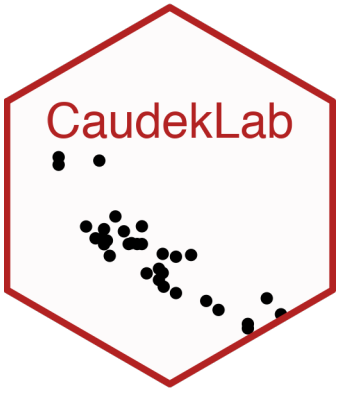
Psicometria

Corrado Caudek

2022-01-16

Questo documento è stato realizzato con:

- LaTeX e la classe memoir (http://www.ctan.org/pkg/memoir);
- R (http://www.r-project.org/) e RStudio (http://www.rstudio.com/);
- bookdown (http://bookdown.org/) e memoiR (https://ericmarcon.github.io/memoiR/).



Nel blog della mia pagina personale sono forniti alcuni approfondimenti degli argomenti qui trattati. https://ccaudek.github.io/caudeklab/

Indice

Data della versione presente: Gennaio 16, 2022.

Prefazione

Data Science per psicologi contiene il materiale delle lezioni dell'insegnamento di *Psicometria B000286* (A.A. 2021/2022) rivolto agli studenti del primo anno del Corso di Laurea in Scienze e Tecniche Psicologiche dell'Università degli Studi di Firenze. *Psicometria* si propone di fornire agli studenti un'introduzione all'analisi dei dati in psicologia. Le conoscenze/competenze che verranno sviluppate in questo insegnamento sono quelle della Data science, ovvero un insieme di conoscenze/competenze che si pongono all'intersezione tra statistica (ovvero, richiedono la capacità di comprendere teoremi statistici) e informatica (ovvero, richiedono la capacità di sapere utilizzare un software).

La psicologia e la Data science

> *It's worth noting, before getting started, that this material is hard. If you find yourself confused at any point, you are normal. Any sense of confusion you feel is just your brain correctly calibrating to the subject matter. Over time, confusion is replaced by comprehension […]*
>
> — Richard McElreath

Sembra sensato spendere due parole su un tema che è importante per gli studenti: quello indicato dal titolo di questo Capitolo. È ovvio che agli studenti di psicologia la statistica non piace. Se piacesse, forse studierebbero Data science e non psicologia; ma non lo fanno. Di conseguenza, gli studenti di psicologia si chiedono: "perché dobbiamo perdere tanto tempo a studiare queste cose quando in realtà quello che ci interessa è tutt'altro?" Questa è una bella domanda.

C'è una ragione molto semplice che dovrebbe farci capire perché la Data science è così importante per la psicologia. Infatti, a ben pensarci, la psicologia è una disciplina intrinsecamente statistica, se per statistica intendiamo quella disciplina che studia la variazione delle caratteristiche degli individui nella popolazione. La psicologia studia *gli individui* ed è proprio la variabilità inter- e intra-individuale ciò che vogliamo descrivere e, in certi casi, predire. In questo senso, la psicologia è molto diversa dall'ingegneria, per esempio. Le proprietà di un determinato ponte sotto certe condizioni, ad esempio, sono molto simili a quelle di un altro ponte, sotto le medesime condizioni. Quindi, per un ingegnere la statistica è poco importante: le proprietà dei materiali sono unicamente dipendenti dalla loro composizione e restano costanti. Ma lo stesso non può dirsi degli individui: ogni individuo è unico e cambia nel tempo. E le variazioni tra gli individui, e di un individuo nel tempo, sono l'oggetto di studio proprio della psicologia: è dunque chiaro che i problemi che la psicologia si pone sono molto diversi da quelli affrontati, per esempio, dagli ingegneri. Questa è la ragione per cui abbiamo tanto bisogno della Data science in psicologia: perché la Data science ci consente di descrivere la variazione e il cambiamento. E queste sono appunto le caratteristiche di base dei fenomeni psicologici.

Sono sicuro che, leggendo queste righe, a molti studenti sarà venuta in mente la seguente domanda: perché non chiediamo a qualche esperto di fare il "lavoro sporco" (ovvero le analisi statistiche) per noi, mentre noi (gli psicologi) ci occupiamo solo di ciò che ci interessa, ovvero dei problemi psicologici slegati dai dettagli "tecnici" della Data

science? La risposta a questa domanda è che non è possibile progettare uno studio psicologico sensato senza avere almeno una comprensione rudimentale della Data science. Le tematiche della Data science non possono essere ignorate né dai ricercatori in psicologia né da coloro che svolgono la professione di psicologo al di fuori dell'Università. Infatti, anche i professionisti al di fuori dall'università non possono fare a meno di leggere la letteratura psicologica più recente: il continuo aggiornamento delle conoscenze è infatti richiesto dalla deontologia della professione. Ma per potere fare questo è necessario conoscere un bel po' di Data science! Basta aprire a caso una rivista specialistica di psicologia per rendersi conto di quanto ciò sia vero: gli articoli che riportano i risultati delle ricerche psicologiche sono zeppi di analisi statistiche e di modelli formali. E la comprensione della letteratura psicologica rappresenta un requisito minimo nel bagaglio professionale dello psicologo.

Le considerazioni precedenti cercano di chiarire il seguente punto: la Data science non è qualcosa da studiare a malincuore, in un singolo insegnamento universitario, per poi poterla tranquillamente dimenticare. Nel bene e nel male, gli psicologi usano gli strumenti della Data science in tantissimi ambiti della loro attività professionale: in particolare quando costruiscono, somministrano e interpretano i test psicometrici. È dunque chiaro che possedere delle solide basi di Data science è un tassello imprescindibile del bagaglio professionale dello psicologo. In questo insegnamento verrano trattati i temi base della Data science e verrà adottato un punto di vista bayesiano, che corrisponde all'approccio più recente e sempre più diffuso in psicologia.

Come studiare

> *I know quite certainly that I myself have no special talent. Curiosity, obsession and dogged endurance, combined with self-criticism, have brought me to my ideas.*
>
> — Albert Einstein

Il giusto metodo di studio per prepararsi all'esame di Psicometria è quello di seguire attivamente le lezioni, assimilare i concetti via via che essi vengono presentati e verificare in autonomia le procedure presentate a lezione. Incoraggio gli studenti a farmi domande per chiarire ciò che non è stato capito appieno. Incoraggio gli studenti a utilizzare i forum attivi su Moodle e, soprattutto, a svolgere gli esercizi proposti su Moodle. I problemi forniti su Moodle rappresentano il livello di difficoltà richiesto per superare l'esame e consentono allo studente di comprendere se le competenze sviluppate fino a quel punto sono sufficienti rispetto alle richieste dell'esame.

La prima fase dello studio, che è sicuramente individuale, è quella in cui è necessario acquisire le conoscenze teoriche relative ai problemi che saranno presentati all'esame. La seconda fase di studio, che può essere facilitata da scambi con altri e da incontri di gruppo, porta ad acquisire la capacità di applicare le conoscenze: è necessario capire come usare un software (R) per applicare i concetti statistici alla specifica situazione del problema che si vuole risolvere. Le due fasi non sono però separate: il saper fare molto spesso ci aiuta a capire meglio.

Sviluppare un metodo di studio efficace

> *Memorization is not learning.*
>
> — Richard Phillips Feynman

Avendo insegnato molte volte in passato un corso introduttivo di analisi dei dati ho notato nel corso degli anni che gli studenti con l'atteggiamento mentale che descriverò qui sotto generalmente ottengono ottimi risultati. Alcuni studenti sviluppano naturalmente questo approccio allo studio, ma altri hanno bisogno di fare uno sforzo per maturarlo.

Fornisco qui sotto una breve descrizione del "metodo di studio" che, nella mia esperienza, è il più efficace per affrontare le richieste di questo insegnamento (Burger & Starbird, 2012).

- Dedicate un tempo sufficiente al materiale di base, apparentemente facile; assicuratevi di averlo capito bene. Cercate le lacune nella vostra comprensione. Leggere presentazioni diverse dello stesso materiale (in libri o articoli diversi) può fornire nuove intuizioni.
- Gli errori che facciamo sono i nostri migliori maestri. Istintivamente cerchiamo di dimenticare subito i nostri errori. Ma il miglior modo di imparare è apprendere dagli errori che commettiamo. In questo senso, una soluzione corretta è meno utile di una soluzione sbagliata. Quando commettiamo un errore questo ci fornisce un'informazione importante: ci fa capire qual è il materiale di studio sul quale dobbiamo ritornare e che dobbiamo capire meglio.
- C'è ovviamente un aspetto "psicologico" nello studio. Quando un esercizio o problema ci sembra incomprensibile, la cosa migliore da fare è dire: "mi arrendo", "non ho idea di cosa fare!". Questo ci rilassa: ci siamo già arresi, quindi non abbiamo niente da perdere, non dobbiamo più preoccuparci. Ma non dobbiamo fermarci qui. Le cose "migliori" che faccio (se ci sono) le faccio quando non ho voglia di lavorare. Alle volte, quando c'è qualcosa che non so fare e non ho idea di come affontare, mi dico: "oggi non ho proprio voglia di fare fatica", non ho voglia di mettermi nello stato mentale per cui "in 10 minuti devo risolvere il problema perché dopo devo fare altre cose". Però ho voglia di *divertirmi* con quel problema e allora mi dedico a qualche aspetto "marginale" del problema, che so come affrontare, oppure considero l'aspetto più difficile del problema, quello che non so come risolvere, ma invece di cercare di risolverlo, guardo come altre persone hanno affrontato problemi simili, opppure lo stesso problema in un altro contesto. Non mi pongo l'obiettivo "risolvi il problema in 10 minuti", ma invece quello di farmi un'idea "generale" del problema, o quello di capire un caso più specifico e più semplice del problema. Senza nessuna pressione. Infatti, in quel momento ho deciso di non lavorare (ovvero, di non fare fatica). Va benissimo se "parto per la tangente", ovvero se mi metto a leggere del materiale che sembra avere poco a che fare con il problema centrale (le nostre intuizioni e la nostra curiosità solitamente ci indirizzano sulla strada giusta). Quando faccio così, molto spesso trovo la soluzione del problema che mi ero posto e, paradossalmente, la trovo in un tempo minore di quello che, in precedenza, avevo dedicato a "lavorare" al problema. Allora perché non faccio sempre così? C'è ovviamente l'aspetto dei "10 minuti" che non è sempre facile da dimenticare. Sotto pressione, possiamo solo agire in maniera automatica, ovvero possiamo solo applicare qualcosa che già sappiamo fare. Ma se dobbiamo imparare qualcosa di nuovo, la pressione è un impedimento.
- È utile farsi da soli delle domande sugli argomenti trattati, senza limitarsi a cercare di risolvere gli esercizi che vengono assegnati. Quando studio qualcosa mi viene in mente: "se questo è vero, allora deve succedere quest'altra cosa". Allora verifico se questo è vero, di solito con una simulazione. Se i risultati della simulazione sono quelli che mi aspetto, allora vuol dire che ho capito. Se i risultati sono diversi da quelli che mi aspettavo, allora mi rendo conto di non avere capito e ritorno indietro a studiare con più attenzione la teoria che pensavo di avere capito – e ovviamente mi rendo conto che c'era un aspetto che avevo frainteso. Questo tipo di verifica è qualcosa che dobbiamo fare da soli, in prima persona: nessun altro può fare questo al posto nostro.
- Non aspettatevi di capire tutto la prima volta che incontrate un argomento nuovo.[1] È utile farsi una nota mentalmente delle lacune nella vostra comprensione e tornare su di esse in seguito per carcare di colmarle. L'atteggiamento naturale, quando

[1] Ricordatevi inoltre che gli individui tendono a sottostimare la propria capacità di apprendere (Horn & Loewenstein, 2021).

non capiamo i dettagli di qualcosa, è quello di pensare: “non importa, ho capito in maniera approssimativa questo punto, non devo preoccuparmi del resto”. Ma in realtà non è vero: se la nostra comprensione è superficiale, quando il problema verrà presentato in una nuova forma, non riusciremo a risolverlo. Per cui i dubbi che ci vengono quando studiamo qualcosa sono il nostro alleato più prezioso: ci dicono esattamente quali sono gli aspetti che dobbiamo approfondire per potere migliorare la nostra preparazione.

- È utile sviluppare una visione d’insieme degli argomenti trattati, capire l’obiettivo generale che si vuole raggiungere e avere chiaro il contributo che i vari pezzi di informazione forniscono al raggiungimento di tale obiettivo. Questa organizzazione mentale del materiale di studio facilita la comprensione. È estremamente utile creare degli schemi di ciò che si sta studiando. Non aspettate che sia io a fornirvi un riepilogo di ciò che dovete imparare: sviluppate da soli tali schemi e tali riassunti.
- Tutti noi dobbiamo imparare l’arte di trovare le informazioni, non solo nel caso di questo insegnamento. Quando vi trovate di fronte a qualcosa che non capite, o ottenete un oscuro messaggio di errore da un software, ricordatevi: “Google is your friend”.

Corrado Caudek

Capitolo 1

Adattare il modello lineare ai dati

Per fare un esempio concreto useremo un famoso dataset chiamato `kidiq` (Gelman et al., 2020) che riporta i dati di un'indagine del 2007 su un campione di donne americane adulte e sui loro bambini di età compres tra i 3 e i 4 anni. I dati sono costituiti da 434 osservazioni e 4 variabili:

- `kid_score`: QI del bambino; è il punteggio totale del *Peabody Individual Achievement Test* (PIAT) costituito dalla somma dei punteggi di tre sottoscale (Mathematics, Reading comprehension, Reading recognition);
- `mom_hs`: variabile dicotomica (0 or 1) che indica se la madre del bambino ha completato le scuole superiori (1) oppure no (0);
- `mom_iq`: QI della madre;
- `mom_age`: età della madre.

```
library("rio")
df <- rio::import(here("data", "kidiq.dta"))
head(df)
#>   kid_score mom_hs mom_iq mom_work mom_age
#> 1        65      1  121.1        4      27
#> 2        98      1   89.4        4      25
#> 3        85      1  115.4        4      27
#> 4        83      1   99.4        3      25
#> 5       115      1   92.7        4      27
#> 6        98      0  107.9        1      18
```

```
stancode <- 'data {real y_mean;} parameters {real y;} model {y ~ normal(y_mean,1);}'
mod <- stan_model(model_code = stancode, verbose = TRUE)
#>
#> TRANSLATING MODEL '16a540c60860868616528e4524def24d9' FROM Stan CODE TO C++ CODE NOW.
#> successful in parsing the Stan model '16a540c60860868616528e4524def24d9'.
#> OS: x86_64, darwin17.0; rstan: 2.21.3; Rcpp: 1.0.8; inline: 0.3.19
#>  >> setting environment variables:
#> PKG_LIBS =  '/Library/Frameworks/R.framework/Versions/4.1/Resources/library/rstan/lib//libStanServices
#> PKG_CPPFLAGS =   -I"/Library/Frameworks/R.framework/Versions/4.1/Resources/library/Rcpp/include/"  -I"
#>  >> Program source :
#>
#>    1 :
#>    2 : // includes from the plugin
#>    3 : // [[Rcpp::plugins(cpp14)]]
#>    4 :
```

```
#>    5 :
#>    6 : // user includes
#>    7 : #include <Rcpp.h>
#>    8 : #include <rstan/io/rlist_ref_var_context.hpp>
#>    9 : #include <rstan/io/r_ostream.hpp>
#>   10 : #include <rstan/stan_args.hpp>
#>   11 : #include <boost/integer/integer_log2.hpp>
#>   12 : // Code generated by Stan version 2.21.0
#>   13 :
#>   14 : #include <stan/model/model_header.hpp>
#>   15 :
#>   16 : namespace model109501d49f077_16a540c6086086816528e4524def24d9_namespace {
#>   17 :
#>   18 : using std::istream;
#>   19 : using std::string;
#>   20 : using std::stringstream;
#>   21 : using std::vector;
#>   22 : using stan::io::dump;
#>   23 : using stan::math::lgamma;
#>   24 : using stan::model::prob_grad;
#>   25 : using namespace stan::math;
#>   26 :
#>   27 : static int current_statement_begin__;
#>   28 :
#>   29 : stan::io::program_reader prog_reader__() {
#>   30 :     stan::io::program_reader reader;
#>   31 :     reader.add_event(0, 0, "start", "model109501d49f077_16a540c6086086816528e4524def24d9");
#>   32 :     reader.add_event(3, 1, "end", "model109501d49f077_16a540c6086086816528e4524def24d9");
#>   33 :     return reader;
#>   34 : }
#>   35 :
#>   36 : class model109501d49f077_16a540c6086086816528e4524def24d9
#>   37 :   : public stan::model::model_base_crtp<model109501d49f077_16a540c6086086816528e4524def24d9> {
#>   38 : private:
#>   39 :         double y_mean;
#>   40 : public:
#>   41 :     model109501d49f077_16a540c6086086816528e4524def24d9(rstan::io::rlist_ref_var_context& conte
#>   42 :         std::ostream* pstream__ = 0)
#>   43 :         : model_base_crtp(0) {
#>   44 :         ctor_body(context__, 0, pstream__);
#>   45 :     }
#>   46 :
#>   47 :     model109501d49f077_16a540c6086086816528e4524def24d9(stan::io::var_context& context__,
#>   48 :         unsigned int random_seed__,
#>   49 :         std::ostream* pstream__ = 0)
#>   50 :         : model_base_crtp(0) {
#>   51 :         ctor_body(context__, random_seed__, pstream__);
#>   52 :     }
#>   53 :
#>   54 :     void ctor_body(stan::io::var_context& context__,
#>   55 :                    unsigned int random_seed__,
#>   56 :                    std::ostream* pstream__) {
#>   57 :         typedef double local_scalar_t__;
#>   58 :
#>   59 :         boost::ecuyer1988 base_rng__ =
```

```
#>   60 :           stan::services::util::create_rng(random_seed__, 0);
#>   61 :         (void) base_rng__;  // suppress unused var warning
#>   62 :
#>   63 :         current_statement_begin__ = -1;
#>   64 :
#>   65 :         static const char* function__ = "model109501d49f077_16a540c6086086816528e4524def24d9_na
#>   66 :         (void) function__;  // dummy to suppress unused var warning
#>   67 :         size_t pos__;
#>   68 :         (void) pos__;  // dummy to suppress unused var warning
#>   69 :         std::vector<int> vals_i__;
#>   70 :         std::vector<double> vals_r__;
#>   71 :         local_scalar_t__ DUMMY_VAR__(std::numeric_limits<double>::quiet_NaN());
#>   72 :         (void) DUMMY_VAR__;  // suppress unused var warning
#>   73 :
#>   74 :         try {
#>   75 :             // initialize data block variables from context__
#>   76 :             current_statement_begin__ = 1;
#>   77 :             context__.validate_dims("data initialization", "y_mean", "double", context__.to_vec
#>   78 :             y_mean = double(0);
#>   79 :             vals_r__ = context__.vals_r("y_mean");
#>   80 :             pos__ = 0;
#>   81 :             y_mean = vals_r__[pos__++];
#>   82 :
#>   83 :
#>   84 :             // initialize transformed data variables
#>   85 :             // execute transformed data statements
#>   86 :
#>   87 :             // validate transformed data
#>   88 :
#>   89 :             // validate, set parameter ranges
#>   90 :             num_params_r__ = 0U;
#>   91 :             param_ranges_i__.clear();
#>   92 :             current_statement_begin__ = 1;
#>   93 :             num_params_r__ += 1;
#>   94 :         } catch (const std::exception& e) {
#>   95 :             stan::lang::rethrow_located(e, current_statement_begin__, prog_reader__());
#>   96 :             // Next line prevents compiler griping about no return
#>   97 :             throw std::runtime_error("*** IF YOU SEE THIS, PLEASE REPORT A BUG ***");
#>   98 :         }
#>   99 :     }
#>  100 :
#>  101 :     ~model109501d49f077_16a540c6086086816528e4524def24d9() { }
#>  102 :
#>  103 :
#>  104 :     void transform_inits(const stan::io::var_context& context__,
#>  105 :                          std::vector<int>& params_i__,
#>  106 :                          std::vector<double>& params_r__,
#>  107 :                          std::ostream* pstream__) const {
#>  108 :         typedef double local_scalar_t__;
#>  109 :         stan::io::writer<double> writer__(params_r__, params_i__);
#>  110 :         size_t pos__;
#>  111 :         (void) pos__; // dummy call to supress warning
#>  112 :         std::vector<double> vals_r__;
#>  113 :         std::vector<int> vals_i__;
#>  114 :
```

```
#>  115 :          current_statement_begin__ = 1;
#>  116 :          if (!(context__.contains_r("y")))
#>  117 :              stan::lang::rethrow_located(std::runtime_error(std::string("Variable y missing")),
#>  118 :          vals_r__ = context__.vals_r("y");
#>  119 :          pos__ = 0U;
#>  120 :          context__.validate_dims("parameter initialization", "y", "double", context__.to_vec());
#>  121 :          double y(0);
#>  122 :          y = vals_r__[pos__++];
#>  123 :          try {
#>  124 :              writer__.scalar_unconstrain(y);
#>  125 :          } catch (const std::exception& e) {
#>  126 :              stan::lang::rethrow_located(std::runtime_error(std::string("Error transforming var
#>  127 :          }
#>  128 :
#>  129 :          params_r__ = writer__.data_r();
#>  130 :          params_i__ = writer__.data_i();
#>  131 :      }
#>  132 :
#>  133 :      void transform_inits(const stan::io::var_context& context,
#>  134 :                           Eigen::Matrix<double, Eigen::Dynamic, 1>& params_r,
#>  135 :                           std::ostream* pstream__) const {
#>  136 :        std::vector<double> params_r_vec;
#>  137 :        std::vector<int> params_i_vec;
#>  138 :        transform_inits(context, params_i_vec, params_r_vec, pstream__);
#>  139 :        params_r.resize(params_r_vec.size());
#>  140 :        for (int i = 0; i < params_r.size(); ++i)
#>  141 :          params_r(i) = params_r_vec[i];
#>  142 :      }
#>  143 :
#>  144 :
#>  145 :      template <bool propto__, bool jacobian__, typename T__>
#>  146 :      T__ log_prob(std::vector<T__>& params_r__,
#>  147 :                   std::vector<int>& params_i__,
#>  148 :                   std::ostream* pstream__ = 0) const {
#>  149 :
#>  150 :          typedef T__ local_scalar_t__;
#>  151 :
#>  152 :          local_scalar_t__ DUMMY_VAR__(std::numeric_limits<double>::quiet_NaN());
#>  153 :          (void) DUMMY_VAR__;  // dummy to suppress unused var warning
#>  154 :
#>  155 :          T__ lp__(0.0);
#>  156 :          stan::math::accumulator<T__> lp_accum__;
#>  157 :          try {
#>  158 :              stan::io::reader<local_scalar_t__> in__(params_r__, params_i__);
#>  159 :
#>  160 :              // model parameters
#>  161 :              current_statement_begin__ = 1;
#>  162 :              local_scalar_t__ y;
#>  163 :              (void) y;  // dummy to suppress unused var warning
#>  164 :              if (jacobian__)
#>  165 :                  y = in__.scalar_constrain(lp__);
#>  166 :              else
#>  167 :                  y = in__.scalar_constrain();
#>  168 :
#>  169 :              // model body
```

```
#>  170 :
#>  171 :             current_statement_begin__ = 1;
#>  172 :             lp_accum__.add(normal_log<propto__>(y, y_mean, 1));
#>  173 :
#>  174 :         } catch (const std::exception& e) {
#>  175 :             stan::lang::rethrow_located(e, current_statement_begin__, prog_reader__());
#>  176 :             // Next line prevents compiler griping about no return
#>  177 :             throw std::runtime_error("*** IF YOU SEE THIS, PLEASE REPORT A BUG ***");
#>  178 :         }
#>  179 :
#>  180 :         lp_accum__.add(lp__);
#>  181 :         return lp_accum__.sum();
#>  182 :
#>  183 :     } // log_prob()
#>  184 :
#>  185 :     template <bool propto, bool jacobian, typename T_>
#>  186 :     T_ log_prob(Eigen::Matrix<T_,Eigen::Dynamic,1>& params_r,
#>  187 :                 std::ostream* pstream = 0) const {
#>  188 :       std::vector<T_> vec_params_r;
#>  189 :       vec_params_r.reserve(params_r.size());
#>  190 :       for (int i = 0; i < params_r.size(); ++i)
#>  191 :         vec_params_r.push_back(params_r(i));
#>  192 :       std::vector<int> vec_params_i;
#>  193 :       return log_prob<propto,jacobian,T_>(vec_params_r, vec_params_i, pstream);
#>  194 :     }
#>  195 :
#>  196 :
#>  197 :     void get_param_names(std::vector<std::string>& names__) const {
#>  198 :         names__.resize(0);
#>  199 :         names__.push_back("y");
#>  200 :     }
#>  201 :
#>  202 :
#>  203 :     void get_dims(std::vector<std::vector<size_t> >& dimss__) const {
#>  204 :         dimss__.resize(0);
#>  205 :         std::vector<size_t> dims__;
#>  206 :         dims__.resize(0);
#>  207 :         dimss__.push_back(dims__);
#>  208 :     }
#>  209 :
#>  210 :     template <typename RNG>
#>  211 :     void write_array(RNG& base_rng__,
#>  212 :                      std::vector<double>& params_r__,
#>  213 :                      std::vector<int>& params_i__,
#>  214 :                      std::vector<double>& vars__,
#>  215 :                      bool include_tparams__ = true,
#>  216 :                      bool include_gqs__ = true,
#>  217 :                      std::ostream* pstream__ = 0) const {
#>  218 :         typedef double local_scalar_t__;
#>  219 :
#>  220 :         vars__.resize(0);
#>  221 :         stan::io::reader<local_scalar_t__> in__(params_r__, params_i__);
#>  222 :         static const char* function__ = "model109501d49f077_16a540c6086086816528e4524def24d9_n
#>  223 :         (void) function__;  // dummy to suppress unused var warning
#>  224 :
```

```
#>  225 :         // read-transform, write parameters
#>  226 :         double y = in__.scalar_constrain();
#>  227 :         vars__.push_back(y);
#>  228 :
#>  229 :         double lp__ = 0.0;
#>  230 :         (void) lp__;  // dummy to suppress unused var warning
#>  231 :         stan::math::accumulator<double> lp_accum__;
#>  232 :
#>  233 :         local_scalar_t__ DUMMY_VAR__(std::numeric_limits<double>::quiet_NaN());
#>  234 :         (void) DUMMY_VAR__;  // suppress unused var warning
#>  235 :
#>  236 :         if (!include_tparams__ && !include_gqs__) return;
#>  237 :
#>  238 :         try {
#>  239 :             if (!include_gqs__ && !include_tparams__) return;
#>  240 :             if (!include_gqs__) return;
#>  241 :         } catch (const std::exception& e) {
#>  242 :             stan::lang::rethrow_located(e, current_statement_begin__, prog_reader__());
#>  243 :             // Next line prevents compiler griping about no return
#>  244 :             throw std::runtime_error("*** IF YOU SEE THIS, PLEASE REPORT A BUG ***");
#>  245 :         }
#>  246 :     }
#>  247 :
#>  248 :     template <typename RNG>
#>  249 :     void write_array(RNG& base_rng,
#>  250 :                      Eigen::Matrix<double,Eigen::Dynamic,1>& params_r,
#>  251 :                      Eigen::Matrix<double,Eigen::Dynamic,1>& vars,
#>  252 :                      bool include_tparams = true,
#>  253 :                      bool include_gqs = true,
#>  254 :                      std::ostream* pstream = 0) const {
#>  255 :       std::vector<double> params_r_vec(params_r.size());
#>  256 :       for (int i = 0; i < params_r.size(); ++i)
#>  257 :         params_r_vec[i] = params_r(i);
#>  258 :       std::vector<double> vars_vec;
#>  259 :       std::vector<int> params_i_vec;
#>  260 :       write_array(base_rng, params_r_vec, params_i_vec, vars_vec, include_tparams, include_gqs,
#>  261 :       vars.resize(vars_vec.size());
#>  262 :       for (int i = 0; i < vars.size(); ++i)
#>  263 :         vars(i) = vars_vec[i];
#>  264 :     }
#>  265 :
#>  266 :     std::string model_name() const {
#>  267 :         return "model109501d49f077_16a540c6086086816528e4524def24d9";
#>  268 :     }
#>  269 :
#>  270 :
#>  271 :     void constrained_param_names(std::vector<std::string>& param_names__,
#>  272 :                                  bool include_tparams__ = true,
#>  273 :                                  bool include_gqs__ = true) const {
#>  274 :         std::stringstream param_name_stream__;
#>  275 :         param_name_stream__.str(std::string());
#>  276 :         param_name_stream__ << "y";
#>  277 :         param_names__.push_back(param_name_stream__.str());
#>  278 :
#>  279 :         if (!include_gqs__ && !include_tparams__) return;
```

```
#>  280 :
#>  281 :         if (include_tparams__) {
#>  282 :         }
#>  283 :
#>  284 :         if (!include_gqs__) return;
#>  285 :     }
#>  286 :
#>  287 :
#>  288 :     void unconstrained_param_names(std::vector<std::string>& param_names__,
#>  289 :                                    bool include_tparams__ = true,
#>  290 :                                    bool include_gqs__ = true) const {
#>  291 :         std::stringstream param_name_stream__;
#>  292 :         param_name_stream__.str(std::string());
#>  293 :         param_name_stream__ << "y";
#>  294 :         param_names__.push_back(param_name_stream__.str());
#>  295 :
#>  296 :         if (!include_gqs__ && !include_tparams__) return;
#>  297 :
#>  298 :         if (include_tparams__) {
#>  299 :         }
#>  300 :
#>  301 :         if (!include_gqs__) return;
#>  302 :     }
#>  303 :
#>  304 : }; // model
#>  305 :
#>  306 : }  // namespace
#>  307 :
#>  308 : typedef model109501d49f077_16a540c6086086816528e4524def24d9_namespace::model109501d49f077_16a54
#>  309 :
#>  310 : #ifndef USING_R
#>  311 :
#>  312 : stan::model::model_base& new_model(
#>  313 :         stan::io::var_context& data_context,
#>  314 :         unsigned int seed,
#>  315 :         std::ostream* msg_stream) {
#>  316 :   stan_model* m = new stan_model(data_context, seed, msg_stream);
#>  317 :   return *m;
#>  318 : }
#>  319 :
#>  320 : #endif
#>  321 :
#>  322 :
#>  323 :
#>  324 : #include <rstan_next/stan_fit.hpp>
#>  325 :
#>  326 : struct stan_model_holder {
#>  327 :     stan_model_holder(rstan::io::rlist_ref_var_context rcontext,
#>  328 :                       unsigned int random_seed)
#>  329 :     : rcontext_(rcontext), random_seed_(random_seed)
#>  330 :      {
#>  331 :      }
#>  332 :
#>  333 :    //stan::math::ChainableStack ad_stack;
#>  334 :    rstan::io::rlist_ref_var_context rcontext_;
```

```
#>  335 :    unsigned int random_seed_;
#>  336 : };
#>  337 :
#>  338 : Rcpp::XPtr<stan::model::model_base> model_ptr(stan_model_holder* smh) {
#>  339 :   Rcpp::XPtr<stan::model::model_base> model_instance(new stan_model(smh->rcontext_, smh->rando
#>  340 :   return model_instance;
#>  341 : }
#>  342 :
#>  343 : Rcpp::XPtr<rstan::stan_fit_base> fit_ptr(stan_model_holder* smh) {
#>  344 :   return Rcpp::XPtr<rstan::stan_fit_base>(new rstan::stan_fit(model_ptr(smh), smh->random_seed_
#>  345 : }
#>  346 :
#>  347 : std::string model_name(stan_model_holder* smh) {
#>  348 :   return model_ptr(smh).get()->model_name();
#>  349 : }
#>  350 :
#>  351 : RCPP_MODULE(stan_fit4model109501d49f077_16a540c6086086816528e4524def24d9_mod){
#>  352 :   Rcpp::class_<stan_model_holder>("stan_fit4model109501d49f077_16a540c6086086816528e4524def24d9
#>  353 :   .constructor<rstan::io::rlist_ref_var_context, unsigned int>()
#>  354 :   .method("model_ptr", &model_ptr)
#>  355 :   .method("fit_ptr", &fit_ptr)
#>  356 :   .method("model_name", &model_name)
#>  357 :   ;
#>  358 : }
#>  359 :
#>  360 :
#>  361 : // declarations
#>  362 : extern "C" {
#>  363 : SEXP file10950377a340( ) ;
#>  364 : }
#>  365 :
#>  366 : // definition
#>  367 : SEXP file10950377a340() {
#>  368 :  return Rcpp::wrap("16a540c6086086816528e4524def24d9");
#>  369 : }
```

Per svolgere l'analisi bayesiana sistemiamo i dati nel formato appropriato per Stan:

```
data_list <- list(
  N = length(df$kid_score),
  x = df$mom_iq
)
```

```
stancode <- '
data {

  int<lower=0> N;
  vector[N] x;

}
parameters {

  real alpha;
  real beta;
  real<lower=0> sigma;
```

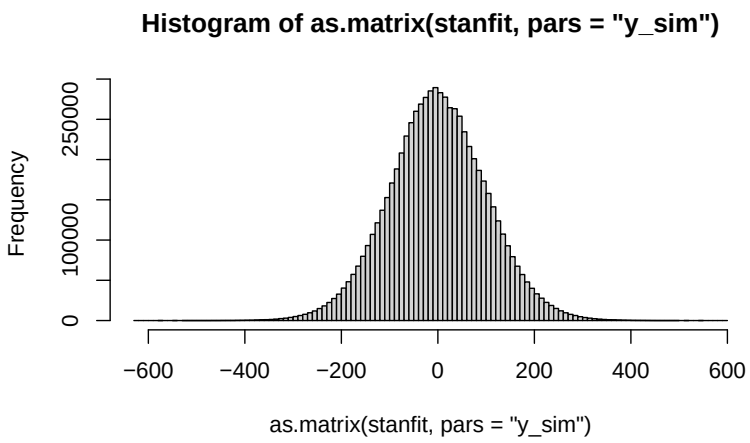
```
}
model {

  alpha ~ normal(0, 1);
  beta ~ normal(0, 1);
  sigma ~ normal(0, 1);

}
generated quantities {
  real y_sim[N];

  for (n in 1:N) {
    real mu = alpha + beta * x[n];
    y_sim[n] = normal_rng(mu, sigma);
  }
}
'
writeLines(stancode, con = "code/prior_pred_check_1.stan")
```

```
file <- file.path("code", "prior_pred_check_1.stan")
mod <- cmdstan_model(file)
```

```
fit <- mod$sample(
  data = data_list,
  iter_sampling = 4000L,
  iter_warmup = 2000L,
  seed = SEED,
  chains = 4L,
  parallel_chains = 2L,
  refresh = 0,
  thin = 1
)
```

```
stanfit <- rstan::read_stan_csv(fit$output_files())
```

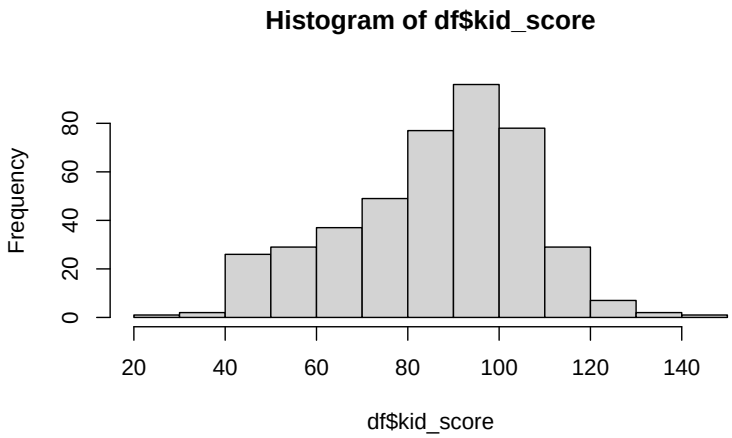
```
hist(as.matrix(stanfit, pars = "y_sim"), breaks = 100)
```



```
#>   y_sim[426] 261.814
#>   y_sim[427] 255.505
#>   y_sim[428] 165.356
#>   y_sim[429] 153.766
#>   y_sim[430] 174.088
#>   y_sim[431] 191.266
#>   y_sim[432] 195.040
#>   y_sim[433] 198.992
#>   y_sim[434] 187.239
#>   lp__        -0.618
```

```
fit$summary(c("y_sim"))
#> # A tibble: 434 × 10
#>   variable   mean median    sd   mad    q5   q95  rhat ess_bulk
#>   <chr>     <dbl>  <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>    <dbl>
#> 1 y_sim[1] -0.320 -1.27  121.  120.  -200.  200.  1.00   11732.
#> 2 y_sim[2] -0.232 -0.877  89.3  88.7 -147.  148.  1.00   11732.
#> 3 y_sim[3] -0.293 -1.42  115.  115.  -190.  190.  1.00   11734.
#> 4 y_sim[4] -0.247 -1.01   99.3  98.9 -164.  164.  1.00   11740.
#> 5 y_sim[5] -0.228 -0.969  92.6  92.1 -153.  153.  1.00   11734.
#> 6 y_sim[6] -0.264 -1.16  108.  107.  -178.  178.  1.00   11731.
#> 7 y_sim[7] -0.349 -1.39  139.  138.  -229.  229.  1.00   11731.
#> 8 y_sim[8] -0.301 -1.28  125.  124.  -206.  206.  1.00   11735.
#> # … with 426 more rows, and 1 more variable: ess_tail <dbl>
```

```
hist(df$kid_score)
```



Bibliografia

Burger, E. B., & Starbird, M. (2012). *The 5 elements of effective thinking*. Princeton University Press. (Cit. a p. ix).

Gelman, A., Hill, J., & Vehtari, A. (2020). *Regression and other stories*. Cambridge University Press. (Cit. a p. 1).

Horn, S., & Loewenstein, G. (2021). Underestimating Learning by Doing. *Available at SSRN 3941441* (cit. a p. ix).

Elenco delle figure

Abstract This document contains the material of the lessons of Psicometria B000286 (2021/2022) aimed at students of the first year of the Degree Course in Psychological Sciences and Techniques of the University of Florence, Italy.

Keywords Data science, Bayesian statistics.