



Universidad
de Alcalá

UNIVERSIDAD DE ALCALÁ
Escuela Politécnica Superior

GRADO EN INGENIERÍA INFORMÁTICA

Trabajo de Fin de Grado

Estado del arte de visualización circular de datos con R

Autor: Pablo García Lacalle

Tutor: Juan José Cuadrado Gallego

TRIBUNAL

Presidente:

Vocal 1:

Vocal 2:

Calificación:

Índice

1. Introducción	5
1.1. Visualización de Datos	6
1.1.1. Círculos	7
2. Análisis de los Diagramas	8
2.1. Diagrama de Anillos	8
2.2. Diagrama de Barras Radiales	9
2.3. Diagrama en Espiral	9
2.4. Diagrama de Columnas Radiales	10
2.5. Diagrama de Sectores	10
2.6. Diagrama Rose of Nightingale	11
2.7. Diagrama de Cuadrícula Circular Ordenada	11
2.8. Diagrama de Cuadrícula Circular Desordenada	12
2.9. Diagrama Rayos de Sol	12
2.10. Diagrama Radar	13
2.11. Diagrama de Columnas Con Formato Iris	13
2.12. Diagrama de Líneas Circulares Multiseries	14
2.13. Diagrama de Burbujas	14
2.14. Diagrama de Mapa de Árbol Circular	15
2.15. Diagrama de Mapa de Árbol Voronoi	15
2.16. Diagrama de Planos Circulares	16
2.17. Diagrama de Mapa Circular	16
2.18. Diagrama de Mapa de Esfera	16
2.19. Dendograma Circular	17
2.20. Diagrama de Cuerdas	17
2.21. Diagrama de Redes Circulares	18
3. Visualización en R	19
3.1. Diagrama de Anillos	19
3.1.1. Paquete ggplot2	19
3.1.2. Paquete circlize	22
3.1.3. Paquete CMplot	23
3.1.4. Comparaciones	25
3.2. Diagrama de Barras Radiales	26
3.2.1. Paquete ggplot2	26
3.2.2. Paquete Plotrix	31
3.2.3. Paquete circlize	32
3.2.4. Comparaciones	34
3.3. Diagrama Espiral	35
3.3.1. Paquete ggplot2	35
3.4. Diagrama de Columnas Radiales	40
3.4.1. Paquete ggplot2	40
3.4.2. Paquete plotrix	43
3.4.3. Paquete cplots	45
3.4.4. Paquete circlize	46
3.4.5. Comparaciones	47
3.5. Diagrama de Sectores	48
3.5.1. R Básico	48
3.5.2. Paquete ggplot2	49
3.5.3. Paquete plotrix	51
3.5.4. Paquete rCharts	52
3.5.5. Paquete plotly	53
3.5.6. Paquete ggiraphExtra	54
3.5.7. Comparaciones	54

3.6.	Diagrama de Rose of Nightingale	55
3.6.1.	Paquete ggplot2	55
3.6.2.	Paquete ggiraphExtra	60
3.6.3.	Comparaciones	61
3.7.	Diagrama de Cuadrícula Circular Ordenada	62
3.7.1.	Paquete circlize	62
3.8.	Diagrama Rayos de Sol	65
3.8.1.	Paquete ggraph	65
3.8.2.	Paquete plotly	68
3.8.3.	Paquete webr	70
3.8.4.	Paquete ggiraphExtra	71
3.8.5.	Paquete sunburstR	71
3.8.6.	Comparaciones	73
3.9.	Diagrama Radar	74
3.9.1.	Paquete fmsb	74
3.9.2.	Paquete plotly	75
3.9.3.	Paquete plotrix	76
3.9.4.	Paquete ggiraphExtra	78
3.9.5.	Comparaciones	79
3.10.	Diagrama de Columnas Con Formato Iris	80
3.10.1.	Paquete ggplot2	80
3.10.2.	Paquete circlize	84
3.10.3.	Comparaciones	86
3.11.	Diagrama de Líneas Circulares Multiseries	87
3.11.1.	Paquete ggplot2	87
3.12.	Diagrama de Burbujas Circular	90
3.12.1.	Paquete ggraph-ggforce	90
3.12.2.	Paquete packcircles-ggplot2	92
3.12.3.	Comparaciones	93
3.13.	Diagrama de Mapa de Árbol Circular	94
3.13.1.	Paquete ggraph	94
3.13.2.	Paquete ciclepackeR	96
3.13.3.	Comparaciones	98
3.14.	Diagrama Mapa de Árbol Voronoi	99
3.14.1.	Paquete voronoiTreemap	99
3.15.	Diagrama de Mapa Cirular	102
3.15.1.	Paquete ggplot2	102
3.16.	Dendograma Circular	104
3.16.1.	R Básico y ape	104
3.16.2.	Paquete ggraph	105
3.16.3.	Paquete ggplot2	107
3.16.4.	Paquete circlize	108
3.16.5.	Comparaciones	111
3.17.	Diagrama de Cuerdas	112
3.17.1.	Paquete Circlize	112
3.17.2.	Paquete chorddiag	114
3.17.3.	Comparaciones	117
3.18.	Diagrama de Redes Circulares	118
3.18.1.	Explosión Centralizada	118
3.18.2.	Implosión Elíptica	119
3.18.3.	Anillo Centralizado, Convergencia Radial y Círculos de Escala	121
3.18.4.	Comparaciones	121

4. Diagramas no Realizables

122

5. Comparativa	123
5.1. Versatilidad	124
5.2. Desarrollo	124
5.3. Usabilidad	126
6. Conclusiones	127
Referencias	128

Resumen

Este trabajo realiza una estudio de que tipos de diagramas con formato circular existen, y como realizarlos con el lenguaje de programación destinado a estadística R[1]. Para ello se necesita explorar los tipos de diagramas circulares gracias a libros, como "The Book of Circles, Visualizing Spheres of Knowledge" de Manuel Lima[2], "Information Graphics, A Comprehensive Illustrated Reference" de Robert L. Harris[3]. Y saber qué paquetes de R se pueden utilizar para representar dichos gráficos. Para una correcta representación de los diagramas en código, es muy recomendable la lectura de "Data Visualization, A Practical Introduction" de Kieran Healy[4].

1. Introducción

El mundo está repleto de datos obtenidos de la observación, experimentos, etc. Pero, estos por si solos, no aportan el conocimiento necesario para que sus receptores saquen información válida.

Dicha información ayuda a la capacidad de entender el entorno que nos rodea (física, biología, geología, etc) y para ser capaces de realizar una mejor toma de decisiones (pandemias, economía, etc). Pero, para transformar estos datos en información válida, hay que condensarlos en una representación gráfica que ayude a, la interpretación y construcción de significado de los datos, y a la comunicación de la información obtenida.

Por ejemplo, si se tienen muchos datos con los sueldos de los habitantes de un país y se transforman siguiendo la estadística, se puede obtener el sueldo medio de éste junto con la moda del sueldo, etc; para que el país tome la decisiones pertinentes con dicha información.

Para cada situación existe un tipo de visualización de datos: para números las tablas; para comparar información de diferentes entidades en un instante de tiempo las barras; para visualizar cambios a lo largo del tiempo u otra variable, las líneas; etc.

Todas estas variedades de visualizaciones se pueden, como es objetivo de este trabajo, realizar con un formato de círculos. Estos son una de las figuras geométricas que más fascinan a la humanidad, debido a que se vislumbraban con mucha facilidad en la naturaleza, como astros celestiales, lagos, ondas generadas en el agua, frutas, etc.

Los círculos empezaron a usarse en arquitectura, por ejemplo, en bóvedas; en ciudades, como las aldeas de nativos americanos; en fortificaciones, como la muralla de Ávila (España); metáforas universales, como de la perfección (God the Architect, William de Bailes), la unidad (Fra Mauro map, Fra Mauro), el movimiento (Yama holding the wheel of life), la infinitud (Miracles of Each Moment); en mucha de la simbología como , la representación del tiempo, el ciclo de la vida, en los logos de empresas como, Vodafone, Bayer, Motorola, Mercedes; y en simbología antigua como los petroglifos prehistóricos de Argyll [2, pág 15-54].

Los humanos están constantemente buscando patrones, orden, pautas, instrucciones, debido a ello consiguen comprender mejor lo que les rodea, como los seres vivos (biología), seres inertes, lo terrenal, lo espacial y en general, los patrones que gobiernan el universo. Y gracias al formato circular de los diagramas, que se representarán a lo largo de este trabajo, se pueden identificar dichas pautas, normalmente cíclicas, en los diferentes datos que recreen los distintos diagramas.

Para este trabajo se ha optado por R[1] , lenguaje de programación con enfoque estadístico basado en software libre, donde cada persona pueda contribuir a su crecimiento mediante un conjunto de funciones agrupadas en paquetes. Este lenguaje es uno de los más usados para Big Data, minería de datos, matemáticas financieras, aprendizaje automático, etc. En este caso, se trabaja en el área de Big Data con la visualización de datos, por lo que estos diferentes diagramas ayudan a la observación y comprensión de grandes cantidades de datos.

1.1. Visualización de Datos

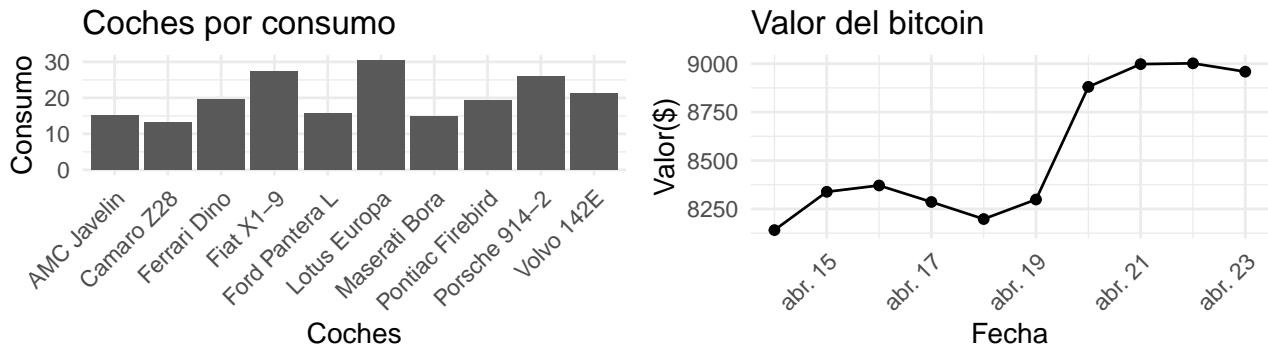
Visualizar: Representar mediante imágenes ópticas fenómenos de otro carácter[5], por consiguiente, la visualización de datos es la representación gráfica de datos. Lo que implica producir imágenes que comunican relaciones entre los datos representados a los observadores. Refiriéndose a las técnicas usadas para comunicar datos o información mediante su codificación como objetos visuales (puntos, barras, líneas, etc.) contenidos en un gráfico.

"La meta principal de la visualización de datos es comunicar información clara y efectivamente gracias a medios gráficos. Esto no significa que la visualización de datos necesite parecer aburrida para ser práctica o extremadamente sofisticada para parecer bonita. Para transmitir ideas de forma eficiente, tanto la estética como la funcionalidad necesitan ir de la mano, proporcionando información sobre un conjunto de datos escasos y complejos mediante la comunicación de sus aspectos clave de una manera más intuitiva. Todavía los diseñadores suelen fallar a alcanzar un balance entre la forma y la función, creando bellas visualizaciones de datos que erran el propósito general de comunicar información"[6, Vitaly Friedman]. "Hechas bien, las visualizaciones explicar complejas ideas de manera simple"[7, Thomas Powell, CEO de ZingChart].

Una visualización ideal no solo debe comunicar de forma clara, sino que se necesita estimular compromiso y atención al espectador[8]. Por ello se considera a la visualización de datos arte aparte de ciencia[9][10].

Durante su historia, este campo y sus profesionales han abordado distintos tipos de retos tanto éticos como analíticos[11], debido a esto se ha vuelto un área activa en la investigación, el desarrollo y la enseñanza[12].

Los gráficos (de barras, líneas, puntos, etc.) usados en la visualización más tradicional (horizontal) suelen estar construidos con un sistema de coordenadas cartesianas[13].



Pero para este trabajo se interesa en los gráficos que se pueden construir con un formato circular mediante el cambio del sistema de coordenadas a las polares[14]. Este formato ayuda a la comprensión en algunos conjuntos de datos, su estructura, comparaciones y relaciones. Este sistema en ocasiones ayuda a la atención del espectador, pero en algunas, como se dijo antes, lleva a cometer el fallo de crear gráficos más estéticos que funcionales, olvidando la meta de comunicar información de forma clara y eficaz.

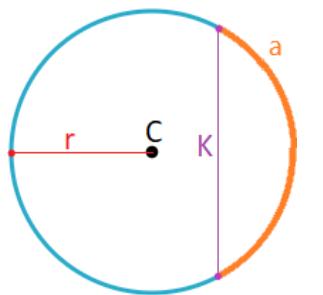
Para ello es necesario presentar el círculo y sus partes para entender cómo se construyen.

1.1.1. Círculos

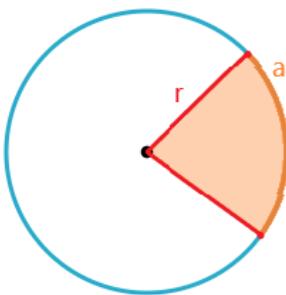
El círculo es un área o superficie plana contenida dentro de una circunferencia[5], o también se puede definir como un conjunto de puntos en un plano equidistantes de un punto dado, llamado centro[15].

Los elementos esenciales para comprender los diagramas y explicaciones del trabajo son:

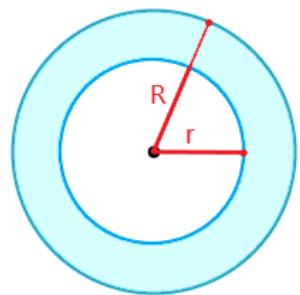
- **Centro:** punto fijo interior, equidistante de su perímetro a una distancia igual al radio. C, fig. 1.
- **Radio:** segmento que une el centro del círculo con cualquier punto del perímetro de este. r, fig. 1.
- **Cuerda:** segmento que une dos puntos del perímetro del círculo sin pasar por el centro. K, fig. 1.
- **Arco:** es la parte del perímetro del círculo que resulta entre dos extremos de una cuerda. a, fig. 1.
- **Sector circular:** parte del círculo comprendida entre dos radios y el arco que delimitan. fig. 2.
- **Círculos concéntricos:** dos o más círculos con mismo centro y distinto radio. fig. 3.
- **Corona circular:** superficie del círculo comprendida entre dos círculos concéntricos. fig. 3.



(a) fig. 1



(b) fig. 2



(c) fig. 3

2. Análisis de los Diagramas

Para poder usar cada uno de los diagramas, es necesario comprender sus virtudes, sus defectos y como se estructuran los datos para después elegir el más adecuado.

En esta sección se estudia la definición de los diagramas, cómo y de qué se componen , para poder identificar después qué paquetes de R pueden implementar dichos diagramas sus ventajas e inconvenientes a la hora de visualizar los datos requeridos respecto a otros formatos.

Esta selección de diagramas circulares se han escogido siguiendo el libro de Manuel de Lima [2] , donde se presenta una clasificación diferenciada en familias según la similaridad de construcción.

Estos se agrupan en siete familias:

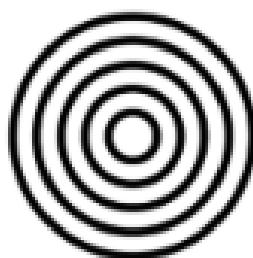
- Familia de anillos y espirales. Estos diagramas presentan patrones de círculos concéntricos.
- Familia de ruedas y tartas. Los diagramas que forman esta familia presentan patrones de líneas radiales.
- Familia de cuadrículas y retículas. La forman diagramas de cuadrículas circulares.
- Familia de flujos. Los diagramas que la conforman tienen patrones de flujo radiales con barras y líneas.
- Familia de formas y límites. En estos diagramas se delimita, con alguna variable, la forma de los objetos y sus límites.
- Familia de mapas y planos. El propio nombre de la familia indica que tipo de diagramas se concentran en esta.
- Familia de nodos y enlaces. Estos diagramas usan patrones de tipo árbol y redes.

2.1. Diagrama de Anillos

El diagrama de Anillos consiste en una división de círculos concéntricos, en los que en cada uno de ellos se pueden visualizar diferentes conjunto de datos.

La estructura de este diagrama, al consistir en varios círculos concéntricos, puede representar diferentes variables, diferentes tipos de gráfico o diferentes conjuntos de datos como, por ejemplo, el uso del ancho de los círculos puede ser preestablecido por una variable mientras el color de cada anillo por otra. Pero también se pueden crear diferentes tipos de gráficos en cada anillo, por ejemplo, representando un espacio de tiempo en los datos de cada círculo.

El problema con este diagrama es que la perspectiva de los círculos según su posición(interior o exterior), dificulta la comprensión. Si, por ejemplo, los mismos datos se implementan en un círculo exterior, resultarán con más magnitud que la que se sitúa en un círculo interior.



2.2. Diagrama de Barras Radiales

El diagrama de Barras Radiales o también llamado diagrama de Barras Circulares, es un gráfico de barras en el que, en lugar de usar un sistema de coordenadas cartesianas como el horizontal, se usa el sistema de coordenadas polares sobre el eje Y.

La estructura de este gráfico se crea con las barras formando círculos concéntricos sobre el círculo que actúa como plantilla, donde los divisores radiales sirven para marcar la longitud de las barras. Esta estructura tiene el mismo problema de malinterpretación de los valores. Éste radica en que, al ser el radio de cada barra diferente dependiendo de que posición ocupa en la plantilla, si se observan dos barras con valores iguales, la que esté posicionada en la parte exterior se interpreta como mayor magnitud que la que está en la parte interior. Es por eso que este diagrama, se usa más con propósitos estéticos que prácticos.



2.3. Diagrama en Espiral

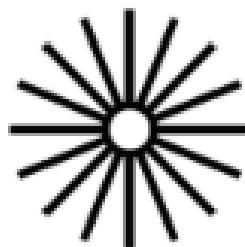
El diagrama en Espiral o espiral de series de tiempo expone datos con una variable temporal, siguiendo una curva que da vueltas indefinidamente alrededor de un punto, alejándose de él más, en cada una de ellas.

Este gráfico se usa primordialmente para representar el paso del tiempo en un conjunto de datos, haciendo coincidir el origen de la espiral con el comienzo temporal de los datos; para ello, se suelen usar barras, líneas o puntos para dibujar los datos que se exponen, dividiéndolos en rangos de tiempo para que resulten más fáciles de leer e implementar. Se puede dar otra dimensión al diagrama usando el color para expresar otra variable y así ayudar a la visualización. Este gráfico es uno de los más útiles para mostrar cómo se actualizan los datos en cada rango de tiempo, siendo ideal para mostrar patrones periódicos.



2.4. Diagrama de Columnas Radiales

Este diagrama, llamado de Columnas Radiales, de Columnas Circulares o de Estrella, es otra de las formas de crear un gráfico de barras con coordenadas polares, pero esta vez con el eje X. En este caso, los divisores radiales se convierten en las barras y la magnitud se calcula con la distancia del centro a su punto más exterior. Este gráfico tiene un problema con la dificultad con la que se visualiza la información, ya que nunca va a ser tan claro destacar una barra en un gráfico con coordenadas polares como uno con cartesianas. Las barras en este diagrama, se implementan del centro hacia el exterior; si se quiere representar valores negativos solo hace falta subir la escala 0 a concéntricos exteriores.

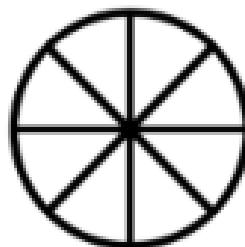


2.5. Diagrama de Sectores

El diagrama de Sectores o de Tarta es uno de los más conocidos y empleados (empresas, oficinas, etc). Este gráfico funciona dividiendo el círculo en sectores o porciones de tarta. La longitud de arco del sector representado coincide con el porcentaje del total que ocupen esos datos, coincidiendo el 100 por cien del total de los datos con los 360 grados del círculo.

Un problema de este tipo de diagrama, es que no se pueden representar grandes cantidades de datos, porque se perdería la ventaja de fácil visualización de la información. También es difícil hacer comparaciones entre varios gráficos de sectores.

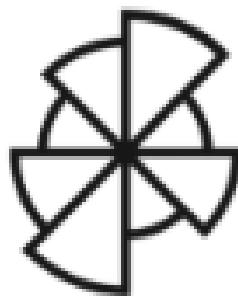
Su principal ventaja es que es muy útil para visualizar la distribución proporcional de un conjunto de datos.



2.6. Diagrama Rose of Nightingale

El gráfico Rose of Nightingale o de Área Polar debe su nombre a la enfermera Florence Nightingale, que sirvió en la guerra de Crimea. Lo usó para visualizar las muertes evitables durante esa guerra.

Se construye como un gráfico de sectores apilados donde cada segmento es un porcentaje del total. Para representar este gráfico, hay que tener en cuenta, que los segmentos no se estiman por su longitud sino que hay que usar el valor de su área. El problema principal de este gráfico, consiste en que resulta difícil calcular el valor del área en los sectores más externos.

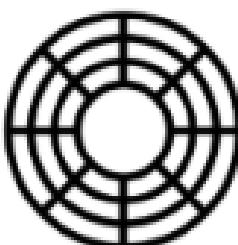


2.7. Diagrama de Cuadrícula Circular Ordenada

El diagrama de cuadrícula ordenada divide el círculo que utiliza como plantilla en diversas celdas, situadas según unos cortes radiales que, a su vez, se subdividen en círculos concéntricos, dejando una cuadrícula donde las celdas están unas encima de otras.

Este tipo de diagramas, al tener una estructura ordenada de celdas, ayuda a crear una correlación entre los datos situados en su vertical. Por ejemplo, si en cada celda de un anillo se sitúa un palabra de una lengua y en el interior su traducción a otro lenguaje, se crea así un correlación entre las celdas.

El problema con estos diagramas, es el que tienen en todos los diagramas que usan círculos concéntricos: las celdas posicionadas en los anillos exteriores tienen una dimensión diferente a los interiores ocasionando el problema de malinterpretación. La ventaja del diagrama es que aumenta la capacidad para visualizar el paralelismo entre diferentes grupos de datos.

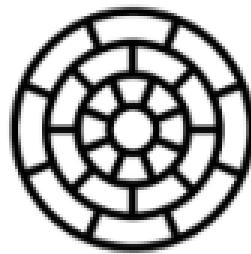


2.8. Diagrama de Cuadricula Circular Desordenada

Este diagrama, al igual que el anterior, utiliza la división en círculos concéntricos y en cortes radiales, pero dichos cortes no se sitúan a la misma distancia, obteniendo una cuadrícula con las celdas desordenadas.

El diagrama permite crear gráficos donde las celdas, no tienen que tener relación directa con las que conforman su vertical, ni una relación de padres a hijos como el de Rayos de Sol, sino que pueden ser relaciones 2:5, 3:2, etc.

Es difícil obtener datos que se puedan representar en este diagrama mejor que en diagramas de redes, por lo que se utiliza más con un propósito estético.

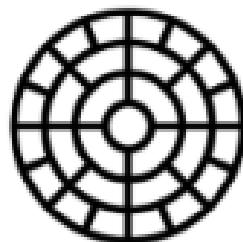


2.9. Diagrama Rayos de Sol

El diagrama Rayos de Sol, también llamado de Tarta Multinivel , de Anillo, de Cinturón o Mapa de Árbol Radial, es uno de los gráficos que representan un mapa de árbol con formato circular.

Reproduce el gráfico mapa de árbol, usando cada casilla de la cuadrícula circular como nodo del árbol. Cada nivel jerárquico es uno de los anillos del diagrama, siendo la raíz el centro y los nodos hoja las celdas exteriores. Para delimitar el ancho de cada celda, se puede usar una división equitativa del espacio o dejar la división al valor porcentual de una variable, pero siempre partiendo del nodo padre. Al igual que muchos gráficos de este trabajo, el color puede ayudar a la correcta visualización del diagrama.

Aunque este gráfico no está diseñado para representar grandes cantidades de datos, si se le añade el componente interactivo se puede llegar a solucionar este problema, al poder crear diferentes arboles simples dependiendo del nodo deseado.



2.10. Diagrama Radar

El diagrama Radial, de Araña o de Estrella (mismo nombre que Diagrama de Columnas Radiales), resulta de utilidad para comparar multiples variables cuantitativas.

Este gráfico consiste en dividir el círculo plantilla en tantas partes como variables se quieran comparar (partes equidistantes). Cada valor se representa en su eje (divisor radial) empezando su escala en el centro y terminando en el exterior. Luego, se suele llenar el polígono resultante de la unión de los valores del gráfico.

Al intentar usar una gran cantidad de variables, el gráfico puede resultar caótico; también, cuando se pretende usar 2 o más polígonos en el mismo círculo plantilla, resulta un diagrama confuso por la superposición de ellos. Por lo que este diagrama solo sirve para comparar pocas variables cuantitativas.

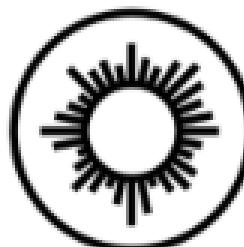


2.11. Diagrama de Columnas Con Formato Iris

El diagrama de Columnas Iris tiene la misma estructura que el Diagrama de Columnas Radiales, pero está vez estableciendo un espacio en el centro para simular la pupila de un ojo.

Este diagrama se suele utilizar, en los ejemplos de gráficos contemporáneos, poniendo énfasis en la cuantificación de las variables y su comparación. Se suele utilizar para representar conjuntos de datos a lo largo del tiempo o, como en el Diagrama de Columnas Radiales, para comparar una variable en diferentes situaciones.

La diferencia entre estos dos diagramas es el espacio en el centro que permite ser rellenado con otra información para cumplimentarlo.



2.12. Diagrama de Líneas Circulares Multiseries

El diagrama de líneas circulares multiseries procede, al igual que el anterior, del proceso de crear un espacio en el centro permitiendo añadir información para completar el diagrama. En este caso, se crea un diagrama de líneas o áreas transformado a formato circular. Al contrario que el Diagrama de Columnas Radiales, este diagrama no se puede realizar sin establecer un espacio en el centro, por el hecho de representar el eje X en su debida forma.

Este diagrama se suele utilizar para representar datos a lo largo del tiempo, para observar como fluctúan, debido a su forma continua de representar los datos al implementar los diagramas de líneas y área, con respecto a la forma discreta del diagrama de barras.

Como el anterior diagrama, éste se utiliza en gran cantidad de gráficos contemporáneos para enfatizar la cuantificación de sus variables, pero esta vez, siempre a lo largo del tiempo.



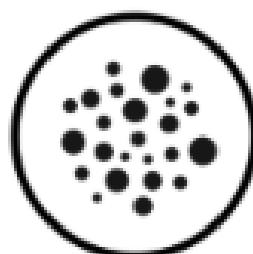
2.13. Diagrama de Burbujas

El diagrama de Burbujas se utiliza para realizar comparaciones y relaciones entre las burbujas según sus dimensiones o posición.

La estructura de este diagrama la conforman multitud de círculos dentro de otro mayor como plantilla. Se personalizan, usualmente, con 3 variables: el valor en el eje X, valor en el eje Y y el área. Otra variable que puede entrar en juego es el color. A diferencia del gráfico de burbujas normal, el posicionamiento de las burbujas tiene que simular el círculo que las encapsula.

El problema que presenta este diagrama, es que el gráfico resulta confuso si se crea con muchas burbujas. Esto puede ser solucionado gracias a la interactividad que se puede emplear para ver la información detrás de cada burbuja.

Una consideración importante sobre su estructura, es el hecho de que las burbujas se deben construir usando el área, no el diámetro ni el radio, lo cual provocaría un crecimiento exponencial de las dimensiones de las burbujas.

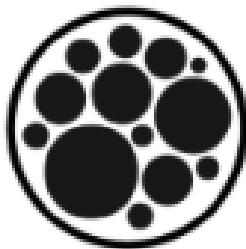


2.14. Diagrama de Mapa de Árbol Circular

Tal y como se vio en el Diagrama Rayos de Sol, aquí se presenta otra de las modalidades de reproducir un mapa de árbol con formato circular.

Este gráfico es una variante del mapa de árbol en el que se usan círculos en lugar de rectángulos para representar los nodos. La división del espacio del nodo-padre para crear a los hijos, puede depender de una variable, o de forma igualitaria no destacando a ningún hijo sobre otro. Y, para crear otra dimensión, se puede usar el color al igual que en los gráficos anteriores.

Al tener la estructura de círculos existe el problema de no tener una gran eficacia para cubrir espacios, dejando espacio vacío dentro de los nodos, pero con esta estética se consigue un mejor entendimiento de la estructura jerárquica detrás del diagrama.



2.15. Diagrama de Mapa de Árbol Voronoi

El diagrama de Voronoi es otra variante del mapa de árbol, pero usando los polígonos de Thiessen, llamados así por el meteorólogo Alfred H. Thiessen. Obtiene el nombre de Voronoi por el matemático ruso que los estudió, Gueorgui Voronoi.

Estos polígonos son estructuras geométricas que permiten crear una partición del plano euclídeo. Son la forma más eficiente de dividir un objeto, al usar uno de los métodos de interpolación más simples basado en distancias euclidianas. Las dimensiones de los polígonos se calculan según su área correspondiendo a alguna variable pre-definida.

El problema principal de estos diagramas es que son estáticos, es decir que, si se quiere añadir otra categoría al gráfico, habría que rehacerlo. Al igual que las anteriores variantes del mapa de árbol se usa el color para representar nodos conectados, pero en este caso esta variable resulta de gran importancia ya que representa a los nodos que forman parte de la misma familia ya que no están encapsulados en ningún otro polígono.



2.16. Diagrama de Planos Circulares

El diagrama de planos circulares consiste en los planos de construcciones con formato circular. Estos se han usado a lo largo de la historia por la atracción de la humanidad hacia los círculos, la facilidad de defensa (múrrallas) y concentración de espacios sin esquinas.

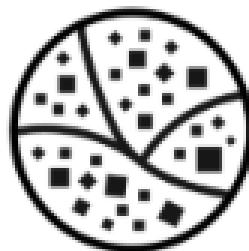
En este diagrama solo se representan los planos sin modificarlos para que encajen con el formato circular.



2.17. Diagrama de Mapa Circular

El diagrama de mapa, o buffer map[3], consiste en un círculo plantilla donde se expone un mapa, ya sea de relieve, carreteras, etc.

Este diagrama sirve para crear un mapa con la información de los alrededores de un punto, que se establece en el centro de él. Se utiliza en diferentes aplicaciones como: venta/alquiler de viviendas, localización de lugares para saber como llegar, etc.



2.18. Diagrama de Mapa de Esfera

El diagrama de mapa de esfera consiste en representar un mapa (relieve, carreteras, etc.) incluido en la forma de una esfera.

Este es un diagrama más artístico, ya que se suele usar en gráficos de mapa como si fueran cuadros.

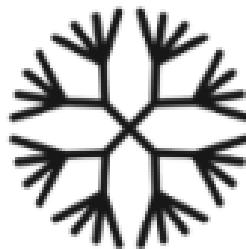


2.19. Dendograma Circular

Este diagrama es una variante del dendograma sólo que con un sistema de coordenadas polares. Para entender esta variante hay que ver qué es un dendograma; es una representación gráfica de una estructura jerárquica o árbol, organizando los datos en niveles jerárquicos, empezando de la raíz hasta los nodos hoja deseados.

Para esta variable de dendograma circular, la raíz se sitúa en el centro del círculo y, a sus hijos, en el círculo concéntrico más cercano, y así hasta llegar al círculo concéntrico exterior, donde se sitúan los nodos hoja. Para representar los grupos(cluster), se suele usar el color y, para las ramas, se usan líneas siguiendo el estilo más parecido de representar un árbol tradicional.

Esta variante, tiene el problema de no traducir igual de sencillamente la información de los datos al lector del diagrama, por lo que se utiliza más por estética que por funcionalidad, aunque, si se tiene que representar gran cantidad de datos, el formato circular ofrece una mayor eficiencia de espacio que su alternativa horizontal.

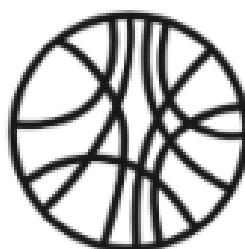


2.20. Diagrama de Cuerdas

El diagrama de cuerdas sirve para representar conexiones y relaciones observando qué y cómo comparten las entidades del gráfico.

Los nodos o entidades se sitúan en el círculo exterior y, para realizar las conexiones entre ellos, se usan arcos o curvas de Béizer, dejando a una variable porcentual determinar el ancho de cada conexión y la posibilidad de representar otra variable o, enfatizar una ya representada, con colores. Tiene el mismo problema que tienen casi todos los gráficos circulares; que al tener espacio limitado, si se intenta representar gran cantidad de entidades y/o conexiones, se vuelve un gráfico difícil de leer y comprender. Si se utiliza la interactividad para centrarse en una entidad o conexión, se da una solución a este problema.

Existe una variante de este diagrama llamado de Cuerdas sin Cintas que simplifica los nodos a puntos y los arcos a simples líneas, dando más importancia a las conexiones entre los nodos que a cómo se relacionan. Esta variante, al ser simplificada, admite una mayor cantidad de nodos y conexiones antes de convertirse en un diagrama confuso para el lector.

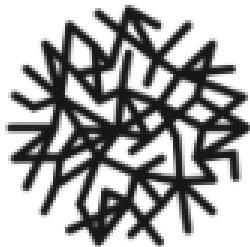


2.21. Diagrama de Redes Circulares

El diagrama de redes circulares es una variante de los de redes o, también llamados gráfico de red, mapa de red y de nodo-enlace, cuyo formato cambia a circular.

Los diagramas de redes son un tipo de visualización que muestra como las diferentes entidades, o nodos, se relacionan entre sí. Su estructura consta de las entidades, representadas como puntos, y las conexiones entre sí, como líneas que pueden ser dirigidas, para representar dependencia, o no, para simples conexiones.

Todos estos diagramas se han estudiado en el trabajo de Cristopher Calva Cárdenas[16] , que estudia, al igual que este proyecto, la implementación de los diferentes diagramas de redes con el lenguaje R. Por lo que se escogerán las variantes circulares de dichas redes.



3. Visualización en R

En este apartado recorre todo el peso del trabajo. Éste se centra en ver qué diagramas se pueden implementar con las funciones que provee el lenguaje R combinado con los demás paquetes.

Con cada diagrama se intenta buscar más de una manera de crearlos con diferentes paquetes, y poder establecer, diferentes formas de implementar dicho diagrama para posibles usos diferenciados de este.

En las siguientes subsecciones se expondrán los diagramas circulares que se han podido implementar con el lenguaje R.

3.1. Diagrama de Anillos

Este diagrama consiste en un grupo de círculos concéntricos. En cada uno de ellos se puede representar: una información diferente con diferentes gráficos o, la misma información con mismos tipos de gráficos solo que con una variable diferente para poder realizar comparaciones.

3.1.1. Paquete ggplot2

Se empieza con uno de los paquetes que más será usado a lo largo del trabajo[17] , debido a que es uno de los más versátiles y personalizables, pues se puede retocar y ajustar cada uno de los componentes del gráfico que resulte de este; como la escala, el sistema de coordenadas, la etiquetas, etc. También, una de las mejores características de este paquete, es su compatibilidad con otros paquetes de modificación de datos o de realización de gráficos.

Lo primero que se va a necesitar en todas las demostraciones de este trabajo es la obtención y manipulación de los datos. En este caso se necesitan 2 variables, indicando cuantos anillos componen el diagrama y el ancho de cada uno de ellos.

La primera forma de crear los anillos usa la función `geom-bar`, la cual reproduce el diagramas de barras clásico, obteniendo su magnitud de la variable que se asigne para indicar su ancho.

Los datos que se van a usar son: una serie de entidades y sus valores porcentuales. Los datos elegidos para representar en este diagrama es `mtcars`, que incluye 32 vehículos junto con su consumo, los carburadores que tienen, los caballos de fuerza, el número de marchas, etc. En este caso obtiene la frecuencia de los vehículos en la columna de carb, exponiendo en el ancho de cada anillo, la cantidad de coches por el número de carburadores .

```
datos <- data.frame(  
  ancho=table(mtcars$carb)  
)
```

En esta prueba se establecen 6 anillos designados el número de carburadores, como el nombre de entidades, y el ancho de cada uno es la cantidad de coches con esa característica.

Lo primero será cargar la librería ggplot2 para poder usar sus funciones.

```
library(ggplot2)
```

Para empezar con la función principal del paquete, es necesario indicar la fuente de sus datos y qué cargar en cada variable. El eje X indicará qué significa cada anillo, para este ejemplo, se deja vacío. Al eje Y se le asigna el ancho de los anillos y, para colorear cada anillo independientemente, se usa la variable fill (rellenar).

```
ggplot(datos, aes(x="", y=ancho.Freq, fill=ancho.Var1)) +
```

Una vez establecidos los parámetros iniciales se procede a usar la función para implementar en la plantilla, y, cómo se van a usar barras, el paquete ofrece la función `geom-bar`. Para este ejemplo, se va a diferenciar cada uno de los valores de Y para una barra (stat = identity) y se establece el ancho al 100 % de su valor y el color de los bordes de cada anillo es gris.

```
  geom_bar(stat="identity", width=1, colour="grey90") +
```

Ahora no es mas que un diagrama de barras tradicional, donde las barras tienen la longitud deseada, asi que, para hacer que las barras se conviertan en anillos, es necesario cambiar su sistema de coordenadas de cartesianas a polares, con la función `coord_polar`. Por defecto se establece que la primera variable con datos sea la que se cambia de sistema de coordenadas, en este caso el eje Y.

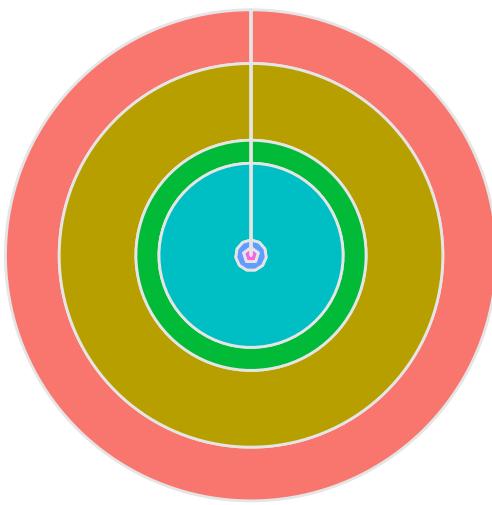
```
coord_polar() +
```

Por ultimo, se usan unas funciones para alterar el tema del gráfico como el fondo, la cuadrícula, la leyenda, etc. Pero, para este ejemplo, se prefiere una estética vacía y la leyenda de los colores.

```
theme_void() + labs(title = "Coches por Carburadores") +  
  labs(fill='N Carburadores') +
```

Resultando en:

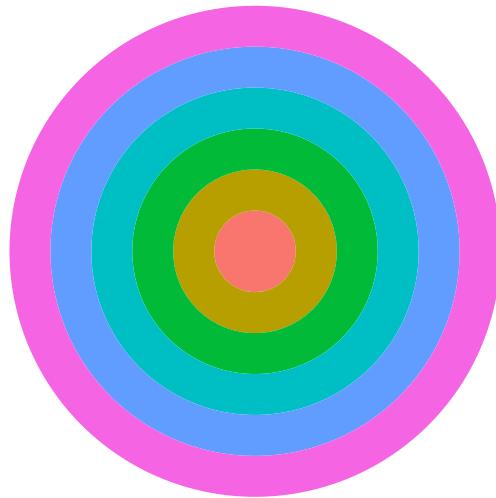
Coches por Carburadores



Otras de las funciones que pueden imitar al `geom-bar` son: `geom-col`, `geom-tile`. Para cada una de ellas, es necesario una aproximación diferente; `geom-col` es muy similar a bar solo que no hace falta el argumento `stat=identity`. La función `geom-tile` sirve para crear rectángulos y, para delimitarlos, no sirven los mismos datos que antes, ya que esto solo se pueden crear anillos con mismo valor, por ejemplo 1. Con esta función se puede representar la variedad de entidades que existen, por ejemplo, los diferentes valores en una columna de cualquier base de datos. Esta vez se utiliza el eje X para los anillos y, como antes, el eje Y para el ancho. Esta vez hay que especificar qué variable se transforma a coordenadas polares aunque, por lo demás, es lo mismo salvo que se experimenta a no colorear los bordes de los rectángulos.

```
datos <- data.frame(
  anillos=datos$ancho.Var1,
  ancho=c(rep(1,nrow(datos)))
)
ggplot(datos, aes(x=anillos, y=ancho)) +
  geom_tile(aes(fill = anillos)) +
  coord_polar('y') +
  theme_void() + labs(title = "Coches por Carburadores") +
  labs(fill='N Carburadores') +
  theme(legend.position = 'bottom', plot.title=element_text(hjust = 0.5))
```

Coches por Carburadores



3.1.2. Paquete circlize

Este es otro de los paquetes más versátiles para los gráficos con diseño circular[18] . Al contrario que ggplot2 que crea gráficos y necesita cambiarlos de sistemas de coordenadas, este paquete ya está diseñado para diagramas circulares.

Lo primero que se necesita hacer es establecer que no se divide cada anillo, ya que circlize permite la posibilidad de dividirlos generando una cuadrícula circular.

```
datos <- data.frame(
  division = sample(letters[1], 150, replace = TRUE),
  x = iris$Sepal.Length,
  y = iris$Sepal.Width
)
```

La variable división establece que cada anillo solo tendrá un vector con 150 datos y las otras dos variables se van a usar en los diferentes diagramas de cada anillo, correlacionando las variables de longitud y ancho de los sépalos, recogidos de la base de datos de plantas iris. Representando la variable X e Y de los gráficos líneas y puntos. Para la parte del histograma sólo se requiere de una variable de los datos, mostrando la frecuencia de los valores de la variable entre intervalos.

Se necesita crear una plantilla para cada diagrama. Esta se genera primero al inicializar el gráfico y luego con las funciones `trackPlotRegion` o `track` para realizar las plantillas de cada anillo.

```
library(circlize)
circos.par("track.height" = 0.22)
circos.initialize( factors=datos$division, x=datos$x)
```

Con la función `trackPlotRegion` se crea la plantilla con la altura que establece la escala de la variable Y, junto con la leyenda del eje X, por fuera del círculo (por defecto).

```
circos.trackPlotRegion(factors = datos$division, y = datos$y, panel.fun = function(x, y) {
  circos.axis()})
```

Para recrear los diferentes gráficos, se van a usar 3 métodos diferentes. El primero consiste en crear el gráfico de líneas después de crear la plantilla.

```
circos.trackLines(datos$division, datos$x[order(datos$x)], datos$y,
  col = rgb(0.1,0.5,0.8,0.3), lwd=2)
```

El segundo método se da cuando una de las funciones ya crea la plantilla; como `trackHist`.

```
circos.trackHist(datos$division, datos$x[order(datos$x)], col = rgb(0.1,0.5,0.8,0.3), lwd=2)
```

Por último, el tercer método consiste en crear la plantilla con `track` y dentro de esta se incluye la función que realiza el gráfico.

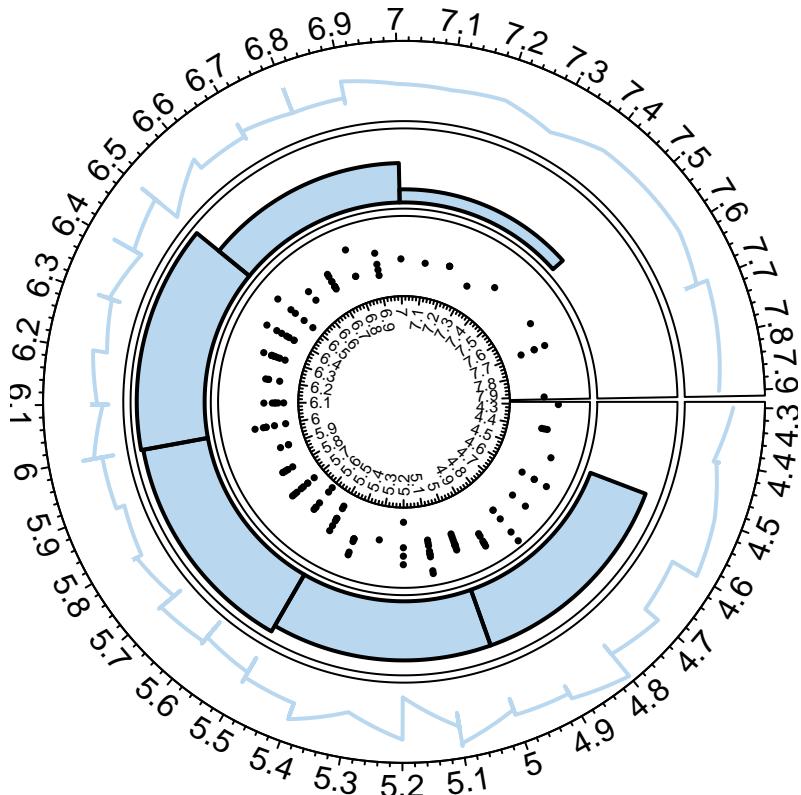
```
circos.track(datos$division, x = datos$x[order(datos$x)], y = datos$y,
  panel.fun = function(x, y) {
    circos.points(x, y, pch = 16, cex = 0.5)})
```

Todas las funciones usadas se pueden realizar con los tres métodos, ya que existe la función para usar sin la función `track` y la que se usa dentro de ella.

Al final se vuelve a representar el eje X para mejor visualización. Y se emplea `circos.clear` para limpiar las variables, para que al volver a crear con circlize todas sus variables no estén modificadas.

```
circos.axis(h="bottom", direction = "inside", labels.facing= "clockwise", labels.cex = 0.5)
circos.clear()
```

Resultando en el siguiente gráfico:



...

3.1.3. Paquete CMplot

Este paquete se sustenta en el diagrama Manhattan convirtiéndolo en una variable circular del Manhattan[19]. Este es un tipo de diagrama de dispersión, generalmente usado para representar datos con una gran cantidad de puntos y con una distribución de valores de gran magnitud.

Este diagrama se usa mucho para representar estudios de genoma, situando en el eje X la posición del cromosoma y en el eje Y el rango de asociación con un rasgo.

Para el ejemplo con este paquete, se va a hacer uso de uno de los datos que proporciona el propio paquete `pig60K`, el cual da los resultados de asociación de todo el genoma de 3 rasgos, cuyos individuos fueron genotipados por chip de cerdo 60k.

```
library(CMplot)
data(pig60K)
```

Ahora se entra en detalle de la función principal del paquete **CMplot**.

Los primeros argumentos consisten: en los datos que se van a usar, el tipo de diagrama (en este caso de puntos), el diseño del diagrama (circular), el radio, el color para los puntos y las etiquetas.

```
CMplot(pig60K,type="p",plot.type="c",r=0.4,col=c("grey30","grey60")
,chr.labels=paste("Chr",c(1:18,"X"),sep=""),
```

Se establece el umbral, el ancho para el límite, la amplificación de los puntos significativos, el tipo de línea y color para el umbral.

```
threshold=c(1e-6,1e-4),cir.chr.h=1.5,amplify=TRUE,threshold.lty=c(1,2)
,threshold.col=c("red","blue")
```

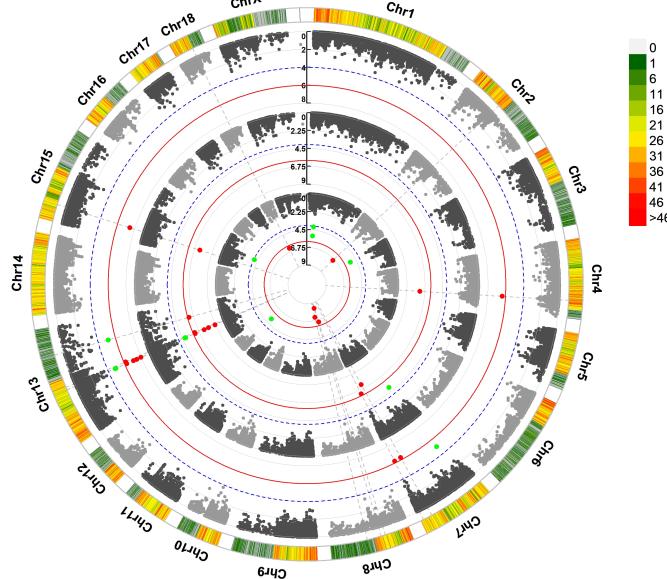
El ancho de la línea se establece con los SNPs significativos, su color, el color para la densidad SNP.

```
,signal.line=1,signal.col=c("red","green"),chr.den.col=c("darkgreen","yellow","red"),
```

El tamaño de bin para el gráfico de densidad SNP, se exponen los puntos de exterior hacia interior, el tipo de fichero que genera y la resolución de la imagen.

```
bin.size=1e6,outward=FALSE,file="jpg",dpi=300
,file.output=TRUE,verbose=TRUE,width=10,height=10)
```

Dando un resultado de:



3.1.4. Comparaciones

En este apartado, que se va a repetir en todos los diagramas que se han podido implementar con algún paquete de R, sirve para visualizar las diferencias o similitudes entre paquetes, en lo que respecta al resultado obtenido.

Para realizar este diagrama se han utilizado 3 paquetes:

El primero es el `ggplot2`, que implementa el diagrama de anillos utilizando de base un diagrama de barras tradicional (con sistema de coordenadas cartesianas), por lo que, en este sentido, el paquete está limitado y no ofrece otras variantes del diagrama de anillos, aunque, en cuanto a personalización se comprueba que se puede modificar cualquier parte estética del gráfico.

El segundo paquete es `circlize`, que al estar basado directamente en gráficos circulares, no necesita un cambio de sistema de coordenadas, y ofrece gran cantidad de gráficos para cada uno de los anillos. En ese sentido, es el mejor de los tres paquetes presentados. Y, en personalización, este paquete no se queda atrás debido a que también se puede retocar cualquier parte del diagrama.

Y por último, el tercer paquete es `CMplot`, consiste en una variante con diseño circular del paquete `qqman`, el cual, como se ha explicado antes, es un diagrama de Manhattan, por lo que este paquete está limitado como `ggplot2` a un solo tipo de diagrama para los anillos. Dentro del gráfico se pueden personalizar muchos aspectos del mismo.

En conclusión; el paquete más versatil a la hora de este diagrama es `circlize`, debido a que, con cada uno de sus gráficos, podría replicar los resultados de los otros dos. Para implementar gráficos de genomas el paquete `CMplot` es más facil de realizar que si se intenta imitar con `circlize`.

3.2. Diagrama de Barras Radiales

Hay varias formas de aproximarse para realizar este tipo de diagramas, una de ellas consiste en realizar un diagrama de barras tradicional y convertir su eje Y al formato radial, la otra forma, se realiza creando un círculo como plantilla y arcos como las barras (círculos concéntricos).

En este apartado se encuentran un paquete para la primera fórmula y dos para la segunda, pero todos los paquetes dan un gráfico muy similar. Aunque unos dan más posibilidades de personalización, son más complejos.

3.2.1. Paquete `ggplot2`

En el punto anterior ya se ha visto este paquete[17] y, durante todos los demás puntos, se demostrará lo versátil que es. En este tipo de grafo, lo que hay que hacer es un buen diagrama de barras tradicional con este paquete y aplicarle la coordenada polar, la cual transforma la coordenada deseada de formato lineal a circular. Primero se crea la tabla con el nombre de las barras y su valor, en este caso la importancia de Internet que tienen las categorías al tomar decisiones. Y se carga la librería para después usar sus funciones:

```
library(ggplot2)
Category <- c("Electronics", "Appliances", "Books", "Music", "Clothing",
             "Cars", "Food", "Hygiene",
             "Health/OTC", "Hair Care")
Percent <- c(81, 77, 70, 69, 69, 68, 62, 62, 61, 60)
internetImportance<-data.frame(Category,Percent)
```

Se transforma los nombres de las barras para que cuando se escriban se adhiera los valores a los nombres.

```
internetImportance$Category <-
  paste0(internetImportance$Category, " - ",internetImportance$Percent, "%")
```

Y se ordenan, de mayor a menor, por como trabaja la librería al crear el diagrama de exterior hacia el interior.

```
internetImportance$Category <-
  factor(internetImportance$Category,
         levels=rev(internetImportance$Category))
```

Y, después de esto, se realiza el plot del diagrama circular.

Gracias a esta función se prepara el espacio para recrear el gráfico con los parámetros básicos de los datos y la estética del gráfico. Las categorías se representan en el eje X y el porcentaje en el eje Y, rellenando la figura geométrica que quiera representar, en función de las categorías.

```
ggplot(internetImportance, aes(x = Category, y = Percent,
                               fill = Category)) +
```

Se va a elegir la figura geométrica de la barra para representar el gráfico que, por ahora, está en formato lineal.

```
geom_bar(width = 0.9, stat="identity") +
```

Después se transforma el eje Y del formato lineal al formato radial.

```
coord_polar(theta = "y") +
```

Para la estética del gráfico se eliminan las etiquetas de los ejes. Y representando los 360 grados del círculo al 100 % de los porcentajes.

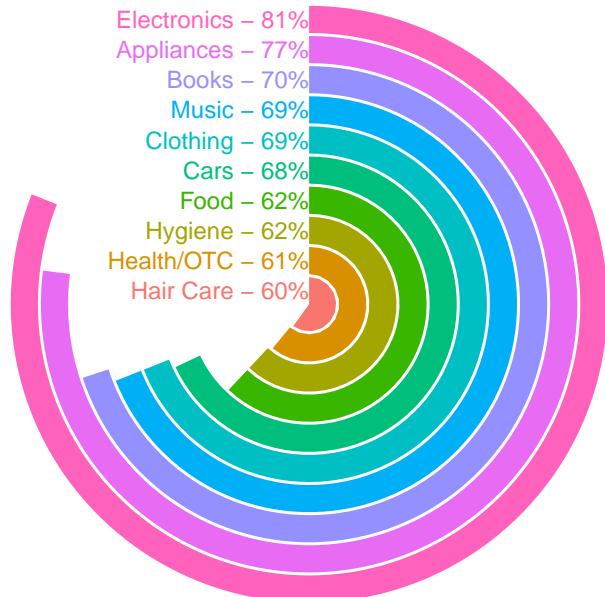
```
xlab("") + ylab("") +  
ylim(c(0,100)) +
```

Se establece el título y el texto de las barras, eliminando el fondo y la leyenda del gráfico para mejor representación.

```
ggtitle("Categorías de Productos Influenciados por Internet") +  
geom_text(data = internetImportance, hjust = 1, size = 3,  
aes(x = Category, y = 0, label = Category, colour = Category)) +  
theme_minimal() +  
theme(legend.position = "none",  
panel.grid.major = element_blank(),  
panel.grid.minor = element_blank(),  
axis.line = element_blank(),  
axis.text.y = element_blank(),  
axis.text.x = element_blank(),  
axis.ticks = element_blank())
```

Y la figura queda como se ve a continuación.

Categorías de Productos Influenciados por Internet



Ahora se va a implementar uno de los grafos del libro[2, pág 76] , para ello se vuelven a preparar los datos como antes:

```
Name <- rev(c("ANTIMONY", "INDIUM", "SILVER", "COPPER", "TITANIUM", "TANTALUM", "PHOSPHORUS",
           "ALUMINIUM", "GAS", "OIL", "COAL", "AGRICUL. LAND", "CORAL REEFS", "RAINFOREST"))
Value <- rev(c(8, 12, 17, 32, 44, 46, 76, 80, 35, 37, 42, 69, 88, 100))
Group <- rev(c("minerals", "minerals", "minerals", "minerals", "minerals", "minerals",
              "minerals", "minerals", "fossil fuels", "fossil fuels", "fossil fuels",
              "ecosystems", "ecosystems", "ecosystems"))
stockCheck<-data.frame(Name, Value, Group)
#Etiquetas
stockCheck$Label <- paste0(stockCheck$name, " ")
```

Los datos representan la cantidad de años que tardan las categorías en desaparecer. Para poder tener un espacio vacío en el centro hay que incluir varias líneas vacías, para que al implementarlas la función las borre, dejando el espacio en blanco

```
len <- 5
df2 <- data.frame(Name = letters[1:len], Value = rep(NA, len),
                    Group = rep(NA, len), Label = rep("", len))
stockCheck <- rbind(stockCheck, df2)
#ordenar por grupos
stockCheck$name <- factor(stockCheck$name,
                           levels = rev(stockCheck$name[order(stockCheck$Group)] ))
```

Y se realiza el grafo de la misma manera que la anterior, sólo que ahora, se deja espacio en el interior para que se visualice mejor el grafo. Al ser de la misma forma que el anterior, se va a mostrar solo las opciones que este gráfico tenga de forma diferente.

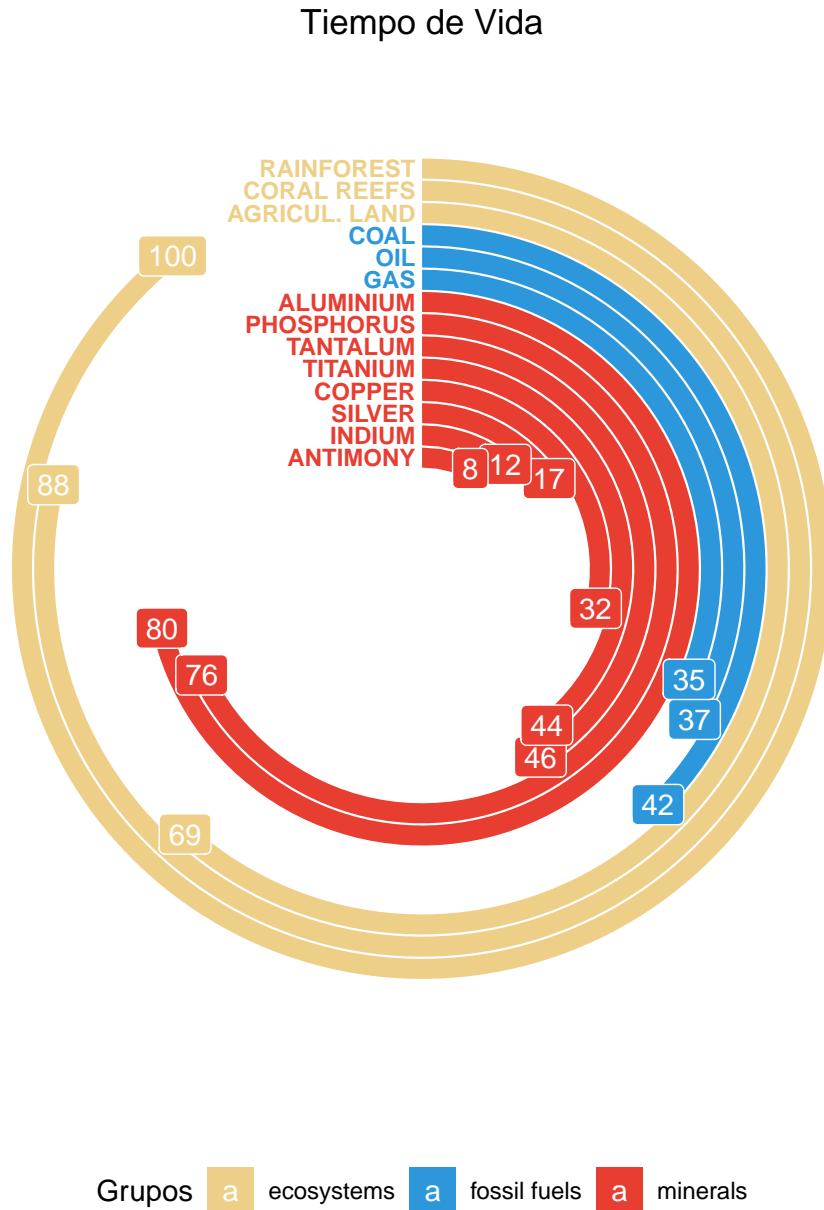
Una de las diferencias con el anterior es la forma de crear el texto con el color según los grupos, y las etiquetas que se sitúan al final de las barras, siendo cuadrados llenos con el color del grupo.

```
geom_text(data = stockCheck, hjust = 1, size = 3,
          aes(x = Name, y = 0, label = Label, colour = factor(Group),
              fontface = "bold")) +
geom_label(aes(fill = factor(Group)), colour = "white") +
```

Y se cambian los valores del color para asemejarlo al gráfico que se intenta implementar; tiene que ser 4 colores, debido a que hay un grupo de valores vacíos, para crear el hueco en el medio.

```
scale_fill_manual(values = c("#edcf87", "#2c98db", "#e83d31", "#e83d31" )) +
scale_color_manual(values = c("#edcf87", "#2c98db", "#e83d31", "#e83d31" )) +
```

Resultando en el siguiente gráfico.



Existe un paquete que proporciona interactividad a los gráficos del paquete ggplot2, y suele funcionar mejor cuando hay una gran cantidad de datos.

El paquete **ggiraph**[20] ayuda a crear la interactividad con algunas funciones que se unen al gráfico de ggplot2, en este caso, se usa **geom_col_interactive**, para crear las barras interactivas. Para que se represente bien hay que dar los valores adecuados a las nuevas variables **tooltip** y **data_id**.

Primero se necesita una gran cantidad de datos, vehículos en mtcars, creando los nombres de los vehículos (barras) y su consumo.

```
data <- mtcars  
data$nombre <- paste0(rownames(data), " - ", data$mpg)
```

Y ahora, asignar los parámetros adecuados al gráfico de ggplot2, con la función de columnas interactivas. Agrupando las barras por cantidad de marchas de cada coche.

```
library(ggiraph)
library(viridis)

## Loading required package: viridisLite

p <- ggplot(data, aes( x = nombre, y = mpg, tooltip = nombre,
                      data_id = nombre, fill = gear) ) +
  geom_col_interactive() +
  coord_polar(theta = 'y') +
  scale_fill_viridis() +
  xlab("") + ylab("") +
  ylim(c(0,36)) + labs(title = "Consumo de coches por cantidad de marchas")+
  theme_minimal() +
  theme(panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        axis.line = element_blank(),
        axis.text.y = element_blank(),
        axis.text.x = element_blank(),
        axis.ticks = element_blank())
```

Para la interactividad se necesita crear el objeto girafe (propio del paquete).

```
girafe(ggobj = p)
```

Dando como resultado el siguiente gráfico.

Consumo de coches por cantidad de marchas



3.2.2. Paquete Plotrix

El paquete plotrix proporciona una serie de plots específicos que ofrecen una fácil personalización (color, posición del texto...).

A lo largo del proyecto, se observará dichos grafos utilizarse en los diferentes tipos de diagramas circulares. En la documentación [21] se encuentran los métodos para crear este Diagrama de Barras Radiales, `draw.circle` y `draw.arc`, que se encargan de recrear los círculos que se usarán de plantilla para que se expongan las barras. La orden `draw.arc` necesita conocer en cual ángulo comienzan las barras y en cual terminan, además de los datos de las barras.

Primero llamar a la librería.

```
library(plotrix)
```

Luego se crea la función para crear el diagrama.

```
circBarPlot <- function(x, labels, colors=rainbow(length(x)), cex.lab=1) {
```

Se establece el área para delimitar donde se realiza el círculo.

```
plot(0,xlim=c(-1.2,1.2),ylim=c(-1.2,1.2),type="n",axes=F, xlab=NA, ylab=NA)
```

Se guardan los datos de las barras en una variable.

```
radii <- seq(1, 0.3, length.out=length(x))
```

Constituir la plantilla para los datos y el ángulo de inicio del diagrama.

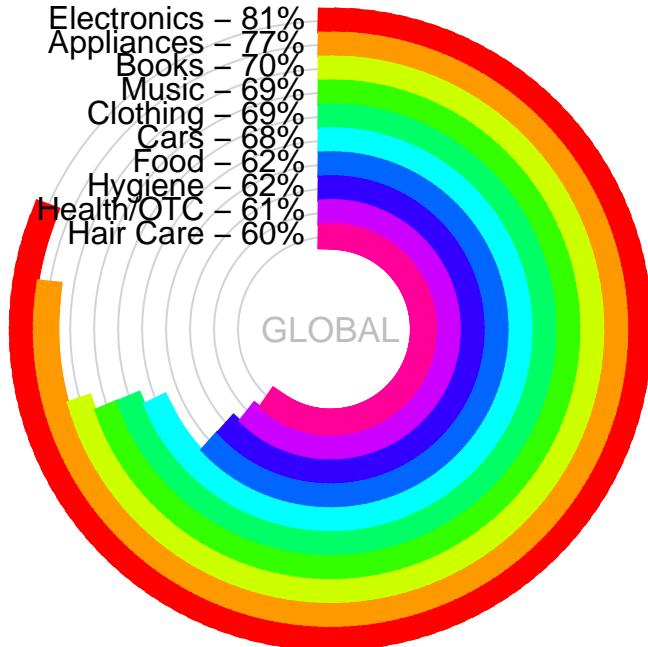
```
draw.circle(0,0,radii,border="lightgrey")
angles <- (1/4 - x)*2*pi
```

Por último se exponen los arcos y la leyenda de cada barra.

```
draw.arc(0, 0, radii, angles, pi/2, col=colors, lwd=130/length(x), lend=2, n=100)
ymult <- (par("usr")[4]-par("usr")[3])/
  (par("usr")[2]-par("usr")[1])*par("pin")[1]/par("pin")[2]
text(x=-0.02, y=radii*ymult, labels=paste(labels, " - ", x*100, "%", sep=""),
  pos=2, cex=cex.lab)
```

Después se usará con los datos del primer gráfico.

```
circBarPlot(Percent/100, Category)
# texto en el centro
text(0,0,"GLOBAL",cex=1.1,col="grey")
```



...

3.2.3. Paquete circlize

El paquete circlize usa la filosofía de descomponer el círculo en figuras geométricas más simples, como líneas y puntos, implementando grafos circulares a partir de gráficos más simples; “implementa funciones gráficas de bajo nivel para agregar gráficos en las regiones de trazado circular”.[18]

Dichas funciones simples son: puntos, líneas, rectángulos, texto, etc. Y, para organizar las regiones de trazado circulan, existen las siguientes funciones: inicializar, actualizar, trazado, limpiar, etc.

En este apartado, se utilizará las funciones de par (parámetros del grafo), inicializar (asigna los sectores al círculo), trazado(crear las regiones de trazado), rectángulos (crea las barras), texto y los ejes.

Se vuelven a preparar los datos para este plot debido a como expone.(De interior a exterior)

```
color = rainbow(length(Percent))
Category <- rev(Category)
Percent <- rev(Percent)
color <- rev(color)
library(circlize)
```

Se inicializa el círculo para que el diagrama empiece a los 90 grados. Esta parte es opcional, pues solo tiene argumentos estéticos y no necesarios para crear un gráfico con este paquete.

```
circos.par("start.degree" = 90, cell.padding = c(0, 0, 0, 0))
```

Se inicializa la plantilla del diagrama no dividiendo el anillo donde se crea el gráfico y fijando los límites del eje X, igualando los 360 grados de la circunferencia al 100 % de los valores de las barras.

```
circos.initialize("a", xlim = c(0, 100))
```

La función para crear el anillo donde se va a realizar el diagrama de Barras Radiales, con los parámetros del límite del eje Y, la altura del anillo en porcentaje (para dejar espacio en medio del diagrama), la eliminación de los bordes y el establecimiento del inicio de la función que establece qué hay dentro del anillo.

```
circos.track(ylim = c(0.5, length(Percent)+0.5), track.height = 0.8,
            bg.border = NA, panel.fun = function(x, y) {
              xlim = CELL_META$xlim
```

Se crean las barras con `circos.rect` recreando los rectángulos con el uso de las variables xInicial, yInicial, xFinal y yFinal, junto a la leyenda de cada una, gracias a la función `circos.text`.

```
circos.rect(rep(0, 10), 1:10 - 0.45, Percent, 1:10 + 0.45,
            col = color, border = "white")
circos.text(rep(xlim[1], 10), 1:10,
            paste(Category, " - ", Percent, "%"),
            facing = "downward", adj = c(1.05, 0.5), cex = 0.8)
```

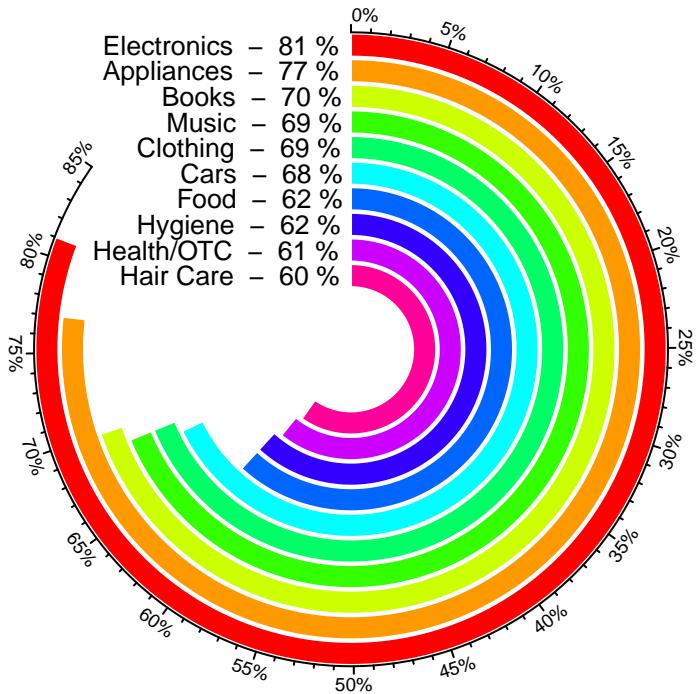
Para una mejor comprensión se pone la leyenda del eje X por el anillo exterior, colocando dicha leyenda según el parámetro `break` que situa cortes cada 5 unidades.

```
breaks = seq(0, 85, by = 5)
circos.axis(h = "top", major.at = breaks, labels = paste0(breaks, "%"),
            labels.cex = 0.6})
```

Y, por último, para no crear conflictos a la hora de crear nuevos gráficos con este paquete, se limpian las variables contenidas en `circos`.

```
circos.clear()
```

Resultando al final en:



• • •

3.2.4. Comparaciones

Se usan tres paquetes, dos de ellos seguirán apareciendo por todo el trabajo gracias a sus versatilidad.

El paquete `ggplot2`, se vuelve a usar de la misma forma que en el diagrama de anillos, reproduciendo un diagrama de barras y cambiando su sistema de coordenadas. Este paquete, en lo que a este diagrama se refiere, ofrece una gran personalización y un correcto/facil uso de las funciones para crearlo. En uno de los ejemplos, este paquete puede juntarse con el paquete `ggiraph`, para proporcionar la variable de interactividad a este gráfico, ampliando su versatilidad para crear diagramas de barras radiales con gran cantidad de barras.

Para el segundo paquete se ha encontrado `plotrix`; este paquete crea un gráfico similar a los otros dos paquetes, pero hay que ir creando los círculos plantilla para después realizar los arcos que actúan como las barras del diagrama. También con este paquete la personalización del diagrama es muy amplia.

El otro paquete que se ha usado en el anterior diagrama, `circlize`, usa un método ya explicado, que consiste en crear rectángulos siguiendo una plantilla circular y, al igual que en el anterior diagrama, se pueden personalizar muchos aspectos del gráfico.

En conclusión, la mejor opción para crear estos diagramas de barras radiales es el paquete `ggplot2`, debido a la facilidad que tiene de crear los diagramas de barras tradicionales y cambiar su sistema de coordenadas. Esto lo diferencia de los otros dos. Todos los paquetes son igual de personalizables, solo que en `ggplot2` es más fácil de realizar.

3.3. Diagrama Espiral

Este diagrama se puede usar para representar datos a lo largo del tiempo, por lo que la preparación de dichos datos, es lo más importante en éste. Para obtener este grafo, habrá que tener unos datos adecuados, que suelen implicar tres variables; una para ver los grados del círculo, otro para su altura dentro del círculo y por último, a veces puede haber otra variable para generar el tamaño de la figura geométrica que se decida (barra, círculo, etc.).

3.3.1. Paquete `ggplot2`

Se vuelve a ver este paquete[17] debido a su versatilidad, pero ahora se depende mas de los datos que del paquete, este usa tres posibles generadores de barras: `rect`, `segment` y `title`. Estos datos son el tiempo en movimiento de una persona durante 5 días, indicando la hora cuando se recogieron dichos datos.

```
library(dplyr)
library(readxl)
library(ggplot2)
dat = read_excel("Data1.xlsx")
head(dat, n=5)

## # A tibble: 5 x 4
##   ...1 Date1      Time `Travel Time`
##   <chr>     <chr>     <chr>    <dbl>
## 1 1     2016-09-04 13:11 34.65
## 2 2     2016-09-04 13:12 34.35
## 3 3     2016-09-04 13:13 33.2 
## 4 4     2016-09-04 13:14 33.016666666667
## 5 5     2016-09-04 13:15 33.25
```

Ahora se procede a realizar la parte más importante de tratar los datos para poder visualizarlos, dar a las fechas un formato más manejable.

```
dat$time = with(dat, as.POSIXct(paste(Date1, Time), tz="GMT"))
```

Se obtienen las horas y los días.

```
dat$hour = as.numeric(dat$time) %% (24*60*60) / 3600
dat$day = as.Date(dat$time)
```

Reformar el tiempo en movimiento pasando a numeros, acortando decimales y generar tramos manejables de datos con rango de 25 min.

```
names(dat)[grep("Travel", names(dat))] = "TravelTime"
dat$TravelTime = as.numeric(dat$TravelTime)
dat.smry = dat %>%
  mutate(hour.group = cut(hour, breaks=seq(0,24,0.25), labels=seq(0,23.75,0.25),
                         include.lowest=TRUE),
        hour.group = as.numeric(as.character(hour.group))) %>%
  group_by(day, hour.group) %>%
  summarise(meanTT = mean(TravelTime)) %>%
  mutate(spiralTime = as.POSIXct(day) + hour.group*3600)
```

Quedando el siguiente formato de datos:

```
head(dat, n=5)

## # A tibble: 5 x 7
##   ...1 Date1     Time  TravelTime time          hour day
##   <chr>    <chr>     <chr>      <dbl> <dttm>        <dbl> <date>
## 1 1     2016-09-04 13:11     34.6 2016-09-04 13:11:00  13.2 2016-09-04
## 2 2     2016-09-04 13:12     34.4 2016-09-04 13:12:00  13.2 2016-09-04
## 3 3     2016-09-04 13:13     33.2 2016-09-04 13:13:00  13.2 2016-09-04
## 4 4     2016-09-04 13:14     33.0 2016-09-04 13:14:00  13.2 2016-09-04
## 5 5     2016-09-04 13:15     33.2 2016-09-04 13:15:00  13.2 2016-09-04
```

Para el primer grafo se usará el método `rect`.

La plantilla, con los datos estableciendo las variables necesarias para crear este gráfico, como la x mínima, la x máxima, la y mínima ,delimitada por el comienzo del tiempo, la y máxima, por el tiempo final del diagrama, y con qué variable se colorearan los rectángulos.

```
ggplot(dat.smry, aes(xmin=as.numeric(hour.group), xmax=as.numeric(hour.group) + 0.25,
                      ymin=spiralTime, ymax=spiralTime + meanTT*1500, fill=meanTT)) +
```

Se crean las barras a cada rango de 25 minutos dado ya por los datos.

```
geom_rect(color="grey40", size=0.2) +
```

Se ajustan los ejes de las horas y la altura donde empieza cada barra.

```
scale_x_continuous(limits=c(0,24), breaks=0:23, minor_breaks=0:24,
                   labels=paste0(rep(c(12,1:11),2), rep(c("AM","PM"),each=12))) +
scale_y_datetime(limits=range(dat.smry$spiralTime) + c(-2*24*3600,3600*19),
                  breaks=seq(min(dat.smry$spiralTime),max(dat.smry$spiralTime),"1 day"),
                  date_labels="%b %e") +
```

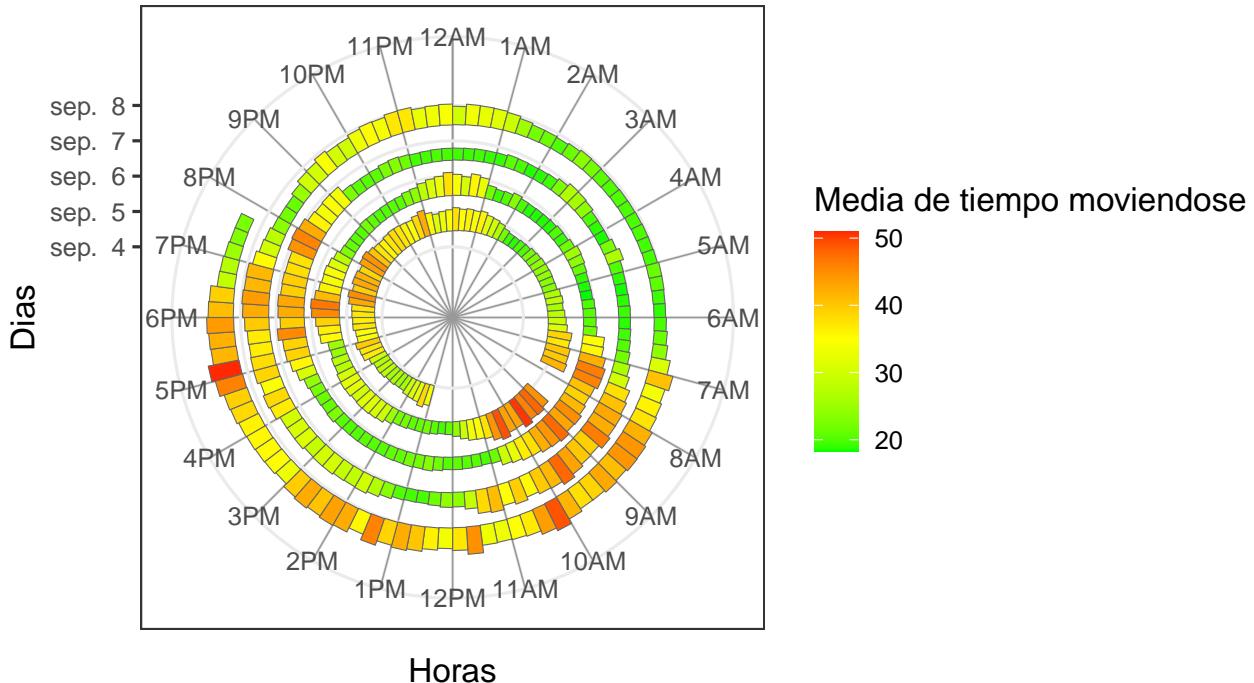
Se fija el color verde para los rangos de poco movimiento, amarillo medio y rojo alto.

```
scale_fill_gradient2(low="green", mid="yellow", high="red", midpoint=35) +
```

Y transformar la coordenada x a sistema de coordenadas polares y se ajustan las estéticas del tema como el fondo, leyenda, cuadrícula de la escala, etc.

```
coord_polar() +
theme_bw(base_size=13) +
labs(x="Horas",y="Días",fill="Media de tiempo moviéndose") +
theme(panel.grid.minor.x=element_line(colour="grey60", size=0.3))
```

Resultando en:



...

Para el segundo grafo se usará el método **segment**.

Primero se usa la misma plantilla que el ejemplo anterior.

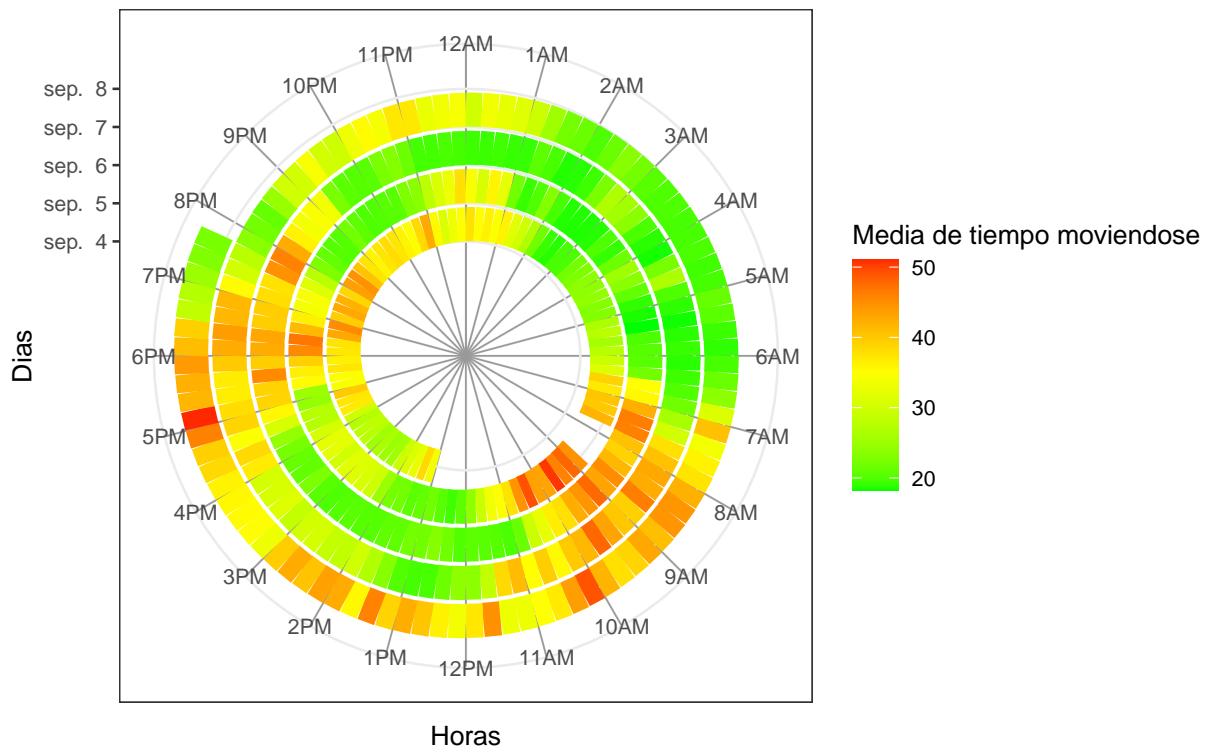
```
ggplot(dat.smry, aes(x=as.numeric(hour.group), xend=as.numeric(hour.group) + 0.25,
y=spiralTime, yend=spiralTime, colour=meanTT)) +
```

Luego, en vez de usar las barras se van a usar segmentos, los cuales, dejan una espiral con el mismo tamaño, con la única diferencia de los colores.

```
geom_segment(size=6) +
```

Y después de usar los mismos métodos que antes se obtiene el siguiente gráfico.

```
scale_x_continuous(limits=c(0,24), breaks=0:23, minor_breaks=0:24,
labels=paste0(rep(c(12,1:11),2), rep(c("AM","PM"),each=12))) +
scale_y_datetime(limits=range(dat.smry$spiralTime) + c(-3*24*3600,0),
breaks=seq(min(dat.smry$spiralTime), max(dat.smry$spiralTime),"1 day"),
date_labels="%b %e") +
scale_colour_gradient2(low="green", mid="yellow", high="red", midpoint=35) +
coord_polar() +
theme_bw(base_size=10) +
labs(x="Horas",y="Días",color="Media de tiempo moviéndose") +
theme(panel.grid.minor.x=element_line(colour="grey60", size=0.3))
```



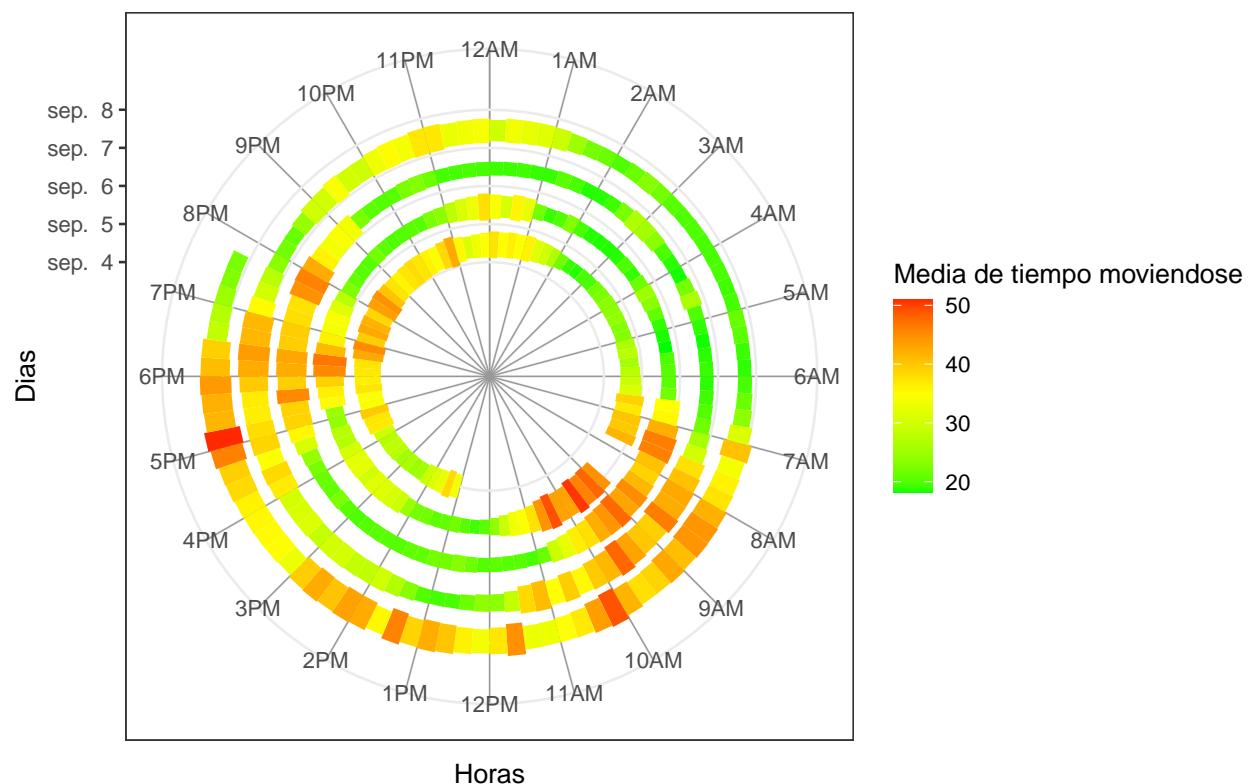
...

Y para el último grafo se usará el método **title**.

Con este método se puede variar su tamaño para dar una forma similar al primer ejemplo.

```
ggplot(dat.smry, aes(x=as.numeric(hour.group) + 0.25/2, xend=as.numeric(hour.group) + 0.25/2,
y=spiralTime, yend=spiralTime, fill=meanTT)) +
geom_tile(aes(height=meanTT*1800*0.9)) +
scale_x_continuous(limits=c(0,24), breaks=0:23, minor_breaks=0:24,
labels=paste0(rep(c(12,1:11),2), rep(c("AM","PM"),each=12))) +
scale_y_datetime(limits=range(dat.smry$spiralTime) + c(-3*24*3600,3600*9),
breaks=seq(min(dat.smry$spiralTime),max(dat.smry$spiralTime),"1 day"),
date_labels="%b %e") +
scale_fill_gradient2(low="green", mid="yellow", high="red", midpoint=35) +
coord_polar() +
theme_bw(base_size=12) +
labs(x="Horas",y="Días",fill="Media de tiempo moviéndose") +
theme(panel.grid.minor.x=element_line(colour="grey60", size=0.3))
```

Dando el resultado de:



3.4. Diagrama de Columnas Radiales

Este diagrama es una variante de los diagramas más usados (diagrama de barras) y, como se vió en el apartado Subsección 3.2 , esta es otra variante que en vez de posicionar el valor de las barras en el eje X lo situá en el eje Y quedando el eje X para los nombres de las barras. Para realizar este tipo de diagramas, la gran mayoría de paquetes lo afrontan creando un diagrama de barras con sistema de coordenadas cartesianas y transforman el sistema del eje X a polares, pero otros paquetes simplemente crear el diagrama cambiando la dirección de las barras.

3.4.1. Paquete ggplot2

Con este paquete [17] se procede a copiar la manera de crear el diagrama de Barras Radiales Subsección 3.2 pero, en esta ocasión, la coordenada a transformar esta vez se la X.

Primero se va a crear los datos de ejemplo para representar en este gráfico, estableciendo las entidades (barras) y sus valores, junto con la llamada de la librería. Se escoge una base de datos de multas por velocidad (<https://vincentarelbundock.github.io/Rdatasets/datasets.html>) recogiendo la velocidad a la que iba el vehículo multado, siendo el eje X la velocidad y el eje Y la cantidad de multas que hay de dicha velocidad.

```
data <- read.csv("amis.csv", header = TRUE)
data <- data.frame(
  velocidad=table(data$speed)
)
library(ggplot2)
head(data,5)

##   velocidad.Var1 velocidad.Freq
## 1              26        103
## 2              27        101
## 3              28        198
## 4              29        212
## 5              30        330
```

Para implementar este gráfico se escogen los datos, asignando los ejes correspondientes y la figura geométricas de la barra.

```
ggplot(data, aes(x=velocidad.Var1, y=velocidad.Freq)) +
  geom_bar(stat="identity", fill=alpha("lightblue")) +
```

Esta vez se puede controlar el espacio del centro mediante la variable `ylim`, en lugar de tener que llenar los datos con valores vacíos para luego no representarlos.

```
  ylim(-10,600) +
```

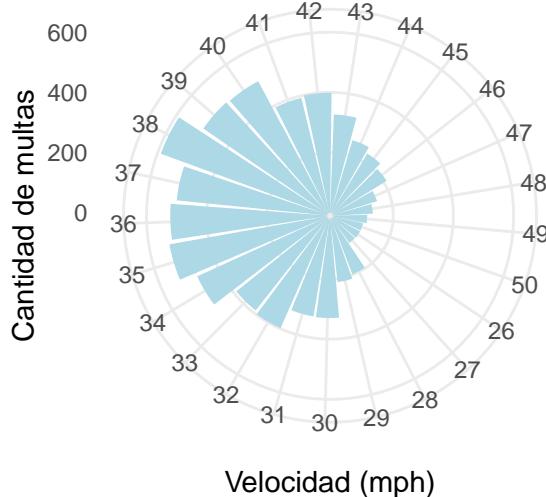
Se transforma el eje x de lineal a radial y se hace coincidir el principio del gráfico con el principio del círculo (0 grados).

```
  coord_polar(start = 90) +
```

Y, por último, se ajusta la estética deseada ajustando el fondo y elementos como el texto de los ejes.

```
  theme_minimal() +
  xlab("Velocidad (mph)") + ylab("Cantidad de multas")
```

Quedando el gráfico de la siguiente forma.



...

Como se ha observado con anterioridad, este paquete deja mucho espacio para la personalización de estos gráficos, por lo que se va a implementar el mismo gráfico que antes con grupos con espacios diferenciados y etiquetas.

Para crear los espacios entre los grupos, se va a añadir al igual que en el diagrama de barras radiales, barras con datos vacíos para que dejen el espacio al no ser representados. Se necesitan unos datos agrupados por alguna variable, para ello se escoge la base de datos de mtcars exponiendo el consumo de cada coche, y agrupandolos por la cantidad de marchas que tienen.

```
datos<- mtcars  
datos$name <- rownames(datos)
```

Se suman varias barras que actúen de espaciadores. Creando una matriz vacía con 8 barras por cada grupo y juntarlo a los datos.

```
empty_bar <- 8  
to_add <- data.frame( matrix(NA, empty_bar*nrow(table(mtcars$gear)), ncol(datos)) )  
colnames(to_add) <- colnames(datos)  
to_add$gear <- rep(rownames(table(datos$gear)), each=empty_bar)  
to_add$name <- rep("", each=empty_bar)  
datatrans <- rbind(datos, to_add)
```

Ahora se reordenan los datos por sus marchas y se establece una variable para representar todas las filas.

```
library(tidyverse)  
datatrans <- datatrans %>% arrange(gear)  
datatrans$id <- seq(1, nrow(datatrans))
```

Para la parte de las etiquetas habrá que poner el datos que se quiera visualizar en todas las columnas y ajustar que no se solapen entre sí. Esto se puede conseguir gracias a la librería ggrepel.

```
library(ggrepel)
```

Para ahora crear el gráfico usamos las siguientes funciones.

Lo primero que es distinto es el hecho de que pinta las columnas dependiendo del grupo (marchas), y el uso de ggrepel para el texto de las barras.

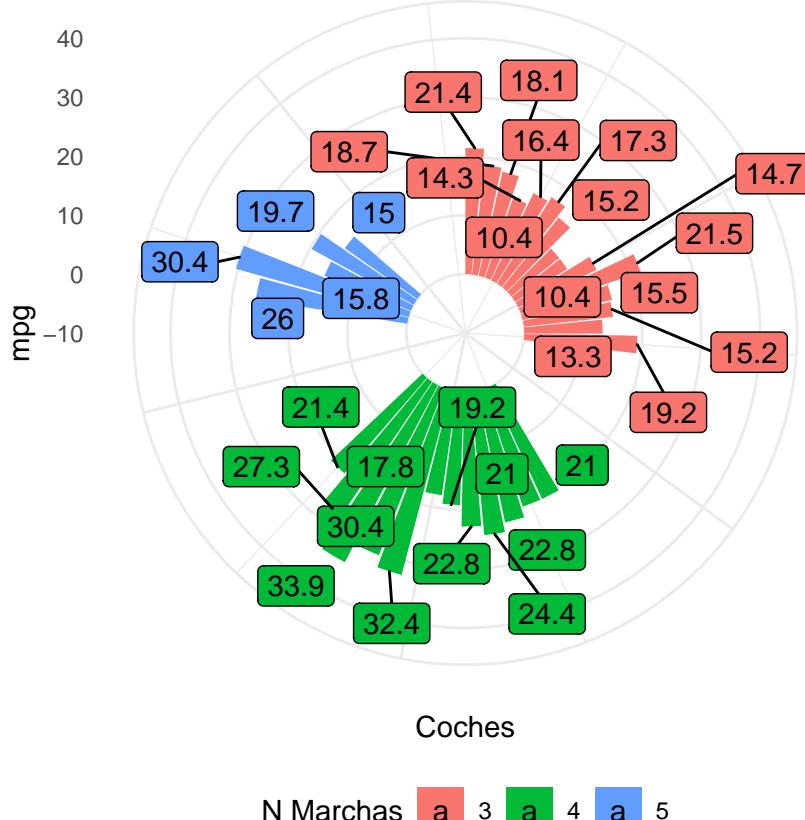
```
ggplot(datatrans , aes(x=id, y=mpg, fill= gear, label=mpg))+  
  geom_bar(stat = "identity") +  
  geom_label_repel()
```

Por lo demás, todo es igual con el ajuste del sistema coordenadas, límite del eje Y, nombre de ejes y legenda, junto al tema.

```
coord_polar() +  
  ylim(-10,40) + labs(title = "Coches por Marchas") +  
  xlab("Coches") + labs(fill='N Marchas') +  
  theme_minimal() +  
  theme(axis.text.x = element_blank(), legend.position = 'bottom')
```

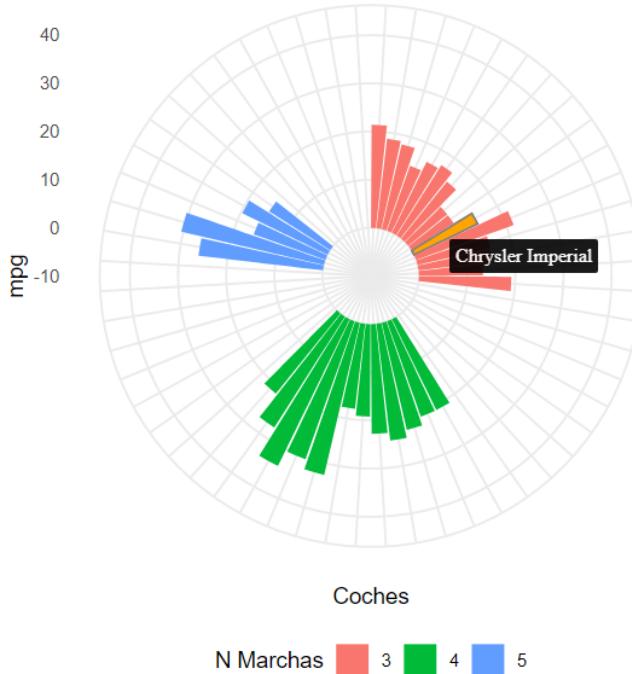
Resultando en este último gráfico

Coches por Marchas



Existe una forma de realizar este mismo gráfico con interactividad con la ayuda de ggplot2 y ggiraph. Ya se ha explicado en otros diagramas comó, así que, en este solo se enseña el resultado.

```
library(ggiraph)
p <- ggplot(datatrans, aes(x=as.factor(id), y=mpg, tooltip=as.factor(name),
  data_id = as.factor(id), fill = gear)) +
  geom_col_interactive() +
  ylim(-10,40) +
  coord_polar(start = 0) +
  xlab("Coches") + labs(fill='N Marchas') +
  theme_minimal() +
  theme(axis.text.x = element_blank(), legend.position = 'bottom')
girafe(ggobj = p)
```



3.4.2. Paquete plotrix

Como se dijo, Subsección 3.2 , este paquete sería usado para otros gráficos más adelante por sus plots específicos. Y gracias a las funciones `polar.plot` y `radial.plot` se pude contar con él en este apartado. Para ello se establecen los datos usados, volviendo a usar la base de datos de multas de vehículos por velocidad. Primero la longitud de las columnas con las cantidad de multas según la velocidad.

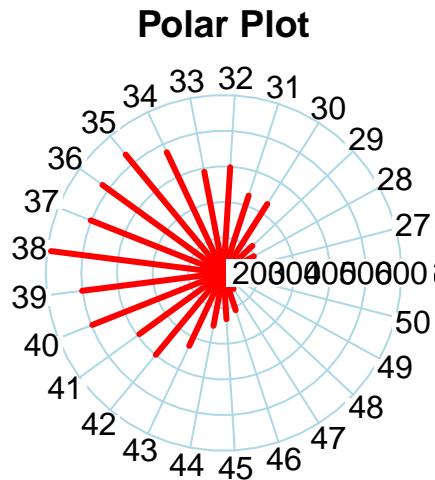
```
long <- data$velocidad.Freq
```

Luego la posición de cada columna en el círculo, siempre en grados. Dividiendo los grados por la cantidad de columnas a representar.

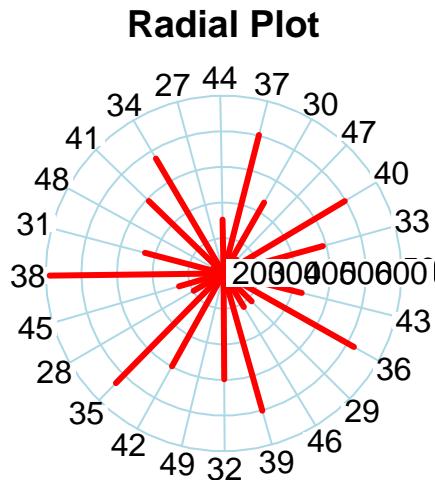
```
pos <- seq(0,345.6,by=14.4)
```

Para luego usar las dos funciones, estas son capaces de usar los mismos parámetros.

```
library(plotrix)
polar.plot(long, pos, main="Polar Plot", lwd=3, line.col=2,
           rad.col= "lightblue", grid.col= "lightblue", start = 0,
           labels=data$velocidad.Var1, label.pos = pos)
```



```
radial.plot(long, pos, main="Radial Plot", lwd=3, line.col=2,
            rad.col= "lightblue", grid.col= "lightblue", start = 0,
            labels=data$velocidad.Var1, label.pos = pos)
```



Cómo se ve, los gráficos no han salido iguales y es debido a que `polar.plot` transforma la posición de los grados a radianes mediante la función `radial.plot`, pero con `radial.plot` la variable `pos` se interpreta como radianes y no grados.

3.4.3. Paquete cplots

Este paquete, tal y como dice su documentación [22] , proporciona algunos diagramas circulares provenientes de datos circulares, incluyendo diagramas de barras, de densidad, puntos apilados, etc.

Para reproducir este diagrama con este paquete se debe preparar los datos. Al solo haber una versión de este paquete, no hay una buena documentación para preparar datos diferentes a los dados por el paquete. (descatalogado a 08/05/2020)

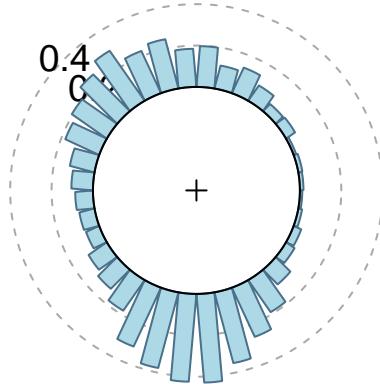
```
library(cplots)
library(circular)
x = c(rvonmises(2000, circular(2*pi/3), 2),
      rvonmises(2000, circular(3*pi/2), 5))
```

La función `rvonmises` está dentro del paquete `circular` y sirve para tener datos de forma circular donde se introduce el número de muestras, la distribución de las muestras indicando en radianes donde se encuentra dentro del círculo y la concentración de las muestras a la distribución pasada.

Y para crear el gráfico de columnas radiales se usa `cbarplot`, el cual necesita los datos anteriores y se puede personalizar un poco como el color, la leyenda, etc.

```
cbarplot(x, radius=0.5, nlabels=3, col="lightblue", border="skyblue4")
```

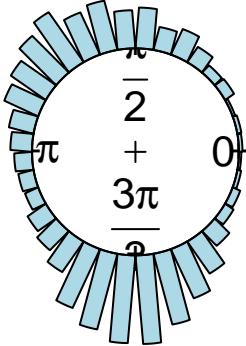
Circular Bar Plot



Y si `nlabels` se iguala a 0 visualiza los radianes en el círculo.

```
cbarplot(x, radius=0.5, nlabels=0, col="lightblue")
```

Circular Bar Plot



...

3.4.4. Paquete circlize

De nuevo se utiliza este paquete[18] de la misma forma que se utilizaba ggplot2, cambiando los ejes para representar columnas en vez de barras. Para los datos y capacidad de comparación de paquetes se eligen los mismos datos que en el primer ejemplo del paquete ggplot2, la cantidad de multas por cada velocidad. Lo primera función es opcional, ya que declara argumentos que son estéticos, como, en este caso, en que ángulo comienza el diagrama, o también, por ejemplo, se puede establecer donde está en centro del diagrama.

```
library(circlize)
circos.par("start.degree" = 90)
```

Lo que si que se necesita hacer es inicializar la plantilla del gráfico fijando en cuantos sectores se divide cada anillo del diagrama, en este caso solo uno para crear todas las columnas en el mismo sector. Y se fijan los límites del eje X con el número de columnas.

```
circos.initialize("a", xlim = c(25.5,50.5))
```

Ahora se establece el anillo fijando el límite del eje Y y la altura del anillo en porcentaje, en este caso un 80 % dejando un hueco del 20 %. Para este ejemplo se elimina el borde del anillo y la función para implementar las barras dentro del anillo.

```
circos.track(ylim = c(0,600), track.height = 0.8,
            bg.border = NA, panel.fun = function(x, y) {
```

La función `circos.rect` realiza retángulos igual que se usó en el diagrama de Barras Radiales, pero cambiando los ejes, poniendo esta vez en el eje X la cantidad de barras y en el eje Y los valores. Para implementarlas se pasan como argumentos `xInicial`, `yInicial`, `xFinal` y `yFinal`; con un color agradable y borde blanco para facil separación de las barras.

```
xlim = CELL_META$xlim
circos.rect( 26:50 - 0.45, rep(0, 25),
            26:50 + 0.45, data$velocidad.Freq ,col = "skyblue", border = "white")
```

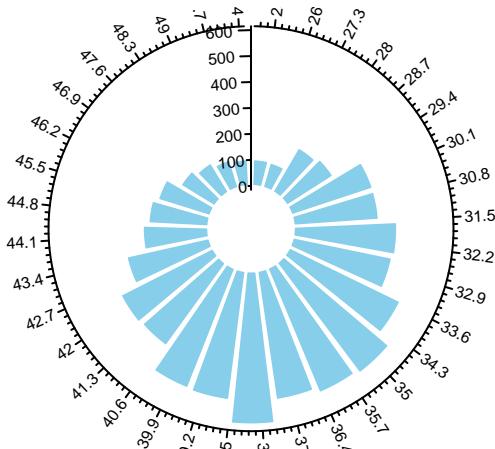
Se crean las escalas de los ejes para ver las medidas de los valores.

```
circos.axis(h="top",direction = "outside",
            labels.facing= "clockwise", labels.cex = 0.5)
circos.yaxis(labels.cex = 0.5)}
```

Y para que se pueda implementar otro ejemplo hay que limpiar las variables que están en circos.

```
circos.clear()
```

Resultando en:



...

3.4.5. Comparaciones

Para realizar este diagrama se han encontrado 4 paquetes; cada uno genera una aproximación diferente. Al igual que en Diagrama de Barras Radiales, el paquete `ggplot2` se usa mediante la transformación del sistema de coordenadas, esto proporciona más versatilidad debido a su capacidad de personalización. En la forma de representar los datos, los tres paquetes tienen diferentes maneras de crear el diagrama; `ggplot2` exemplifica las columnas mediante barras resultando más estético, `plotrix` representa las columnas con líneas no dejando que cada una de las líneas cambie el color ni ancho para representar importancia, y `cplots` utiliza el diagrama de histograma con formato circular, pero solo puede representar datos con algún tipo de distribución no pudiendo utilizar cualquier dato.

El paquete `circlize` es muy parecido a `ggplot2` ya que los dos son complejos y completos, por su gran capacidad de personalización, por el poder de retocar casi cualquier aspecto.

En conclusión, de los cuatro paquetes presentados los que resultan más útiles a la hora de representar el diagrama de columnas radiales son `ggplot2` y `circlize`, por su facilidad de uso cuando ya se conocen sus funciones, y la gran capacidad de personalización. Para diferenciar entre los dos es importante saber, que el paquete `ggplot2` junto al paquete de `ggiraph` añade la capacidad interactiva, y el hecho que sus ejemplos se pueden almacenar en una variable para poder ser usada en un futuro o modificada sin necesidad de escribir de nuevo el código.

3.5. Diagrama de Sectores

Para crear este tipo de diagramas, casi siempre se necesita de la misma estructura de datos, unas veces son simplemente unos valores representándolos como un porcentaje del total de la suma de todos los valores, o pueden estar acompañados de las categorías correspondientes. Para los ejemplos se elige la base de datos, antes usada, `mtcars`, con los datos de distintos vehículos, como su consumo, caballos, marchas, carburadores, etc.

3.5.1. R Básico

Una de las formas más sencillas para representar los diagramas de sectores es la función `pie`. Esta función necesita de los propios valores dichos antes y se pueden añadir más parámetros para su personalización, como las etiquetas, colores y bordes. En este caso se elige la cantidad de vehículos con cierta cantidad de carburadores.

```
Porcentaje <- table(mtcars$carb)
```

Para mejores colores y una paleta uniforme se opta por el paquete `RColorBrewer`.

```
library(RColorBrewer)
miPaleta <- brewer.pal(6, "Set2")
```

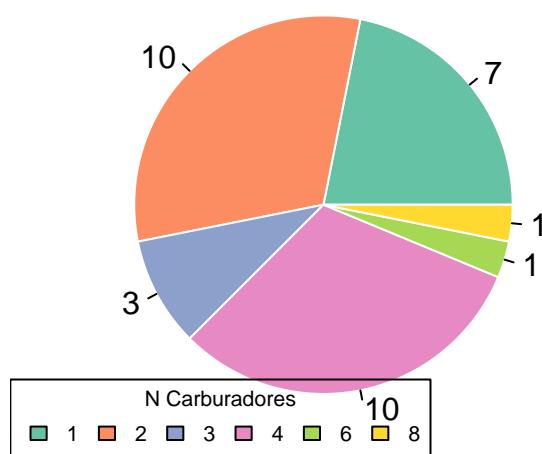
Para la función principal se usa: (dando como parámetros los datos, las etiquetas y la paleta de colores)

```
pie(Porcentaje, labels = Porcentaje, border = "white", col = miPaleta)
```

Y por último, se crea la leyenda con otra función de R básico y título.

```
legend("bottomleft", c("1", "2", "3", "4", "6", "8"), cex = 0.7,
       fill = miPaleta, title = "N Carburadores", horiz=TRUE)
title(main = "Coches por carburadores")
```

Coches por carburadores



3.5.2. Paquete ggplot2

Una vez más, se encuentra en el proyecto este paquete [17] . Como se ha comprobado antes este paquete es muy versátil para crear los diagramas ofreciendo muy buena personalización.

Para la realización de estos diagramas circulares la función `coord_polar` ha resultado ser muy útil y esta vez, al igual que en el diagrama de barras radiales, se va a transformar la Y de forma lineal a radial, pero la diferencia con aquel diagrama es que al dejar el eje X sin asignar cambia del diagrama de barras radiales al de sectores. Para ello se vuelven a preparar los datos necesarios.

```
library(ggplot2)
data <- data.frame(
  table(mtcars$carb)
)
```

El problema de este diagrama con respecto a otros que usan `ggplot2` es la posición de las etiquetas ya que solo con la función `geom_label` no sitúa las etiquetas en su correcto lugar, por lo que hay que preparar el texto para ello.

```
library(tidyverse)
data <- data %>%
  arrange(desc(Var1)) %>%
  mutate(Porcentaje = round(Freq / sum(Freq) *100, digits = 0)) %>%
  mutate(ypos = cumsum(Porcentaje) - 0.5*Porcentaje )
```

Y ahora se procede a la representación del gráfico, pasando como parámetros los datos a usar y asignando al eje X nada (como se dijo antes), al eje Y el porcentaje. Los colores se obtienen en función del grupo. Se usa la figura geométrica de la barra para representar los datos y se convierte el eje Y de lineal a radial.

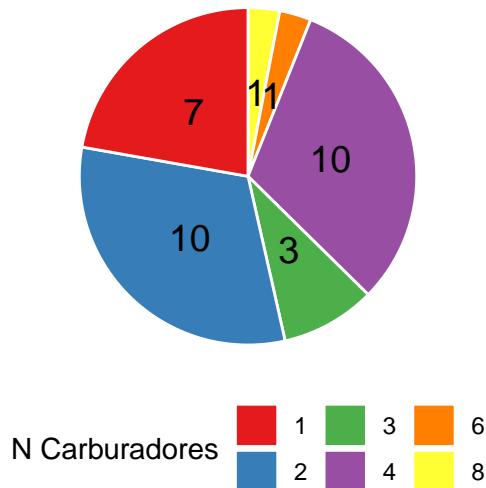
```
ggplot(data, aes(x="", y=Porcentaje, fill=Var1)) +
  geom_bar(stat="identity", width=1, color="white") +
  coord_polar("y", start=0) +
  theme_void() + labs(title="Coches por Carburadores") +
  theme(legend.position="bottom") + labs(fill = "N Carburadores") +
```

Se coloca el texto en la posición anteriormente calculada y se ajusta la paleta de colores con el paquete usado antes.

```
geom_text(aes(y = ypos, label = Freq), color = "black", size=5) +
  scale_fill_brewer(palette="Set1")
```

Y así resulta el gráfico.

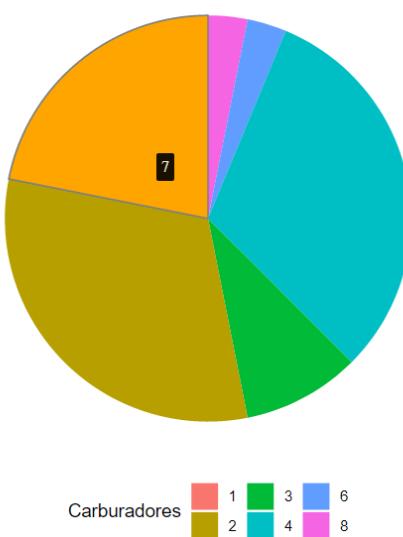
Coches por Carburadores



Al igual que muchos de los gráficos que usan ggplot2, se puede crear con el paquete ggiraph [20] los gráficos de ggplot2 interactivos, con la sustitución de la función que implemente los sectores por sus variante interactiva, en este caso `geom_col_interactive`.

```
library(ggiraph)
p <- ggplot(data, aes(x="", y=Freq, fill=Var1, tooltip = Freq, data_id = Var1)) +
  geom_col_interactive() +
  coord_polar("y", start=0) +
  theme_void() + labs(fill="Carburadores") + labs(title="Coches por Carburadores") +
  theme(legend.position = "bottom")
girafe(ggobj = p)
```

Coches por Carburadores



3.5.3. Paquete plotrix

Este paquete [21] tiene dos posibles formas de representar los diagramas de sectores, una en 2D y otra en 3D. Esto no lo tiene ningún otro paquete que se haya investigado en este proyecto.

Primero se verá la forma de representar el gráfico en 2D. Para ello se usa la función `floating.pie` y `pie.labels` para las etiquetas. Se obtienen los datos de los valores anteriores al igual que las etiquetas pero, para que el gráfico resulte algo distinto, también se va a usar el parámetro `explode`.

Para ello se necesita la librería y crear un espacio para luego realizar los sectores.

```
library(plotrix)
plot(0,xlim=c(1.5,5),ylim=c(1,5),type="n",axes=FALSE,xlab="",ylab="", main = "Coches por carburadores")
```

Luego para las etiquetas se calculan las posiciones y ángulos donde deben estar, proporcionados ya la por función `floating.pie`.

```
angulos <- floating.pie(3.25, 3, data$Freq, radius = 1.5,
                         col = c("#ff0000", "#80ff00", "#00ffff", "#44bbff", "#8000ff"))
```

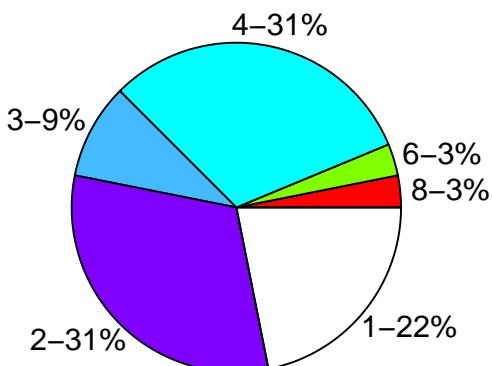
Y para las etiquetas `pie.labels` necesita dichos angulos, la posición del centro del diagrama y las etiquetas.

```
data$label <- paste0(data$Var1, "-", data$Porcentaje, "%")
pie.labels(3.25, 3, angulos, data$label, minangle = 0.2, radius = 1.6)
```

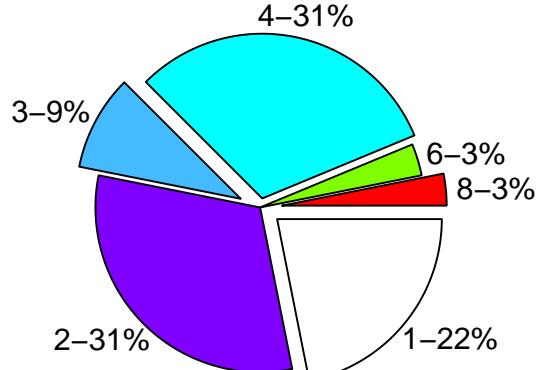
Después se realiza el mismo trabajo para el parámetro `explode`, pero incluyendo el vector con las distancias al centro (tanto en el gráfico como en las etiquetas).

```
plot(0,xlim=c(1.5,5),ylim=c(1,5),type="n",axes=FALSE,xlab="",ylab="", main = "Coches por carburadores")
angulos <- floating.pie(3.25, 3, data$Freq, radius = 1.5,
                         col = c("#ff0000", "#80ff00", "#00ffff", "#44bbff", "#8000ff"),
                         explode = c(0.2,0,0.1,0.2,0))
pie.labels(3.25, 3, angulos, data$label, minangle = 0.2, radius = 1.6,
           explode = c(0.2,0,0.1,0.2,0))
```

Coches por carburadores



Coches por carburadores



Y, por último, se realiza el mismo gráfico pero en 3D, gracias a la función `pie3D`. Esta forma requiere guardar el resultado en, por ejemplo, png.

Para ello se establece el nombre del archivo.

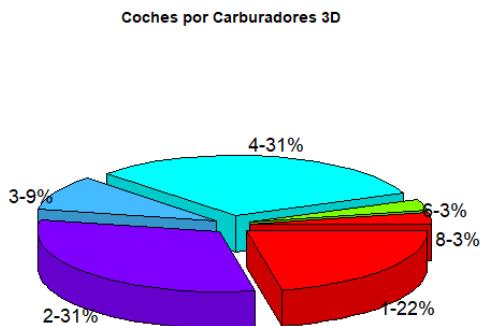
```
png(file = "3d_sectores.png")
```

Y crear el gráfico con los valores iguales que antes.

```
pie3D(data$Freq, labels = data$label, explode = 0.1, main = "Coches por Carburadores 3D",
       col=c("#ff0000", "#80ff00", "#00ffff", "#44bbff", "#8000ff"))
```

Y al final guardarlo, resultando en el siguiente gráfico.

```
dev.off()
```



...

3.5.4. Paquete rCharts

Tanto en este paquete como en el siguiente se van a traer unos diagramas de sectores interactivos. Al optar por el formato del papel y no página web en este trabajo, se procede al final de cada paquete a mostrar la forma de guardar los gráficos en html para su posterior visualización.

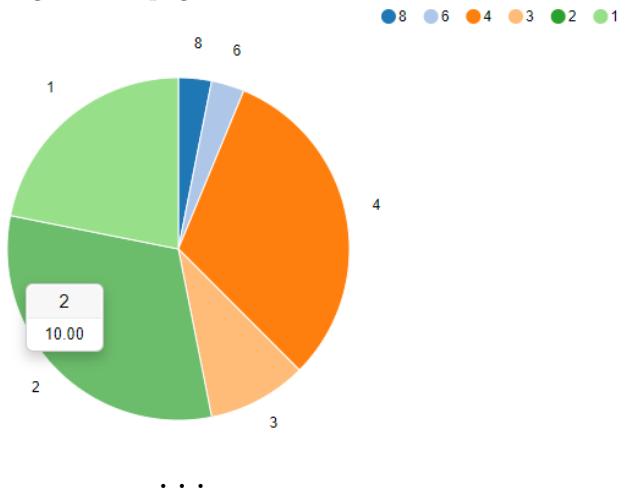
Este paquete [23] sirve para crear y personalizar gráficos interactivos en javascript transformandolos a partir de R. Para representar este gráfico se va a reutilizar los mismos valores que antes e implementarlos con la función `nPlot`.

```
library(rCharts)
p <- nPlot(Freq~Var1, data = data, type = 'pieChart')
```

El problema viene a la hora de mostrarlo ya que p es una script de javascripts, por lo que para mostrarse con interactividad se necesita incluir en una pagina html. Para ello, este gráfico solo necesita de la etiqueta `<html>` al principio y `</html>` al final, copiando el script del resultado de la siguiente orden.

```
p$print(include_assets=T)
```

Dando como resultado la siguiente imagen como página web.



3.5.5. Paquete plotly

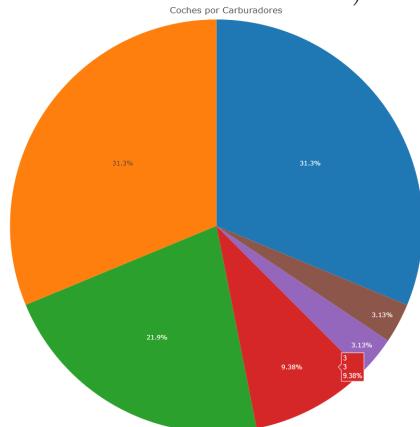
Este paquete [24] se dedica principalmente a crear gráficos interactivos, de los gráficos más comunes y otros más específicos que se verán mas adelante. Por ello no se visualiza el gráfico resultante sino una foto de él. Para realizarlo plotly ofrece la función `plot_ly`, básica para cualquier plot con este paquete, si se le da el tipo `pie` reproduce el diagrama deseado. Como hasta ahora se va a seguir utilizando los mismos datos.

```
library(plotly)
p <- plot_ly(data, labels = data$Var1, values = data$Freq, type = 'pie') %>%
  layout(title = 'Coches por Carburadores',
         xaxis = list(showgrid = FALSE, zeroline = FALSE, showticklabels = FALSE),
         yaxis = list(showgrid = FALSE, zeroline = FALSE, showticklabels = FALSE))
```

Aunque esta vez la forma de guardarla en una página web, es mucho mas sencilla gracias al paquete `htmlwidgets`, mediante la siguiente orden.

```
library(htmlwidgets)
saveWidget(p, file = "plotly_pie.html")
```

Resultando en el siguiente grafo (no interactivo salvo en el html).



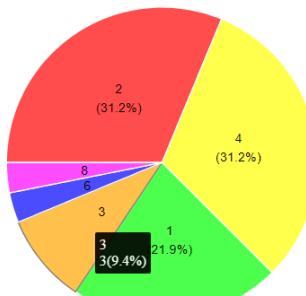
3.5.6. Paquete ggiraphExtra

Este paquete [25] es una extensión del paquete `ggiraph`, creando sus propias funciones al haber sustituido las variantes interactivas de las funciones de `ggplot2` con las funciones de `ggiraph`. Por lo que otra forma de crear un diagrama de sectores interactivo es mediante este paquete. Si se requiere el uso de funciones de `ggplot2`, el gráfico pierde interactividad y pasa a ser un paquete ayudante de `ggplot2`.

Para la comparación entre paquetes se opta por el uso de los datos ya usados anteriormente.

```
library(ggiraphExtra)
ggPie(data, aes( pies=Var1, count=Freq), interactive = TRUE, title = "Coches por Carburadores")
```

Coches por Carburadores  



3.5.7. Comparaciones

Este diagrama es uno de los dos que se pueden crear mediante el uso del lenguaje R sin ningún paquete adicional pero, aunque el gráfico resultante es correcto, no tiene ninguna forma de personalización. Con `ggplot2` ofrece el mismo buen resultado que todas las veces que se usa, ofreciendo un gráfico que cumple su misión además de ofrecer gran capacidad de personalización. En este diagrama el paquete `ggiraph` ofrece una ayuda al paquete `ggplot2` añadiendo el componente de interactividad.

Otro paquete que ofrece una función interesante dentro de este diagrama es `plotrix`, con el cual, aparte de crear un gráfico adecuado, con personalización (de difícil aprendizaje a la hora de crearlo comparado con los demás paquetes encontrados), se añade la función de crear el diagrama de sectores en 3D.

Los siguientes tres paquetes que ofrecen funciones para crear el diagrama de sectores, incluyendo siempre la variable de interactividad, que ayuda a la compresión de diagramas de sectores con una cantidad de datos mayor de los normal son: `Rcharts` que crea el diagrama con su función predeterminada, aumentando su facilidad de aprendizaje, especificando que el tipo de diagrama que se requiere es el de sectores. `Plotly` al igual que `rcharts` puede crear el diagrama solo con su función principal especificando su tipo, pero a la hora de personalización proporciona más variantes que el paquete anterior, añadiendo mas funciones al estilo que lo hace `ggplot2`. Con `ggiraphExtra` se vuelve al caso de `rcharts` de solo una función para representar el diagrama, pero carece de la capacidad de más personalización.

En conclusión, todos los paquetes encontrados para implementar el diagrama de sectores ofrecen un gráfico similar, salvo en la capacidad de personalización e interactividad, pero por su sencillez prevalece la función que ofrece el lenguaje R sin paquete. Aunque, si lo que se busca es más personalización, se recomienda el uso de `ggplot2` y, si se busca interactividad se puede escoger entre `ggplot2-ggiraph` y `plotly`.

3.6. Diagrama de Rose of Nightingale

Al recopilar las formas para realizar este diagrama se descubre que es la misma que si el diagrama constara de gráficos de columnas apiladas, pero dando el formato de unión entre barras que aporta el diagrama de sectores.

3.6.1. Paquete ggplot2

Otra vez en el proyecto se cruza este paquete [17] aportando su versatilidad. El gráfico consistirá de nuevo en crear un diagrama normal (lineal) y transformarlo con la ayuda de `coord polar`. Esta vez se realiza un gráfico de barras apiladas y transformar el eje X.

Primero necesita los datos para crear las barras apiladas. En este caso los datos son el porcentaje de muertes accidentales y suicidio, por ciudad y mundo rural, agrupados por meses.

```
data=read.csv("rosa.csv", sep = ";", header = TRUE, fileEncoding = 'UTF-8-BOM')
head(data)

##      mes Suici..Ciudad Suici..Rural Acc..Ciudad Acc..Rural
## 1 January      0.9      0.7     5.3     5.4
## 2 February     0.7      0.6     5.3     5.3
## 3 March        0.8      0.8     5.2     5.7
## 4 April         1.1      0.9     5.6     5.6
## 5 May           1.1      1.0     5.4     5.8
## 6 June          0.8      0.7     7.9     6.6
```

Se necesitan datos agrupados por solo una variable que en esta vez se usan las causas. Gracias a `tidyverse` se consigue el agrupamiento por causa.

```
library(tidyverse)
data <- data %>% gather(key = "observation", value="value", -c(1))
```

Y ahora, para crear el gráfico, lo único que hay que hacer es crear el diagrama de barras apiladas y aplicarle la transformación, además de la estética deseada.

Para ello se llama a la función `ggplot` con los datos de `data3` y asignando el eje X a los meses, el Y, a los valores y los colores en función del grupo. Y se crea el diagrama de barras tradicional con `geom_bar(stat="identity")`.

```
library(ggplot2)
ggplot(data,aes(x=mes,y=value,fill=observation))+ 
  geom_bar(stat="identity",width=1,colour="white",size=0.1)+
```

Ahora se transforma la coordenada X y, se establece el título y nombre de la leyenda.

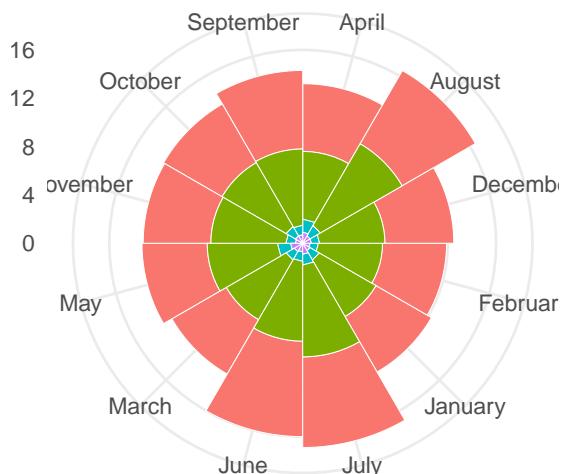
```
coord_polar('x')+
  labs(fill='Causas') + labs(title = "Procentaje de Muertes") +
```

Y por último la estética del gráfico deseada (sin ejes, fondo blanco, etc. y leyenda debajo).

```
xlab("")+ylab("") +
theme_minimal() +
theme(legend.position = "bottom",
  axis.line = element_blank(),
  axis.ticks = element_blank())
```

Dando como resultado el gráfico.

Procentaje de Muertes



Causas Acc..Ciudad Acc..Rural Suici..Ciudad Suici..Rural

...

Ahora, para ver el gráfico que le da nombre al diagrama, se van a obtener los datos de la librería `HistData` y se va a simular el grafo de Rose of Nightingale.

```
library(HistData)
data(Nightingale)
```

Como en este gráfico solo se muestra la fecha, las muertes y su causa, se va a tener que transformar estos datos para que se puedan mostrar al igual que el gráfico que se muestra en [2, pág 113]

Primero solo se necesita la fecha y las muertes por enfermedad, heridas y otras causas y ordenarlas por la causa de la muerte al igual que en el ejemplo anterior.

```
require(reshape)
Night<- Nightingale[,c(1,8:10)]
melted <- melt(Night, "Date")
names(melted) <- c("Fecha", "Causa", "Muertes")
melted$Causa <- sub("\\.rate", "", melted$Causa)
```

Luego, al igual que el gráfico en el libro, separa los datos para crear dos gráficos, con la separación a partir de la fecha marzo de 1855.

```
Night <- melted
Night$Month <- format(Night$Fecha, "%b %Y")
Night1 <- subset(Night, Fecha < as.Date("1855-04-01"))
Night2 <- subset(Night, Fecha >= as.Date("1855-04-01"))
```

Ahora se exponen los gráficos, uno al lado del otro con la misma fórmula que el ejemplo anterior con el eje X las fechas, el Y las muertes y el color en función de las causas.

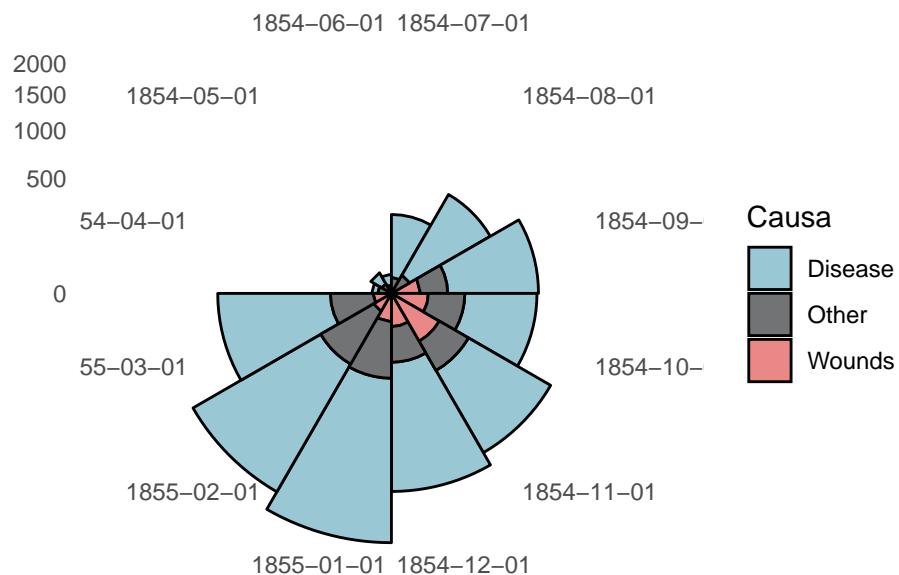
Para realizar los diagramas se emplea el siguiente código que no se explica porque en esencia es el ejemplo anterior.

```

ggplot(Night1, aes(x = factor(Fecha), y = Muertes, fill = Causa)) +
  geom_bar(width = 1, stat = "identity", color="black") +
  coord_polar(start = 3*pi/2) +
  xlab("") + ylab("") +
  scale_y_sqrt() + labs(title = "Muertes por Causa") +
  scale_fill_manual(values = c("#9ac7d4", "#727374", "#ea8888")) +
  theme_minimal() +
  theme(panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        axis.line = element_blank(),
        axis.ticks = element_blank())

```

Muertes por Causa

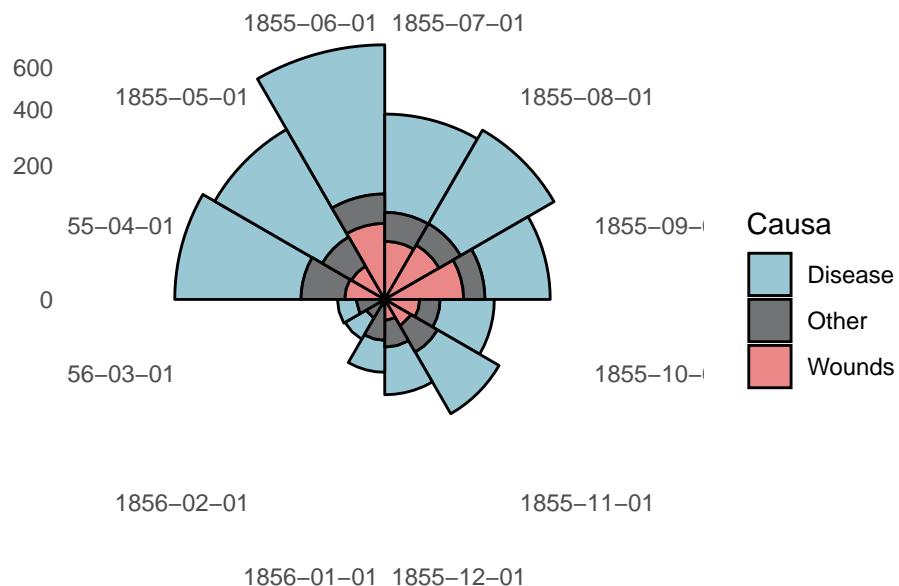


```

ggplot(Night2, aes(x = factor(Fecha), y = Muertes, fill = Causa)) +
  geom_bar(width = 1, stat = "identity", color="black") +
  coord_polar(start = 3*pi/2) +
  xlab("") + ylab("") +
  scale_y_sqrt() + labs(title = "Muertes por Causa") +
  scale_fill_manual(values = c("#9ac7d4", "#727374", "#ea8888")) +
  theme_minimal() +
  theme(panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        axis.line = element_blank(),
        axis.ticks = element_blank())

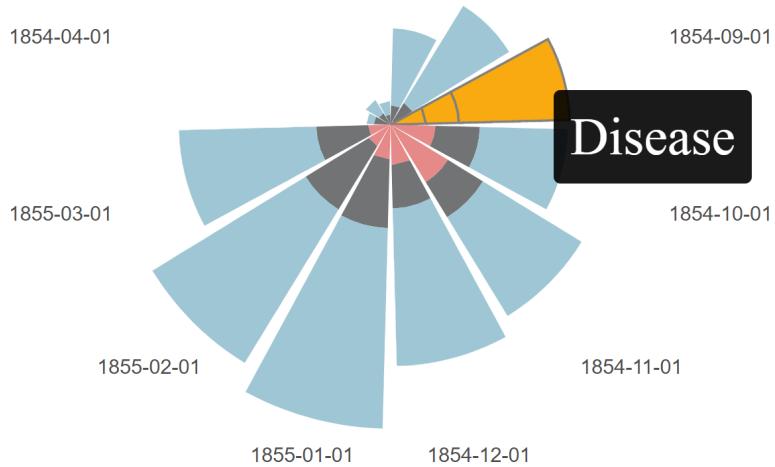
```

Muertes por Causa



Una forma de poder usar la interactividad con diagrama de ggplot2 es con ggiraph [20] y en este caso la función `geom_col_interactive` en vez de `geom_bar`.

```
library(ggiraph)
p <- ggplot(Night1, aes(x = factor(Fecha), y = Muertes, fill = Causa, tooltip=Causa,
                         data_id =factor(Fecha))) +
  geom_col_interactive() +
  coord_polar(start = 3*pi/2) +
  xlab("") + ylab("") +
  scale_y_sqrt() +
  scale_fill_manual(values = c("#9ac7d4", "#727374", "#ea8888")) +
  theme_minimal() +
  theme(legend.position = "none",
        panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        axis.line = element_blank(),
        axis.text.y = element_blank(),
        axis.ticks = element_blank())
girafe(ggobj = p)
```



3.6.2. Paquete ggiraphExtra

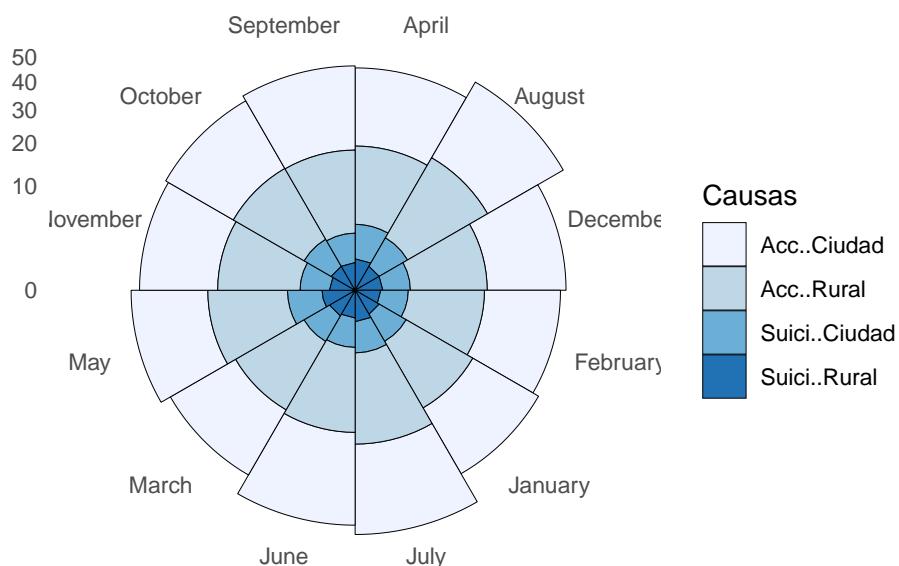
Este paquete [25] permite crear gráficos interactivos a partir de los gráficos de ggplot2. El propio paquete tiene un dataframe llamado Rose con datos para este diagrama, pero se usarán los mismos datos de porcentaje de muertes en ciudad y pueblos.

Una de las dos funciones que permite realizar esto es ggRose y necesita de los datos, la estética, la paleta de color, etc.

Primero se va a representar sin interactividad, para el formato de papel, pero, a continuación, se muestra la forma de pasarlo a html para la interactividad.

```
library(ggiraphExtra)
ggRose(data,aes(x=mes,fill=observation,y=value),stat="identity",interactive=FALSE
       , palette = "Blues") +
  xlab("") + ylab("") +
  labs(fill='Causas') + labs(title = "Procentaje de Muertes") +
  scale_y_sqrt() +
  theme_minimal() +
  theme(panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        axis.line = element_blank(),
        axis.ticks = element_blank())
```

Procentaje de Muertes

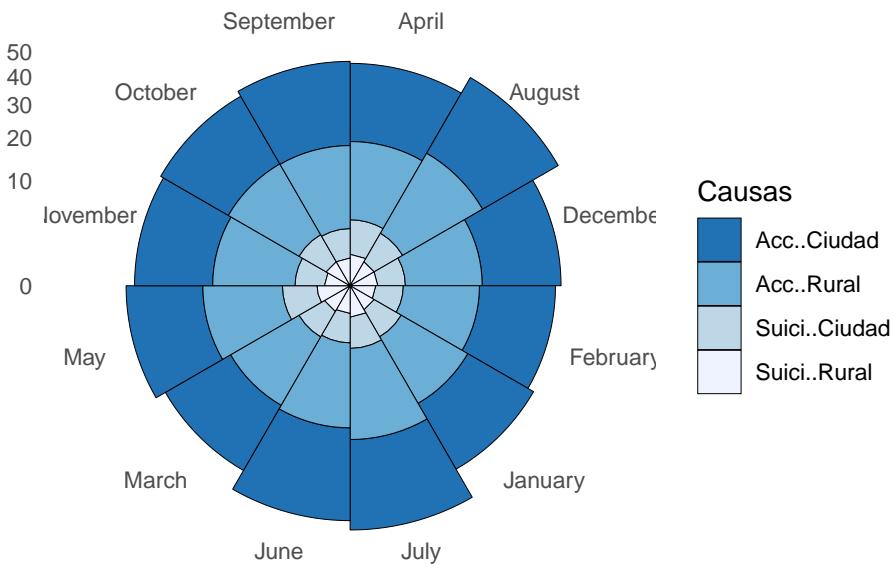


```
p <- ggRose(data,aes(x=mes,fill=observation,y=value),stat="identity",interactive=TRUE
            , palette = "Blues")
library(htmlwidgets)
saveWidget(p, file = "rose_interactive.html")
```

Otro método para realizar el mismo diagrama es `ggBar`. Como con el paquete `ggplot2` lo que se realiza es un diagrama de columnas apiladas tradicional transformando la coordenada X de lineal a radial. Como antes con la interactividad desactivada pero con la forma de verlo en html.

```
ggBar(data, aes(x=mes, fill=observation, y=value), stat="identity", polar=TRUE, palette="Blues"
      , width=1, color="black", size=0.1, reverse=TRUE, interactive=FALSE) +
  labs(fill='Causas') + labs(title = "Procentaje de Muertes") +
  scale_y_sqrt() +
  theme_minimal() +
  theme(panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        axis.line = element_blank(),
        axis.ticks = element_blank())
```

Procentaje de Muertes



```
p <- ggBar(data, aes(x=mes, fill=observation, y=value), stat="identity", polar=TRUE, palette="Blues"
            , width=1, color="black", size=0.1, reverse=TRUE, interactive=TRUE)
library(htmlwidgets)
saveWidget(p, file = "rose2_interactive.html")
```

...

3.6.3. Comparaciones

Para este diagrama se han encontrados dos paquetes, pero que en realidad es solo `ggplot2` ya que `ggiraphExtra` es un paquete que extiende la capacidad interactiva de las funciones de `ggplot2`. Los dos paquetes, al estar basados en el mismo, ofrecen el mismo diagrama diferenciándose con el factor de interactividad. Los dos parten de una mezcla de diagrama de barras apiladas y diagrama de sectores, por lo que cualquiera de los dos paquetes sería una buena opción para implementar el diagrama de Rose of Nightingale. Ya que si se le suma el paquete `ggiraph` a `ggplot2` se tendría el mismo diagrama interactivo que el que resulta del segundo paquete.

3.7. Diagrama de Cuadrícula Circular Ordenada

Para este diagrama, al consistir en crear una cuadrícula ordenada, es necesario buscar un paquete que permita crear ese tipo de diagramas y pueda incluir en cada celda algún tipo de gráfico, imágenes, texto, etc. El único paquete que permite hacer eso es `circlize`.

3.7.1. Paquete `circlize`

Se vuelve a ver este paquete [18] en el trabajo y, como se ha visto en otras ocasiones, dentro de este paquete existe una función que permite crear las celdas y, dentro de ellas implementar distintos gráficos, con cada una de las funciones que permiten crearlos.

Lo primero es crear dichas celdas con la función `circos.track`. Esta función crea las celdas divididas por anillos empezando de exterior a interior y, para dividir cada anillo en diferentes celdas (para este paquete se llaman sectores), se utiliza la variable `factors`. Para poder usar dicha función se necesita crear la plantilla.

Al iniciar la plantilla hay que pasar como argumentos la cantidad de sectores (con este paquete se diferencian por letras) y los límites del eje X de los sectores.

```
library(circlize)
factors = letters[1:8]
circos.initialize(factors, xlim = c(0, 5))
```

Para crear el primer anillo, en este ejemplo solo se usa este anillo como vacío para enseñar el estado vacío, se necesita saber los límites del eje Y. Se pueden personalizar varios aspectos de los sectores, como bordes de ellos, altura en porcentaje, el color de fondo, qué anillo se pretende realizar o actualizar, etc. Cada sector se representa mediante letras y los anillos mediante números.

```
circos.track(ylim = c(0, 1), bg.border = "black", track.height = 0.15)
circos.info(plot = TRUE)
```

Para el segundo ejemplo se usan los colores del fondo de las celdas (sectores).

```
color=c("black", "grey", "red", "blue", "green", "orange", "yellow", "pink")
circos.track(ylim = c(0, 1), bg.col = color, track.height = 0.1, bg.border = "white")
```

Para poder insertar los distintos tipos de gráficos que ofrece circlize dentro de cada celda, se deben elegir estos después de crear el anillo deseado y proporcionar como parámetro el sector donde se deseé dicho gráfico. En este anillo se exponen algunas posibles variantes de diagramas de líneas y puntos, en cada celda.

```
circos.track(ylim = c(0, 1), track.height = 0.2)
circos.lines(sort(runif(10)*5), runif(10), col = "orange", lwd=2, sector.index = "a")
circos.lines(sort(runif(10)*5), runif(10), col = rgb(0.1,0.5,0.8,0.3), lwd=2,
            sector.index = "b", type = "o")
circos.lines(sort(runif(10)*5), runif(10), col = rgb(0.7,0.5,0.5,0.3), lwd=3,
            sector.index = "c", type = "s")
circos.lines(sort(runif(10)*5), runif(10), col = "#4DAF4A", sector.index = "d",
            type = "s", area = TRUE)
circos.points(runif(10)*5, runif(10), sector.index = "e", pch = 16, col = "chocolate")
circos.points(runif(10)*5, runif(10), sector.index = "f", pch = 8, col = "steelblue")
circos.points(runif(10)*5, runif(10), sector.index = "g", pch = 1, col = "salmon")
circos.points(runif(10)*5, runif(10), sector.index = "h", pch = 20, col = "tomato")
```

Ahora también se puede usar la función que especifica qué ocurre dentro de la propia función track.

```
value = (sample(1:5, 5))
circos.track(ylim = c(0, 5), panel.fun = function(x, y) {
  xlim = c(0,5)
  circos.rect( 0:4,rep(0, 5), 1:5, value,
              col = "orchid", border = "white", sector.index = "a"))}
```

Para poder seguir añadiendo este tipo de gráficos de barras y columnas radiales, hace falta implementarlos, como en el ejemplo de los diagramas de puntos y líneas, fuera de la función track.

```
circos.rect(rep(0, 5), 0:4, value, 1:5,
            col = "plum", border = "black", sector.index = "b")
circos.rect( 0:4,rep(0, 5), 1:5, value,
            col = "maroon", border = "white", sector.index = "c")
circos.rect(rep(0, 5), 0:4, value, 1:5,
            col = "skyblue", border = "black", sector.index = "d")
circos.rect( 0:4, 5-value, 1:5, rep(5, 5),
            col = "steelblue", border = "white", sector.index = "e")
circos.rect(5-value, 0:4, rep(5, 5), 1:5,
            col = "sienna", border = "black", sector.index = "h")
```

Otra función interesante es la de poder insertar imágenes en una celda de la cuadrícula con ayuda de la librería `png`, para que R pueda leer los datos de la imagen y reproducirla dentro del sector seleccionado y, como parte opcional, también se puede indicar el anillo deseado (con números).

```
library(png)
image = system.file("extdata", "Rlogo.png", package = "circlize")
image = as.raster(readPNG(image))
circos.raster(image, CELL_META$xcenter, CELL_META$ycenter, width = "8mm",
              sector.index = "f", track.index = 4)
circos.raster(image, CELL_META$xcenter, CELL_META$ycenter, width = "8mm",
              sector.index = "g", track.index = 4)
```

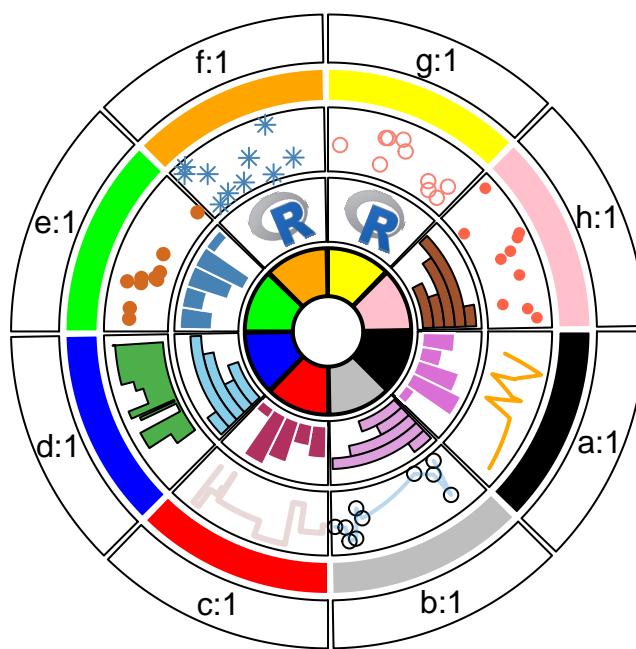
Y, por último, está la función `trackPlotRegion` de la que la función antes usada, `track`, es solo un atajo.

```
circos.trackPlotRegion(ylim = c(0, 1), bg.border = "black", , track.height = 0.15,
                      bg.col =color, bg.lwd = 2)
```

Y para que se pueda volver a crear otro diagrama con circlize siempre es necesario limpiar las variables de `circos`.

```
circos.clear()
```

Este conjunto de funciones da como resultado:



3.8. Diagrama Rayos de Sol

En este apartado se encuentra una de las formas de representar un diagrama de árbol con formato circular. Este diagrama se organiza en niveles donde, el nivel interior es la raíz del árbol y, a medida que se escala hacia el exterior del círculo, se encuentran los hijos hasta llegar a los nodos hoja.

Este diagrama, como es una de las formas de representar un árbol con formato circular, necesita una estructura de datos jerárquica para poder implementarse (raíz y hojas ó padres e hijos).

3.8.1. Paquete ggraph

Este paquete [26] surge de la necesidad de crear mejores gráficos de redes que los que ofrece ggplot2, por lo que, se creó este paquete como una ampliación de la API de ggplot2, para que ofrezca una mejor visualización de gráficos de capa por capa.

Para el ejemplo se va a necesitar unos datos jerárquicos, los cuales se obtienen de la base de datos flare siendo apoyada por la librería igraph [27] para transformarla en datos que entienda ggraph.

```
library(ggraph)
ramas <- flare$edges
head(ramas,5)

##                               from          to
## 1 flare.analytics.cluster flare.analytics.cluster.AgglomerativeCluster
## 2 flare.analytics.cluster   flare.analytics.cluster.CommunityStructure
## 3 flare.analytics.cluster   flare.analytics.cluster.HierarchicalCluster
## 4 flare.analytics.cluster       flare.analytics.cluster.MergeEdge
## 5 flare.analytics.graph    flare.analytics.graph.BetweennessCentrality

vertices <- flare$vertices
head(vertices,5)

##                                name size      shortName
## 1 flare.analytics.cluster.AgglomerativeCluster 3938 AgglomerativeCluster
## 2 flare.analytics.cluster.CommunityStructure 3812 CommunityStructure
## 3 flare.analytics.cluster.HierarchicalCluster 6714 HierarchicalCluster
## 4 flare.analytics.cluster.MergeEdge 743 MergeEdge
## 5 flare.analytics.graph.BetweennessCentrality 3534 BetweennessCentrality

library(igraph)
mygraph <- graph_from_data_frame( ramas, vertices=vertices)
```

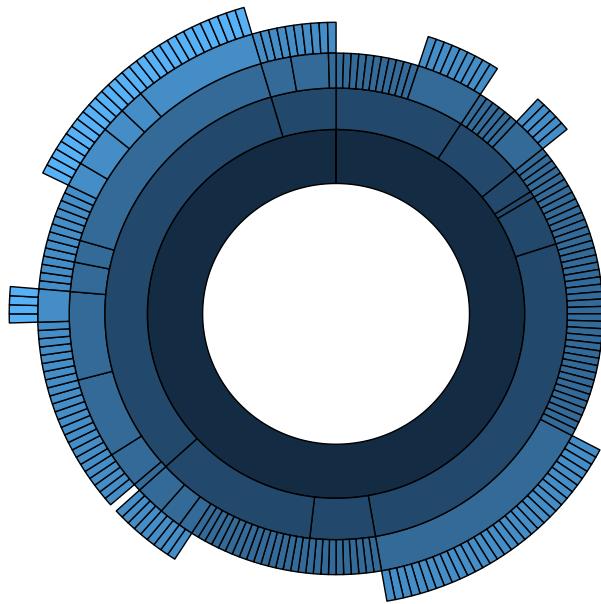
La base de datos, flare, consiste en unos datos con estructura jerárquica; de esta estructura se escogen las ramas, tabla con datos de como se relacionan los nodos (from-to), y los vértices, llamados también nodos con el nombre de cada nodo, el tamaño del nodo y el nombre de la ruta que de raíz hasta él (raíz.grupo1.subgrupo1.nodo).

Para preparar la plantilla del gráfico se necesitan los datos, el tipo particion y establecerlo como circular.

```
ggraph(mygraph, 'partition', circular = TRUE) +
```

La función elegida para crear el gráfico es `geom-node-arc-bar`, ya que establece los nodos del grafo como barras, las cuales se van a colorear en función de la profundidad. Y, para mejor estética, se quitan el fondo y la leyenda.

```
geom_node_arc_bar(aes(fill = depth), size = 0.25) +
theme_void() +
theme(legend.position="FALSE")
```



Como se ve en el resultado del gráfico, para que se puedan poner etiquetas y entenderse, se necesitara quitar algún nivel de profundidad. Y al final reconstruir el gráfico de nuevo con la función de igraph.

```

library(tidyverse)
ramas <- flare$edges %>%
  filter(to %in% from) %>%
  droplevels()
head(ramas,5)

##           from                      to
## 1 flare.analytics      flare.analytics.cluster
## 2 flare.analytics      flare.analytics.graph
## 3 flare.analytics flare.analytics.optimization
## 4      flare          flare.animate
## 5  flare.animate   flare.animate.interpolate

vertices <- flare$vertices %>%
  filter(name %in% c(ramas$from, ramas$to)) %>%
  droplevels()
vertices$size <- runif(nrow(vertices))
head(vertices,5)

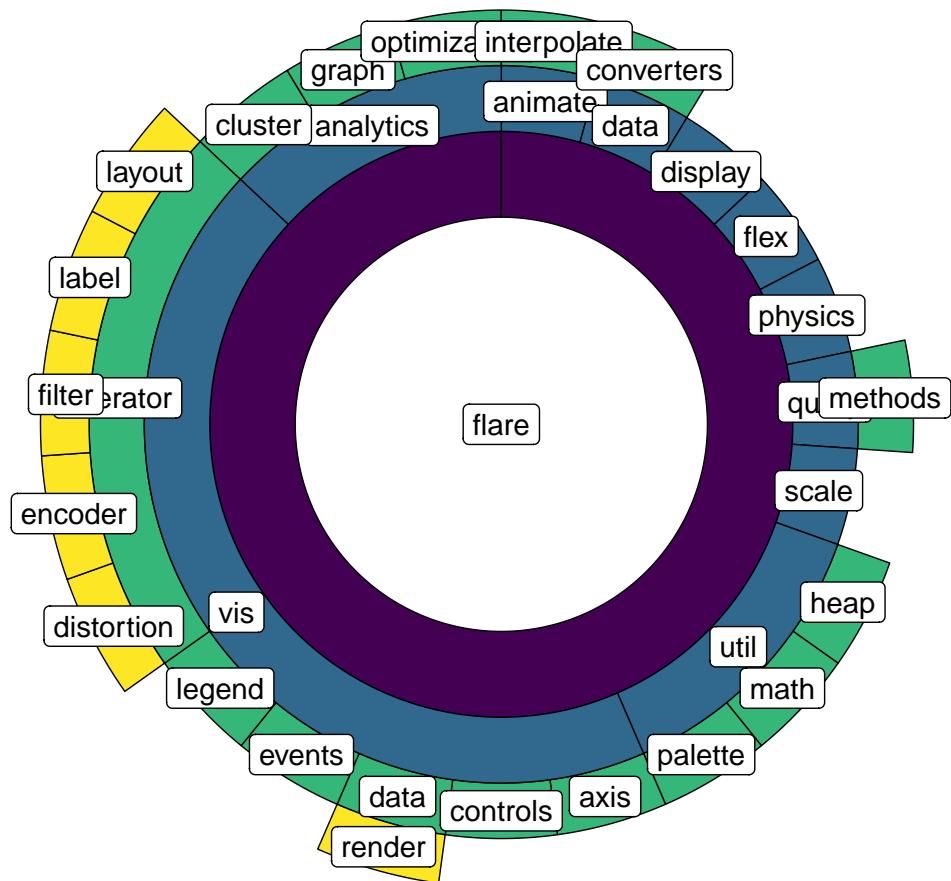
##                 name      size    shortName
## 1 flare.analytics.cluster 0.2653981    cluster
## 2      flare.analytics.graph 0.9943222      graph
## 3 flare.analytics.optimization 0.5703646 optimization
## 4          flare.animate 0.5288665     animate
## 5  flare.animate.interpolate 0.3277325 interpolate

mygraph <- graph_from_data_frame( ramas, vertices=vertices)

```

El nuevo gráfico sólo es distinto al anterior en 2 líneas. (las etiquetas y la paleta de colores)

```
geom_node_label( aes(label=shortName,) ) +  
scale_fill_viridis() +
```



3.8.2. Paquete plotly

Gracias a este paquete [24] y sus gráficos específicos, se puede realizar un gráfico parecido al del paquete anterior, pero con una interactividad muy bien implementada, ya que, cuando se pulsa en un nodo se recrea el gráfico como si el nodo seleccionado fuera la raíz, y así con los demás nodos. Para volver se pulsa en el nodo raíz y se vuelve al gráfico donde el nodo, antes raíz, pasa a hijo, y su padre a raíz; así hasta la raíz real del gráfico. Se escogen los sets de datos de un repositorio de github sobre los sabores de café, con esta estructura jerárquica.

```
github <- "https://raw.githubusercontent.com/plotly/datasets/"
archivo <- "master/coffee-flavors.csv"
arcom <- "718417069ead87650b90472464c7565dc8c2cb1c/sunburst-coffee-flavors-complete.csv"
url1 <- paste0(github, archivo)
url2 <- paste0(github, arcom)
d1 <- read.csv(url1)
d2 <- read.csv(url2)
head(d1, n=4L)

##           ids   labels parents
## 1      Enzymatic-Flower Flower
## 2      Enzymatic-Fruity Fruity
## 3      Enzymatic-Herby Herby
## 4      Sugar Browning-Nutty Nutty

head(d2, n=4L)

##           ids   labels parents
## 1      Aromas Aromas
## 2      Tastes Tastes
## 3 Aromas-Enzymatic Enzymatic Aromas
## 4 Aromas-Sugar Browning Sugar Browning Aromas

library(plotly)
```

Al representarlos d2 expresa el gráfico completo y, al no poder verse los nodos más profundos, d1 ayuda a representar dichos nodos.

Mediante su función principal y el tipo sunburst se añaden los trazos de los dos gráficos.

Como se puede observar, para añadir un trazo de este tipo, se necesitan establecer las ids y los padres junto a la máxima profundidad para que se vea mejor.

```
p <- plot_ly() %>%
  add_trace(
    ids = d1$ids,
    labels = d1$labels,
    parents = d1$parents,
    type = 'sunburst',
    maxdepth = 2,
    domain = list(column = 0)
  ) %>%
```

Para juntar los dos gráficos se termina el completo (d2) con profundidad 3, para que los nodos hojas de éste sean los nodos raíz del primer gráfico (d1).

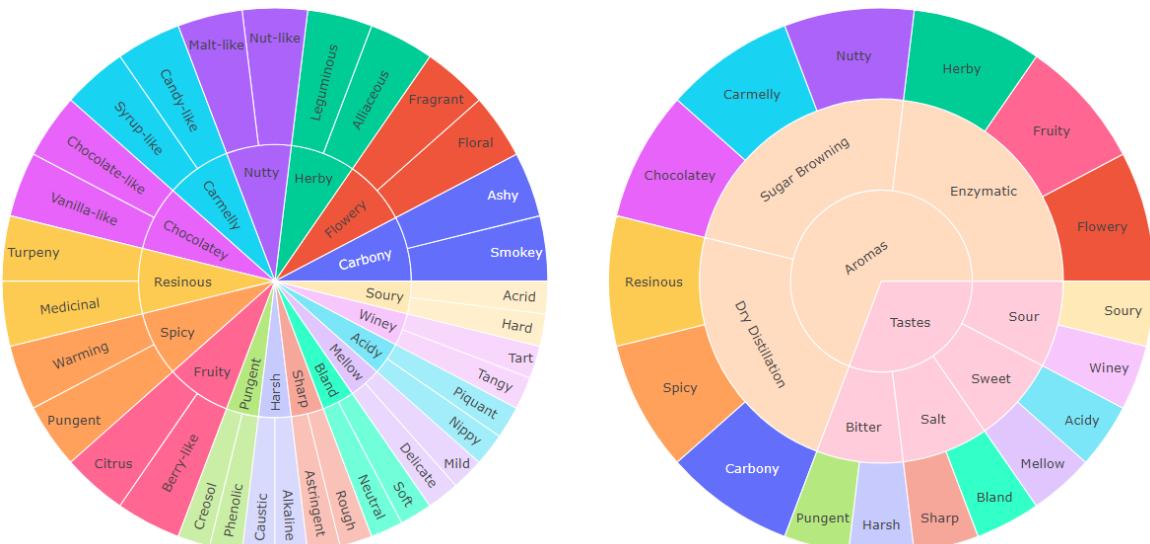
```
add_trace(
  ids = d2$ids,
  labels = d2$labels,
  parents = d2$parents,
  type = 'sunburst',
  maxdepth = 3,
  domain = list(column = 1)
) %>%
```

Para que se visualizan como conectados mediante los colores se usa `extendsunburstcolors` y se sitúa uno al lado del otro con `grid`.

```
layout(
  grid = list(columns = 2, rows = 1),
  margin = list(l = 0, r = 0, b = 0, t = 0),
  sunburstcolorway = c(
    "#636efa", "#EF553B", "#00cc96", "#ab63fa", "#19d3f3",
    "#e763fa", "#FECB52", "#FFA15A", "#FF6692", "#B6E880"
  ),
  extendsunburstcolors = TRUE)
```

Y para que se pueda usar su interactividad la librería `htmlwidgets` transforma el gráfico en html.

```
library(htmlwidgets)
saveWidget(p, file = "rayos_interactive.html")
```



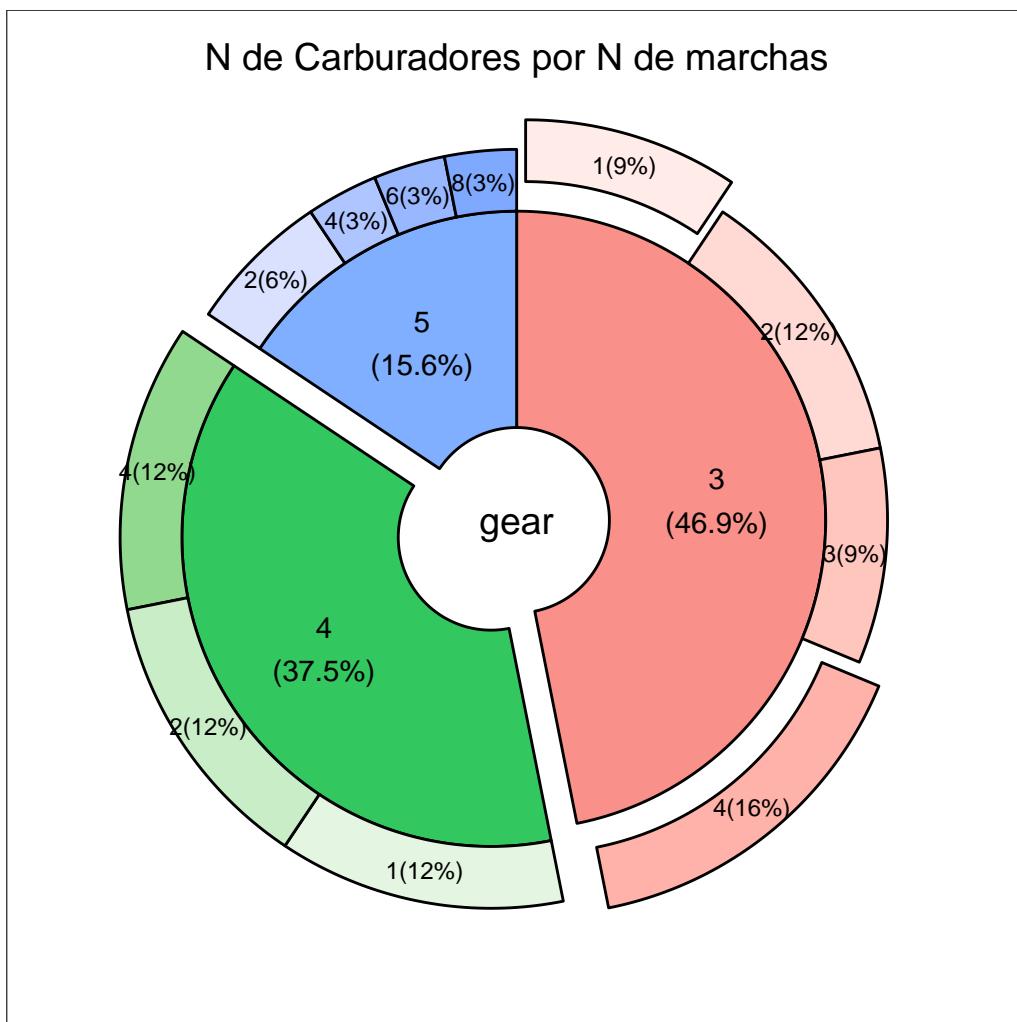
3.8.3. Paquete webr

Este paquete [28] consiste en una serie de funciones que implementan las del libro "Web-based Analysis without R in Your Computer". Dentro del paquete se encuentra la función que se va a usar: `pieDonut`, que consiste en un digrama de sectores unido a un diagrama de donut. Para poder implementarlo los datos se suelen poner como un data frame con los padres y los hijos.

```
library(dplyr)
df=mtcars %>% group_by(gear,carb) %>% summarize(n=n())
```

Usando `pieDonut` se puede elegir que datos seran los padres (pies) y cuales los hijos (donut).Para resultar en:

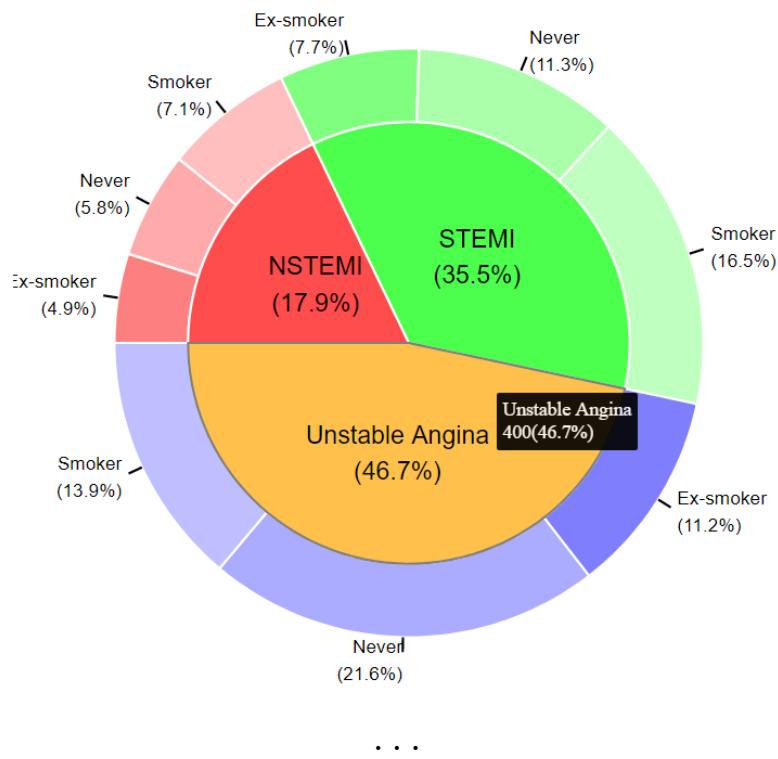
```
library(webr)
library(ggplot2)
PieDonut(df,aes(pies=gear,donuts=carb,count=n),ratioByGroup=FALSE, color = 'black',
          explode = 2, selected = c(1,4), explodeDonut = TRUE, labelposition=0,
          title="N de Carburadores por N de marchas")
```



3.8.4. Paquete ggiraphExtra

Otra forma de crear estos pieDonuts es con este paquete [25] , pero en este caso pueden ser interactivos. Este paquete es una extensión del paquete ggplot2 [17] usando las funciones sustitutivas del paquete ggiraph [20] . Se usan datos de las bases de datos moonbook. Con pacientes con STEMI (Síndrome coronario agudo con elevación del segmento ST), NSTEMI (Síndrome coronario agudo sin elevación del segmento ST), Unstable Angina (Angina inestable); relacionándolos con los hábitos de fumar de dichos pacientes.

```
library(ggiraphExtra)
library(ggplot2)
library(moonBook)
ggPieDonut(acs,aes(pies=Dx,donuts=smoking), interactive = TRUE)
```



3.8.5. Paquete sunburstR

Este paquete [29] nació en el proyecto para crear widgets interactivos con integración con htmlwidget [30] y solo está destinado a crear este tipo de gráfico. Para tener un ejemplo de este gráfico se necesita, primero unos datos con estructura jerárquica (árbol) y para ello se usa una base de datos con la población mundial para representar con porcentajes los continentes y países dependiendo de su población.

```
github ="https://raw.githubusercontent.com/holtzy/data_to_viz/master"
dataset = "/Example_dataset/11_SevCatOneNumNestedOneObsPerGroup.csv"
url = paste0(github, dataset)
data <- read.table(url, header=T, sep=";")
data[ which(data$value==1), "value"] <- 1
colnames(data) <- c("Continent", "Region", "Country", "Pop")
head(data, 1)

##   Continent      Region      Country      Pop
## 1       Asia Southern Asia Afghanistan 25500100
```

Estos datos necesitan un paso intermedio de cambio de formato con la ayuda de tidyverse [31] , para que la función principal los pueda usar. Creando la ruta de cada país (continente-región-país).

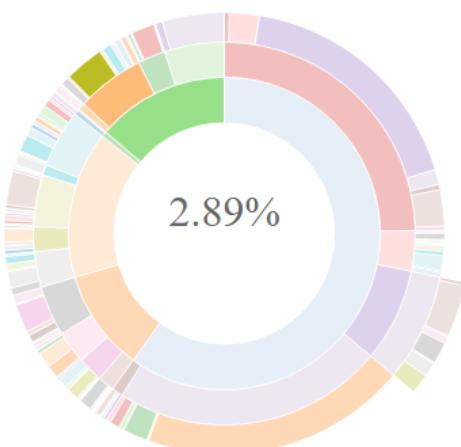
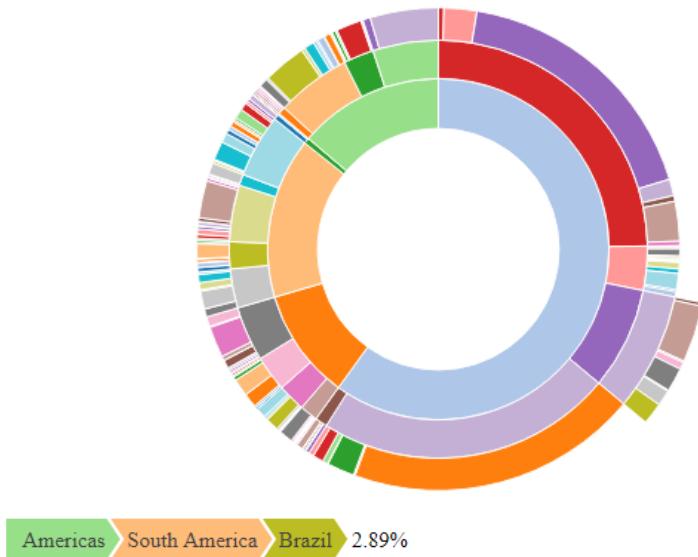
```
library(tidyverse)
data <- data %>%
  filter(Continent != "") %>%
  mutate(path = paste(Continent, Region, Country, sep="-")) %>%
  dplyr::select(path, Pop)
head(data, 1)

##                                     path      Pop
## 1 Asia-Southern Asia-Afghanistan 25500100
```

Y para poder representarlo con la interactividad, sin tener que ejecutarlo, cada vez se requiere de htmlwidget para crear el .html.

```
library(sunburstR)
p <- sunburst(data, legend = FALSE)

library(htmlwidgets)
saveWidget(p, file = "sunburts_interactive.html")
```



3.8.6. Comparaciones

El primero de los paquetes usados, **ggraph**, al derivarse de **ggplot2** ofrece su gran capacidad de personalización, al ser compatible con todas las funciones que son compatibles con **ggplot2**. Este paquete se comporta igual que **ggplot2** usando la función de plantilla y luego la función que implementa, seguido de las modificadoras. El problema con este paquete es la obtención y manipulación de sus datos, que resulta más difícil que con los demás. En el segundo paquete, **plotly**, se encuentra una de las mejores implementaciones de interactividad para este diagrama, aparte de presentar su gran personalización. Esta interactividad consiste en que, cada vez que se pulsa sobre un nodo del árbol, se crea un nuevo diagrama de rayos de sol con este nodo actuando como raíz. Pero se plantea el problema de su dificultad de uso, ya que no es sencillo de aprender a manejar. El tercer y cuarto paquete, **webr** y **ggiraphExtra**, crean el mismo gráfico de solo dos posibles niveles, brindando el primero más opciones desde su función principal. Pero, con el segundo, al igual que con **ggraph**, el paquete y funciones de **ggplot2** son compatibles. El último de los paquetes encontrados, **sunburstR**, está diseñado específicamente para crear este tipo de diagrama, por lo que todas sus funciones van dedicadas a este propósito, ofreciendo un diagrama complejo y fácil de aprender, con la ayuda de la interactividad.

En conclusión, todos los paquetes, menos **webr** y **ggiraphExtra**, implementan una diagrama de rayos de sol correcto, algunos con interactividad, otros con más personalización, pero en definitiva cualquiera de los paquetes puede ser usado para un buen gráfico de rayos de sol.

3.9. Diagrama Radar

En este diagrama, al usarse para realizar comparaciones cuantitativas entre diferentes variables, se requiere de una estructura de datos donde se proporcionan, el nombre de la variable y su valor numérico. Para realizar estos diagramas se han encontrado cuatro paquetes, algunos ofrecen interactividad, otros varios polígonos si las variables son las mismas y otros las dos características juntas.

3.9.1. Paquete fmsb

El nombre de este paquete [32] viene de Functions for Medical Statistics Book with some Demographic Data. Las funciones que tiene el paquete son implementaciones del libro Practices of Medical and Health Data Analysis using R. Y, para recrear este diagrama, el paquete cuenta con la función `radarchart`. Para realizarlo se necesitan los datos adecuados, basados en los nombres de las entidades y sus valores, por lo que se va a optar por representar uno de los gráficos del libro de Manuel Lima [2, pág 151], el ratio de muertes por accidentes por mes.

```
values <- c(5.3, 5.3, 5.2, 5.6, 6.4, 7.9, 7.5, 7.0, 6.5, 5.6, 5.6, 5.7)
names <- c("Ene", "Feb", "Mar", "Abr", "May", "Jun", "Jul", "Ago", "Sep",
         "Oct", "Nov", "Dec")
data <- as.data.frame(matrix(values, ncol = 12))
colnames(data) <- names
```

Con este paquete se usa la función `radarchart` para crear el diagrama radar; como esta función necesita unos valores máximos y unos mínimos, y el rango del gráfico es de 0 a 10, esos serán los máximos y mínimos.

```
data <- rbind(rep(10,12), rep(0,12), data)
```

Ahora se procede a usar la función para implementar el gráfico, pasando por parámetros: los datos, el tipo de eje...

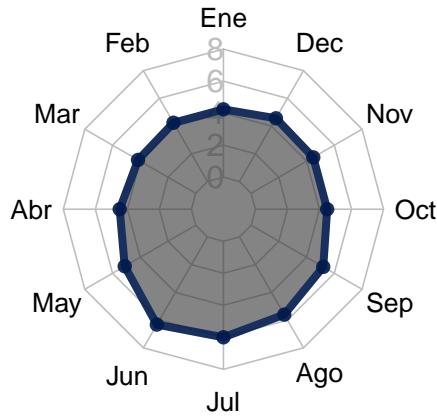
```
library(fmsb)
radarchart( data , axistype=1 ,
```

Y luego se eligen los colores de los puntos, el área, la cuadrícula y las etiquetas.

```
    pcol=rgb(0,0.1,0.3,0.9) , pfcol=rgb(0.2,0.2,0.2,0.6) , plwd=4 ,
    cglcol="grey" , cglty=1, axislabcol="grey" , caxislabels=seq(0,10,2)
    , cglwd=0.8, vlcex=0.8, title = "Muertes por accidentes en ciudades" )
```

Resultando en el gráfico.

Muertes por accidentes en ciudades



...

3.9.2. Paquete plotly

Se vuelve a usar este paquete [24] por lo que se sabe que usa la función `plot_ly` como principal para crear el diagrama requerido, solo le hacen falta los datos y el tipo de gráfico que se desea generar. En esta ocasión, el paquete es capaz de representar dos o más polígonos dentro de un mismo gráfico, por lo que se procede a la creación de los nuevos datos del gráfico de [2, pág 151].

```
values_rural <- c(5.4, 5.3, 5.7, 5.6, 5.8, 6.6, 7.6, 7.8, 6.3, 6.1, 6.2, 5.4)
```

Primero se necesita la librería y escoger el tipo de gráfico deseado para implementar el ejemplo.

```
library(plotly)
p <- plot_ly(type = "scatterpolar", fill = "toself") %>%
```

Y con `add-trace` se añade el trazo con los valores usados antes y sus etiquetas.

```
add_trace(r = values,
          theta = names,
          name = "Ciudades") %>%
```

Con este paquete, al poder visualizar más de un diagrama radar para compararlos en el mismo gráfico, se procede a usar de nuevo `add-trace`. Para la comparación se han usado los valores de la parte rural del gráfico que se está implementando.

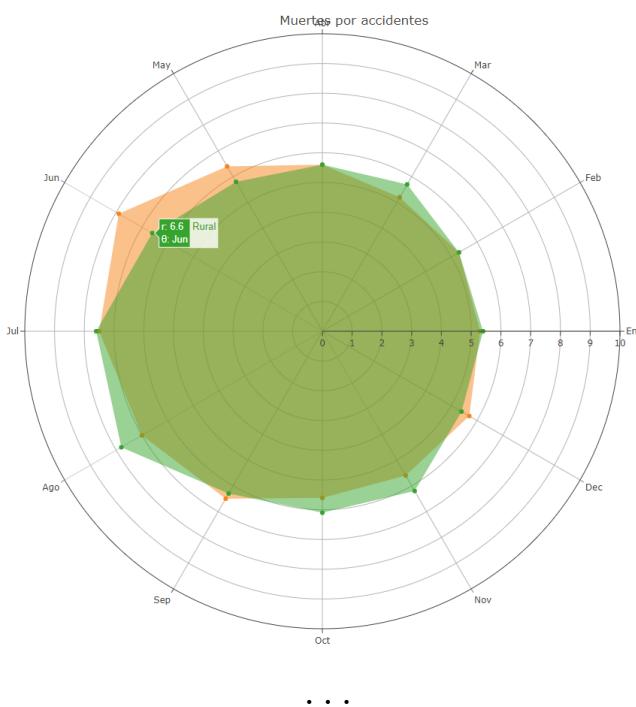
```
add_trace(r = values_rural,
          theta = names,
          name = "Rural") %>%
```

Y, para terminar, se le añade el diseño apropiado, como el rango de los valores de las variables y el hecho de que su sistema de coordenadas sean polares y no cartesianas.

```
layout( title= "Muertes por accidentes",
  polar = list(
    radialaxis = list(
      visible = T,
      range = c(0, 10))))
```

Debido a que es un diagrama interactivo, se usa `htmlwidgets` [30] para usar dicha interactividad, por lo que aquí solo se muestra una foto, no el gráfico interactivo.

```
library(htmlwidgets)
saveWidget(p, file="radar_interactive.html")
```



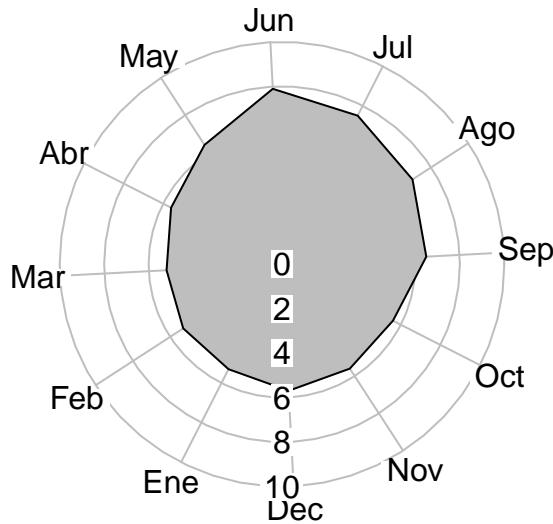
3.9.3. Paquete plotrix

Con este paquete [21] también es muy fácil crear estos diagramas radar gracias a la función `radial.plot`. Aparte de poder realizar el diagrama al igual que el anterior, se pueden comparar en un mismo gráfico dos radares y esta vez con el poder de que el gráfico empiece donde se requiera y en el sentido deseado. Primero se realiza el mas básico de los radares con los valores anteriores.

Para ello se pasan los datos y se establecen los parámetros, pero, lo más importante, es que se establece el tipo del grafo a `rp.type="p"` lo que indica que se va a crear un polígono.

```
library(plotrix)
radial.plot(values,labels=names, start = 2.7*pi/2, clockwise = TRUE, rp.type="p",
            main="Muertes por accidentes en ciudades", radial.lim=c(0,10),
            poly.col="grey", show.grid.labels=1)
```

Muertes por accidentes en ciudades



Ahora, lo único que se necesita, es una matriz con los datos que se quieren visualizar.

```
data_plotrix <- matrix(rbind(values, values_rural), nrow = 2, ncol = 12)
```

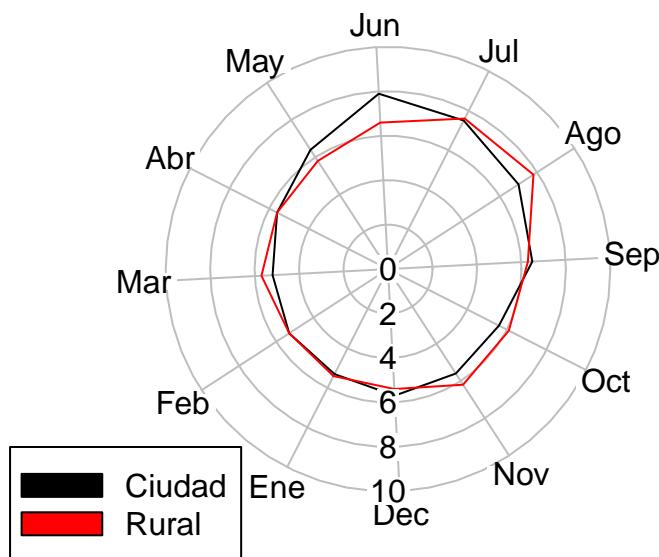
Se vuelve a realizar la misma función solo que sin rellenar el polígono para mejor estética.

```
radial.plot(data_plotrix, labels=names, start = 2.7*pi/2, clockwise = TRUE, rp.type="p",
            main="Muertes por accidentes", radial.lim=c(0,10), show.grid.labels=1)
```

Y para mejor visualización se usa el método `legendg` del propio paquete para la leyenda.

```
legendg(-17,-8,c("Ciudad", "Rural"),fill=list(1,2))
```

Muertes por accidentes



3.9.4. Paquete ggiraphExtra

Se vuelve a ver este paquete [25] para dar interactividad a partir de funciones de los paquetes con ggplot2 y ggiraph. Con este paquete se usa la función `ggRadar`, la cual se ayuda de ggplot2 para mejorar su aspecto. Con las dos librerías cargadas.

```
library(ggiraphExtra)
library(ggplot2)
```

Ahora con ggradar y los dos grupos de datos podemos recrear el diagrama y comparar la ciudad con lo rural.

```
p <- ggRadar(data_gg, aes(group = Muerte),
               rescale = FALSE, legend.position = "none",
               size = 1, interactive = FALSE, use.label = TRUE) +
```

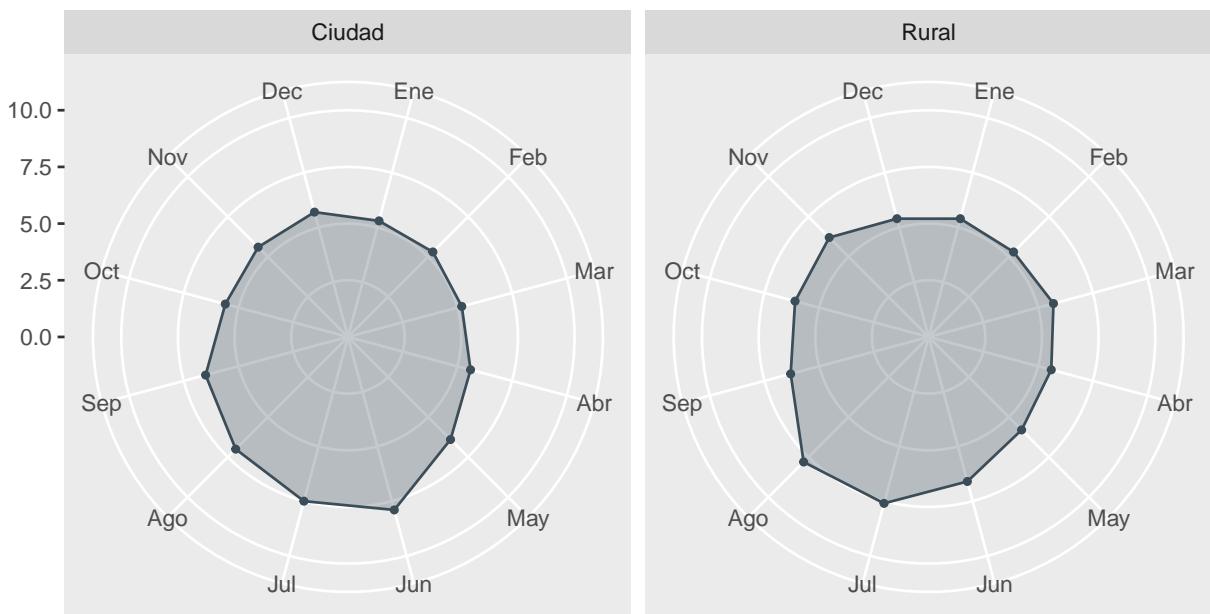
Con este método podemos comparar en dos gráficos separados. Si no lo estuvieran serían como los anteriores, los dos conjuntos de datos en el mismo gráfico.

```
facet_wrap(~Muerte) +
```

Y por último el rango, el color y el título.

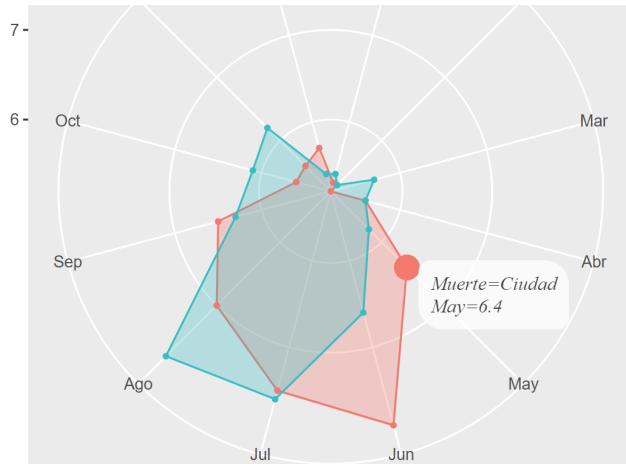
```
ylim(0,10) +
scale_fill_manual(values = rep(micolor, nrow(data_gg))) +
scale_color_manual(values = rep(micolor, nrow(data_gg))) +
ggtitle("Muertes por Accidente")
print(p)
```

Muertes por Accidente



Para dar la parte de interactividad hay que quitar la parte de ggplot2 y poder poner `interactive=TRUE`.

```
ggRadar(plot.df, aes(group = Muerte),
         rescale = FALSE, legend.position = "none",
         size = 1, interactive = TRUE, use.label = TRUE)
```



...

3.9.5. Comparaciones

Para implementar este gráfico se han conseguido 4 paquetes que ofrecen un diagrama muy similar. Una diferencia es que el paquete `fsmb` no está diseñado para la versión actual del lenguaje R, por lo que no se garantiza que no hubiera problemas de compatibilidad. Todos los paquetes ofrecen la misma capacidad de personalización, pero en cuanto a la facilidad de uso, todos excepto `plotly`, son iguales al usar solo una función para crear el gráfico. Al añadir la interactividad con el paquete `ggiraphExtra` el gráfico pierde la potencia de personalización que ofrece la compatibilidad con `ggplot2`.

3.10. Diagrama de Columnas Con Formato Iris

Este diagrama es muy similar al de las Columnas Radiales. Como se desea imitar el iris de un ojo, es necesario que los paquetes sean capaces de controlar el espacio que se establece en medio del diagrama. Se reducen los paquetes del diagrama de columnas a cuatro, pero en realidad son dos paquetes diferentes y los otros dos son ampliaciones de uno de ellos.

Para ejemplificar este diagrama se va a optar por un diagrama de columnas apiladas ya que en el Diagrama de Columnas radiales ya se ejemplificó con diagramas de columnas con solo un valor.

3.10.1. Paquete `ggplot2`

Este paquete [17] vuelve a salir con las mismas funciones que se expusieron en el Diagrama de Columnas Radiales Subsección 3.4 con `geom-bar` y para poder personalizar el espacio que se deja en medio, se utiliza la función `ylim` que delimita qué valores representa el valor del eje Y del gráfico, estableciendo el límite inferior por debajo de los valores que se representen en dicho eje.

Para empezar se cargan las librerías básicas para crear el diagrama: `ggplot2` para implementar y `dplyr` para modificar los datos.

```
library(dplyr)
library(ggplot2)
```

Para datos del diagrama de columnas con formato iris, y el formato de columnas apiladas se eligen los fallecidos del COVID-19 por fecha y por Comunidad Autónoma, actualizados a 20 de mayo por el Centro Nacional de Epidemiología (<https://cnecovid.isciii.es/covid19/>).

```
data <- read.csv("agregados.csv", header = TRUE)
```

Ahora se va a modificar el data frame agrupando los valores por CCAA y, realizando la modificación de pasar de datos acumulados de fallecidos a datos día a día.

```
data <- data %>%
  group_by(ComunidadAutonoma) %>%
  mutate(newcases = Fallecidos - lag(Fallecidos))
```

Otra modificación a tener en cuenta es el formato de la fecha, para ser leída por R.

```
data$date <- strptime(as.character(data$FECHA), "%d/%m/%Y")
data$date <- format(data$date, "%Y-%m-%d")
data$date <- as.Date(data$date)
```

Debido a la gran cantidad de CCAA no se ha encontrado una paleta que soporte, con colores distintivos, esta cantidad. Por ello se procede a la creación de una propia gracias a `RColorBrewer`. Resultando en un vector de colores distintos.

```
library(RColorBrewer)
n <- 60
qual_col_pals = brewer.pal.info[brewer.pal.info$category == 'qual',]
col_vector = unlist(mapply(brewer.pal, qual_col_pals$maxcolors, rownames(qual_col_pals)))
```

Para proceder a implementar el diagrama se dan a la plantilla los datos y qué valor se representa en qué eje, coloreando según la CCAA.

```
ggplot(data, aes(x=date, y= newcases, fill = CCAA)) +
```

Se utiliza la función `geom-bar` para crear las barras apiladas, fijando cada barra a los valores y no a la suma de ellos con `identity` y `position stack`.

```
geom_bar(stat = "identity", position = "stack") +
```

Se transforma el sistema de coordenadas del eje X a sistema polares con `coord-polar` que, por defecto, transforma el eje X.

```
coord_polar(start = pi/2) +
```

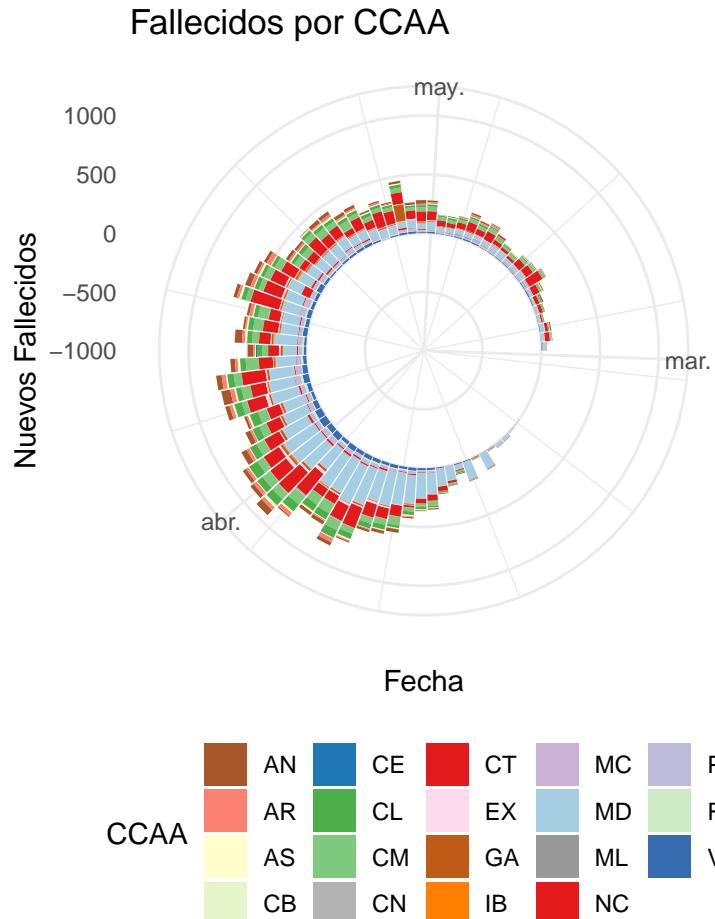
Ahora se usa la función principal para establecer la estética del formato de iris de un ojo. Se fija el límite inferior en un valor lo suficientemente bajo para ajustarse a la estética, y el límite superior.

```
ylim(-1000,1000) +
```

Las últimas funciones ayudan a cambiar el tema y colores del gráfico, ofreciendo un tema mínimo sin fondo y con colores agradables a la vista. También se procede a quitar la leyenda y texto de los ejes porque en este ejemplo no son necesarios.

```
scale_fill_manual(values = sample(col_vector, 19)) +  
theme_minimal() + labs(title = "Fallecidos por CCAA") +  
scale_x_date(minor_breaks = "1 week") +  
xlab("Fecha") + ylab("Nuevos Fallecidos") +  
theme(legend.position = 'bottom')  
NA
```

Dando el siguiente resultado:



Debido a que los dos siguientes paquetes no pueden implementar el diagrama sin recurrir a la función principal contenida en `ggplot2`, se van a explicar como ampliación de este paquete.

El paquete `ggiraph` [20] ofrece la posibilidad de crear barras interactivas dentro de las funciones de `ggplot2`. Para ello se proporcionan funciones que las sustituyen para realizar en la plantilla. En este caso, se va a sustituir la función `geom_bar` por `geom_bar_interactive`.

Lo primero es llamar a la librería y cargar la plantilla.

```
library(ggiraph)
p <- ggplot(data, aes(x=date, y= newcases, fill = CCAA,
    tooltip= as.factor(newcases), data_id =as.factor(date))) +
```

Como se puede observar, se han añadido 2 nuevos argumentos `tooltip`, `data-id`. Estos argumentos establecen qué se muestra a la hora de pasar el ratón por encima del gráfico. `Tooltip` fija el valor que se debe mostrar en este caso, por cada barra de valor muestra dicha variable. `Data-id` fija qué se selecciona, en este ejemplo, se usa cada barra individualmente pero se puede implementar que se seleccione cada grupo de valores. Ahora se procede al uso de la función sustituta con los mismos argumentos que en el anterior ejemplo.

```
geom_bar_interactive(stat="identity") +
```

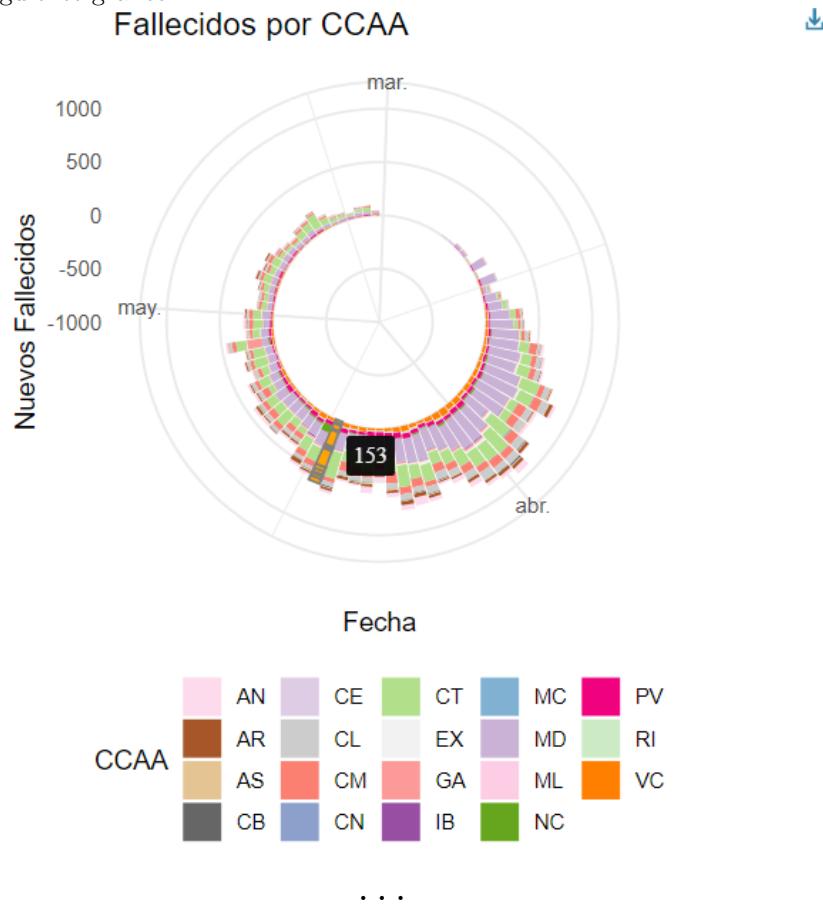
Se utilizan las mismas funciones que en el ejemplo anterior con solo `ggplot2` para crear el diagrama de iris de un ojo donde, entre ellas, figura la función principal para esta estética, `ylim`.

```
ylim(-1000,1000) +
coord_polar() +
scale_fill_manual(values = sample(col_vector, 19)) +
theme_minimal()+labs(title = "Fallecidos por CCAA") +
xlab("Fecha")+ylab("Nuevos Fallecidos")+
theme(legend.position = 'bottom')
```

Por último, para crear un gráfico interactivo, se necesita crear dicho objeto con una función propia del paquete `ggiraph`, `girafe`.

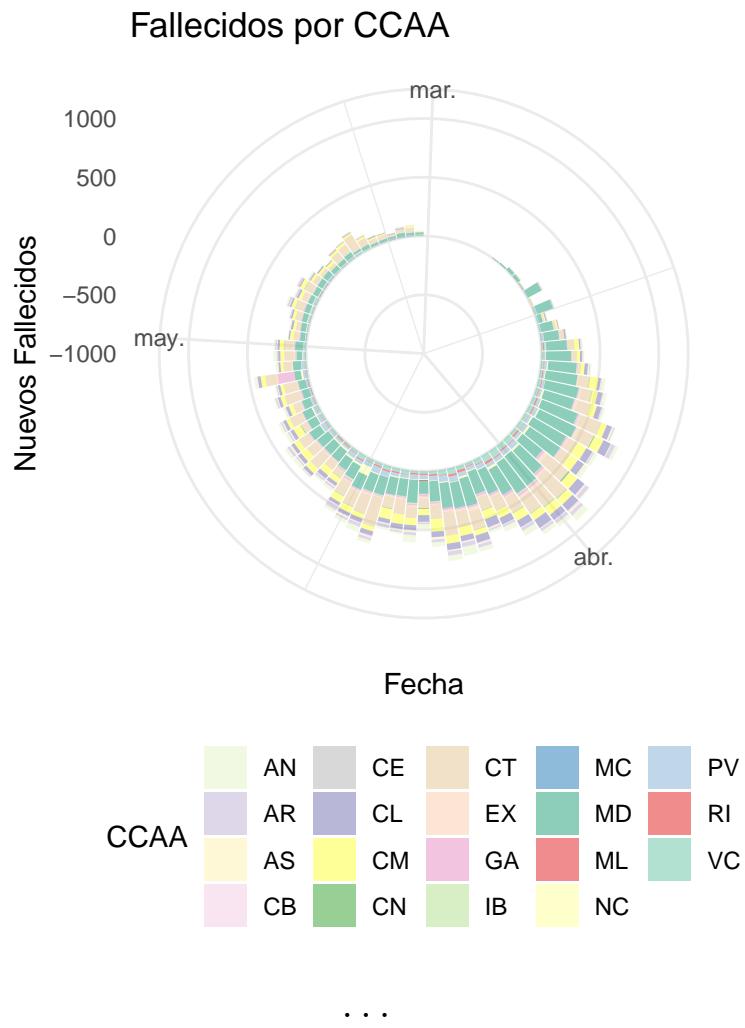
```
girafe(ggobj = p)
```

Resultando en el siguiente gráfico:



El otro paquete que depende de `ggplot2` es `ggiraphExtra` [25]. No es como su paquete `ggiraph` que sustituye una función, sino que tiene una propia para crear el diagrama de columnas radiales sin necesidad de apoyo de `ggplot2`. Para obtener la estética del diagrama que se está implementando es necesario hacer uso de la función `ylim` y, ya que se hace uso de esa función, se utilizan las demás para mejorar estética. Este paquete ofrece un parámetro para convertir el gráfico a interactivos, pero si se usan las funciones de otro paquete para la ayuda a la implementación se vuelve incompatible. Dentro de la función tiene el argumento de decidir si aplicar la función `coord-polar` por defecto.

```
library(ggiraphExtra)
ggBar(data, aes(x=date, y= newcases, fill = CCAA), stat="identity",
  alpha=0.5, polar=TRUE) +
  scale_fill_manual(values = sample(col_vector, 19)) +
  ylim(-1000,1000) +
  theme_minimal() + labs(title = "Fallecidos por CCAA") +
  xlab("Fecha") + ylab("Nuevos Fallecidos") +
  theme(legend.position = 'bottom')
```



3.10.2. Paquete circlize

Para realizar el diagrama con este paquete [18] se va a optar por la misma función que se utilizó en el Diagrama de Columnas Radiales Subsección 3.4, solo que esta vez se van a implementar la variante de columnas apiladas y con una función similar a `ylim` de `ggplot2`.

Se vuelven a usar los mismos datos que en el paquete anterior para mejorar la comparación entre ellos, pero debido a la gran cantidad de CCAA, y que cada barra necesita de dos líneas de código, se opta por seleccionar las tres provincias con mayores valores.

```
dataMD <- data[data$CCAA == ("MD"),]
dataCT <- data[data$CCAA == ("CT"),]
dataCM <- data[data$CCAA == ("CM"),]
```

Lo primero que se puede hacer es fijar parámetros, como el de en qué ángulo del círculo empieza el diagrama. Esta parte es opcional, solo aumenta la personalización.

```
library(circlize)
circos.par("start.degree" = 90, cell.padding = c(0, 0, 0, 0))
```

Lo que sí se necesita para implementar el diagrama, es inicializar el gráfico fijando de cuantos sectores se compone cada anillo del gráfico de circlize y cuales son los límites del eje X, estableciendo como se construye cada anillo.

```
circos.initialize("a", xlim = c(0.5, 91+0.5)) # 'a` just means there is one sector
```

Ahora se procede a crear solo el anillo para implementar las columnas, donde la variable `track.height` es con la que hay que jugar para tener la estética de un iris de un ojo. Y fijar los límites del eje Y que se representan en el anillo.

```
circos.track(ylim = c(0, 1000), track.height = 0.6,
             bg.border = NA, panel.fun = function(x, y) {
```

Dentro del anillo se establece con `panel.function` lo que se realiza dentro. Y para implementar las columnas, al igual que en el Diagrama de Columnas Radiales y Barras Radiales, se vuelve a utilizar `circos.rect` que crea un rectángulo pasando por argumentos los parámetros de xInicial, yInicial, xFinal y yFinal. Para el primer grupo de datos el rectángulo empieza en 0 y termina en su valor.

```
xlim = CELL_META$xlim
circos.rect( 1:91 - 0.45, rep(0, 91), 1:91 + 0.45, dataMD$newcases,
            col = "burlywood", border = "white")
```

Para los dos siguientes grupos es necesario poner esos valores empezando en el final del valor anterior y terminando en el valor del anterior mas el propio.

```
circos.rect( 1:91 - 0.45, dataMD$newcases+0.001, 1:91 + 0.45,
             dataCT$newcases+dataMD$newcases, col = "coral", border = "white")
circos.rect( 1:91 - 0.45, dataCT$newcases+dataMD$newcases+0.001, 1:91 + 0.45,
             dataCM$newcases+dataCT$newcases+dataMD$newcases,
             col = "blueviolet", border = "white")
```

Para tener una mejor estética se escogen colores agradables y que el borde de los rectángulos sea blanco para una mejor diferenciación. Para mejor comprensión se recrea el eje con posición debajo del anillo y las etiquetas apuntando fuera del anillo.

```
circos.axis(h="top", direction = "inside", labels.facing= "clockwise", labels.cex = 0.5,
            labels = dataMD$FECHA})
circos.yaxis(labels.cex = 0.4)
```

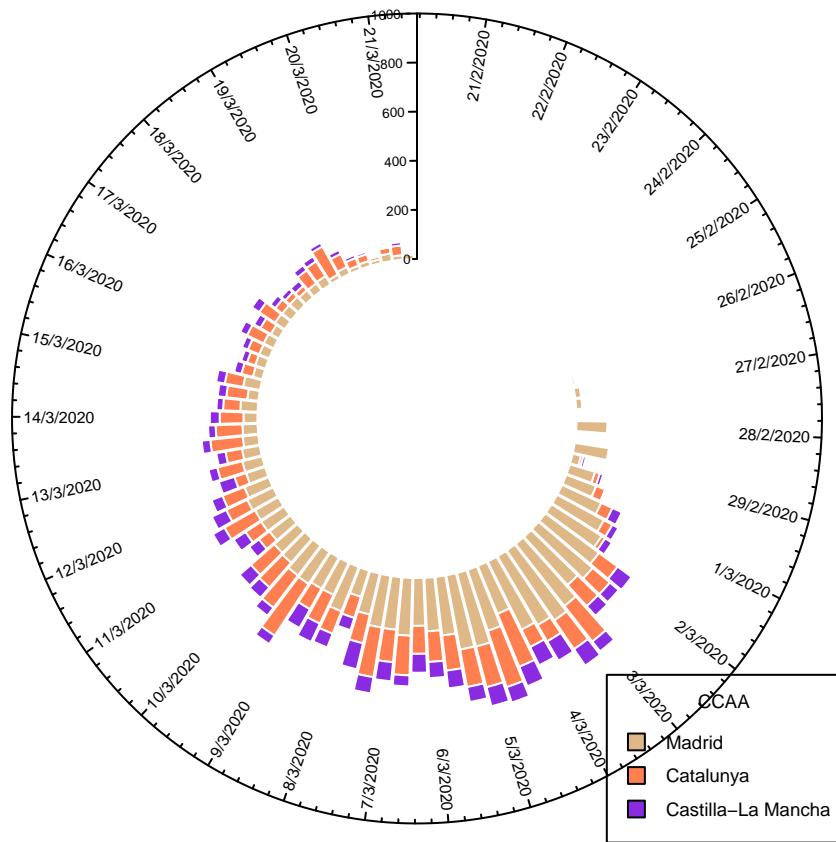
Y, por último, para volver a realizar un gráfico con este paquete es necesario limpiar los parametros guardados en `circos`.

```
circos.clear()
```

Para la leyenda del diagrama se opta por `legend`, una función básica de R para este propósito.

```
legend("bottomright", inset=.02, title="CCAA",
circos.clear()
circos.yaxis(labels.cex = 0.4)
           labels = dataMD$FECHA})
circos.axis(h="top", direction = "inside", labels.facing= "clockwise", labels.cex = 0.5,
            col = "blueviolet", border = "white")
            dataCM$newcases+dataCT$newcases+dataMD$newcases,
circos.rect( 1:91 - 0.45, dataCT$newcases+dataMD$newcases+0.001, 1:91 + 0.45,
            dataCT$newcases+dataMD$newcases, col = "coral", border = "white")
```

Resultando en el siguiente gráfico:



...

3.10.3. Comparaciones

Para implementar este diagrama se han encontrado cuatro paquetes. En tres de ellos, la función que crea el diagrama se basa en la función de uno de ellos y dos de ellos son expansiones del otro. Por ello solo se van a considerar dos paquetes a comparar: `ggplot2` y `circlize`.

Estos dos paquetes son los más completos; ya se han visto en otros diagramas, por lo que se sabe que los dos son complejos, pues necesitan de muchas funciones para crear el diagrama; con `ggplot2` para implementar un diagrama básico solo se recurre a la plantilla, las barras y el espacio en medio del círculo. Los dos paquetes tienen una gran capacidad de personalización, permitiendo incluso retocar cada aspecto del diagrama (barras, ejes, leyenda, escala de colores, etc). Lo único que puede aumentar el interés sobre un paquete u otro es que en el paquete `ggplot2`, el diagrama se puede guardar como una variable para poder utilizarse y/o modificarse más adelante y la parte interactiva que ofrecen los paquetes que pueden acompañar a `ggplot2` que se han visto en su ejemplo.

3.11. Diagrama de Líneas Circulares Multiseries

En este diagrama, al implementar un gráfico de líneas multivariable, queda descartado el paquete circlize y cualquier otro que ofrezca gráficos de líneas radial que no puedan crear diagramas donde se muestren varias líneas y no solo un grupo de datos.

Para realizar los posibles diagramas se van a intentar implementar dos tipos de gráficos, uno donde se representan solamente las multiples líneas y el otro tipo consiste en el gráfico de área donde se escogen las áreas de las multiples líneas representadas.

3.11.1. Paquete ggplot2

Este paquete [17] ofrece las dos posibles opciones, uno solo con las líneas y el otro con las áreas que crean dichas líneas, gracias a los métodos de `geom-line` y `geom-area`.

Para los dos métodos se recurren a los mismos datos para poder realizar una comparativa visual. Estos datos se escogen de la base de datos `babynames`, la cual ofrece los nombres más populares de los bebes en cada año, por lo que resulta en unos diagramas temporales donde el eje X serán los años y el eje Y la probabilidad, ofreciendo una línea por cada nombre.

```
library(babynames)
```

Pero para tener unos datos más manejables se va a acortar a solo ciertos nombres de bebes con género femenino. Para ello se empleara la libreria por excelencia para modificar datos.

```
library(tidyverse)
datos <- babynames %>%
  filter(name %in% c("Mary", "Emma", "Ida", "Ashley", "Amanda", "Jessica",
                     "Patricia", "Linda", "Deborah", "Dorothy", "Betty", "Helen")) %>%
  filter(sex=="F")
```

Después de tener los datos se procede a realizar el diagrama con el primer método, cargando las librerías necesarias.

```
library(ggplot2)
library(viridis)
```

Primero se establecen los datos que se usan, y en qué eje se distribuyen.

```
datos %>%
  ggplot(aes(x=year, y=n, group=name, color=name)) +
```

Ahora se hace uso de la función principal para el gráfico pero, como no se desea personalizar su ancho ni ninguna de las características que dispone ggplot2 para esta función, se deja con las características estándar.

```
  geom_line() +
```

Luego, para la estética del gráfico, se opta por la escala de colores viridis. También se define el título junto con el tema del fondo.

```
  scale_color_viridis(discrete = TRUE) +
  ggtitle("Nombres de bebes por popularidad") +
  theme_minimal() +
```

Se limitan los ejes del gráfico para dejar una espacio al principio del mismo con xlim, el espacio del centro con ylim y se cambia el sistema de coordenadas a polares indicando que empiece a 90 grados.

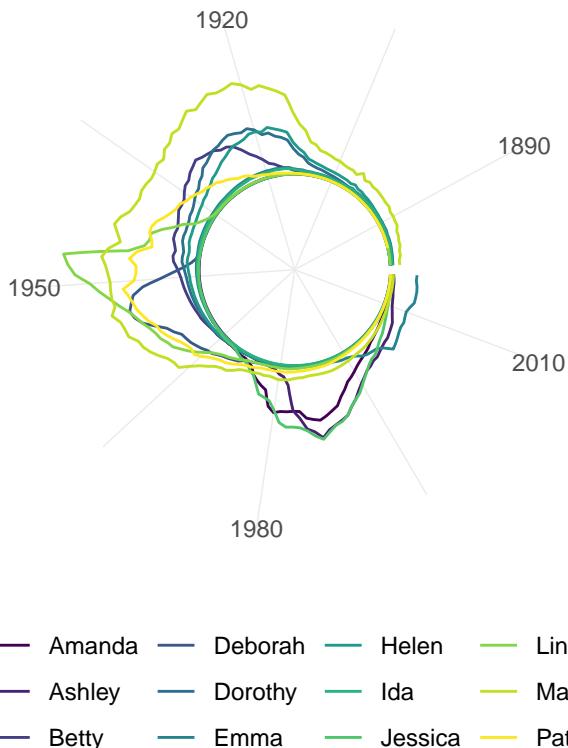
```
xlim(2018, 1879) +  
coord_polar(start = pi/2) +  
ylim(-70000,100000)+
```

Y, por último, se establece la leyenda y si se van a ver los ejes o texto de cada uno junto con el tamaño del título.

```
theme(plot.title = element_text(size=10),  
      legend.position = "bottom",  
      legend.title = element_blank(),  
      panel.grid.major = element_blank(),  
      axis.text.y = element_blank(),  
      axis.line = element_blank(),  
      axis.title.x = element_blank(),  
      axis.title.y = element_blank())
```

Resultando en:

Nombres de bebés por popularidad

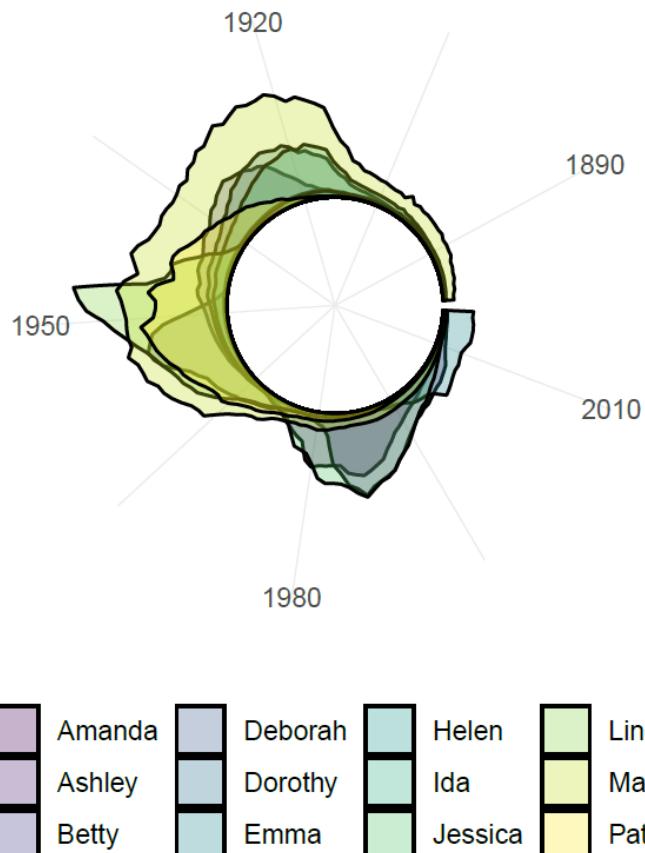


Para el segundo método se va usar las mismas funciones, solo que en vez de geom-line, se usa la función geom-area y estableciendo los mismos colores y argumentos del ejemplo anterior:

```
ggplot(datos, aes(x=year, y=n, fill = name)) +
  geom_area(position = "identity", alpha=0.3, colour= "black") +
  scale_fill_viridis(discrete = TRUE) +
  ylim(-70000,100000) +
  ggtitle("Nombres de bebes por popularidad") +
  theme_minimal() +
  xlim(2018, 1879) +
  coord_polar(start = pi/2) +
  theme(plot.title = element_text(size=10),
        legend.position = "bottom",
        legend.title = element_blank(),
        panel.grid.major = element_blank(),
        axis.text.y = element_blank(),
        axis.line = element_blank(),
        axis.title.x = element_blank(),
        axis.title.y = element_blank())
```

Resultando en un gráfico de áreas circulares multiseries

Nombres de bebes por popularidad



3.12. Diagrama de Burbujas Circular

Para poder representar este tipo de diagramas se requiere de una serie de datos como son: la posición de los círculos (posición X e Y) y el radio de estos. Para poder implementar este diagrama se ha conseguido encontrar dos paquetes que lo permiten; los dos usan una fórmula similar para crear los diagramas, utilizando una función para crear el layout que dice donde están los círculos para luego juntarlo con los datos de los radios.

3.12.1. Paquete ggraph-ggforce

En este apartado se va a necesitar del uso de la combinación de dos paquetes, `ggraph` [26] siendo este una ampliación del paquete `ggplot2` [17], para ayudar a este a poder realizar diagramas de redes mas complejos y completos, y `ggforce` [33], siendo otra ampliación de `ggplot2`, pero dedicado a mejor uso de datos para paquetes compatibles con `ggplot2` (como `ggraph`). Lo primero de todo será llamar a las librerías para luego poder trabajar con ellas.

```
library(ggraph)
library(ggforce)
```

Se utiliza una base de datos de población mundial, que consiste en continente, región, país y población.

```
github ="https://raw.githubusercontent.com/holtzy/data_to_viz/master"
dataset = "/Example_dataset/11_SevCatOneNumNestedOneObsPerGroup.csv"
url = paste0(github, dataset)
data <- read.table(url, header=T, sep=";")
data[ which(data$value==1), "value"] <- 1
colnames(data) <- c("Continent", "Region", "Country", "Pop")
head(data, 1)

##   Continent      Region      Country      Pop
## 1       Asia Southern Asia Afghanistan 25500100
```

Después se crean las áreas de las burbujas para representar la población de cada país.

```
areas <- data$Pop
```

Ahora, gracias a la función `pack-circles` del paquete `ggraph`, se obtienen las posiciones de las burbujas dependiendo del vector de áreas

```
posicion <- pack_circles(areas)
```

Por último, a la hora de confeccionar los datos se crea la tabla con X, Y, el radio de las burbujas a partir de sus áreas, el continente para diferenciación de color y, nombre de cada burbuja.

```
datos <- data.frame(x = posicion[, 1], y = posicion[, 2], r = sqrt(areas / pi),
                      group = data$Continent, name= data$Country)
```

A la hora de implementar el gráfico se hace uso de `ggplot` y la función de `ggforce geom-circle`

Para ello se necesita establecer el espacio para implementar las burbujas (`ggplot`), y establecerlas usando las posiciones, y colorear en función del continente.

```
ggplot() +
  geom_circle(data=datos, aes(x0 = x, y0 = y, r = r, fill = group)) +
```

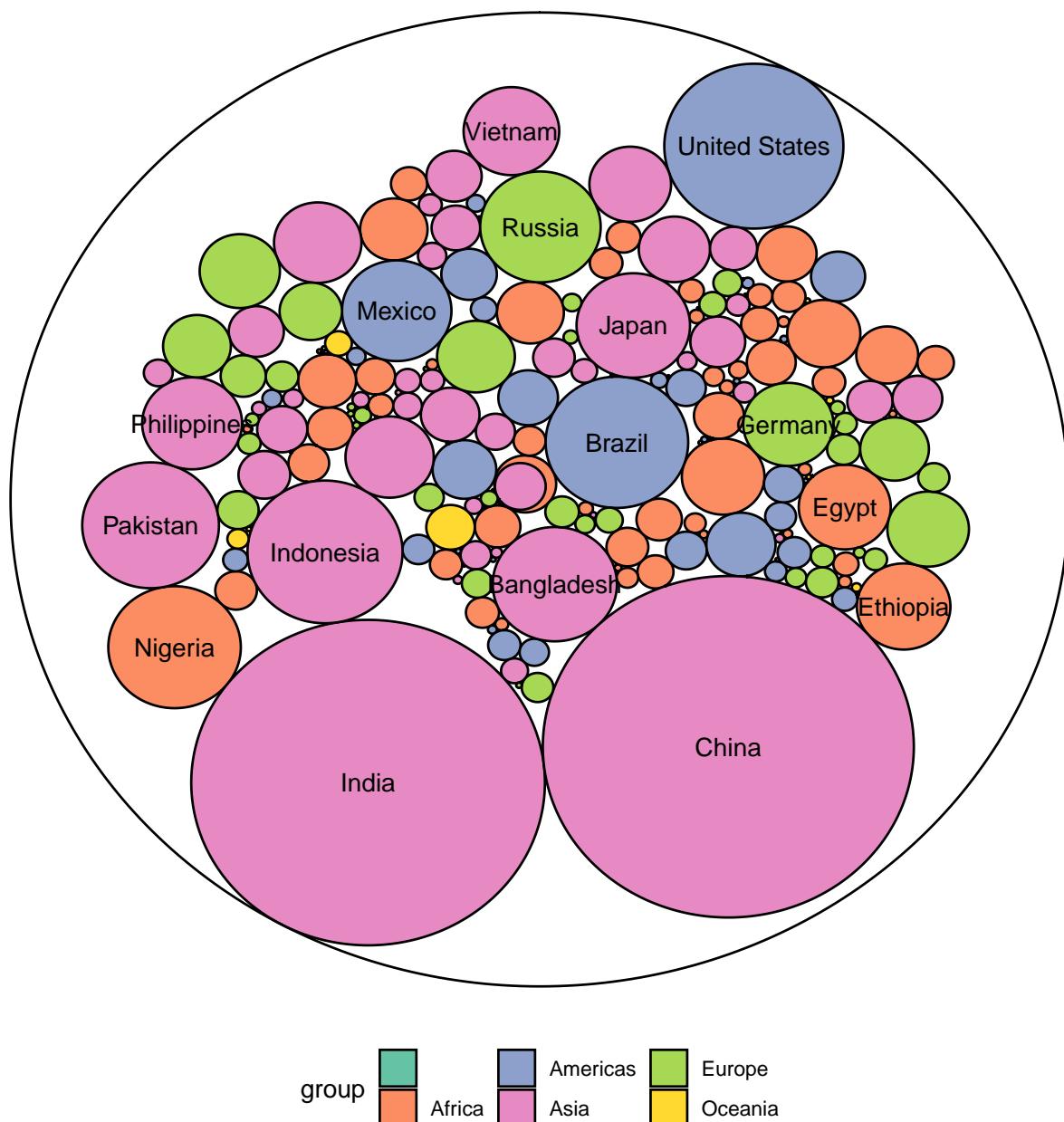
Luego para descartar el gráfico se crea un círculo que encapsula a las burbujas.

```
geom_circle(aes(x0 = 0, y0 = 0, r = attr(posicion, 'enclosing_radius')))) +
```

Y para la estética se opta por la escala de colores Set2, del propio paquete, y dejar el gráfico limpio de texto, salvo la leyenda y el nombre de las burbujas que tengan mayor radio.

```
scale_fill_brewer(palette = "Set2") +  
theme_void() +  
theme(legend.position = 'bottom') +  
geom_text(data = subset(datos, r > 5000), aes(x,y,label=name), color = "black")
```

Resultando en:



3.12.2. Paquete packcircles-ggplot2

Este paquete, [34] destinado a funciones de agrupamiento no solapado de círculos, ofrece una forma facil de empaquetar círculos dentro de otro para resultar en un diagrama de burbujas circular. Para poder crear las burbujas se necesita un área, para ello se puede utilizar una función del propio paquete para realizar el ejemplo. Este paquete solo ofrece los datos, para poder realizarlos es necesario el paquete ggplot2 [17] , y este se ofrece una gran capacidad de personalización de todo el gráfico. Se establecen los mismos datos que en el anterior gráfico, con la población mundial. Se crea el layout, con las burbujas no solapadas, gracias a `circleProgressiveLayout`, resultando en un data.frame con la posición en el eje X, Y y el radio de las burbujas (según población).

```
library(packcircles)
pack <- circleProgressiveLayout(data$Pop, sizetype = 'area')
```

Se unen los dos data.frames para luego agruparlo según el número de puntos que se desean (Cantidad de países).

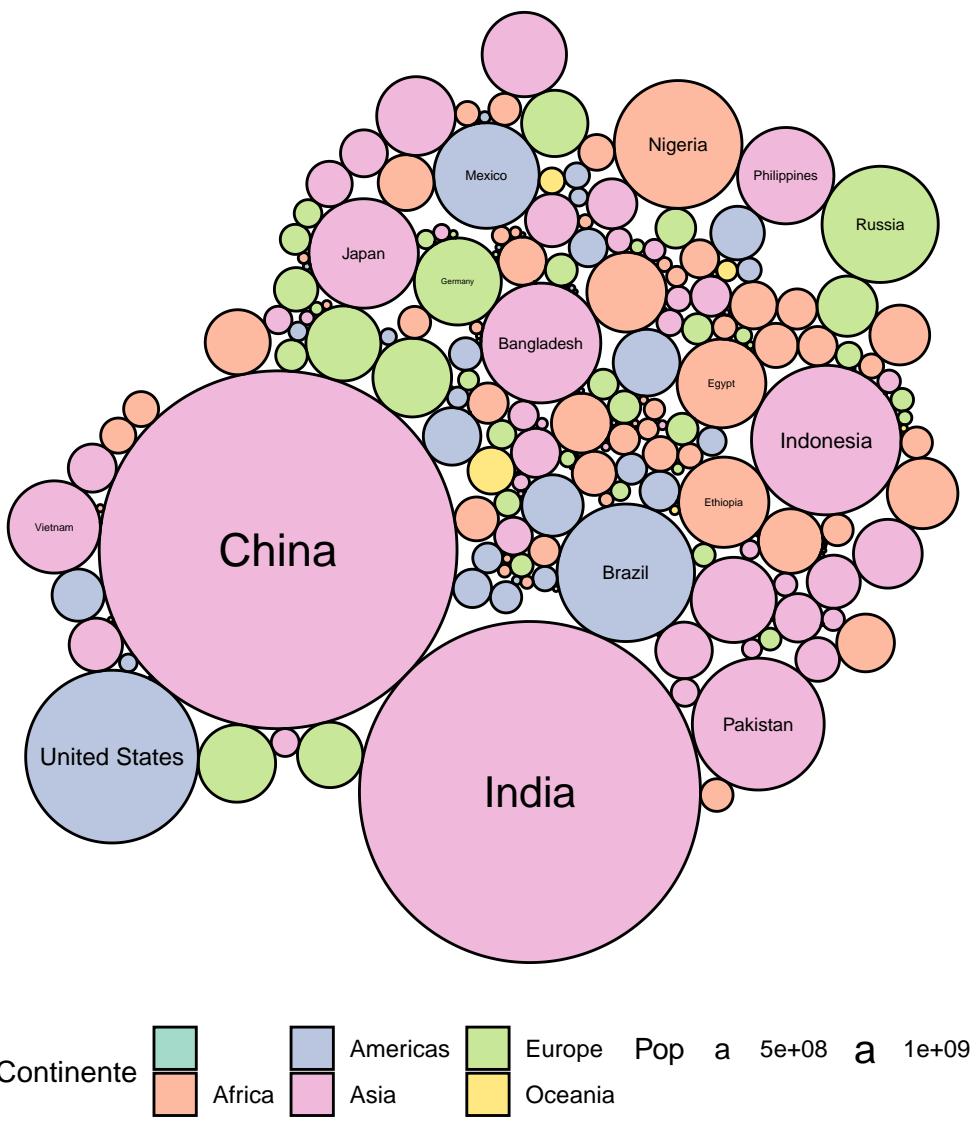
```
dat <- cbind(data, pack)
dat.gg <- circleLayoutVertices(pack, npoints = nrow(data))
```

Para poder diferenciar las burbujas por continentes, es necesario que en estos nuevos datos incluyan el continente.

```
dat.gg$Continent <- rep(data$Continent, each=251)
```

Y gracias a la librería ggplot2, se va a proceder a recrear el resultado con el uso de `geom-polygon` para establecer los círculos que encierran los datos pasados como argumento. Y como en el anterior gráfico, se pone la etiqueta de los países que superen un cierto radio (Mayores Poblaciones).

```
library(ggplot2)
ggplot() +
  geom_polygon(data = dat.gg, aes(x, y, group = id, fill=as.factor(Continent)),
               colour = "black", alpha = 0.6) +
  geom_text(data = subset(dat, radius > 5000), aes(x, y, size=Pop, label = Country),
            color = 'black') +
  scale_fill_brewer(palette = "Set2") +
  theme_void() + labs(fill = "Continente") +
  theme(legend.position='bottom') +
  coord_equal()
```



3.12.3. Comparaciones

Los dos paquetes que se han encontrado, en realidad son variantes del paquete `ggplot2`, por lo que se obtiene la capacidad de personalización de éste. Una creada para la mejor implementación de redes y otra para una mejor agrupación de datos sobre círculos que no se solapan. El primer paquete `ggraph-ggforce`, es una combinación del paquete que ya se había visto; `ggforce` para crear la cuadrícula con forma circular al encapsular las burbujas y `ggraph` para trazarlas.

En el segundo paquete, `packcircles`, se utiliza de la misma manera que `ggraph` implementando con otra función dentro de `ggplot2`. Al provenir del mismo paquete raíz no hay diferencia a la hora de crear el diagrama, pero si a la hora de agrupar las burbujas.

3.13. Diagrama de Mapa de Árbol Circular

En este apartado se encuentra otra manera de representar diagramas de árbol con formato circular. En el anterior Diagrama Rayos de Sol se utilizaba una cuadrícula para representar los nodos y esta vez se usan círculos.

Para crear este diagrama, primero hay que conseguir unos datos con una estructura jerárquica que indique: como esta constituido el árbol, su raíz, las ramas, sus hojas, y para este grafo también es útil un valor que estime el tamaño de los círculos (como la población de un país).

Para implementar este gráfico se pueden usar dos paquetes, los dos fáciles de usar, ya que lo difícil es crear esas bases de datos.

3.13.1. Paquete ggraph

En este paquete [26], como se ha visto en el diagrama de Grid Árbol (3.8), se va a recurrir a la misma función y misma base de datos, con estructura de datos jerárquicos, donde las ramas son como se unen los nodos (from-to) y los vértices los propios nodos, con su nombre, tamaño y ruta. Sólo que ahora cambia el layout de `partition` a `circlepack`, pero por lo demás, consiste en lo mismo.

Se escogen los datos del propio paquete (`flare`) y gracias al paquete `igraph` se transforman en unos datos con los que poder trabajar.

```
library(ggraph)
ramas <- flare$edges
vertices <- flare$vertices
library(igraph)
migrafo <- graph_from_data_frame(ramas, vertices = vertices)
```

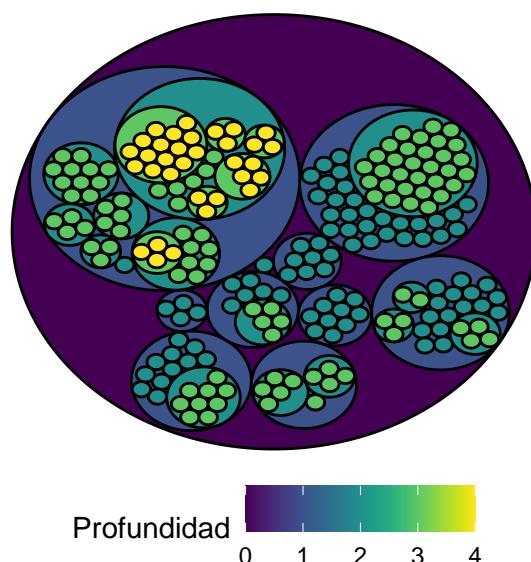
Para recrear el mapa de árbol circular se necesita llamar a la función principal de paquete, y para adjudicar la forma se establece que sean nodos circulares y que el color dependa de la profundidad (nivel de los nodos dentro del árbol).

```
ggraph(migrafo, layout = 'circlepack') +
  geom_node_circle(aes(fill = depth)) +
```

Y para obtener una estética vistosa se usa la escala de viridis, el fondo blanco y con leyenda abajo.

```
scale_fill_viridis() +
theme_void() +
theme(legend.position = "bottom") + labs(fill = "Profundidad")
```

Resultando en el gráfico deseado.



Como con estos datos no se pueden poner unas etiquetas claras para saber qué es cada uno de los nodos se procede, como en el diagrama grid árbol, a eliminar un nivel para mejor visualización.

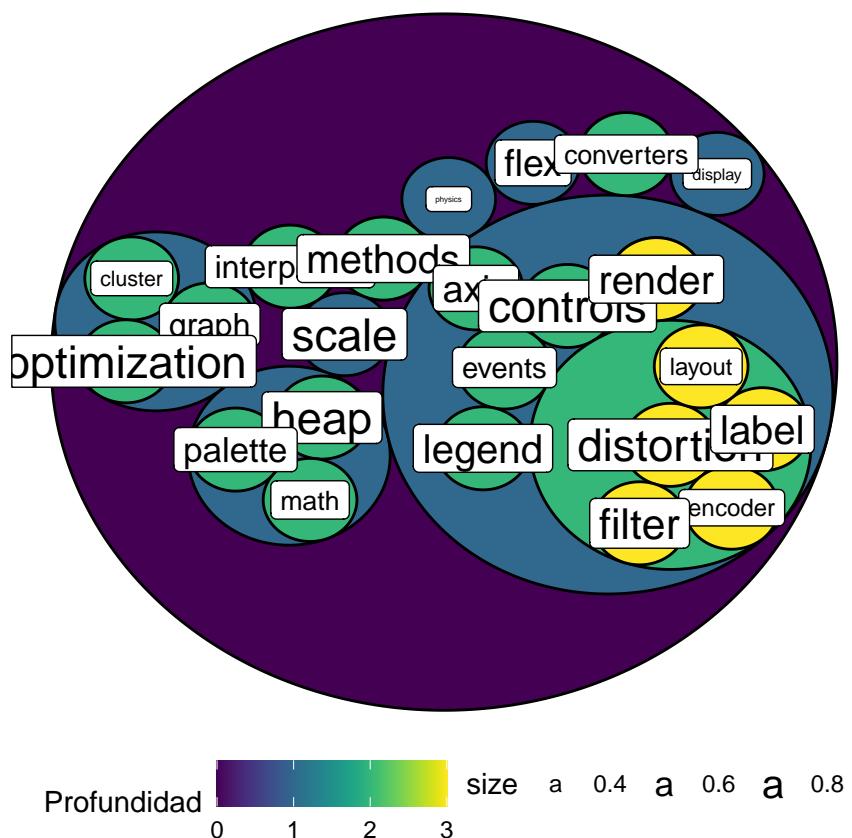
```
library(tidyverse)
ramas <- flare$edges %>%
  filter(to %in% from) %>%
  droplevels()
vertices <- flare$vertices %>%
  filter(name %in% c(ramas$from, ramas$to)) %>%
  droplevels()
vertices$size <- runif(nrow(vertices))
```

Y se necesita volver a construir de nuevo los datos para el gráfico.

```
migrafo <- graph_from_data_frame( ramas, vertices=vertices )
```

Y ahora, gracias a este acortamiento de los datos, se pueden poner la etiquetas con el añadido de la orden `geom_node_label`, resultando en la siguiente orden compleja y el gráfico resultante.

```
ggraph(migrafo, layout = 'circlepack') +
  geom_node_circle(aes(fill = depth)) +
  geom_node_label(aes(label = shortName, filter = leaf, size = size)) +
  scale_fill_viridis() +
  theme_void() +
  theme(legend.position = "bottom") + labs(fill = "Profundidad")
```



3.13.2. Paquete ciclepackeR

Este paquete [35] está dentro de un proyecto para crear nuevos diagramas con el paquete `htmlwidget` [30], por lo que ofrece un diagrama interactivo de un mapa de árbol circular. También necesita una estructura de datos parecida para poder utilizarse pero, aparte de esta complejidad, este gráfico genera mapas de arboles circulares interactivos. Al igual que con los diagramas de sectores se mostrará el gráfico, pero también una forma de guardarlo en html para usar la interactividad.

Primero se necesita descargar el paquete del creador.

```
devtools::install_github("jeromefroe/circlepackeR")
library(circlepackeR)
```

Y para los datos, por ejemplo, se encuentra, como se ha dicho en la introducción de este diagrama, una base de datos con la poblacion de cada pais. Donde se obtiene en cada fila el valor de poblacion, su nombre (País), el padre inmediato (Región) y el 'abuelo' (Continente).

```
url<- "https://raw.githubusercontent.com/"
github <- "holtzy/data_to_viz/master/Example_dataset/"
archivo <- "11_SevCatOneNumNestedOneObsPerGroup.csv"
library(tidyverse)
data <- paste0(url, github, archivo) %>%
  read.table( header=T, sep=";")
```

Se preparan los datos quitando lineas problemáticas y transformandolos a una estructura en forma jerárquica.

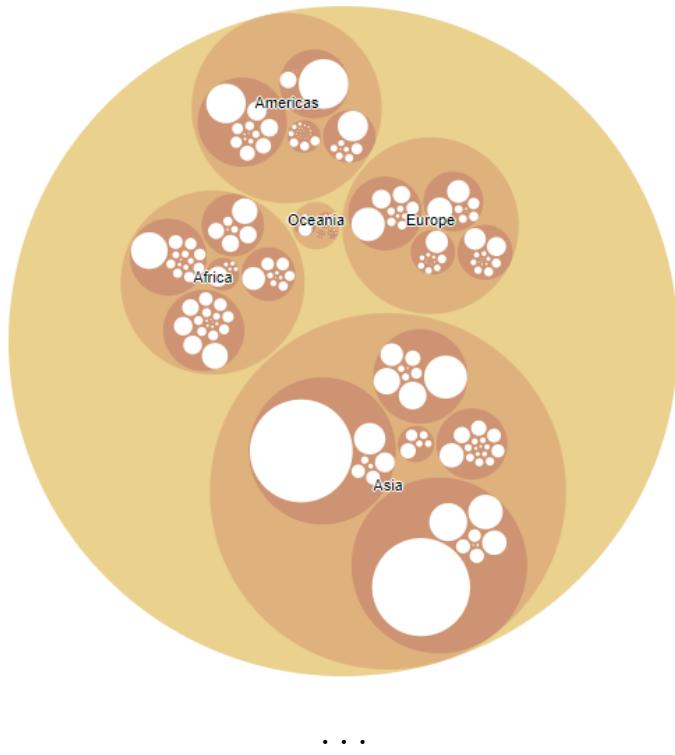
```
data[ which(data$value==1), "value"] <- 1
colnames(data) <- c("Continente", "Region", "Pais", "Pob")
data <- data %>% filter(Continente!="") %>% droplevels()
data$pathString <- paste("Mundo", data$Continente, data$Region, data$Pais, sep = "/")
library(data.tree)
poblacion <- as.Node(data)
```

Y gracias a su función principal se crea el gráfico .

```
p <- circlepackeR(poblacion, size = "Pob", color_min = "hsl(56,80%,80%)"
, color_max = "hsl(341,30%,40%)")
```

Para guardalo en html y disfrutar de la interactividad se necesita de lo siguiente.

```
library(htmlwidgets)
saveWidget(p, file="tree_interactive.html")
```



Otra forma de recrear este diagrama.

Se va a optar por la creación de la estructura de datos. Para ello se crea la matriz con los datos de la raíz, sus grupos, subgrupos y sub-subgrupos, con valores entre 1 y 50.

```
data_tree <- data.frame(
  root=rep("raiz", 30),
  group=c(rep("grupo A",15), rep("grupo B",5), rep("grupo C",10)),
  subgroup= rep(letters[1:5], each=6),
  subsubgroup=rep(letters[1:6], 5),
  value=sample(seq(1:50), 30)
)
```

Para que el paquete pueda tratar con ellos, se necesita transformarlos con el paquete `data.tree`, usado para crear la estructura de árbol requerida.

```
data_tree$pathString <- paste("mundo", data_tree$group, data_tree$subgroup,
                             data_tree$subsubgroup, sep = "/")
poblacion <- as.Node(data_tree)
```

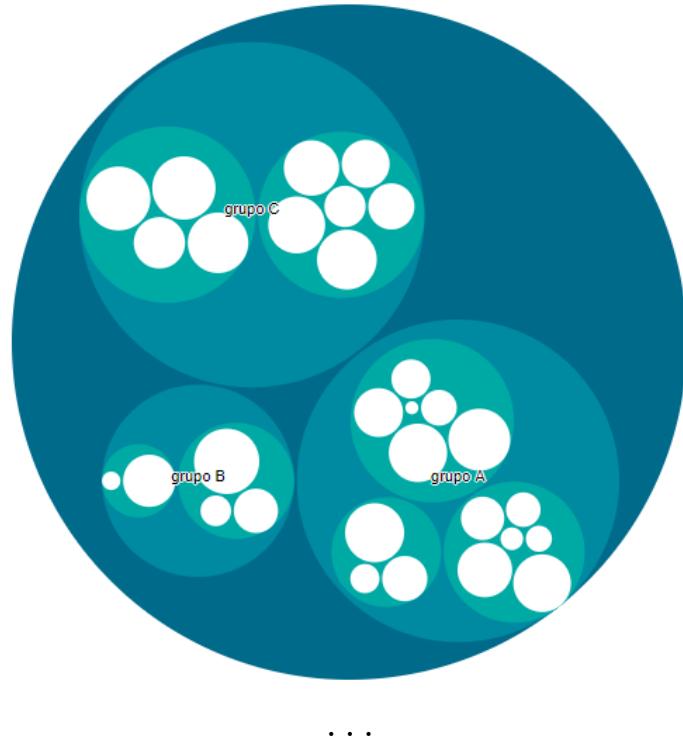
Ahora se procede a crear el gráfico, de la misma forma que el ejemplo anterior.

```
p <- circlepacker(poblacion, size = "value", color_min = "hsl(197, 81%, 22%)",
                    color_max = "hsl(119, 100%, 50%)")
```

Y luego guardarlo en html.

```
saveWidget(p, file="tree_pob_interactive.html")
```

Resultando en:



3.13.3. Comparaciones

Los dos paquetes encontrados para crear los mapas de árbol circulares proporcionan gráficos muy similares, salvo por la variable interactiva y la gran capacidad de personalización que ofrece **ggraph**, gracias a la compatibilidad con las funciones de **ggplot2**. El paquete **circlepackeR** utiliza sólo una función para crear el gráfico, por lo que es sencillo de aprender, y con **ggraph** se necesita saber: que primero hay que crear la plantilla para después elegir qué función se escoge para trazar en ella.

3.14. Diagrama Mapa de Árbol Voronoi

Para realizar este tipo de mapa de árbol solo se ha conseguido encontrar un paquete que realice este diagrama. De forma normal el diagrama voronoi tiene varios paquetes, pero para que tenga un formato de árbol circular solo se ha encontrado este paquete más específico.

3.14.1. Paquete voronoiTreemap

Este paquete [36] está destinado para este diagrama, ya que solo tiene una función para crear gráficos, unos datos de ejemplo (la economía según continente y país) y otra base de datos simulando el precio de los productos en Canadá. Primero se obtienen los datos necesarios, se cogen del propio paquete, consisten en una variable con los países, para dividir el grafo según su peso en la economía, y los continentes a los que pertenecen para agruparlos.

```
library(voronoiTreemap)
data(ExampleGDP)
head(ExampleGDP)

##      h1    h2        h3   color weight codes
## 1 Total Asia     China #f58321 14.84    CN
## 2 Total Asia     Japan #f58321  5.91    JP
## 3 Total Asia     India #f58321  2.83    IN
## 4 Total Asia South Korea #f58321  1.86    KR
## 5 Total Asia    Russia #f58321  1.80    RU
## 6 Total Asia Indonesia #f58321  1.16    ID
```

Debido a que la función principal `vt-d3` necesita datos en json, se usa otra de las funciones del paquete para crear el json.

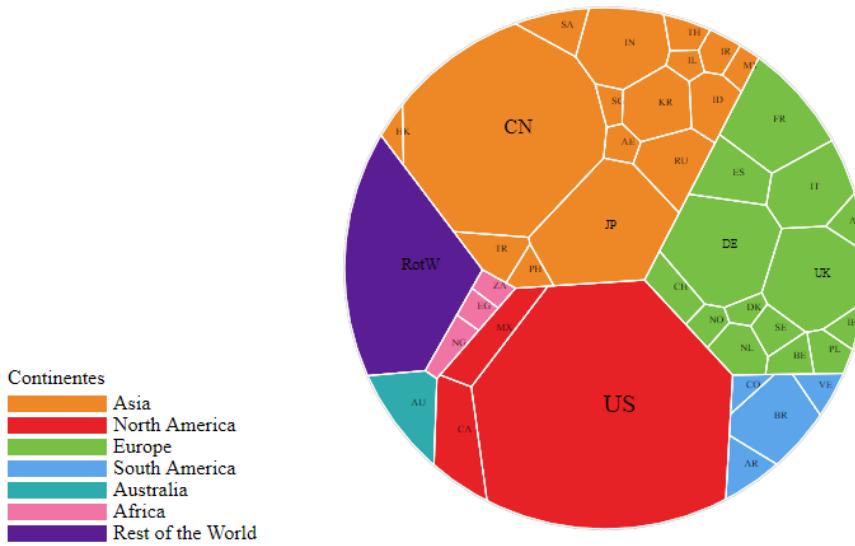
```
gdp_json <- vt_export_json(vt_input_from_df(ExampleGDP))
```

Y por último se emplea la función para crear el diagrama

```
p <- vt_d3(gdp_json, legend = TRUE, legend_title = "Continentes", seed = 1)
```

Debido a que cuenta con interactividad solo se muestra una foto del diagrama obtenido por este método.

```
library(htmlwidgets)
saveWidget(p,file="pol_interactive.html")
```



...

Para poder realizar los datos que se necesiten existen dos posibilidades: una, seguir la estructura de ExampleGDP, y dos crear la estructura de nodos gracias a `vt-create-node` y `vt-add-nodes` como en el siguiente ejemplo recreando el gráfico de [2, pág 194]. Representando la cantidad de proteínas de diferentes tejidos de un ratón, agrupándolas por su función.

Se ve como se pueden construir los nodos mediante las dos funciones.

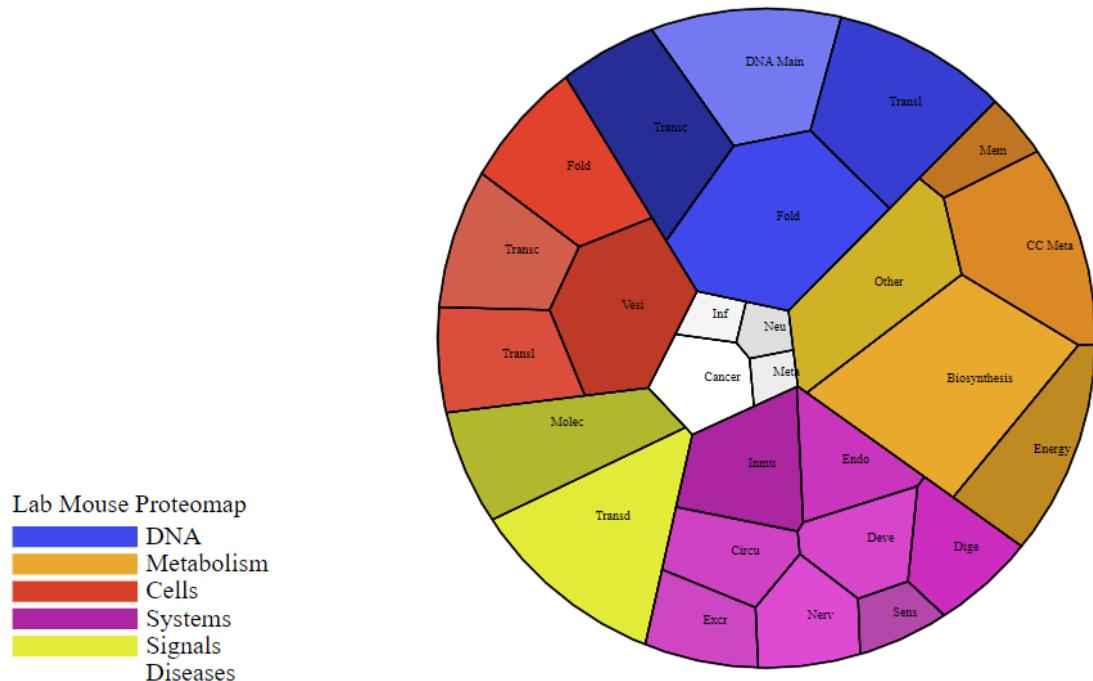
```
n <- vt_create_node("Proteinas")
n <- vt_add_nodes(n, refnode="Proteinas", node_names=c("DNA", "Metabolism", "Cells",
                                                       "Systems", "Signals", "Diseases"),
                  colors = c("#3f48eb", "#e8a92c", "#d64029", "#ab26a0", "#e2eb38", "#ffffff"))
n <- vt_add_nodes(n, refnode="DNA", node_names=c("DNA Maintenance", "Folding", "Translation",
                                                 "Transcription"), weights=c(0.7, 1, 0.8, 0.7),
                  codes=c("DNA Main", "Fold", "Transl", "Transc"),
                  colors = c("#7379f0", "#3f48eb", "#363ed1", "#272d96"))
```

Creando todos los nodos nos resulta la siguiente estructura.

		levelName	weight	code	color
1	Proteinas		NA		
2	--DNA		NA	DNA Main	#3f48eb
3	--DNA Maintenance		0.70	Fold	#3f48eb
4	--Folding		1.00	Transl	#363ed1
5	--Translation		0.80	Transc	#272d96
6	°--Transcription		0.70		
7	--Metabolism		NA		#e8a92c
8	--Biosynthesis		1.20	Biosynthesis	#e8a92c
9	--Central Carbon Metabolism		0.70	CC Meta	#db8925
10	--Energy Metabolism		0.50	Energy	#bf8a21
11	--Membrane transport		0.20	Mem	#bf7521
12	°--Other Enzymes		0.70	Other	#d1b226
13	--Cells		NA		#d64029
14	--Vesicular Transport		0.70	Vesi	#bd3a26
15	--Cytoskeleton		0.60	Fold	#e0422d
16	--Cell Grow and death		0.50	Transl	#db4e3b
17	°--Cell Communications		0.50	Transc	#d15d4d
18	--Systems		NA		#ab26a0
19	--Immune System		0.50	Inmu	#ab26a0
20	--Digestive System		0.35	Dige	#cc2dbf
21	--Endocrine System		0.35	Endo	#c936bd
22	--Circulatory System		0.35	Circu	#cf42c3
23	--Developement		0.35	Deve	#d645ca
24	--Nervous System		0.35	Nerv	#de4bd2
25	--Excretory System		0.30	Excr	#cc47c1
26	°--Sensory System		0.15	Sens	#b347aa
27	--Signals		NA		#e2eb38
28	--Signal Transduction		0.90	Transd	#e2eb38
29	°--Signals Molecules		0.70	Molec	#b1b82e
30	--Diseases		NA		#ffffff
31	--Cancers		0.30	Cancer	#ffffff
32	--Infectios Diseases		0.10	Inf	#f5f5f5
33	--Metabolic Diseases		0.08	Meta	#dedede
34	°--Neurodegerative Diseases		0.10	Neu	#dedede

Y ofreciendo el siguiente gráfico.

```
data<-vt_export_json(n)
p <- vt_d3(data, label = TRUE,color_border = 'black', seed = 28
, legend = TRUE, legend_title = "Lab Mouse Proteomap")
saveWidget(p, file = "mouse_interactive.html")
```



3.15. Diagrama de Mapa Cirular

Para este diagrama, al consistir en establecer un punto en el mapa y crear un círculo en torno a él, se necesita de un mapa que se pueda modificar. El mapa que se ha encontrado que se pueda retocar de la forma deseada es uno de carreteras, por eso no hay diferentes tipos de mapas.

3.15.1. Paquete ggplot2

Para representar este tipo de diagrama hay que poder manipular datos de un mapa de carreteras. Para poder exponer se ha encontrado una función `geom-sf` para el paquete ggplot2 [17] que utiliza objetos del paquete sf para modificar datos geográficos extraídos del paquete tigris [37] , que ofrece datos sobre carreteras del censo de EEUU.

Se cargan las librerías y se pone sf como la clase por defecto para datos geográficos.

```
library(tigris)
library(sf)
options(tigris_class = "sf")
```

Para este ejemplo se va a usar la ciudad de Las Vegas en Nevada. Se descarga el condado a la que pertenece y se transforma para que la clase sf pueda crear con ggplot2.

```
clark_roads <- roads(state = "NV", county = "Clark", year = 2012) %>%
  st_transform(26910)
```

Después se busca el centro de las calles que se guardan en `clark_roads`.

```
centroides <- clark_roads %>% st_centroid()
```

Para representar el buffer map hay que escoger un punto que actúe como centro de él. En esta ocasión se elige una calle céntrica, Vegas DR

```
calle <- centroides[grep("Vegas DR", centroides$FULLNAME, ignore.case=T),]
pt <- calle[1,] # "Vegas DR"
```

Ahora hay que establecer la magnitud del mapa, partiendo del centro y usando el método de prueba y error, se consigue una cantidad adecuada para visualizar el mapa.

```
pt_buffer <- st_buffer(pt, 9000)
```

Para que solo se muestren las carreteras que están dentro del buffer se usa la intersección entre el mapa completo y la magnitud del buffer con `st_intersection`.

```
interseccion <- st_intersection(clark_roads, pt_buffer)
```

Y, por último, gracias a ggplot2 se puede visualizar este conjunto de carreteras formando un círculo.

```
library(ggplot2)
ggplot(interseccion) +
  geom_sf(color = "#515b72") +
  theme_void() +
  theme(panel.grid.major = element_line("transparent"))
```



3.16. Dendograma Circular

Esta es la última forma de representar los diagramas de árbol con formato circular; en éste cada nodo se representa con un punto y la unión entre padres e hijos se representa con líneas.

Para crear este diagrama, como los anteriores de este tipo, se busca unas estructuras de datos jerárquicas para que se indique qué nodos hijo tiene cada padre y viceversa.

3.16.1. R Básico y ape

En esta parte se van a implementar los dendogramas circulares gracias al tratamiento de datos que ofrece la librería ape y una función básica de R.

Primero se obtienen los datos para poder trabajar con ellos, creando unos datos gracias al agrupamiento jerárquico (hierarchical clustering). Los datos constan de valores porcentuales de arrestos por asesinato, asalto, etc., agrupándolos por territorio.

```
data(USArrests)
head(USArrests,5)

##           Murder Assault UrbanPop Rape
## Alabama     13.2     236      58 21.2
## Alaska      10.0     263      48 44.5
## Arizona      8.1     294      80 31.0
## Arkansas     8.8     190      50 19.5
## California    9.0     276      91 40.6

dd <- dist(scale(USArrests), method = "euclidean")
hc <- hclust(dd, method = "ward.D2")
```

Ahora con la ayuda de ape [38] se transforman los datos y se realiza el gráfico.

```
library(ape)
```

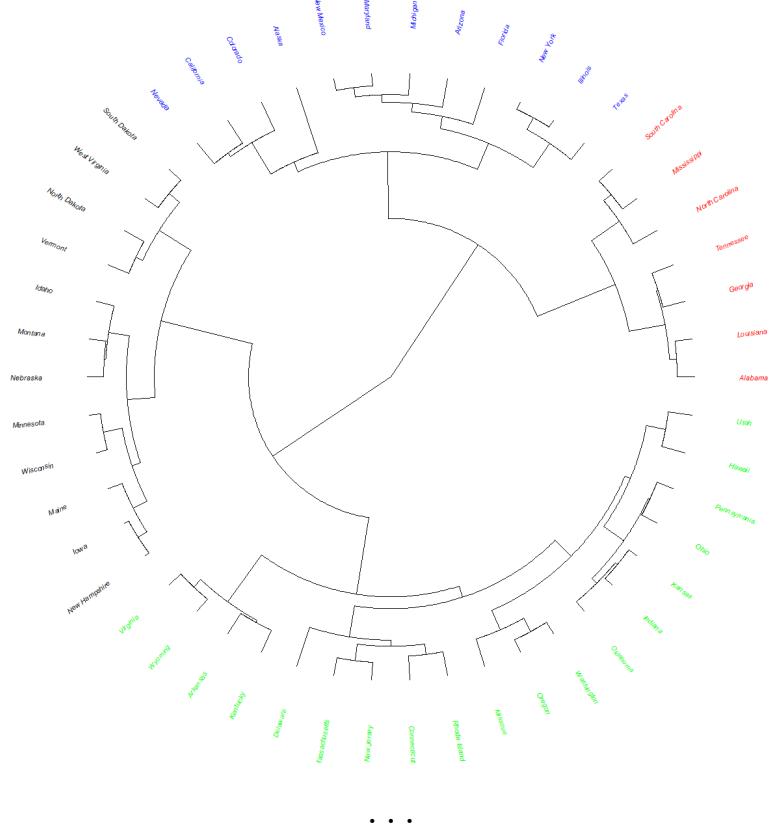
Se diferencian 4 grupos usando la clusterización, agrupándose según similaridad de los valores de las columnas, para una mejor visualización.

```
colors = c("red", "blue", "green", "black")
clus4 = cutree(hc, 4)
```

Y al final se usan la función de r básico plot, dando formato a los datos con `as.phylo` y con el tipo `fan` para darle el formato circular.

```
plot(as.phylo(hc), type = "fan", tip.color = colors[clus4],
     label.offset = 1, cex = 0.7)
```

Resultando en:



3.16.2. Paquete ggraph

Este paquete [26] ofrece una gran cantidad de gráficos de redes y, debido a que es una ampliación de ggplot2, tiene mucha capacidad de personalización para la creación de estos diagramas. Para poder realizar con este paquete un dendrograma se necesita crear los datos jerárquicos y, con este propósito entra el paquete **igraph** con la función **graph-from-data-frame**.

Primero se deben crear los data frame de la estructura de árbol con la raíz a sus hijos y los hijos a los suyos propios. Para este ejemplo se usará la base de datos flare ya usada en Diagrama Rayos de Sol.

```
library(ggraph)
ramas <- flare$edges
vertices <- flare$vertices
vertices$group = ramas$from[match( vertices$name, ramas$to )]
```

Al igual que la otra vez, se procede a reducir los niveles de profundidad, quitando el último.

```
library(tidyverse)
ramas <- flare$edges %>%
  filter(to %in% from) %>%
  droplevels()
vertices <- flare$vertices %>%
  filter(name %in% c(ramas$from, ramas$to)) %>%
  droplevels()
vertices$size <- runif(nrow(vertices))
vertices$group = ramas$from[ match( vertices$name, ramas$to ) ]
```

Ahora gracias a `igraph` se crea la estructuras jerárquica para que `ggrpah` las pueda leer y usar para el gráfico.

```
library(igraph)
mygraph <- graph_from_data_frame( ramas, vertices=vertices)
```

Para poder crear el diagrama se necesita la función principal de `ggrpah` aplicando el layout del dendograma y el formato circular.

```
ggraph(mygraph, layout = 'dendrogram', circular = TRUE) +
```

Para crear las ramas del árbol se hace uso de la siguiente función.

```
geom_edge_diagonal(colour="grey") +
```

Ahora se va a añadir el texto de los nodos junto con estos, representados por puntos, dando color según el grupo al que pertenezcan.

```
geom_node_label(aes(x = x*1.15, y=y*1.15, label = shortName, colour=group)) +
geom_node_point(aes(size = size, colour=group, filter = (!is.na(group)))) +
```

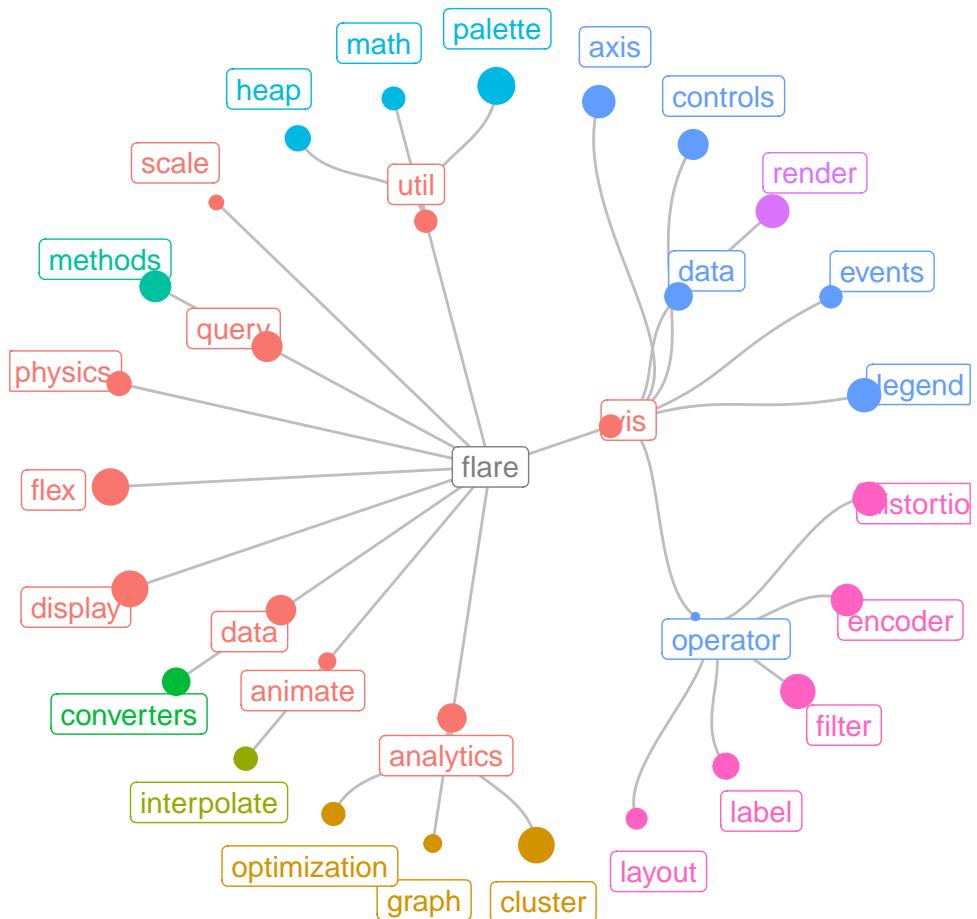
Se realiza un cambio de paleta de color con `RdPu`, para una mejor estética.

```
scale_edge_colour_distiller(palette = "RdPu") +
```

Y para quitar la leyenda y dejar un fondo blanco, para mejor visualización del gráfico, se usa:

```
theme_void() +
theme(legend.position = 'none')
```

Al final resultando en:



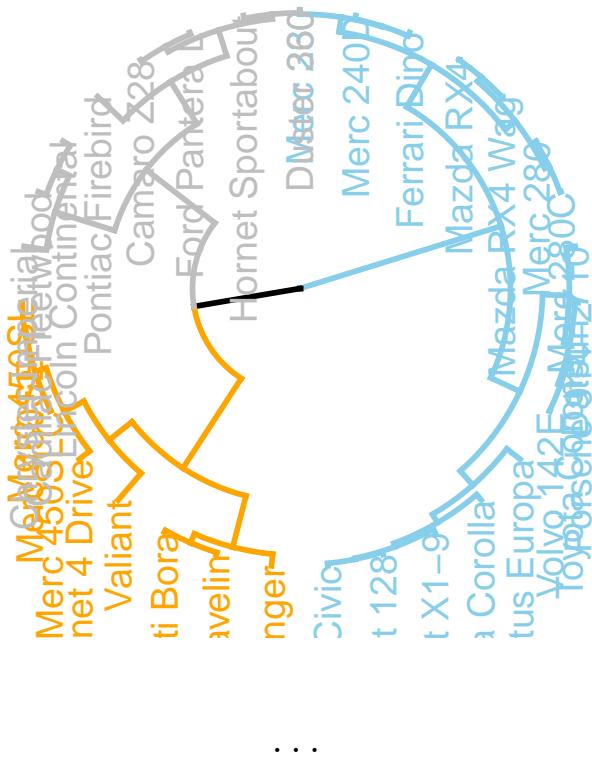
3.16.3. Paquete ggplot2

Para recrear un dendograma con este paquete [17] se tiene que obtener una estructura especial proporcionada por `dendextend`. Para crear el dendrograma se obtienen datos numéricos de la base de datos `mtcars` (datos de vehículos como consumo, cilindros, marchas, carburadores, etc.), como ejemplo, y se crean grupos con `hclust` mediante las distancias entre los valores numéricos, agrupando los vehículos por consumo, volumen de motor (`disp`) y cilindros (`cyl`), se establecen los colores de las ramas y nodos hoja.

```
library(tidyverse)
mtcars %>%
  select(mpg, cyl, disp) %>%
  dist() %>%
  hclust() %>%
  as.dendrogram() -> dend
library(dendextend)
datos <- dend %>%
  set("labels_col", value = c("skyblue", "orange", "grey"), k=3) %>%
  set("branches_k_color", value = c("skyblue", "orange", "grey"), k = 3)
```

Por último, se usa la transformación, al igual que en el Diagrama de Columnas Radiales; se transforma el eje X a formato radial. Pero se necesita revertir los datos por como los expone `ggplot2`.

```
library(ggplot2)
ggplot(datos) +
  scale_y_reverse() +
  coord_polar(theta = "x")
```



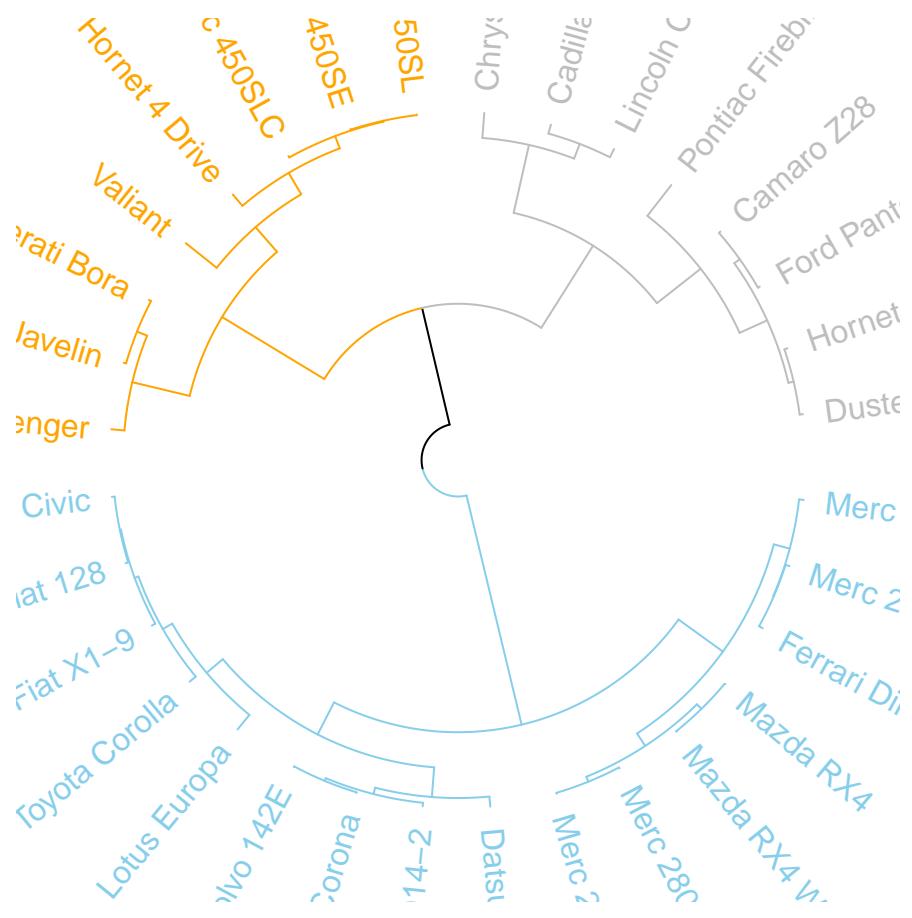
3.16.4. Paquete circlize

En este paquete [18] se observan dos maneras de crear un dendograma: una de ellos es gracias a la función `circlize-dendrogram`, la forma más sencilla, y la otra creando el texto y las ramas manualmente, más difícil. Para las dos formas se necesita otra vez la ayuda de `dendextend` para preparar los datos.

En la primera se va a usar los datos de `mtcars` para crear el árbol agrupando (clusters) los datos según su distancias entre mpg, cyl y disp como en el ejemplo de `ggplot2`.

Para realizar el dendrograma es tan sencillo como llamar a la función, y para personalizarlo basta con jugar con sus parámetros.

```
circlize_dendrogram(datos, dend_track_height = 0.8)
```



...

Para realizar la segunda forma del dendograma se van a usar los datos de una base de datos de ejemplo de circlize, `bird`. Y para la personalización se obtienen los nombres, el árbol cortado para mejor manejo, y la cantidad de pájaros que hay en el árbol.

```
load(system.file(package = "circlize", "extdata", "bird.orders.RData"))
nombres = hc$labels
ct = cutree(hc, 6)
n = length(nombres)
dend = as.dendrogram(hc)
```

Para realizar se establece el centro donde se va exponer el dendrograma.

```
circos.par(cell.padding = c(0, 0, 0, 0))
```

Y se inicia el gráfico dejando claro que se va a usar solo un sector, es decir, sin dividir el anillo y se fija los límites del eje x.

```
circos.initialize(factors = "a", xlim = c(0, n))
```

Ahora para representar las etiquetas del dendograma se realiza con la función `trackPlotRegion`.

```
circos.trackPlotRegion(ylim = c(0, 1), bg.border = NA, track.height = 0.3,
                      panel.fun = function(x, y) {
```

Con la función de texto, las etiquetas se ordenan como en las agujas de un reloj, cambian su orientación según el lugar que ocupen y el color dependiendo del grupo al que pertenezcan.

```
for(i in seq_len(n)) {
  circos.text(i-0.5, 0, nombres[i], adj = c(0, 0.5),
              facing = "clockwise", niceFacing = TRUE,
              col = ct[nombres[i]], cex = 0.7)})}
```

De nuevo, como en la primera forma, se establece el color de las ramas para que coincidan con el texto.

```
dend = color_branches(dend, k = 6, col = 1:6)
```

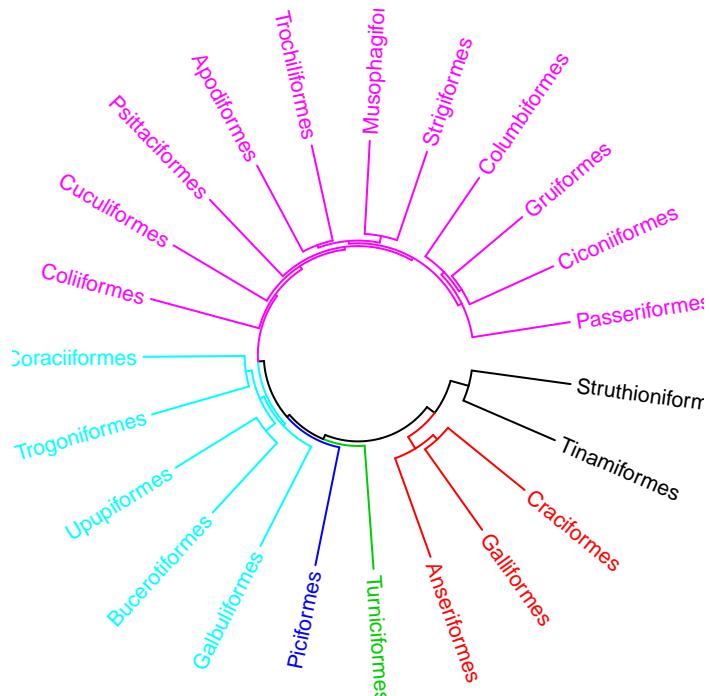
Por último, se procede a la realización de las ramas gracias a `trackPlotRegion` y la función `circos.dendrogram` para crear el dendograma con la altura del los arboles de `max-height`.

```
max_height = attr(dend, "height")
```

```
circos.trackPlotRegion(ylim = c(0, max_height), bg.border = NA,
                      track.height = 0.4, panel.fun = function(x, y) {
                        circos.dendrogram(dend, max_height = max_height)
                      })
```

Para dejar espacio a nuevos diagramas, con este paquete siempre se usa `circos.clear` para limpiar las variables usadas.

```
circos.clear()
```



3.16.5. Comparaciones

Como se hizo en el Diagrama de Sectores, este diagrama también se puede realizar con las funciones que proveé el lenguaje R sin paquetes adicionales, pero para crear los datos, sí que hace falta `ape`, por lo que se vuelve sencillo de usar con este método, aunque no con tanta personalización como con otros paquetes. El segundo paquete hallado, `ggraph`, es un gran conocido entre los diagramas de redes; para el diseño circular solo hace falta concretar un argumento. Este paquete es un poco más complejo de usar debido a que se necesitan varias funciones para crear el diagrama y personalizarlo.

El tercer paquete, `ggplot2`, al igual que con las funciones básicas de R, sólo sirve para realizar el diagrama, usando `dendextend` para los datos. Este paquete, al igual que el anterior, proporciona una gran personalización, aunque necesite más de una función para ello, convirtiéndolo en un paquete complejo. Una vez que sabes las funciones clave se vuelve intuitivo y sencillo de retocar.

El último paquete, `circlize`, es el más difícil de usar, pero también es completo y con gran capacidad de personalización. Este paquete ofrece dos posibles opciones para crear el diagrama, siendo la primera, la manera más fácil pero menos personalizable, ya que la segunda se realiza paso a paso.

En conclusión, los paquetes `ggraph` y `circlize` son los más potentes para implementar este diagrama por estar diseñados con diagramas de redes en mente; `circlize` solo con redes circulares.

3.17. Diagrama de Cuerdas

La estructura de datos necesaria para crear este diagrama es muy parecida al digrama de árbol circular, ya que se requiere algún tipo de estructura de origen y destino entre los nodos que componen el diagrama. Se puede aplicar a matrices de adyacencia por lo que se va a optar por ellas para realizar las estructuras. Para realizar este digrama se han encontrado dos paquetes que ofrecen el diagrama de cuerdas, uno de ellos interactivo.

3.17.1. Paquete Circlize

Se vuelve a ver este paquete [18], pero esta vez en vez de usar las funciones `circos.track`, `circos.trackPlotRegion` como las principales, solo son una ayuda para la función `chorDiagram`. Primero se preparan los datos con la estructura de origen y destino. Se ha optado por un set de datos que expone la migración entre países.

```
repositorio <- "https://raw.githubusercontent.com/"
github <- "holtzy/data_to_viz/master/Example_dataset/"
archivo <- "13_AdjacencyDirectedWeighted.csv"
url <- paste0(repositorio, github, archivo)
data <- read.table(url, header = TRUE)
colnames(data) <- c("Africa", "Este Asia", "Europa", "Ame. Latina", "Norte Ame.", "Oceania", "Sur Asia", "Sureste Asia", "Union Sovietica", "Oeste Asia")
rownames(data) <- colnames(data)
head(data)

##           Africa Este Asia Europa Ame. Latina Norte Ame. Oceania
## Africa      3.142471 0.000000 2.107883 0.000000 0.540887 0.155988
## Este Asia   0.000000 1.630997 0.601265 0.000000 0.973060 0.333608
## Europa      0.000000 0.000000 2.401476 0.000000 0.000000 0.000000
## Ame. Latina  0.000000 0.000000 1.762587 0.879198 3.627847 0.000000
## Norte Ame.  0.000000 0.000000 1.215929 0.276908 0.000000 0.000000
## Oceania     0.000000 0.000000 0.170370 0.000000 0.000000 0.190706
##           Sur Asia Sureste Asia Union Sovietica Oeste Asia
## Africa        0 0.000000          0 0.673004
## Este Asia     0 0.380388          0 0.869311
## Europa        0 0.000000          0 0.000000
## Ame. Latina    0 0.000000          0 0.000000
## Norte Ame.    0 0.000000          0 0.000000
## Oceania        0 0.000000          0 0.000000

library(tidyverse)
data_long <- data %>%
  rownames_to_column %>%
  gather(key = 'key', value='value', -rowname)
```

Para iniciar el gráfico, como se vió en el anterior uso de este paquete, se tiene que crear el `circos` con unos parámetros iniciales para tener una plantilla.

```
library(circlize)
circos.clear()
circos.par(start.degree = 90, gap.degree = 4,
          track.margin = c(-0.1, 0.1), points.overflow.warning = FALSE)
par(mar = rep(0, 4))
```

Para mejor visualización se opta por una paleta de color del paquete viridis.

```
library(viridis)
micolor <- viridis(10, alpha = 1, begin = 0, end = 1, option = "D")
micolor <- micolor[sample(1:10)]
```

Y, por último, se crea el plot final con la función `chorDiagram` y el texto que le acompaña.

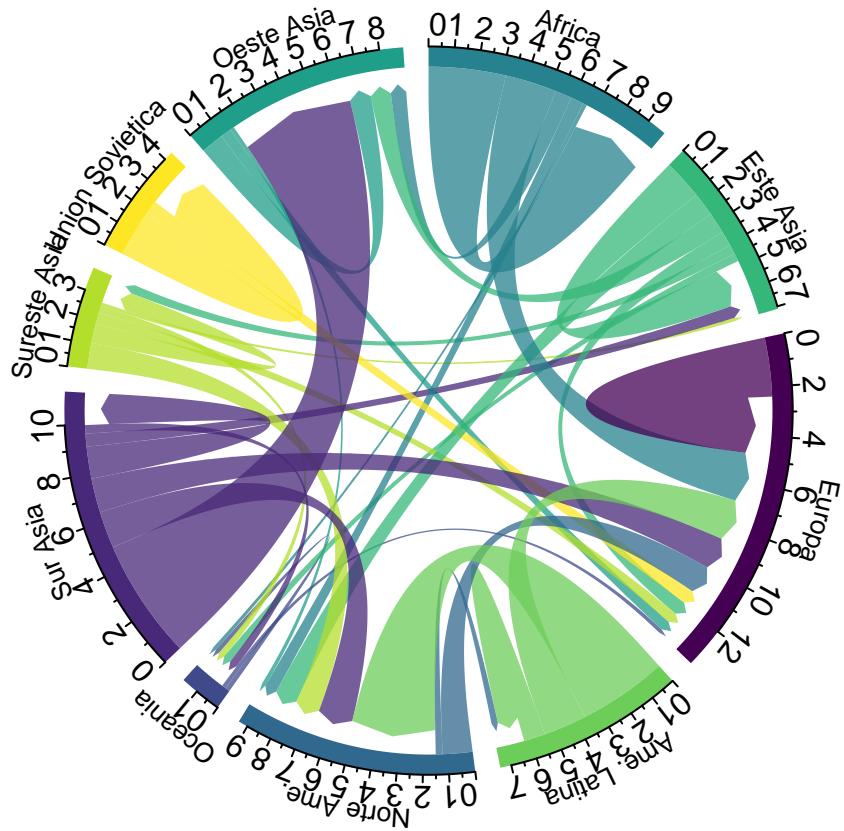
Primero se le pasa la matriz de adyacencia, los colores, qué tipo de grafo se elige (en este caso flechas direccionales) y los diferentes valores de la personalización.

```
chordDiagram(  
  x = data_long,  
  grid.col = micolor,  
  transparency = 0.25,  
  directional = 1,  
  direction.type = c("arrows", "diffHeight"),  
  diffHeight = -0.04,  
  annotationTrack = "grid",  
  annotationTrackHeight = c(0.05, 0.1),  
  link.arr.type = "big.arrow",  
  link.sort = TRUE,  
  link.largest.onTop = TRUE)
```

Luego, para personalizar el texto del diagrama, se puede usar la función `circos.trackPlotRegion` y dentro de ella `circos.text` y `circos.axis`.

```
circos.trackPlotRegion(  
  track.index = 1,  
  bg.border = NA,  
  panel.fun = function(x, y) {  
    xlim = get.cell.meta.data("xlim")  
    sector.index = get.cell.meta.data("sector.index")  
    circos.text(  
      x = mean(xlim),  
      y = 3.2,  
      labels = sector.index,  
      facing = "bending",  
      cex = 0.8  
    )  
    circos.axis(  
      h = "top",  
      major.at = seq(from = 0, to = xlim[2], by = ifelse(test = xlim[2]>10,  
                                                       yes = 2, no = 1)),  
      minor.ticks = 1,  
      major.tick.percentage = 0.5,  
      labels.niceFacing = FALSE)  
  }  
)
```

Resultando al siguiente gráfico de la migración.



...

3.17.2. Paquete chorddiag

Este paquete [39] sirve para crear los diagramas de cuerdas usando la librería D3 de visualización JavaScript, y utilizando el paquete `htmlwidgets` para guardar los diagramas interactivos.

Para el ejemplo se procede a realizar una representación de los supervivientes del Titanic, agrupados por sus clases.

Lo primero es obtener una lista que tiene los valores: clase, sexo, edad, si ha muerto y, la cantidad de personas con esas características.

```
library(tidyverse)
titanic_tbl <- dplyr::tbl_df(Titanic)
```

Ahora se ordena la lista por clase y si ha muerto.

```
titanic_tbl <- titanic_tbl %>%
  mutate_at(vars(Class:Survived), funs(factor))
```

Se eliminan los datos de edad y sexo, que no se necesitan.

```
by_class_survival <- titanic_tbl %>%
  group_by(Class, Survived) %>%
  summarize(Count = sum(n))
```

Se crea una matriz ponderada con los datos de la lista.

```
titanic.mat <- matrix(by_class_survival$Count, nrow = 4, ncol = 2)
dimnames(titanic.mat) <- list(Class = levels(titanic_tbl$Class),
                                Deceased = levels(titanic_tbl$Survived))
head(titanic.mat)
```

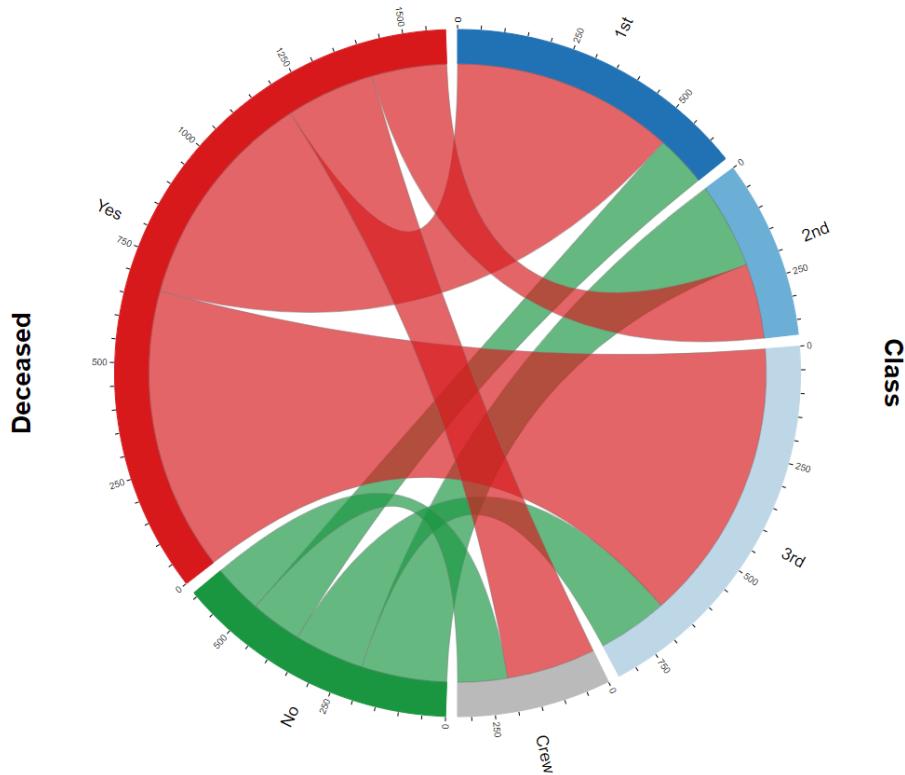
```
##      Deceased
## Class  No Yes
##   1st 122 528
##   2nd 203 178
##   3rd 167 673
## Crew 118 212
```

Y se seleccionan los colores para los grupos.

```
groupColors <- c("#2171b5", "#6baed6", "#bdd7e7", "#bababa",
                  "#1a9641", "#d7191c")
```

Y usando la función `chorddiag` para realizar el digrama.

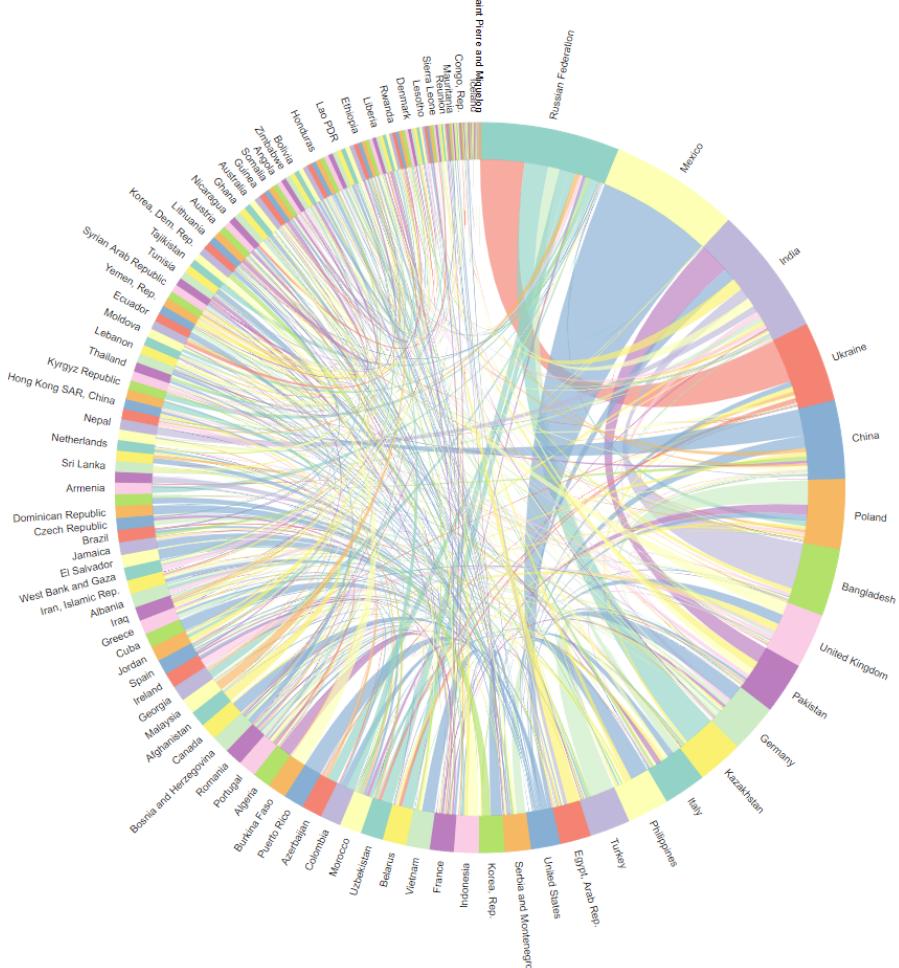
```
library(chorddiag)
p <- chorddiag(titanic.mat, type = "bipartite",
                 groupColors = groupColors,
                 tickInterval = 50)
library(htmlwidgets)
saveWidget(p, file = "cuerdas_inte.html")
```



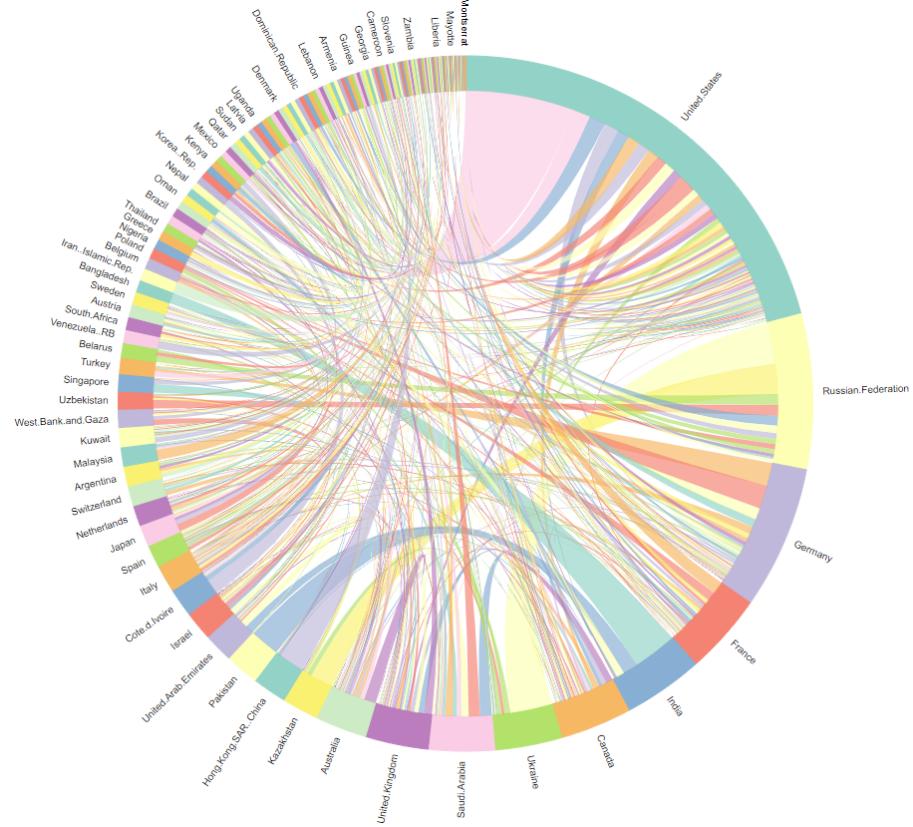
Para crear un ejemplo más complicado, se opta por representar las migraciones entre países como en el paquete anterior. Se prepara la matriz de adyacencia con los datos del paquete migration.indices.

```
library(migration.indices)
library(RColorBrewer)
data("migration.world")
sort.by.orig <- sort(rowSums(migration.world),
                      decreasing = TRUE, index.return = TRUE)
mig.sorted.by.orig <- migration.world[sort.by.orig$ix, sort.by.orig$ix]
row.names(mig.sorted.by.orig) <- names(sort.by.orig$x)
n <- dim(mig.sorted.by.orig)[1]
groupColors <- rep(brewer.pal(12, "Set3")[c(1:8, 10:12)], length.out = n)
groupNames <- rep("", n)
ix <- c(1:50, seq(52, 100, by = 2),
       seq(105, 150, by = 5),
       160, 180, 226)
groupNames[ix] <- colnames(mig.sorted.by.orig)[ix]
tooltipNames <- colnames(mig.sorted.by.orig)
```

Este es el gráfico resultante de la función chorddiag.



Este gráfico representa dónde emigran los habitantes de cada país, por lo que ahora simplemente con la matriz traspuesta se puede saber para cada país de qué países vienen sus inmigrantes.



3.17.3. Comparaciones

Para este diagrama se han encontrado dos paquetes; uno destinado a todo tipo de diagramas circulares, `circlize`, y otro, cuyo propósito es crear este tipo de diagramas, `chorddiag`. El primero de los paquetes, que ya se ha visto anteriormente, utiliza una función para el diagrama, pero para personalizarlo en todos los aspectos posibles se necesita hacer uso de una gran variedad de funciones, lo que lo vuelve un paquete más complejo y difícil de dominar.

El segundo paquete, al haberse realizado para este tipo de diagramas, está más centrado para esta tarea; con este paquete, aparte de crear el diagrama de cuerdas normal, permite implementar la variable interactiva que, para grandes cantidades de datos, es de lo más útil. Este paquete proporciona casi la misma capacidad de personalización que `circlize`, por lo que en conclusión, `chorddiag` es el más completo e implementa mejor el diagrama, debido a la interactividad, ya que con un simple click aisla todas las cuerdas no relacionadas para mejor entendimiento de este.

3.18. Diagrama de Redes Circulares

Para esta sección se va a hacer uso de un TFG [16] similar a este, pero dirigido a redes dentro de R. Se estudia el trabajo para identificar qué redes son potenciales de ser circulares y qué paquetes hacen los gráficos de dichas redes para generar ejemplos diferentes a los que ya estan expuestos.

3.18.1. Explosión Centralizada

La primera red que es posible realizar con diseño circular es una red que consiste en una organización caótica y desordenada de sus nodos.

Tal y como escoge su autor, se exploran 2 paquetes para realizar este diagrama; en el primero, **ggraph** [26] , se van a usar los mismos datos, ya que para su ejemplo solo se utiliza el algoritmo aleatorio para la colocación de los nodos y no se han explorado otros métodos.

Lo primero es la carga de librerías y datos.

```
library(igraph)
library(ggraph)
```

Se carga la librería **igraph** [27] para la manipulacion de los datos y transformarlos en una red que R entienda y la librería **ggraph** para el gráfico de la red.

```
archivo <-"weighted_adelman_social_songbird.graphml"
bird <- read_graph( archivo , format ="graphml")
```

El algoritmo elegido para que la red guarde un diseño circular es el de Fruchterman-Reingold, el cual realiza gráficos de redes dirigidos a la fuerza para que el resultado sea estéticamente placentero.

Se cargan los datos de los pájaros y se escoge el algoritmo.

```
ggraph(bird, layout ="fr") +
```

Para recrear los enlaces entre nodos se opta por líneas de color azul con poco grosor para que se confundan entre ellas lo menos posible.

```
geom_edge_link(edge_colour="lightblue", edge_alpha=0.5, edge_width=0.4) +
```

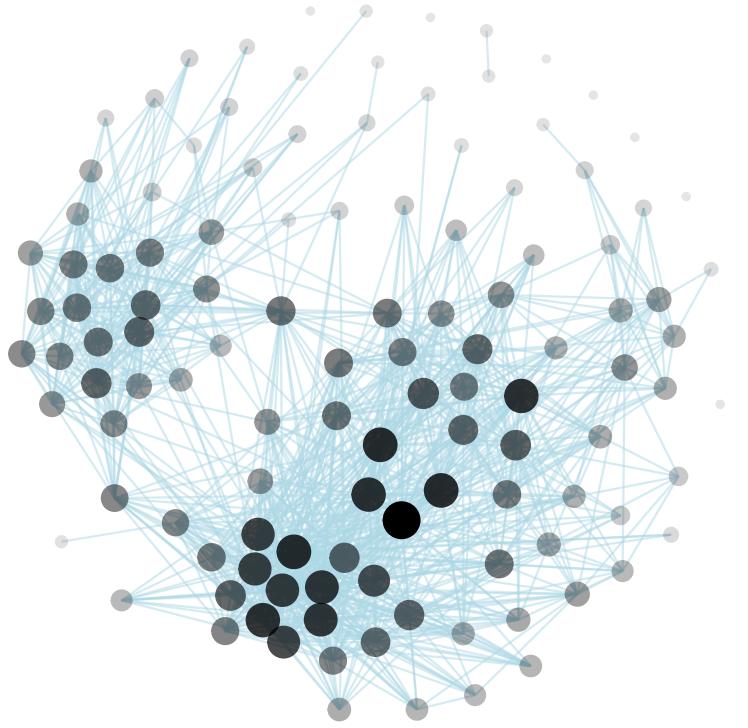
Y para los nodos se hace uso de **igraph** para la manipulación de los nodos en la magnitud y transparencia.

```
geom_node_point(aes(size = igraph::degree(bird)/2, alpha = igraph::degree(bird)/2)) +
```

Por último se limpia el fondo del gráfico y se elimina la leyenda para la mejor apreciación del diseño circular.

```
theme_void() +
theme( legend.position = "none")
```

Y resultando es:



...

3.18.2. Implosión Elíptica

Para este diagrama, el autor usa el paquete `qgraph` [40] para realizar estas redes ponderadas y, el ejemplo que usa para mostrar el diagrama, es el único con la función usada ya que no existe otro layout que trace un círculo. Pero si existe otra función que implementa este tipo de gráfico solo que es de forma dinámica.

Para usar la función se crean los datos de la red ponderada. Creando una red con los valores de origen y destino.

Se establecen el número de nodos, se crea un vector vacío para los grados y la matriz de vértices vacía.

```
n = 100
Grad <- rep(0, n)
V <- matrix(NA, n - 1, 2)
```

Primero se conectan los dos primeros nodos y se añade su grado.

```
V[1, ] <- 1:2
Grad[1:2] <- 1
```

Para luego realizar este proceso con todos los nodos restantes añadiéndolos con probabilidad proporcional a los grados.

```
for (i in 2:(n - 1)){
  V[i, 2] <- i + 1
  con <- sample(1:i, 1, prob = Grad[1:i]/sum(Grad[1:i]), i)
  Grad[c(con,i+1)] <- Grad[c(con,i+1)] + 1
  V[i, 1] <- con
}
```

Como la función para crear el gráfico exige una matriz de adyacencia, hay que crear una función para que se pase de una simple lista de vértices a la matriz deseada.

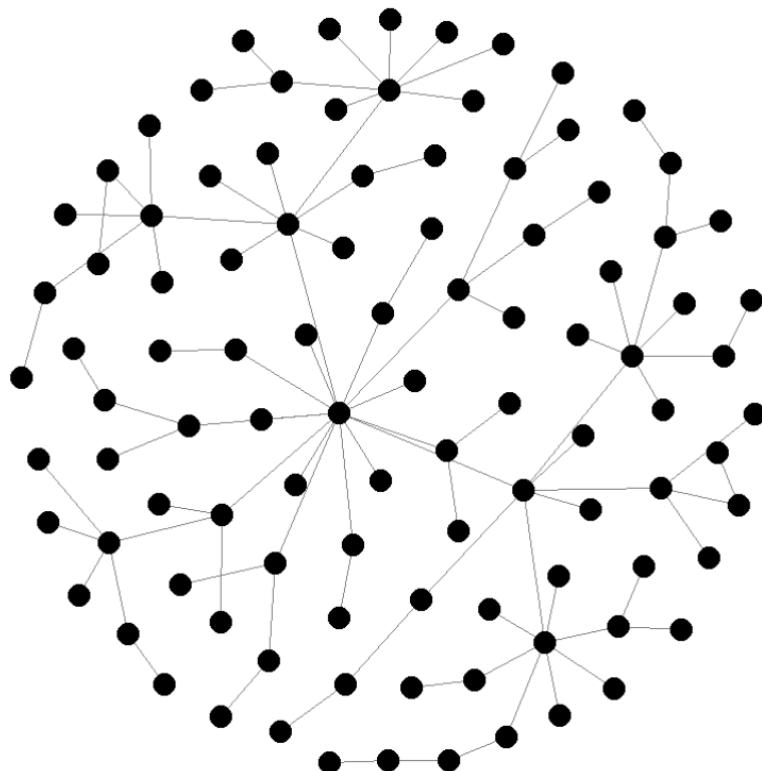
```
V2adj <- function(V,n){  
  adj <- matrix(0,n,n)  
  for (i in 1:nrow(V))  
  {  
    adj[V[i,1],V[i,2]] <- 1  
  }  
  adj <- adj + t(adj)  
  return(adj)  
}
```

Para usar la función en la matriz de vértices se hace uso de `lapply`, que devuelve una lista de la misma longitud que la de entrada, pero con el resultado de que cada elemento pase por la función deseada.

```
adjs <- lapply(1:nrow(V),function(i) V2adj(V[1:i,,drop=FALSE],n))
```

Por último, se hace uso de la función `qgraph.animate`, la cual realiza los pasos de la red descansando entre cada gráfico según el parámetro de sleep, dando como resultado todos los gráficos hasta llegar al último nodo, pero para mostrarlo aquí solo se va a exponer ese último gráfico.

```
library(qgraph)  
qgraph.animate(adjs,color="black",labels=FALSE,sleep=0.1, smooth = FALSE)
```



3.18.3. Anillo Centralizado, Convergencia Radial y Círculos de Escala

Por último, dentro de este apartado de redes con diseño circular, se observa para estos tres tipos de diagramas que: en el tipo de Anillo Centralizado no se han encontrado otros ejemplos más que los que explica el autor; en Convergencia Radial, los ejemplos que da el autor en su trabajo, son los que se han podido encontrar de este tipo de diagrama y no hace falta recalcar nada salvo que, en algunos casos de graph, se podría haber usado más personalización para distinguir mejor los nodos; y en Círculos de Escala, se ha explicado un gráfico similar en el Diagrama de Mapa de Árbol Circular antes en este trabajo, por lo que no hace falta que se expliquen en esta sección. Sólo un pequeño inciso: dentro del trabajo del autor se pueden distinguir dos tipos de forma de representar este diagrama, uno donde los nodos, con el tamaño de cada uno según un valor, están unidos por ramas como la mayoría de redes, y el otro una variación del mapa de árbol solo que los nodos son círculos en vez de rectángulos. El primer ejemplo no necesita de la estructura jerárquica de red de árbol como sí necesita el segundo caso.

3.18.4. Comparaciones

Esta sección al ya haberse tocado en el otro TFG no se van a exponer las comparaciones, por que el autor hizo muy buen trabajo comparando los paquetes para realizar los distintos diagramas de redes.

4. Diagramas no Realizables

No se han podido realizar los siguientes diagramas en este proyecto, debido a diferentes factores como: no encontrar a la hora de la creación del trabajo un paquete que adapte datos y recree el diagrama deseado, o no encontrar datos para realizar los diagramas, o ser imposibles de implementar en el lenguaje R.

De los siguientes diagramas ya se ha procedido a su explicación en el punto Análisis de los Diagramas 2. En este apartado se van a explicar los posibles problemas que hacen que no sea posible su implementación.

Los diagramas que no se han podido realizar son:

- **Diagrama de Cuadrícula Desordenada**
- **Diagrama de Planos Circulares**
- **Diagrama de Mapa de Esfera**

- Para el caso del diagrama de cuadrícula desordenada no se ha podido encontrar ningún paquete que realice este diagrama con una función, debido a que los paquetes que usan la cuadrícula como estructura se basa en sectores (Diagrama de Cuadrícula Circular Ordenada) y el otro tipo se basa en estructuras de datos con forma de árbol por lo que no hay paquete que cree las celdas desordenadas. Solo se pueden crear ejemplos como los de [2, pág 136] mediante los paquetes que realizan el Diagrama de Barras Radiales 3.2 , pero no se crearían las celdas sino que se realizaría un diagrama de barras radiales con barras apiladas, perdiendo su significado de cuadrícula.

- Para el caso del diagrama de planos circulares sí que se encuentran algunos paquetes que podrían representarlo, como ggplot2 en Diagrama de Mapa Circular 3.15, pero no se han encontrado datos que puedan implementar dichos paquetes resultando en el diagrama de planos.

- Y, por último, el diagrama de mapa de esfera tiene el problema antes mencionado; es imposible de implementar por ser más un diagrama artístico que un diagrama que represente datos, por lo que no se ha encontrado ningún paquete que genere dichas esferas artísticas. Estos diagramas se pueden dibujar pixel a pixel con algún paquete pero, estos perderían el hecho de que una función represente un gráfico en función de los datos, no dibujarlos.



5. Comparativa

Durante la visualización de todos los diagramas realizables, se ha realizado una comparación entre paquetes que realizan un mismo diagrama, dejando los diagramas con solo una implementación de ellos sin dicha comparación.

En esta sección se procede a comparar todos los paquetes que han contribuido en la creación de los ejemplos del trabajo. Para realizar esta comparación global se van a revisar distintas características como: personalización, complejidad, facilidad de instalación, desarrollo y cantidad de grafos que se han realizado con un mismo paquete.

Para tener una mejor comprensión de esta comparación según las características mencionadas, se procede a utilizar tablas donde se expongan diferentes variables. La primera de ellas, Cuadro1, consiste en qué paquetes se han usado para implementar los diagramas. Gracias a esta primera tabla, se observa que hay un paquete que se repite a lo largo de todos los diagramas, dejando ver su gran versatilidad que se comentará en un punto posterior.

También se observa que en la mayoría de diagramas varios paquetes pueden representar el mismo y solo unos pocos son tan específicos que solo un paquete puede implementarlos, como el mapa de árbol de polígonos voronoi. Esta capacidad de varios paquetes para implementar un diagrama, ayuda a elegir el gráfico según las necesidades que se presenten.

Otro de los patrones que se repiten son las letras gg al principio del nombre de varios paquetes. Esto ocurre porque de hecho esos paquetes como ggraph, ggiraph, ggiraphExtra son ampliaciones del paquete ggplot2, cada uno con un destino distinto; como ggraph consistiendo en una ampliación destinada a la mejor representación de diagramas de redes.

Diagramas	Paquetes	Diagramas	Paquetes
Anillos	ggplot2 circlize CMplot	Cuadrícula Circular Ordenada	circlize
Barras radiales	ggplot2 plotrix circlize	Radar	fsmb plotly plotrix ggiraphExtra
Espiral	ggplot2	Columnas Iris	ggplot2 circlize
Columnas radiales	ggplot2 plotrix cplots circlize	Líneas multiseries	ggplot2
Sectores	ggplot2 plotrix rCharts plotly ggiraphExtra	Burbujas	ggraph-ggforce packcircles-ggplot2
Rose of Nightingale	ggplot2 ggiraphExtra	Mapa árbol circular	ggraph circlepackeR
Rayos de sol	ggraph plotly webr ggiraphExtra sunburstR	Mapa árbol Voronoi	voronioTreemap
		Mapa circular	ggplot2
		Dendograma circular	R básico-ape ggraph ggplot2 circlize
		Cuerdas	circlize chorddiag
		Redes Circulares	ggraph qgraph

Cuadro 1: Paquetes utilizados para implementar los diagramas.

5.1. Versatilidad

En esta sección se discute qué cantidad de diagramas distintos puede implementar un paquete para comprobar su versatilidad. Para ello se hace uso del cuadro 1 para crear otra, mas específica, con la frecuencia de cada uno de los paquetes, Cuadro2.

Como se puede ver en el cuadro 2 y, como se dijo anteriormente, el paquete ggplot2 es el que más se utiliza, debido a que la mayoría de los diagramas son muy conocidos con el sistema de coordenadas cartesianas, como el diagrama de barras, líneas, área y variaciones de estos y al poder cambiar este sistema de coordenadas a uno polar se genera un diagrama circular partiendo del horizontal.

Otro de los paquetes que más se utilizan es circlize y, es debido a que, al contrario que ggplot2, crea directamente diagramas circulares de muchos tipos (barras, líneas, cuerdas, puntos); esto proporciona un paquete muy versatil.

Aunque la mayoría de los paquetes mostrados en el cuadro 2 son de un solo uso, en la mayoría no es debido a que sean mediocres, sino a que los diagramas en los que se usan son muy específicos y estos paquetes estan diseñados para ellos, como sunburstR y voronoiTreemap.

Paquetes	Frecuencia
chorddiag	1
circlepackeR	1
circlize	7
CMplots	1
cplots	1
fsmb	1
ggiraph	4
ggiraphExtra	4
ggplot2	11
ggraph	5

Paquetes	Frecuencia
ggforce	1
packcircles	1
plotly	3
plotrix	4
qgraph	1
R básico	2
rCharts	1
sunburstR	1
voronoiTreemap	1
webr	1

Cuadro 2: Frecuencia de paquetes usados.

5.2. Desarrollo

Ahora que se ha visualizado qué paquetes se han utilizado para los diagramas, es necesario saber como se instalan y si son paquetes obsoletos o se siguen actualizando y con qué periodicidad. Es necesario tener dicha información ya que puede indicar si el paquete se va a adaptar a futuros formatos de datos.

Lo primero es saber como se instalan los paquetes y, cómo la mayoría están disponibles en el repositorio de paquetes de la versión actual del lenguaje R, se instalan solo con la siguiente orden.

```
install.packages("paquete")
```

Pero para cuatro paquetes hay que seguir un tipo distinto de instalación, ya que 3 de ellos no estan en el repositorio de R y uno de ellos no está en el repositorio de R de la versión actual.

Primero se procede por el paquete que sí está en el repositorio de R pero no en el de la versión actual, fsmb. Para poder instalarlo hay que descargarlo con el formato .tar.gz e instalarlo mediante el instalador desde archivo del propio RStudio, si se utiliza, o con la orden.

```
install.packages("ruta-al-archivo", repos = NULL, type="source")
```

Para los otros tres paquetes que directamente no están en el repositorio de R solo están disponibles es sus respectivos github. Primero hay que instalar el paquete devtools que si esta en R y usar la función install-github.

```
require(devtools)
install_github("usuario/repositorio-paquete")
```

Lo bueno que tienen estos tres paquetes es que en el README.md de sus repositorios de gihub ya está expuesta la orden para poder instalarlos.

Una vez que ya se han instalado los paquetes, es hora de revisar lo estables que son según la cantidad de versiones y las fechas de publicación de dichas versiones. Para ello se recogen los datos de la documentación de R [41] . Esta información solo está disponible para los paquetes que tienen versiones publicadas en el repositorio de R, por lo que los tres paquetes de R restantes se va a proceder a buscar dicha información en sus repositorios de github o con la herramienta Cauldron.io [42] , la cual ofrece datos de la actividad de los repositorios de distintas empresas como github, gitlab, etc.

Con los datos ya reunidos se procede a la creación del cuadro 3, que muestra todas las versiones que ha tenido el paquete; su versión actual, cantidad de versiones, la fecha de publicación de la versión actual, la fecha de publicación de la primera versión y si el paquete es estable según los parámetros anteriores.

Para concretar la variable de estabilidad, se va a seguir la actividad que tiene el paquete en su repositorio en desarrollo, en la mayoría se situa en github, y, para visualizar la cantidad de commits que se realizan, se hace uso de la herramienta Cauldron.io con el gráfico de barras que genera con las fechas de los commits viendo su periodicidad. Por ejemplo, en el cuadro 3 algunos paquetes tienen la estabilidad en -2 significando que solo se ha realizado commits para realizar la versión del paquete, pero no se ha seguido actualizando el repositorio; para los paquetes con -1 se suele deber a que se ve desarrollo con espacio de meses entre ellos y no se ha vuelto a actualizar mas allá de la versión actual. Otros paquetes tienen la estabilidad 0 a eliminar la variable de no actualizarse después de la última versión, pero se crean espacios de meses entre desarrollos; la estabilidad a +1 equivale a las mismas variables que con la estabilidad a 0 pero con los desarrollos menos espaciados en el tiempo. Y por último, para la estabilidad +2 es necesario que el paquete se suela desarrollar con regularidad y con varias versiones lanzadas.

Paquete	Versión	Versiones	Fecha última versión	Fecha primera versión	Estabilidad
chorddiag ¹	0.1.2	3	24/07/2017	24/04/2016	-1
circlepackeR ¹	-	-	02/09/2016	02/09/2016	-2
circlize	0.4.8	38	05/09/2019	25/05/2013	+1
CMplots	3.6.0	14	25/03/2020	21/05/2015	+2
cplots	0.4.	1	05/04/2019	05/04/2019	-2
fsmb	0.7.0	29	15/12/2019	25/08/2010	0
ggiraph	0.7.0	15	31/10/2019	28/01/2016	+2
ggiraphExtra	0.2.9	2	22/07/2018	03/12/2016	-1
ggplot2	3.3.0	33	03/03/2020	09/06/2007	+2
ggraph	2.0.2	7	17/03/2020	02/02/2016	0
ggforce	0.3.1	8	20/08/2019	02/02/2016	0
packcircles	0.3.3	6	18/09/2018	26/07/2015	-1
plotly	4.9.2.1	16	04/04/2020	17/11/2015	+1
plotrix	3.7.8	87	15/04/2020	21/07/2004	+2
qgraph	1.6.5	44	21/02/2020	08/01/2011	+1
R básico	4.0.0	83	01/04/2020	01/02/2000	+2
rCharts ¹	-	-	-	-	-
sunburstR	2.1.3	11	05/11/2019	25/08/2016	0
voronoiTreemap	0.2.0	1	08/01/2019	08/01/2019	-1
webr	0.1.5	2	26/01/2020	07/05/2018	-1

Cuadro 3: Estado actual del desarrollo de los paquetes y su estabilidad representada desde -2, muy inestable, hasta +2, muy estable.

¹Datos obtenidos de sus repositorios con cauldron. No desde la documentación de R.

5.3. Usabilidad

Esta propiedad de los paquetes, es su capacidad de ser utilizado con mayor o menor facilidad para el cometido de representar los diagramas con formato circular, que se han expuesto a lo largo del trabajo. En este apartado se procede a argumentar, según distintas variables, qué paquetes son de mayor usabilidad. Las variables, para delimitar esta propiedad, son: la capacidad de personalización y la complejidad a la hora de usar el paquete. Para ponderar estas características se opta por un sistema de clasificación entre -2, para muy poca o ninguna relación con la característica, hasta +2, para la mayor relación.

- **Personalización.** Esta capacidad es el poder de retocar cualquier aspecto de los diagramas que resultan del paquete empleado.
- **Complejidad.** Esta característica es la medición de la complejidad de las funciones que emplea el paquete para crear el gráfico. Y también se refiere a la complejidad de la utilización del paquete.

Paquetes	Complejidad	Personalización	Paquetes	Complejidad	Personalización
chorddiag	-1	+1	ggforce	+1	0
circlepackeR	-1	+1	packcircles	+1	0
circlize	+2	+2	plotly	+1	+2
CMplots	0	+2	plotrix	+1	+1
cplots	-2	-1	qgraph	0	+1
fsmb	-1	+2	R básico	-2	0
ggiraph	+1	+1	rCharts	-2	-1
ggiraphExtra	+1	+1	sunburstR	0	+1
ggplot2	+1	+2	voronoiTreemap	-1	-1
ggraph	+1	+1	webr	-1	+1

Cuadro 4: Complejidad y personalización de los paquetes.

Para empezar la explicación de la calificación de los paquetes, se escogen aquellos, del cuadro 2, con más cantidad de gráficos representados, hasta los que son menos usados. El primero es **ggplot2**, que cuenta con la mejor capacidad de personalización pero, para conseguirlo, se necesitan un gran cantidad de funciones, aunque son poco complejas, por eso obtiene la calificación del cuadro 4. **Circlize** tiene la misma personalización que el anterior, solo que sus funciones, que necesitaría muchas, son más complejas. **Plotrix** se encuentra entre los que tienen mucha complejidad a la hora de crear algunos diagramas pero, en otros se necesita solo una función. Otro paquete con la misma situación es **plotly**.

Los paquetes, **ggraph**, **ggiraph**, **ggiraphExtra**, **ggforce** y **packcircles**, son complejos por su necesidad de utilizar muchas funciones pero, dichas no son difíciles, por lo que obtienen la calificación de +1. Pero en cuanto a la personalización, estos paquetes, por si solos sin la ayuda de ggplot2, no obtienen tanta puntuación como él. Aunque si utilizan sus funciones, los diagramas pasarían a tener una gran personalización, llegando a +2 en la escala. Hay algunos cuya complejidad es una suma de usar una función poco compleja, con gran cantidad de variables que aumentan su personalización en diferente medida; estos paquetes son: **chorddiag**, **CMplots**, **circlepackeR** y **webr**.

Otros paquetes son tan sencillos que a su vez provocan perdida de personalización, como en **cplots** y **rCharts**. Hay dos paquetes, **sunburstR** y **voronoiTreemap**, que son paquetes con una función sencilla (poco compleja) pero, obtienen una puntuación más alta por la necesidad del tratamiento de los datos. Y con **sunburstR** sube la calificación, en complejidad y personalización, debido a la gran cantidad de parámetros que se pueden emplear. Las funciones usadas de **R básico**, en este trabajo, consisten en una función (poco compleja) y personalización media.

6. Conclusiones

Durante todo el trabajo se han explicado e implementado los diagramas circulares que expone Manuel Lima en su libro [2] , explicando en cada caso unos sencillos pasos a realizar para crear el gráfico con los datos necesarios y forma deseada, en el lenguaje de programación R. Para ello se han usado agrupamientos de funciones, llamados paquetes, que representan graficamente los datos obtenidos con estas funciones. Se han conseguido realizar, de 21 diagramas totales, 18 (85 %) lo que es un gran porcentaje de implementación. Gran cantidad de los realizados, consisten en recrear un gráfico, conocido en su variante horizontal (barras, líneas, burbujas, etc.), con formato circular, pero para algunos de los más específicos, solo existe la variante circular.

Como se ha visto en las secciones anteriores, hay varios paquetes con gran versatilidad a la hora de implementar los gráficos circulares. `Ggplot2` y `circlize` utilizan dos formas diferentes de crear los diagramas, `ggplot2` cambiando el sistema de coordenadas, y `circlize` ya con formato circular. Los otros paquetes se dividen igualmente en estas dos maneras de implementar los gráficos. La mayoría de los que empieza, su nombre, por `gg` , al ser ampliaciones de `ggplot2`, se reutiliza su manera pero, en todos los demás paquetes se sigue la variante de `circlize`, siendo gráficos circulares sin necesidad de transformarlos, unos más difíciles que otros.

Se ha comprobado la utilidad de R a la hora de usarlo con Big Data, por su gran capacidad de obtención y manipulación de datos, y su capacidad para representarlos graficamente, gracias a los diferentes paquetes que la comunidad ha añadido al lenguaje.

Debido al carácter colaborativo de R se ayuda a la creación de diferentes caminos para, en este caso, crear diferentes diagramas en un futuro, como los que no se han podido implementar (celdas desordenadas, planos circulares y mapa de esfera).

Por último, se ha visualizado como en algunos de los diagramas, debido a su formato circular, no se pueden representar gran cantidad de variables pero si, gran cantidad de datos con ellas. Y en otros, este formato de círculos, ayuda en diferentes aspectos como: rapidez de comprensión, ayuda a la estructura de los datos, eficacia en visualizar los datos en algunos de ellos, etc. Debido a que en algunos de los diagramas existen diferentes problemas como la malinterpretación de valores, estos se vuelven gráficos más estéticos que prácticos.

Referencias

- [1] “Lenguaje de programación r.” [Online]. Available: <https://www.r-project.org/>
- [2] M. Lima, *The Book of Circles Visualizing Spheres of Knowledge*. Princeton Architectural Press, New York, U.S.A., 2017.
- [3] R. L. Harris, *Information Graphics, A Comprehensive Illustrated Reference*. Management Graphics, Atlanta Georgia, U.S.A., 1996.
- [4] K. Healy, *Data Visualization, A Practical Introduction*. Princeton University Press, New Jersey, U.S.A., 2019.
- [5] REAL ACADEMIA ESPAÑOLA, *Diccionario de la lengua española*, 23rd ed., 14/06/2020. [Online]. Available: <https://dle.rae.es>
- [6] V. Friedman, “Data visualization and infographics,” *Smashing Magazine*, Enero 2008. [Online]. Available: <https://www.smashingmagazine.com/2008/01/monday-inspiration-data-visualization-and-infographics/>
- [7] P. Simon, “The increasing importance of data visualization: An interview with zingchart’s thomas powell,” *Huffpost*, Diciembre 2017. [Online]. Available: https://www.huffpost.com/entry/the-increasing-importance_b_9837722
- [8] F. V. y Martin Wattenberg, “How to make data look sexy,” *CNN*, Abril 2011. [Online]. Available: http://edition.cnn.com/2011/OPIION/04/19/sexy.data/index.html?_s=PM:OPIION
- [9] S. Min, “Data visualization design and the art of depicting reality,” *MoMA*, Diciembre 2015. [Online]. Available: https://www.moma.org/explore/inside_out/2015/12/10/data-visualization-design-and-the-art-of-depicting-reality/
- [10] J. Urist, “From paint to pixels,” *The Atlantic*, Mayo 2015. [Online]. Available: <https://www.theatlantic.com/entertainment/archive/2015/05/the-rise-of-the-data-artist/392399/>
- [11] G. Press, “A very short history of data science,” *Forbes*, Mayo 2013. [Online]. Available: <https://www.forbes.com/sites/gilpress/2013/05/28/a-very-short-history-of-data-science/#76f33e6f55cf>
- [12] G. M. N. y. G.-P. B. Frists H. Post, *Data Visualization, The State of the Art*, 2003.
- [13] E. W. Stover, Christopher y Weisstein, “Cartesian coordinates,” *MathWorld*. [Online]. Available: <https://mathworld.wolfram.com/CartesianCoordinates.html>
- [14] ——, “Polar coordinates,” *MathWorld*. [Online]. Available: <https://mathworld.wolfram.com/PolarCoordinates.html>
- [15] E. W. Weisstein, “Circle,” *MathWorld*. [Online]. Available: <https://mathworld.wolfram.com/Circle.html>
- [16] C. C. Cárdenas, “Estado del arte en visualización de redes con r,” *Universidad de Alcalá, Escuela Politécnica Superior*, 2019.
- [17] H. Wickham, *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York, U.S.A., 2016. [Online]. Available: <https://ggplot2.tidyverse.org>
- [18] Z. Gu, L. Gu, R. Eils, M. Schlesner, and B. Brors, “circlize implements and enhances circular visualization in r,” *Bioinformatics*, vol. 30, pp. 2811–2812, 2014. [Online]. Available: <https://CRAN.R-project.org/package=circlize>
- [19] LiLin-Yin, *CMplot: Circle Manhattan Plot*, 2019, r package version 3.5.1. [Online]. Available: <https://CRAN.R-project.org/package=CMplot>
- [20] D. Gohel and P. Skintzos, *ggiraph: Make 'ggplot2' Graphics Interactive*, 2019, r package version 0.7.0. [Online]. Available: <https://CRAN.R-project.org/package=ggiraph>
- [21] L. J., “Plotrix: a package in the red light district of r,” *R-News*, vol. 6, no. 4, pp. 8–12, 2006. [Online]. Available: <https://CRAN.R-project.org/package=plotrix>

- [22] D. Xu and Y. Wang, *cplots: Plots for Circular Data*, 2019, r package version 0.4-0. [Online]. Available: <https://CRAN.R-project.org/package=cplots>
- [23] R. Vaidyanathan, *rCharts: Interactive Charts using Javascript Visualization Libraries*, 2013, r package version 0.4.5. [Online]. Available: <https://ramnathv.github.io/rCharts>
- [24] C. Sievert, *plotly for R*, 2018. [Online]. Available: <https://plotly-r.com>
- [25] K.-W. Moon, *ggiraphExtra: Make Interactive 'ggplot2'. Extension to 'ggplot2' and 'ggiraph'*, 2018, r package version 0.2.9. [Online]. Available: <https://CRAN.R-project.org/package=ggiraphExtra>
- [26] T. L. Pedersen, *ggraph: An Implementation of Grammar of Graphics for Graphs and Networks*, 2020, r package version 2.0.2. [Online]. Available: <https://CRAN.R-project.org/package=ggraph>
- [27] G. Csardi and T. Nepusz, “The igraph software package for complex network research,” *InterJournal*, vol. Complex Systems, p. 1695, 2006. [Online]. Available: <http://igraph.org>
- [28] K.-W. Moon, *webr: Data and Functions for Web-Based Analysis*, 2020, r package version 0.1.5. [Online]. Available: <https://CRAN.R-project.org/package=webr>
- [29] M. Bostock, K. Rodden, K. Warne, and K. Russell, *sunburstR: Sunburst 'Htmlwidget'*, 2019, r package version 2.1.3. [Online]. Available: <https://CRAN.R-project.org/package=sunburstR>
- [30] R. Vaidyanathan, Y. Xie, J. Allaire, J. Cheng, and K. Russell, *htmlwidgets: HTML Widgets for R*, 2019, r package version 1.5.1. [Online]. Available: <https://CRAN.R-project.org/package=htmlwidgets>
- [31] H. Wickham, M. Averick, J. Bryan, W. Chang, L. D. McGowan, R. François, G. Grolemund, A. Hayes, L. Henry, J. Hester, M. Kuhn, T. L. Pedersen, E. Miller, S. M. Bache, K. Müller, J. Ooms, D. Robinson, D. P. Seidel, V. Spinu, K. Takahashi, D. Vaughan, C. Wilke, K. Woo, and H. Yutani, “Welcome to the tidyverse,” *Journal of Open Source Software*, vol. 4, no. 43, p. 1686, 2019. [Online]. Available: <https://CRAN.R-project.org/package=tidyverse>
- [32] M. Nakazawa, *fmsb: Functions for Medical Statistics Book with some Demographic Data*, 2019, r package version 0.7.0. [Online]. Available: <https://cran.r-project.org/package=fmsb>
- [33] T. L. Pedersen, *ggforce: Accelerating 'ggplot2'*, 2019, r package version 0.3.1. [Online]. Available: <https://CRAN.R-project.org/package=ggforce>
- [34] M. Bedward, D. Eppstein, and P. Menzel, *packcircles: Circle Packing*, 2018, r package version 0.3.3. [Online]. Available: <https://CRAN.R-project.org/package=packcircles>
- [35] M. Bostock and J. Froelich, *circlepackeR: htmlwidget for Mike Bostock d3.js circle packing visualization*, 2015, r package version 0.0.0.9000. [Online]. Available: <https://github.com/jeromefroe/circlepackeR>
- [36] A. Kowarik, B. Meindl, M. Vojvodic, M. Bostock, and F. Lebeau, *voronoiTreemap: Voronoi Treemaps with Added Interactivity by Shiny*, 2019, r package version 0.2.0. [Online]. Available: <https://CRAN.R-project.org/package=voronoiTreemap>
- [37] K. Walker, *tigris: Load Census TIGER/Line Shapefiles*, 2020, r package version 0.9.2. [Online]. Available: <https://CRAN.R-project.org/package=tigris>
- [38] E. Paradis and K. Schliep, “ape 5.0: an environment for modern phylogenetics and evolutionary analyses in R,” *Bioinformatics*, vol. 35, pp. 526–528, 2018. [Online]. Available: <https://CRAN.R-project.org/package=ape>
- [39] M. Flor, *chorddiag: Interactive Chord Diagrams*, 2020, r package version 0.1.2. [Online]. Available: <http://github.com/mattflor/chorddiag/>
- [40] S. Epskamp, A. O. J. Cramer, L. J. Waldorp, V. D. Schmittmann, and D. Borsboom, “qgraph: Network visualizations of relationships in psychometric data,” *Journal of Statistical Software*, vol. 48, no. 4, pp. 1–18, 2012. [Online]. Available: <http://www.jstatsoft.org/v48/i04/>

- [41] J. Cornelissen, *RDocumentation: Integrate R with 'RDocumentation'*, 2018, r package version 0.8.2. [Online]. Available: <https://CRAN.R-project.org/package=RDocumentation>
- [42] “Level up software development analytics.” [Online]. Available: <https://cauldron.io/>