a small video feed of a man with grey hair, wearing a patterned shirt, speaking into a microphone. In the top right corner of the browser window, there is a logo for NPTEL (National Programme on Technology Enhanced Learning).



(Refer Slide Time: 17:48)

CRRao.jl

- $E(\mu_i) = g(\mu_i)$ , and
- $\text{Var}(\mu_i) = \sigma^2$ .
- The  $f(\tau)$  is  $f$  distribution with  $\nu$  degrees of freedom.

CRRao.Prior\_horseShoe — Type



Prior\_horseShoe

Type representing the HorseShoe Prior.

Prior model

$$\tau \sim \text{HalfCauchy}(0, 1),$$
$$\lambda_j \sim \text{HalfCauchy}(0, 1), j = 1, 2, \dots, p$$
$$\sigma \sim \text{HalfCauchy}(0, 1),$$
$$\alpha, \tau \sim N(0, \tau * \sigma),$$
$$\beta_j | \sigma, \lambda_j, \tau \sim \text{Normal}(0, \lambda_j * \tau * \sigma),$$

Likelihood or data model

$$\mu_i = \alpha + \mathbf{x}_i^T \beta$$
$$w_i \sim \text{Diagonal}(\sigma), i = 1, 2, \dots, n$$


Now, let us see what else are we do we have other than Cauchy Prior we have also T distributed prior and then HorseShoe Prior HorseShoe Prior is very popular. Let us try to implement the HorseShoe Prior, ok. So, let us try to implement the HorseShoe Prior.

(Refer Slide Time: 18:02)

**Bayesian Logistic Regression using Cauchy Prior**

```
In [24]: mod_fa_bs_prior = fit({formula(Y~Loci1+Loci2+Loci3+Loci4+Loci5+Loci6+Loci7+Loci8+Loci9+Loci10+Loci11+Loci12),train,LogisticRegression(),Logit(),Prior_Horseshoe()})
```

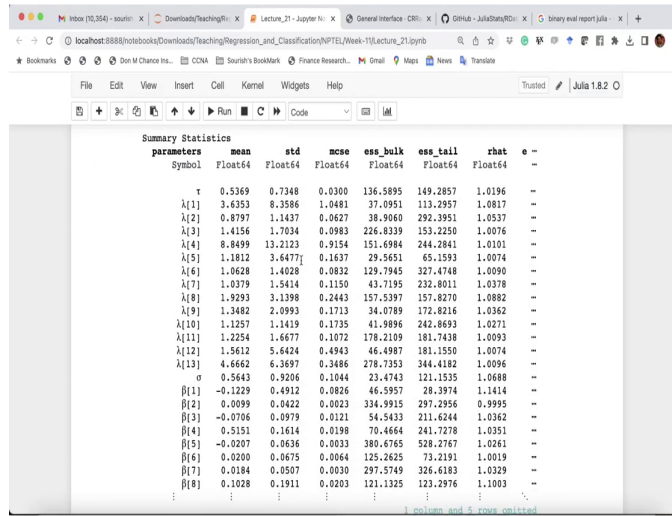
Info: Found initial step size  
 $\epsilon = 0.025$   
Sampling: 100% Time: 0:00:16

```
Out[24]: Formula: Y ~ 1 + Loci1 + Loci2 + Loci3 + Loci4 + Loci5 + Loci6 + Loci7 + Loci8 + Loci9 + Loci10 + Loci11 + Loci12  
Link: Logit(CRan.Logit_Link)  
Chain: Chains MCMC chain (1000+40*1 Array{Float64, 3}):  
  
Iterations = 501:1:1500  
Number of chains = 1  
Samples per chain = 1000  
Wall duration = 22.26 seconds  
Compute duration = 22.26 seconds  
parameters =  $\tau, \lambda\{1\}, \lambda\{2\}, \lambda\{3\}, \lambda\{4\}, \lambda\{5\}, \lambda\{6\}, \lambda\{7\}, \lambda\{8\}, \lambda\{9\}, \lambda\{10\}, \lambda\{11\}, \lambda\{12\}, \lambda\{13\}, \phi, \beta\{1\}, \beta\{2\}, \beta\{3\}, \beta\{4\}, \beta\{5\}, \beta\{6\}, \beta\{7\}, \beta\{8\}, \beta\{9\}, \beta\{10\}, \beta\{11\}, \beta\{12\}, \beta\{13\}$   
internals = lp, n_steps, is_accept, acceptance_rate, log_density, hamiltonian_energy, hamiltonian_energy_error, max_hamiltonian_energy_error, tree_depth, numerical_error, step_size, e, non_step_size
```

Summary Statistics						
parameters	mean	std	mcmc	ess_bulk	ess_tail	rhat
Symbol	Float64	Float64	Float64	Float64	Float64	Float64
						-

So, let me just do that and here is the. So, all I have to do just go copy this model and now instead of in the prior area just instead of Cauchy Prior just say Horseshoe. So, I will just say H and then tab it will fill up the rest and I am pretty much done. Of course, I do I want to change the name to Horseshoe Prior and run. Horseshoe Prior may require a little bit of more run maybe thousand simulation is not enough because it has lot of parameters in it.

(Refer Slide Time: 19:01)



Summary Statistics

parameters	mean	std	mse	ess_bulk	ess_tail	rhat	e
Symbol	Float64	Float64	Float64	Float64	Float64	Float64	Float64
$\tau$	0.5369	0.7348	0.0300	136.5895	149.2857	1.0196	--
$\lambda[1]$	3.6353	8.3586	1.0481	37.0951	113.2957	1.0817	--
$\lambda[2]$	0.8797	1.1437	0.0627	38.9060	292.5951	1.0537	--
$\lambda[3]$	1.4156	1.7034	0.0983	226.8339	153.2250	1.0076	--
$\lambda[4]$	8.8499	13.2123	0.9154	151.6984	244.2841	1.0101	--
$\lambda[5]$	1.1812	3.6477	0.1637	29.5651	65.1593	1.0074	--
$\lambda[6]$	1.0628	1.4028	0.0832	129.7945	327.4748	1.0090	--
$\lambda[7]$	1.0379	1.5414	0.1150	43.7195	232.8011	1.0378	--
$\lambda[8]$	1.9283	3.1398	0.2443	157.5397	157.8270	1.0882	--
$\lambda[9]$	1.3482	2.0993	0.1713	34.0789	172.8216	1.0362	--
$\lambda[10]$	1.1257	1.1419	0.1735	41.9896	242.8693	1.0271	--
$\lambda[11]$	1.2254	1.6677	0.1072	178.2109	181.7438	1.0093	--
$\lambda[12]$	1.5612	5.6424	0.4943	46.4987	181.1550	1.0074	--
$\lambda[13]$	4.6462	6.1697	0.2485	278.7353	344.4182	1.0096	--
$\alpha$	0.5643	0.9206	0.1044	23.4743	121.1535	1.0688	--
$\beta[1]$	-0.1229	0.4912	0.0826	46.5957	28.3974	1.1414	--
$\beta[2]$	0.0099	0.0422	0.0023	334.5955	297.2956	0.9995	--
$\beta[3]$	-0.0706	0.0979	0.0121	54.5433	211.6244	1.0362	--
$\beta[4]$	0.5151	0.1614	0.0198	70.4664	241.7278	1.0351	--
$\beta[5]$	-0.0207	0.0636	0.0033	380.6765	529.2767	1.0261	--
$\beta[6]$	0.0200	0.0675	0.0064	125.2625	73.2191	1.0019	--
$\beta[7]$	0.0184	0.0507	0.0030	297.5749	326.6183	1.0329	--
$\beta[8]$	0.1028	0.1911	0.0203	121.1325	123.2976	1.1003	--



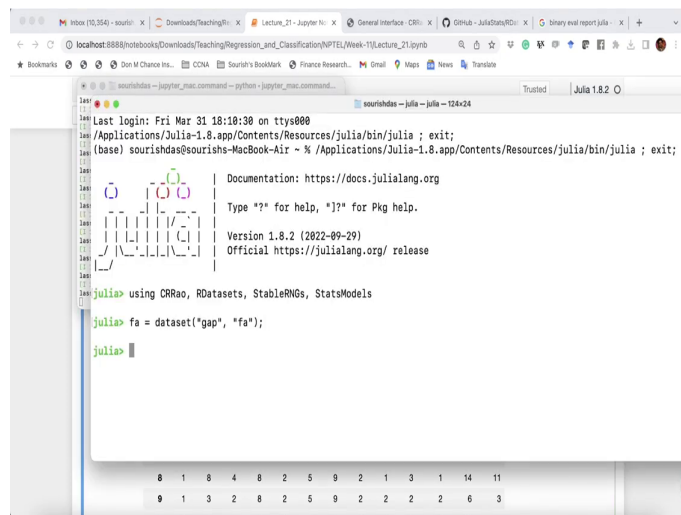
(Refer Slide Time: 19:07)

The screenshot shows a Jupyter Notebook interface with a table of parameter estimates. The table is titled "Quantiles" and has columns for "parameters", "Symbol", and five quantiles: "2.5%", "25.0%", "50.0%", "75.0%", and "97.5%". The parameters listed are  $\tau$ ,  $\lambda[1]$  through  $\lambda[13]$ ,  $\sigma$ ,  $\beta[1]$  through  $\beta[8]$ . The values are displayed in a grid format. A small video inset of a man is visible in the bottom right corner of the notebook window.

parameters	Symbol	2.5%	25.0%	50.0%	75.0%	97.5%
$\tau$		0.0245	0.1465	0.3502	0.7190	2.2066
$\lambda[1]$		0.0779	0.4869	0.9225	3.0995	23.2555
$\lambda[2]$		0.0445	0.2521	0.5525	1.0189	4.2473
$\lambda[3]$		0.0781	0.5107	0.9583	1.6893	6.6345
$\lambda[4]$		1.1778	3.1843	5.5341	8.5564	49.4256
$\lambda[5]$		0.0357	0.2156	0.5585	1.1988	4.7989
$\lambda[6]$		0.0700	0.3041	0.7504	1.2256	5.0372
$\lambda[7]$		0.0513	0.1992	0.5680	1.2142	4.6857
$\lambda[8]$		0.0758	0.4936	0.9303	1.9144	12.0225
$\lambda[9]$		0.0721	0.2640	0.6878	1.5342	7.3106
$\lambda[10]$		0.0659	0.3219	0.7310	1.5736	4.3176
$\lambda[11]$		0.0710	0.3638	0.6324	1.4518	6.1502
$\lambda[12]$		0.1048	0.2703	0.7019	1.3718	5.6731
$\lambda[13]$		0.5856	1.7784	3.0018	4.6037	21.0425
$\sigma$		0.0378	0.0960	0.2845	0.6266	3.0904
$\beta[1]$		-1.5316	-0.0973	-0.0036	0.0247	0.8135
$\beta[2]$		-0.0793	-0.0081	0.0062	0.0234	0.1205
$\beta[3]$		-0.3146	-0.1313	-0.0334	-0.0001	0.0477
$\beta[4]$		0.2382	0.3810	0.4973	0.6316	0.8434
$\beta[5]$		-0.1738	-0.0467	-0.0035	0.0086	0.0967
$\beta[6]$		-0.0929	-0.0150	0.0099	0.0367	0.2011
$\beta[7]$		-0.0812	-0.0040	0.0061	0.0446	0.1434
$\beta[8]$		-0.0634	0.0009	0.0272	0.1268	0.7379

So, we will see interesting, but we are seeing that lot of most of the parameters have been converged. So, ok, but unfortunately it is not printing all the parameters ok, that is in Jupyter Notebook that sometimes a problem.

(Refer Slide Time: 19:32)



The screenshot shows a terminal window with the following content:

```
Last login: Fri Mar 31 18:18:38 on ttys000
/Applications/Julia-1.8.app/Contents/Resources/julia/bin/julia ; exit;
(base) sourishdas@sourishdas-MacBook-Air ~ % /Applications/Julia-1.8.app/Contents/Resources/julia/bin/julia ; exit;


      _   _
     / \ / \
    / _ \|_| | |_/ \
   / ___ \|_| | |_/ \
  / ___ \|_| | |_/ \
 / ___ \|_| | |_/ \
/ ___ \|_| | |_/ \
\___/ \|_| | |_/ \

Documentation: https://docs.julialang.org
Type "?" for help, "]" for Pkg help.
Version 1.8.2 (2022-09-29)
Official https://julialang.org/ release

julia> using CR Rao, R Datasets, StableRNGs, StatsModels

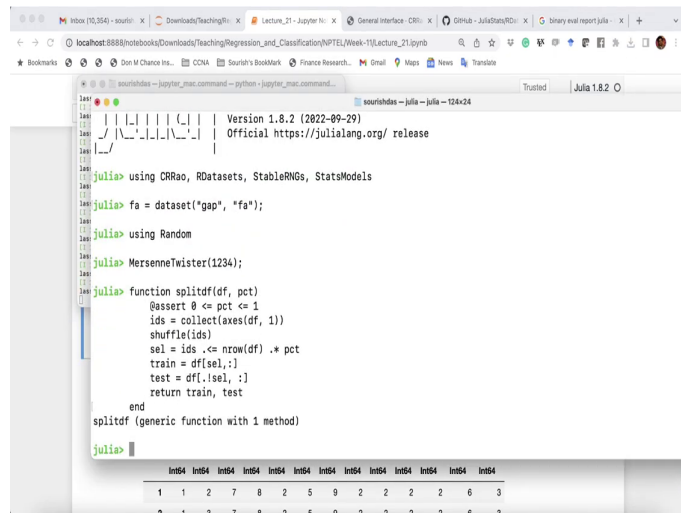
julia> fa = dataset("gap", "fa");

julia>
```



So, what I will do I will I will run it in the maybe code Jupyter Notebook co code Julia. So, I will open code Julia so, that we can see the results, ok. So, I will just call the you know let me just run this few lines there, ok. And then if I just I will just call this guy, ok.

(Refer Slide Time: 20:08)



The screenshot shows a Julia REPL window with the following code and output:

```
julia> using CR Rao, RDatasets, StableRNGs, StatsModels
julia> fa = dataset("gap", "fa");
julia> using Random
julia> MersenneTwister(1234);
julia> function splitdf(df, pct)
    @assert 0 <= pct <= 1
    ids = collect(axes(df, 1))
    shuffle!(ids)
    sel = ids <= nrow(df) .* pct
    train = df[sel,:]
    test = df[!sel,:]
    return train, test
end
splitdf (generic function with 1 method)
julia>
```

Below the REPL window, there is a small video inset showing a man speaking into a microphone. At the bottom of the screen, there is a table of data:

	Int64	Int64	Int64	Int64	Int64	Int64	Int64	Int64	Int64	Int64	Int64	
1	1	2	7	8	2	5	9	2	2	2	6	3
2	1	3	7	8	2	5	9	2	2	2	6	3



(Refer Slide Time: 20:23)

```
train = df[sel,:]  
test = df[.!sel,:]  
return train, test  
end  
splitdf (generic function with 1 method)  
julia> train, test = splitdf(fa, 0.7);  
julia> first(train, 10)  
10x13 DataFrame  
 Row Y Loci1 Loci2 Loci3 Loci4 Loci5 Loci6 Loci7 Loci8 Loci9 Loci10 Loci11 Loci12  
 Int64 Int64 Int64 Int64 Int64 Int64 Int64 Int64 Int64 Int64 Int64 Int64 Int64  
 1 1 2 7 8 2 5 9 2 2 2 2 6 3  
 2 1 3 7 8 2 5 9 2 2 2 2 6 3  
 3 1 3 1 8 2 5 9 2 2 2 2 14 5  
 4 1 2 5 7 2 5 9 2 2 2 2 14 3  
 5 1 14 7 8 5 6 2 3 2 2 2 14 9  
 6 1 8 4 7 7 3 2 3 4 2 2 14 9  
 7 1 8 4 8 2 5 9 2 1 3 1 14 11  
 8 1 8 4 8 2 5 9 2 1 3 1 14 11  
 9 1 3 2 8 2 5 9 2 2 2 2 6 3  
 10 1 8 4 8 2 5 9 2 2 2 2 10 6  
  
julia> mod_fa_HS_prior = fit(@formula(Y~Loci1+Loci2+Loci3+Loci4+Loci5+Loci6+Loci6+Loci7+Loci8+Loci9+Loci10+Loci11+  
 ,train, LogisticRegression(), Logit(), Prior_HorseShoe())  
 Info: Found initial step size  
 ε = 0.003125  
 Sampling 64% | ETA: 0:09:30
```

And I will call this split function train and test and then, ok then finally, I will just use the run the last data set last model, ok. Let us see how much time it takes yeah, its relatively faster actually, ok.

(Refer Slide Time: 21:11)

```
3 | 1 3 1 8 2 5 9 2 2 2 2 14 5
4 | 1 2 5 7 2 5 9 2 2 2 2 14 3
5 | 1 14 7 8 5 6 2 3 2 2 2 14 9
6 | 1 8 4 7 7 3 2 3 4 2 2 14 9
7 | 1 8 4 8 2 5 9 2 1 3 1 14 11
8 | 1 8 4 8 2 5 9 2 1 3 1 14 11
9 | 1 3 2 8 2 5 9 2 2 2 2 6 3
10 | 1 8 4 8 2 5 9 2 2 2 2 10 6

julia> mod_fa_HS_prior = fit(@formula(Y=Loci1+Loci2+Loci3+Loci4+Loci5+Loci6+Loci7+Loci8+Loci9+Loci10+Loci11+Loci12)
    ,train,LogisticRegression(),Logit(),Prior_HorseShoe())
[ Info: Found initial step size
    ε = 0.003125
Sampling 100% | Time: 0:00:29
Formula: Y ~ 1 + Loci1 + Loci2 + Loci3 + Loci4 + Loci5 + Loci6 + Loci7 + Loci8 + Loci9 + Loci10 + Loci11 + Loci12
Link: Logit(CRRao.Logit_Link)
Chain: Chains MCMC chain (1000×40×1 Array{Float64, 3}):

Iterations = 501:1:1500
Number of chains = 1
Samples per chain = 1000
Wall duration = 29.76 seconds
Compute duration = 29.76 seconds
parameters = τ, λ[1], λ[2], λ[3], λ[4], λ[5], λ[6], λ[7], λ[8], λ[9], λ[10], λ[11], λ[12], λ[13], σ, β[1], β
, β[4], β[5], β[6], β[7], β[8], β[9], β[10], β[11], β[12], β[13]
internals = lp, n_steps, is_accept, acceptance_rate, log_density, hamiltonian_energy, hamiltonian_energy_er
hamiltonian_energy_error, tree_depth, numerical_error, step_size, non_step_size

Summary Statistics
```





(Refer Slide Time: 21:14)

Quantiles parameters	2.5%	25.0%	50.0%	75.0%	97.5%
Symbol	Float64	Float64	Float64	Float64	Float64
$\beta[13]$	-0.2512	0.1022	0.0099	112.2952	138.6843
$\tau$	0.0448	0.1858	0.3741	0.6854	2.4824
$\lambda[1]$	0.0686	0.3783	0.9776	2.0487	10.3659
$\lambda[2]$	0.0168	0.1999	0.4936	1.0608	3.6866
$\lambda[3]$	0.0896	0.4442	1.0361	2.3358	8.8854
$\lambda[4]$	1.1317	3.0449	5.3431	8.7824	30.4613
$\lambda[5]$	0.0435	0.3840	0.7849	1.6689	4.3695
$\lambda[6]$	0.0561	0.2818	0.6471	1.2948	5.7463
$\lambda[7]$	0.0498	0.2066	0.5525	1.1298	4.7579
$\lambda[8]$	0.0926	0.5026	1.0171	2.0604	9.4914
$\lambda[9]$	0.0388	0.3298	0.6857	1.4236	5.7221
$\lambda[10]$	0.0679	0.3682	0.8375	1.4783	5.9469
$\lambda[11]$	0.0398	0.2181	0.5745	1.1829	4.3867
$\lambda[12]$	0.1013	0.4366	0.9015	1.4378	4.7194
$\lambda[13]$	0.4595	1.4928	2.5181	4.4382	14.2899
$\sigma$	0.0260	0.1057	0.2357	0.5544	1.9331
$\beta[1]$	-0.4893	-0.0390	0.0015	0.0676	0.4693
$\beta[2]$	-0.0772	-0.0107	0.0035	0.0331	0.1224
$\beta[3]$	-0.3291	-0.1268	-0.0345	0.0039	0.0557
$\beta[4]$	0.2264	0.4056	0.5106	0.6402	0.8183
$\beta[5]$	-0.2086	-0.0865	-0.0183	0.0082	0.0896
$\beta[6]$	-0.1097	-0.0274	0.0010	0.0392	0.1788
$\beta[7]$	-0.0656	-0.0097	0.0061	0.0421	0.1362
$\beta[8]$	-0.0793	0.0026	0.0336	0.1589	0.7289
$\beta[9]$	-0.2728	-0.0610	-0.0099	0.0074	0.1029
$\beta[10]$	-0.2158	-0.0573	-0.0038	0.0188	0.1154
$\beta[11]$	-0.1492	-0.0282	-0.0034	0.0154	0.1203
$\beta[12]$	-0.2074	-0.1010	-0.0451	-0.0140	0.0304
$\beta[13]$	-0.4347	-0.3244	-0.2532	-0.1836	-0.0293



(Refer Slide Time: 21:38)

Summary Statistics							
parameters	mean	std	mcse	ess_bulk	ess_tail	rhat	ess_per_sec
Symbol	Float64	Float64	Float64	Float64	Float64	Float64	Float64
$\tau$	0.6057	1.1656	0.0485	213.7638	279.2188	1.0098	7.1817
$\lambda[1]$	2.6673	28.9486	0.6698	65.2821	24.4273	1.0147	2.1933
$\lambda[2]$	0.8842	1.4997	0.0725	98.9558	19.2813	1.0183	1.7119
$\lambda[3]$	2.1101	2.7614	0.4457	59.7784	61.1237	1.0034	2.0083
$\lambda[4]$	7.8439	18.9166	0.6629	232.1859	381.8317	1.0171	7.9086
$\lambda[5]$	1.2158	1.4758	0.0764	171.6405	381.7291	1.0138	5.7645
$\lambda[6]$	2.0184	14.1987	0.8775	127.5979	168.4914	1.0218	4.2868
$\lambda[7]$	0.9578	1.4082	0.0899	141.1977	377.9457	1.0083	4.7437
$\lambda[8]$	2.1867	7.7913	0.3888	287.1347	364.7326	1.0046	6.9598
$\lambda[9]$	1.3876	2.2644	0.1557	146.8121	158.6438	1.0112	4.9324
$\lambda[10]$	1.3518	1.8582	0.0956	264.7104	354.4881	0.9996	8.8933
$\lambda[11]$	0.9866	1.3688	0.0985	146.8681	348.2527	1.0108	4.9074
$\lambda[12]$	1.2857	1.3825	0.0592	362.3571	441.7875	1.0043	12.1739
$\lambda[13]$	3.8526	4.7133	0.2952	147.1958	271.1536	1.0148	4.9452
$\sigma$	0.4555	0.5912	0.0358	183.8816	314.3923	1.0042	6.1482
$\beta[1]$	0.0082	0.2333	0.0125	486.4616	274.4591	0.9992	13.6624
$\beta[2]$	0.0124	0.0478	0.0038	298.8766	223.0786	0.9997	18.0412
$\beta[3]$	-0.0718	0.1056	0.0068	218.9314	299.9385	1.0085	7.8866
$\beta[4]$	0.5221	0.1637	0.0184	82.3646	321.5278	1.0108	2.7672
$\beta[5]$	-0.0396	0.0766	0.0181	78.1478	293.5113	1.0217	2.3567
$\beta[6]$	0.0069	0.0745	0.0076	94.3588	91.0872	1.0725	3.1698
$\beta[7]$	0.0169	0.0475	0.0031	218.9477	495.1743	1.0016	7.8871
$\beta[8]$	0.1148	0.1987	0.0288	198.2772	95.3975	1.0136	6.3926
$\beta[9]$	-0.0365	0.0989	0.0065	337.8955	285.8968	0.9998	11.3521
$\beta[10]$	-0.0215	0.0886	0.0051	263.1513	258.6645	1.0072	8.8418
$\beta[11]$	-0.0086	0.0621	0.0048	326.2493	131.9959	1.0019	18.9688
$\beta[12]$	-0.0618	0.0647	0.0048	198.6986	378.8219	1.0018	6.6753
$\beta[13]$	-0.2512	0.1022	0.0099	112.2952	138.6843	1.0108	3.7727

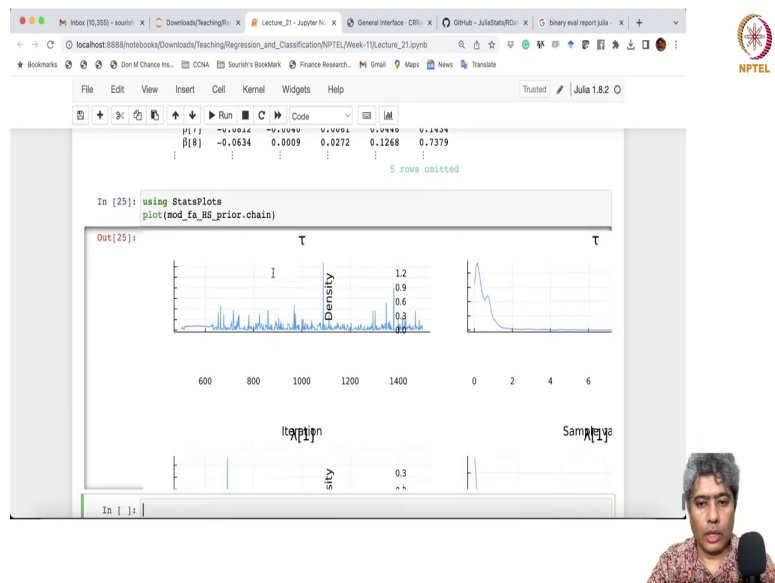
Quantiles					
parameters	2.5%	7.5%	92.5%	97.5%	99.75%



So, alright so, if I just run right this yeah now, I can see that all the parameters here are being printed nicely you know rhat is all the rhat are very close to one means convergence did have taken place that is a very good news for us. And then for each parameters if we will see how this parameter for each coefficient there will be a scale parameter in the HorseShoe Prior.

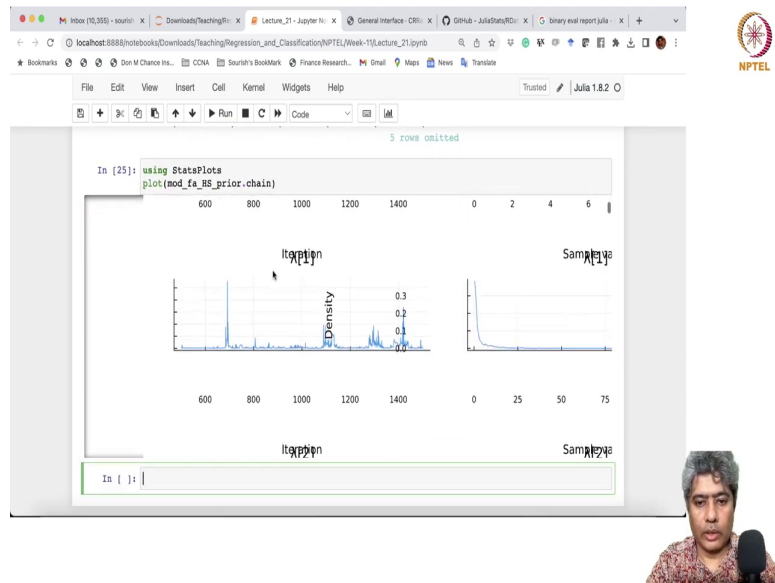
So, each scale parameter perform confidence interval are also being printed. And now what you can see that the third one as expected is also very strongly positive and the last one has a very strongly negative whereas, these ones are not you know very strongly positive or negative.

(Refer Slide Time: 22:41)

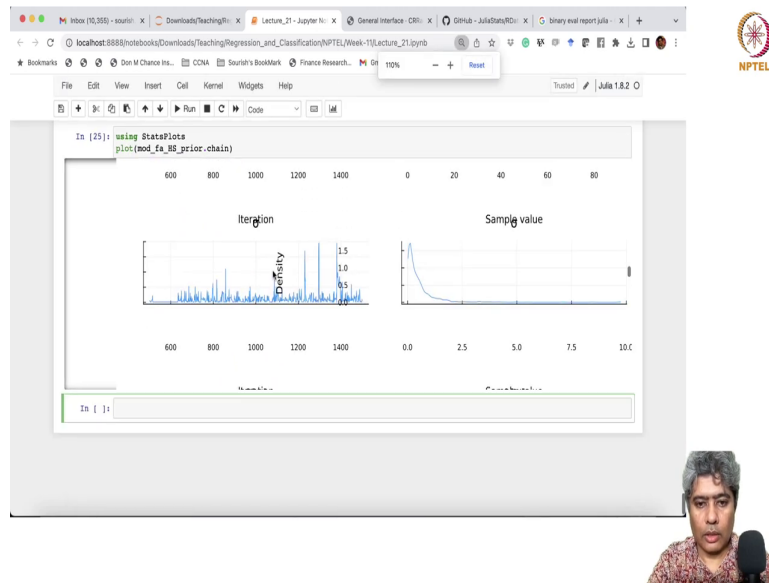


So, if I go back and if I just do the plot let me do a plotting and you will see typically Horseshoe Prior behaves a very interesting way. So, let me just try to do the plotting. So, (Refer Time: 22:46) ok let me just see if I can create this plot here Horseshoe prior chain.

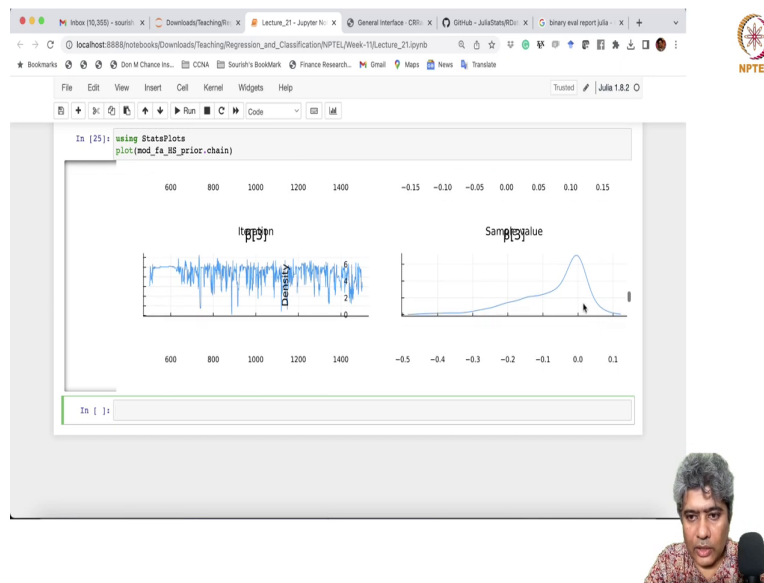
(Refer Slide Time: 23:15)



(Refer Slide Time: 23:25)

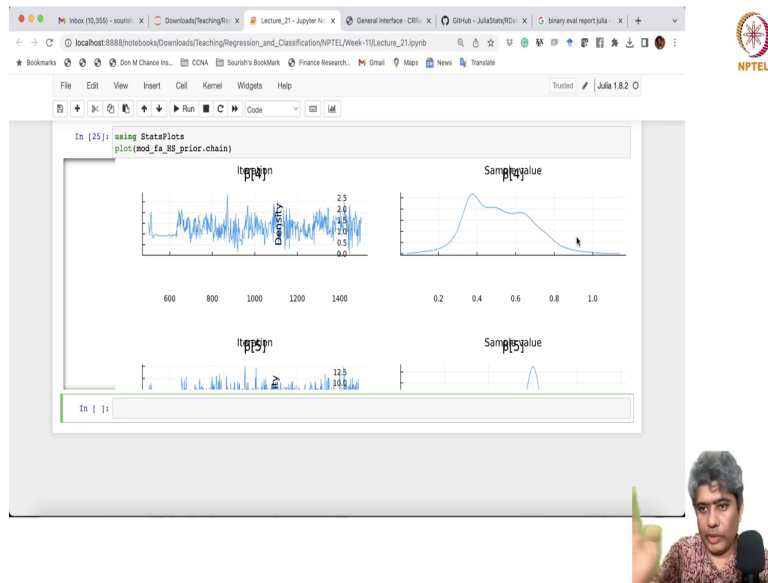


(Refer Slide Time: 23:45)

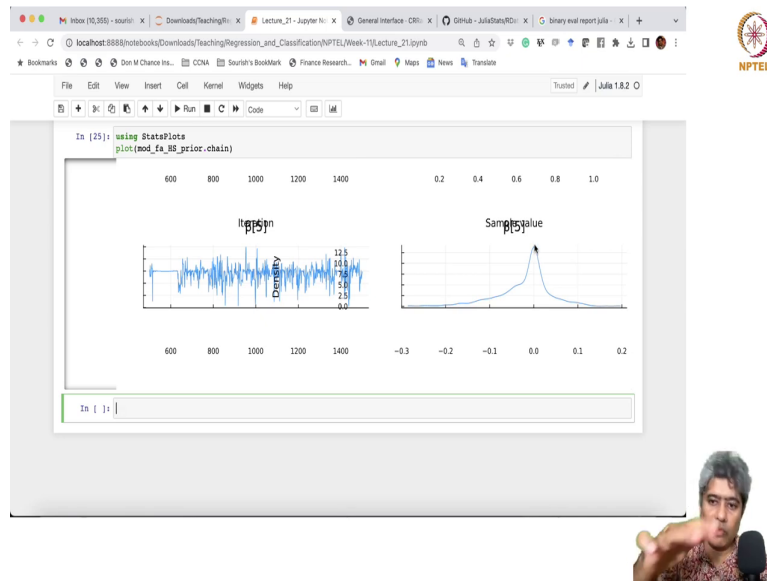


Yeah, it is better. So, first are all lambda parameters and then I will we will see here the betas. Now you can see that betas are all very close to 0 they are bit close to 0s and they are very close to sharp to the 0 and not too much. So, they see there are lot of they are not sticking and then the third one this is the second one and then this is the third one they are all positive and the entire thing is positive being shifted, ok.

(Refer Slide Time: 23:52)



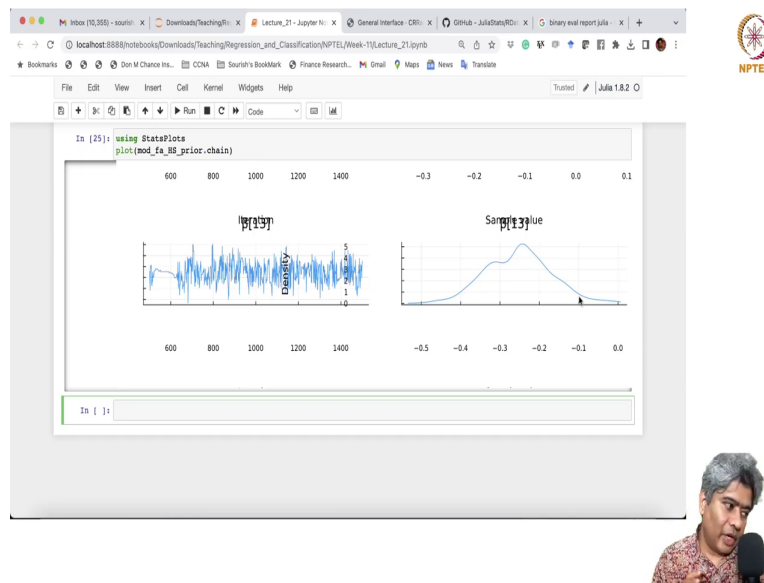
(Refer Slide Time: 23:55)



Where the others are like you know very stick to the like you know very stick to the 0 and then they have a flat distribution.

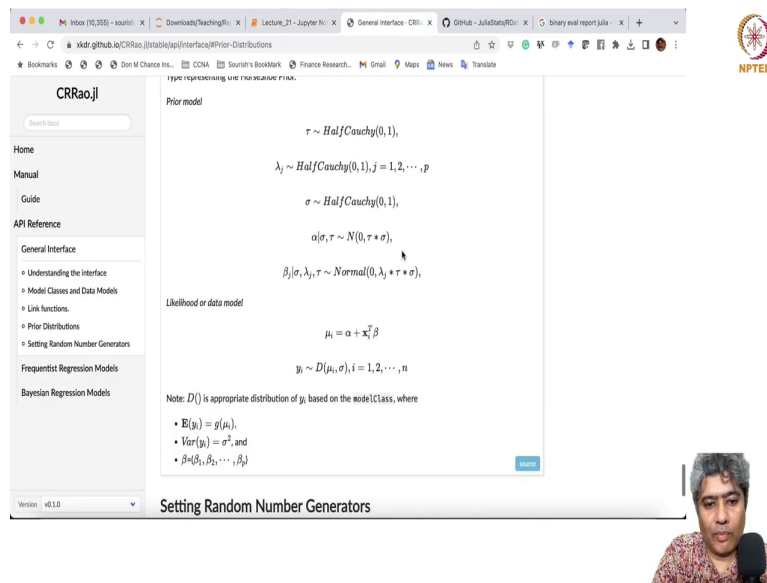


(Refer Slide Time: 24:14)



So, this is a very peculiar of the HorseShoe prior they have lot of mass on the 0 it tries to put. And similarly, but when it has a because it has a lot of mass on the tail in the prior itself when the posterior if it have any coefficient has true positive mass on the non-zero place then it gets you know it is nicely picked up that behavior also. So, you can implement HorseShoe prior in genetic you know in logistic regression very easily with just one line of code using CRRao package.

(Refer Slide Time: 24:46)



The screenshot shows the CRRao.jl website interface. The main content area displays the following mathematical expressions:

Prior model

$$\tau \sim \text{HalfCauchy}(0, 1),$$
$$\lambda_j \sim \text{HalfCauchy}(0, 1), j = 1, 2, \dots, p$$
$$\sigma \sim \text{HalfCauchy}(0, 1),$$
$$\alpha | \sigma, \tau \sim N(0, \tau * \sigma),$$
$$\beta_j | \sigma, \lambda_j, \tau \sim N(0, \lambda_j * \tau * \sigma),$$

Likelihood or data model

$$\mu_i = \alpha + \sum_j^2 \beta_j$$
$$y_i \sim D(\mu_i, \sigma), i = 1, 2, \dots, n$$

Note:  $D()$  is appropriate distribution of  $y_i$  based on the modelClass, where

- $E(y_i) = g(\mu_i)$ ,
- $\text{Var}(y_i) = \sigma^2$ , and
- $\beta = (\beta_1, \beta_2, \dots, \beta_p)$

Setting Random Number Generators

The detail about the Horseshoe Prior are given here I have given that the all these things like you know this is if  $D$  is a distribution it distribution could be normal Bernoulli anything  $\mu_i$  is  $x$  transpose beta.

So, you have to be  $\mu_i$  is in with distribution with respect to a mean and variance or standard deviation. And then beta follow Normal alpha follow Normal, but conditional normal the scale parameters are follow HalfCauchy distribution this lambda js follow HalfCauchy sigma follow HalfCauchy and the tau distribution also follow HalfCauchy.

So, with this is a very peculiar kind of you know very popular actually in the Bayesian literature and you can implement Horseshoe Prior using CRRao very easily. So, I hope you

enjoyed this video and we will keep doing more such analysis using CRRaO in the coming videos.

Thank you very much, see you in the next video.