

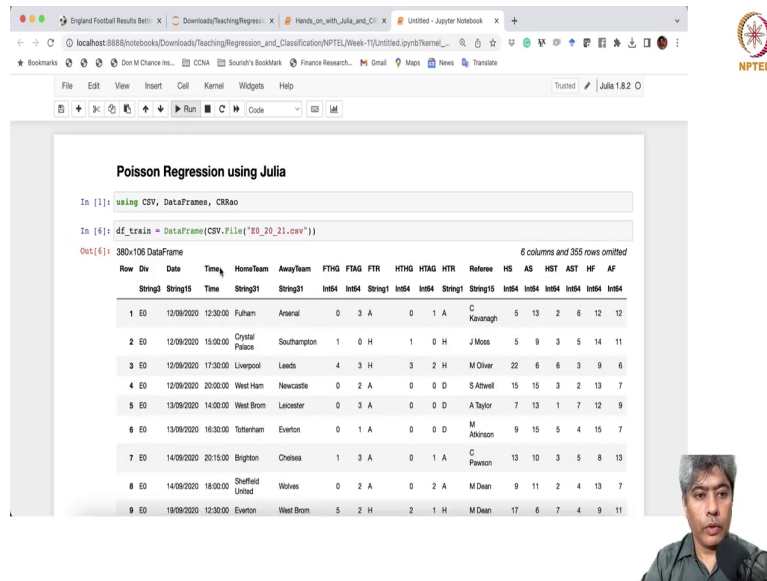
**Predictive Analytics - Regression and Classification**  
**Prof. Sourish Das**  
**Department of Mathematics**  
**Chennai Mathematical Institute**

**Lecture - 61**  
**Hands on with Julia\_Bayesian Poisson Regression with Horse Shoe English**  
**Prior\_League Data**

Hi all, in this video, I am going to do Poisson regression or count regression using Julia. And in this video, I am going to show how we can use the UK football data or English prior league data, because we have seen in the previous videos that English prior league data, the home teams code the number of goal. And if we want to model number of goal, it will be either through a Poisson regression or negative binomial regression.

So, what I am going to do in this video, I am going to show you how to do Poisson regression using Julia, ok.

(Refer Slide Time: 01:06)



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

### Poisson Regression using Julia

```
In [1]: using CSV, DataFrames, CRRao
```

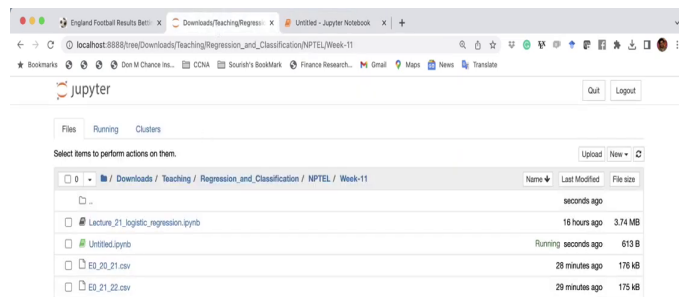
```
In [6]: df_train = DataFrames(CSV.File("E0_20_21.csv"))
```

```
Out[6]: 380x108 DataFrame
```

Row	Div	Date	Time	HomeTeam	AwayTeam	FTHG	FTAG	FTR	HTHG	HTAG	HTR	Referee	HS	AS	HST	AST	HF	AF
String1	String15	String15	String15	String21	String21	Int64	Int64	String1	Int64	String1	String15	String15	Int64	Int64	Int64	Int64	Int64	Int64
1	ED	12/09/2020	12:30:00	Fulham	Arsenal	0	3	A	0	1	A	C. Keane	5	13	2	6	12	12
2	ED	12/09/2020	15:00:00	Crystal Palace	Southampton	1	0	H	1	0	H	J. Moss	5	9	3	5	14	11
3	ED	12/09/2020	17:30:00	Liverpool	Leeds	4	3	H	3	2	H	M. Oliver	22	6	6	3	9	6
4	ED	12/09/2020	20:00:00	West Ham	Newcastle	0	2	A	0	0	D	S. Attwell	15	15	3	2	13	7
5	ED	13/09/2020	14:00:00	West Brom	Leicester	0	3	A	0	0	D	A. Taylor	7	13	1	7	12	9
6	ED	13/09/2020	16:30:00	Tottenham	Everton	0	1	A	0	0	D	M. Atkinson	9	15	5	4	15	7
7	ED	14/09/2020	20:15:00	Brighton	Chelsea	1	3	A	0	1	A	C. Pawson	13	10	3	5	8	13
8	ED	14/09/2020	18:00:00	Sheff Wed	Wolves	0	2	A	0	2	A	M. Dean	9	11	2	4	13	7
9	ED	19/09/2020	12:30:00	Everton	West Brom	5	2	H	2	1	H	M. Dean	17	6	7	4	9	11

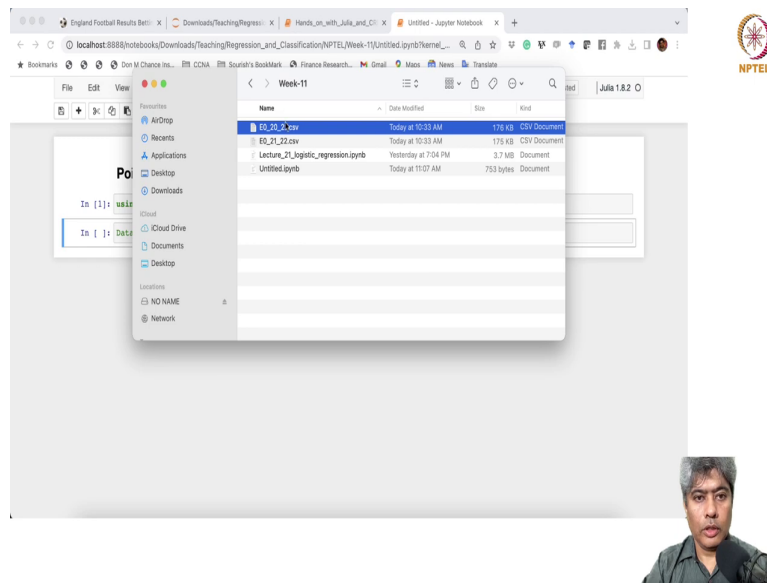
So, first in my Jupyter, I am going to start our Jupyter Notebook. So, first I will write give a title to my Notebook, say Poisson Regression using Julia, ok. And then the first thing I am going to do, I am going to call CSV DataFrames and CRRao is 3 for sure. Let me run this.

(Refer Slide Time: 01:50)



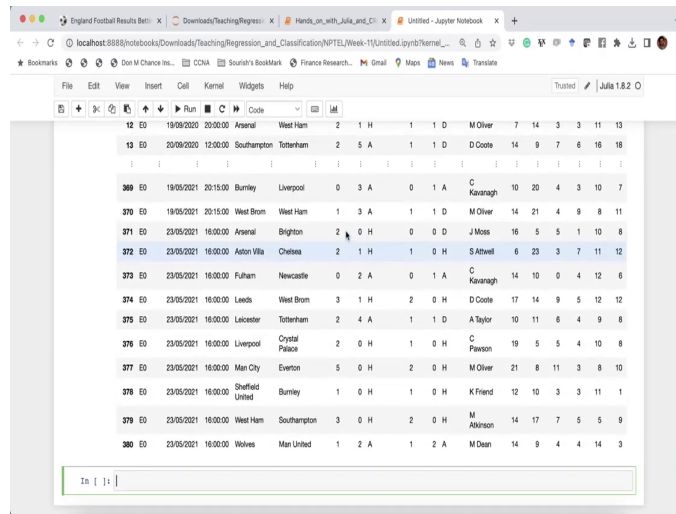
Now, in my the same folder where you are starting your Jupyter Notebook, I have downloaded 2021 data and 21 22 data. So, I am going to use 2021 data, ok.

(Refer Slide Time: 02:18)



So, what I will do DataFrame CSV dot File quote unquote, I just need the name. So, file name, you keep the file name in the same folder. So, that will be helpful, ok.

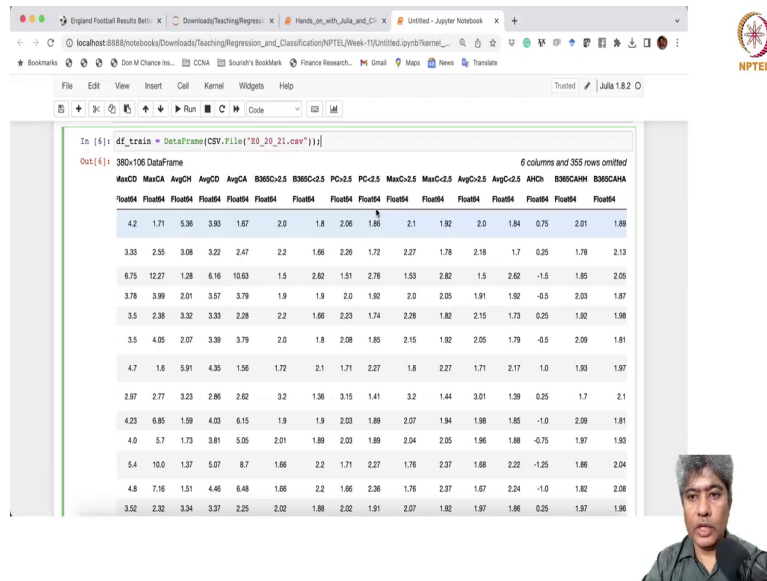
(Refer Slide Time: 02:36)



ID	Date	Time	Home Team	Away Team	Score	Goalkeeper	Goals	Assists	Yellow Cards	Red Cards	Points
12	19/09/2020	20:00:00	Arsenal	West Ham	2 1	H	1	1	D	M Oliver	7 14 3 3 11 13
13	20/09/2020	12:00:00	Southampton	Tottenham	2 5	A	1	1	D	D Coote	14 9 7 6 16 18
369	19/05/2021	20:15:00	Burnley	Liverpool	0 3	A	0	1	A	C Kavanagh	10 20 4 3 10 7
370	19/05/2021	20:15:00	West Brom	West Ham	1 3	A	1	1	D	M Oliver	14 21 4 9 8 11
371	23/05/2021	16:00:00	Arsenal	Brighton	2 0	H	0	0	D	J Moss	16 5 5 1 10 8
372	23/05/2021	16:00:00	Aston Villa	Chelsea	2 1	H	1	0	H	S Atwell	6 23 3 7 11 12
373	23/05/2021	16:00:00	Fulham	Newcastle	0 2	A	0	1	A	C Kavanagh	14 10 0 4 12 6
374	23/05/2021	16:00:00	Leeds	West Brom	3 1	H	2	0	H	D Coote	17 14 9 5 12 12
375	23/05/2021	16:00:00	Leicester	Tottenham	2 4	A	1	1	D	A Taylor	10 11 6 4 9 8
376	23/05/2021	16:00:00	Liverpool	Crystal Palace	2 0	H	1	0	H	C Pearson	19 5 5 4 10 8
377	23/05/2021	16:00:00	Man City	Everton	5 0	H	2	0	H	M Oliver	21 8 11 3 8 10
378	23/05/2021	16:00:00	Sheffield United	Burnley	1 0	H	1	0	H	K Friend	12 10 3 3 11 1
379	23/05/2021	16:00:00	West Ham	Southampton	3 0	H	2	0	H	M Alkinson	14 17 7 5 5 9
380	23/05/2021	16:00:00	Wolves	Man United	1 2	A	1	2	A	M Dean	14 9 4 4 14 3



(Refer Slide Time: 02:50)



In [6]: `df_train = DataFrame(CSV.File("E9_20_21.csv"))`

Out[6]: 380x106 DataFrame

MaxCD	MaxCA	AvgCH	AvgCD	AvgCA	B365C-2.5	B365C-2.5	PC-2.5	PC-2.5	MaxC-2.5	MaxC-2.5	AvgC-2.5	AvgC-2.5	AHCh	B365CAH	B365CAH
Float64	Float64	Float64	Float64	Float64	Float64	Float64	Float64	Float64	Float64	Float64	Float64	Float64	Float64	Float64	Float64
4.2	1.71	5.36	3.93	1.67	2.0	1.8	2.06	1.86	2.1	1.92	2.0	1.84	0.75	2.01	1.89
3.33	2.55	3.08	3.22	2.47	2.2	1.66	2.26	1.72	2.27	1.76	2.18	1.7	0.25	1.78	2.13
6.75	12.27	1.28	6.16	10.63	1.5	2.82	1.51	2.76	1.53	2.82	1.5	2.82	-1.5	1.85	2.05
3.78	3.99	2.01	3.57	3.79	1.9	1.9	2.0	1.92	2.0	2.05	1.91	1.92	-0.5	2.03	1.87
3.5	2.38	3.32	3.33	2.28	2.2	1.66	2.23	1.74	2.28	1.82	2.15	1.73	0.25	1.92	1.98
3.5	4.05	2.07	3.39	3.79	2.0	1.8	2.08	1.85	2.15	1.92	2.05	1.79	-0.5	2.09	1.81
4.7	1.8	5.91	4.35	1.56	1.72	2.1	1.71	2.27	1.8	2.27	1.71	2.17	1.0	1.93	1.97
2.97	2.77	3.23	2.86	2.62	3.2	1.96	3.15	1.41	3.2	1.44	3.01	1.98	0.25	1.7	2.1
4.23	6.85	1.59	4.03	6.15	1.9	1.9	2.03	1.89	2.07	1.94	1.98	1.85	-1.0	2.09	1.81
4.0	5.7	1.73	3.81	5.05	2.01	1.89	2.03	1.89	2.04	2.05	1.96	1.88	-0.75	1.97	1.93
5.4	10.0	1.37	5.07	8.7	1.86	2.2	1.71	2.27	1.76	2.37	1.68	2.22	-1.25	1.86	2.04
4.8	7.16	1.51	4.46	6.48	1.66	2.2	1.66	2.36	1.76	2.37	1.67	2.24	-1.0	1.82	2.08
3.52	2.32	3.34	3.37	2.25	2.02	1.88	2.02	1.91	2.07	1.82	1.97	1.96	0.25	1.97	1.96

So, this gives me the data set that I am looking for. So, there are 380 gaming are being played, ok. And there are 106 columns are there. So, there is lots of columns are there, ok.

(Refer Slide Time: 03:04)

The screenshot shows the bet365 website interface. On the left, there are three match cards:

- Spain Primera Liga: Real Madrid v Valladolid** (Mon 01:15)  
Odds: 1 (1.20), X (1.20), 2 (1.20)
- Italy Serie A: Napoli v AC Milan** (Mon 05:45)  
Odds: 1 (1.83), X (1.83), 2 (1.83)

The main content area lists seasons from 2022/2023 down to 2018/2019, with links for Premier League, Championship, League 1, League 2, and Conference. Each link includes a brief description: "(FT & HT results; match stats; match, total goals & AH odds)".

On the right side, there are several navigation menus:

- BETTING ARTICLES:** Football-Data, Pinnacle Sportsbook
- BET CALCULATORS:** Fair Odds, P-value, Yields, Bank growth, EV-odds, Staking Animation
- ODDS & RESULTS: MAIN LEAGUES:** Latest Matches, England, Scotland, Germany, Italy, Spain, France, Netherlands, Belgium



(Refer Slide Time: 03:06)

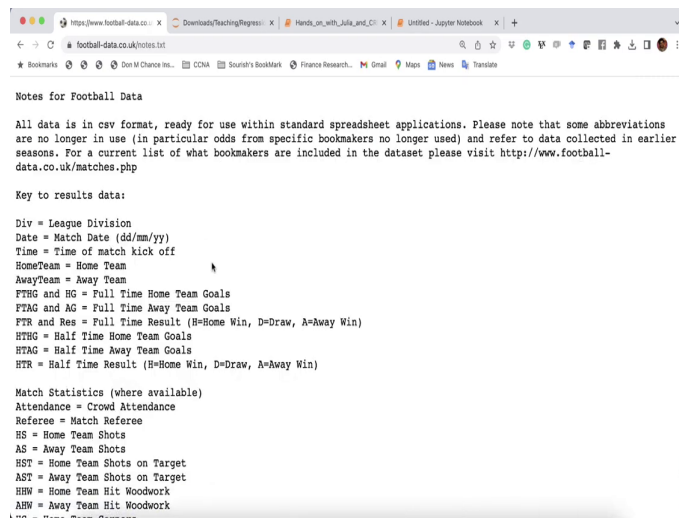
The screenshot shows a web browser displaying the website <https://www.football-data.co.uk/notes.txt>. The page features a sidebar with logos for BoyleSports, William Hill, Betfred, and Betway. The main content area includes a 'Notes.txt' section with a text file key to data files and data source acknowledgements. Below this, there are sections for 'Season 2022/2023', 'Season 2021/2022', and 'Season 2020/2021', each listing various leagues and their corresponding data files. A prominent advertisement for bet365 is visible, showing a match between Real Madrid and Valladolid on Monday, 01:15, with odds of 1.20 for home, draw, and away.

1	X	2
1.20	1.20	1.20





(Refer Slide Time: 03:07)



```
Notes for Football Data

All data is in csv format, ready for use within standard spreadsheet applications. Please note that some abbreviations
are no longer in use (in particular odds from specific bookmakers no longer used) and refer to data collected in earlier
seasons. For a current list of what bookmakers are included in the dataset please visit http://www.football-
data.co.uk/matches.php

Key to results data:

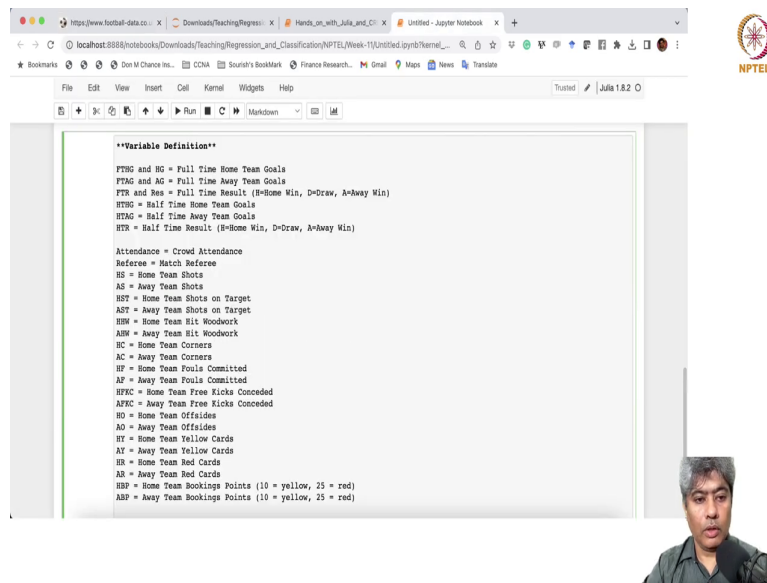
Div = League Division
Date = Match Date (dd/mm/yy)
Time = Time of match kick off
HomeTeam = Home Team
AwayTeam = Away Team
FTHG and HG = Full Time Home Team Goals
FTAG and AG = Full Time Away Team Goals
FTR and Res = Full Time Result (H=Home Win, D=Draw, A=Away Win)
HTHG = Half Time Home Team Goals
HTAG = Half Time Away Team Goals
HTR = Half Time Result (H=Home Win, D=Draw, A=Away Win)

Match Statistics (where available)
Attendance = Crowd Attendance
Referee = Match Referee
HS = Home Team Shots
AS = Away Team Shots
HST = Home Team Shots on Target
AST = Away Team Shots on Target
HHW = Home Team Hit Woodwork
AHW = Away Team Hit Woodwork
```



And now my what I am going to do, I want to model number of goals scored by. So, if I just go back there and you know, if I Note dot txt click on that. So, I want to come full-time home team goal that is FTHG. This is what I want to model, ok. So, what I want to model is let us the model is, ok.

(Refer Slide Time: 03:30)



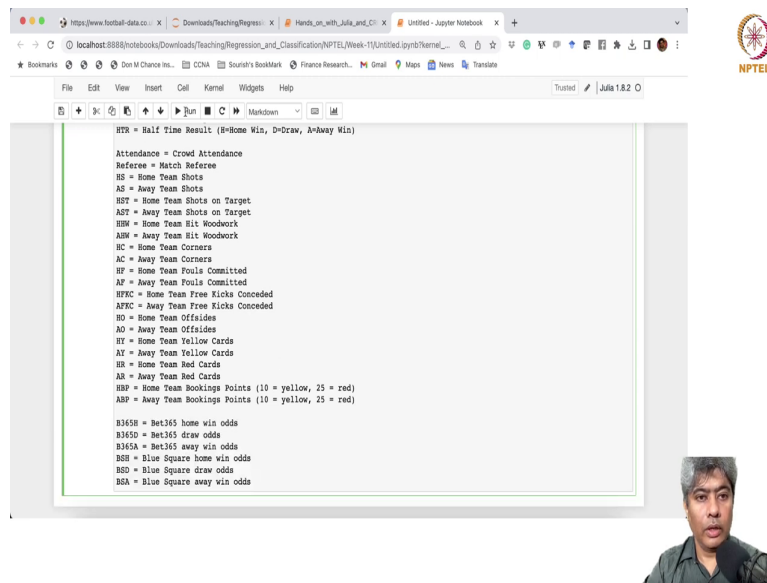
The screenshot shows a Jupyter Notebook interface with a browser window at the top. The notebook contains a code cell with the following text:

```
**Variable Definition**  
FTHG and HG = Full Time Home Team Goals  
FTAG and AG = Full Time Away Team Goals  
FTR and Res = Full Time Result (H=Home Win, D=Draw, A=Away Win)  
HTHG = Half Time Home Team Goals  
HTAG = Half Time Away Team Goals  
HTR = Half Time Result (H=Home Win, D=Draw, A=Away Win)  
  
Attendance = Crowd Attendance  
Referee = Match Referee  
HS = Home Team Shots  
AS = Away Team Shots  
HST = Home Team Shots on Target  
AST = Away Team Shots on Target  
HSW = Home Team Hit Woodwork  
ASW = Away Team Hit Woodwork  
HC = Home Team Corners  
AC = Away Team Corners  
HF = Home Team Fouls Committed  
AF = Away Team Fouls Committed  
HFKC = Home Team Free Kicks Conceded  
AFKC = Away Team Free Kicks Conceded  
HO = Home Team Offsides  
AO = Away Team Offsides  
HY = Home Team Yellow Cards  
AY = Away Team Yellow Cards  
HR = Home Team Red Cards  
AR = Away Team Red Cards  
HBP = Home Team Bookings Points (10 = yellow, 25 = red)  
ABP = Away Team Bookings Points (10 = yellow, 25 = red)
```

In the bottom right corner of the notebook interface, there is a small video feed window showing a man's face.

First is let me just at least write down few things FTHG. These are the basic definitions. Let me just copy it down, ok. These are the variable definition, ok. Variable Definition, ok. And then I will just copy few more things. Can there is all these information's so I am going to, ok.

(Refer Slide Time: 04:23)



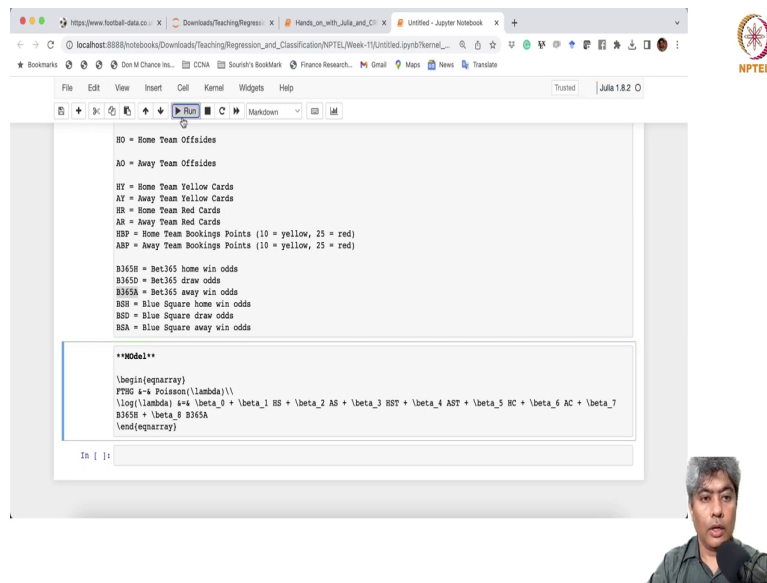
The screenshot shows a Jupyter Notebook interface with a list of variables for a regression model. The variables are:

- HTR = Half Time Result (H=Home Win, D=Draw, A=Away Win)
- Attendance = Crowd Attendance
- Referee = Match Referee
- RS = Home Team Shots
- AS = Away Team Shots
- RS7 = Home Team Shots on Target
- AS7 = Away Team Shots on Target
- HSH = Home Team Hit Woodwork
- ASH = Away Team Hit Woodwork
- HC = Home Team Corners
- AC = Away Team Corners
- HF = Home Team Fouls Committed
- AF = Away Team Fouls Committed
- HFKC = Home Team Free Kicks Conceded
- AFKC = Away Team Free Kicks Conceded
- HO = Home Team Offsides
- AO = Away Team Offsides
- HY = Home Team Yellow Cards
- AY = Away Team Yellow Cards
- HR = Home Team Red Cards
- AR = Away Team Red Cards
- HBP = Home Team Bookings Points (10 = yellow, 25 = red)
- ABP = Away Team Bookings Points (10 = yellow, 25 = red)
- B365H = bet365 home win odds
- B365D = bet365 draw odds
- B365A = bet365 away win odds
- BBS = Blue Square home win odds
- BSD = Blue Square draw odds
- BBSA = Blue Square away win odds

An NPTEL logo is visible in the top right corner of the notebook interface. A small video feed of a person is visible in the bottom right corner of the notebook window.

And some of the betting statistics as well, ok. Opsee; alright so, let me just stop here. So, these are the, so, now, FTHG is the one of the variable here FTHG. This is the home teams code basically full-time. How many goals scored by the home team?

(Refer Slide Time: 05:04)



The screenshot shows a Jupyter Notebook interface with a list of variables and a model definition. The variables are:

- HO = Home Team Offsides
- AO = Away Team Offsides
- HY = Home Team Yellow Cards
- AY = Away Team Yellow Cards
- HR = Home Team Red Cards
- AR = Away Team Red Cards
- HSP = Home Team Bookings Points (10 = yellow, 25 = red)
- ASP = Away Team Bookings Points (10 = yellow, 25 = red)
- B365H = Bet365 home win odds
- B365D = Bet365 draw odds
- B365A = Bet365 away win odds
- B5S = Blue Square home win odds
- B5D = Blue Square draw odds
- B5A = Blue Square away win odds

The model definition is:

```
**Model**  
\begin{eqnarray}  
FT9S \sim \text{Poisson}(\lambda) \\ \log(\lambda) \sim \beta_0 + \beta_1 HS + \beta_2 AS + \beta_3 HST + \beta_4 AST + \beta_5 BC + \beta_6 AC + \beta_7  
B365H \sim \beta_8 B365A  
\end{eqnarray}
```

So, this is what the, what we want to model. And we want to model as a function of what? We want to model it as a function of maybe, you know, home team, how many shots were taken by the home team? And like HS. And how many shots were taken by away team? Ok. And then how many home team shots on target? Ok. And then away team shots on target. Then home team corners. How many corners were taken by home team away team corners?

And then maybe BETrix 65, the score by BETrix 365. And away teams score all these things we want to model. This is the model that we want to model. But the home team is. So, model this is my model. This is the model that we want to fit, ok. So, this is, so, effectively this will be like home Poisson,  $\lambda$  Poisson  $\lambda$ . And then  $\log$  of  $\lambda$  BDA  $\lambda$  equals to  $\beta_0$  plus  $\beta_1$  HS so, this way  $\beta_2$ ,  $\beta_3$ ,  $\beta_4$ ,  $\beta_5$ ,  $\beta_6$ ,  $\beta_7$  and  $\beta_8$ .

So, this is pretty much what we have, ok. So, let me not take dollar here. So, maybe, I will just put it like this. I will put it like this. Let me run it, ok. Yeah so, maybe I will just put, yeah. If I just do this, probably this is, we will make, ok. So, maybe I will just put begin equation array, e q n a r r a y and end eqnarray. And it will work.

(Refer Slide Time: 08:41)

```

ASP = Away Team Bookings Points (10 = yellow, 25 = red)

B365H = Bet365 home win odds
B365D = Bet365 draw odds
B365A = Bet365 away win odds
BSH = Blue Square home win odds
BSD = Blue Square draw odds
BSA = Blue Square away win odds

Model
FTHG Poisson(λ)
log(λ) = β₀ + β₁HS + β₂AS + β₃HST + β₄AST + β₅HC + β₆AC + β₇B365H + β₈B365A

MLE Estimates

In [7]: using CRRao
model = fit(#formula=FTHG ~ BS + AS + BSH + AST + BC + AC +B365H + B365A),df_train,PoissonRegression()

LoadError: UndefinedError: #formula not defined
in expression starting at In[7]:3

In [ ]:

```

Yeah. So, this is what the, this is what the model I want to fit. And so, for that, I am going to call CRRao. And the first model that we are going to fit is fit, ok. At the rate formula, ok and first is FTHG, FTHG and then HS plus AS plus HST plus AST plus HC plus AC plus B365, opsee B365H plus B365A. Now, after that, I have to give the name of the data set df train, ok. And then I have to give, I have to say the class of the thing, but X.

(Refer Slide Time: 10:18)

The screenshot shows the GitHub repository page for `xKDR/CRRao.jl`. The repository is public and has 16 branches and 2 tags. The main branch is selected. The repository has 240 commits and 17 stars. The repository is licensed under MIT. The repository has 3 watchers and 12 forks. The repository has 1 release, `v0.1.0`, which is the latest version. The repository has no packages published.

File	Description	Time
<code>.github</code>	Update doc CI to trigger Documenter	3 months ago
<code>docs</code>	Merge branch 'main' into 'doc_update'	6 months ago
<code>src</code>	Add Bootstrap Regression (Linear Regression)	2 months ago
<code>test</code>	Updated tests to use new version of Bijectors	last month
<code>.JuliaFormatter.toml</code>	Added config file for JuliaFormatter, and did prettification on	10 months ago
<code>.gitignore</code>	Initiating repository	last year
<code>CONTRIBUTING.md</code>	Create CONTRIBUTING.md	2 months ago
<code>LICENSE</code>	Initiating repository	last year
<code>Project.toml</code>	Added brush pagan test	2 months ago
<code>README.md</code>	Fix readme	2 months ago



(Refer Slide Time: 10:22)

https://www.facebook-data.co... Downloads/Teaching/Regress... Handson\_with\_Julia\_and\_CI... Untitled - Jupyter Notebook... GitHub - xDR/CRRao.jl

codetalker7 Siddhant Chaudhary  
sourish-omi Sourish  
ShouvikGhosh2048 Shouvik Ghosh  
ayushpatnalkgt Ayush Patnalk

Languages  
Julia 100.0%

README.md

docs stable docs API try CI testing Documentation testing codecov styleciemr precommit

To install:

For version 0.1.0 (stable)

```
using Pkg; Pkg.add("CRRao")
```

For version 0.1.1 (under development)

```
using Pkg; Pkg.add(url = "https://github.com/xDR/CRRao.jl.git")
```



**CRRao: A single API for diverse statistical models**

Many statistical models can be estimated in Julia, and the diversity of the model ecosystem is steadily improving. Drawing inspiration from the `Zelig` package in the R world, the CRRao package gives a simple and consistent API to the end user. The end-user then faces the fixed cost of getting a hang of this, once, and after that a wide array of models and associated capabilities become available with a consistent syntax. We hope others developing statistical models will build within this framework.

Here's an example of estimating the linear regression

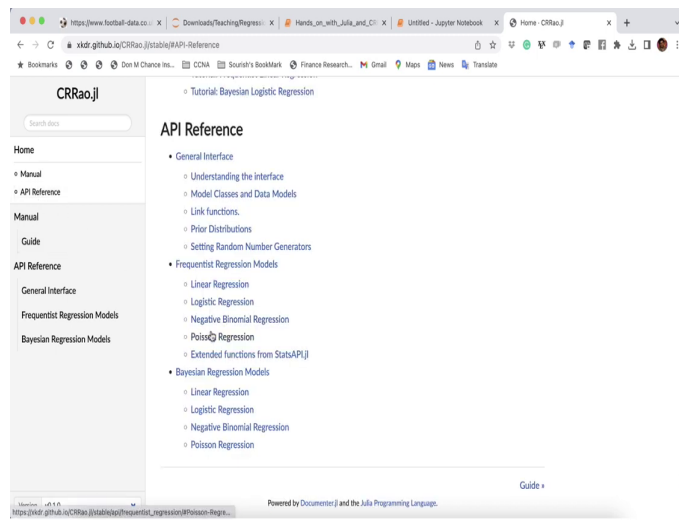
$$MPG = \beta_0 + \beta_1 HP + \beta_2 WT + \beta_3 Gear + \epsilon$$

Waiting for collector.github.com...



So, maybe I will just CRRao package. I will just go to the stable, doc package.

(Refer Slide Time: 10:32)



The screenshot shows a web browser displaying the CRRao.jl website. The browser's address bar shows the URL: `https://www.footbal-data.co.uk/.../vkrdr.github.io/CRRao.jl/stable/API-Reference`. The website has a dark theme and a sidebar on the left with navigation links: Home, Manual, API Reference, Manual, Guide, API Reference, General Interface, Frequentist Regression Models, and Bayesian Regression Models. The main content area is titled "API Reference" and contains a list of topics under "General Interface":

- General Interface
  - Understanding the Interface
  - Model Classes and Data Models
  - Link Functions.
  - Prior Distributions
  - Setting Random Number Generators
- Frequentist Regression Models
  - Linear Regression
  - Logistic Regression
  - Negative Binomial Regression
  - Poisson Regression
  - Extended functions from StatsAPI.jl
- Bayesian Regression Models
  - Linear Regression
  - Logistic Regression
  - Negative Binomial Regression
  - Poisson Regression

At the bottom of the page, it says "Powered by Documenter.jl and the Julia Programming Language." and "Guide" with a right-pointing arrow.





(Refer Slide Time: 10:35)

**CRRao.jl**

StatsAPI: fit - Method

```
fit(formula::FormulaTerm, data::DataFrame, modelClass::PoissonRegression)
```

Fit a Poisson Regression model on the input data (with the default link function being the Log link). Uses the `glm` method from the GLM package under the hood. Returns an object of type `FrequentistRegression{PoissonRegression}`.

**Example**

```
Julia> using CRRao, Rdatasets, StatsModels
Julia> sanction = dataset("Zelig", "sanction")
78x8 DataFrame
Row      Mil      Coop      Target      Import      Export      Cost      Num      NCost
      Int32      Int32      Int32      Int32      Int32      Int32      Int32      Cat...
1         1         4         3         1         1         4         15      major loss
2         0         2         3         0         1         3         4         modest loss
3         0         1         3         1         0         2         1         little effect
4         1         1         3         1         1         2         1         little effect
5         0         1         3         1         1         2         1         little effect
6         0         1         3         0         1         2         1         little effect
...
73        1         3         1         1         1         2         14         little effect
74         0         2         1         0         0         1         2         net gain
75         0         1         3         0         1         2         1         little effect
76         0         4         3         1         0         2         13         little effect
77         0         1         2         0         0         1         1         net gain
78        1         3         1         1         1         2         10         little effect
```



(Refer Slide Time: 10:38)

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

```
Julia> container = fit([formula(Nun ~ Target + Coop + WCost), sanction], PoissonRegression())
```

	Coef.	Std. Error	z	Pr(> z )	Lower 95%	Upper 95%
(Intercept)	-1.91392	0.261667	-7.31	<1e-12	-2.43678	-1.40106
Target	0.167769	0.0653822	2.41	0.0168	0.0296218	0.265916
Coop	1.15127	0.0561861	20.49	<1e-92	1.04114	1.26139
WCost: major loss	-0.324051	0.230955	-1.41	0.1590	-0.774951	0.126848
WCost: modest loss	1.73973	0.109518	17.11	<1e-64	1.52272	1.91674
WCost: net gain	0.463967	0.16992	2.73	0.0063	0.13987	0.786944



So, API reference Poisson regression, what I will do is what I have to do. Yeah. So, Poisson regression, that is what we have to do here. So, PoissonRegression. So, if I just, if you write POI and then tab, it will fill it up for itself. Now, if you just do this, this will fill a, set a fit a maximum likelihood estimate, ok.

Maximum likelihood estimate or MLE estimates, ok. Let me just put it up there. Yeah, and now let me just run this. Why it is not formula? Ok. So, I have to, maybe I have to call a StatsModels. Yeah StatModels, I need. Yeah so, yeah. And I think now it is fine, ok.

(Refer Slide Time: 11:50)

	Coef.	Std. Error	z	Pr(> z )	Lower 95%	Upper 95%
(Intercept)	-0.352743	0.236089	-1.49	0.1351	-0.815468	0.109983
BS	-0.034822	0.014178	-2.46	0.0140	-0.062696	-0.00703447
AS	-0.023019	0.0139061	-1.66	0.0979	-0.0502744	0.00423639
HST	0.238636	0.0229936	10.38	<1e-24	0.193569	0.283703
AST	0.0255725	0.0271946	0.94	0.3470	-0.0277279	0.0788729
BC	-0.0428155	0.0186027	-2.25	0.0290	-0.0804321	-0.00719804
AC	0.0344269	0.0193173	1.78	0.0747	-0.00343431	0.0722882
B365H	-0.0248462	0.0300651	-0.83	0.4086	-0.0837727	0.0340803
B365A	0.0369102	0.0145484	2.53	0.0114	0.00829595	0.0653245

Bayesian Poisson Regression with Ridge Prior

```
In [ ]: mod_ridge = fit(formula(FBGO ~ BS + AS + HST + AST + BC + AC + B365H + B365A),  
                    ,df_train,PoissonRegression(),  
                    ,Prior_Ridge())
```

So, if I look into it, HST home, number of shot on target by home team is either going to have a very strong effect, positive effect. Obviously more shot you have on the target, the chance of having score is I. And then this is another home team, now shot and has a effect. And then this is also home team number of goal, number of corner by home team. And what is the weight 365s, hours odds for the away team? That also is the important role.

And you can see that this coefficient has a positive effect alright interesting phenomena that we are getting. Now, suppose we want to predict, suppose we want to fit Bayesian regression model, Bayesian Poisson regression model with, say ridge prior Bayesian Poisson let me Bayesian Poisson Regression with Ridge Prior, Ridge Prior. So, if I just run this. So, the if you just go there, all I have to do is essentially just copy this thing. And; obviously, now I want to change the name, maybe I will say ridge.

And after the Poisson regression, I would say Prior Ridge. And if you just Prior underscore R, and then if you just tab, it will take the rest of the thing, ok. So, you do not have to really worry about that. Let me just put it here. So, that you know it is kind of aligned. And you can see the entire model.

(Refer Slide Time: 14:19)

The screenshot shows a Jupyter Notebook interface with a Julia code cell. The code defines a model with Ridge regularization and fits it to training data. The output includes the fitted formula, link function, chain configuration, and a summary statistics table for the parameters.

```
In [9]: mod_ridge = fit((formula(FBFG ~ BS + AS + BST + AST + BC + AC + B365H + B365A)
                        , fit_train.PoissonRegression()
                        , Prior_Ridge()))

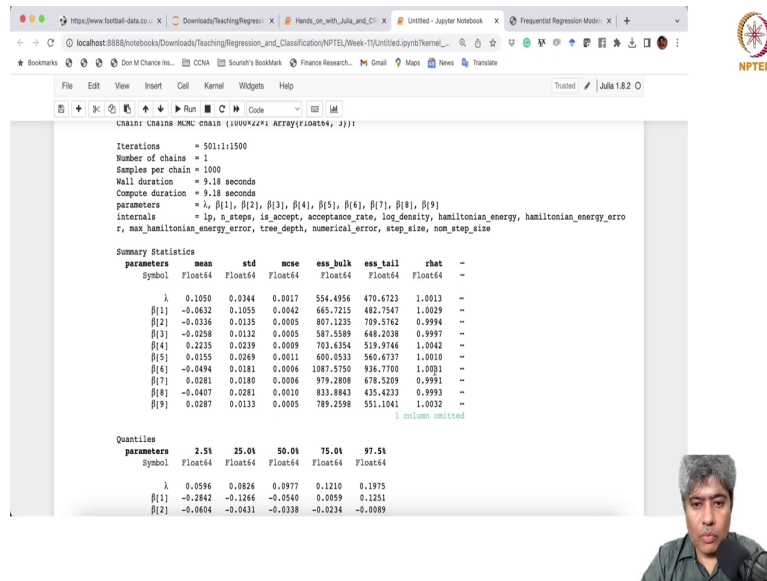
r Info: Found initial step size
      ε = 2.273736754432207e-14
      Sampling: 1001 | Time: 0:00:02

Out[9]: Formula: FBFG ~ 1 + BS + AS + BST + AST + BC + AC + B365H + B365A
Link: CRao.Identity(CRao.Identity_Link)
Chain: Chains MCMC chain (1000×2×1 Array{Float64, 3}):

Iterations = 5011j1500
Number of chains = 1
Samples per chain = 1000
Wall duration = 9.18 seconds
Compute duration = 9.18 seconds
parameters = λ, β[1], β[2], β[3], β[4], β[5], β[6], β[7], β[8], β[9]
internals = lp, n_steps, is_accept, acceptance_rate, log_density, hamiltonian_energy, hamiltonian_energy_error
r_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 0.1050 0.0344 0.0017 554.4956 470.6723 1.0013 -
β[1] -0.0632 0.1055 0.0042 665.7215 482.7547 1.0029 -
β[2] -0.0336 0.0135 0.0005 807.1235 709.5762 0.9994 -
β[3] -0.0258 0.0132 0.0005 587.5589 648.2038 0.9997 -
β[4] 0.2235 0.0239 0.0009 703.6354 519.9746 1.0042 -
β[5] 0.0155 0.0269 0.0011 600.0533 560.6737 1.0010 -
β[6] -0.0494 0.0181 0.0006 1087.5750 936.7700 1.0031 -
β[7] 0.0281 0.0180 0.0006 979.2808 678.5209 0.9991 -
```

(Refer Slide Time: 14:26)



Summary statistics

parameters	mean	std	mcse	ess_bulk	ess_tail	rhat	
Symbol	Float64	Float64	Float64	Float64	Float64	Float64	-
$\lambda$	0.1050	0.0344	0.0017	554.4956	470.6723	1.0013	-
$\beta[1]$	-0.0632	0.1055	0.0042	665.7215	482.7547	1.0029	-
$\beta[2]$	-0.0236	0.0135	0.0005	807.1235	709.5762	0.9994	-
$\beta[3]$	-0.0258	0.0132	0.0005	587.5589	646.2038	0.9997	-
$\beta[4]$	0.2235	0.0239	0.0009	703.6354	519.9746	1.0042	-
$\beta[5]$	0.0155	0.0269	0.0011	600.0533	560.6737	1.0010	-
$\beta[6]$	-0.0494	0.0181	0.0006	1087.5750	936.7700	1.0001	-
$\beta[7]$	0.0281	0.0180	0.0006	979.2698	678.5209	0.9991	-
$\beta[8]$	-0.0407	0.0281	0.0010	833.8843	435.4233	0.9993	-
$\beta[9]$	0.0287	0.0133	0.0005	789.2598	551.1041	1.0032	-

Quantiles

parameters	2.5%	25.0%	50.0%	75.0%	97.5%
Symbol	Float64	Float64	Float64	Float64	Float64
$\lambda$	0.0596	0.0826	0.0977	0.1210	0.1975
$\beta[1]$	-0.2842	-0.1266	-0.0540	0.0059	0.1251
$\beta[2]$	-0.0604	-0.0431	-0.0338	-0.0234	-0.0099

So, let me just run it. So, if you look into the, by default, it always simulate 1000 samples. And if you look into the rhat, you see the rhats are all close to 1. So, it took 500 burning. And after that, it simulated 1000 samples after burning. It is very fast because it uses Hamiltonian Monte Carlo method.

(Refer Slide Time: 14:50)

```
r_max_hamiltonian_energy_error, tree_optn, numerical_error, step_size, non_step_size

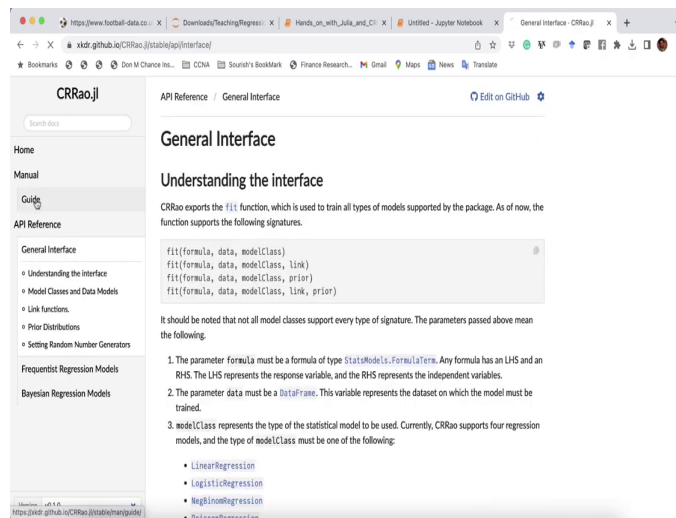
Summary Statistics
parameters      mean      std      mcr     ess_bulk  ess_tail  rhat  -
Symbol  Float64  Float64  Float64  Float64  Float64  Float64  -
lambda         0.1050  0.0344  0.0017  554.4956  470.6723  1.0013  -
beta[1]       -0.0632  0.1095  0.0042  665.7215  482.7547  1.0029  -
beta[2]       -0.0336  0.0135  0.0005  807.1235  709.5762  0.9994  -
beta[3]       -0.0258  0.0132  0.0005  587.5589  648.2038  0.9997  -
beta[4]       0.2235  0.0239  0.0009  703.6354  519.9746  1.0042  -
beta[5]       0.0355  0.0269  0.0011  606.0533  560.6737  1.0010  -
beta[6]       -0.0494  0.0181  0.0006  1087.5750  936.7700  1.0031  -
beta[7]       0.0281  0.0180  0.0006  979.2808  678.5209  0.9991  -
beta[8]       -0.0407  0.0281  0.0010  833.8943  435.4233  0.9993  -
beta[9]       0.0287  0.0133  0.0005  789.2598  551.1041  1.0032  -
1 column omitted

Quantiles
parameters      2.5%    25.0%    50.0%    75.0%    97.5%
Symbol  Float64  Float64  Float64  Float64  Float64
lambda         0.0596  0.0826  0.0977  0.1210  0.1975
beta[1]       -0.2842  -0.1266  -0.0540  0.0859  0.1251
beta[2]       -0.0604  -0.0431  -0.0338  -0.0234  -0.0089
beta[3]       -0.0516  -0.0347  -0.0258  -0.0166  -0.0094
beta[4]       0.1749  0.0076  0.2236  0.2400  0.2691
beta[5]       -0.0389  -0.0032  0.0163  0.0338  0.0655
beta[6]       -0.0859  -0.0617  -0.0492  -0.0365  -0.0154
beta[7]       -0.0093  0.0162  0.0285  0.0396  0.0644
beta[8]       -0.0941  -0.0590  -0.0416  -0.0221  0.0157
beta[9]       0.0023  0.0202  0.0286  0.0369  0.0563
```

And these are the coefficients estimates. So, here you have 1, 2, 3, 4, 5, 6, 7, 8, 8 coefficients and intercept and now here also. So, total 9. So, you can see there are 9 coefficients, including first one is the intercept and lambda is the scale parameter that is required, ok.

So, intercept is negative 0.35 in MLE. And we are getting slightly off here, negative 0, 6. Others are like negative 0, 3 for HS, negative 0 3, negative 0 2, 4, yeah. So, the other value that looks like 0.238 for HST, which is 2, 3, 4th coefficient 0.22, yeah. So, the coefficients looks like similar to close to that of MLE. But it is very simple.

(Refer Slide Time: 16:17)



The screenshot shows a web browser displaying the API Reference for the CRRao.jl package, specifically the "General Interface" section. The page title is "General Interface" and it includes a search bar and navigation links. The main content area is titled "Understanding the interface" and explains that CRRao exports the `fit` function. It lists four function signatures: `fit(formula, data, model{Class})`, `fit(formula, data, model{Class}, link)`, `fit(formula, data, model{Class}, prior)`, and `fit(formula, data, model{Class}, link, prior)`. A note states that not all model classes support every type of signature. A list of three points explains the parameters: 1. `formula` is a `StatModels.FormulaTerm` with LHS and RHS. 2. `data` is a `DataFrame`. 3. `model{Class}` is the statistical model type, with examples like `LinearRegression`, `LogisticRegression`, and `RegBinomialRegression`.



(Refer Slide Time: 16:19)

Manual / Guide [Edit on GitHub](#)

## Package Guide

### Installation

To install the package, type `] in the Julia REPL to enter the Pkg mode and run`

```
Julia> using Pkg
Julia> Pkg.add("https://github.com/xkr/CRRao.jl")
```

### Tutorial: Frequentist Linear Regression

The goal of the CRRao package is to try to unify calling variety of statistical models under the same API. Note that this is different from what something like `StatsAPI.jl` is doing: instead of introducing namespaces for development of packages, CRRao tries to call those packages with a uniform API. A very similar package comes from the R world: the [Zelig Project](#).

To see how this API works, we will go over an example in which we'll train a linear regression model with the usual ordinary least squares method (which falls under the category of the frequentist viewpoint of statistics). For our example, we will be working with the `mtcars` dataset.

We first import the required packages.

```
Julia> using CRRao, RDatasets, StatsPlots, Plots, StatsModels
```

Then we import the dataset.





(Refer Slide Time: 16:22)

CRRao.jl

### Prior Distributions

CRRao.Prior.Sauss - Type

Prior.Sauss

Type representing the Gaussian Prior. Users have specific prior mean and standard deviation, for  $\alpha$  and  $\beta$  for linear regression model.

Prior model

$$\sigma \sim \text{InverseGamma}(a_0, b_0),$$
$$\alpha | \sigma, v \sim \text{Normal}(\alpha_0, \sigma_\alpha),$$
$$\beta | \sigma, v \sim \text{Normal}(\beta_0, \sigma_\beta),$$

Likelihood or data model

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

Note:  $\mathcal{N}()$  is Gaussian distribution of  $y_i$ , where

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



(Refer Slide Time: 16:28)

The screenshot shows the CRRao.jl API reference page for the Ridge Prior. The page is titled "Prior\_Ridge" and contains the following content:

Type representing the Ridge Prior.

Prior model

$$v \sim \text{InverseGamma}(h, k),$$
$$\sigma \sim \text{InverseGamma}(a_0, b_0),$$
$$\alpha, \sigma, v \sim \text{Normal}(0, v * \sigma),$$
$$\beta | \sigma, v \sim \text{Normal}_p(0, v * \sigma),$$

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 modelClass, where

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

Version v0.1.0

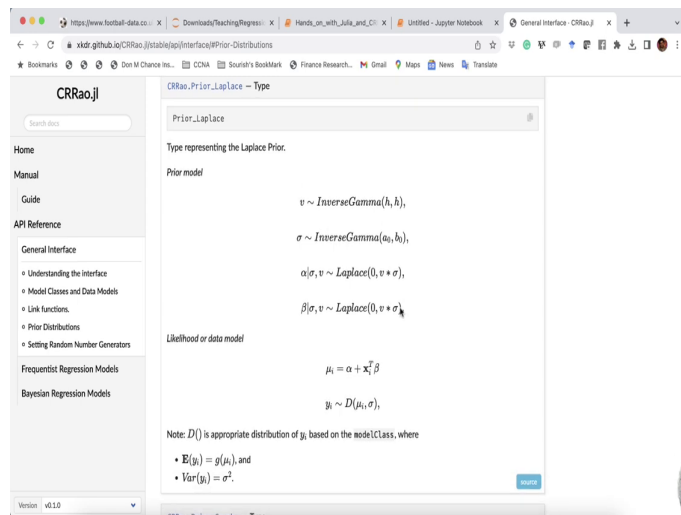
CRRao.Prior.Laplace - Type



If you want to see the detail of the Ridge Prior, I think you can go to the right, API References of General Interface General Interface and then Prior Distribution. So, the first thing you can say prior Gauss and then here is the ridge prior and all the definition of the ridge prior.

The way it is written, we have taken a appropriate distribution means Poisson, then you just take the Poisson and automatically it will do the rest of the thing. So, this is the ridge prior, the way it is defined in this in CRRao.

(Refer Slide Time: 16:48)



The screenshot shows the CRRao.jl documentation page for the `Prior.Laplace` type. The page is titled "CRRao.jl" and "Prior.Laplace - Type". It includes a search bar and a navigation menu on the left with sections like Home, Manual, Guide, API Reference, and Bayesian Regression Models. The main content area displays the following information:

Type representing the Laplace Prior.

Prior model

$$v \sim \text{InverseGamma}(h, h),$$
$$\sigma \sim \text{InverseGamma}(a_0, h_0),$$
$$\alpha|\sigma, v \sim \text{Laplace}(0, v * \sigma),$$
$$\beta|\sigma, v \sim \text{Laplace}(0, v * \sigma)$$


Likelihood or data model

$$\mu_i = \alpha + x_i^T \beta$$
$$y_i \sim D(\mu_i, \sigma)$$

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

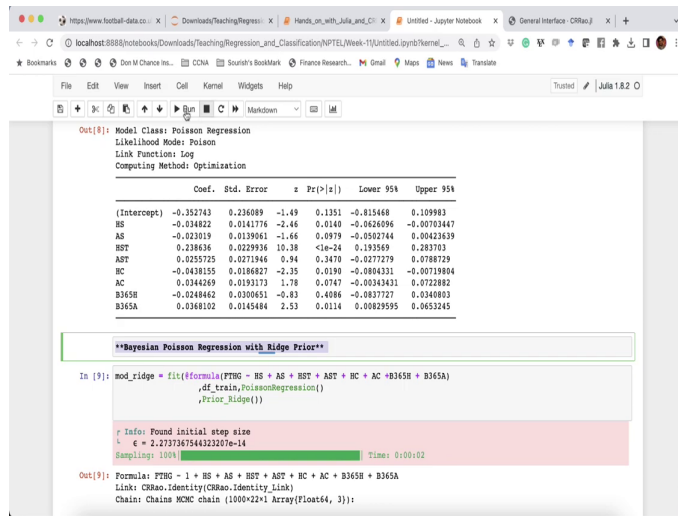
- $\mathbb{E}(y_i) = g(\mu_i)$ , and
- $\text{Var}(y_i) = \sigma^2$ .

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



Next, we can try Laplace priors on the beta, Laplace distribution being imposed and inverse gamma prior is being imposed on the Poisson regression.

(Refer Slide Time: 17:08)



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

Out[8]: Model Class: Poisson Regression  
Likelihood Mode: Poisson  
Link Function: Log  
Computing Method: Optimization

	Coef.	Std. Error	z	Pr(> z )	Lower 95%	Upper 95%
(Intercept)	-0.352743	0.236089	-1.49	0.1351	-0.815468	0.109983
HS	-0.034822	0.014174	-2.46	0.0140	-0.062696	-0.0070347
AS	-0.022019	0.013961	-1.66	0.0979	-0.050744	0.00623639
HST	0.238636	0.022936	10.38	<1e-24	0.193669	0.283703
AST	0.0255725	0.0271946	0.94	0.3470	-0.0277279	0.0788729
HC	-0.0438155	0.018627	-2.35	0.0190	-0.0804331	-0.00719804
AC	0.0344569	0.0193173	1.78	0.0747	-0.00249431	0.0722882
B365B	-0.0248462	0.0300651	-0.83	0.4086	-0.0837727	0.0340803
B365A	0.0368102	0.0145484	2.53	0.0114	0.00929595	0.0653245

**\*\*Bayesian Poisson Regression with Ridge Prior\*\***

```
In [9]: mod_ridge = fit(formula(FBBO ~ HS + AS + HST + AST + HC + AC + B365B + B365A),  
                    ,df_train,PoissonRegression(),  
                    ,Prior_Ridge())
```

+ Info: Found initial step size  
t  $\epsilon = 2.27326754432207e-14$   
Sampling: 1000 | Time: 0:00:02

Out[9]: Formula: FBBO ~ 1 + HS + AS + HST + AST + HC + AC + B365B + B365A  
Link: CRao.Identity(CRao.Identity\_Link)  
Chain: Chains MCMC chain (1000\*22\*1 Array(Float64, 3)):



(Refer Slide Time: 17:13)

```
beta[4] 0.1749 0.2076 0.2236 0.2400 0.2691
beta[5] -0.0389 -0.0032 0.0163 0.0330 0.0655
beta[6] -0.0859 -0.0617 -0.0492 -0.0365 -0.0154
beta[7] -0.0093 0.0162 0.0285 0.0396 0.0644
beta[8] -0.0941 -0.0590 -0.0416 -0.0221 0.0157
beta[9] 0.0023 0.0202 0.0286 0.0369 0.0563
```

**Bayesian Poisson Regression with Laplace Prior**

```
In [11]: mod_laplace = fit(@formula{PTHC ~ BS + AS + BST + AST + BC + AC + B365H + B365A},
                        df_train, PoissonRegression(),
                        Prior_Laplace())
r Info: Found initial step size
      ε = 7.888609952210118e-32
Sampling: 100
Time: 0:00:03
```

```
Out[11]: Formula: PTHC ~ 1 + BS + AS + BST + AST + BC + AC + B365H + B365A
Link: CRRAo.Identity(CRRAo.Identity_Link)
Chain: Chains MCMC chain (1000*22+1 Array{Float64, 3}):

Iterations = 5011:1500
Number of chains = 1
Samples per chain = 1000
Wall duration = 0.22 seconds
Compute duration = 0.22 seconds
parameters = 4, [β[1], β[2], β[3], β[4], β[5], β[6], β[7], β[8], β[9]]
internals = 1p, n_steps, is_accept, acceptance_rate, log_density, hamiltonian_energy, hamiltonian_energy_error
r, max_hamiltonian_energy_error, tree_depth, numerical_error, step_size, non_step_size

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

So, if I want Poisson regression with Poisson regression with Laplace priors, you can plot very easily using CRRao. So, Laplace Prior, all you have to do Laplace Prior, opsee, sorry, I have to do a mark down, ok. Now, it is fine. So, Bayesian Regression with Laplace Prior so, all you have to do is just copy this guy.

So, now on the coefficient, we are applying Laplace Prior, Laplace Prior distribution. On the coefficient, this is typically. So, you just change the name of the prior Laplace and it should and run. And CRRao will understand automatically that, ok, this is Laplace Prior that the user want to apply and boom, it runs and it gives you the all the estimates. So, by default, it will simulate 1000 samples after 500 burning. So, from 501 to 1500 iteration is being reported, that are all close to 1.

(Refer Slide Time: 18:24)

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

```
internals = lp, n_steps, is_accept, acceptance_rate, log_density, hamiltonian_energy, hamiltonian_energy_error
r, max_hamiltonian_energy_error, tree_depth, numerical_error, step_size, non_step_size

Summary Statistics
parameters      mean      std      mcmc      ess_bulk      ess_tail      rhat      e -
Symbol Float64  Float64  Float64  Float64      Float64  Float64  -
λ      0.0786  0.0337  0.0013  724.3468  725.0954  1.0020  -
β[1]  -0.0682  0.1186  0.0060  547.0632  340.0748  1.0023  -
β[2]  -0.0371  0.0135  0.0006  560.2454  454.6632  1.0035  -
β[3]  -0.0241  0.0127  0.0005  713.8269  548.4412  0.9994  -
β[4]  0.2213  0.0235  0.0010  585.1491  632.0193  1.0015  -
β[5]  0.0121  0.0228  0.0008  750.4048  714.8428  1.0028  -
β[6]  -0.0442  0.0185  0.0008  589.7102  573.6921  1.0035  -
β[7]  0.0229  0.0180  0.0007  670.7255  664.5972  0.9997  -
β[8]  -0.0352  0.0267  0.0009  628.5700  559.5652  0.9994  -
β[9]  0.0251  0.0138  0.0005  796.4113  750.2832  1.0017  -
1 column omitted

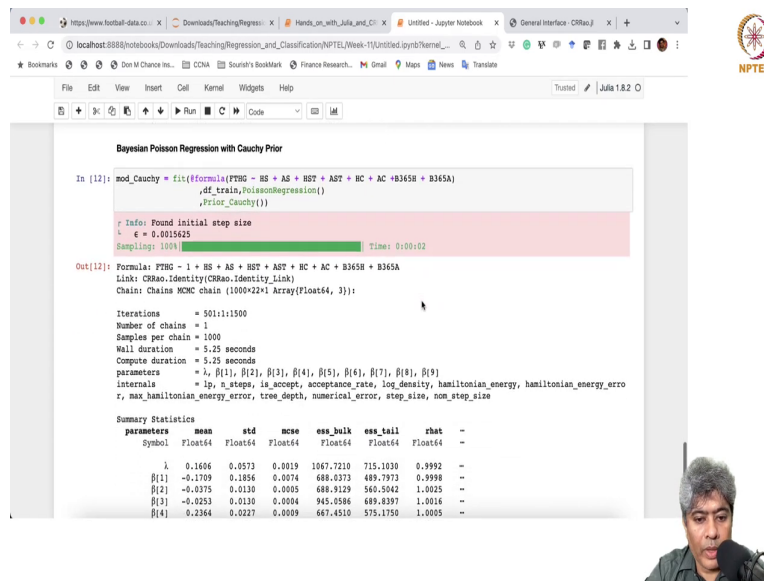
Quantiles
parameters      2.5%      25.0%      50.0%      75.0%      97.5%
Symbol Float64  Float64  Float64  Float64  Float64
λ      0.0369  0.0554  0.0704  0.0921  0.1668
β[1]  -0.3835  -0.1179  -0.0391  0.0055  0.1107
β[2]  -0.0634  -0.0458  -0.0374  -0.0277  -0.0102
β[3]  -0.0402  -0.0216  -0.0239  -0.0156  0.0005
β[4]  0.1886  0.2152  0.2302  0.2479  0.2783
β[5]  -0.0308  -0.0039  0.0126  0.0272  0.0589
β[6]  -0.0802  -0.0571  -0.0438  -0.0308  -0.0098
β[7]  -0.0146  0.0101  0.0234  0.0346  0.0581
β[8]  -0.0889  -0.0531  -0.0342  -0.0159  0.0135
β[9]  -0.0013  0.0156  0.0253  0.0345  0.0504
```

So, this is a good news; that means, the chain has converged and you can see similar kind of things behavior you can see, ok. So, this second coefficient plus H is effective, this is strong effective, the fourth coefficient, both coefficient was my 1, 2, 3, if HST, HST is strongly have strong effect. And with the last one here, it is saying that the odds does not have any effect, the odds does not have any effect.

The 6th one is, I think it was corner, I think it was corner 1, 2, 3, 4, 5, 6, yeah, corner does have had effect yeah. So, Laplace is saying that in the MLE, we got 3 in MLE method, we it got 3 at a 5 percent level, we can say home shot, number of shot taken, number of shot on target and the bet365s odd for awaiting. But it was surprising that I would not expect that it will awaiting, so, it will have no effect on number of goals code and the Bayesian methods is actually rejecting this method.

I mean, saying though ridge prices, yes, it did indeed, but Laplace Prior is saying no, no, we are not. So, this is important that you apply different kind of models and then you compare the results. And then next we can try Cauchy Prior, ok.

(Refer Slide Time: 20:23)



```

Bayesian Poisson Regression with Cauchy Prior

In [12]: mod_cauchy = fit(formula(FRIG ~ HS + AS + HST + AST + HC + AC + B365H + B365A)
,df_train,PoissonRegression()
,prior_cauchy())

r Info: Found initial step size
l ̵ = 0.0015625
Sampling: 1000 Time: 0:00:02

Out[12]: Formula: FRIG ~ 1 + HS + AS + HST + AST + HC + AC + B365H + B365A
Link: CRANO.Identity(CRANO.Identity_Link)
Chain: Chains MCMC chain (1000x22x4 Array{Float64, 3}):

Iterations = 5011:1500
Number of chains = 1
Samples per chain = 1000
Wall duration = 5.25 seconds
Compute duration = 5.25 seconds
parameters = λ, β[1], β[2], β[3], β[4], β[5], β[6], β[7], β[8], β[9]
internals = lp, n_steps, is_accept, acceptance_rate, log_density, hamiltonian_energy, hamiltonian_energy_error
r, 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 0.1606 0.0573 0.0019 1067.7210 715.1030 0.9992 -
β[1] -0.1709 0.1856 0.0074 688.0373 489.7973 0.9998 -
β[2] -0.0375 0.0130 0.0009 688.9129 560.5042 1.0025 -
β[3] -0.0253 0.0130 0.0004 945.9286 689.9397 1.0016 -
β[4] 0.2364 0.0227 0.0009 667.4510 575.1750 1.0005 -

```

So, again, very simple fitting Cauchy Prior, first we will give a name, first we will put it as a mark down and then say instead of Laplace, we see first Cauchy prior and then what we are going to do is simply plot, just copy this guy. Cauchy prior and instead of Laplace, we just let C and then tab, it will fill it up by itself and then run. So, while, ok, it just, it was very fast, it was very fast. So, and it is indeed all converged as usual; the rhat is all close to 1.

(Refer Slide Time: 21:17)

```
r, max_hamiltonian_energy_error, tree_depth, numerical_error, step_size, non_step_size
```

Summary Statistics						
parameters	mean	std	mcase	esp_bulk	esp_tail	rhat
Symbol	Float64	Float64	Float64	Float64	Float64	Float64
$\lambda$	0.1606	0.0573	0.0019	1067.7210	715.1030	0.9992
$\beta[1]$	-0.1709	0.1856	0.0074	686.0373	489.7973	0.9998
$\beta[2]$	-0.0215	0.0130	0.0005	688.9129	560.5042	1.0025
$\beta[3]$	-0.0253	0.0130	0.0004	945.0586	689.8397	1.0016
$\beta[4]$	0.2364	0.0227	0.0009	667.4510	575.1750	1.0005
$\beta[5]$	0.0206	0.0254	0.0008	909.0332	852.1355	0.9995
$\beta[6]$	-0.0453	0.0185	0.0006	948.7694	694.8675	0.9998
$\beta[7]$	0.0301	0.0191	0.0006	937.8106	658.4386	0.9993
$\beta[8]$	-0.0351	0.0282	0.0009	1032.2565	716.1674	1.0021
$\beta[9]$	0.0308	0.0141	0.0004	1110.7356	781.6497	1.0018

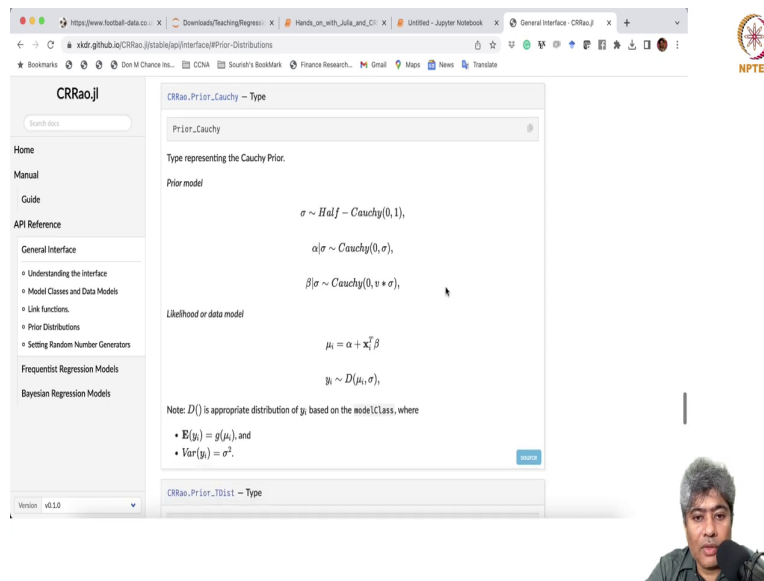
Quantiles					
parameters	2.5%	25.0%	50.0%	75.0%	97.5%
Symbol	Float64	Float64	Float64	Float64	Float64
$\lambda$	0.0857	0.1209	0.1484	0.1868	0.3057
$\beta[1]$	-0.6035	-0.2743	-0.1442	-0.0379	0.1277
$\beta[2]$	-0.0644	-0.0463	-0.0374	-0.0282	-0.0113
$\beta[3]$	-0.0511	-0.0335	-0.0258	-0.0167	0.0011
$\beta[4]$	0.1933	0.2207	0.2386	0.2512	0.2810
$\beta[5]$	-0.0313	0.0037	0.0216	0.0382	0.0681
$\beta[6]$	-0.0814	-0.0569	-0.0460	-0.0340	-0.0071
$\beta[7]$	-0.0071	0.0171	0.0301	0.0432	0.0681
$\beta[8]$	-0.0097	-0.0540	-0.0359	-0.0153	0.0186
$\beta[9]$	0.0027	0.0213	0.0309	0.0404	0.0589

So, the this one has a effect, strong effect, this one has a strong effect, the entire 95 percent confidence bill does not include 0 and the last one also has a strong effect, this does not include 0, 6th one, the corner also has a effect.

So, corner number of shots, number of shots on target, number of shots, number of shots on target, number of corners, these are going to have an effect, the model is again and again saying and the odds of the Away Team by Bet365 seems like having an effect, not maybe strong or something, but it seems like it does have an effect.



(Refer Slide Time: 22:04)



The screenshot shows a web browser displaying the CRRao.jl documentation. The main content area is titled "CRRao.Prior.Cauchy - Type" and describes the "Prior.Cauchy" type. It states: "Type representing the Cauchy Prior." and "Prior model". The model is defined by the following equations:

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

The likelihood or data model is given by:

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

A note states: "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 NPTEL logo is visible in the top right corner of the browser window. A small video feed of a person is visible in the bottom right corner of the slide.

So, what is Cauchy prior? Let us see it will Cauchy prior in the everything of the model is same, but on the beta and alpha, on the coefficient we put Cauchy prior with a scale parameter have a half Cauchy distribution. So, this is very robust prior and next what we will try, we will try T-distributed prior, ok.

(Refer Slide Time: 22:27)

CRRao.jl

Search docs

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

Bayesian Regression Models

Version v0.1.0

CRRao.Prior\_TDist - Type

Prior\_TDist

Type representing the T-Distributed Prior.

Prior model

$$v \sim \text{InverseGamma}(h, h),$$
$$\sigma \sim \text{InverseGamma}(a_0, h_0),$$
$$\alpha | \sigma, v \sim \sigma t(v),$$
$$\beta | \sigma, v \sim \sigma t(v),$$



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 `modelClass`, where

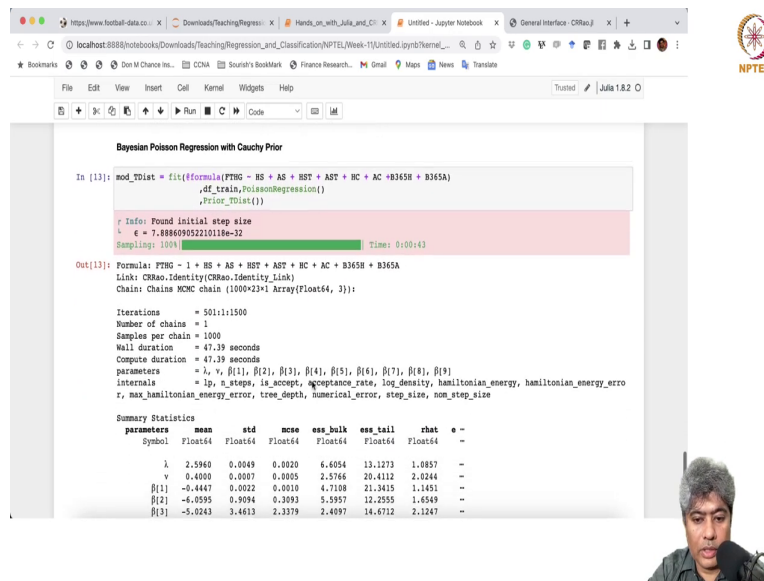
- $\mathbb{E}(y_i) = g(\mu_i)$ , and
- $\text{Var}(y_i) = \sigma^2$ .
- The  $t(v)$  is  $t$  distribution with  $v$  degrees of freedom.

source



So, here we are going to scaled T-distribution, distributed prior on the beta and alpha and on the scale, we implement it InverseGamma prior. So, very variety of prior options we have in the CRRao.

(Refer Slide Time: 22:58)



```
Bayesian Poisson Regression with Cauchy Prior

In [13]: mod_TDist = fit({formula:FPHC ~ BS + AS + BST + AST + HC + AC + B365H + B365A},
                        df_train,PoissonRegression(),
                        Prior_TDist())

r Info: Found initial step size
      ε = 7.88860952210118e-32
      Sampling: 1001 Time: 0:00:43

Out[13]: Formula: FPHC ~ 1 + BS + AS + BST + AST + HC + AC + B365H + B365A
Link: CR Rao Identity(CR Rao Identity Link)
Chain: Chains MCMC chain (1000*23)* Array{Float64, 3}:

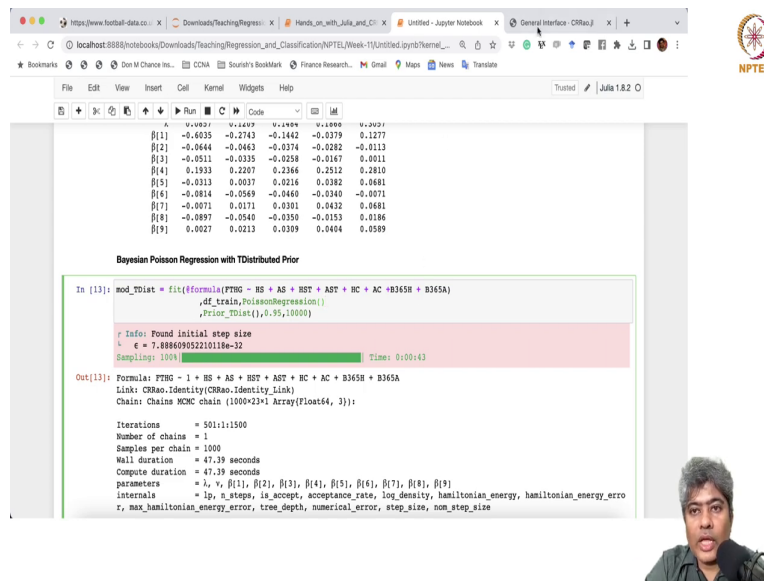
Iterations = 5011:1500
Number of chains = 1
Samples per chain = 1000
Wall duration = 47.39 seconds
Compute duration = 47.39 seconds
parameters = λ, v, β[1], β[2], β[3], β[4], β[5], β[6], β[7], β[8], β[9]
Intervals = lp, n_steps, ik_accept, acceptance_rate, log_density, hamiltonian_energy, hamiltonian_energy_error
r, max_hamiltonian_energy_error, tree_depth, numerical_error, step_size, non_step_size

Summary Statistics
parameters mean std mcmc ess_bulk ess_tail rhat e -
Symbol Float64 Float64 Float64 Float64 Float64 Float64
λ 2.5969 0.0049 0.0020 6.4054 13.1273 1.0857 -
v 0.4000 0.0007 0.0005 2.5766 20.4112 2.0244 -
β[1] -0.4447 0.0022 0.0010 4.7108 21.3415 1.1451 -
β[2] -6.0595 0.9094 0.3093 5.5957 12.2555 1.6549 -
β[3] -5.0243 3.4613 2.3379 2.4097 14.0712 2.1247 -
```

So, what we are going to do next is T-distributed prior, we are going to implement the T-distributed prior, ok. T-Distributed, ok. Just you write T and press tab and that should be fine. It is very fast looks like, (Refer Time: 23:41) sometimes it is it is bit slow, but the T-distributed might take a little bit time, ok.

Let us see r you see T-distributed prior dies not did not able to converge you can see the r hat r 2 near 2. So, that means, definitely these the algorithm quantum integral did not converge by the 1000 samples, we need to increase the number of samples. So, and you saw it is slow.

(Refer Slide Time: 24:50)



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

```
β[1] -0.6035 -0.2743 -0.1442 -0.0379 0.1277
β[2] -0.0644 -0.0463 -0.0374 -0.0282 -0.0113
β[3] -0.0511 -0.0235 -0.0358 -0.0167 0.0011
β[4] 0.1933 0.2207 0.2366 0.2512 0.2810
β[5] -0.0313 0.0037 0.0216 0.0382 0.0681
β[6] -0.0814 -0.0569 -0.0480 -0.0340 -0.0071
β[7] -0.0071 0.0171 0.0301 0.0432 0.0681
β[8] -0.0897 -0.0540 -0.0350 -0.0153 0.0186
β[9] 0.0027 0.0213 0.0309 0.0404 0.0589
```

**Bayesian Poisson Regression with T-Distributed Prior**

```
In [12]: mod_TDist = fit(Formula(FYBD ~ BS + AS + BST + AST + RC + AC + B365H + B365A)
                        ,df_train,PoissonRegression()
                        ,Prior_TDist(),0.95,10000)

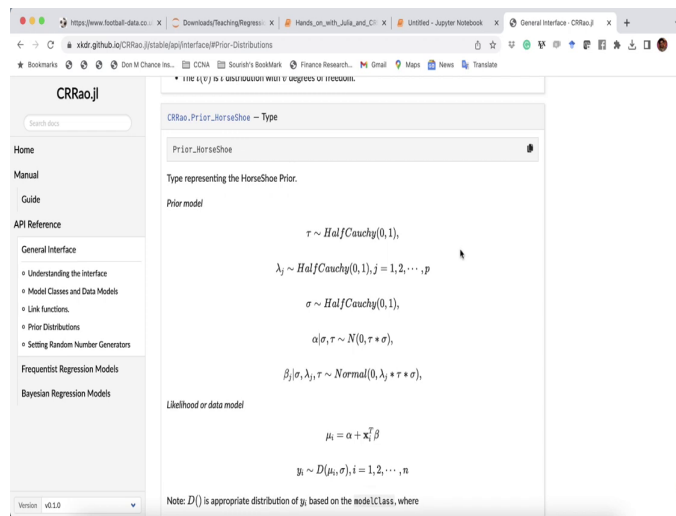
r INFO: Found initial step size
l ε = 7.8886095221018e-32
Sampling: 1001 Time: 0:00:43

Out[12]: Formula: FYBD ~ 1 + BS + AS + BST + AST + RC + AC + B365H + B365A
Link: CRao.Identity(CRao.Identity_Link)
Chain: Chains MCMC chain (1000×23×1 Array{Float64, 3}):

Iterations = 5011:1500
Number of chains = 1
Samples per chain = 1000
Wall duration = 47.39 seconds
Compute duration = 47.39 seconds
parameters = λ, ν, β[1], β[2], β[3], β[4], β[5], β[6], β[7], β[8], β[9]
internals = lp, n_steps, is_accept, acceptance_rate, log_density, hamiltonian_energy, hamiltonian_energy_error
r, max_hamiltonian_energy_error, tree_depth, numerical_error, step_size, non_step_size
```

So, I am not going to I am not going to again increase the samples and run it because it will just take time. It will, all you have to do just increase the sample. I can show you all you have to do just probably you just paid 0.95 and then you have to instead of 1000 you have to just set 10,000 and run it. But I am not going to do that because it will just take longer time. So, you try it yourself, but I am going to try another prior, which is very popular prior called HorseShoe Prior, ok.

(Refer Slide Time: 25:23)



The screenshot shows the CRRao.jl documentation page for the `Prior_HorseShoe` model. The page is titled "CRRao.Prior\_HorseShoe - Type" and includes a search bar and a navigation menu on the left. The main content area displays the following information:

Type representing the HorseShoe Prior.

Prior model

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

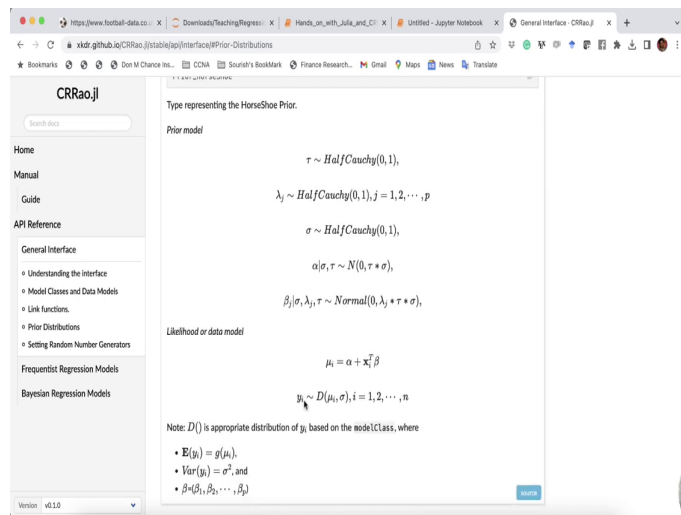
Likelihood or data model

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

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



(Refer Slide Time: 25:32)



The screenshot shows the CRRao.jl documentation page. The main content area displays the following model definition:

Type representing the HorseShoe Prior.

Prior model


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

Likelihood or data model

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

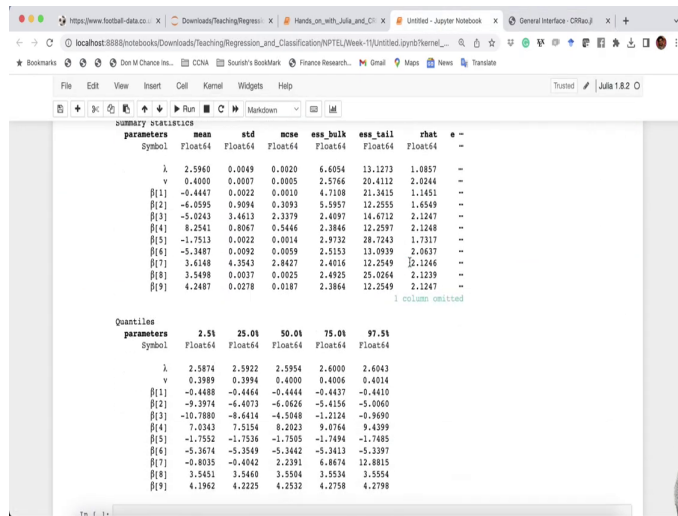
Note:  $D()$  is appropriate distribution of  $y_i$  based on the `modelClass`, where

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



So, let me try HorseShoe Prior and HorseShoe Prior is let me show you how the HorseShoe Prior works. So, this is the T Dist y follow proper distributed typically in natural exponential family. In the our case it will be Poisson with mu i and. So, mu i will be alpha plus x i transpose beta. And on the beta and alpha there will be a conditional normal distribution and on the scale parameters of the conditional normal distributions we will have half cauch, HalfCauchy distribution.

(Refer Slide Time: 26:11)



```
summary statistics
parameters
  Symbol      mean      std      mse      ess_bulk  ess_tail  rhat  e -
  Float64     Float64  Float64  Float64  Float64   Float64  Float64
λ            2.5960    0.0049   0.0020    6.6054   13.1273   1.0857  -
γ            0.4000    0.0007   0.0005    2.5766   20.4112   2.0244  -
β[1]        -0.4447   0.0022   0.0010    4.7109   21.3415   1.1451  -
β[2]        -6.0595   0.9094   0.3993    5.3957   12.2555   1.6549  -
β[3]        -5.0243   3.4613   2.3379    2.4097   14.6712   2.1247  -
β[4]         8.2541   0.8067   0.5446    2.3846   12.2597   2.1248  -
β[5]        -1.7513   0.0022   0.0014    2.9732   28.7243   1.7317  -
β[6]        -5.3487   0.0092   0.0059    2.5153   13.0939   2.0037  -
β[7]         3.6148   4.3543   2.8427    2.4016   12.2549   2.1246  -
β[8]         3.5498   0.0037   0.0025    2.4925   25.0264   2.1239  -
β[9]         4.2487   0.0278   0.0187    2.3864   12.2549   2.1247  -

Quantiles
parameters
  Symbol      2.5%      25.0%     50.0%     75.0%     97.5%
  Float64     Float64  Float64  Float64  Float64
λ            2.5974    2.5922    2.5954    2.6000    2.6043
γ            0.3989    0.3994    0.4000    0.4006    0.4014
β[1]        -0.4488   -0.4444   -0.4444   -0.4437   -0.4410
β[2]        -9.3974   -6.4073   -6.0626   -5.4156   -5.0060
β[3]       -10.7880   -8.6418   -4.5048   -1.2124   -0.9690
β[4]         7.0243    7.5154    8.0023    8.4764    8.4999
β[5]        -1.7552   -1.7536   -1.7505   -1.7494   -1.7485
β[6]        -5.3674   -5.3549   -5.3442   -5.3413   -5.3397
β[7]        -0.8035   -0.4042    2.2381    6.8674   12.0815
β[8]         3.5451    3.5460    3.5504    3.5534    3.5554
β[9]         4.1962    4.2225    4.2532    4.2758    4.2798
```







(Refer Slide Time: 27:15)

```
Compute duration = 13.84 seconds
parameters      =  $\tau, \lambda[1], \lambda[2], \lambda[3], \lambda[4], \lambda[5], \lambda[6], \lambda[7], \lambda[8], \lambda[9], \beta[1], \beta[2], \beta[3], \beta[4], \beta[5], \beta[6], \beta[7], \beta[8], \beta[9]$ 
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
Symbol  mean  std  mcmc  ess_bulk  ess_tail  rhat  e -
Float64 Float64 Float64 Float64 Float64 Float64 Float64
 $\tau$       0.0773  0.0517  0.0026  291.2524  304.2228  1.0015  -
 $\lambda[1]$  3.2023  8.2151  0.4274  155.3011  120.2487  0.9998  -
 $\lambda[2]$  1.3887  2.2846  0.1133  311.6991  190.5004  0.9992  -
 $\lambda[3]$  0.9755  1.8254  0.3737  413.4077  457.4641  1.0030  -
 $\lambda[4]$  5.4401  6.8286  0.4178  295.5781  210.1061  1.0010  -
 $\lambda[5]$  0.9926  3.8789  0.2017  272.4269  229.1802  0.9991  -
 $\lambda[6]$  1.5914  1.9399  0.0900  467.0118  509.0880  1.0003  -
 $\lambda[7]$  0.8090  1.0011  0.0472  317.4933  139.3713  1.0044  -
 $\lambda[8]$  1.1756  2.0286  0.0926  274.9769  223.6003  1.0008  -
 $\lambda[9]$  0.9210  1.1914  0.0519  429.7081  220.8793  0.9996  -
 $\beta[1]$  -0.1249  0.1794  0.0148  177.7063  236.5974  1.0027  -
 $\beta[2]$  -0.0331  0.0151  0.0009  278.8325  283.5533  1.0049  -
 $\beta[3]$  -0.0180  0.0122  0.0007  329.3381  563.0966  1.0025  -
 $\beta[4]$  0.2208  0.0224  0.0011  457.9989  390.7896  0.9996  -
 $\beta[5]$  0.0033  0.0002  0.0008  583.7845  639.4779  1.0026  -
 $\beta[6]$  -0.0451  0.0197  0.0011  348.4827  412.7304  1.0004  -
 $\beta[7]$  0.0154  0.0177  0.0009  438.9326  492.9998  1.0000  -
 $\beta[8]$  -0.0262  0.0262  0.0016  266.0719  443.9529  1.0005  -
 $\beta[9]$  0.0235  0.0144  0.0008  301.8840  423.2496  1.0117  -
1 column omitted

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



(Refer Slide Time: 27:21)

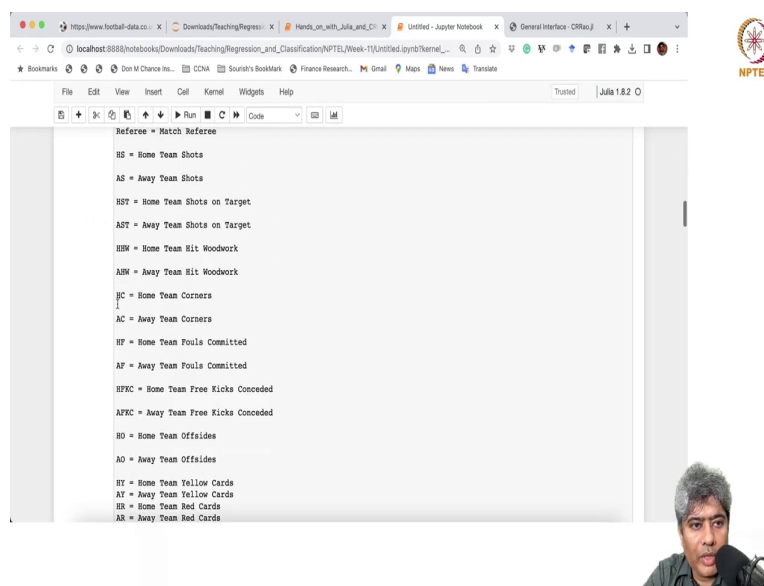
Quantiles	2.5%	25%	50%	75%	97.5%
parameters	Float64	Float64	Float64	Float64	Float64
Symbol					
$\tau$	0.0172	0.0416	0.0631	0.0998	0.2146
$\lambda[1]$	0.0577	0.5966	1.3918	3.2378	14.3843
$\lambda[2]$	0.0611	0.4343	0.8139	1.4733	7.3089
$\lambda[3]$	0.0424	0.2465	0.3958	1.0436	5.3096
$\lambda[4]$	0.7673	1.9693	3.5732	6.0910	23.1610
$\lambda[5]$	0.0354	0.2142	0.4726	0.8703	4.1831
$\lambda[6]$	0.1471	0.5318	0.9718	1.8006	7.7041
$\lambda[7]$	0.0331	0.2316	0.5113	1.0350	3.1699
$\lambda[8]$	0.0395	0.3087	0.6993	1.4311	4.9062
$\lambda[9]$	0.0561	0.3184	0.6120	1.0565	3.8878
$\beta[1]$	-0.6010	-0.2231	-0.0449	-0.0010	0.0811
$\beta[2]$	-0.0645	-0.0437	-0.0332	-0.0225	-0.0021
$\beta[3]$	-0.0439	-0.0258	-0.0177	-0.0089	0.0018
$\beta[4]$	0.1812	0.2146	0.2313	0.2473	0.2768
$\beta[5]$	-0.0353	-0.0075	0.0024	0.0139	0.0485
$\beta[6]$	-0.0815	-0.0590	-0.0454	-0.0310	-0.0064
$\beta[7]$	-0.0133	0.0025	0.0123	0.0272	0.0535
$\beta[8]$	-0.0868	-0.0437	-0.0230	-0.0046	0.0144
$\beta[9]$	-0.0019	0.0136	0.0238	0.0335	0.0538

Yes, you see it did converge. It did converge and the way it has been Horseshoe prior works is for each beta  $i$  there will be a local scale distributed. So, now you can see that. So, this is so; that means, for each beta 1 there will be a lambda 1 beta 2 there will lambda 2 and beta 9 there will lambda 9. And then these are the local shrinkage parameter and tau is the global shrinkage parameter.

So, all these parameters makes interesting local shrinkage and global shrinkage make the Horseshoe prior very very attractive to the Bayesian community. And what happens is what we see let us this is the intercept then next is the HS the home team shot on target. We see this is fully negative all the entire 95 percent confidence interval. Then this is number of shots on target is completely positive and this is number of corners completely negative.

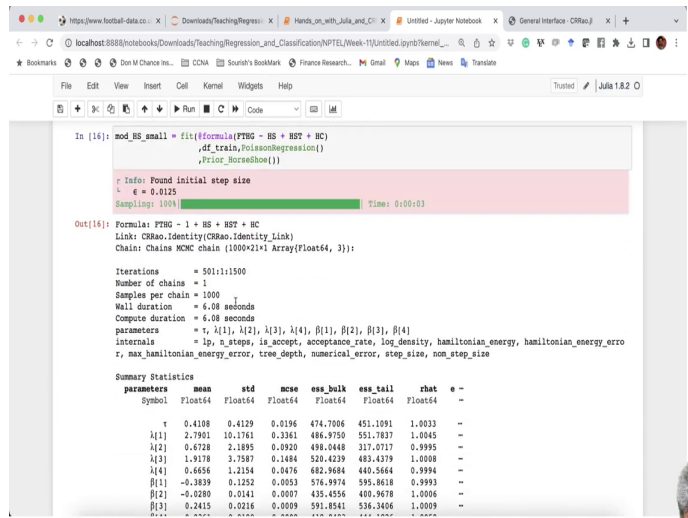
If you get too many corners your number of goals score will be less somehow corner has to do with the you know negative something to do with the negative. And what we are seeing that away teams and the Bet365 does not have the odds does not have an effect by HorseShoe prior.

(Refer Slide Time: 29:06)



So, what we are seeing according to the model of. So, we can safely say HS which is home teams home team home team shot HST home team shots on target. HC HC that number of corners by home team. These are the three are the sort of sure for sure kind of model.

(Refer Slide Time: 29:32)



```
In [16]: mod_RS_small = fit((Formula{PTBG - RS + HST + HC}
                           ,df_train,PoissonRegression()
                           ,Prior_HorserShoe()))
r Info: Found initial step size
ε = 0.0125
Sampling: 1000 | Time: 0:00:03

Out[16]: Formula: PTBG ~ 1 + RS + HST + HC
Link: CBMO.Identity{CBMO.Identity{Link}}
Chain: Chains MCMC chain (1000*2*1 Array{Float64, 3}):

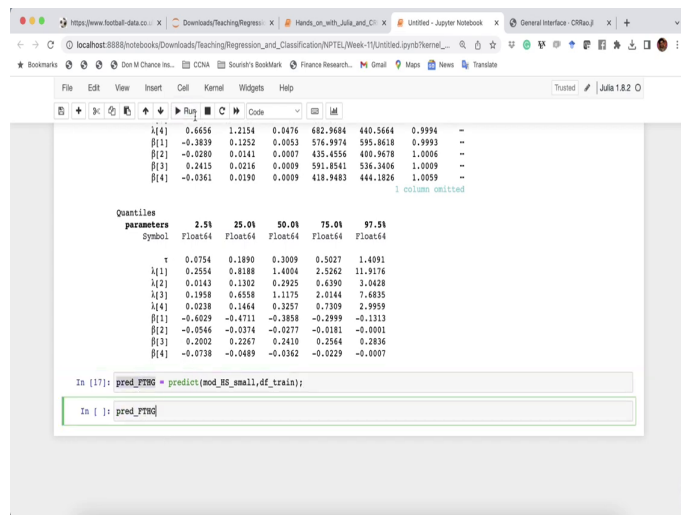
Iterations = 5011:1500
Number of chains = 1
Samples per chain = 1000
Wall duration = 6.08 seconds
Compile duration = 6.08 seconds
Parameters = τ, λ[1], λ[2], λ[3], λ[4], β[1], β[2], β[3], β[4]
Internals = lp, n_steps, is_accept, acceptance_rate, log_density, hamiltonian_energy, hamiltonian_energy_error
r_max_hamiltonian_energy_error, tree_depth, numerical_error, step_size, non_step_size

Summary Statistics
parameters mean std mcmc ess_bulk ess_tail rhat e -
Symbol Float64 Float64 Float64 Float64 Float64 Float64
τ 0.4188 0.4129 0.0196 474.7006 451.1091 1.0033 -
λ[1] 2.7901 10.1761 0.3361 486.9750 551.7837 1.0045 -
λ[2] 0.6728 2.1895 0.8920 498.0448 317.0717 0.9995 -
λ[3] 1.9178 3.7587 0.1464 528.4239 482.4379 1.0008 -
λ[4] 0.6656 1.2154 0.0476 682.9684 440.5664 0.9994 -
β[1] -0.3839 0.1252 0.0053 576.9974 595.8618 0.9993 -
β[2] -0.0280 0.0141 0.0007 435.4556 408.9678 1.0006 -
β[3] 0.2415 0.0216 0.0009 591.8541 536.3406 1.0009 -
```



So, we can what we will try; we will try a slightly smaller model let us try a smaller model and home just home team shot on target home team corner number of, ok. So, that is how so, now, we have a shorter model and let us run this.

(Refer Slide Time: 29:56)



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

```
λ[4] 0.6656 1.2154 0.0476 682.9684 440.5664 0.9994 --  
β[1] -0.3839 0.1252 0.0053 576.3974 595.8628 0.9993 --  
β[2] -0.0280 0.0141 0.0007 435.4556 400.9678 1.0006 --  
β[3] 0.2415 0.0216 0.0089 591.8541 536.3406 1.0009 --  
β[4] -0.0361 0.0190 0.0089 418.9483 444.1826 1.0059 --  
                                     1 column omitted
```

Quantiles					
parameters	2.5%	25.0%	50.0%	75.0%	97.5%
Symbol	Float64	Float64	Float64	Float64	Float64
1	0.0754	0.1890	0.3009	0.5027	1.4091
λ[1]	0.2554	0.8088	1.4004	2.5262	11.9176
λ[2]	0.0143	0.1302	0.2925	0.6390	3.0428
λ[3]	0.1958	0.6558	1.1175	2.0144	7.6835
λ[4]	0.0238	0.1464	0.3257	0.7309	2.9959
β[1]	-0.4009	-0.4711	-0.3858	-0.2899	-0.1313
β[2]	-0.0546	-0.0374	-0.0277	-0.0181	-0.0001
β[3]	0.2002	0.2267	0.2410	0.2564	0.2836
β[4]	-0.0738	-0.0489	-0.0362	-0.0229	-0.0007

```
In [17]: pred_FTHG = predict(mod_RS_small, df_train);  
In [ ]: pred_FTHG
```



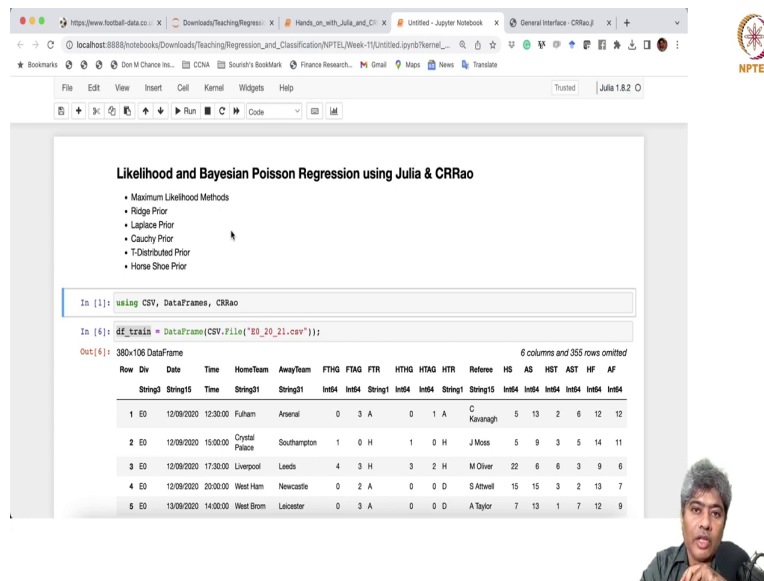
So, it will be faster equals faster yeah. So, we can see that each of these parameters were wet fast. And now what I am going to do we are going to just say predict if I just say predict you can just do use these models to predict also. And then you give the model and give the data set name. So, the data set name is say here I am using train you can have a test data also. If you just give say predict FTHG right FTHG.

(Refer Slide Time: 30:53)

```
Out[18]: 380-element Vector {Float64}:
 0.8957182574839925
 0.9574768708831988
 1.1398670124726489
 0.6953063721198229
 0.6651267795751596
 1.4835082859236188
 0.8477689365662795
 0.565363348925784
 1.5544271939290937
 2.342248689611296
 0.9033746016493
 0.903037716119328
 2.245741655117256
 ⋮
 1.0923664792474326
 0.9761119731621007
 0.986389290331586
 1.1007284326081283
 0.3876021530119148
 2.8000512376646998
 1.8567243614402185
 0.815098918893326
 4.2056832640004695
 0.7559916495958403
 2.330116803449872
 0.9761119731621007
```

And if you can see so, you can see that they are giving you the prediction also. So, this is one of the advantage of CRRao we do not have to when I mean you can fit any models and you can get the prediction also right away using just all the predict functions and it will be done. So, this is how you implement Poisson regression using you know Bayesian prior. So, Poisson regression both likelihood and the methods and the Bayesian implementation of the Poisson regression with Julia.

(Refer Slide Time: 31:34)



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

- Likelihood and Bayesian Poisson Regression using Julia & CRRao**
- Maximum Likelihood Methods
  - Ridge Prior
  - Laplace Prior
  - Cauchy Prior
  - T Distributed Prior
  - Horse Shoe Prior

```
In [1]: using CSV, DataFrames, CRRao
```

```
In [6]: df_train = DataFrame(CSV.File("E9_20_21.csv"));
```

```
Out[6]: 380x108 DataFrame (6 columns and 355 rows omitted)
```

Row	Div	Date	Time	HomeTeam	AwayTeam	FTHG	FTAG	FTR	HTHG	HTAG	HTR	Referee	HS	AS	HST	AST	HF	AF
String3	String15	String15	String15	String21	String21	Int64	Int64	String1	Int64	String1	String15	String15	Int64	Int64	Int64	Int64	Int64	Int64
1	ED	12/09/2020	12:30:00	Fulham	Arsenal	0	3	A	0	1	A	C Kavanagh	5	13	2	6	12	12
2	ED	12/09/2020	15:00:00	Crystal Palace	Southampton	1	0	H	1	0	H	J Moss	5	9	3	5	14	11
3	ED	12/09/2020	17:30:00	Liverpool	Leeds	4	3	H	3	2	H	M Oliver	22	6	6	3	9	6
4	ED	12/09/2020	20:00:00	West Ham	Newcastle	0	2	A	0	0	D	S Atwell	15	15	3	2	13	7
5	ED	13/09/2020	14:00:00	West Brom	Leicester	0	3	A	0	0	D	A Taylor	7	13	1	7	12	9

So, let me just write Likelihood and Bayesian Poisson Regression using Julia the number of implementation that we have done. Likelihood Maximum Likelihood Maximum Likelihood Methods so, actually we use Julia and CRRao ok. And then we used Ridge Prior, Bayesian Ridge Prior then we used Laplace Prior, Laplace Prior, then we used Cauchy Prior, Cauchy Prior, then we used T Distributed Prior, but it was not very successful it did not converge nicely.

We had to run more samples, T Distributed Prior and finally, HorseShoe Prior, HorseShoe Prior. So, all these we implemented very nicely and using CRRao in Julia. So, I hope you enjoyed this video and we will continue this in the next video. This next video we will be doing negative binomial regression.

Thank you very much.