FACULTAD DE CIENCIAS AGRARIAS
UNIVERSIDAD NACIONAL DE ROSARIO

Paquete de R y aplicación Web para el análisis de datos provenientes de ensayos multiambientales

JULIA ANGELINI

TRABAJO FINAL PARA OPTAR AL TITULO DE ESPECIALISTA EN BIOINFORMÁTICA

DIRECTOR: Gerardo Cervigni
CO-DIRECTOR: Marcos Prunello

AÑO: 2020

Paquete de R y aplicación Web para el análisis de datos provenientes de ensayos multiambientales

Julia Angelini

Licenciada en Estadística – Universidad Nacional de Rosario

Este Trabajo Final es presentado como parte de los requisitos para optar al grado académico de Especialista en **Bioinformática**, de la Universidad Nacional de Rosario y no ha sido previamente presentada para la obtención de otro título en ésta u otra Universidad. El mismo contiene los resultados obtenidos en investigaciones llevadas a cabo en **el Centro de Estudios Fotosintéticos y Bioquímicos (CEFOBI)**, durante el período comprendido entre **los años 2017 y 2020**, bajo la dirección del **Dr. Gerardo Cervigni y Mgs. Marcos Prunello**.

Nombre y firma del autor

Nombre y firma del Director

Nombre y firma del Co - Director

Defendida: _____ de 20____.

Agradecimientos

En este trabajo final, directa o indirectamente, participaron muchas personas a las que les quiero agradecer.

En primer lugar al Dr. Gerardo Cervigni por confiar en mí y permitirme explorar el mundo de la Bioinformática durante mi tesis doctoral, para que hoy sea parte de mis conocimientos. Al Mgs. Marcos Prunello por acompañarme en el desarrollo del trabajo final, por su dedicación y sus consejos.

Todo esto nunca hubiera sido posible sin el apoyo y el cariño de mis padres, de mi hermano, de Segundo y Kalita. Siempre estuvieron a mi lado, las palabras nunca serán suficientes para agradecerles.

A mis compañeras Jor y Lu, por su ayuda y por compartir excelentes momentos.

A Gaby y Euge mis compañeras de CEFOBI, gracias a ustedes este camino ha sido mas fácil!

A mis amigos, por estar siempre presentes.

Muchas gracias a todos!

Abreviaturas y Símbolos

EMA: ensayos multiambientales **IGA**: interacción genotipo ambiente **NCOI**: interacción sin cambio de rango, del inglés *no crossover interaction* **COI**: interacción con cambio de rango, del inglés *crossover interaction* **ANOVA**: análisis de la variancia, del inglés *analysis of variance* **AMMI**: modelo de los efectos principales aditivos y interacción multiplicativa, del inglés *Additive Main effects and Multiplicative Interaction* **ACP**: análisis de componentes principales **SREG**: modelo de regresión por sitio, del inlés *Site Regression model* **DVS**: descomposición de valores singulares **GNU**: *General Public Licence* **CRAN**: *Comprehensive R Archve Network* **EM**: maximización de la esperanza, del inglés *Expectation-Maximization*

Resumen

Las variedades mejoradas son el resultado del trabajo de desarrollo genético llevado a cabo en los programas de fi

tomejoramiento, los cuales se extienden a lo largo de varios años y requieren cuantiosas inversiones. En etapas avanzadas, los ensayos multiambientales (EMA), que comprenden experimentos en múltiples ambientes, son herramientas fundamentales para incrementar la productividad y rentabilidad de los cultivos. La vigencia comercial de las variedades puede extenderse durante varias décadas, por lo que su elección es crítica para que el productor evite pérdidas ecoómicas por malas campañas y el suministro al mercado sea constante. Consecuentemente, un análisis adecuado de la información de los EMA es indispensable para que el programa de mejoramiento de los cultivos sea efie

caz. Los programas informáticos se han convertido, hoy en día, en una herramienta esencial par el análisis de datos. Actualmente, R es uno de los programas más utilizados debido a su potencia y a su distribución como software libre. Actualmente existen numerosos paquetes de R lo cual provoca que no sea sencillo encontrar un paquete que sea útil para un determinado

propósito sino que se debe recurrir a varios de ellos. Frecuentemente, los mejoradores no tienen un manejo fluido de paquetes estadísticos que permitan entender la dinámica del problema. En este sentido el paquete Shiny que permite crear una interfaz gráfica entre R y el usuario, permitiendo acercar la potencia de R a todo tipo de usuarios. En el presente trabajo se desarrolló un paquete de R que permita analizar los datos provenientes de EMA para aquellos usuarios que tengan manejo del lenguaje de programación y una interfaz en Shiny que permita realizar los principales análisis del paquete sin necesidad de programar.

Palabras Clave:

Abstract

Keywords:

Índice general

Capítulos **Página**

Índice de figuras

Índice de tablas

Lista de tareas pendientes

This is a note . . . . . see 1.0 at p. 1

Capítulo 1

Introducción

This is a note

A lo largo de la historia de la agricultura, el hombre ha desarrollado el mejoramiento vegetal en forma sistemática y lo ha convertido en un instrumento esencial para incrementar la producción agrícola en términos de cantidad, calidad y diversidad.

El fitomejoramiento, en un sentido amplio, busca alterar la frecuencia alélica de los genes para obtener cultivares genéticamente superiores, adaptados a condiciones específicas, con mayor rendimiento y mejor calidad que las variedades nativas o criollas (Allard, 1967). En otras palabras, su objetivo es desarrollar plantas cuyo patrimonio hereditario esté de acuerdo con las condiciones, necesidades y recursos de todos aquellos que producen, transforman y consumen productos vegetales.

Las variedades mejoradas son el resultado del trabajo de desarrollo genético llevado a cabo en los programas de fitomejoramiento, los cuales se extienden a lo largo de varios años y requieren cuantiosas inversiones. Generalmente, en etapas tempranas de estos programas existe un gran número de genotipos experimentales con pocos antecedentes de evaluación; mientras que en etapas posteriores se trabaja con pocos genotipos que se evalúan con más repeticiones y en más ambientes/años. Estos ensayos multiambientales (EMA) son herramientas fundamentales para evaluar la productividad para así asegurar la rentabilidad de los cultivos.

Como consecuencia de que los EMA se llevan a cabo en múltiples ambientes/años, la aparición de la interacción genotipo × ambiente (IGA) es inevitable debido a las variaciones en las condiciones climáticas y de suelo. La IGA es considerada por los fitomejoradores como el principal factor factor limitante de respuesta a la selección y, en general, de la eficiencia de los programas de mejoramiento, por provocar respuestas altamente variables en los diferentes ambientes (Crossa et al., 1990; Cruz Medina, 1992; Kang y Magari, 1996).Gauch y Zobel (1996) explicaron la importancia de IGA como: “Si no hubiera interacción, una sola variedad de trigo (*Triticum aestivum* L.) o de maíz (*Zea mays* L.) o cualquier otro cultivo rendiría al máximo en todo el mundo, y además las variedades se

deberían evaluar en un sólo lugar para proporcionar resultados universales".

Peto (1982) ha distinguido las interacciones cuantitativas, conocida también como interacción sin cambio de rango o *no crossover* (NCOI), de las interacciones cualitativas, denominada también con cambio de rango (COI) o *crossover* (Cornelius et al., 1996). Cuando dos genotipos X e Y tienen una respuesta diferencial en dos ambientes diferentes, y su ordenación permanece sin cambios, se dice que la IGA es del tipo simple o *no crossover* (Figura 1.1(A)) es es compleja o *crossover* cuando hay cambios en el orden de los genotipos (Figura 1.1(B)), y. finalmente, es inexistente cuando los genotipos responden de manera similar en ambos ambientes (Figura 1.1(C)).

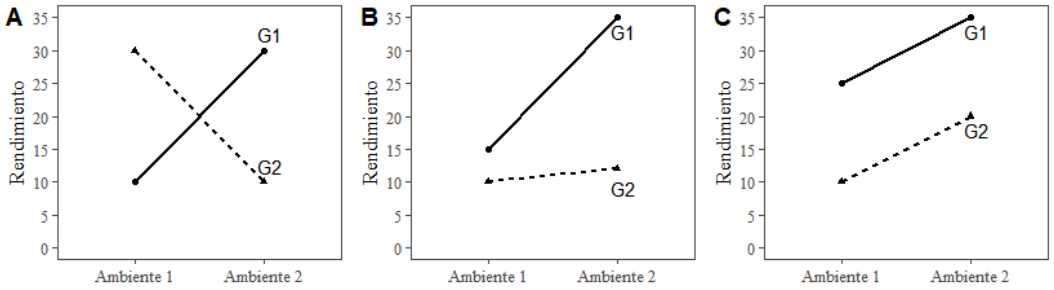


Figura 1.1: Representación gráfica de tipos de IGA: (A)IGA no crossover, (B) IGA crossover y (C) no IGA

Entre las implicancias negativas de la IGA en los programas de mejoramiento se encuentra el impacto negativo sobre la heredabilidad, cuanto menor sea la heredabilidad de un caracter, mayor será la dificultad para mejorar ese caracter mediante la selección. Como consecuencia, la información sobre la estructura y la naturaleza de la IGA es particularmente útil para determinar si se deben desarrollar cultivares con adaptación amplia o específica (Bridges, 1989). La decisión sobre qué tipo de estrategia seguir, involucra considerar y analizar conceptos como regiones ecológicas, ecotipos y mega-ambientes (Kang et al., 2004).

La vigencia comercial de las variedades puede extenderse durante varias décadas, por lo que su elección es crítica para que el productor evite pérdidas económicas por malas campañas y el suministro al mercado sea constante. Consecuentemente, un análisis adecuado de la información de los EMA es indispensable para que el programa de mejoramiento genético de los cultivos sea eficaz. El rendimiento medio en los ambientes es un indicador suficiente del rendimiento genotípico solo en ausencia de IGA (Yan y Kang, 2003). Sin embargo, la aparición de IGA es inevitable y no basta con la comparación de las medias de los genotipos, sino que se debe recurrir a una metodología estadística más aporopiada. La metodología estadística más difundida para analizar los datos provenientes de EMA

se basa en modificaciones de los modelos de regresión, análisis de variancia (ANOVA) y técnicas de análisis multivariado.

Particularmente, para el estudio de la IGA y los análisis que de ella se derivan, dos modelos multiplicativos han aumentado su popularidad entre los fitomejoradores, especialmente como una herramienta de análisis gráfico: el modelo de los efectos principales aditivos e interacción multiplicativa (*Additive Main effects and Multiplicative Interaction*, AMMI) (Kempton 1984, Gauch, 1988), y el de regresión por sitio (*Site Regression model*, SREG) (Cornelius et al., 1996; Crossa y Cornelius, 1997 y 2002). Estos modelos combinan un ANOVA con la descomposición de valores singulares (DVS) o un análisis de componentes principales (ACP) sobre la matriz residual de ANOVA. En SREG, el ANOVA se realiza sobre el eefecto principal o ambiental (A), mientras que en AMMI el ANOVA se realiza sobre los efectos principales de genotipos (G) y A. Si bien a través del modelo AMMI se obtiene el gráfico biplot Genotipeo × Ambiente (GA), el cual es usado para explorar patrones puramente atribuibles a los efectos la IGA, para el modelo SREG el biplot GGE es usado para explorar simultáneamente patrones de variación conjunta de G e IGA (Yan y Hunt (2002)).

Una limitación importante de la mayoría de las propuestas del análisis de EMA es que requieren que el conjunto de datos esté completo. Aunque los EMA están diseñados para que la totalidad de los genotipos se evalúen en todos los ambientes, las tablas de datos genotipo × ambiente completas son poco frecuentes (no todos los genotipos se encuentran en todos los ambientes). Esto ocurre, por ejemplo, debido a errores de medición o pérdidas de plantas por animales, inundaciones o problemas durante la cosecha, además de la dinámica propia de la evaluaciones en las que se incorporan y se descartan genotipos debido a su pobre desempeño (Hill y Rosemberg, 1985). En estos casos, entre las posibles soluciones para tratar con una tabla de datos incompleta se encuentran: (i) el uso de un subconjunto completo de datos, eliminando aquellos genotipos que tienen valores faltantes (Ceccarelli et al., 2007, Yan et al., 2011), (ii) completar datos faltantes con la media ambiental, o (iii) imputar los datos faltantes con valores estimados utilizando modelos multiplicativos (Kumar et al., 2012).

En este contexto, el análisis de datos provenientes de EMA requiere metodología estadística más sofisticada cuyas rutinas informáticas se encuentran disponibles en programas desarrollados por diferentes empresas. Esto genera el inconveniente de tener que disponer de todos los programas necesarios para los distintos análisis, atender los requerimientos de formatos de datos usados por cada uno de ellos, y comprender los diversos tipos de salidas en las que se ofrecen los resultados obtenidos. Además, algunos procedimientos, especialmente aquellas metodologías recientes, no se encuentran disponibles, y los costos de las licencias de dichos programas resultan muy elevados.

Desarrollado por *The R Foundation for Statistical Computing* el programa R se trata de un proyecto colaborativo de uso libre y distribuido bajo los términos de la *General Public Licence* (GNU). R, surge como resultado de la implementación del lenguaje S uno de los más utilizados en investigación por la comunidad estadística. A diferencia de los programas estadísticos utilizados frecuentemente, R es un lenguaje de programación y no dispone de una interfaz gráfica en la cual los distintos análisis se realizan por menús, generando cierta dificultad en su uso para aquéllos que no se encuentran familiarizados con el lenguaje. Sin embargo, por ser un software libre, permite a los usuarios definir sus propias funciones dando lugar a mayores posibilidades respecto del manejo y análisis de los datos disponibles. Si bien la versión básica del programa dista mucho de ser amigable, RStudio, un entorno de desarrollo integrado (IDE) gratuito y de código abierto para R, permite una interacción más fluida con el programa, actuando como una interfaz amigable con el usuario. Al formar parte de un proyecto colaborativo, R promueve el hecho de que los usuarios compartan con la comunidad las funciones creadas por ellos, es decir que está en continuo desarrollo y actualización. A menudo, no resulta sencillo reutilizar una función creada por algún usuario, por ello, se ha introducido la posibilidad de crear paquetes (*package*) o librerías. Éstas son una colección de objetos desarrollados y organizados siguiendo un protocolo fijo, lo cual garantiza un soporte mínimo para el usuario así como la ausencia de errores (de sintaxis) en la programación.

Actualmente, R cuenta con 14 paquetes básicos y 29 recomendados para su funcionamiento que se instalan automaticamente en él, tal como *base* o *stats*. Entre los paquetes que extienden las funciones básicas de R se encuentran, *plyr*, *lubridate*, *reshape2* y *stringr* para la manipulación de los datos; *ggplot2* y *rgl* para la visualización; *knitr* y *xtable* para la presentación de resultados; entre otros. La lista completa de los paquetes oficiales puede consultarse en CRAN[1], que contaba con más de 14.000 paquetes disponibles hasta junio de 2019. Esta gran variedad de paquetes es una de las razones de la gran difusión de R, ya que cada usuario puede tratar su problematica utilizando un paquete desarrollado por otro usuario. Además de los paquetes oficiales, existen otros que pueden instalarse desde repositorios como Github. Sin embargo, no es sencillo encontrar un paquete con todas las tutinas necesarias para el análisis, sino que debe recurrirse a varios de ellos.

Con frecuencia, los mejoradores usan programas con interfaz gráfica para realizar el análisis estadístico, sin necesidad del manejo de un lenguaje de programación. En el año 2012 se creó el paquete *Shiny* de R que facilita el desarrollo de aplicaciones Web utilizando R, acercando la potencia de R a todo tipo de usuarios, sin tener que programar.

El objetivo del presente trabajo es considerar R para: (i) crear un paquete de que

[1]CRAN (Comprehensive R Archve Network) es el repositorio oficial de paquetes de R, el lugar donde se publican las nuevas versiones del programa, etc. Contiene la lista completa de paquetes oficiales. https://cran.r-project.org/web/packages/available_packages_by_name.html

integre las funciones que permitan analizar los datos provenientes de EMA, incluyendo metodologías recientemente publicadas,no disponibles en R; (ii) usando Shiny crear una interfaz gráfica para agilizar el uso del paquete.

Elousa, Paula. 2009. "¿EXISTE VIDA MÁS ALLÁ DEL SPSS? DESCUBRE R." Revista Psicothema 21 (4): 652–55. http://www.psicothema.com/psicothema.asp?id=3686.

FSF. 2019. "¿Qué Es El Software Libre?" Free Software Foundation. https://www.fsf.org/es/recurs
es-el-software-libre.

Capítulo 2

Objetivos

2.1. Objetivo general

Construir un paquete para el programa R con funciones necesarias para analizar datos provenientes de EMA y desarrollar una Shiny Web APP que permita a los usuarios utilizar las funciones del paquete sin necesidad de conocer el lenguaje de programación utilizado.

2.2. Objetivos específicos

- Mostrar un flujo de trabajo reproducible para la construcción de paquetes de R.
- Modificar funciones existentes para el análisis de datos provenientes de EMA de manera que sean más flexibles que las actuales.
- Incorporar metodología recientemente publicada en el paquete de R.
- Desarrollar una Shiny Web APP para analizar los datos provenientes de EMA, presentando un flujo de trabajo reproducible.

Capítulo 3

Métodos

3.1. Métodos estadísticos

3.1.1. Modelo AMMI y SREG

El modelo AMMI propuesto por Zobel et al. (1988) es un modelo multiplicativo en el cual se expresa el fenotipo de un genotipo en un ambiente de la siguiente forma:

$$y_{ij} = \mu + G_i + A_j + \sum_{k=1}^{q} \lambda_k \alpha_{ik} \gamma_{jk} \qquad i = 1, ..., g; \;\; j = 1, ..., a; \;\; q = min(g-1, a-1)$$

donde

- y_{ij} es el caracter fenotípico evaluado (rendimiento o cualquier otro caracter de interes) del i-ésimo genotipo en el j-ésimo ambiente,
- μ es la media general,
- G_i es el efecto del i-ésimo genotipo,
- A_j es el efecto del j-ésimo ambiente
- $\sum_{k=1}^{q} \lambda_k \alpha_{ik} \gamma_{jk}$ es la sumatoria de componentes multiplicativas utilizadas para modelar la IGA. Siendo, λ_k el valor singular para la k-ésima componente principal (PC) α_{ik} y γ_{jk} son los scores de las PC para el i-ésimo genotipo y el j-ésima ambiente para la k-ésima componente, respectivamente;

En cambio, el modelo SREG (Cornelius et al., 1996; Crossa y Cornelius, 1997 y 2002) expresa el fenotipo de un genotipo en un ambiente en función del efecto ambiente aditivo y los efectos genotipo e interacción agrupados y en forma multiplicativa:

$$y_{ij} = \mu + A_j + \sum_{k=1}^{q} \lambda_k \alpha_{ik} \gamma_{jk} \qquad i = 1, ..., g; \;\; j = 1, ..., a; \quad q = min(g-1, a-1)$$

Los parámetros multiplicativos, tanto en el modelo AMMI como en el SREG, se estiman por medio de la DVS de la matriz que contiene los residuos del modelo aditivo luego de ajustar por mínimos cuadrados el modelo de efectos principales. Generalmente los dos

primeros términos multiplicativos son suficientes para explicar los patrones de interacción; la variabilidad remanente se interpreta como ruido.

Para visualizar el efecto de IGA o conjuntamente el efecto de G y IGA se proponen los gráficos biplots GE y GGE (proveniente de las siglas en inglés G de Genotipe y GE de Genotipe-Environment) (Yan et al., 2000) respectivamente. El concepto del biplot fue presentado por Gabriel (1971), que consiste en una representación de las filas (individuos) y las columnas (variables) de una matriz de datos en un mismo gráfico. Éstos biplots, son herramientas poderosas definidas para el análisis e interpretación de la estructura de datos provenientes de ensayos multiambientales utilizados en los programas de mejoramiento (Ebdon y Gauch, 2002; Samonte et al., 2004; Yan et al., 2000; Zobel et al., 1988).

El biplot GE ayuda a interpretar la variación producida por los efectos de la IGA. Por otro lado, el biplot GGE permite investigar la diferenciación de mega-ambientes entre los ambientes en estudio, seleccionar cultivares superiores en un mega-ambiente dado y seleccionar los mejores ambientes de evaluación para analizar las causas de la interacción GA. Se define como mega-ambiente a un grupo de ambientes en donde los cultivares de mejor desempeño son los mismos.

3.1.2. Modelo AMMI robusto

El modelo AMMI, en su forma estándar, asume que no hay valores atípicos en el conjunto de datos. La presencia de observaciones atípicas es más una regla que una excepción cuando se consideran datos agronómicos debido a errores de medición, algunas plagas / enfermedad que puede influir en algunos genotipos resultando por ejemplo en un rendimiento inferior al esperado en un ambiente, o incluso debido a alguna característica inherente de los genotipos que se evalúan.

Rodrigues et al. (2015) proponen una generalización robusta del modelo AMMI, que se puede obtener en dos etapas: primero ajustar la regresión robusta basada en el estimador M-Huber (Huber, 1981) para reemplazar el ANOVA; y luego utilizar un procedimiento DVS / PCA robusto para reemplazar la DVS estándar. En la segunda etapa, consideraron varios métodos dando lugar a total de cinco modelos robustos llamados: R-AMMI, H-AMMI, G-AMMI, L-AMMI, PP-AMMI.

El empleo de la versión robusta del modelo AMMI puede ser extremadamente útil debido a que una mala representación de genotipos y ambientes en los biplots puede dar como resultado un mala decisión con respecto a qué genotipos seleccionar para un conjunto dado de ambientes (Gauch1997,Yanetal2000). A su vez, la elección de los genotipos incorrectos pueden provocar grandes pérdidas en términos de producción de rendimiento. Los biplots obtenidos de los modelos robustos mantienen las características e interpretación estándar del modelo AMMI clásico (Rodrigues et al. (2015)).

3.1.3. Métodos de imputación

Una limitación importante que presentan los modelos multiplicativos descriptos previamente es que requieren que el conjunto de datos este completo, es decir no admiten valores perdidos. Aunque los EMA están diseñados para que todos los genotipos se evalúen en todos los ambientes, la presencia de valores faltantes es muy común debido a errores de medición o pérdidas de plantas por animales, inundaciones o problemas durante la cosecha, además de la dinámica propia de la evaluaciones en las que se incorporan y se descartan genotipos debido a su pobre desempeño (Hill y Rosemberg, 1985)

Se han propuesto numerosas metodologías para superar el problema de valores ausentes en el conjunto de datos, entre las cuales se encuentran:

- EM-AMMI: Gauch y Zobel (1990) desarrollaron un procedimiento iterativo que utiliza el algoritmo de maximización de la esperanza (EM, del inglés *Expectation-Maximization*) incorporando el modelo AMMI. **Este método reemplaza... (poner como abajo)**. Dependiendo del número de términos multiplicativos empleados, el método de imputación puede denominarse EM-AMMI0, EM-AMMI1, etc. (Gauch y Zobel 1990).
- EM-SVD: Perry (2009a) propone un método de imputación que combina el algoritmo EM con DVS. Este método, en forma similar al anterior, reemplaza los valores faltantes de una matriz $G \times E$ inicialmente por valores arbitrarios para obtener una matriz completa, y luego la DVS se calcula iterativamente en esa matriz.
- EM-PCA:
- Gabriel Eigen: Arciniegas-Alarcón et al. (2010) propuso un método de imputación que combina regresión y aproximación de rango inferior usando DVS. El método reemplaza inicialmente las celdas faltantes por valores arbitrarios, y posteriormente el las imputaciones se refinan a través de un esquema iterativo que define una partición de la matriz para cada valor que falta a su vez y utiliza una regresión lineal de columnas (o filas) para obtener el nueva imputación. En esta regresión, la matriz de diseño se aproxima por una matriz de menor rango usando la DVS.
- WGabriel Eigen:

3.2. Paquete de R

Una librería o paquete (*package*) es una colección de objetos creados y organizados siguiendo un protocolo fijo que garantiza la ausencia de errores (de sintaxis) en la programación. Éstos son las unidades fundamentales de un código reproducible de R ya que

incluyen funciones reutilizables, la documentación que describe cómo usar cada una de ellas y, además pueden incluir datos de ejemplo.

Los pasos necesarios para la creación de un paquete son:

- Creación del esqueleto del paquete.
- Inclusión de los objetos que contendrá el paquete (funciones y/o datos).
- Redacción de la documentación.
- Compilación del paquete en Linux y creación de la versión para Windows.
- Instalación.
- Prueba y publicación.

Para la cración del paquete se utilizan numerosas funciones incluidas en el paquete *devtools* que permiten realizar diversos aspectos del desarrollo de paquetes. Por lo tanto, antes de comenzar a crear el paquete se deben instalar el mismo como se indica a continuación:

```
# Instalar el paquete devtools desde CRAN
install.packages("devtools")
# Instalar el paquete devtools desde GitHub:
install_github("r-lib/devtools")
```

3.2.1. Esqueleto y estructura del paquete

Para crear la estructura del paquete se utiliza la función create_package(). El principal y único argumento requerido por dicha función es el directorio donde el nuevo paquete se alojará. Por lo general, si el directorio se llama "geneticae", entonces el nombre del paquete también será "geneticae":

```
# Cargar la libreria devtools
library(devtools)
# Crear el paquete geneticae
create_package("C:/Users/Julia/Desktop/geneticae")
```

El resultado de ejecutar la función create_package() es un paquete con los siguientes componentes:

- Un directorio R/.
- DESCRIPTION, un archivo simple cuyo objetivo es almacenar metadatos importantes sobre el paquete, epecifica el título, la versión del mismo, identifica al autor y brinda un mail de contacto, una breve descripción del paquete, la lista de los paquetes que el paquete creado necesita para funcionar, la licencia, entre otros.
  El contenido básico en un archivo DESCRIPTION es:

```
Package: geneticae
```

```
Title: What the Package Does (One Line, Title Case)
Version: 0.0.0.9000
Authors@R:
    person(given = "First",
           family = "Last",
           role = c("aut", "cre"),
           email = "first.last@example.com",
           comment = c(ORCID = "YOUR-ORCID-ID"))
Description: What the package does (one paragraph).
License: What license it uses
Encoding: UTF-8
LazyData: true
```

- Un archivo NAMESPACE

Estas carpetas se iran modificando a medida que el paquete se vaya creando. También puede incluir un archivo de proyecto de RStudio pkgname.Rproj, que hace que el paquete sea fácil de usar con RStudio; .Rbuildignore enumera los archivos que se necesitan, pero que no deben incluirse al compilar el paquete R desde la fuente; .gitignore anticipa el uso de Git. Se crearán, a través de devtools, las siguientes carpetas: data/ y man/.

3.2.2. Creación de funciones y conjuntos de datos

Una vez creada la estructura del paquete se deben incluir las funciones que el mismo contendrá. Cada una de ellas debe ser guardada en un archivo de extensión .R, en el subdirectorio R/. Para ello, se utiliza la función use_r() la cual crea y/o abre un script de la carpeta R/.

Una vez creada una función, se realizan pruebas para asegurar que el código realice lo que realmente se desea utilizando la función load_all() que simula el proceso de construcción, instalación y conexión del paquete. Permite que las funciones creadas estén disponible rápidamente para uso interactivo, del mismo modo que si se hubiera construido e instalado el paquete y luego cargada en la sesión de R a través de la función library(geneticae).

Muy frecuentemente se utilizan funciones que se encuentran disponibles en otros paquetes, para ello se utiliza la función use_package() que agrega el paquete al archivo DESCRIPTION.

A menudo es útil incluir datos en un paquete a fin de proporcionar ejemplos de aplicaciones de las funciones incluidas en él. Ellos se almacenan en el directorio data/, siendo cada archivo un .RData que sólo contiene un objeto. Para esto, se utiliza la función

usethis::use_data(). Notar que el archivo DESCRIPTION creado con la función create_package(), mencionada anteriormente, contiene el campo LazyData: true, lo cual genera que los conjuntos de datos no ocupen memoria hasta que sean usados.

3.2.3. Documentación

La documentación es uno de los aspectos más importantes del código, sin ella, los usuarios no sabrán cómo usar el paquete. Existen múltiples formas de documentar un paquete, la forma estándar es escribir archivos .Rd en la carpeta man, los cuales utilizan una sintaxis personalizada, basada en LaTeX. Sin embargo, el paquete *roxygen2*, utilizado en este trabajo, convierte los comentarios con formato especial en archivos .Rd. Esta última proporciona una serie de ventajas sobre la estándar:

- El código y la documentación son adyacentes, de modo que cuando el código se modifique, será fácil actualizar la documentación.
- Inspecciona dinámicamente los objetos que está documentando, para que pueda agregar automáticamente los datos que de otra forma se deben escribir a mano.
- Resume las diferencias en la documentación de los métodos S3 y S4, los genéricos y las clases, por lo que necesita aprender menos detalles.

Además de generar archivos .Rd, *roxygen2* también creará el archivo NAMESPACE. El flujo de trabajo para crear la documentación con el paquete *roxygen2* es el siguiente:

- Agregar comentarios a los archivos .R, los cuales comienzan con #'y preceden a una función. La primera oración se convierte en el título y el segundo párrafo es una descripción de la función. Seguidamente, las funciones son documentadas, la mayoría de las funciones tienen tres etiquetas: @param, @examples y @return.
    - @param describe los parámetros de la función, indica de que clase es el parámetro y para que sirve.
    - @examples proporciona un código ejecutable que muestra cómo usar la función en la práctica.
    - @return describe el resultado de la función.
- Ejecutar devtools::document() para convertir los comentarios de roxygen en archivos .Rd.

Roxygen2 permite utilizar la descripción de los parámetros de otras funciones usando @inheritParams. Esta documentará los parámetros que no están documentados en la función actual, pero que si lo están en la función fuente. La fuente puede ser una función en el paquete actual, vía @inheritParams function, u otro paquete, vía @inheritParams package::function.

A diferencia de las funciones que son documentadas directamente, para los objetos en data/, se debe crear un archivo y guardarlo en el directorio R/.

Viñetas

A diferencia de la documentación, en la cual se detalla como se utiliza cada una de las funciones del paquete, una viñeta es una descripción el problema que el paquete está diseñado para resolver y muestra al lector cómo resolverlo.

Muchos de los paquetes existentes tienen viñetas la cual puede ser buscada mediante la función browseVignettes("packagename"). Cada viñeta proporciona el archivo fuente original, una página HTML o PDF y un archivo de código R. Las viñetas de paquetes que no han sido instalados pueden ser consultados en su página de CRAN, por ejemplo para el paquete *dplyr*: `http://cran.r-project.org/web/packages/dplyr`.

Las Viñetas se pueden construir de diversas formas, en este trabajo se utiliza *devtools* para crear la estructura de la misma y luego se añade el contenido que se desee en formato Rmarkdown. Rmarkdown permite combinar, texto plano, bloques de código y salidas.

Para crear la viñeta, se utiliza usethis::use_vignette("my-vignette"). La misma crea un directorio vignettes/, agrega las dependencias necesarias a DESCRIPTION y redacta la viñeta. Las tres componentes fundamentales de la misma son las siguientes:

- El bloque inicial de metadatos, que contiene la siguiente información:

```
---
title: "Vignette Title"
output: rmarkdown::html_vignette
vignette: >
  %\VignetteIndexEntry{Vignette Title}
  %\VignetteEngine{knitr::rmarkdown}
  \usepackage[utf8]{inputenc}
---
```

- Markdown para formatear texto.
- Knitr para interpretar texto, código y resultados.

3.2.4. Pruebas del flujo de trabajo

Las pruebas resultan fundamentales en el desarrollo de paquetes, asegura que el código haga lo que realmente se desea. Existen pruebas informales como aquellas realizadas con la función load_all(). Sin embargo, estas pruebas interactivas pueden convertirse en scripts reproducibles, los cuales resultan superiores debido a que:

- Se indica explícitamente cómo debería comportarse el código, provocando que los errores solucionados no vuelvan a ocurrir.
- El código que es fácil de probar generalmente está mejor diseñado, reduce la duplicación en el código. Como resultado, las funciones serán más fáciles de probar, comprender y trabajar.
- Si toda la funcionalidad del paquete tiene una prueba asociada, se pueden hacer grandes cambios sin preocuparse por generar errores.

Para ello se utiliza la función usethis::usetestthat() (Wickham,2011). Esta crea un directorio tests/testthat, agrega testthat al campo Suggests en el archivo DESCRIPTION y además, crea un archivo tests/testthat.R.

Las pruebas se organizan jerárquicamente, las expectativas se agrupan en pruebas que se organizan en archivos :

- Una expectativa describe el resultado esperado de un cálculo.
- Una prueba agrupa múltiples expectativas para probar la salida de una función simple, un rango de posibilidades para un solo parámetro de una función más complicada o una funcionalidad estrechamente relacionada de varias funciones.
- Un archivo agrupa múltiples pruebas relacionadas.

Existen tres formas de llevar a cabo las pruebas:

- Ejecutar todas las pruebas en un archivo o directorio test_file() o test_dir().
- Ejecutar pruebas automáticamente cada vez que algo cambie con la función auto-test(). Estas son útiles cuando las pruebas se ejecutan con frecuencia. Si se modifica un archivo de prueba, probará ese archivo; si se modifica un archivo de código, volverá a cargar ese archivo y volverá a ejecutar todas las pruebas.
- Hacer que R CMD check ejecute sus pruebas.

3.2.5. Compilación e instalación

Mediante la función load_all() fue utilizado para simular el proceso de construcción, instalación y conexión del paquete, con el fin de ir probando las funciones creadas. Sin embargo, R CMD check ejecutado en el shell o la función check(), es utilizado para verificar que un paquete R esta en pleno funcionamiento. La misma verificará que no haya errores de sintaxis o no se generen warnings. Está compuesto por más de 50 chequeos individuales entre los cuales se encuentran: la estructura del paquete, el archivo descripción, namespace, el código de R, los datos, la documentación, entre otros. Se aconseja realizar verificaciones completas de que todo funciona a medida que se van incorporando funciones ya que si se incorporan muchas y luego se verifican será dificil identificar y resolver los problemas. Una vez que las verificaciones completas no encuentran errores, advertencias o notas, se ejecuta la función install(), con el objetivo de instalar el paquete en la biblioteca.

3.2.6. Publicación

Un repositorio es el lugar dónde se encuentran alojados los paquetes y desde el cuál se pueden descargarlos. Entre los repositorios más populares de paquetes R se encuentran:

- **CRAN**: es el principal repositorio de paquetes de R, está coordinado por la fundación R. Previa a la publicación en este repositorio el paquete debe pasar por diferentes pruebas para asegurar que cumple con las políticas de CRAN.
- **Bioconductor**: se trata de un repositorio específico para bioinformática. Del mismo modo que CRAN, tiene sus propias políticas de publicaciones y procesos de revisión.
- **GitHub**: a pesar que no es específico para R, github es con toda seguridad el repositorio más popular para la publicación de proyectos *open source* (del inglés, código abierto). Su popularidad procede del espacio ilimitado que proporciona para el alojamiento de proyectos *open source*, la integración con git (un software de control de versiones) y, la facilidad de compartir y colaborar con otras personas. Una de sus desventajas es que no proporciona procesos de control.
- **R-Forge** y **RForge**: son entornos de desarrollo de paquetes y repositorios. Eso significa que incluyen control de fuente, seguimiento de errores y otras características. Puede obtener versiones de desarrollo de paquetes de estos.

El paquete *geneticae* se encuentra en GitHub, para instalar el mismo se deben seguir las siguientes instrucciones:

```
library(devtools)
install_github("jangelini/geneticae")
```

3.3. Shiny APP

Una aplicación web es una aplicación o herramienta informática accesible desde cualquier navegador, bien sea a través de internet (lo habitual) o bien a través de una red local. Estas aplicaciones son muy populares hoy en día para los usuarios no expertos, debido a la facilidad de su uso, ya que no requiere de una instalación en el ordenador del usuario, simplemente se accede a través de un navegador. Por lo que es posible utilizar una aplicación web desde desde cualquier dispositivo con conexión a internet, ya sea un ordenador, un smartphone o una tablet, es decir que es independiente del sistema operativo del usuario. Otra gran ventaja es el bajo consumo de recursos, ya que la mayor parte del tiempo estos se consumen en el servidor donde se encuentra alojada la aplicación, que generalmente tiene mucha más potencia de cómputo que cualquier ordenador personal.

Crear aplicaciones web puede resultar difícil para la mayoría de los usuarios de R debido a que se necesita un conocimiento profundo de las tecnologías web como HTML,

CSS y JavaScript; y además hacer aplicaciones interactivas complejas requiere un análisis cuidadoso de los flujos de interacción para asegurarse de que cuando una entrada cambie, solo se actualicen las salidas relacionadas. Shiny es un paquete R que te permite crear aplicaciones web interactivas, permitiendo exhibir un trabajo de R a través de un navegador web para que cualquiera pueda usarlo. Este paquete hace que sea mucho más fácil para el programador R crear aplicaciones web al proporcionar un conjunto de funciones de interfaz de usuario (UI para abreviar) que generan el HTML, CSS y JavaScript que necesita para tareas comunes. Esto significa que no se necesita conocer los detalles de HTML / CSS / JS.

Los dos componentes clave de una Shiny APP son:

- ui (*user interfaz*): la interfaz de usuario controla el diseño de la aplicación, recibe los inputs y muestra los outputs en el navegador.
- server, funciones de R que contienen las instrucciones que se necesitan para construir los resultados de los análisis incluidos en la aplicación.
- shinyApp, función que crea objetos de aplicación Shiny a partir de ui / servidor.

El esquema interno de la aplicación puede observarse en la Figura 3.1.

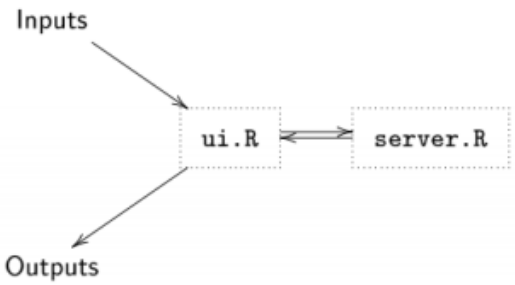


Figura 3.1: Esquema interno de la aplicación.

3.3.1. Flujo de trabajo

El esta sección se motrará como mejorar dos flujos de trabajo de Shiny importantes: el ciclo de desarrollo básico de crear aplicaciones, realizar cambios y experimentar con los resultados; y la depuración, proceso de identificar y corregir errores de programación.

1. Flujo de trabajo de desarrollo

El objetivo de optimizar el flujo de trabajo de desarrollo es reducir el tiempo entre hacer un cambio y ver el resultado. Cuanto más rápido se pueda iterar, más rápido se podrá experimentar y más rápido se podrá obtener la Shiny APP. Aquí hay dos flujos de

trabajo principales para optimizar: crear la aplicación por primera vez y acelerar el ciclo iterativo de ajustar el código y probar los resultados.

Creación de la Shiny APP

Para poder crear una shiny APP se debe tener instado R, RStudio, y el paquete shiny. Una forma de crear la aplicación es crear un nuevo directorio con un sólo archivo llamado app.R. Este archivo especificará la interfaz de usuario así como también las funciones de R que se incluirán.

```
library(shiny)
ui<- ...
server<- ...
shinyApp(ui = ui, server = server)
```

Por lo tanto, en el archivo app.R se realizan las siguientes tareas:

- Carga el paquete shiny: library(shiny)
- Define la interfaz de usuario, la página web HTML con la que los usuarios interactúan.
- Especifica el comportamiento de la aplicación definiendo la función server.
- Se ejecuta función shinyApp(ui, server) para construir e iniciar una aplicación Shiny desde la interfaz de usuario y el servidor.

La sesión de R estará monitoreando la aplicación y ejecutando las reacciones de la aplicación mientras la aplicación Shiny esté activa, por lo que no se podrá ejecutar ningún comando.

En todo tipo de programación, es una mala práctica tener código duplicado; puede ser un desperdicio computacional y, lo que es más importante, aumenta la dificultad de mantener o depurar el código. En la secuencia de comandos R tradicional, se utilizan dos técnicas para lidiar con el código duplicado: capturar el valor usando una variable o capturar el cálculo con una función. Desafortunadamente, ninguno de estos enfoques funciona en una Shiny APP y se necesita un nuevo mecanismo: expresiones reactivas. Una expresión reactiva tiene una diferencia importante con una variable: sólo se ejecuta la primera vez que se llama y luego almacena en caché el resultado de la misma hasta que necesite actualizarse.

La programación reactiva es un estilo de programación que enfatiza valores que cambian con el tiempo, y cálculos y acciones que dependen de esos valores. Esto es importante para las aplicaciones Shiny porque son interactivas: los usuarios cambian los inputs, lo que hace que la lógica se ejecute en el servidor que finalmente resultan en actualización de los outputs/resultados.

Ver los cambios

Al crear o modificar la aplicacion, se la ejecuta para poder ver los cambios realizados, por lo que el dominio de flujo de trabajo de desarrollo es especialmente importante. La primera forma de reducir la velocidad de iteración es evitar hacer clic en el botón "Ejecutar aplicación" y, en su lugar, aprender el método abreviado de teclado Cmd/Ctrl+ Shift+ Enter. Esto brinda el siguiente flujo de trabajo de desarrollo:

1. Escribir un código.
2. Iniciar la aplicación con Cmd/Ctrl+ Shift+ Enter.
3. Experimentar interactivamente con la aplicación.
4. Cerrar la aplicación.
5. Ir a 1.

Otra forma de reducir aún más la velocidad de iteración es activar la recarga automática (options(shiny.autoreload = TRUE)) y luego ejecutar la aplicación en un trabajo en segundo plano. Con este flujo de trabajo cuando se guarde un archivo, su aplicación se reiniciará: no es necesario cerrarla y reiniciarla. Esto conduce a un flujo de trabajo aún más rápido:

1. Escribir un código y presione Cmd/Ctrl+S para guardar en el archivo.
2. Experimentar interactivamente.
3. Ir a 1.

La principal desventaja de esta técnica es que debido a que la aplicación se ejecuta en un proceso separado, es considerablemente más difícil de depurar.

De manera predeterminada, cuando ejecuta la aplicación, aparecerá en una ventana emergente. Sin embargo ,existen otras dos opciones que puede elegir del menú desplegable *Run App*

1. La ejecución en el panel del visor es útil para aplicaciones más pequeñas porque puede verla al mismo tiempo que ejecuta el código de la aplicación.
2. Ejecutar en un navegador externo es útil para aplicaciones más grandes, o si desea verificar que su aplicación se ve exactamente de la manera que espera en el contexto que la mayoría de los usuarios la verán.

2. Depuración

Entre los problemas que pueden surgir al crear una Shiny app se encuentran los siguientes:

- Error inesperado. Este es el caso más fácil, porque obtendrá un rastreo que le permitirá averiguar exactamente de dónde proviene el error. Una vez que haya identificado el problema, deberá probar sistemáticamente su suposición hasta que encuentre una diferencia entre sus expectativas y lo que realmente está sucediendo. El depurador interactivo es una herramienta poderosa para este proceso.
- No obtiene ningún error, pero un valor es incorrecto. Aquí, generalmente es mejor transformar esto en el primer problema utilizando stop()para arrojar un error cuando se produce el valor incorrecto.
- Todos los valores son correctos, pero no se actualizan cuando espera. Este es el problema más desafiante porque es exclusivo de Shiny, por lo que no puede aprovechar sus habilidades de depuración de R.

Una vez localizado la fuente del error, la herramienta más poderosa es el depurador interactivo. El depurador detiene la ejecución y le brinda una consola R interactiva donde puede ejecutar cualquier código para descubrir qué salió mal. Hay dos formas de iniciar el depurador:

- Agregar una llamada a la función browser() en código fuente. Esta es la forma estándar de R de iniciar el depurador interactivo, y funcionará sin embargo, se está ejecutando brillante.
- Agregar un punto de interrupción RStudio haciendo clic a la izquierda del número de línea. Puede eliminar el punto de interrupción haciendo clic en el círculo rojo. La ventaja de los puntos de interrupción es que no son código, por lo que nunca tendrá que preocuparse por registrarlos accidentalmente en su sistema de control de versiones.

3.3.2. Compartiendo una Shiny Web App

Una vez creada la aplicación, resulta conveniente ponerlas a disposición de los usuarios. En este caso la Shiny Web App encuentra disponible en el servidor de CONICET `www.cefobi.com`. Además el proyecto se encuentra en GitHub `https://github.com/jangelini/shinyAPP_geneticae`.

Capítulo 4

Resultados

4.1. Paquete de R *geneticae*

El paquete *geneticae* permite analizar datos provenientes de etapas avanzadas de los programas de mejoramiento, donde se evalúan pocos genotipos en diversos ambientes.

Una vez instalado el paquete, se debe cargar en la sesion de R mediante el comando: library(geneticae). Es posible obtener información detallada sobre las funciones del paquete geneticae mediante help(package = "geneticae"). La ayuda para una función, por ejemplo, imputation(), en una sesión R se puede obtener usando *?imputation* o help(imputation). Además, a partir de la función browseVignettes("geneticae") se obtiene la viñeta del paquete, es decir una descripción el problema que está diseñado para resolver asi como ejemplos de aplicación del mismo.

4.1.1. Conjuntos de datos en geneticae

El paquete geneticae proporciona dos conjuntos de datos que permiten ilustrar la metodología incluida para analizar los datos obtenidos de EMA.

- yan.winterwheat dataset: rendimiento de 18 variedades de trigo de invierno cultivadas en nueve ambientes en Ontario en 1993. No hay réplicas disponibles en los datos. Este conjunto de datos se obtuvo del paquete agridat.

```
data(yan.winterwheat)
dat_yan <- yan.winterwheat
head(dat_yan)
```

```
##   gen  env yield
## 1 Ann BH93 4.460
## 2 Ari BH93 4.417
## 3 Aug BH93 4.669
## 4 Cas BH93 4.732
## 5 Del BH93 4.390
## 6 Dia BH93 5.178
```

- plrv dataset: rendimiento, peso de planta y parcela de 28 clones de la población del virus del enrollamiento de la papa (PLRV) evaluada en seis ambientes. Las réplicas están disponibles en los datos. Este conjunto de datos se obtuvo del paquete agricolae.

```
data(plrv)
dat_rep <- plrv
head(dat_rep)
```

```
##   Genotype Locality Rep WeightPlant WeightPlot    Yield
## 1   102.18     Ayac   1   0.5100000       5.10 18.88889
## 2   104.22     Ayac   1   0.3450000       2.76 12.77778
## 3   121.31     Ayac   1   0.5425000       4.34 20.09259
## 4   141.28     Ayac   1   0.9888889       8.90 36.62551
## 5   157.26     Ayac   1   0.6250000       5.00 23.14815
## 6    163.9     Ayac   1   0.5120000       2.56 18.96296
```

4.1.2. Funciones en geneticae

Modelo de regresión por sitio

Para ejecutar la función GGEmodel(), se debe proporcionar un conjunto de datos con genotipos, ambientes, repeticiones (si hay disponibles), el fenotipo observado y los nombres que dichas variables tienen en el archivo de entrada. Además, se debe indicar el método de centrado, escala y SVD.

Para el conjunto de datos yan.winterwheat, en el cual no se dispone de replicas, el modelo GGE se indica de la siguiente manera:

```
GGE1 <- GGEmodel(dat_yan, genotype = "gen", environment = "env", response = "yield"
    , centering = "tester")
```

Mientras que para el conjunto de datos plrv, la función a utilizar debe incluir además el nombre de la columna que contiene las réplicas en el conjunto de datos:

```
GGE1_rep <- GGEmodel(dat_rep, genotype = "Genotype", environment = "Locality",
    response = "Yield", rep = "Rep",   centering = "tester")
```

La salida de la función GGEmodel() es una lista que contiene las coordenadas para genotipos de todos los componentes, coordenadas para los ambientes de todos los componentes, vector de valores propios de cada componente, variancia total, porcentaje de variancia explicado por cada componente, entre otros.

Biplot GGE

Para ejecutar la función GGEPlot(), se requiere un objeto de la clase GGEmodel(). La salida es un biplot construido a través de los componentes principales generados por GGEmodel(). Los diferentes biplots que se pueden obtener usando la función GGEPlot() se muestran usando el conjunto de datos yan.winterwheat. En caso de contar con réplicas, se debe indicar el nombre de la columna que contiene las mismas en el archivo de entrada.

- *Biplot básico*

  FALTA COMENTARIO

  En la figura 4.1

```
GGEPlot(GGE1, type = "Biplot")
```

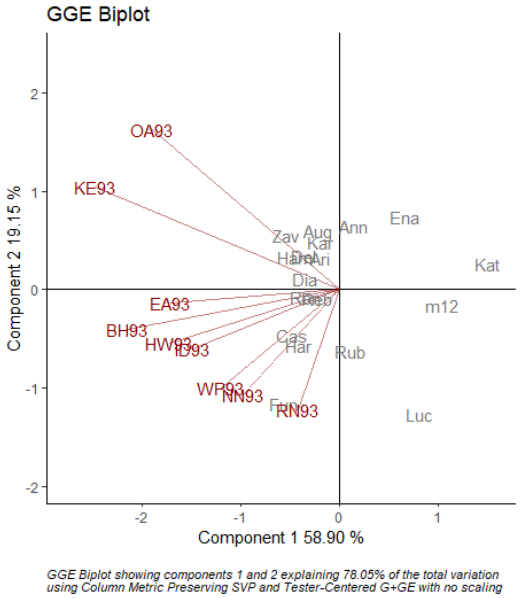


Figura 4.1: Biplot básico obtenido de la función GGEPlot()

- *Ranking de los cultivares en función de su rendimiento en el ambiente OA93*

Para identificar los mejores genotipos en un ambiente a través del biplot GGE, Yan y Hunt (2002) sugieren constituir un eje del ambiente trazando una recta que pase por el identificador del ambiente y el origen. Las proyecciones de los marcadores de los genotipos sobre ese eje, proveen un ranking de los genotipos en ese ambiente, siendo el genotipo de mayor rendimiento en el ambiente es aquel cuya proyección sobre el eje está más alejada del origen hacia el semi-eje donde se encuentra el marcador del ambiente. Aquel cuya proyección sea la segunda más alejada del origen en ese sentido, será el de segundo mejor rendimiento y así hasta llegar al de menor rendimiento en el ambiente, que es aquel cuya proyección está más alejada del origen en sentido contrario al identificador del ambiente. La perpendicular al eje del ambiente que pasa por el origen, divide a los genotipos de rendimiento superior e inferior al promedio del ambiente.
Como se observa en la Figura 4.2, el genotipo de mayor rendimiento en el ambiente OA93 es Luc seguido por Kat, m12, y así sucesivamente hasta llegar al genotipo Zav, que es el de rendimiento mas bajo en ese ambiente. Los marcadores de los genotipos Luc, m12, Kat, Rub, Fun, Har y Ema quedan del lado del marcador del ambiente OA93, de acuerdo a la división de la perpendicular que pasa por el origen, por que se interpreta que estos genotipos tienen un rendimiento mayor al promedio del ambiente OA93. Los restantes genotipos tienen un rendimiento inferior al promedio.

GGEPlot(GGE1, type = "Selected Environment", selectedE = "OA93")

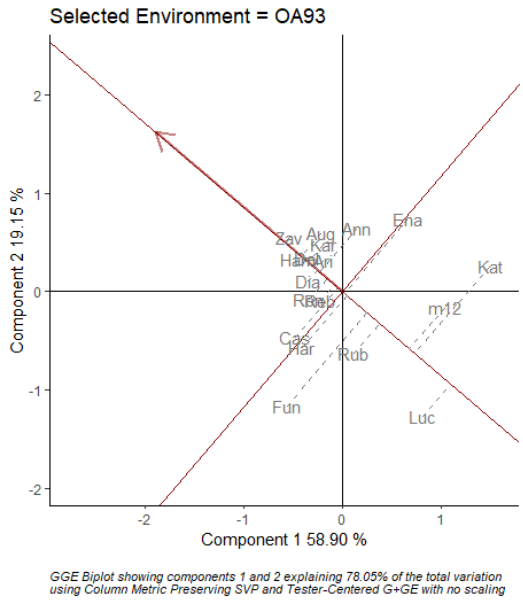


Figura 4.2: Ranking de cultivares para un ambiente determinado obtenido de la función GGEPlot()

- *Ranking de los ambientes en función del rendimiento relativo del cultivar Kat*

Para visualizar el desempeño de un genotipo en los diferentes ambientes, Yan y Hunt (2002) sugieren graficar una línea que una el marcador del genotipo con el origen y luego trazar otra línea perpendicular a la primera. Esta última perpendicular es la que separa los sitos favorables y desfavorables para el genotipo. Los sitios cuyos marcadores queden en el mismo lado donde está el genotipo son los mejores para ese genotipo y los restantes son aquellos donde el genotipo rinde por debajo de su promedio.
Como se puede apreciar en la Figura **??**, la perpendicular al marcador del genotipo Kat determina que ninguno de los ambientes evaluados son favorables para ese genotipo.

GGEPlot(GGE1, type = "Selected Genotype", selectedG = "Kat")

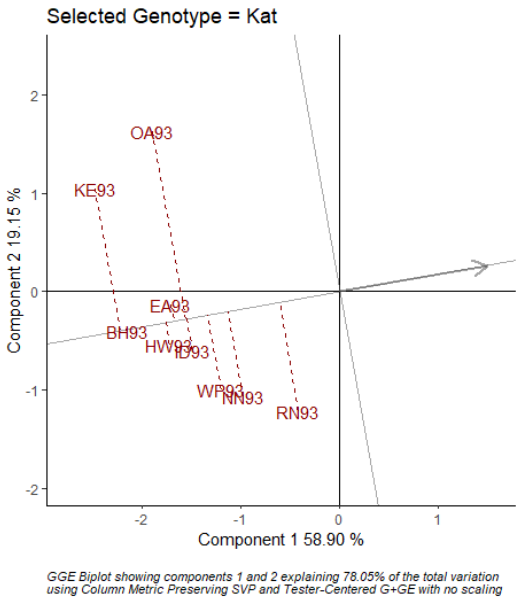


Figura 4.3: Ranking de ambientes para cultivar determinado obtenido de la función GGEPlot()

- *Relación entre ambientes*

figu 4.4

GGEPlot(GGE1, type = "Relationship Among Environments")

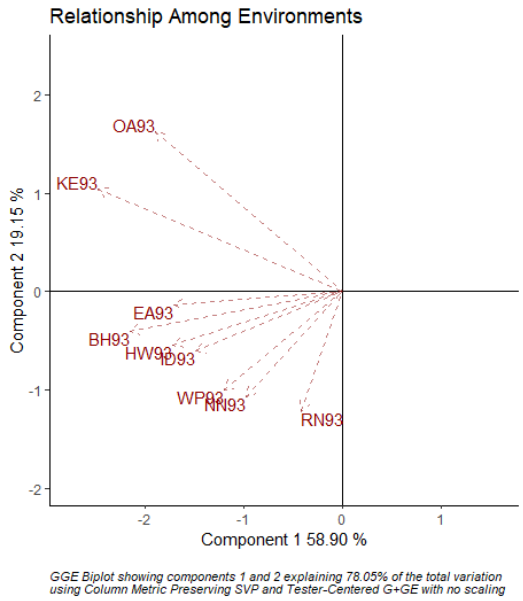


Figura 4.4: Relación entre ambientes obtenido de la función GGEPlot()

- *Comparación entre los genotipos Kat y Cas*

Para comparar dos genotipos, se propone unir mediante una línea recta los genotipos a comparar, luego trazar una línea que pase por el origen y que sea perpendicular a la línea que une a los genotipos. Esta última línea es la que separa sitios favorables a uno y a otro genotipo.
Los sitios cuyos marcadores queden en el mismo lado donde está el marcador del genotipo son los mejores para ese genotipo. Si un ambiente queda posicionado sobre la línea perpendicular, los dos genotipos tienen rendimientos similares en ese ambiente. Si dos genotipos están cercanos, sus rendimientos son similares en los ambientes evaluados. Por último, si todos los ambientes quedan a un lado de la línea perpendicular, el genotipo cuyo identificador está de ese lado rinde más que el otro en todos los ambientes.
En la Figura 4.5 se puede ver la comparación de los desempeños de los genotipos Kat y Cas. Todos los ambientes resultan favorables para Cas y ninguno para Kat.

GGEPlot(GGE1, type = "Comparison of Genotype", selectedG1 = "Kat", selectedG2 = "Cas")

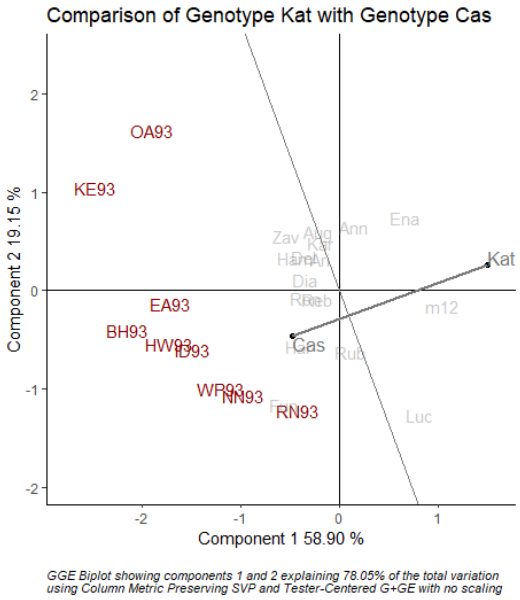


Figura 4.5: Comparación entre dos genotipos obtenido de la función GGEPlot()

- *Identificación del mejor cultivar en cada ambiente*

Para poder identificar mega-ambientes y los mejores genotipos en cada uno de ellos se propone graficar un polígono envolvente. Este polígono se forma uniendo los genotipos más extremos en el biplot con segmentos continuados. Luego se trazan líneas rectas que pasan por el origen y que son perpendiculares a cada uno de los lados del polígono (o a sus proyecciones). De esta forma, el biplot queda dividido en cuadrantes, y los sitios que quedan dentro un mismo cuadrante se consideran pertenecientes a un mismo mega-ambiente. Generalmente cada cuadrante contiene un genotipo en el vértice, que es el de mayor rendimiento en el mega-ambiente.
En la Figura 4.6 se presenta el biplot GGE con el polígono envolvente y las perpendiculares a sus lados, que ayudan a la interpretación del mismo. En primer lugar se observa que los marcadores de los ambientes OA93 y KE93 son mayores a los restantes, lo que indica que la variabilidad en esos ambientes es superior, es decir, en ellos es donde mejor se diferencian los efectos de los genotipos. Las perpendiculares a los lados del polígono envolvente determinan dos mega-ambientes: - uno formado por los ambientes OA93 y KE93, en donde el genotipo de mejor desempeño es el Zav (se encuentra en el vértice del polígono encerrado por las perpendiculares). - el segundo mega-ambiente lo componen los restantes ambientes, en este caso el genotipo ganador es el Fun.

GGEPlot(GGE1, type = "Which Won Where/What")

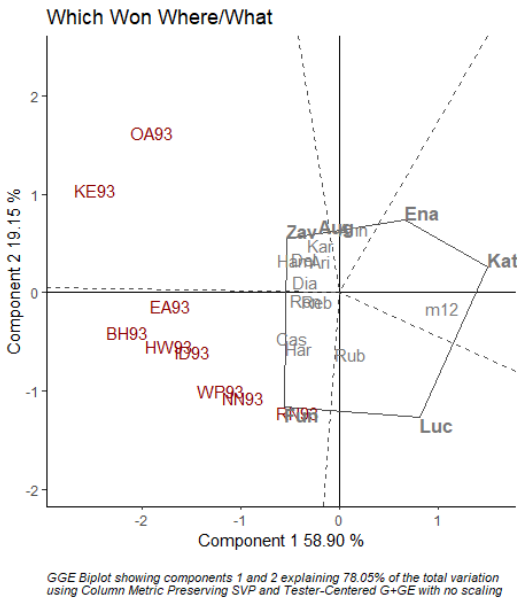


Figura 4.6: Identificación del mejor cultivar en cada ambiente a partir de la función GGEPlot()

- *Evaluación de los ambientes basados tanto en la capacidad de discriminación como en la representatividad*

figu 4.7

GGEPlot(GGE1, type = "Discrimination vs. representativeness")

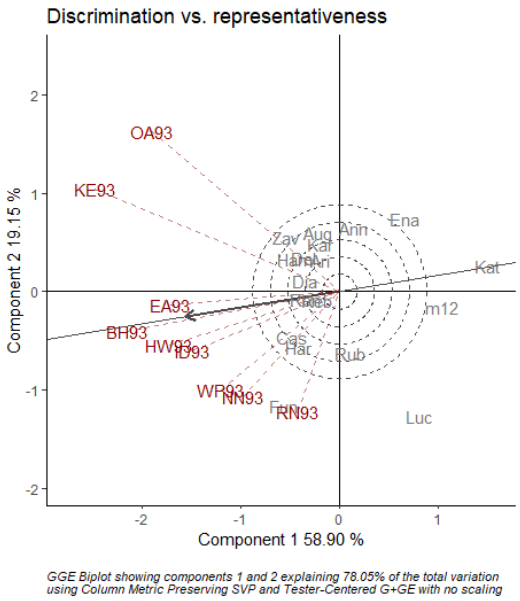


Figura 4.7: Evaluación de los ambientes basados tanto en la capacidad de discriminación y representatividad a partir de la función GGEPlot()

- *Clasificación de ambientes con respecto al ambiente ideal*

figu 4.8

GGEPlot(GGE1, type = "Ranking Environments")

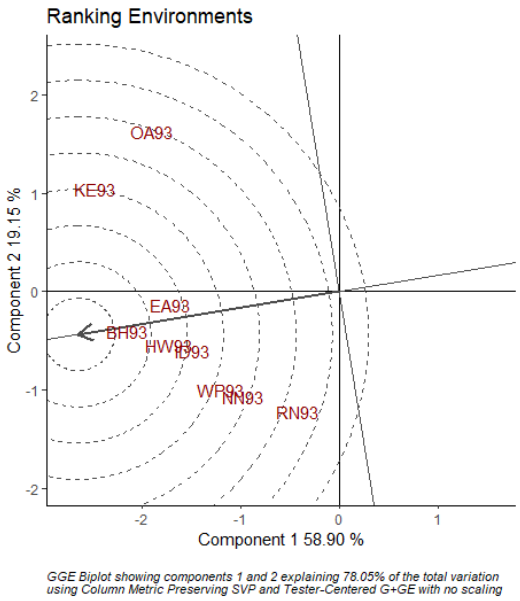


Figura 4.8: Clasificación de ambientes con respecto al ambiente ideal a partir de la función GGEPlot()

- *Clasificación de genotipos con respecto al genotipo ideal*

figu 4.9

GGEPlot(GGE1, type = "Ranking Genotypes")

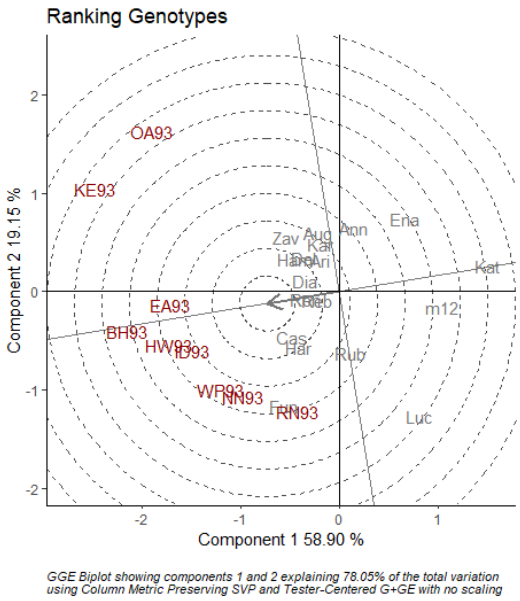


Figura 4.9: Clasificación de genotipos con respecto al genotipo ideal a partir de la función GGEPlot()

- *Evaluación de los cultivares con base en el rendimiento promedio y la estabilidad*

Como se puede observar en el biplot de la figura 4.10 el orden de los genotipos (de mayor a menor rendimiento) es: Kat, m12, Ena, Luc, Ann todos ellos con rendimientos superiores al promedio, seguidos por los de rendimiento menor al promedio y por último Fun, el de peor rendimiento medio en ese mega-ambiente. Debido a que las proyecciones sobre el eje perpendicular al eje medio de ambiente dan una idea de la estabilidad, se observa que el genotipo Luc y Fun son los más inestables. También se observa que el genotipo Kat, además de tener el mejor rendimiento medio es de los más estables en el megaambiente.**VER... ES CONTRADICTORIO... VER SI EL AEC ES SOBRE UN MEGA AMBIENTE O SI ES SOBRE TODOS**

GGEPlot(GGE1, type = "Mean vs. Stability")

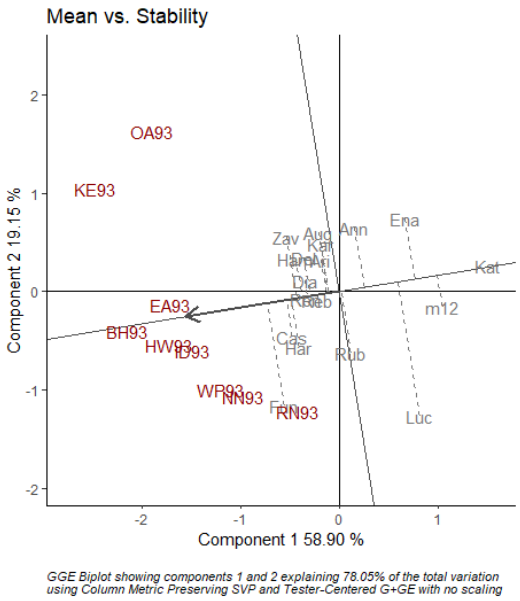


Figura 4.10: Evaluación de los cultivares con base en el rendimiento promedio y la estabilidad a partir de la función GGEPlot()

Classic AMMI model

Para ejecutar la función rAMMI(), como en la función GGEmodel(), se debe proporcionar un conjunto de datos con genotipo, entorno, repeticiones (si las hay) y la variable de respuesta. Se debe indicar el nombre de las columnas que contienen cada una de estas variables en el conjunto de datos de entradas. La salida de la función es un biplot.

A continuación se muestra el biplot GE obtenido del modelo AMMI clásico obtenido con el conjunto de datos yan.winterwheat.

```
rAMMI(dat_yan, genotype = "gen", environment = "env", response = "yield", type = "
    AMMI")
```

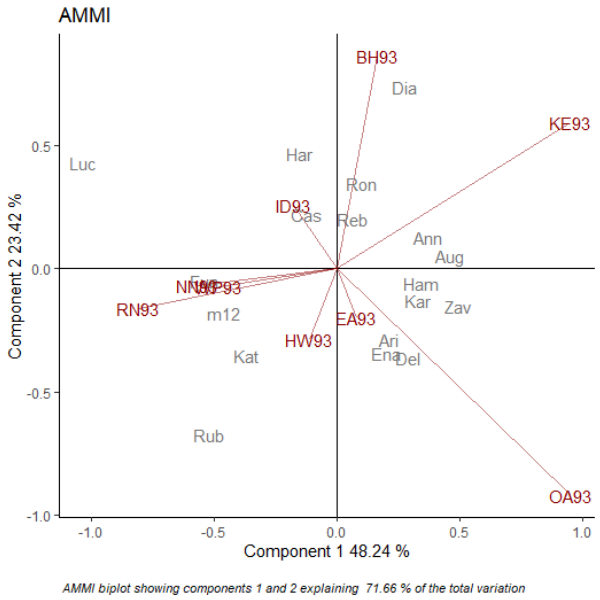


Figura 4.11: Biplot GE obtenido del modelo clasico AMMI

Robust AMMI model

Como se dijo anteriormente, el modelo AMMI clasico, en su forma estándar, no funciona bien en presencia de observaciones atípicas. Dado que los outliers son muy comun en los datos agronómicos, Rodrigues et al. (2015) proponen cinco modelos AMMI robustos, que permiten superar el problema de la contaminación de datos con observaciones atípicas. Los biplots de los cinco modelos AMMI robustos propuestos por Rodrigues et al. (2015), se pueden obtener utilizando la función rAMMI() A continuación se muestran los biplots obtenidos con dichos modelos robustos usando el conjunto de datos yan.winterwheat.

- modelo rAMMI"

rAMMI(dat_yan, genotype = "gen", environment = "env", response = "yield", type = "rAMMI")

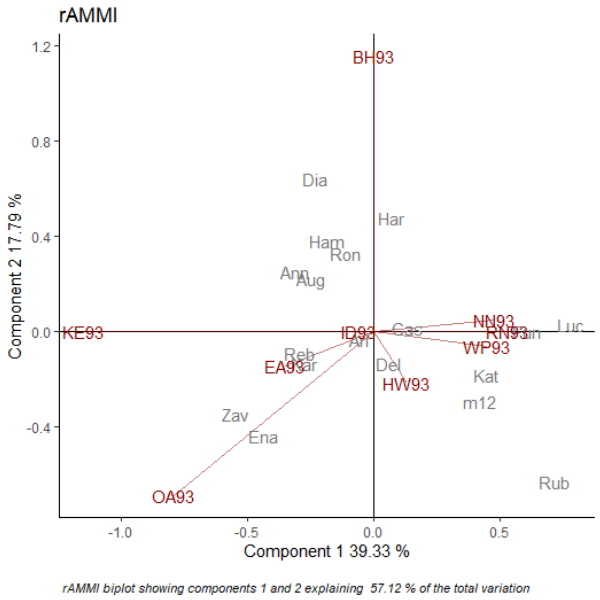


Figura 4.12: Biplot GE obtenido del modelo robusto rAMMI

- modelo "hAMMI"

rAMMI(dat_yan, genotype = "gen", environment = "env", response = "yield", type = "hAMMI")

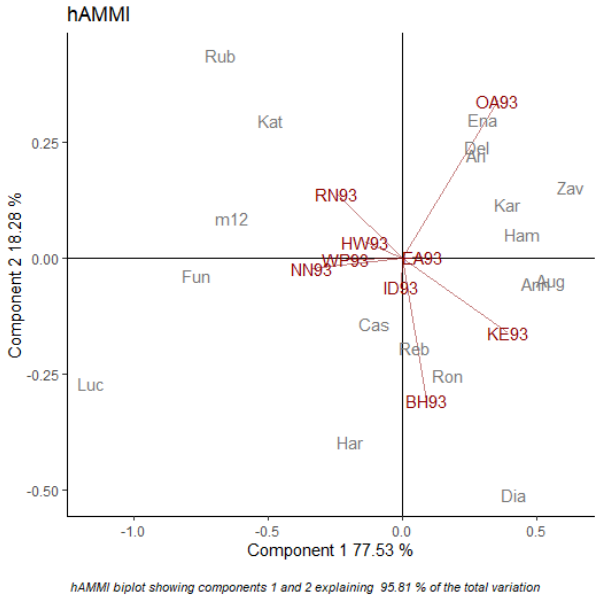


Figura 4.13: Biplot GE obtenido del modelo robusto hAMMI

- modelo "gAMMI"

```
rAMMI(dat_yan, genotype = "gen", environment = "env", response = "yield", type = "
    gAMMI")
```

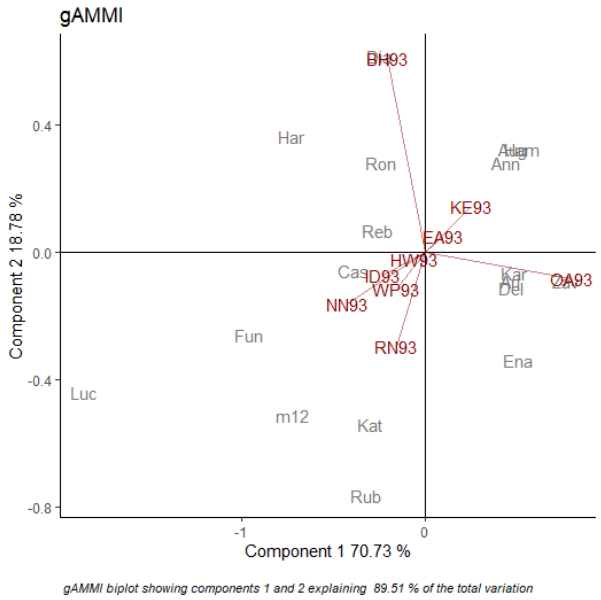


Figura 4.14: Biplot GE obtenido del modelo robusto gAMMI

- modelo "lAMMI"

```
rAMMI(dat_yan, genotype = "gen", environment = "env", response = "yield", type = "
    lAMMI")
```

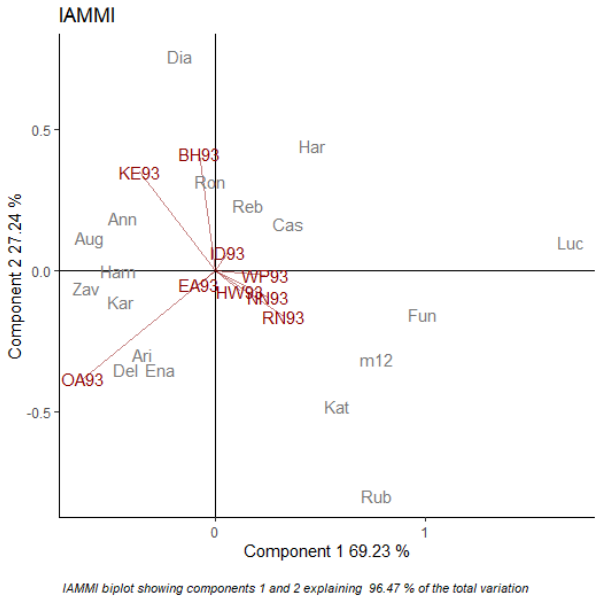


Figura 4.15: Biplot GE obtenido del modelo robusto lAMMI

- modelo "ppAMMI"

```
rAMMI(dat_yan, genotype = "gen", environment = "env", response = "yield", type = "
    ppAMMI")
```

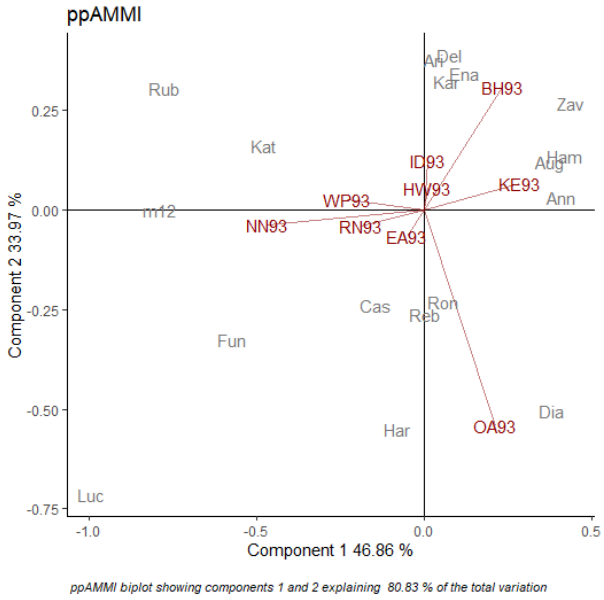


Figura 4.16: Biplot GE obtenido del modelo robusto ppAMMI

Métodos de imputación

Una limitación importante de los modelos presentados anteriormente es que requieren una que el conjunto de datos este completo. Por lo tanto, en el paquete se incluyen una serie de metodologías propuestas, algunas de las cuales no se encuentran disponible en R, para superar el problema de las observaciones perdidas.

El conjunto de datos yan.winterwheat se utilizó como ejemplo. Como el conjunto de datos no contaba con observaciones perdidas, algunas fueron eliminadas con el objetivo de mostrar las metodologías de imputación incluidas.

```
# generates missing data
dat_yan[1, 3] <- NA
dat_yan[3, 3] <- NA
dat_yan[2, 3] <- NA
```

- *GabrielEigein*

```
imputation(dat_yan, PC.nb = 2, genotype = "gen", environment = "env", response = "yield
    ", type = "EM-AMMI")
```

```
##         BH93  EA93  HW93  ID93  KE93  NN93  OA93  RN93  WP93
## Ann 4.150120 4.150 2.849 3.084 5.940 4.450 4.351 4.039 2.672
## Ari 4.035814 4.771 2.912 3.506 5.699 5.152 4.956 4.386 2.938
## Aug 4.305244 4.578 3.098 3.460 6.070 5.025 4.730 3.900 2.621
## Cas 4.732000 4.745 3.375 3.904 6.224 5.340 4.226 4.893 3.451
## Del 4.390000 4.603 3.511 3.848 5.773 5.421 5.147 4.098 2.832
## Dia 5.178000 4.475 2.990 3.774 6.583 5.045 3.985 4.271 2.776
## Ena 3.375000 4.175 2.741 3.157 5.342 4.267 4.162 4.063 2.032
## Fun 4.852000 4.664 4.425 3.952 5.536 5.832 4.168 5.060 3.574
## Ham 5.038000 4.741 3.508 3.437 5.960 4.859 4.977 4.514 2.859
## Har 5.195000 4.662 3.596 3.759 5.937 5.345 3.895 4.450 3.300
## Kar 4.293000 4.530 2.760 3.422 6.142 5.250 4.856 4.137 3.149
## Kat 3.151000 3.040 2.388 2.350 4.229 4.257 3.384 4.071 2.103
## Luc 4.104000 3.878 2.302 3.718 4.555 5.149 2.596 4.956 2.886
## Reb 4.375000 4.701 3.655 3.592 6.189 5.141 3.933 4.208 2.925
## Ron 4.940000 4.698 2.950 3.898 6.063 5.326 4.302 4.299 3.031
## Rub 3.786000 4.969 3.379 3.353 4.774 5.304 4.322 4.858 3.382
## Zav 4.238000 4.654 3.607 3.914 6.641 4.830 5.014 4.363 3.111
## m12 3.340000 3.854 2.419 2.783 4.629 5.090 3.281 3.918 2.561
```

- *EM-AMMI*

```
imputation(dat_yan, PC.nb = 1, genotype = "gen", environment = "env", response = "yield
    ", type = "EM-AMMI")
```

```
##         BH93  EA93  HW93  ID93  KE93  NN93  OA93  RN93  WP93
## Ann 4.136249 4.150 2.849 3.084 5.940 4.450 4.351 4.039 2.672
## Ari 4.474249 4.771 2.912 3.506 5.699 5.152 4.956 4.386 2.938
## Aug 4.386299 4.578 3.098 3.460 6.070 5.025 4.730 3.900 2.621
## Cas 4.732000 4.745 3.375 3.904 6.224 5.340 4.226 4.893 3.451
## Del 4.390000 4.603 3.511 3.848 5.773 5.421 5.147 4.098 2.832
## Dia 5.178000 4.475 2.990 3.774 6.583 5.045 3.985 4.271 2.776
## Ena 3.375000 4.175 2.741 3.157 5.342 4.267 4.162 4.063 2.032
## Fun 4.852000 4.664 4.425 3.952 5.536 5.832 4.168 5.060 3.574
## Ham 5.038000 4.741 3.508 3.437 5.960 4.859 4.977 4.514 2.859
## Har 5.195000 4.662 3.596 3.759 5.937 5.345 3.895 4.450 3.300
## Kar 4.293000 4.530 2.760 3.422 6.142 5.250 4.856 4.137 3.149
## Kat 3.151000 3.040 2.388 2.350 4.229 4.257 3.384 4.071 2.103
## Luc 4.104000 3.878 2.302 3.718 4.555 5.149 2.596 4.956 2.886
## Reb 4.375000 4.701 3.655 3.592 6.189 5.141 3.933 4.208 2.925
## Ron 4.940000 4.698 2.950 3.898 6.063 5.326 4.302 4.299 3.031
## Rub 3.786000 4.969 3.379 3.353 4.774 5.304 4.322 4.858 3.382
## Zav 4.238000 4.654 3.607 3.914 6.641 4.830 5.014 4.363 3.111
## m12 3.340000 3.854 2.419 2.783 4.629 5.090 3.281 3.918 2.561
```

- *EM-SVD*

```
imputation(dat_yan, genotype = "gen", environment = "env", response = "yield", type = "
    EM-SVD")
```

```
##           [,1]  [,2]  [,3]  [,4]  [,5]  [,6]  [,7]  [,8]  [,9]
##  [1,] 4.332467 4.150 2.849 3.084 5.940 4.450 4.351 4.039 2.672
##  [2,] 4.332467 4.771 2.912 3.506 5.699 5.152 4.956 4.386 2.938
##  [3,] 4.332467 4.578 3.098 3.460 6.070 5.025 4.730 3.900 2.621
##  [4,] 4.732000 4.745 3.375 3.904 6.224 5.340 4.226 4.893 3.451
##  [5,] 4.390000 4.603 3.511 3.848 5.773 5.421 5.147 4.098 2.832
##  [6,] 5.178000 4.475 2.990 3.774 6.583 5.045 3.985 4.271 2.776
##  [7,] 3.375000 4.175 2.741 3.157 5.342 4.267 4.162 4.063 2.032
##  [8,] 4.852000 4.664 4.425 3.952 5.536 5.832 4.168 5.060 3.574
##  [9,] 5.038000 4.741 3.508 3.437 5.960 4.859 4.977 4.514 2.859
## [10,] 5.195000 4.662 3.596 3.759 5.937 5.345 3.895 4.450 3.300
## [11,] 4.293000 4.530 2.760 3.422 6.142 5.250 4.856 4.137 3.149
## [12,] 3.151000 3.040 2.388 2.350 4.229 4.257 3.384 4.071 2.103
## [13,] 4.104000 3.878 2.302 3.718 4.555 5.149 2.596 4.956 2.886
## [14,] 4.375000 4.701 3.655 3.592 6.189 5.141 3.933 4.208 2.925
## [15,] 4.940000 4.698 2.950 3.898 6.063 5.326 4.302 4.299 3.031
## [16,] 3.786000 4.969 3.379 3.353 4.774 5.304 4.322 4.858 3.382
## [17,] 4.238000 4.654 3.607 3.914 6.641 4.830 5.014 4.363 3.111
## [18,] 3.340000 3.854 2.419 2.783 4.629 5.090 3.281 3.918 2.561
```

- *WGabriel*

```
imputation(dat_yan, genotype = "gen", environment = "env", response = "yield", type = "
    WGabriel")
```

```
##         BH93  EA93  HW93  ID93  KE93  NN93  OA93  RN93  WP93
## Ann 4.004664 4.150 2.849 3.084 5.940 4.450 4.351 4.039 2.672
## Ari 4.455727 4.771 2.912 3.506 5.699 5.152 4.956 4.386 2.938
## Aug 4.328095 4.578 3.098 3.460 6.070 5.025 4.730 3.900 2.621
## Cas 4.732000 4.745 3.375 3.904 6.224 5.340 4.226 4.893 3.451
## Del 4.390000 4.603 3.511 3.848 5.773 5.421 5.147 4.098 2.832
## Dia 5.178000 4.475 2.990 3.774 6.583 5.045 3.985 4.271 2.776
## Ena 3.375000 4.175 2.741 3.157 5.342 4.267 4.162 4.063 2.032
## Fun 4.852000 4.664 4.425 3.952 5.536 5.832 4.168 5.060 3.574
## Ham 5.038000 4.741 3.508 3.437 5.960 4.859 4.977 4.514 2.859
## Har 5.195000 4.662 3.596 3.759 5.937 5.345 3.895 4.450 3.300
## Kar 4.293000 4.530 2.760 3.422 6.142 5.250 4.856 4.137 3.149
## Kat 3.151000 3.040 2.388 2.350 4.229 4.257 3.384 4.071 2.103
## Luc 4.104000 3.878 2.302 3.718 4.555 5.149 2.596 4.956 2.886
## Reb 4.375000 4.701 3.655 3.592 6.189 5.141 3.933 4.208 2.925
## Ron 4.940000 4.698 2.950 3.898 6.063 5.326 4.302 4.299 3.031
## Rub 3.786000 4.969 3.379 3.353 4.774 5.304 4.322 4.858 3.382
## Zav 4.238000 4.654 3.607 3.914 6.641 4.830 5.014 4.363 3.111
## m12 3.340000 3.854 2.419 2.783 4.629 5.090 3.281 3.918 2.561
```

- *EM-PCA*

```
imputation(dat_yan, genotype = "gen", environment = "env", response = "yield", type = "
    EM-PCA")
```

```
##         BH93  EA93  HW93  ID93  KE93  NN93  OA93  RN93  WP93
## Ann 3.980317 4.150 2.849 3.084 5.940 4.450 4.351 4.039 2.672
## Ari 4.463093 4.771 2.912 3.506 5.699 5.152 4.956 4.386 2.938
## Aug 4.327731 4.578 3.098 3.460 6.070 5.025 4.730 3.900 2.621
## Cas 4.732000 4.745 3.375 3.904 6.224 5.340 4.226 4.893 3.451
## Del 4.390000 4.603 3.511 3.848 5.773 5.421 5.147 4.098 2.832
## Dia 5.178000 4.475 2.990 3.774 6.583 5.045 3.985 4.271 2.776
## Ena 3.375000 4.175 2.741 3.157 5.342 4.267 4.162 4.063 2.032
## Fun 4.852000 4.664 4.425 3.952 5.536 5.832 4.168 5.060 3.574
## Ham 5.038000 4.741 3.508 3.437 5.960 4.859 4.977 4.514 2.859
## Har 5.195000 4.662 3.596 3.759 5.937 5.345 3.895 4.450 3.300
## Kar 4.293000 4.530 2.760 3.422 6.142 5.250 4.856 4.137 3.149
## Kat 3.151000 3.040 2.388 2.350 4.229 4.257 3.384 4.071 2.103
## Luc 4.104000 3.878 2.302 3.718 4.555 5.149 2.596 4.956 2.886
## Reb 4.375000 4.701 3.655 3.592 6.189 5.141 3.933 4.208 2.925
## Ron 4.940000 4.698 2.950 3.898 6.063 5.326 4.302 4.299 3.031
## Rub 3.786000 4.969 3.379 3.353 4.774 5.304 4.322 4.858 3.382
## Zav 4.238000 4.654 3.607 3.914 6.641 4.830 5.014 4.363 3.111
## m12 3.340000 3.854 2.419 2.783 4.629 5.090 3.281 3.918 2.561
```

4.2. Geneticae Shiny Web App

La aplicación Geneticae permite a los usuarios realizar muchos de los análisis incluidos en el paquete geneticae. La misma se organiza en las siguientes pestañas:

- Los datos
- Análisis descriptivo
- ANOVA
- Biplot GGE
- Biplot GE
- Ayuda

En muchos casos, algunos atributos estilísticos de salida pueden personalizarse para que el usuario obtenga la salida a su gusto. A su vez, los gráficos obtenidos pueden ser descargados.

4.2.1. Los datos

Al iniciar la aplicación Geneticae, se muestra una pantalla en la cual se carga el conjunto de datos a analizar. La aplicación admite datos en formato .csv, delimitados por coma o punto y coma; y permite el siguiente formato de datos:

- Cada fila contiene una observación, en la cual deben estar presentes los siguientes datos: nombre del cultivar, ambiente, repetición si está disponible y valor fenotipico medido. Pueden estar presentes otras variables que no serán utilizadas por la aplicación.
- La primera fila de encabezado contiene los nombres de cada variable. Los encabezados pueden dar cualquier nombre que elija, y deben indicarse al cargar el archivo de datos.
- El número de repeticiones puede diferir con los genotipos y los ambientes.

Se utilizan dos conjuntos de datos, incluidos en el paquete geneticae, para ilustrar la aplicación. Estos conjuntos de datos, uno de los cuales tiene repeticiones (plrv dataset) y el otro no (yan.winterwheat dataset), los cuales se pueden ver y descargar en la pestaña *The data* → *Example datasets* (Figura 4.17,4.18).

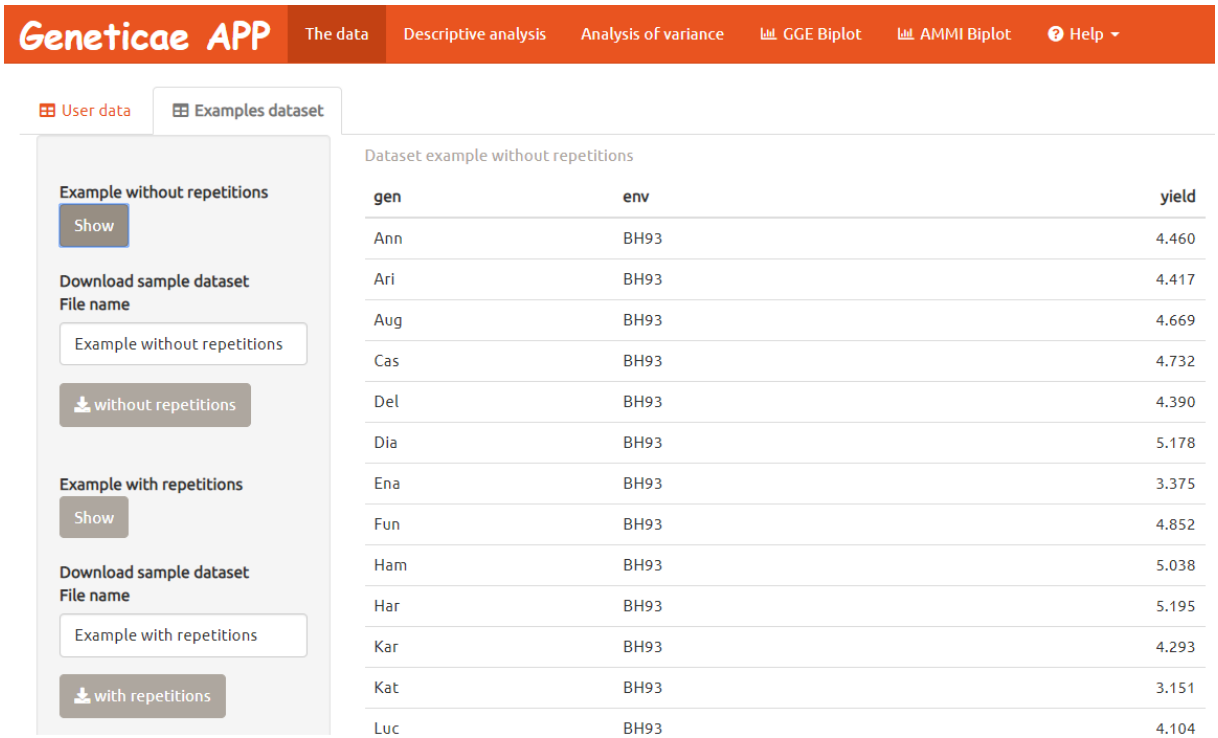


Figura 4.17: yan.winterwheat dataset disponible en Shiny Web App

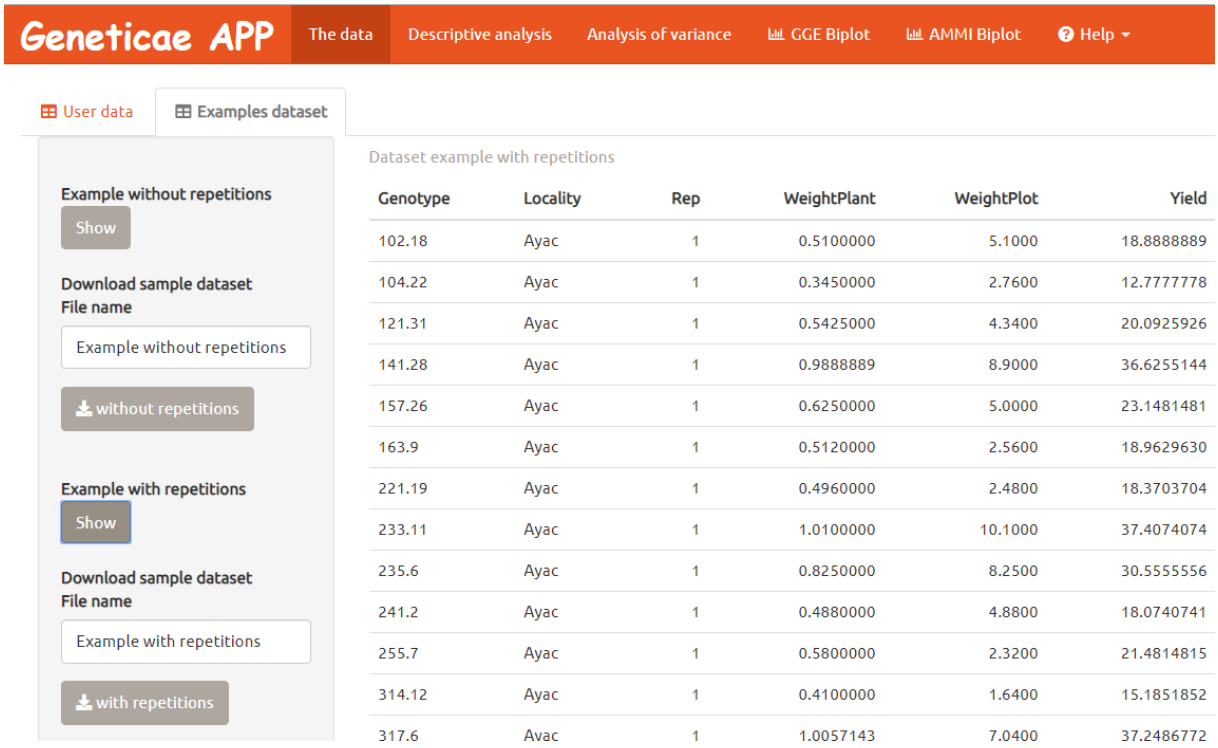


Figura 4.18: plrv dataset disponible en Shiny Web App

4.2.2. Análisis descriptivo

El menú *Descriptive analysis* le permite describir un conjunto de datos utilizando diagrama de caja (o *boxplot*), gráfico y matriz de correlación y gráfico de interacción.

4.2.2.1. *Boxplot*

El *boxplot* proporciona una medida central, la mediana y una idea de la dispersión a través del rango y el rango intercuartil. La posición de la mediana dentro de la caja y la similitud en la longitud de los bigotes nos dan una idea de la simetría de la distribución.

Un boxplot intetactivo que compara el caracter cuantitativo de interés a través de genotipos, así como a través de los ambientes se pueden obtener (Figura 4.19,4.20). A partir de los mismos se pueden obtener medidas resumen en forma interactiva usando el *Toggle Spike Lines* como se muestra en la figura 4.19. Estos gráficos se pueden descagar en formato interactivo (.HTML) a partir del boton *Download* (Figura 4.19 y 4.20), así como también en formato .png como se muestra en la Figura 4.20.

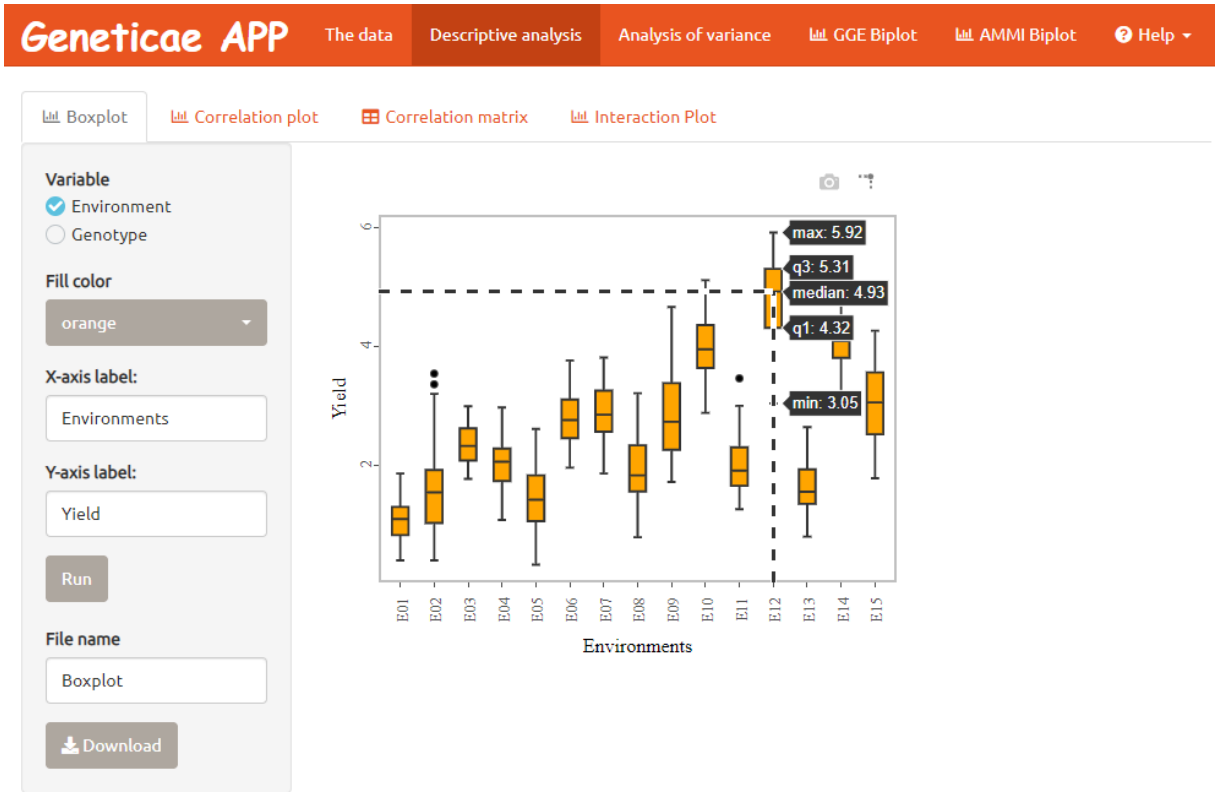


Figura 4.19: Boxplot de ambientes a través de los genotipos para el conjunto de datos Plrv

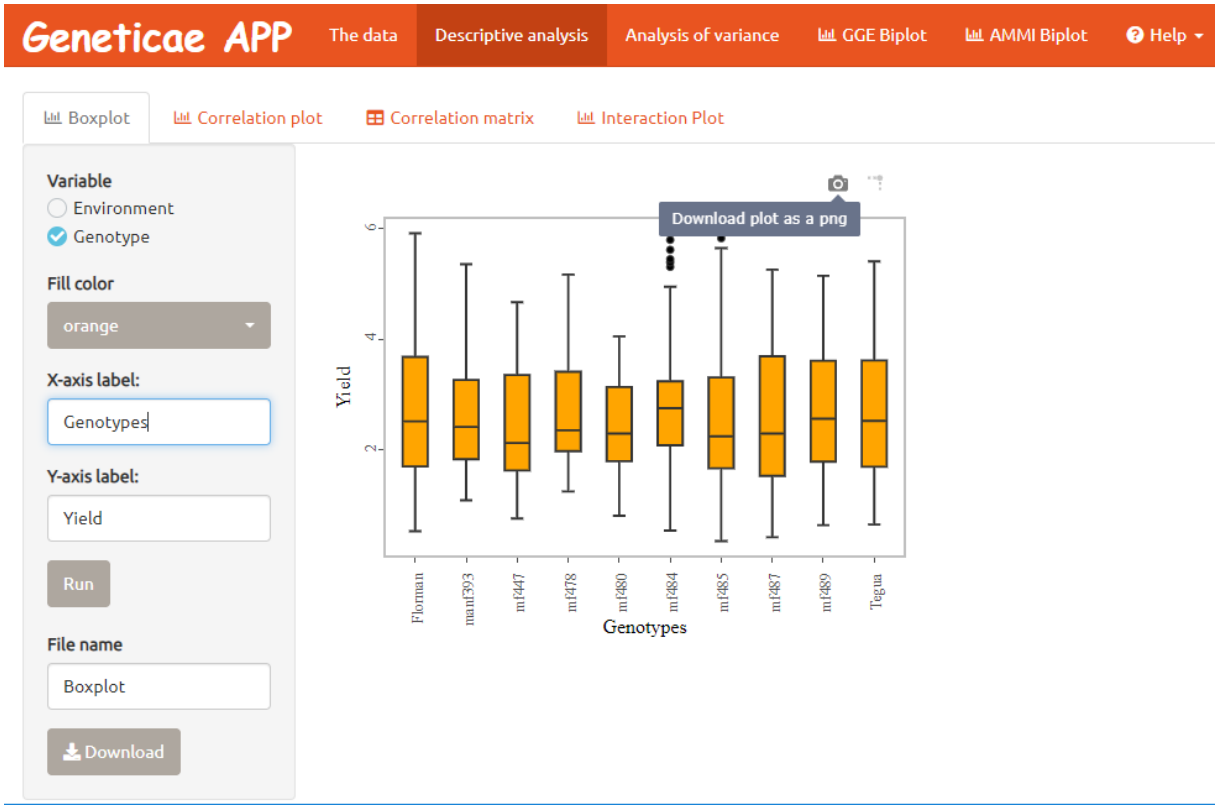


Figura 4.20: Boxplot de genotipos a través de los ambientes para el conjunto de datos Plrv

4.2.2.2. Gráfico de correlación

El correlograma o gráfico de correlación muestra la correlación tanto entre los genotipos como entre los ambientes (Figura 4.21 y 4.22). Se pueden mostrar las correlaciones de Pearson y Spearman. Las correlaciones positivas se muestran en azul y las negativas en rojo. La intensidad del color y el tamaño del círculo son proporcionales a los coeficientes de correlación.

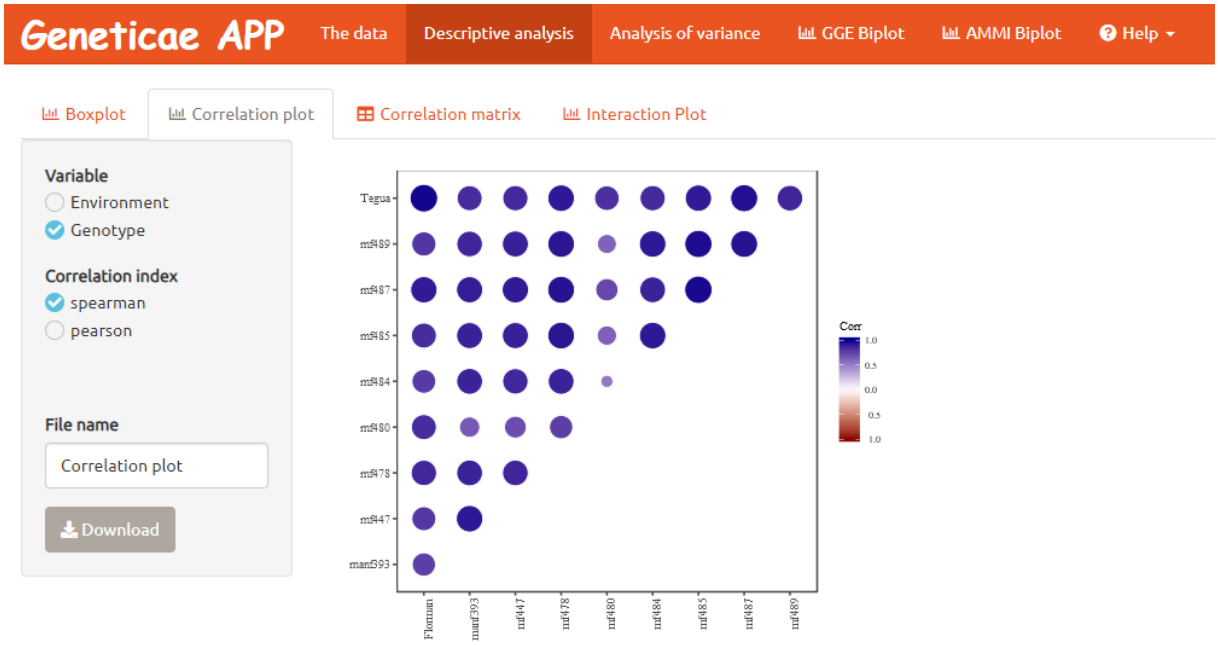


Figura 4.21: Boxplot de genotipos a través de los ambientes para el conjunto de datos Plrv

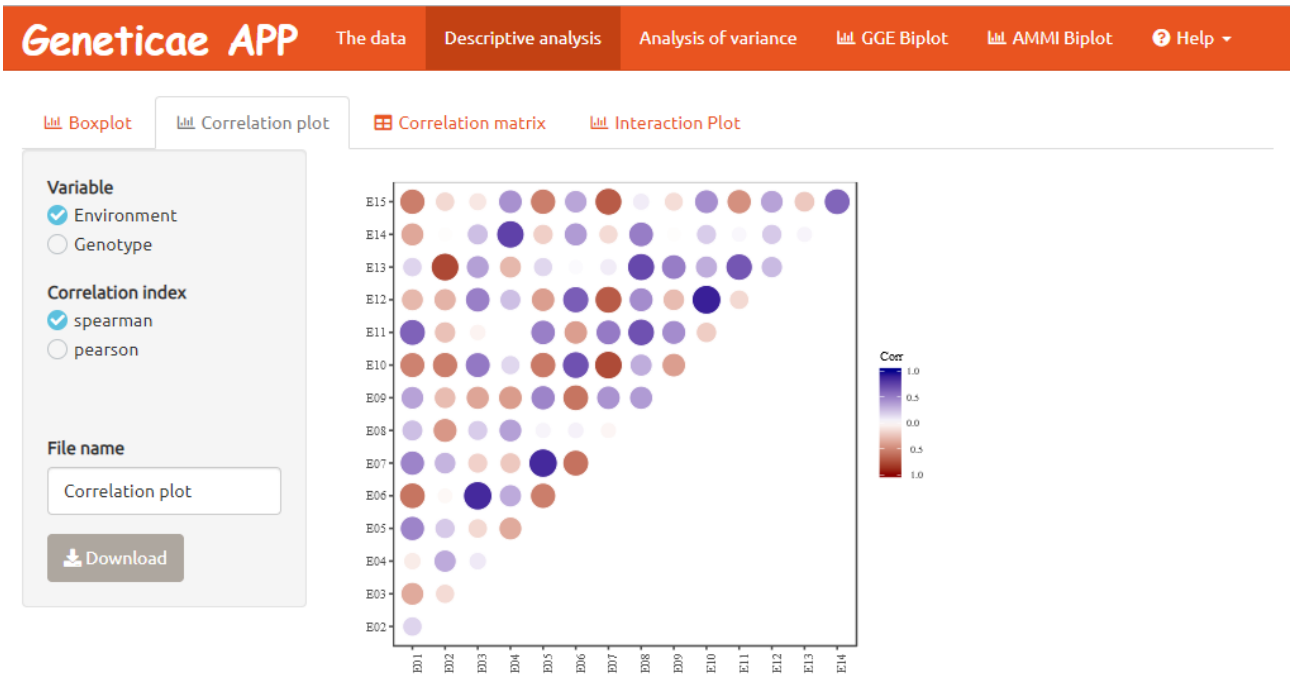
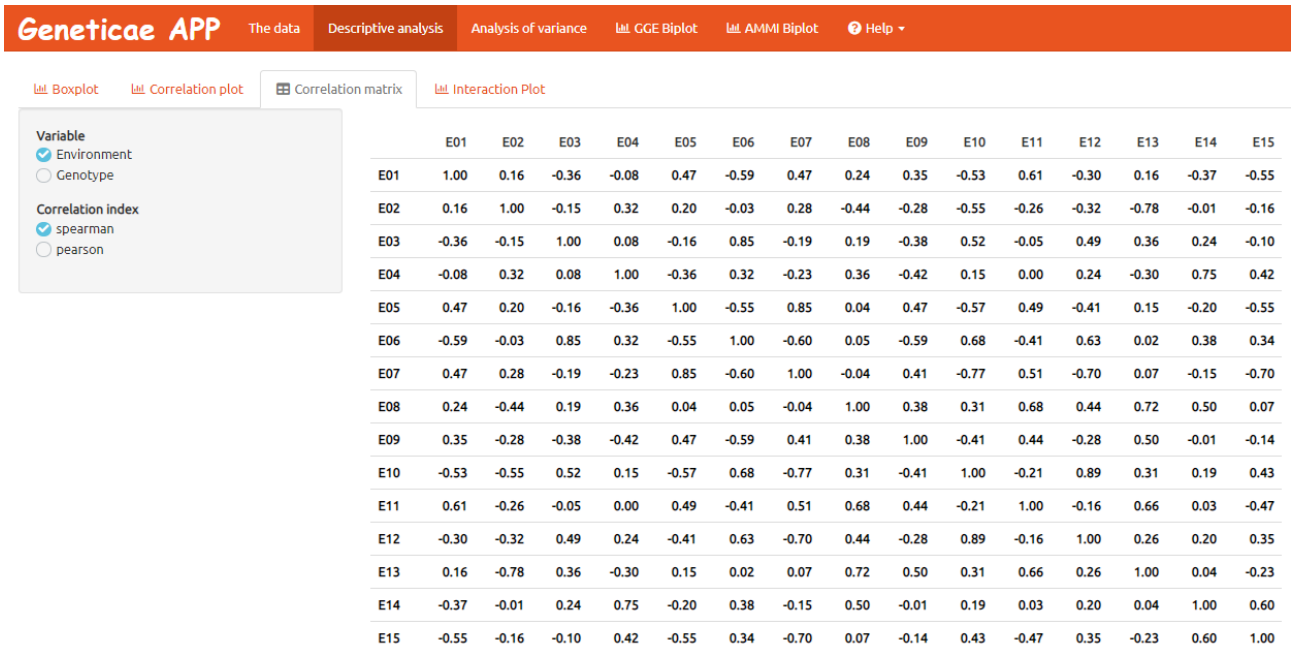


Figura 4.22: Boxplot de ambientes a través de los genotipos para el conjunto de datos Plrv

4.2.2.3. Matriz de correlación

Una matriz de correlación se utiliza como una forma de resumir datos. Muestra los coeficientes de correlación de pares de variables. Las correlaciones de Spearman o Pearson se pueden calcular tanto para ambientes como para genotipos (Figura 4.23).



	E01	E02	E03	E04	E05	E06	E07	E08	E09	E10	E11	E12	E13	E14	E15
E01	1.00	0.16	-0.36	-0.08	0.47	-0.59	0.47	0.24	0.35	-0.53	0.61	-0.30	0.16	-0.37	-0.55
E02	0.16	1.00	-0.15	0.32	0.20	-0.03	0.28	-0.44	-0.28	-0.55	-0.26	-0.32	-0.78	-0.01	-0.16
E03	-0.36	-0.15	1.00	0.08	-0.16	0.85	-0.19	0.19	-0.38	0.52	-0.05	0.49	0.36	0.24	-0.10
E04	-0.08	0.32	0.08	1.00	-0.36	0.32	-0.23	0.36	-0.42	0.15	0.00	0.24	-0.30	0.75	0.42
E05	0.47	0.20	-0.16	-0.36	1.00	-0.55	0.85	0.04	0.47	-0.57	0.49	-0.41	0.15	-0.20	-0.55
E06	-0.59	-0.03	0.85	0.32	-0.55	1.00	-0.60	0.05	-0.59	0.68	-0.41	0.63	0.02	0.38	0.34
E07	0.47	0.28	-0.19	-0.23	0.85	-0.60	1.00	-0.04	0.41	-0.77	0.51	-0.70	0.07	-0.15	-0.70
E08	0.24	-0.44	0.19	0.36	0.04	0.05	-0.04	1.00	0.38	0.31	0.68	0.44	0.72	0.50	0.07
E09	0.35	-0.28	-0.38	-0.42	0.47	-0.59	0.41	0.38	1.00	-0.41	0.44	-0.28	0.50	-0.01	-0.14
E10	-0.53	-0.55	0.52	0.15	-0.57	0.68	-0.77	0.31	-0.41	1.00	-0.21	0.89	0.31	0.19	0.43
E11	0.61	-0.26	-0.05	0.00	0.49	-0.41	0.51	0.68	0.44	-0.21	1.00	-0.16	0.66	0.03	-0.47
E12	-0.30	-0.32	0.49	0.24	-0.41	0.63	-0.70	0.44	-0.28	0.89	-0.16	1.00	0.26	0.20	0.35
E13	0.16	-0.78	0.36	-0.30	0.15	0.02	0.07	0.72	0.50	0.31	0.66	0.26	1.00	0.04	-0.23
E14	-0.37	-0.01	0.24	0.75	-0.20	0.38	-0.15	0.50	-0.01	0.19	0.03	0.20	0.04	1.00	0.60
E15	-0.55	-0.16	-0.10	0.42	-0.55	0.34	-0.70	0.07	-0.14	0.43	-0.47	0.35	-0.23	0.60	1.00

Figura 4.23: Boxplot de genotipos a través de los ambientes para el conjunto de datos Plrv

4.2.2.4. Gráfico de interacción

Un diagrama de interacción es una representación visual de la interacción entre los efectos de dos factores, o entre un factor y una variable numérica.

Se puede obtener el gráfico interactivo que muestra el cambio en el efecto genotípico a través de los entornos y también el que muestra el cambio en el efecto ambiental a través de los genotipos (Figura 4.28,**??**). Es posible descargarlo en formato interactivo (.HTML) a partir del boton *Download* (Figura 4.28), así como también en formato .png como se muestra en la Figura **??**.

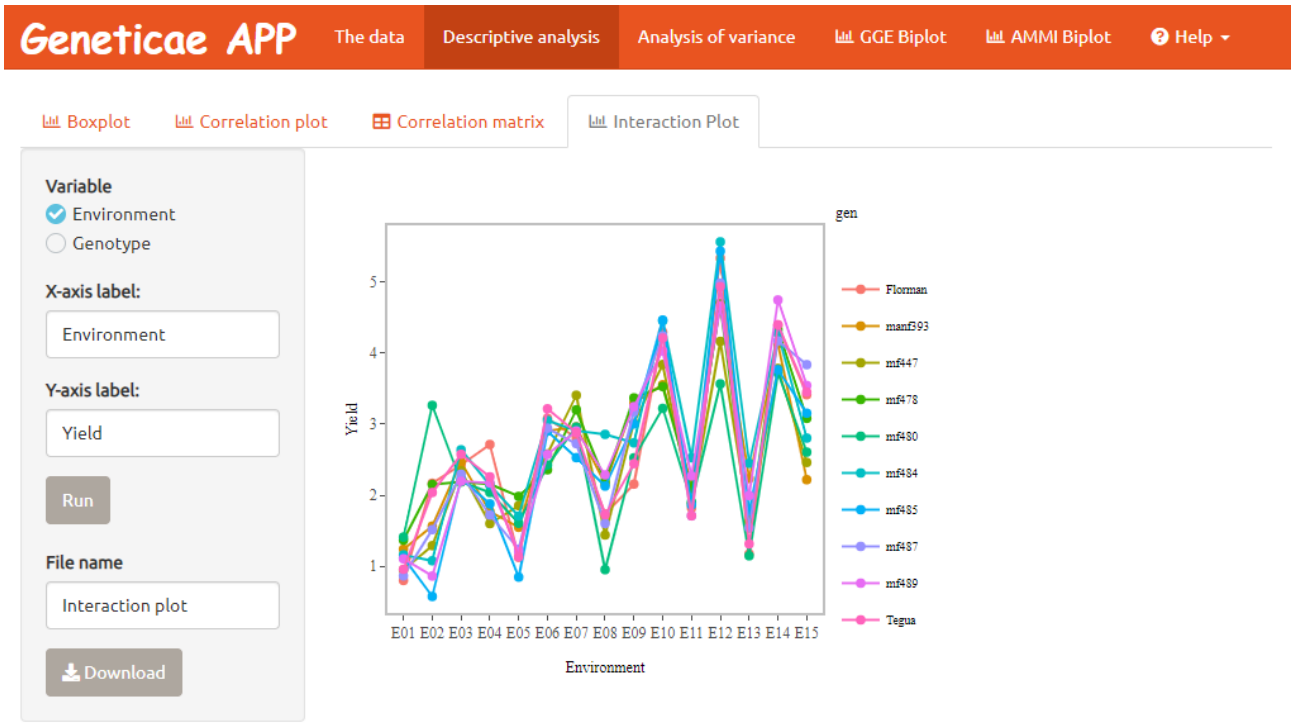
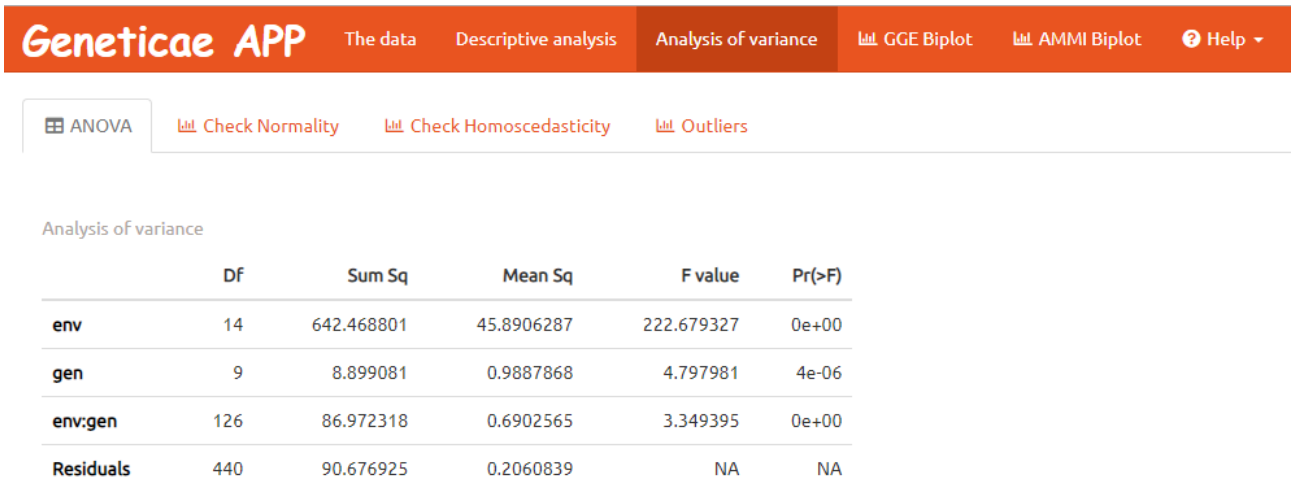


Figura 4.24: Boxplot de genotipos a través de los ambientes para el conjunto de datos Plrv

4.2.3. Análisis de la variancia

Cuando se pretende llevar a cabo el análisis de la variancia si el conjunto de datos tiene repeticiones entonces saldrá un mensaje en el cual se aclara que la interacción puede ser testada debido a la presencia de repeticiones "The interaction effect can be tested since there are repetitions in the data set", si no hay repeticiones disponibles entonces el mensaje será que la interacción no puede testarse.



	Df	Sum Sq	Mean Sq	F value	Pr(>F)
env	14	642.468801	45.8906287	222.679327	0e+00
gen	9	8.899081	0.9887868	4.797981	4e-06
env:gen	126	86.972318	0.6902565	3.349395	0e+00
Residuals	440	90.676925	0.2060839	NA	NA

Figura 4.25: Boxplot de genotipos a través de los ambientes para el conjunto de datos Plrv

El ANOVA depende del cumplimiento de los supuestos de que los errores tengan distribución normal con media cero y variancia constante. Por ello, tres pestañas: *Check normality*, *Check homocedasticity* y *Outliers* se encuentran disponibles para la verificación de los supuestos mencionados.

Para verificar el supuesto de normalidad, se puede realizar un histograma, un gráfico de probabilidad normal y la prueba de shapiro-wilks sobre los residuos del ANOVA.

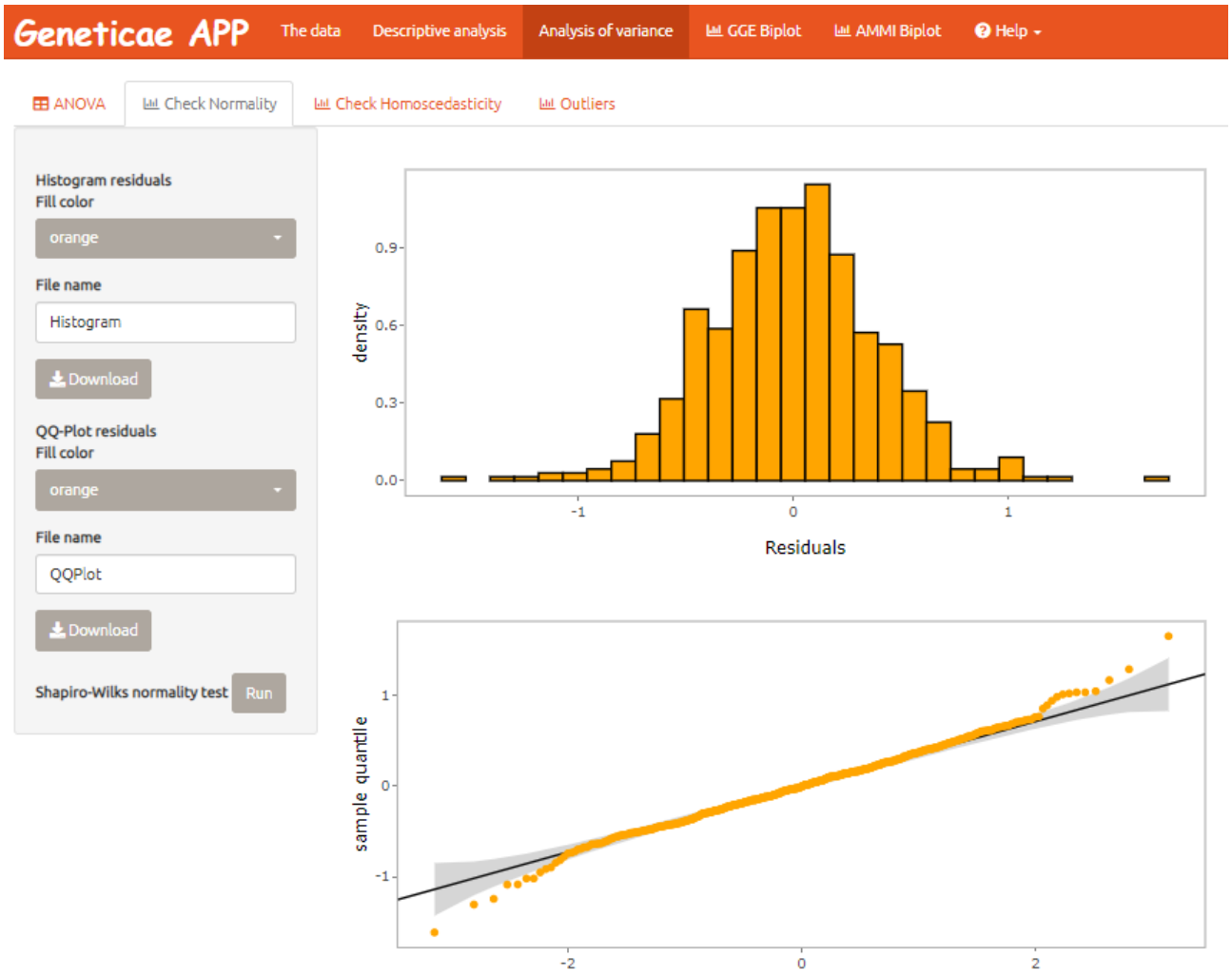


Figura 4.26: Boxplot de genotipos a través de los ambientes para el conjunto de datos Plrv

El grafico de residuos vs. valores predichos y las pruebas de levene permiten verificar el supuesto de variancia constante u homocedasticidad.

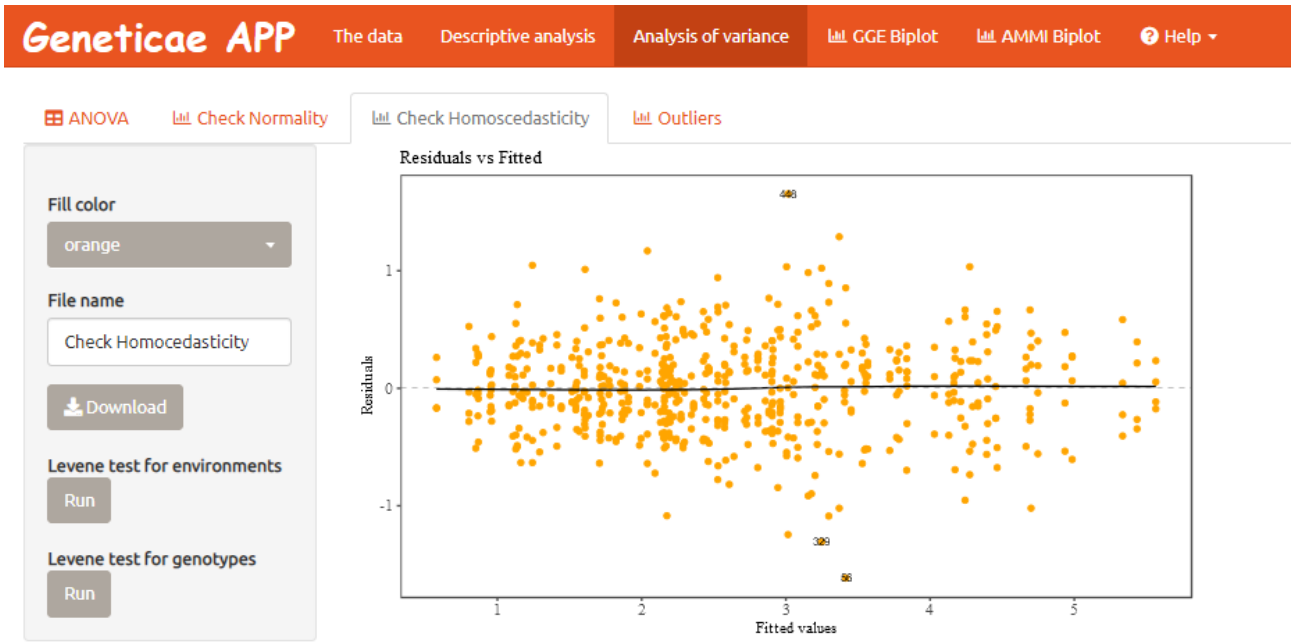


Figura 4.27: Boxplot de genotipos a través de los ambientes para el conjunto de datos Plrv

Por último, la presencia de observaciones atipicas u outliers provoca que el ANOVA no de buenos resultados, un grafico para detectar outliers es posible realizarlo.

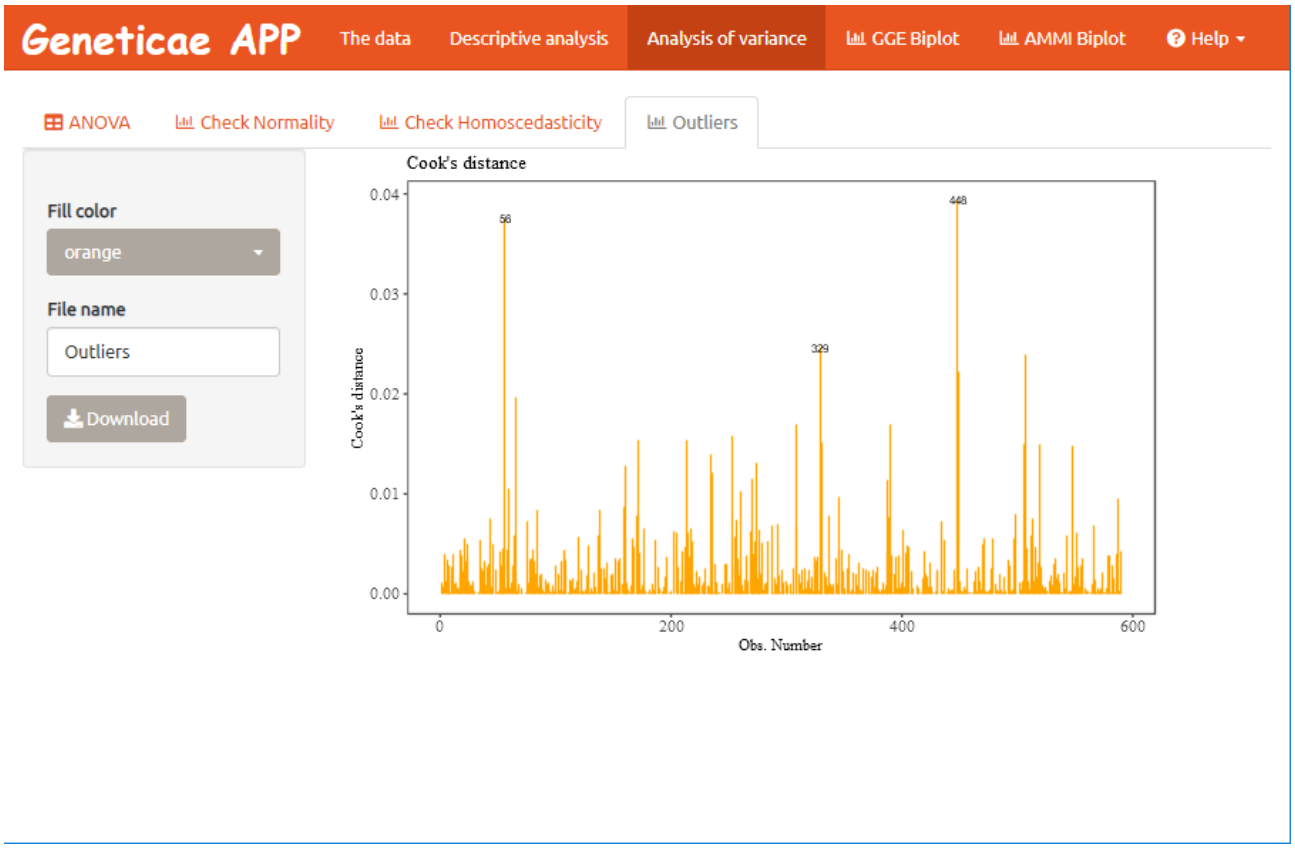


Figura 4.28: Boxplot de genotipos a través de los ambientes para el conjunto de datos Plrv

4.2.4. Biplot GGE

El biplot GGE aborda visualmente muchos problemas relacionados con la evaluación de los genotipo y ambientes de prueba. En el caso de repeticiones disponibles en el conjunto de datos, se obtiene el valor fenotípico promedio para cada combinación de genotipo y ambiente. Los valores faltantes no están permitidos. Se incluyen en la aplicación web los biplots mas utilizados.

4.2.5. Biplot GE

Capítulo 5

Conclusiones

Bibliografía

R.W. Allard. *Principios de la mejora genética de las plantas*. Ediciones Omega, 1967.

H. G. Gauch y R. W. Zobel. Identifying mega-environments and targeting genotypes. *Crop Science*, 37:311—-326, 1997.

W. Yan, L. A. Hunt, Q. Sheng, y Z. Szlavnics. Cultivar evaluation and mega-environment investigation based on the GGE biplot. *Crop Science*, 40:597—-605, 2000.