USO DE AMBIENTES REPRODUCIBLES PARA LA VALIDACIÓN DE MODELOS DE APRENDIZAJE ESTADÍSTICO

Fabricio Machado, Mauro Loprete

Resumen

Este trabajo documenta la implementación de un flujo de trabajo reproducible para un modelo de aprendizaje estadístico, utilizando datos socioeconómicos de Uruguay como caso de estudio. Mediante el uso de Docker, se encapsula el proceso completo de análisis asegurando la consistencia de los resultados independientemente del entorno de cómputo. El artículo describe las decisiones técnicas adoptadas y ofrece una guía práctica para facilitar la incorporación de estas herramientas en la investigación empírica.

Resumen

This paper documents the implementation of a reproducible workflow for a statistical learning model, using socioeconomic data from Uruguay as a case study. Through Docker, the entire analysis process—from data preparation in R to cross-validation—is encapsulated, ensuring result consistency regardless of the computing environment. The article describes the technical decisions made and offers a practical guide to facilitate the adoption of these tools in empirical research.

1. Introducción

El uso de técnicas de aprendizaje estadístico se ha extendido a múltiples campos de investigación. Estos métodos requieren fundamentar las decisiones tomadas para su implementación y, además, pueden ser computacionalmente muy exigentes (Hastie (2009), James et al. (2018)). Por este motivo, se han desarrollado herramientas complementarias a los programas y entornos de software habituales que permiten afrontar estos desafíos y, al mismo tiempo, fomentan la reproducibilidad. En este contexto, la reproducibilidad no debe entenderse como un objetivo final, sino como parte del proceso de trabajo a incorporar desde el inicio, lo que facilita la verificación de resultados, mejora la colaboración y aumenta la credibilidad del trabajo ante la comunidad académica (Peng (2011), Stodden, Leisch, and Peng (2014), Stodden et al. (2016)).

Aunque la discusión sobre reproducibilidad y ciencia abierta ha ganado espacio en la literatura (Baker (2016), Christensen, Freese, and Miguel (2019)), son todavía relativamente escasos los ejemplos detallados que muestren cómo llevar estas recomendaciones a la práctica en investigaciones aplicadas en estadística y ciencias sociales. Este trabajo busca contribuir en ese sentido, ofreciendo un caso de estudio que sirva como puente entre los principios generales de la reproducibilidad y las decisiones concretas que toman quienes trabajan con datos en su día a día. Al hacerlo, apunta en particular a la comunidad académica local y regional, donde la adopción de entornos reproducibles aún es incipiente, con el objetivo de facilitar su incorporación progresiva y fomentar la calidad y la transparencia de las investigaciones empíricas.

En este trabajo documentamos el uso de entornos reproducibles en Docker para implementar y validar un modelo de aprendizaje estadístico en el marco de una investigación en ciencias económicas. Partimos del trabajo previo presentado en Machado (2023). El caso de estudio se basa en un modelo de análisis discriminante con validación cruzada k-fold, aplicado a datos de encuestas de hogares para clasificar trabajadores en grupos socioeconómicos. Si bien retomamos brevemente el contexto de la investigación y los métodos de aprendizaje estadístico allí aplicados, en esta ocasión nos centramos en el proceso de trabajo que suele quedar oculto en los documentos finales. En dicho proyecto, el uso de entornos reproducibles fue un componente integral de la investigación y se tradujo en ganancias concretas de eficiencia, reproducibilidad y portabilidad del análisis.

Describimos paso a paso la implementación del modelo en entornos reproducibles con Docker e introducimos los conceptos básicos necesarios para su uso. Además, incorporamos un componente interactivo al ofrecer una guía para descargar y ejecutar el proceso, disponible públicamente en GitHub Container Registry, y complementamos el ejemplo con otras herramientas como GitHub Codespaces, de modo que cualquier persona pueda adaptar el flujo de trabajo a sus propios proyectos. Con ello buscamos acercar a la comunidad académica al uso de estas herramientas, reduciendo la barrera de entrada y ofreciendo una guía práctica que facilite que quienes leen este documento puedan incorporarlas efectivamente en su actividad de investigación. De esta manera, el documento combina un aporte metodológico, al proponer un flujo de trabajo reproducible, con un aporte pedagógico orientado a facilitar la adopción de estas prácticas en la comunidad académica.

Docker es una herramienta de software que permite ejecutar aplicaciones en entornos aislados, consistentes y reproducibles, reduciendo los problemas habituales asociados a versiones, dependencias y diferencias entre sistemas (Boettiger (2015)). Facilita que un programa y todos los componentes necesarios para su funcionamiento se configuren una sola vez y puedan ejecutarse de forma prácticamente idéntica en distintos contextos (equipos personales, servidores institucionales o infraestructuras en la nube), lo que simplifica las tareas de prueba y despliegue, mejora la colaboración y favorece la automatización de entornos complejos (Rad, Bhatti, and Ahmadi (2017)).

En el contexto de la investigación empírica, donde es habitual combinar varios lenguajes y herramientas y trabajar en diferentes máquinas, Docker permite definir un entorno único para cada proyecto y ejecutarlo de forma homogénea en cualquier equipo. De este modo se favorece la reproducibilidad en el tiempo, al hacer posible volver a correr los archivos de replicación de un paper años después, y se facilita el intercambio con coautores, tribunales o replicadores, evitando dedicar horas a instalar dependencias o resolver conflictos de versiones. Además, Docker es compatible con prácticamente todos los programas y herramientas que se utilizan habitualmente en investigación aplicada, incluyendo, entre otros, R y RStudio Server, Python y Jupyter, Stata (en servidores con licencia) y sistemas de composición de documentos como LaTeX o Quarto.

En suma, nuestros objetivos son: (i) documentar un flujo de trabajo reproducible para la implementación

y validación de un modelo de aprendizaje estadístico en ciencias sociales con Docker, y (ii) ofrecer una guía práctica para que otros investigadores puedan adaptar este enfoque a sus propios proyectos con esta herramienta.

El documento se organiza de la siguiente manera. La sección 2 presenta el marco conceptual y define con claridad los principales conceptos utilizados a lo largo del texto. La sección 3 describe el contexto de la investigación, los datos empleados, los modelos de aprendizaje estadístico considerados y los paquetes de R utilizados para su implementación. En la sección 4 se introducen los componentes principales de Docker y su funcionamiento, a partir del ejemplo de implementación del modelo presentado en la sección anterior. La sección 5 ofrece una guía para la ejecución del desarrollo disponible públicamente en GitHub Container Registry e introduce el uso de GitHub Codespaces para adaptar el flujo de trabajo a proyectos propios. Por último, la sección 6 recoge los comentarios finales y destaca tanto los resultados del modelo como los beneficios observados del uso de Docker en términos de eficiencia y reproducibilidad.

2. Marco conceptual

En esta sección presentamos el marco conceptual que organiza el resto del trabajo y concreta, en términos operativos, las ideas introducidas en la sección anterior. Definimos y discutimos una serie de nociones que consideramos centrales para pensar la relación entre métodos de aprendizaje estadístico y buenas prácticas de investigación empírica: ciencia abierta, archivos de replicación, entorno de cómputo, reproducibilidad computacional, entorno reproducible, contenedores de software, portabilidad y flujo de trabajo reproducible. El objetivo es fijar un lenguaje común y explicitar cómo estos conceptos se articulan entre sí, de modo que el uso de Docker y los desarrollos posteriores puedan entenderse no solo como una solución técnica, sino como una forma concreta de materializar esos principios en el trabajo cotidiano con datos.

Ciencia abierta: La ciencia abierta es un conjunto de prácticas orientadas a hacer más transparentes y accesibles los distintos componentes del proceso de investigación (ARCA-DPSS (s.f.)): datos, código, materiales, resultados y, cada vez más, entornos de cómputo. No se limita a la simple publicación de artículos, sino que promueve que los archivos de replicación, los protocolos y las decisiones de análisis se compartan de forma tal que otros puedan inspeccionarlos, reutilizarlos o extenderlos. En ese marco, la transparencia sobre cómo se procesan los datos y en qué entorno se ejecutan los análisis se vuelve tan importante como los resultados finales. Esta visión es coherente con el enfoque de la ciencia abierta como "término paraguas" discutido por Fecher and Friesike (2013) y con las propuestas específicas para la investigación en ciencias sociales de Christensen, Freese, and Miguel (2019).

Archivos de replicación: Los archivos de replicación (o materiales de replicación) reúnen los insumos necesarios para volver a ejecutar un análisis empírico: datos[1], código de procesamiento y estimación, y documentación mínima sobre cómo reproducir las tablas y figuras principales. Desde la perspectiva de la ciencia abierta, los archivos de replicación son el soporte concreto de la reproducibilidad computacional: permiten que otras personas verifiquen los resultados de una investigación sin depender del entorno de trabajo original del autor.

Entorno de cómputo: El entorno de cómputo, o entorno de software, es el conjunto de componentes técnicos sobre los cuales se ejecuta un análisis: sistema operativo, versiones de lenguajes (R, Python, Stata), bibliotecas y paquetes instalados, herramientas de compilación de documentos (LaTeX, Quarto), así como configuraciones específicas (localización, rutas, variables de entorno). En investigaciones empíricas complejas, este entorno rara vez es neutro: diferencias de versión o de configuración pueden hacer que un script falle o que ciertos resultados cambien marginalmente, aun cuando los datos y el código "parezcan" ser los mismos.

Reproducibilidad computacional: La reproducibilidad computacional se refiere a la capacidad de obtener exactamente los mismos resultados numéricos (tablas, figuras, estadísticas, métricas de desempeño) a partir de los mismos datos y el mismo código, incluso cuando el análisis se ejecuta en un momento distinto, por una persona distinta o en una máquina distinta (Sandve et al. (2013), Stodden, Leisch, and Peng (2014)). En la práctica, esto exige que los datos estén disponibles o adecuadamente documentados, que el código de

[1] Cuando no puedan difundirse los datos, se recurre a versiones anonimizadas o instrucciones claras para acceder a ellos.

procesamiento y análisis sea accesible y que el entorno de cómputo esté fijado o descrito con suficiente detalle como para que pueda ser recreado (Bechhofer et al. (2013)).

Entorno reproducible: Un entorno reproducible es un entorno de cómputo que ha sido fijado y documentado de modo tal que otra persona pueda reconstruirlo y ejecutar los mismos análisis con la seguridad de obtener resultados equivalentes. No se trata solo de listar versiones de software, sino de proporcionar un mecanismo concreto para reconstruir ese entorno: scripts de instalación, archivos de configuración, descriptores de paquetes o un contenedor que ya integra todos los componentes necesarios. De esta manera, el entorno deja de ser una condición implícita del análisis y pasa a ser un objeto explícito y compartible.

Contenedores de software: Los contenedores de software son entornos aislados y ligeros que empaquetan aplicaciones junto con sus dependencias y configuraciones, compartiendo el kernel del sistema operativo anfitrión pero manteniendo su propio sistema de archivos y su propia pila de software (Boettiger (2015)). Se centran en proporcionar un espacio consistente para ejecutar aplicaciones de forma eficiente y portable, evitando conflictos de versiones y simplificando el despliegue en diferentes máquinas. En el contexto de la investigación empírica, los contenedores permiten agrupar el código de análisis, las librerías necesarias y las herramientas auxiliares en una unidad autocontenida que puede trasladarse y ejecutarse de forma homogénea en distintos entornos físicos.

Portabilidad: La portabilidad es la capacidad de trasladar y ejecutar un sistema o una aplicación en distintos equipos y plataformas sin necesidad de realizar adaptaciones significativas ni correr el riesgo de que falle por diferencias de configuración. Para un proyecto de investigación, un entorno portable implica que el análisis pueda pasar sin fricciones desde la máquina del investigador al servidor institucional o a una máquina en la nube, produciendo los mismos resultados sin reconfiguraciones extensas. La portabilidad es complementaria a la reproducibilidad: mientras esta se enfoca en poder rehacer los resultados, aquella se preocupa por poder hacerlo en múltiples contextos y con el menor costo de adaptación posible. El uso de contenedores como solución a los problemas de dependencia y portabilidad en el marco de la investigación reproducible ha sido discutido, entre otros, por Boettiger (2015).

Flujo de trabajo reproducible: Un flujo de trabajo reproducible es la organización sistemática de los pasos de una investigación empírica (ingreso y limpieza de datos, transformación de variables, estimación de modelos, validación, generación de tablas y figuras, compilación del documento final) de forma que puedan ejecutarse de manera automática y determinista a partir de scripts y de un entorno de cómputo fijado (Wilkinson et al. (2016)). Más allá de la idea general de "ordenar el proyecto", un flujo reproducible implica que el proceso completo pueda rehacerse con un único comando o con un conjunto reducido de instrucciones bien documentadas, evitando operaciones manuales irrepetibles o decisiones implícitas que no quedan registradas. En este trabajo, el flujo de trabajo reproducible se materializa en la combinación de scripts (por ejemplo, en R o Stata) y contenedores Docker, de manera que tanto la lógica del análisis como el entorno en el que se ejecuta queden formalizados y puedan ser reutilizados por otros.

En conjunto, estos conceptos delinean el marco en el que se inscribe nuestro trabajo. La ciencia abierta aporta el horizonte normativo de transparencia y acceso; la reproducibilidad computacional fija el criterio mínimo para que los resultados puedan ser verificados; y la portabilidad asegura que ello pueda lograrse en distintos entornos físicos y organizacionales. Para alcanzar esos objetivos, resulta necesario explicitar el entorno de cómputo, transformarlo en un entorno reproducible y apoyarse en herramientas como los contenedores de software, que permiten encapsularlo y distribuirlo. Sobre esta base se construye el flujo de trabajo reproducible que proponemos, en el que tanto los pasos del análisis como el sistema en que se ejecutan se conciben como partes integrales del método de investigación.

3. Modelo de aprendizaje estadístico

En esta sección resumimos el modelo de aprendizaje estadístico que utilizamos como caso de estudio para ilustrar el flujo de trabajo reproducible implementado con Docker. El objetivo no es ahondar en la discusión sustantiva sobre desigualdad económica (desarrollada en detalle en Machado (2023)), sino describir los componentes del modelo que son relevantes para su implementación, validación y posterior encapsulamiento en entornos reproducibles.

Partimos de un análisis sobre la desigualdad económica cuyo objetivo es clasificar a los individuos en grupos sociales a partir de variables relevantes para estudiar su participación en el ingreso laboral (Machado (2023)). En una primera etapa se aplica un **análisis de clúster**, que permite detectar grupos relativamente homogéneos entre los perceptores de ingresos laborales en un año base, utilizando variables que resumen las características de los trabajadores y de sus puestos de trabajo. En una segunda etapa se emplea **análisis discriminante** para construir, a partir de esos clústeres, una regla de clasificación que pueda aplicarse a otros años y así seguir la evolución de los grupos y su participación en el ingreso laboral. Dado que esta regla se estima en un conjunto de datos y luego se aplica sobre información externa, se utilizan técnicas de **validación cruzada** para evaluar su desempeño y reducir el riesgo de sobreajuste.

A continuación se detallan los datos utilizados y los métodos que forman parte del proceso de implementación en Docker, estos incluyen en particular los componentes de análisis discriminante y validación cruzada. El paso previo de análisis de clúster puede consultarse en el Anexo A.

3.1. Datos

La fuente de datos utilizada es la Encuesta Continua de Hogares (ECH) elaborada por el Instituto Nacional de Estadística (INE). Utilizamos información de personas y hogares para los años 2006, 2011, 2019 y 2021. En cada año se consideran las personas ocupadas de 20 años o más de todo el país con ingresos laborales, definiéndose ocupación como haber trabajado al menos una hora en la semana de referencia. Para el análisis se seleccionan variables que recogen información sobre las tareas realizadas en el trabajo, el sector de actividad, la categoría de ocupación, la informalidad, las horas trabajadas, el nivel educativo, la propiedad de la vivienda y el cobro de rentas. Estos datos constituyen la base sobre la que se estiman el modelo discriminante y los procedimientos de validación descritos a continuación.

El trabajo con datos se realiza íntegramente en R mediante un flujo de trabajo basado en el script `main.R`. La lectura de los archivos originales de la ECH se lleva a cabo con la librería `vroom` (vroom v1.6.6; Hester, Wickham, and Bryan (2025)), aprovechando su eficiencia para importar grandes volúmenes de datos, mientras que `here` (here v1.0.2; Müller (2020)) se utiliza para manejar las rutas de archivos de forma relativa al proyecto. La manipulación y depuración de los datos se estructura mediante el operador tubería de `magrittr` (magrittr v2.0.4; Bache and Wickham (2025)) y las funciones de `tidytable` (tidytable v0.11.2; Fairbanks (2025)), que permiten filtrar la población de interés, generar las variables y seleccionar los subconjuntos necesarios para el modelo. Estas dependencias de software forman parte del entorno que luego se fijará explícitamente en el contenedor de Docker.

3.2. Análisis discriminante

El objetivo de esta aplicación es obtener una misma clasificación de grupos para todos los años seleccionados, manteniendo las características distintivas de cada uno. A partir de un conjunto de datos en el que la pertenencia a los grupos es conocida (el año base en el que se aplicó el análisis de clúster), se deriva una regla para asignar las observaciones a los grupos (McLachlan (2005), Hastie (2009)), minimizando la probabilidad de clasificación incorrecta. En la práctica, se clasifican los datos de 2006, 2019 y 2021 en la misma cantidad de grupos y con la misma caracterización resultante del análisis de clúster realizado en 2011, año elegido como base por presentar la clasificación más estable (véase el Sección A).

Entre las distintas funciones discriminantes posibles se eligió un modelo de regresión logística multinomial, dado que todas las variables explicativas son cualitativas (McFadden (1972), Pohar, Blas, and Turk (2004), Hosmer Jr, Lemeshow, and Sturdivant (2013)). Dado que se espera encontrar más de dos grupos, se asume que la variable que indica la subpoblación de pertenencia proviene de una distribución multinomial. Las probabilidades logarítmicas relativas entre cada grupo k y un grupo de referencia K se modelan como (James et al. (2018)):

$$\log(\tfrac{Pr(Y=k|X=x)}{Pr(Y=K|X=x)}) = \beta_{k0} + \beta_{k1}x_1 + ... + \beta_{kp}x_p$$

En nuestra especificación se predefinen cuatro grupos ($K = 4$), por lo que dejando fuera al de referencia nos quedan tres funciones de enlace para $k = 1, 2, 3$ ($k = 1, ..., K - 1$). Se denotan como $X = (X_1, ..., X_p)$ el conjunto de variables utilizadas, siendo p la cantidad de categorías de todas las variables[2].

Con la información obtenida del año base se pretende discriminar a los individuos de los demás años. Los individuos se asignan a un grupo u otro comparando las probabilidades de pertenencia a cada grupo predichas por el modelo. Según esta regla de clasificación, el grupo asignado es aquel con probabilidad posterior más alta. Para evaluar el modelo se utiliza una medida de precisión definida como la cantidad de datos bien clasificados sobre el total. Valores altos de la misma implican un buen desempeño del modelo.

En términos computacionales, el análisis discriminante se implementa en R utilizando la función `multinom()` de la librería `nnet` (nnet v7.3-20; Venables and Ripley (2002), Venables and Ripley (2025)), que permite estimar el modelo de regresión logística multinomial sobre el conjunto de datos del año base y aplicar posteriormente la regla de clasificación mediante el operador `predict()` de R (R v4.4.1; R Core Team (2025)) a los demás años. Este modelo y su regla de clasificación constituyen el núcleo del flujo analítico que, en las secciones siguientes, será automatizado y encapsulado en el entorno reproducible definido con Docker.

3.3. Validación cruzada

Como es de interés clasificar datos externos, para evitar problemas de sobreajuste se utiliza el método de **validación cruzada** $k - fold$ (Stone (1974), James et al. (2018)). Se dividen los datos aleatoriamente en k pliegues de un tamaño similar, se estima la precisión tomando uno de los pliegues para el testeo y los restantes $k - 1$ para el entrenamiento, el procedimiento se repite k veces haciendo variar el pliegue de testeo de tal forma que todos los datos hayan sido utilizados como testeo y entrenamiento. Como resultado se obtienen k estimaciones de la precisión y se toma su promedio como una medida global del ajuste sobre el total de los datos.

La validación cruzada $k-fold$ se programó íntegramente en R a partir de funciones propias. El script principal `main.R` organiza el procedimiento, llamando a la función de validación `cross_validation()` definida en el script auxiliar `CV.R`. Posteriormente se llama a la función de cálculo de precisión `prf`, definida en el script auxiliar `evaluation.R`. Ambos scripts auxiliares se encuentran en el directorio de dependencias de R del proyecto. Se explota la independencia de los procesos de validación cruzada para correrlos en paralelo mediante las funciones del paquete base `parallel` de R (R v4.4.1; R Core Team (2025)).

Como se verá en la sección 4, estas decisiones de implementación (paquetes utilizados, estructura de las funciones de validación y uso de computación en paralelo) son las que se fijan posteriormente en el entorno reproducible definido con Docker.

4. Implementación

Las secciones previas fijaron los conceptos y el modelo que motivan este estudio de caso; ahora trasladamos esas ideas al plano operativo. Empezamos repasando los componentes que intervienen en el ciclo de vida de una imagen Docker y cómo se articulan con un proyecto que combina scripts, datos y documentación. Luego mostramos un ejemplo completo de Dockerfile, junto con los comandos `docker` necesarios para construir, ejecutar y compartir el flujo analítico.

El énfasis está puesto en describir qué problema resuelve cada pieza y por qué, frente a alternativas como gestores de dependencias específicos por lenguaje o máquinas virtuales completas, un contenedor resulta más liviano y repetible para la colaboración cotidiana (Potdar et al. (2020)). La intención es que quienes sigan esta guía puedan reconstruir el entorno sin ambigüedades, independientemente del sistema operativo o el equipo desde el que trabajen.

[2]Los parámetros del modelo se estiman bajo el método de máxima verosimilitud con técnicas numéricas de optimización, en concreto el método Broyden-Fletcher-Goldfarb-Shanno.

4.1. Componentes de Docker

En el contexto de la investigación empírica, Docker permite "congelar" un entorno de trabajo completo (sistema operativo, librerías, versiones de R, Stata, Python, LaTeX/Quarto, etc.) y reutilizarlo o compartirlo sin depender de la configuración específica de cada máquina. En la práctica, lo que hace Docker es ejecutar estos programas dentro de un entorno Linux aislado, pero nosotros seguimos trabajando desde nuestro navegador o editor habitual (como RStudio, Positron o VS Code), conectándonos al contenedor.

Docker funciona como una capa de abstracción a nivel del sistema operativo, que permite empaquetar aplicaciones y sus dependencias en contenedores ligeros y portables. Se apoya en el kernel para aislar procesos y recursos, y en un sistema de archivos por capas (copy-on-write) que ahorra espacio y acelera las compilaciones. De este modo, el software se ejecuta de forma consistente en distintos entornos (equipo local, servidores o nube). Docker se ejecuta mediante comandos en la terminal (CLI) o a través de interfaces gráficas (Docker Desktop) y consta de varios componentes clave que facilitan la creación, distribución y ejecución de contenedores.

Los principales componentes de Docker son:

- `Dockerfile`: archivo de texto con instrucciones para construir la imagen.
- Imagen: plantilla estática y versionada del entorno completo.
- Contenedor: instancia en ejecución de la imagen.
- Registro: repositorio para almacenar y compartir imágenes (Docker Hub, GHCR).

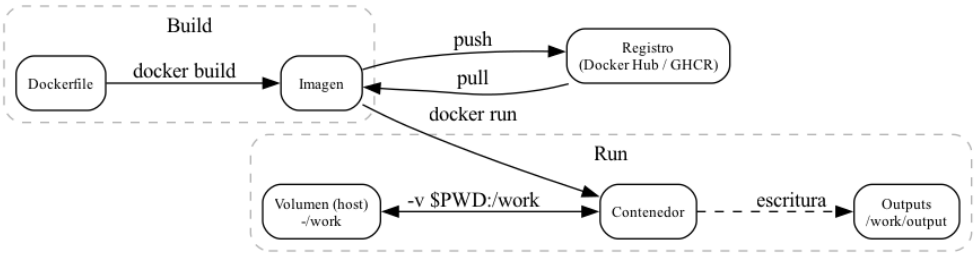

Figura 1: Ciclo de vida simplificado de Docker.

El flujo presentado en la Figura 1 permite integrar los componentes y ver como se interrelacionan en el ciclo de vida de Docker. Todo comienza escribiendo un `Dockerfile`, que resume el sistema operativo base, los paquetes y los archivos del proyecto. Con `docker build` se genera la imagen resultante, un artefacto inmutable que encapsula ese entorno y puede versionarse mediante etiquetas.

Una vez creada, la imagen puede publicarse en un registro como Docker Hub o GitHub Container Registry usando `docker push`, y descargarse en cualquier otra máquina con `docker pull`. Al ejecutar `docker run`, la imagen se convierte en un contenedor activo. En esa ejecución se monta el directorio del proyecto del equipo anfitrión (`-v $PWD:/work`), lo que permite que el contenedor lea datos existentes y escriba resultados persistentes en rutas como `/work/output`. La flecha punteada hacia "Outputs" recuerda que los artefactos relevantes deben guardarse en ese volumen compartido, porque todo lo que quede dentro del contenedor se descarta al finalizar la sesión.

Este flujo contrasta con las máquinas virtuales tradicionales, que copian un sistema operativo completo y requieren más recursos para inicializarse, y con gestores de dependencias específicos por lenguaje (por ejemplo, `renv` o `virtualenv`), que no encapsulan herramientas externas como compiladores o LaTeX. Docker reutiliza el kernel del anfitrión y empaqueta el resto de los componentes, por lo que resulta más ágil para alternar entre entornos y compartir una configuración única con colaboradores.

Para sostener la reproducibilidad a lo largo del tiempo, se recomienda etiquetar imágenes con fecha o versión (y, cuando sea crítico, referenciarlas por `digest`), congelar paquetes (R: `renv`; Python: `requirements.txt`)

y guardar `sessionInfo()` o `pip freeze` junto a los resultados. También es recomendable limitar recursos (CPU y memoria) para obtener ejecuciones controladas y comparables. Los datos y las credenciales no deben incorporarse en la imagen: se montan a través de un volumen con `-v` y se suministran por variables de entorno o archivos `.env`. También es conveniente fijar semillas aleatorias, normalizar configuraciones regionales y zonas horarias y ejecutar como usuario no privilegiado (o con `--user $(id -u):$(id -g)`). Documentar los comandos exactos de `docker build` y `docker run` (y los límites de recursos) en un README o Makefile mejora la trazabilidad.

El bloque de código 1 contiene los comandos esenciales para el uso de Docker para inspeccionar, ejecutar, observar y limpiar:

Código 1

```
# Inspección
docker images # Lista todas las imágenes locales
docker ps -a # Lista todos los contenedores (también los detenidos)

# Ejecutar con límites y variables
docker run --rm -it -v "$PWD":/work -w /work --cpus=4 -m 8g \
  --env-file .env ghcr.io/fabriciomsv/cv:latest /bin/bash

# Logs y acceso
docker logs -f <nombre_o_id> # Seguir logs en tiempo real
docker exec -it <nombre_o_id> /bin/bash # Acceder a la shell del contenedor

# Copiar artefactos (scripts, resultados, etc.)
docker cp <nombre_o_id>:/work/output/resultados.csv ./output/

# Limpieza
docker system prune -f # Limpia volúmenes, redes y contenedores detenidos
docker image prune -f # Limpia imágenes no usadas
```

A modo de cierre, es útil recordar algunas pautas para evitar errores frecuentes: utilizar siempre una imagen etiquetada y el Dockerfile versionado; fijar dependencias y semillas; montar datos y escribir resultados en `/work/output`; documentar el comando de ejecución y los recursos asignados; y conservar registros y metadatos de sesión junto a los outputs. Si los resultados no aparecen, suele deberse a que no se montó `-v "$PWD":/work` o no se escribió en la ruta montada; si faltan paquetes, deben instalarse en la imagen base o en un paso de inicialización; si surgen problemas de permisos, ejecutar con `--user $(id -u):$(id -g)`; y si hay limitaciones de memoria, aumentar `-m` o reducir el tamaño de los lotes.

4.2. Ejemplo

Para reproducir el flujo de trabajo descrito en la Sección 3, se preparó un Dockerfile que encapsula las dependencias de R, los scripts y la estructura de carpetas del proyecto. El Dockerfile utiliza como base la imagen `eddelbuettel/r2u:22.04`, instala los paquetes citados en la Sección 3.1 (`nnet`, `vroom`, `here`, `tidytable`, `magrittr`), crea los directorios `/home/ubuntu/model/{data,R,output}`, copia los archivos necesarios y define como comando por defecto la ejecución de `main.R`. Con este archivo se puede construir la imagen, ejecutarla en cualquier máquina con Docker instalado y obtener los resultados de forma reproducible (véase el bloque de código 2).

Una vez creada la imagen a partir de este `Dockerfile`, los contenedores ejecutarán el script `main.R`, que implementa el modelo de aprendizaje estadístico con validación cruzada y guarda los resultados en la carpeta `output/`. El comando por defecto puede ser sobrescrito al ejecutar el contenedor (mediante `docker run`)

Código 2

```
FROM eddelbuettel/r2u:22.04  # Ubuntu 22.04

# Estructura de proyecto
RUN mkdir -p /home/ubuntu/model/{data,R,output}
WORKDIR /home/ubuntu/model

# Paquetes de R del sistema (binarios)
RUN apt-get update && apt-get install -y \
    r-cran-nnet \
    r-cran-vroom \
    r-cran-here \
    r-cran-tidytable \
    r-cran-magrittr \
  && rm -rf /var/lib/apt/lists/*

# Copiar archivos (opcional; preferible montar con -v en ejecución)
COPY R/* R/
COPY data/ data/
COPY main.R main.R

# Variables de entorno de ejemplo
ARG RUN_MODE=prod
ENV RUN_MODE=${RUN_MODE}

# Comando por defecto (En este script se ejecutan los modelos con CV)
CMD ["Rscript", "main.R"]
```

si se desea correr otro script o ingresar a una shell interactiva (`docker run imagen -it bash`) dentro del contenedor.

Al construir esta imagen, los comandos de ejecución deben montar el proyecto en la misma ruta definida como `WORKDIR` para que los scripts y outputs queden en el lugar esperado. Como puede verse en el bloque de código 3.

Esta imagen tiene un ejemplo básico para que devuelva la métrica de precisión del modelo, definida en la Sección 3.3 como la forma correcta de validar los resultados. En otros casos puede ser útil guardar otras métricas, modelos intermedios o datos procesados en la carpeta `output/`, que está montada desde el equipo anfitrión.

La imagen ya se encuentra construida y publicada en base a la ultima versión del código en el repositorio de GitHub y mediante GitHub Actions se actualiza automáticamente al crear nuevas etiquetas.

5. Ejecución

Para trabajar con la imagen pública de este proyecto alojada en GitHub Container Registry (GHCR: `ghcr.io/fabriciomsv/cv`) no es necesario modificar nada: basta con descargarla, ejecutarla montando la carpeta del proyecto y, si se desea mantener una variante local, construir una imagen propia a partir del Dockerfile. De manera esquemática, un flujo típico de trabajo puede resumirse del siguiente modo.

Pasos de uso (imagen GHCR):

Código 3

```
# Construir la imagen local
docker build -t cv_multinom .

# Abrir una shell en la ruta del proyecto dentro del contenedor
docker run --rm -it \
  -v "$PWD":/home/ubuntu/model -w /home/ubuntu/model \
  cv_multinom /bin/bash

# Ejecutar el script principal y escribir resultados en ./output
docker run --rm \
  -v "$PWD":/home/ubuntu/model -w /home/ubuntu/model \
  cv_multinom Rscript main.R
```

1. Obtener la imagen
   - Para imágenes públicas (como en este caso), ejecutar:

     ```
     docker pull ghcr.io/fabriciomsv/cv:latest
     ```

   - Si fuera privada, autenticarse previamente en GitHub Container Registry (GHCR) con un personal access token (PAT) asociado a la cuenta de GitHub.
2. Vincular el proyecto local
   - Desde la carpeta del proyecto (donde residen código y datos), ejecutar:

     ```
     docker run --rm -it -v "$PWD":/work -w /work ghcr.io/fabriciomsv/cv:latest /bin/bash
     ```

3. Ejecutar el script de análisis
   - Por ejemplo:

     ```
     docker run --rm -v "$PWD":/work -w /work ghcr.io/fabriciomsv/cv:latest Rscript main.R
     ```

4. (Opcional) Congelar una variante propia
   - Clonar el repositorio, adaptar el Dockerfile si es necesario y construir:

     ```
     docker build -t cv_multinom --build-arg test_arg=TRUE .
     ```

Para la ejecución cotidiana y colaborativa, resulta útil acompañar el flujo de trabajo con herramientas auxiliares. La integración continua (GitHub/GitLab CI) permite construir y publicar imágenes automáticamente al crear etiquetas; un Makefile facilita encapsular comandos repetitivos como `docker run` o `docker build`; y servicios como Amazon EC2 o AWS Batch resultan apropiados para ejecutar los mismos contenedores cuando se requiere mayor capacidad de cómputo.

Para obtener resultados de forma consistente:

- Descargar o actualizar la imagen y verificar la etiqueta/versión utilizada.
- Ejecutar el contenedor montando el proyecto y escribir los outputs en la carpeta compartida.
- Registrar logs, versiones de paquetes y metadatos de sesión junto a los resultados.

Para ilustrar la simplicidad de uso, el bloque de código 4 muestra los comandos mínimos para descargar y ejecutar la imagen pública del proyecto, montando la carpeta `output` para guardar los resultados generados por el script principal.

Código 4

```
docker pull ghcr.io/fabriciomsv/cv:main
docker run -v ${PWD}/output/:/home/ubuntu/model/output ghcr.io/fabriciomsv/cv:main
```

Este comando descarga la imagen pública y ejecuta el contenedor, montando la carpeta `output` del proyecto local para guardar los resultados generados por el script principal. Al tener en cuenta las buenas prácticas mencionadas, se asegura que los análisis sean reproducibles y portables entre distintos entornos y colaboradores.

5.1. Desarrollo en la nube con GitHub Codespaces

Una vez que se cuenta con la imagen Docker, existen múltiples formas de utilizarla para ejecutar el análisis de forma reproducible tanto en entornos locales (utilizando Docker Desktop o Docker Engine) como en la nube como GitHub Codespaces.

GitHub Codespaces es un servicio que permite abrir un entorno de desarrollo completo en la nube, basado en Visual Studio Code, directamente desde un repositorio de GitHub. Este entorno puede configurarse para utilizar contenedores Docker como base, lo que facilita la ejecución de análisis reproducibles sin necesidad de instalar software adicional en el equipo local.

A continuación se describen los pasos básicos para utilizar la imagen Docker del proyecto en un entorno de GitHub Codespaces (Figura 2). Una vez abierto el Codespace, se puede trabajar directamente con el código. El repositorio incluye un editor de texto listo para ejecutar el contenedor Docker (Figura 3). Esto puede ser especialmente útil para colaborar con otros investigadores, docentes, alumnos, ya que todos pueden acceder al mismo entorno sin preocuparse por las diferencias en la configuración local de sus equipos.

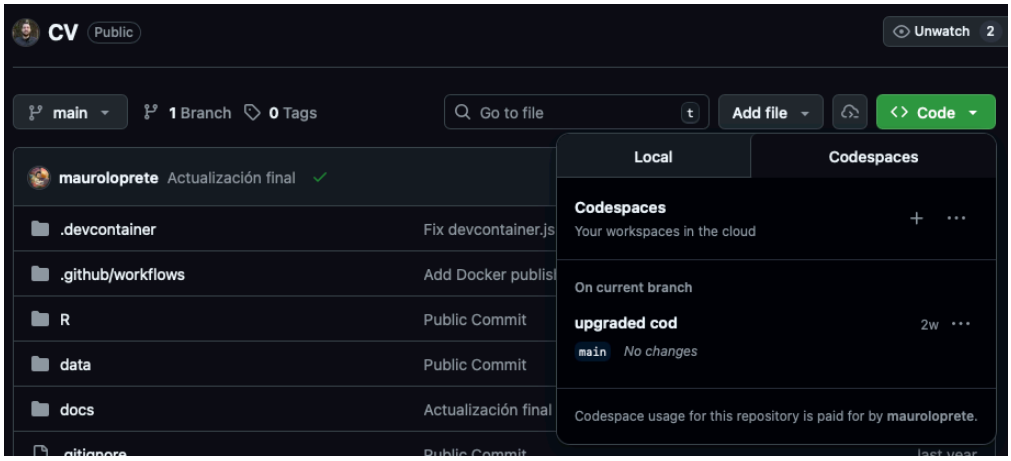

Figura 2: Apertura de un GitHub Codespace desde la interfaz del repositorio.

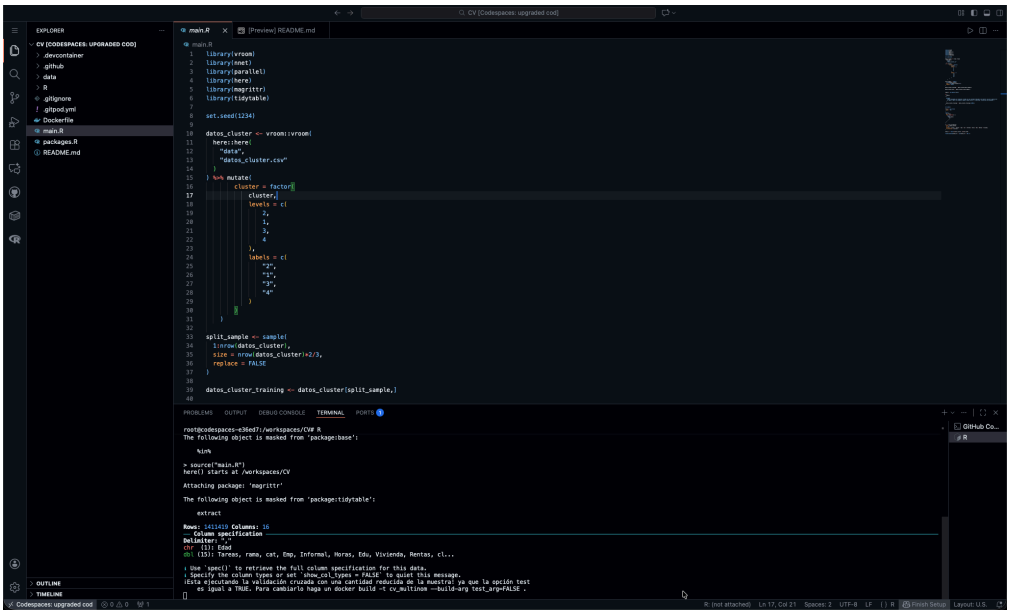

Figura 3: Entorno de desarrollo en la nube provisto por GitHub Codespaces.

GitHub Codespaces puede entenderse, en este contexto, como un complemento a los flujos basados en contenedores: permite abrir un entorno de desarrollo efímero en la nube, definido a partir del propio repositorio (por ejemplo mediante un archivo de configuración `.devcontainer.json`), en el que se puede inspeccionar el código, ajustar scripts y lanzar contenedores Docker con el mismo entorno que se utiliza localmente.

Es importante tener en cuenta que GitHub Codespaces presenta ciertas limitaciones en cuanto a recursos (CPU, memoria y almacenamiento) y tiempo de ejecución, por lo que puede no resultar adecuado para análisis muy intensivos o de larga duración. Servicios de cómputo en la nube como Amazon EC2 u otras plataformas de infraestructura como servicio suelen ser más apropiados cuando se necesita alquilar capacidad de procesamiento para ejecutar cargas pesadas en segundo plano (por ejemplo, corridas extensivas de validación cruzada, simulaciones o entrenamientos de modelos de mayor porte). En estos casos, la práctica recomendable es mantener la lógica del análisis y la definición del entorno en una imagen Docker versionada, y desplegar esa misma imagen tanto en el entorno local como en instancias de cómputo en la nube, de modo que la decisión sobre el 'dónde' ejecutar (equipo propio o nube) no afecte al 'cómo' se ejecuta el análisis.

6. Comentarios finales

Este trabajo presentó un estudio de caso en el que se documenta el uso de entornos reproducibles basados en Docker para la implementación y validación de un modelo de aprendizaje estadístico en ciencias sociales. Partiendo de una investigación sobre la distribución del ingreso laboral y la clasificación de trabajadores en grupos socioeconómicos, mostramos cómo traducir un flujo de análisis concreto, que combina datos de encuestas de hogares, regresión logística multinomial y validación cruzada, a un entorno de cómputo fijado, portable y ejecutable en distintos contextos. En la aplicación empírica, el modelo discriminante alcanzó una **precisión del 94,35 %**, desempeño al que puede llegar cualquier usuario que replique el proceso siguiendo los pasos que detallamos.

El aporte es doble. En primer lugar, proponemos un flujo de trabajo reproducible que integra de manera explícita el modelo de aprendizaje estadístico, los paquetes de R utilizados y el entorno de software subyacente, encapsulados en una imagen Docker versionada. En segundo lugar, acompañamos este diseño con una guía práctica que va desde la definición del Dockerfile hasta la ejecución del análisis mediante una imagen pública alojada en GitHub Container Registry, incluyendo ejemplos de uso local y en la nube (por ejemplo GitHub

Codespaces, Gitpod o instancias en Amazon EC2). De este modo, el trabajo no se limita a argumentar a favor de la reproducibilidad, sino que muestra paso a paso cómo alcanzarla en una aplicación empírica concreta.

En términos prácticos, la experiencia de implementación dejó beneficios claros. La encapsulación del entorno redujo fricciones habituales al mover el proyecto entre equipos y sistemas operativos, facilitó la reejecución del análisis en distintos momentos del tiempo y en distintas máquinas, y simplificó la colaboración con replicadores. El uso combinado de contenedores, scripts organizados y validación cruzada programada permitió automatizar tareas costosas, controlar de forma más precisa las dependencias de software y asegurar que el "cómo" se ejecuta el análisis permanezca estable aunque cambie el "dónde" se ejecuta (máquina personal, servidor institucional o nube).

Hemos comprobado que Docker puede ser una herramienta valiosa en la investigación en estadística y ciencias sociales. Sus principales fortalezas son la reproducibilidad, el aislamiento de dependencias, la portabilidad y la eficiencia computacional, especialmente cuando se trabaja con combinaciones de datos y métodos de alta exigencia computacional o cuando se detectan espacios para paralelizar los procesos. La experiencia documentada muestra que la adopción de contenedores Docker aporta beneficios tangibles para proyectos de métodos cuantitativos: reduce fricciones en la instalación de dependencias, estabiliza versiones de software, facilita la colaboración entre equipos y mejora la trazabilidad de los resultados. Al encapsular el entorno de ejecución, los análisis pueden replicarse en el tiempo y compartirse con terceros sin la carga de reconstruir manualmente las condiciones de trabajo.

Al mismo tiempo, el enfoque presentado no pretende cubrir todas las situaciones posibles. El caso de estudio se centra en un modelo relativamente acotado, implementado en un único lenguaje (R) y sobre un conjunto de datos específico, por lo que ciertos desafíos que enfrentan proyectos más grandes, multicéntricos o multilanguage quedan fuera del alcance de este ejercicio. La adopción de Docker y herramientas afines también supone familiarizarse con algunos conceptos básicos de contenedores y con la línea de comandos, lo que puede requerir un período de adaptación. A ello se suman consideraciones habituales en proyectos con datos reales, como las restricciones de confidencialidad o licencias, y condicionantes de infraestructura en ciertos entornos de cómputo, en particular cuando se trabaja con recursos gratuitos o limitados en la nube. Más que obstáculos insalvables, estos elementos señalan ámbitos en los que es necesario acompañar la adopción de contenedores con materiales de apoyo, capacitación y buenas prácticas de gestión de datos.

Como línea posible de extensión se podría profundizar la automatización mediante herramientas de gestión de dependencias (como renv en R), pruebas automatizadas y pipelines de integración continua que construyan y verifiquen imágenes con cada cambio relevante en el código. Finalmente, a nivel de comunidad, el uso de plantillas de proyectos, repositorios de ejemplo y actividades de formación podría facilitar una adopción progresiva de estos entornos reproducibles en grupos de investigación, cursos de grado y posgrado o instancias de docencia aplicada.

En conjunto, el caso presentado sugiere que los contenedores de software pueden desempeñar un papel central en la traducción de los principios de ciencia abierta y reproducibilidad computacional a prácticas concretas de trabajo con datos. Al fijar el entorno, automatizar los pasos clave del análisis y documentar de forma clara cómo ejecutar el flujo completo, se refuerza la posibilidad de verificar, compartir y extender los resultados de la investigación empírica. En síntesis, el uso de ambientes reproducibles con Docker constituye una práctica recomendable para fortalecer la solidez empírica, la transparencia y la portabilidad de los análisis en contextos académicos, a la vez que habilita escalabilidad y mantenimiento a lo largo del ciclo de vida de los proyectos.

A. Análisis de clúster

Mediante esta técnica se pretende clasificar individuos en grupos (Blanco et al. (2006), Kaufman and Rousseeuw (2009)). Para esto es necesario definir una métrica que cuantifique la similitud entre los individuos de la población. La variedad de métricas tienen distinta sensibilidad a los datos atípicos, así como difieren en el tipo de variables que pueden analizar entre cuantitativas y categóricas, nominales u ordinales.

Dado que se trabajará con variables categóricas la métrica elegida es el coeficiente de similitud de Jaccard, definido como el tamaño de la intersección dividido por el tamaño de la unión de los conjuntos de muestras (Jaccard (1901), Jaccard (1912)):

$$J(A,B) = \frac{|A \bigcap B|}{|A \bigcup B|} = \frac{|A \bigcap B|}{|A| + |B| - |A \bigcap B|}$$

En nuestra aplicación, las observaciones se representan como vectores de indicadores de categorías (presencia/ausencia), y el coeficiente de Jaccard mide cuánta coincidencia hay en esas categorías entre dos individuos. Así, individuos que comparten un conjunto similar de categorías (por ejemplo, sector, tamaño de empresa, tipo de vínculo) obtendrán valores altos de similitud y tenderán a ser asignados al mismo grupo. De esta forma luego del análisis individuos que compartan un conjunto de categorías estarán asignados a un mismo grupo (Real and Vargas (1996)). Analizando posteriormente las características de los individuos pertenecientes a cada grupo, se podrá llegar a una caracterización a partir de la cual compararlos.

Se opta por métodos de clusterización no jerárquicos debido al tamaño de las bases de datos a utilizar (Guénoche, Hansen, and Jaumard (1991)). Los métodos no jerárquicos se caracterizan por comenzar con una primera solución de clasificación, y luego de forma iterativa aplicar diferentes combinaciones para obtener particiones más homogéneas.

El método a utilizar es K-Medoides o partición en torno a los medoides (PAM) tal como presentan Kaufman and Rousseeuw (2009), donde cada clúster está representado por un medoide, es decir, un individuo representativo del grupo. Un medoide es un objeto representativo del conjunto de datos, tal que su diferencia promedio con los demás objetos es mínima. Con la cantidad k de grupos definida previamente se asignan k medoides dentro de los n individuos, los demás individuos son asignados al grupo del medoide más cercano. La asignación de medoides se actualiza iterativamente, minimizando la suma de distancias entre los individuos de cada grupo y su medoide. Como el arranque es aleatorio se recomienda aplicar más de una vez el método para llegar a conclusiones válidas.

Dado el tamaño de los datos a utilizar se optará por la implementación CLARA (Kaufman and Rousseeuw (2009)) de K-Medoides, que utiliza el enfoque de muestreo repetido. Este enfoque ha sido ampliamente utilizado y extendido en aplicaciones con grandes volúmenes de datos (Park and Jun (2009)). Considera una muestra fija de los datos y aplica en ella el algoritmo PAM encontrando el conjunto óptimo de medoides para la muestra. Se repite el proceso de muestreo y agrupación un número predefinido de veces para minimizar el sesgo del muestreo. La cantidad de muestras se definirá probando un valor creciente de las mismas y verificando a partir de cual los resultados se estabilizan.

Como se mencionó anteriormente, este tipo de métodos depende de predefinir la cantidad de grupos. Una alternativa para esto es maximizar la silueta promedio. La silueta es un estadístico que determina cuán similar es un individuo respecto al grupo al que pertenece en comparación a los demás grupos. Presenta un rango acotado en valor absoluto menor a la unidad. Un valor alto indica que el individuo es muy similar al grupo, y un valor bajo indica que el individuo es muy diferente al grupo. En términos funcionales puede expresarse como:

$$s(i) = \frac{b(i) - a(i)}{max\{a(i), b(i)\}}$$

donde $a(i)$ es la distancia promedio de la observación i a los objetos de su grupo y $b(i)$ es la distancia promedio de la observación i a los objetos de los demás grupos (distancia intra y entre grupos). A nivel

agregado se utiliza la silueta promedio para una cantidad dada de grupos, calculada como la media de las siluetas individuales. Para obtener una recomendación sobre la cantidad de grupos óptima en este tipo de algoritmos se puede comparar la silueta promedio de cada configuración. Aquella que obtenga un valor más alto será una buena candidata a cantidad de grupos.

Dadas las particularidades del año 2021 (INE (2020)) se decidió no tomarlo como año base para aplicar el análisis de cluster. Los datos de 2006 fueron descartados para este análisis por la constatación de datos faltantes en la variable referente al tamaño de la empresa, considerada importante para diferenciar los grupos. Los análisis de clúster realizados para 2011 y 2019 muestran una mejor separación de grupos en el año 2011, con una silueta promedio más elevada y sin problemas relevantes de solapamiento entre grupos específicos. Se eligen por lo tanto los datos del año 2011 para realizar el análisis de cluster y servir de año base para los análisis posteriores.

Referencias

ARCA-DPSS. s.f. "Manual de Ciencia Abierta — Capítulo Docker." https://arca-dpss.github.io/manual-open-science/docker-chapter.html.

Bache, Stefan Milton, and Hadley Wickham. 2025. *Magrittr: A Forward-Pipe Operator for r*. https://CRAN.R-project.org/package=magrittr.

Baker, Monya. 2016. "1,500 Scientists Lift the Lid on Reproducibility." Nature Publishing Group UK London.

Bechhofer, Sean, Iain Buchan, David De Roure, Paolo Missier, John Ainsworth, Jiten Bhagat, Philip Couch, et al. 2013. "Why Linked Data Is Not Enough for Scientists." *Future Generation Computer Systems* 29 (2): 599–611.

Blanco, Jorge, G Camano, L Nalbarte, and R Alvarez-Vaz. 2006. "Introducción Al análisis Multivariado." *IESTA. Montevideo* 23.

Boettiger, Carl. 2015. "An Introduction to Docker for Reproducible Research." *ACM SIGOPS Operating Systems Review* 49 (1): 71–79.

Christensen, Garret, Jeremy Freese, and Edward Miguel. 2019. *Transparent and Reproducible Social Science Research: How to Do Open Science*. University of California Press.

Fairbanks, Mark. 2025. *Tidytable: Tidy Interface to Data.table*. https://CRAN.R-project.org/package=tidytable.

Fecher, Benedikt, and Sascha Friesike. 2013. "Open Science: One Term, Five Schools of Thought." In *Opening Science: The Evolving Guide on How the Internet Is Changing Research, Collaboration and Scholarly Publishing*, 17–47. Springer International Publishing Cham.

Guénoche, Alain, Pierre Hansen, and Brigitte Jaumard. 1991. "Efficient Algorithms for Divisive Hierarchical Clustering with the Diameter Criterion." *Journal of Classification* 8 (1): 5–30.

Hastie, Trevor. 2009. "The Elements of Statistical Learning: Data Mining, Inference, and Prediction." Springer.

Hester, Jim, Hadley Wickham, and Jennifer Bryan. 2025. *Vroom: Read and Write Rectangular Text Data Quickly*. https://CRAN.R-project.org/package=vroom.

Hosmer Jr, David W, Stanley Lemeshow, and Rodney X Sturdivant. 2013. *Applied Logistic Regression*. John Wiley & Sons.

INE. 2020. "Metodología de La ECH No Presencial." Montevideo, Uruguay: Instituto Nacional de Estadística, División Estadísticas Sociodemográficas.

Jaccard, Paul. 1901. "Étude Comparative de La Distribution Florale Dans Une Portion Des Alpes Et Du Jura." *Bulletin de La Société Vaudoise Des Sciences Naturelles* 37 (142): 547–79. https://doi.org/10.5169/seals-266450.

———. 1912. "The Distribution of the Flora in the Alpine Zone." *New Phytologist* 11 (2): 37–50. https://doi.org/10.1111/j.1469-8137.1912.tb05611.x.

James, Gareth, Daniela Witten, Trevor Hastie, and Robert Tibshirani. 2018. *An Introduction to Statistical Learning: With Applications in r*. Springer. https://doi.org/10.25334/q4ht55.

Kaufman, Leonard, and Peter J Rousseeuw. 2009. *Finding Groups in Data: An Introduction to Cluster Analysis*. John Wiley & Sons.

Machado, Fabricio. 2023. "Aproximación a La Distribución Funcional Del Ingreso Entre Trabajadores. Un Análisis a Partir de Técnicas Multivariadas Para El Período 2006–2021 En Uruguay." Serie Documentos de Investigación Estudiantil DIE 02/2023. Montevideo, Uruguay: Instituto de Economía, Facultad de Ciencias Económicas y de Administración, Universidad de la República.

McFadden, Daniel. 1972. "Conditional Logit Analysis of Qualitative Choice Behavior."

McLachlan, Geoffrey J. 2005. *Discriminant Analysis and Statistical Pattern Recognition*. John Wiley & Sons.

Müller, Kirill. 2020. *Here: A Simpler Way to Find Your Files*. https://CRAN.R-project.org/package=here.

Park, Hae-Sang, and Chi-Hyuck Jun. 2009. "A Simple and Fast Algorithm for K-Medoids Clustering." *Expert Systems with Applications* 36 (2): 3336–41. https://doi.org/10.1016/j.eswa.2008.01.039.

Peng, Roger D. 2011. "Reproducible Research in Computational Science." *Science* 334 (6060): 1226–27.

Pohar, Maja, Mateja Blas, and Sandra Turk. 2004. "Comparison of Logistic Regression and Linear Discriminant Analysis: A Simulation Study." *Metodoloski Zvezki* 1 (1): 143.

Potdar, Amit M, DG Narayan, Shivaraj Kengond, and Mohammed Moin Mulla. 2020. "Performance Evaluation of Docker Container and Virtual Machine." *Procedia Computer Science* 171: 1419–28.

R Core Team. 2025. *R: A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation for Statistical Computing. https://www.R-project.org/.

Rad, Babak Bashari, Harrison John Bhatti, and Mohammad Ahmadi. 2017. "An Introduction to Docker and Analysis of Its Performance." *International Journal of Computer Science and Network Security (IJCSNS)* 17 (3): 228.

Real, Raimundo, and Juan M Vargas. 1996. "The Probabilistic Basis of Jaccard's Index of Similarity." *Systematic Biology* 45 (3): 380–85.

Sandve, Geir Kjetil, Anton Nekrutenko, James Taylor, and Eivind Hovig. 2013. "Ten Simple Rules for Reproducible Computational Research." *PLoS Computational Biology* 9 (10): e1003285.

Stodden, Victoria, Friedrich Leisch, and Roger D Peng. 2014. *Implementing Reproducible Research*. Vol. 546. Crc Press Boca Raton, FL.

Stodden, Victoria, Marcia McNutt, David H Bailey, Ewa Deelman, Yolanda Gil, Brooks Hanson, Michael A Heroux, John PA Ioannidis, and Michela Taufer. 2016. "Enhancing Reproducibility for Computational Methods." *Science* 354 (6317): 1240–41.

Stone, Mervyn. 1974. "Cross-Validatory Choice and Assessment of Statistical Predictions." *Journal of the Royal Statistical Society: Series B (Methodological)* 36 (2): 111–33.

Venables, W. N., and B. D. Ripley. 2002. *Modern Applied Statistics with s*. Fourth. New York: Springer. https://www.stats.ox.ac.uk/pub/MASS4/.

———. 2025. *Nnet: Feed-Forward Neural Networks and Multinomial Log-Linear Models*. https://CRAN.R-project.org/package=nnet.

Wilkinson, Mark D, Michel Dumontier, IJsbrand Jan Aalbersberg, Gabrielle Appleton, Myles Axton, Arie Baak, Niklas Blomberg, et al. 2016. "The FAIR Guiding Principles for Scientific Data Management and Stewardship." *Scientific Data* 3 (1): 1–9.