Estadística Computacional Guía – Yago Tobio Souto – 2024

Indice:

Introduction to R

R packages:

- Ggplot2 → Visualization and graphics
- Dplyr & tidyr → Data transformation
- Lubridate → Date Management
- Stringr → Wrappers for string ops
- Shiny → To create user interfaces
- Mass, h20, kernlab, glmnet, caret, rpart, randomforest, gmb → ML

Advantages:

- Free, open source, state of the art, active user community

Disadvantages:

- No commercial support, easy to make mistakes without knowing, unclear relevant package choice.

Cheatsheet for code → https://www.rstudio.com/resources/cheatsheets/

Plots:

- Dotplots and lines:
  ```
  plot(x) and plot(x, type = '1')
  ```
- Scatterplots:
  ```
  plot(x,y, xlab='x-axis', ylab='y-axis', main= 'Plot X v. Y')
  ```
- Histograms:
  ```
  Hist(x, main= 'Histogram title', xlab= 'x-axis')
  ```
- Boxplots:
  ```
  Boxplot(x, main= 'boxplot title', ylab= 'y-axis')
  ```

Tools for data wrangling:

- Aggregate and rapply functions
- Tidyr and dplyr libraries

Variable types: Numeric (Continuous/Discrete), Factors (Categorical), Char(Text)
GGPlot2 Cheatsheet: https://www.maths.usyd.edu.au/u/UG/SM/STAT3022/r/current/Misc/data-visualization-2.1.pdf

Conditionals:

- if
- else if
- ifelse() ← Use this when a conditional needs to be applied to each member of a vector
- switch() ← Handy for deciding between options, never used
- && (AND) || (OR)

For further technical details, check the code guide:

For() vs. while():

For is better when the number of times to repeat is clear in advance.
While is better when you can recognise a condition to stop iterations, even if you can't guess to begin.

While is more general. Every for could be replaced with a while, but not viceversa. REMEMBER TO USE VECTORISATION WHENEVER POSSIBLE.

Why use functions?

- Data structures tie values to a single object
- Functions tie commands into one object
- Easier to understand and work with

Function structure:

- Inputs (Arguments), Body (Code that's executed), Output (Return value)

Function side effects:

- Printing something out to the console
- Plotting something on the display
- Save a data file.

What's an R environment?

- A collection of objects: Functions, Variables, etc…
- Each function has its own env.
- Names here override names in the global env.
- Internal env start with name arguments.
- Assignments inside the function only change the internal environment.

Top-down function design:

- Start with the big picture view of the task.
- Divide and conquer.
- Join them together.

Introduction to tidyverse

It's a group of libraries of functions created with R which help us stay consistent. 3 types of libraries:

- **Data wrangling**: dplyr, tidyr, readr
  - **dplyr** → Mutate dataframes, for summarising, contuing, pipes, etc…
  - **Pipe function:**
    - Single argument → **x %>% f %>% g %>%.**
    - **This the same as h(g(f(x)))**
    - Multiple argument → x %>% f(y)
    - **can be written as x %>% f(., y)**
  - **Select** → Creates a data.frame equal to subset of columns from a data.frame. *select(data.frame, columnA, columnB)*
  - **Filter using pipes:**

    ```
    year_country_gdp <- gapminder %>%
         filter(continent==“Europe”) %>%
         select(year, country, gdpPercap)
    ```
  - **Grouping data** → group_by with pipes:
    Gapminder %>% group_by(a) or group_by(a,b)
  - **Summarise** → Apply a user-defined operation to the rows of a data:
    `gapminder %>% group_by(a) %>% summarize(mean_b = mean(b))`
  - count(), n()
  - etc… Yo aquí tiraría de las diapos para las distintas funciones.
- **Iteration**: purr → Useful for mapping a function through a list of vectors.
  - **map, map_type(), map_dfr()** → type, dataframe, etc.
- **Visualization**: ggplot2

dplyr and SQL are essentially the same:

- Select is SELECT
- Filter is WHERE
- Arrange is ORDER

dplyr cheat-sheet: https://github.com/rstudio/cheatsheets/blob/main/data-transformation.pdf

You can also create pivot tables, like in Machine Learning III

Efficient Programming

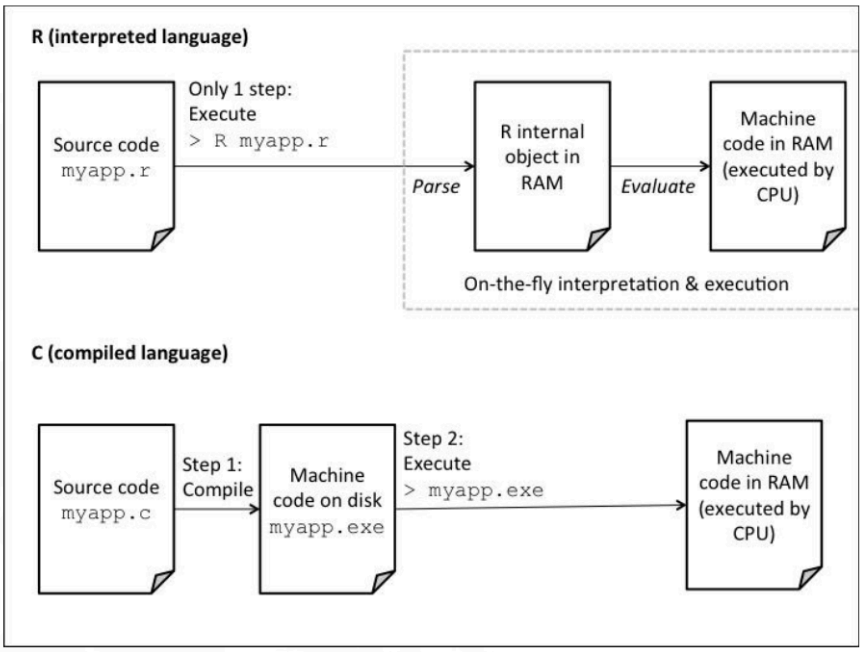
Libraries to analyse performance:

- microbenchmark → measures the computing time.
- profvis → interactive visualizations of code profile
- profmem → profile memory usage

Efficiency → Formal definition as the ratio of work W done per unit effort Q:

$\eta = \frac{W}{Q}$ → How quickly the computer can undertake a piece of code.

Low-level programming like C or C++ is faster than 'interpreted' languages like R or Python. (Which are non-native and require a compiler)



Bottlenecks on performance:

- Speed and performance of the CPU
- RAM size
- Speed at which the data reads

Characteristics for efficient coding:

- Touch typing.
- Consistent style and code conversion.
- Benchmarking and profiling
    - Benchmarking → Process of testing the performance of specific operations repeatedly. Runs lines of code to find out bottlenecks.
    - Profiling → Runs many benchmarks, for every line in a script and compares results line by line.
- Never grow vectors.
- Vectorise code when possible.
- Use factors when appropriate.
- Avoid re-computation by storing in cache.
- Byte compile packages for faster boosting.

5 efficient R set-up tips

1. System monitoring to identify bottlenecks in hardware or code.
2. R and packages up to date
3. Make use of R autocompletion and shortcuts
4. Store API keys in the .Renviron file
5. Use BLAS if R processing time is to slow.

Hay muchas maneras de hacer la función

Vectorised code:

Técnica que nos permite hacer operaciones sobre todos los elementos de un vector en una sola ejecución. Mucho más eficiente y paralelo.

```
log_sum = 0
for (i in 1:length(x))
  log_sum = log_sum + log(x[i])
```

```
log_sum = sum(log(x))
```

Cuantas menos llamadas a funciones hagas mejor. Encima queda más ordenado.

Cachear variables:

Una forma avanzada de usar el caching es usar el paquete **memoise**. Que nos permite almacenar el valor de una llamada a la función, cuando se vuelve a ejecutar.

Trade off the memory vs. speed. Por lo tanto, va más rápido pero tenemos menos memoria disponible.

Byte compile:

La librería **compiler**, permite que las funciones se ejecuten más rapido. Lo cual permitiría ahorrarse muchas operaciones.

enableJIT → Función que intenta compilar la función al ejecutarse. Permite 4 valores:

- 0 – JIT disabled.
- 1 – Compila secciones grandes previas a su primer uso.
- 2 – Compila closures pequeños.
- 3 – Top level loops también se compilan ← La que se recomienda.
-

Planificación de programación

1. Planifica a papel y boli
2. Haz un plan
3. Selecciona las librerías bien, te podría ahorrar teimpo
4. Documenta bien cada etapa de tu trabajo (Sprints)
5. Haz to workflow lo más reproducible possible → knitr.

Project Management

El enfoque de cada tipo de proyecto varía:

1. Data Analysis
   Foco: Velocidad de manipulación de datos para obtener información, la formalidad es menos importante, las interacciones entre los equipos y los analistas son vitales.

2. Package Creation
   Crear código reutilizable, focaliza la documentación, la escritura de código es importante. Testing también.
3. Reporting
4. Software Applications

Softwares para planificar y manejar los proyectos:

1. GitHub – Version Control
2. Zenhub – Para manejar sprints
3. Trello and Jira – Para Kanban boards y graficos Gantt

Cuando te propones partir tus datasets y tareas los haces por objetivos SMART.

Efficient input/output

- El formato nativo de archivos de R es .Rds. Estos archivo se pueden importar y exportar con las funciones de readRDS() y saveRDS() para almacenamiento rápido y eficiente.
- Usamos import() para obtener los datos bien, y readr para formatearlos como tabla.
- File.size() y object.size() para mantener lista del tamaño de los archivos y objetos de R.
- Una vez manipulado los datos, guardalos en formato binario, el cual ahorra tiempo. **Es de ahí de donde viene .Rds y .Rdata**

Para esto la librería más potente es **rio, y tener archivos en formato .feather**
La librería **tibble**

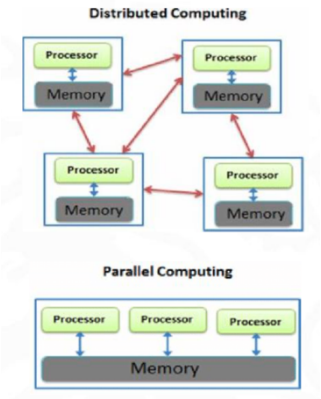
Tío, el resto de este tema es un rollo, Ch1_EffProgramming del tema 43 en adelante no esta en los apuntes.

Parallel Computing

Fundamentals

Nos permite hacer tareas más rápidas y utilizar un mayor número de recursos. Especialmente con GPU's, ya que CPU's son mejores para procesamiento lineal, no paralelo.

Rompemos problemas grandes en partes pequeñas, independientes, parecidas que se pueden ejecutar de manera simultánea en procesos distintos.

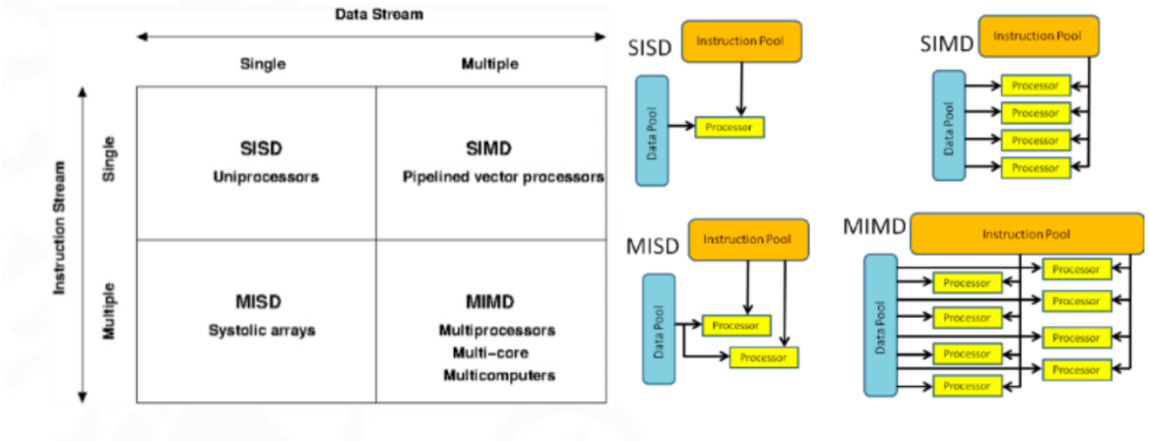


- Sequential → Una tras la otra.
- Parallel → Un problema, muchos subproblemas, ejecutar en distintos procesadores.
- Distributed → Sección de memoria dedicada al procesamiento de un problema específico.

Flynn's taxonomy (Sistemas Digitales II):

- SISD → Single Instruction, Single Data
- SIMD → Single Instruction, Multiple Data
- MISD
- MIMD

A classification of parallel computers is given by the Flynn's taxonomy:



Como diseñar un programa en paralelo:

- Descomposición por dominio
- Descomposición por función

Siempre considera lo siguiente:

- Cuál es la tarea que más tiempo lleva? Porque le va a limitar a las demas. El partitioning de tareas debe de ser la misma, o dinámicamente asignada.
- Sincrono o asíncrono?
- Como se van a interconectar?

Designing Parallel Programs

Terminology

Do Ctrl + F for any definitions which you might find in Chapter 2 of parallel computing including:

Parallel Computing Basics

- **Parallel computer**: a computer with multiple processors that can perform parallel processing.
- **Parallel processing**: processing information concurrently using multiple processors to solve a single problem.
- **Supercomputer/HPC**: a high-performance computer that solves problems extremely fast compared to other computers of the same time period.

Parallel Computing Terminology

- **Core**: a single processing unit in a multicore processor or a single machine in a cluster network.
- **Cluster**: a collection of objects that can host cores, either a network or a collection of cores in a computer.
- **Critical path**: the longest chain of dependent calculations in a program.
- **Task**: a logically discrete section of computational work executed by a processor.
- **Shared memory**: a model where parallel tasks have direct access to the same memory location.
- **Distributed memory**: network-based memory access for physical memory.

Parallel Computing Concepts

- **Communications**: data exchange between workers.
- **Synchronization**: coordinating parallel tasks in real-time, often involving waiting by at least one task.
- **Granularity**: a measure of the ratio of computation to communication.
- **Coarse**: large amounts of parallel computational work between communication events.
- **Fine**: small amounts of parallel computational work between communication events.
- **Parallel overhead**: time required to coordinate parallel tasks, including task start-up, synchronization, data communication, and software overhead.

Performance Metrics

- **Throughput**: the number of results produced per unit of time.
- **Pipeline**: a computation process designed in consecutive independent steps.
- **Sequential vs Parallelism Speedup** (S(n)): the ratio of the time needed for the most efficient sequential algorithm to the time needed for an algorithm incorporating pipelining and/or parallelism.
- **Parallelization Speedup** (S(n)): the ratio of the time required to run a program on a single processor to the time required on n parallel clusters/cores.

Hardware and Software

- **Degree of parallelism**: the total number of processors required to execute a program.
- **Hardware parallelism**: built into the machine architecture/chip design.
- **Software parallelism**: concurrent execution of machine language instructions in a program.
- **Clock rate**: the inverse of the constant cycled time that drives the CPU.
- **Instruction Count** (IC): the number of machine instructions to be executed by a program.
- **Cycles Per Instruction (CPI):** the ratio of CPU clock cycles to instruction count.

Processing Units

- **Central Processing Unit** (CPU): the electronic circuitry that carries out instructions of a computer program.
- **Graphics Processing** Unit (GPU): a specialized electronic circuit designed for rapid image manipulation and parallel processing.
- **Multi-Core Processor**: a single computing component with two or more independent CPUs (cores) that can run multiple instructions simultaneously.
- **Coprocessor**: a computer processor that supplements the functions of the CPU by offloading processor-intensive tasks.

GPU Computing

GPU (Graphics Processing Unit) is a processing unit designed to render graphics quickly by having a large number of simple processing units for massively parallel calculations. The idea of general-purpose GPU computing (GPGPU) is to exploit this capability for general computing.

GPU Computing Providers

Nvidia and AMD. Intel is also entering the market with its ARC GPUs.

GPU Computing Platforms/Libraries

To use a GPU for general programming, you need to use a specific platform/library to send code to the GPU:

- CUDA is a platform for programming on GPUs, specifically for NVIDIA GPUs, that allows you to submit C/C++/Fortran code for execution on the GPU.
- OpenCL is an alternative that works with a wider variety of GPUs.

GPU Computing Limitations

- Many processing units but somewhat limited memory.
- They can only use data in their own memory, not the CPU's memory, so data must be transferred between the CPU (the host) and the GPU (the device).
- This copy can, in some calculations, constitute a very large fraction of the overall calculation.

CPU, GPU, TPU, and FPGA

- **Central Processing Unit (CPU):** executes instructions comprising a computer program, performs basic arithmetic, logic, controlling, and input/output (I/O) operations specified by the instructions in the program.
- **Graphics Processing Unit (GPU):** specialized processing units designed to process images and videos, ideal for applications in which data need to be processed in parallel.
- **Tensor Processing Unit (TPU):** an Application-Specific Integrated Circuit (ASIC) developed by Google specifically for neural network machine learning, optimized for TensorFlow models.
- **Field Programmable Gate Array (FPGA):** an integrated circuit designed to be configured by a customer or a designer after manufacturing, can achieve higher performance, lower cost, and lower power consumption compared to other options like CPUs and GPUs.

GPU Computing Sequence of Operations

1. Allocate memory on the GPU.
2. Transfer data from CPU to GPU.
3. Launch the CUDA kernel to operate on threads, with a block/grid specific configuration.
4. (Optionally) run another kernel, which can access data stored on the GPU, including the results of the previous kernel.
5. Transfer the results back to the CPU.

GPU-Accelerated R Software Stack

1. Use R GPU packages from CRAN
2. Access the GPU through CUDA libraries and/or CUDA-accelerated programming languages, including C, C++, and Fortran.

Testing

Introduction to Testing

Tests ensure our program works as intended and that changes to the codebase do not break existing functionality.

What is Testing?

- Testing is a contract with our code - given a value, we expect a certain result to be returned.
- Tests are a way to ensure our code is working in the manner defined by the test.

Properties of Tests

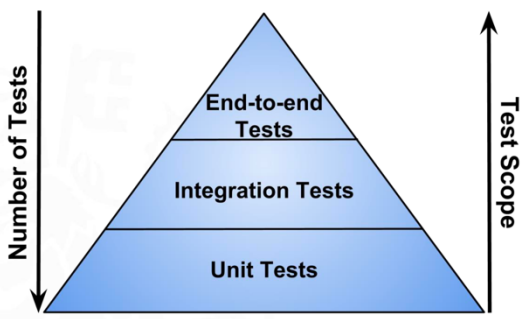
- **Fast**: tests should be fast to run to ensure quick feedback.
- **Deterministic**: tests should be deterministic, meaning the same input will always result in the same output.
- **Automated**: tests should be automated to quickly verify our program works.

Benefits of Testing

- Modify code with confidence: tests ensure changes to the codebase do not break existing functionality.
- Identify bugs early: fixing bugs gets more expensive the further you are in the project.
- Improve system design: writing tests forces you to use your own API by writing modular code.

Types of Testing

- **Black Box Testing:** testing techniques in which the tester cannot see the inner workings of the item being tested.
- **White Box Testing**: testing techniques in which the tester can see the inner workings of the item being tested.
- **Unit tests:** low-level tests that focus on testing a specific part of our system.
- **Integration tests:** tests that combine various parts of the system and test them together as a group.
- **End-to-End tests:** tests that check to see if the system meets our defined business requirements.



Test Patterns

- **Arrange-Act-Assert (AAA):** a pattern for structuring tests that separates the different parts of our tests into arrange, act, and assert phases.
- **Given-When-Then**: a pattern for structuring tests that separates the different parts of our tests into given, when, and then phases.

What to Test

- **Functional requirements**: make sure that all relevant requirements have been implemented.
- **Basis Path**: test each statement at least once.
- **Equivalence Partitioning**: two test cases that result in the same output are said to be equivalent.
- **Boundary Analysis**: test the boundary conditions of our code.

Test Results

- **Code Coverage**: a white box testing technique that monitors the extent to which the code has been executed.
- **Test Coverage**: a black box testing technique that monitors the number of tests that have been executed.

When to Test

- It doesn't matter when you write tests, it just matters that you write tests.

Statistical Analysis – Desde este punto en Adelante son los más importantes

Overview
Statistics is a collection of tools and methods for:

- Evaluating. Interpreting, Displaying, Making decisions based on data

Applications
Used by data analysts to:

- Describe characteristics of datasets (size, quantity, accuracy)
- Employ data visualization and statistical methods

Key Terms in Statistics

- **Data**: Collections of observations, such as measurements or survey responses.
- **Population**: Complete collection of all individuals to be studied.
- **Sample**: Subcollection of members selected from a population.
- **Census**: Collection of data from every member of the population.
- **Parameter**: Numerical measurement describing a population characteristic.
- **Statistic**: Numerical measurement describing a sample characteristic.

Types of Statistical Analysis
Descriptive Statistics

- **Purpose**: Summarize the data.
- **Methods**: Use of summary charts, graphs, and tables.
- **Examples**: Mode, median, mean, range, variance, standard deviation.

Inferential Statistics

- **Purpose**: Draw conclusions from data samples to apply to larger populations.
- **Methods**: Use of estimates, confidence intervals, and credible intervals.

Subcategories of Statistical Analysis

- **Observational**: Data collection without interference.
- **Experimental**: Subjects randomly assigned to treatments to establish causality.
- **Retrospective**: Uses past data.
- **Prospective**: Data collected throughout the study.

Confounding Variables

- Variables that affect both the explanatory and response variables, misleading apparent relationships.

Types of Sampling

- **Simple Random Sample (SRS)**
  - Each case in the population has an equal chance of selection.
- **Stratified Sampling**
  - Population divided into homogenous strata; SRS performed within each stratum.

- **Cluster Sampling**
    - Population divided into clusters; SRS of clusters, then sample all observations within these clusters.
- **Multistage Sampling**
    - Combination of cluster and simple random sampling methods.

Sampling bias:

- **Convenience Sample:** Individuals who are easily accesible are more likely to be included in the sample.
- **Non-response:** If only a fraction of the randomly sampled people respond to a survey such that the sample is no longer representative of the population.
- **Voluntary response:** Occurs when the sample consists of people who volunteer to respond because they have strong opinions on the issue.

Principles of Experimental Design

- Explanatory Variables: Conditions imposed on experimental units.
- Blocking Variables: Characteristics of units controlled for during the experiment.

Blocking vs. Stratifying

Blocking is used during random assignment. Stratifying is used during random sampling.

Descriptive Statistics

Measures of Central Tendency

- Mean: Mathematical average, sensitive to outliers.
    - **Arithmetic Mean**: The most-commonly used average.
    - **Geometric Mean**: Average on a set of values which are multiplicative.
    - **Harmonic Mean**: Average of a set of values that are rates or ratios.
- Median: Middle value, less affected by outliers.
- Mode: Most frequently occurring value, useful for categorical data.

Measures of Dispersion

- Range: Difference between the highest and lowest values.
- Quartiles: Divide data into four equal parts.
- Variance: Average of squared deviations from the mean.
- Standard Deviation: Square root of variance, measures data spread.

Graphical Representation of Data

- Bar Chart: Represents categorical or discrete data.
- Line Chart: Shows trends over time.
- Scatter Plot: Depicts relationship between two numerical variables.
- Histogram: Summarizes distribution of interval data.

"Statistics is a collection of tools for decision-making based on data."
"Inferential statistics use data to draw larger truths."
"The mean is sensitive to outliers, whereas the median is less so.

Probability

The probability of an event is a proportion of chance between 0 and 1, that an event is likely to happen. There are multiple interpretations:

- *Frequentist interpretation:*
  The probability is the proportion of times the **outcome would occur if we observed the process an infinite number** of times.
- *Bayesian interpretation*:
  Probability as a **subjective measure that varies between observers**, integrating prior beliefs and evidence.

Law of Large Numbers: The average of the results from a large number of trials should be close to the expected value and will tend to become closer as more trials are performed.

Gambler's Fallacy: The mistaken belief that past random events affect the probability of future independent events.

Disjoint Events:

Events that can't occur simultaneously are known as disjoint or **mutually exclusive.** For example, in a single coin toss, the probability of getting heads and tails is imposibbly

Union of Disjoint Events:
The probability that either event A or event B occurs is simply

$$P(A \cup B) = P(A) + P(B)$$

General Addition Rule:
For non-disjoint events,

$$P(A \cup B) = P(A) + P(B) - P(A \cap B)$$

Conditional probabilities:

Conditional Probability refers to the probability of one event given the occurrence of another. It's denoted as $P(A|B)$, which is the **probability of A given B has occurred**.

$$P(A|B) = \frac{P(A \cap B)}{P(B)} \rightarrow Bayes\ Theorem$$

Bayesian Inference is a method of statistical inference in which Bayes' theorem is used to update the probability for a hypothesis as more evidence or information becomes available.

Probability Distribution

A **Probability Distribution** describes how probabilities are distributed over the values of a random variable.

- **Normal Distribution (Gaussian distribution)**: Characterized by its bell-shaped curve, symmetric about the mean. It is defined by two parameters: mean (μ) and standard deviation (σ).

$$P(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

- **Binomial Distribution**: Describes the number of successes in a fixed number of independent Bernoulli trials.

$$P(k\ successes\ in\ n\ trials) = \binom{n}{k} p^k (1-p)^{n-k}$$

Where $\binom{n}{k} = \frac{n!}{k!(n-k)!}, \mu = n * p, \sigma = \sqrt{n \cdot p \cdot (1-p)}$

We can also approximate a normal distribution using the formulas in the line above if the number of trials is high with the following conditions:

$$n \cdot p \geq 10\ or\ n \cdot (1-p) \geq 10$$

Inferential Statistics

Central Limit Theorem

Estos es lo del z value.

The Central Limit Theorem states that the distribution of sample means approximates a normal distribution as the sample size increases, regardless of the population's distribution shape, provided the samples are independent and identically distributed. The sample mean x will have a mean μ and a standard error $SE = \frac{\sigma}{\sqrt{n}}$ where σ is the population standard deviation and nn is the sample size.

$$x \sim \mathrm{N}\left(\mathrm{mean} = \mu, \mathrm{SE} = \frac{\sigma}{\sqrt{n}}\right)$$

Conditions for the CLT:

- **Independence:** Sample observations must be independent.
  - Random sample/assignment
  - If sampling with replacement, n < 10% of the population
- **Sample size/skew:** Eihter the population distribution is normal, or the sample size is large (n > 30)

Confidence Intervals

A confidence interval (CI) is a range of values that is used to estimate the true value of a population parameter. This interval captures the uncertainty around the estimate, providing both a central value (the point estimate) and a margin around that estimate where the true parameter value is expected to lie with a certain probability.

Importance of Confidence Intervals:

- **Point Estimates Alone are Misleading:** Solely reporting a point estimate can be misleading as it gives a false impression of certainty. It's improbable to hit the exact population parameter with just a point estimate.
- **Range of Plausible Values:** Reporting a range (confidence interval) around the point estimate is more informative, increasing the likelihood that the interval includes the true parameter value.

Calculation of confidence intervals:

- **Using the CLT:**
  If the sample x is normally distributed (or approximately normal with a large enough sample size), a confidence interval for the population mean μ can be constructed.
- **Formula:** CI = $x \pm z^* \times \frac{s}{\sqrt{n}}$, where the sample

When the confidence level increases, the confidence interval is wider, as it's able to capture more values.

Factors influencing the Width of a confidence interval:

- **Confidence Level:** Higher confidence levels (e.g., 99% vs. 95%) lead to wider intervals. This is because a wider interval is more likely to capture the true population parameter.
- **Sample Size**: Larger sample sizes reduce the standard error, thereby narrowing the confidence interval.
- **Population Variability:** Greater variability (standard deviation) in the population results in wider confidence intervals, as the data are more spread out.

Accuracy vs. precision:

Accuracy refers to how close a measured or calculated value is to the true value. It reprensents

Real-world example:

Hypothesis Testing

Comparing two means

Inference for proportions

Introduction to Bayes

Bayesian Interference