

Resumen comandos para la Ciencia de Datos

Gabriel Pinilla

28 de mayo de 2024

Índice general

1. SQL en Postgres	1
1.1. Notas previas...	1
1.2. Comandos básicos	1
1.2.1. Estructura básica de una consulta	1
1.2.2. Where vs Having	2
1.2.3. Crear una base de datos	2
1.2.4. Crear una tabla	2
1.2.5. Insertar datos a una tabla	2
1.2.6. Ver todos los datos	2
1.3. Limpieza, orden y transformación de datos	3
1.3.1. Manipulación de filas y columnas	3
1.3.2. Orden	4
1.3.3. Conversión de tipos	4
1.4. Análisis y visualización	5
1.4.1. Cálculos matemáticos básicos	5
1.4.2. Número de caracteres de cada fila	5
1.4.3. Extraer un substring de un string	6
1.4.4. Devolver valores nulos	6
1.4.5. Subconsulta	6
1.4.6. Consultas en múltiples tablas	6
2. Terminal Windows	9
2.1. Comandos comunes	9
3. Excel y Google Sheets	11
3.1. Notas previas...	11
3.2. Comandos básicos	11
3.3. Limpieza, orden y transformación de datos	11
3.3.1. Filtrado	11
3.3.2. Orden	12
3.3.3. Quitar duplicados	12

3.3.4.	Funciones lógicas	12
3.3.5.	Manipulación de strings	12
3.3.6.	Formatos de fechas	13
3.3.7.	BURSCARV y BURSCARH	13
3.3.8.	Formato de datos	14
3.3.9.	Formato condicional	14
3.4.	Análisis y visualización	14
3.4.1.	Cálculos matemáticos básicos	14
3.4.2.	Cálculos con múltiples criterios	15
3.4.3.	Tablas dinámicas	15
3.4.4.	Creación de gráficos	15
3.5.	Importación de datos	15
3.6.	Errores	15
4.	Tableau Public	17
4.1.	Creación de visualizaciones	17
4.1.1.	Importación de datos y conexiones	17
4.1.2.	Campos calculados	18
4.1.3.	Conceptos y elementos de Tableau	20
4.1.4.	Filtros	21
4.2.	Creación de visualizaciones básicas	23
4.3.	Diseño y personalización de paneles	29
4.3.1.	Funciones de agregación	30
4.3.2.	Funciones LOD	30
4.4.	Dashboards	31
4.4.1.	Creación de un dashboard	31
5.	Python	33
5.1.	Notas previas...	33
5.2.	Librerías	33
5.2.1.	Importación de datos	33
5.3.	Conceptos previos	34
5.3.1.	Manipulación de strings	34
5.3.2.	Estructuras de datos	36
5.4.	Limpieza, orden y transformación de datos	37
5.4.1.	Bibliotecas	37
5.4.2.	NumPy Arrays	38
5.4.3.	Tipos de datos de Pandas	39
5.4.4.	Importación datos	41
5.4.5.	Resúmenes de datos	41
5.4.6.	Manipulación y limpieza de datos	41

5.4.7. Cruce de tablas	47
5.5. Probabilidad y estadística	47
5.5.1. Estadística	47
5.5.2. Probabilidad	50
5.6. Estadística inferencial	53
5.6.1. Prueba de hipótesis	54
5.7. Regresión	55
5.7.1. Coeficiente de correlación	55
5.7.2. Coeficiente de determinación	56
5.7.3. Regresión lineal	56
5.7.4. Residuos	60
5.8. Gráficas con matplotlib y seaborn	61
6. R	73
6.1. Notas previas...	73
6.2. Paquetes	73
6.3. Limpieza	74
6.3.1. Resúmenes de datos	74
6.4. Orden	74
6.5. Transformación de datos	75
6.6. Cálculos matemáticos	75
6.7. Visualización	75
6.7.1. Suavizado	77
6.7.2. Estética y facetas	77
6.7.3. Guardar gráficas	77
7. Markdown	79
8. Machine Learning	81
8.1. Conceptos previos	81
8.2. Aprendizaje supervisado	82
8.2.1. Aprendizaje no supervisado	112
8.2.2. Aprendizaje reforzado	121
8.2.3. Inteligencia artificial generativa	121
8.3. Ensamblados	121
8.3.1. Ensamblados Bagging	121
8.3.2. Ensamblados Random Forest	122
9. Redes neuronales	125
9.1. Resumen	129

10. Git y GitHub	131
10.1. Git	131
10.1.1. Repositorios	131
10.1.2. .gitignore	132
10.1.3. Ramas	132
11. Glosario de términos de programación	135

Capítulo 1

SQL en Postgres

1.1. Notas previas...

1. Al finalizar una consulta se debe terminar con ;.
2. El texto se debe escribir entre comillas simples o dobles, dependiendo del programa a utilizar.
3. En SQL, el primer caracter de un string inicia con ' y se escriben entre comillas simples.
4. Para insertar comentarios se debe usar --

1.2. Comandos básicos

1.2.1. Estructura básica de una consulta

La siguiente consulta muestra la estructura básica y el orden que deben seguir los comandos:

```
1 SELECT col-sep-por-coma
2 FROM conj-de-datos.tabla
3 WHERE condicion
4 GROUP BY columnas
5 HAVING condicion
6 ORDER BY columnas DESC
7 LIMIT numero;
```

Si se requieren todas las columnas de la tabla, se debe agregar un * después del comando **SELECT**.

1.2.2. Where vs Having

En esta consulta, la cláusula **WHERE** filtra y después el **GROUP BY** agrupa:

```
1 SELECT col-sep-por-coma
2 FROM conj-de-datos.tabla
3 WHERE condicion
4 GROUP BY columnas;
```

Sin embargo, en la siguiente consulta el **GROUP BY** agrupa y luego el **HAVING** filtra:

```
1 SELECT col-sep-por-coma
2 FROM conj-de-datos.tabla
3 GROUP BY columnas
4 HAVING condicion;
```

1.2.3. Crear una base de datos

Para crear una base de datos:

```
1 CREATE DATABASE nom-base-de-datos;
```

1.2.4. Crear una tabla

Para crear una tabla:

```
1 CREATE TABLE nom-tabla (
2     col1 tipo-de-variable,
3     col2 tipo-de-variable,
4 );
```

El comando **CREATE TABLE IF NOT EXISTS** crea una tabla si es que dicha tabla no existe en la base de datos.

1.2.5. Insertar datos a una tabla

Insertar datos a una tabla ya creada:

```
1 INSERT INTO tabla (col-sep-por-coma)
2 VALUES (valores-sep-por-coma);
```

1.2.6. Ver todos los datos

Para ver la tabla completa:

```
1 SELECT *
2 FROM conj-de-datos.tabla;
```


1.3. Limpieza, orden y transformación de datos

1.3.1. Manipulación de filas y columnas

Para eliminar registros:

```
1 DELETE FROM tabla
2 WHERE nom-col = condicion;
```

Agregar una columna:

```
1 ALTER TABLE tabla
2 ADD columna tipo-de-dato;
```

Eliminar una datos de una columna:

```
1 DELETE conj-de-datos.tabla
2 WHERE condicion;
```

Eliminar una columna:

```
1 ALTER TABLE tabla
2 DROP COLUMN columna;
```

Actualizar datos:

```
1 UPDATE conj-de-datos.tabla
2 SET nom-col = valor-a-cambiar
3 WHERE condicion;
```

Eliminar espacios en blanco:

```
1 SELECT columna
2 FROM conj-de-datos.tabla
3 WHERE
4 TRIM (columna)=criterio;
```

Borrar una base de datos:

```
1 DROP DATABASE base-de-datos;
```

Borrar una tabla:

```
1 DROP TABLE tabla;
```

Evitar selecciones duplicadas:

```
1 SELECT
2 DISTINCT (columna)
3 FROM conj-de-datos.tabla
4 WHERE condicion;
```

Para contar valores distintos:

```
1 SELECT COUNT(DISTINCT(col))
2 FROM conj-de-datos.tabla ;
```

¿Cuántas veces aparece cada elemento?

```
1 SELECT columna, COUNT (*)
2 FROM conj-de-datos.tabla
3 GROUP BY columnas;
```

1.3.2. Orden

Ordenar datos:

```
1 SELECT columnas
2 FROM conj-de-datos.tabla
3 ORDER BY columna DESC;
```

En las consultas SQL, la cláusula **WHERE** filtra los datos según la condición que se le otorgue. Se pueden ordenar los datos en orden ascendente (**ASC**) o descendente (**DESC**)

1.3.3. Conversión de tipos

Para convertir los datos de un tipo a otro se usa la función **CAST**:

```
1 SELECT
2 CAST(columna AS INT) \\
3 FROM conj-de-datos.tabla; \\
```

Los tipos de datos más comunes son:

1. **SMALLINT**
2. **INTEGER** o **INT**
3. **BIGINT**
4. **SERIAL**: Número entero que se autoincrementa.
5. **NUMERIC** o **DECIMAL**
6. **VARCHAR()**: Cadena de caracteres de longitud variable.
7. **TEXT**: Cadena de caracteres de longitud variable que no tiene límite.
8. **DATETIME**
9. **DATE**: Año, mes, día (formato standar).
10. **TIME**: Hora, minuto, segundo.
11. **TIMESTAMP**: Fecha y hora.

12. **BOOLEAN**: TRUE o FALSE.

Si la consulta anterior falla, se puede usar la siguiente consulta:

```
1 SELECT
2 SAFE_CAST(columna AS TIPO-DE-DATO)
3 FROM conj-de-datos.tabla;
```

1.4. Análisis y visualización

1.4.1. Cálculos matemáticos básicos

Suma (**SUM**) y promedio (**AVG**):

```
1 SELECT col, SUM(col2), AVG(col2)
2 FROM conj-de-datos.tabla
3 GROUP BY col;
```

Mínimo y máximo: Devuelve el valor mínimo de una columna con el nombre de min-columna. Lo mismo para el máximo.

```
1 SELECT MIN(columna) AS min-columna,
2 MAX(col2) AS max-columna
3 FROM conj-de-datos.tabla;
```

Conteo de registros:

```
1 SELECT COUNT (*)
2 FROM conj-de-datos.tabla;
```

La consulta anterior contará todos los registros de la tabla. Si se le agrega la cláusula **WHERE** al final, cuenta los registros dependiendo de la condición que tenga.

1.4.2. Número de caracteres de cada fila

Para que devuelva el número de caracteres:

```
1 SELECT
2 LENGTH(col-para-comprobar-su-largo)
3 FROM conj-de-datos.tabla;
```

También se puede escribir como:

```
1 SELECT columna
2 FROM conj-de-datos.tabla
3 WHERE
4 LENGTH(columna)=criterio;
```

En algunos programas usan **LEN**.

Ejemplo:

```
1 SELECT columna1
2 FROM conj-de-datos.tabla
3 WHERE
4 LENGTH(columna1)=5;
```

Esta consulta devolverá todos los registros de la columna 1 que tengan 5 caracteres.

1.4.3. Extraer un substring de un string

```
1 SELECT columna
2 FROM conj-de-datos.tabla
3 WHERE
4 SUBSTRS (col, inicio, num-de-letras-a-extraer) + criterio;
```

1.4.4. Devolver valores nulos

```
1 SELECT columna
2 FROM conj-de-datos.tabla
3 WHERE columna IS NULL;
```

1.4.5. Subconsulta

Es una consulta dentro de otra. Se ejecutan desde la más interna hacia la más externa y van entre paréntesis:

```
1 SELECT col-sep-por-coma
2 FROM conj-de-datos.tabla
3 WHERE col > (SELECT col-sep-por-coma
4              FROM conj-de-datos.tabla);
```

1.4.6. Consultas en múltiples tablas

Para hacer consultas en múltiples tablas, se puede hacer de dos maneras. La primera es usar **SELECT** y **FROM**:

```
1 SELECT tabla1.*, tabla2.*
2 FROM tabla1, tabla 2
3 WHERE tabla1.columna = tabla2.columna;
```

La segunda es usando **JOIN**, los cuales se usan para unir dos tablas bajo la siguiente sintaxis:

```
1 SELECT *  
2 FROM conj-de-datos.tabla1  
3 INNER JOIN tabla2  
4 ON tabla1.col= tabla2.col;
```

La primera tabla o tabla izquierda siempre irá junto al **FROM**, luego irá la segunda tabla o tabla derecha. Se lee *Tabla 1 hará un cruce con la tabla 2*.

Existen 4 formas de cruzar datos:

1. **INNER JOIN**: Devuelve los registros con valores coincidentes de ambas tablas. No muestra valores nulos.



Figura 1.1: Diagrama de Venn de INNER JOIN

2. **LEFT JOIN**: Devuelve todos los registros de la tabla izquierda y solo los registros coincidentes de la tabla derecha. Tanto en el **LEFT JOIN** como el **RIGHT JOIN** van a mostrar valores nulos. Si hace el cruce con columnas que tienen valores nulos, no cruza esos valores.



Figura 1.2: Diagrama de Venn de LEFT JOIN

3. **RIGHT JOIN**: Devuelve todos los registros de la segunda tabla y solo los coincidentes de la primera tabla.



Figura 1.3: Diagrama de Venn de RIGHT JOIN

4. **FULL JOIN**: Une todos los registros de las dos tablas, sean coincidentes o no.



Figura 1.4: Diagrama de Venn de FULL JOIN

5. **CROSS JOIN**: Combina cada uno de los registros de una tabla con los registros de la otra. No se hace sobre una clave, es decir, no va el **ON**. La tabla resultante tendrá un número de filas igual al producto entre los números de filas de cada tabla, repitiéndose los datos de estas. Debido a esto, ocupa más recursos.

Capítulo 2

Terminal Windows

2.1. Comandos comunes

1. \d tabla: muestra información de una tabla y los tipos de datos de cada columna.
2. \c base-de-datos: conectarse a una base de datos.
3. \dt: da una lista de todas las bases de datos.
4. \q: salir de la terminal.
5. nom-programa version: verifica la versión instalada del programa en el sistema.
6. clear: limpia la pantalla.

Capítulo 3

Excel y Google Sheets

3.1. Notas previas...

- Algunas funciones de Excel son parecidas a las de Google Sheets.
- En Google Sheets se usan ; y en Excel ,.
- Para los rangos se usa C:C, por ejemplo: =MAX(A3:A9).

3.2. Comandos básicos

La sintaxis de una función en Excel y Google Sheets es =NOM-FUNCION(argumento1; argumento2;...), donde los argumentos son los valores que se usan como entrada para la función.

Se puede definir la hoja de cálculo con 'nom-hoja'!rango, por ejemplo: =CONTAR.SI('hoja 1'!G:G). En este ejemplo toma toda la columna G de la hoja 1.

3.3. Limpieza, orden y transformación de datos

3.3.1. Filtrado

=FILTRAR(rango; fila-o-col = filtro;""): Devolverá todos los registros del rango, y si no hay devuelve una cadena vacía (""). Se pueden crear filtros en Excel: Datos > Filtro.

3.3.2. Orden

Para ordenar datos en hojas de cálculo, se debe seleccionar Datos > Ordenar. También se puede hacer mediante la función =ORDENAR().

En el siguiente comando, FALSO (o -1 en Excel) indica que el orden es descendiente. Por otro lado, VERDADERO (o 1 en Excel) señala orden ascendente.
=ORDENAR(rango-para-ordenar;segun-columna;FALSO)

Usando =ORDENARPOR():

=ORDENARPOR(rango;col-1; FALSO; col-2;VERDADERO)

3.3.3. Quitar duplicados

Para quitar duplicados en Excel: Datos > Quitar espacios duplicados

3.3.4. Funciones lógicas

Las funciones lógicas utilizadas en Excel y Google Sheets son:

- Y(expresion1;expresion2;...)
- O(expresion1;expresion2;...)
- NO(valor-logico)
- =SI(expresion;valor-si-verdadero;valor-si-falso)
- =SI.ERROR(valor;valor-si-error)

3.3.5. Manipulación de strings

1. =ESPACIOS(valores): Elimina espacios del texto.
2. =MAYUSC(celda): Cambia de minúscula a mayúscula.
3. =MINUSC(celda): Cambia de mayúscula a minúscula.
4. =NOMPROPIO(celda): Cambia a nombre propio.
5. =CONCATENAR(): Une dos o más cadenas de texto en una celda.
6. =ENCONTRAR(): Busca una cadena de texto y devuelve su posición.
7. =DERECHA(): Devuelve el número de caracteres iniciando desde la derecha.

8. =IZQUIERDA(): Devuelve el número de caracteres iniciando desde la izquierda.
9. =EXTRAE(): Devuelve un número específico de caracteres de una posición.
10. =LARGO(): Devuelve la longitud de una cadena de texto.
11. =REEMPLAZAR(): Reemplaza una cadena de texto por otra en una celda.

3.3.6. Formatos de fechas

1. =HOY(): Devuelve la fecha actual.
2. =AHORA(): Devuelve fecha y hora actual.
3. =FECHA(): Crea una fecha. Ej: =FECHA(2023,6,10).
4. =DIAS360(): Calcula el número de días entre dos fechas en un año de 365 días.
5. =DIAS.LAB(): Calcula el número de días laborales entre dos fechas.
6. =DIAS(): Calcula el número de días entre dos fechas.
7. =MES(): Devuelve el número de mes de una fecha.
8. =AÑO(): Devuelve el año de una fecha.
9. =DIASEM(): Devuelve el número de día de la semana de 1 a 7 para una fecha.
10. =DIAS.LAB.INTL(): Calcula el número de días laborales entre dos fechas usando una definición personalizada de días laborales.
11. =FIN.MES(): Devuelve la fecha del último día del mes antes o después de un número determinado de meses.

3.3.7. BURSCARV y BURSCARH

Para relacionar tablas y buscar datos de una tabla a partir de una clave de búsqueda, se usan los comandos BUSCARV y BUSCARH. La principal diferencia entre ellos es que BUSCARV solo realiza la búsqueda en una columna, mientras que BUSCARH lo hace con las filas.

Si se escribe FALSO, la coincidencia será exacta. Si se escribe VERDADERO, la coincidencia será cercana. En Excel 1 es verdadero y 0 es falso:

=BUSCARV(valor-buscado; rango-de-busqueda; num-para-indicar-col-de-busqueda; FALSO)

3.3.8. Formato de datos

Se usa la función =CONVERTIR() se usa para transformar un número de un sistema de medición a otro.

=CONVERTIR(num-a-convertir;unidad-del-num;unidad-resultado)

3.3.9. Formato condicional

El formato condicional es una herramienta que se utiliza para identificar tendencias resaltándolas con colores. Se le añade una condición para que cambie el aspecto de la celda.

Se puede aplicar el formato condicional de la siguiente forma:

Excel: Se debe seleccionar el rango de celdas donde se quiera aplicar el formato > Inicio > Formato condicional > Reglas para resaltar celdas.

Google Sheets: Se debe seleccionar el rango de celdas donde se quiera aplicar el formato > Formato > Formato condicional.

3.4. Análisis y visualización

3.4.1. Cálculos matemáticos básicos

1. =SUMA(rango)
2. =PROMEDIO(rango)
3. =MIN(rango)
4. =MAX(rango)
5. =RESIDUO(): Da como resultado el resto cuando al dividir dos números.
6. =CONTAR.SI(rango; "criterio"): Cuenta el número de celdas que cumplen con un criterio.
7. =SUMAR.SI(rango; "criterio"): Suma los valores de un rango si cumplen con el criterio.
8. =SUMAPRODUCTO(matriz1;matriz2;...): Multiplica las matrices y muestra el resultado de la suma de esos productos.

3.4.2. Cálculos con múltiples criterios

1. =SUMAR.SI.CONJUNTO(rango-suma;rango-criterio1;criterio1;rango-criterio2;criterio2;...)
2. =CONTAR.SI.CONJUNTO(rango-criterio1;criterio1;rango-criterio2;criterio2;...)
3. =MAX.SI.CONJUNTO(rango-max;rango1;criterio1;rango2;criterio2;...)

3.4.3. Tablas dinámicas

Una tabla dinámica o Pivot Table es una tabla que resume, calcula y analiza datos para observar tendencias entre ellos.

Para crear una tabla dinámica tanto en Excel como en Google Sheets, se debe seleccionar Insertar > Tabla dinámica, luego seleccionar los datos preferentemente limpios y con columnas.

3.4.4. Creación de gráficos

Para crear un gráfico tanto en Excel como en Google Sheets, se debe seleccionar Insertar > Gráfico. Se puede personalizar cambiando los colores, dándole nombre a los ejes y variar el tipo de gráfico.

3.5. Importación de datos

Si se desea importar datos de otras hojas de cálculo:

Excel: Datos>Obtener datos> Desde archivo > Desde libro> seleccionar archivo > Importar> seleccionar en el navegador la hoja de trabajo que se quiere importar > Cargar o Transformar datos.

Google Sheets: =IMPORTRANGE(), el cual permite especificar un rango de celdas en la otra hoja de cálculo para duplicarlo en la hoja que se esté trabajando.

3.6. Errores

- #DIV/0!: Fórmula que intenta dividir por cero un valor en una celda o por una celda vacía. Se soluciona con: =SI.ERROR(valor; valor-si-hay-error).
- #ERROR!: Error que solo devuelve Google Sheets. Señala que la fórmula no se puede interpretar tal como se ingresa, es decir, hay un error en la fórmula.

- #N/A: Indica que la hoja de cálculo no puede encontrar los datos de la fórmula. Ocurren generalmente cuando se usa la función =BUSCARV().
- #NAME? o #NOMBRE?: ocurre cuando el nombre de una fórmula no se reconoce.
- #NUM!: Señala que el cálculo de una fórmula no se puede realizar según lo especificado por los datos, como por ejemplo una fecha negativa.
- #VALUE!: Señala un problema con la fórmula o con las celdas con las que hace referencia.
- #REF!: Aparece cuando las celdas de una fórmula se han eliminado.

Capítulo 4

Tableau Public

Tableau Public es una plataforma para crear y compartir visualizaciones de datos en línea.

Una visualización de datos es una representación gráfica de los mismos para comunicar información de manera efectiva. Se usa principalmente para identificar patrones que pueden ser difíciles de detectar en una tabla de datos, explorar datos, comunicar información, ayudar a tomar decisiones informadas, hacer seguimiento a procesos, identificar problemas de procesos y más.

4.1. Creación de visualizaciones

4.1.1. Importación de datos y conexiones

Para importar datos, se pueden cargar archivos desde el equipo o conectarse con Google Drive. Si se desean agregar más de un archivo, se debe seleccionar el signo + al lado de Conexiones.

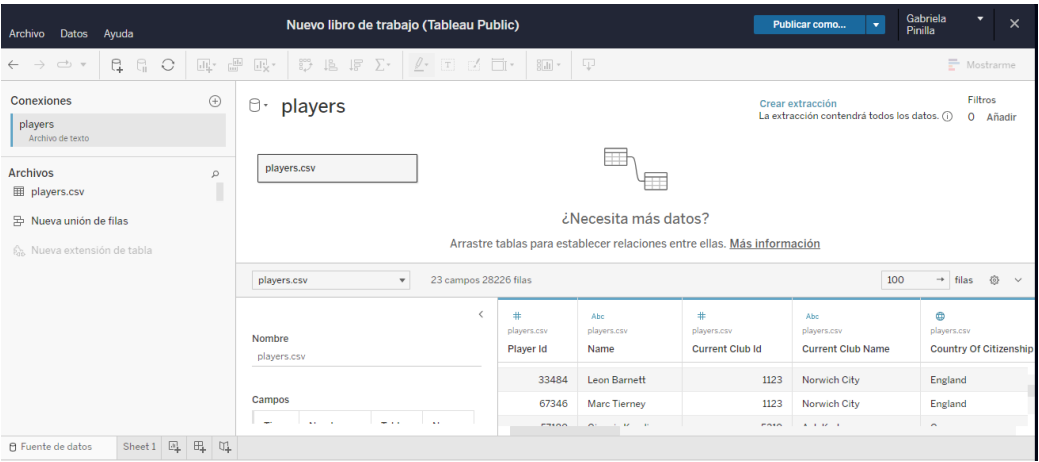


Figura 4.1: Fuente de datos

En la esquina inferior derecha se muestran los datos en una tabla. Hacia la izquierda en Conexiones, se muestran las fuentes de los datos, los cuales se pueden conectar unos con otros arrastrándolos hacia el panel que dice ¿Necesita más datos?, quedando de la siguiente forma:

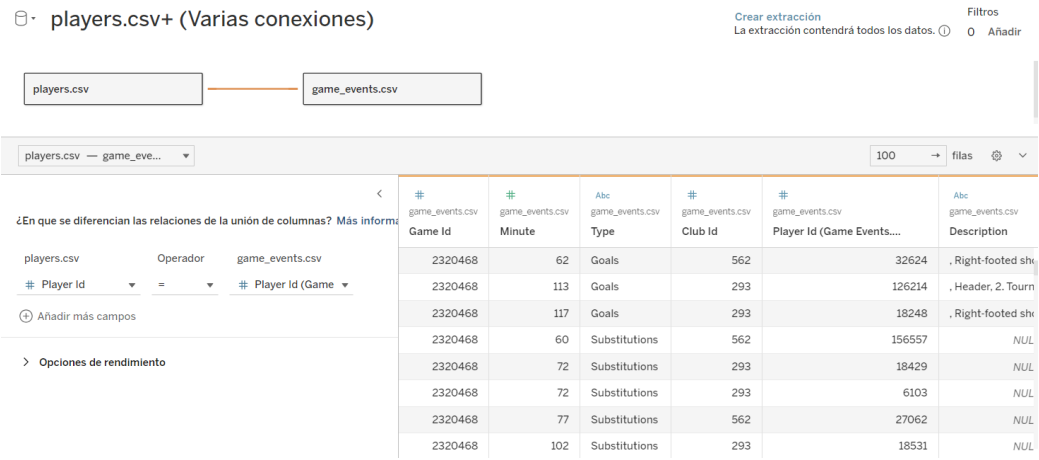


Figura 4.2: Campos en común

Tableau detecta si hay un campo en común entre ambos archivos csv la cual muestra bajo la conexión de las fuentes.

4.1.2. Campos calculados

Desde la misma hoja de Fuentes de datos se pueden realizar cálculos en las tablas. Para hacerlo, se debe seleccionar la tabla, luego la flecha de la columna en

la que se desea crear un campo calculado, > Crear > Campo calculado...

Se le puede asignar un nombre al campo calculado y escribir la fórmula para calcular los valores del campo seleccionado con los nombres de los campos entre paréntesis de corchetes. Ejemplo: [Campo 1]*5.

En Tableau existen funciones básicas que se pueden utilizar para las fórmulas de los campos calculados:

1. CONTAINS: Retorna verdadero si una cadena contiene una subcadena. Ejemplo: CONTAINS("Hola a todos", "todos"), devolverá verdadero.
2. LTRIM: Elimina los espacios en blanco del lado izquierdo de una cadena. Ejemplo: LTRIM("Hola"), devuelve "Hola".
3. TRIM: Elimina los espacios en blanco de ambos lados de una cadena. Ejemplo: TRIM("Hola "), retorna "Hola"
4. SPLIT(): Divide un campo según un separador especificado y un token. Ejemplo: SPLIT([Nombre completo], ' ', 2), en el caso de que el nombre sea Juan Soto, devolverá Soto. Si el token es 1, devolverá Juan.
5. LEN: Devuelve el largo de una cadena de texto. Ejemplo: LEN("Hola a todos"), devuelve 12.
6. LEFT: Devuelve los primeros caracteres de una cadena según una cantidad especificada. Ejemplo: LEFT("Hola a todos", 4), devuelve "Hola".
7. RIGHT: Devuelve los caracteres iniciando desde la derecha hacia la izquierda. Ejemplo: RIGHT("Hola a todos", 5), devuelve "todos".
8. LOWER: Convierte el texto en minúsculas. Ejemplo: LOWER("HOLA"), devuelve hola.
9. UPPER: Convierte el texto en mayúsculas. Ejemplo: UPPER("hola"), devuelve "HOLA".
10. MAX: Retorna el valor máximo de una expresión. Ejemplo: MAX([Ventas]), retornará el valor máximo de la columna Ventas.
11. REPLACE: Reemplaza una subcadena de texto por otra. Ejemplo: REPLACE("Hola a todos", "todos", "todas las mujeres presentes aquí"), devuelve "Hola a todas las mujeres presentes aquí".
12. COUNT: Muestra la cantidad de filas de un campo. Ejemplo: COUNT([Libros]), devolverá la cantidad de libros.

13. CEILING: Devuelve el número entero más pequeño mayor o igual que un número especificado. Ejemplo: CEILING(5,13), devuelve 5.
14. FLOOR: Devuelve el número entero más grande menor o igual que un número especificado. Ejemplo: FLOOR(5,16), devuelve 3.
15. ROUND: Redondea un número con decimales especificados. Ejemplo: ROUND("5,1634546", 2), devuelve 5,16.
16. LOG: Devuelve el logaritmo en base e de un número.
17. DATE: Devuelve la fecha. Ejemplo: DATE(01/05/2023 00:00:00), devuelve 01/05/2023.
18. DATEDIFF: Devuelve la diferencia entre dos fechas. Ejemplo: DATE-DIFF(01/09/2023, 10/09/2023, DAY), devolverá la cantidad de días entre ambas fechas.
19. DATENAME: Devuelve el nombre de un mes, día de la semana o del año.
20. DATEPART: Devuelve una parte específica de una fecha. Ejemplo: DATE-PART(01/09/2023, YEAR), devolverá 2023.

4.1.3. Conceptos y elementos de Tableau

Al momento de crear un dashboard, se debe tener en consideración ciertos términos y elementos de la plataforma.

Una hoja de trabajo es un espacio donde se puede construir la visualización de datos. Sus elementos principales son:

- Filas y columnas: aquí es donde el usuario puede arrastrar los campos de datos para definir las dimensiones y medidas de la visualización.
- Dimensiones y medidas: en el panel de datos se pueden encontrar los datos divididos por una línea donde los de arriba representan las dimensiones y los de abajo las medidas.
 - Las dimensiones son los datos categóricos. Su fin es clasificar la información. Los tipos de dimensiones son: fechas, cadenas de texto, booleanos (verdadero o falso) y rol geográfico. Ejemplo: género, estado civil.
 - Las medidas son datos de origen cuantitativos que asignan valores a las dimensiones. Los tipos de medidas son: números y rol geográfico (latitud y longitud). Ejemplo: cantidad de habitantes.

En este mismo panel, se encuentran datos de color azules, los cuales corresponden a datos discretos. Se tratan como finitos. Por el contrario, los verdes son datos continuos. Se tratan como un intervalo infinito.

- **Marcas:** en la tarjeta de marcas se configura la forma en que las dimensiones y medidas serán representadas. Sus elementos son: Tipo de marca: el cual indica la forma en la que se van a graficar los datos; Color: permite cambiar los colores de la visualización; Tamaño: varía el tamaño de la visualización; Texto: permite definir el texto de la etiqueta a mostrar en la gráfica; Detalle: permite agregar información a las marcas; Descripción emergente: permite que al pasar el mouse por una marca se muestre una descripción emergente con información.

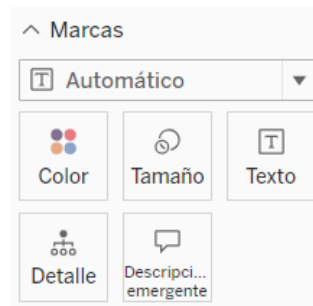


Figura 4.3: Tarjeta de marcas

- **Filtros:** aplica filtros al arrastrar los campos a este estante.
- **Páginas:** se pueden arrastrar campos para crear páginas en la visualización.

4.1.4. Filtros

1. **Filtros de extracción:** se usan para limitar el conjunto de datos desde la Fuente de datos. Al aplicarlo, se abrirá una ventana para añadir los filtros que se deseen.

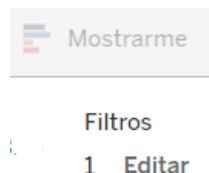


Figura 4.4: Filtro de extracción

2. Filtros de contexto: se usan para definir un contexto específico en los cálculos. En lugar de aplicar un filtro para el conjunto de datos, crea un subconjunto de datos para hacer cálculos. Para crear un filtro de contexto, se debe arrastrar un campo al estante de filtros en la hoja de trabajo, luego seleccionar la flecha de la cápsula del campo y seleccionar Añadir a contexto.



Figura 4.5: Filtro de contexto

3. Filtro de dimensión: se usan para filtrar datos de dimensiones discretas.

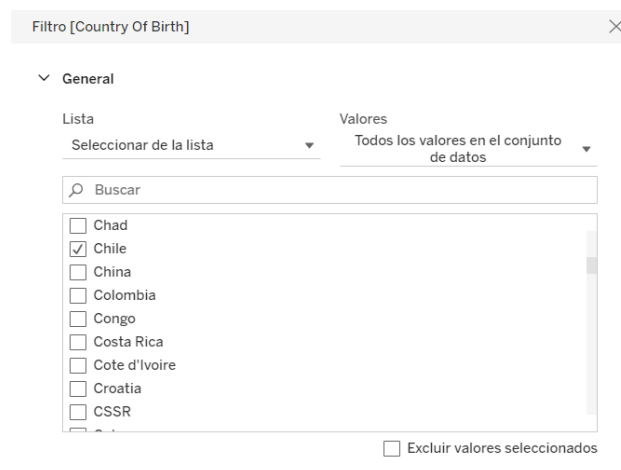


Figura 4.6: Filtro de dimensión

4. Filtro de medida: se usan para filtrar datos en función de una medida continua.

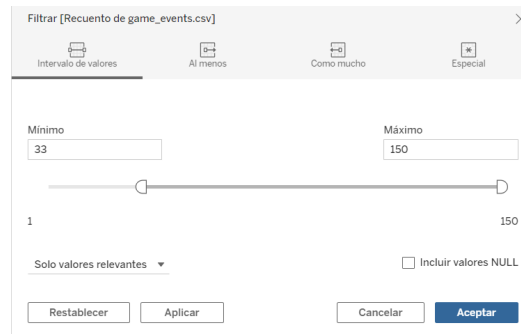


Figura 4.7: Filtro de medida

5. Filtro en gráfico: se pueden filtrar datos en un gráfico.

4.2. Creación de visualizaciones básicas

La creación de visualizaciones en Tableau se hace arrastrando las tablas de datos hacia los estantes de filas, columnas, marcas y filtros.

En el siguiente ejemplo se utilizó la base de datos del siguiente enlace de Kaggle <https://www.kaggle.com/datasets/ishikajohari/shazam-global-top-200-per-week?resource=download>, el cual muestra datos del top 200 por semana de Shazam, una aplicación para identificar música y otros desde los dispositivos móviles.

Con la base de datos de la primera semana (Week1 -22-Jul-to-28-Jul-2023.csv), se construyó la siguiente tabla:

The screenshot shows the Tableau interface. On the left, the 'Datos' pane lists available fields: Artist, Title, Nombres de medidas, Rank, Week1 - 22-Jul-to-28-Ju..., and Valores de medidas. The 'Marcas' shelf contains 'SUM(Rank)'. The 'Filas' shelf contains 'Artist' and 'Title'. The main view displays a table with the following data:

Artist	Title	Rank
A.V.G	Я плачу	56
Aaron Smith	Dancin (feat. Luvli) [Krono...	118
Adele	Set Fire to the Rain	45
Aerosmith	Dream On	98
Akon	Lonely	194
	Right Now (Na Na Na)	84
Alec Benjamin	Let Me Down Slowly	117
Aqua	Barbie Girl	120
Asake	Lonely At The Top	33
AYLIVA	Aber sie	170
Ayra Starr	Rush	42
Bad Bunny	WHERE SHE GOES	60
Bad Gyal, Tokíscha & YOUN...	Chulo pt. 2 (Pt. 2)	167
Bakar	Hell N Back	198
Beastie Boys	No Sleep Till Brooklyn	105
Becky G.	Arranca (feat. Omega)	173
Ben E. King	Stand By Me	146
Benny Benassi & The Biz	Satisfaction (Radio Edit)	100
Bibi Babydoll & Dj Brunin ..	Automotivo Bibi Fogosa	5
Billie Eilish	headline (edit)	104

Figura 4.8: Tabla de datos creada en Tableau

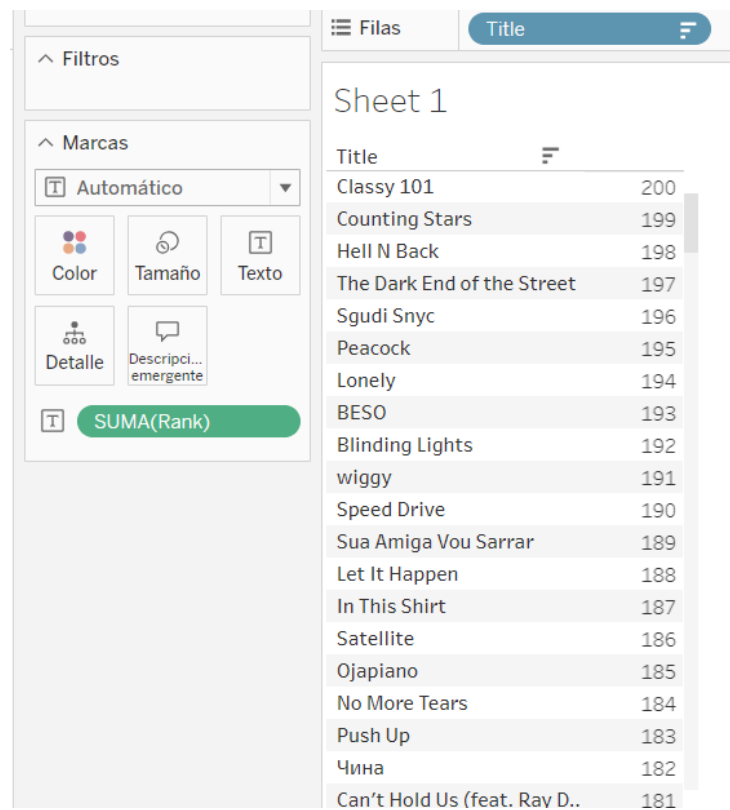
En el estante de filas se añadieron Artist (dimensión) y Title (dimensión), mientras que Rank (medida) se añadió a Texto en Marcas, mostrando en la tabla el rankin de la canción y al artista a quien le pertenece. Por defecto, Tableau añade una función de agregación, que en el caso del ejemplo es SUMA, sin embargo, como se trata de un rankin, solo corresponde a un número y no a una suma.

En la misma tabla, se pueden ordenar los datos por el rankin, como de mayor a menor, sin embargo, al tener dos dimensiones en las filas no mostrará cuál canción es menos escuchada en el rankin, sino que mostrará un conjunto de canciones con su rankin perteneciente al mismo artista.

Artist	Title	
Travis Scott	FE!N (feat. Playboi Carti)	131
	FRANCHISE (feat. Young T..	133
	I KNOW ?	88
	MELTDOWN (feat. Drake)	39
	MY EYES	83
	TELEKINESIS (feat. SZA & ..	29
OneRepublic	Counting Stars	199
	I Ain't Worried	172
	RUNAWAY	123
Lana Del Rey	Radio	138
	Say Yes To Heaven	180
	Summertime Sadness	153
The Weeknd	After Hours	99
	Blinding Lights	192
	Save Your Tears	127
Taylor Swift	cardigan	169
	Cruel Summer	64
	seven	161

Figura 4.9: Tabla de datos creada en Tableau

Para descubrir cuál es la canción menos escuchada, se debe dejar solo una dimensión en filas, como Title, y luego ordenarlas. Si se quiere saber cuál es el artista menos escuchado, se debe hacer lo mismo con la dimensión Artist.

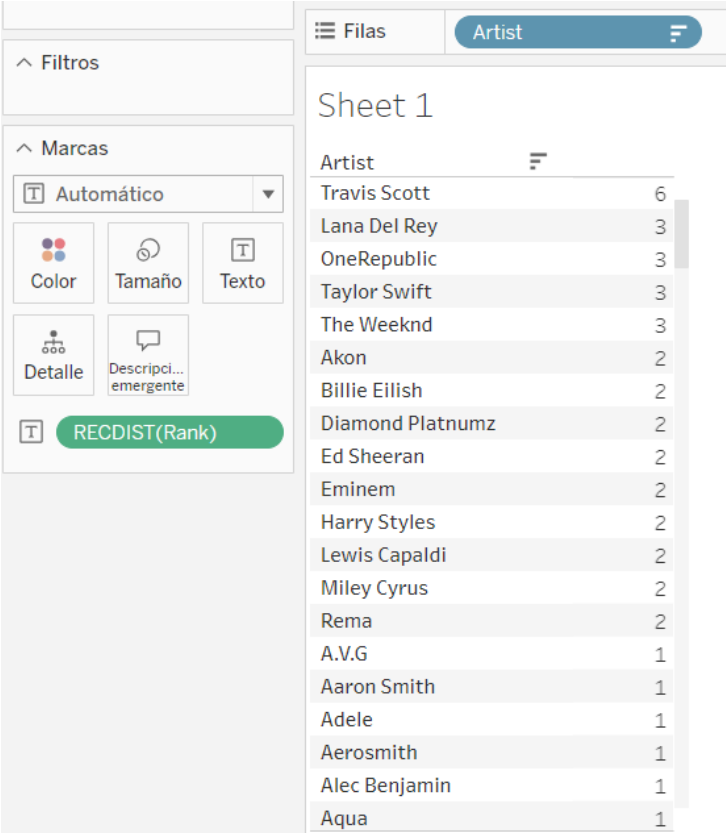


Sheet 1

Title	
Classy 101	200
Counting Stars	199
Hell N Back	198
The Dark End of the Street	197
Sgudi Snyc	196
Peacock	195
Lonely	194
BESO	193
Blinding Lights	192
wiggy	191
Speed Drive	190
Sua Amiga Vou Sarrar	189
Let It Happen	188
In This Shirt	187
Satellite	186
Ojapiano	185
No More Tears	184
Push Up	183
Чина	182
Can't Hold Us (feat. Ray D..	181

Figura 4.10: Tabla de datos creada en Tableau

También, se puede modificar la medida Rank para que cuente la cantidad de canciones con la que los artistas aparecen en el rankin.



The screenshot shows the Tableau interface. On the left, the 'Filtros' (Filters) and 'Marcas' (Marks) shelves are visible. The 'Marcas' shelf contains a green pill labeled 'RECDIST(Rank)'. The main view area displays a table titled 'Sheet 1' with the following data:

Artist	RECDIST(Rank)
Travis Scott	6
Lana Del Rey	3
OneRepublic	3
Taylor Swift	3
The Weeknd	3
Akon	2
Billie Eilish	2
Diamond Platnumz	2
Ed Sheeran	2
Eminem	2
Harry Styles	2
Lewis Capaldi	2
Miley Cyrus	2
Rema	2
A.V.G	1
Aaron Smith	1
Adele	1
Aerosmith	1
Alec Benjamin	1
Aqua	1

Figura 4.11: Tabla de datos creada en Tableau

Para el siguiente ejemplo, se tomó la base de datos mensual de las 50 principales cuentas de redes sociales de <https://www.kaggle.com/datasets/amyrmahdy/monthly-top-50-social-media-accounts-dataset>.

Con la base de datos del top 50 de Youtube (youtube_top_50_2023-07-03), se construyó la siguiente visualización:



Country	Category	AGR(subs sum)
Argentina	Music	58,1
Belarus	Entertainment	46,1
Brazil	Music	66,5
Canada	Music	71,5
Chile	Entertainment	48,0
Cyprus[a]	How-to	80,1
El Salvador	Games	46,0
India	Entertainment	526,2
	Music	437,9
	Music and film	245,0
	Education	118,3
	Film	86,6
	News	57,5
Mexico	Entertainment	46,8
Romania	Music	53,9
Russia	Entertainment	145,8
Russia- United States	Entertainment	106,0
South Korea	Music	236,0
	Education	67,9
Sweden	Entertainment	111,0

Figura 4.12: Tabla de datos creada en Tableau

Esta tabla muestra la categoría (Category) de cada país (Country) ordenado de mayor a menor por la suma de los subscriptores (subs sum) de cada categoría. Esta suma se realizó en el campo calculado de la Fuente de datos con el código `SUM([Subscribers (millions)])`.

Sin embargo, estos datos también se pueden representar de forma geoespacial por medio de un mapa.



Figura 4.13:

Ordenando los subscriptores de forma descendente, se obtiene un mapa que

muestra la categoría con más subscriptores en los países, donde el color azul indica un número más alto que el color celeste claro.

Para hacer esto, se agregó la tabla de subscriptores a la marca de Color, de esta forma Tableau puede hacer un mapa de calor indicando qué países tienen una mayor cantidad de subscriptores. La categoría y el país se arrastraron hacia la marca de Detalles, por lo que al pasar el mouse por encima de alguno de estos países mostrará el nombre del país, la categoría y la cantidad de subscriptores. La latitud y longitud en filas y columnas se generó a partir de la tabla Country.

4.3. Diseño y personalización de paneles

Al graficar, se debe tener en consideración el tipo de datos que presenta la base de datos dado que no todos los gráficos sirven para representar cualquier tipo de datos.

Para elegir un gráfico que va a representar un conjunto de datos, es importante saber qué es lo que se quiere mostrar con la visualización, conocer los tipos de gráficos y saber cuál es la audiencia a quienes se les va a presentar.

A continuación, se presentan algunos tipos de gráficos:

1. Gráfico de columnas: Compara categorías usando barras verticales.
2. Gráfico de barras: Compara categorías usando barras horizontales.
3. Gráfico circular: Proporciona las categorías en relación al total.
4. Gráfico de líneas: Muestra las tendencias a lo largo del tiempo usando líneas.
5. Gráfico de área: Muestra las tendencias a lo largo del tiempo usando un área sombreada bajo la línea.
6. Gráfico de dispersión: Relaciona dos conjuntos de datos usando puntos.
7. Gráfico de burbujas: Relaciona dos conjuntos de datos usando burbujas de diferentes tamaños para mostrar la magnitud de un tercer conjunto de datos.
8. Histograma: Distribuye los datos en un rango continuo.
9. Gráfico de radar: Expone los datos en un formato circular con categorías que se extienden radialmente.

4.3.1. Funciones de agregación

Las funciones de agregación son una funciones que operan sobre un conjunto de datos que devuelven un único valor. Se pueden obtener el promedio de un conjunto, máximo, mínimo, recuento o suma.

Las funciones de agregación más comunes en Tableau son:

- Promedio
- Suma
- Promedio o media
- Mediana
- Recuento: devuelve la cantidad de filas. Ejemplo: $\text{Rec}(a,a,b,b)=4$.
- Recuento (distintos): devuelve la cantidad de valores distintos de una columna. Ejemplo: $\text{RecDis}(a,a,b,b)=2$.
- Máximo y mínimo
- Desviación estándar y desviación estándar poblacional
- Varianza y varianza poblacional

Para agregar una función de agregación, se debe seleccionar Medida > Suma, Promedio, Mediana...

Por defecto, Tableau agrega la suma para medidas numéricas y el recuento para las no numéricas.

4.3.2. Funciones LOD

Las funciones LOD (Level of Detail) en Tableau permite realizar cálculos en un nivel de detalle específico en los datos. Son útiles cuando se requiere realizar cálculos que no se pueden lograr con las funciones de agregación.

Tableau tiene tres tipos de funciones LOD:

1. **FIXED**: permite definir un nivel de detalle específico para un cálculo. Ejemplo: Para calcular la suma de ventas por categoría: `FIXED [Categoría] : SUM([Ventas])`.
2. **INCLUDE**: permite incluir un nivel de detalle específico a un cálculo sin afectar el nivel de detalle general de la visualización. Ejemplo: Para calcular la suma de ventas por categorías e incluir la ciudad: `INCLUDE [Categoría],[Ciudad] : SUM([Ventas])`.

3. EXCLUDE: permite excluir un nivel de detalle específico a un cálculo sin afectar el nivel de detalle general de la visualización.

4.4. Dashboards

Un dashboard consiste en una pantalla que presenta una colección de visualizaciones que resumen información importante.

Uno de los propósitos de un dashboard es monitorear el rendimiento de una empresa por medio de KPI, presentando la información de forma clara para que el público pueda entenderla.

Un KPI (Key Performance Indicator o Indicador Clave de Rendimiento) es una medida que evalúa el rendimiento de una empresa o proceso con respecto al objetivo empresarial. Los KPI siguen la metodología S.M.A.R.T., un acrónimo que explica las características básicas de un objetivo SMART. Estos deben ser específicos (Specific), medibles (Measurable), alcanzables (Attainable), realistas (Realistic) y con un tiempo límite (Time).

Ejemplos de métricas que se pueden usar para los KPI:

- ROI
- Número de clientes nuevos
- Tasa de satisfacción
- Tasa de rotación del personal
- Porcentaje de clics en una página web
- Cantidad de carritos de compras abandonados
- IPC

4.4.1. Creación de un dashboard

Para crear un dashboard, se debe seleccionar el icono de Nuevo dashboard. Luego, arrastrar las vistas desde la lista de Hojas hacia el dashboard.

Capítulo 5

Python

5.1. Notas previas...

1. Python es un lenguaje de programación orientado a objetos de alto nivel.
2. Los tipos de datos en Python son int (números enteros), float (decimales), string (cadenas de texto) y booleanos.
3. Los strings se escriben entre comillas simples o dobles.
4. Los comentarios se hacen con el numeral (#). Los comentarios que contengan más de una línea se hacen en un bloque de tres comillas simples:

```
# Esto es un comentario
```

```
'''  
Esto  
es un  
comentario  
'''
```

5.2. Librerías

En ciencia de datos, las librerías más usadas son NumPy, Matplotlib y Pandas.

5.2.1. Importación de datos

Para importar las librerías, se utiliza el comando `import` y, usualmente, se le asigna un alias:

```
import libreria as alias
```

Si se quiere importar solo una función de una librería:

```
from libreria import funcion as alias
```

Ejemplo:

```
# Funcion que genera un numero aleatorio entre 0 y 10
import random
num = random.randint(0, 10)
print('El numero es ', num)
```

En Google Colaboratory, se importan los datos de drive con Pandas:

```
from google.colab import drive
drive.mount('/content/drive')
```

Luego, ir a Archivos > Drive > Copiar ruta de acceso. Se debe importar la librería Pandas con un alias:

```
import pandas as pd
```

Finalmente, para leer un archivo en formato csv:

```
nom_ds = pd.read_csv('ruta-del-archivo.csv')
```

5.3. Conceptos previos

5.3.1. Manipulación de strings

Los strings son cadenas de textos que contienen una secuencia de caracteres, como palabras. Se definen usando comillas simples o dobles y se pueden realizar operaciones con ellas.

Para interactuar con el usuario, se usan los comandos `print` y `input`. El primero imprime un mensaje en pantalla, mientras que el segundo permite que el usuario ingrese datos.

```
# Imprimir texto en pantalla
print('Hola mundo')
```

```
# Toma informacion y la puede guardar en una variable
nombre=input('Cual es tu nombre?')
```



```
# Sustituir variables en un string
print(f'Tu nombre es: {nombre}')
```

```
# Concatenacion
print('hola' + ' ' + 'mundo')
# salida: hola mundo
```

```
# Repeticion
print('Ja' * 4)
# salida: JaJaJaJa
```

Para transformar un número a string:

```
str(15)
# salida: 15
```

```
print('Quiero ' + str(15) + ' panes')
```

Y de un string a número:

```
edad = int(input('Cual es tu edad?'))
```

Operaciones con strings:

```
texto = 'Hola a todos'
# 1. lower(): convierte los caracteres a minusculas
texto_1 = texto.lower()
print(texto_1)
# salida: hola a todos
```

```
# 2. upper(): convierte los caracteres a mayusculas
texto_2 = texto.upper()
print(texto_2)
# salida: HOLA A TODOS
```

```
# 3. replace(sub,new_sub): reemplaza texto por uno nuevo
texto_3 = texto.replace('todos', 'Marta')
print(texto_3)
# salida: Hola a Marta
```

```
# 4. split(sep): divide una cadena en una lista con un
↪ separador
textonuevo = 'Hola, Josea'
```

```

lista = textonuevo.split(',')
print(lista)
# salida: ['Hola','Josefa']

# 5. strip(): elimina los espacios en blanco al inicio y
↪ final de una cadena
otrotexto = '      Hola      '
texto_4 = otrotexto.strip()
print(texto_4)
# salida: Hola

```

5.3.2. Estructuras de datos

Listas o arreglos: son secuencias de elementos ordenados por un índice, empezando por el 0.

```

# Crear una lista
lista_1 = ['elemento0', 'elemento1', 'elemento2']
print(lista_1) # Imprime la lista

# Para acceder a los elementos de la lista:
print(lista_1[0])
# salida: elemento0

print(lista_1[2]) # o print(lista_1[-1])
# salida: elemento2

# Para agregar un elemento al final de la lista
lista_1.append('elemento3')
print(lista_1)
# salida: ['elemento0','elemento1','elemento2','elemento3']

# Para obtener la longitud
longitud = len(lista_1)
print(longitud)
# salida: 4

```

Diccionarios: es una colección de par clave-valor. Su sintaxis es *{clave : valor}*:

```

diccionario_1={'nombre':'Marcelo',
               'apellido':'Soto',

```

```
'ciudad': 'Santiago',  
'edad': 38,  
'profesion': 'Presidente'}  
  
# Imprimir un valor  
print({diccionario_1['profesion']})  
# salida: Presidente  
  
# Agregar un nuevo par clave-valor  
diccionario_1['estado civil'] = 'Soltero'
```

Tuplas: las tupla no se pueden modificar después de haber sido creada.

```
tupla_1=('elemento0', 'elemento1', 'elemento3')  
  
# Para que devuelva 'elemento0'  
tupla_1[0]
```

Set: no permite tener elementos repetidos. Guardará solo los datos que no se repiten.

```
set_1 = {'banana', 'manzana', 'pera'}
```

5.4. Limpieza, orden y transformación de datos

5.4.1. Bibliotecas

Las bibliotecas que más se usan en Python para la ciencia de datos son Pandas, NumPy, Scikit-learn y Matplotlib.

NumPy se especializa en el cálculo y análisis de datos, la cual permite manejar datos de forma rápida por medio de los NumPy Arrays o ndarray, una estructura de datos que puede ser de una dimensión (vector), dos dimensiones (matriz), tres (cubo) o más.

Por lo general, se importa con el alias np:

```
import numpy as np
```

Por otro lado, la biblioteca Pandas permite manipular y analizar estructuras de datos. Se basa en NumPy y proporciona los siguientes tipos de datos:

1. Series: Estructuras de una dimensión.
2. DataFrame: Estructura de dos dimensiones.

3. Panel: Estructura de tres dimensiones.

Se importa bajo el alias de pd:

```
import pandas as pd
```

Scikit-learn es una biblioteca de aprendizaje automático que proporciona acceso a algoritmos comunes.

Matplotlib se especializa en la generación de gráficos en dos dimensiones y personalización de estos. Se pueden crear diagramas de barras, mapas de calor, histogramas y más.

Por lo general, se importa con el alias plt:

```
import matplotlib.pyplot as plt
```

5.4.2. NumPy Arrays

Un NumPy Array se puede crear a partir de una lista usando el comando `np.array`:

```
# Arreglo 1d
a = np.array([1,2,3,4,5])
```

```
# Arreglo 2d
b = np.array([[3,4,5],
              [4,3,2]])
```

O por medio de `np.arange`, el cual crea un arreglo de una dimensión con valores del 0 a n:

```
# Sintaxis
# a = np.arange(inicio,fin-1,paso)
d = np.arange(3,8,1)
```

Propiedades:

```
# Dimension del arreglo
arreglo.ndim
```

```
# Forma del arreglo (filas,columnas)
arreglo.shape
```

```
# Cantidad de datos en el arreglo
arreglo.size
```

Para buscar valores en un arreglo, Python toma cada fila como un arreglo, donde el primero tendrá índice cero. Lo mismo ocurre con sus columnas, como se muestra a continuación:

```
#      0  1  2
array([[1, 0, 3], # indice 0
      [2, 6, 9], # indice 1
      [4, 5, 8]]) # indice 2
```

De esta forma, para seleccionar el número 6, se debe insertar `arreglo[1,1]`.

Generación de números aleatorios:

```
# Crea un arreglo de numeros aleatorios entre 0 y 1
a_1 = np.random.default_rng(n)

# Genera n valores aleatorios que siguen una distribucion
↳ normal.
normal(media, desv_estandar,n)

# Genera n numeros aleatorios entre el 0 y fin-1.
integers(fin-1, size = n)

# Genera n numeros aleatorios entre el 0 y fin-1 sin
↳ repetirse.
choice(fin-1, size = n, replace = False)
```

5.4.3. Tipos de datos de Pandas

Series

Las Series se definen bajo la siguiente sintaxis:

```
Series(data=datos, index= indices, dtype='int') # tipo de
↳ dato
```

Por ejemplo:

```
# Serie creada con una lista de numeros
s = Series([1,2,3,4,5,6], dtype='int')

# Serie creada a partir de un diccionario
sdiccionario = pd.Series({'Bajo':1.40, 'Medio':1.60,
↳ 'Alto:170'})
```

DataFrame

Un DataFrame es una tabla similar a las hojas de cálculo de Excel. Posee filas y columnas. Las columnas solo puede tener datos de un solo tipo y poseen un encabezado. Cada fila se identifica por un índice único.

Es posible crear un DataFrame mediante listas de diccionarios, listas de listas, un array, etc. Se utiliza el siguiente código:

```
df = pd.DataFrame(data = datos, index = 'filas', columns =
    ↪ 'columnas', dtype = 'int')
# index=False: retorna indices numericos
```

Ejemplo de un DataFrame:

```
indice = ['mes 1', 'mes 2', 'mes 3', 'mes 4', 'mes 5']
df1 = pd.DataFrame({'mes': ['marzo', 'abril', 'mayo',
    ↪ 'junio', 'julio'], 'ventas': np.random.randint(1,100,
    ↪ 5)}, index = indice)
df1
```

Esto mostrará un resultado similar a:

	mes	ventas
mes 1	marzo	18
mes 2	abril	79
mes 3	mayo	18
mes 4	junio	21
mes 5	julio	59

Figura 5.1: DataFrame df1 en Colab

De la misma forma que en un NumPy Array, se pueden seleccionar datos específicos mediante un localizador de etiquetas con `loc` o índices con `iloc`:

```
# Busca el numero ubicado en la fila con indice 3 y columna
    ↪ con indice 1
df1.iloc[3,1]
# 21

# Busca los datos entre marzo y mayo de df1
df1.loc[:,['mes 1': 'mes 3']] # puede ser el nombre de una
    ↪ fila o columna
```

```
#      mes  ventas
# mes 1  marzo   18
# mes 2  abril   79
# mes 3  mayo    18
```

5.4.4. Importación datos

La importación de datos se hace mediante el comando `pd.read_csv` en el caso de un archivo `.csv`.

```
df = pd.read_csv('archivo.csv', index_col = 'nom_indice')
```

5.4.5. Resúmenes de datos

Por defecto, la función `head()` muestra las primeras cinco filas del DataFrame. De lo contrario, el comando `tail()` muestra las últimas cinco:

```
# Muestra los primeros 5 datos
df.head()

# Muestra los ultimos 5
df.tail()
```

Devuelve una tabla con la media, mediana, desviación estándar, máximo, mínimo y los percentiles del DataFrame:

```
df.describe()
```

Tipo de datos de las columnas del DataFrame:

```
df.info()
```

5.4.6. Manipulación y limpieza de datos

Transformación de datos

Cuando se trabaja con fechas, es posible que estos valores aparezcan de tipo string. Para transformarlos a un formato de fecha, se ocupa `pd.to_datetime()`:

```
df['fecha'] = pd.to_datetime(df['fecha'])
```

Luego, es posible acceder a los métodos de las fechas usando `.dt`:

```
# Accediendo a los años de las fechas
df['fecha'].dt.year

# Accediendo a los meses de las fechas
df['fecha'].dt.month

# Accediendo a los nombres de los meses de las fechas
df['fecha'].dt.month_name()

# Accediendo a los días de las fechas
df['fecha'].dt.day

# Convirtiendo a formato de fecha y hora especificando el
  ↪ formato
df['fecha'] = pd.to_datetime(df['fecha'], format='%d-%m-%Y')

También, es posible reemplazar datos mediante replace() y cambiar su tipo con
astype()

# Reemplazando datos tipo str
df2 = df['species'].str.replace('Adelie', 'Adelia')

# Pasando de , a .
# df['flipper_length_mm'] =
  ↪ df['flipper_length_mm'].str.replace(',', '.')

# Transformando el tipo de dato a int64
# df['flipper_length_mm'] =
  ↪ df['flipper_length_mm'].astype('int64')
```

Búsqueda de valores

Además de `iloc` y `loc`, se puede buscar valores en un DataFrame con `isin()`, en el cual se le debe indicar entre paréntesis los valores a buscar:

```
# Devuelve el df donde encuentre los valores 23 y 45 en la
  ↪ columna1
df[df.columna1.isin([23,45])]
```

Tratamiento de datos nulos

Elimina datos `np.nan`:


```
# Devuelve la cantidad de nulos en cada columna
df.isna().sum()

# Borra las columnas y filas que tengan datos faltantes
df.dropna()

# Borrando definitivamente los valores nulos
df.dropna(inplace=True)

# Creando un df nuevo sin nulos
df_nuevo = df.dropna()

# Reemplazando los valores nulos con 0
df.dropna(0)

# Reemplazando los valores nulos con 0
df.fillna(0)
```

Otra forma de buscar valores nulos en un DataFrame:

```
missing_values = df.apply(lambda x:
    ↪ sum(x.isnull()),axis=0)
```

Desglosando el código:

- `df.apply()`: aplica una función a lo largo de un eje del DataFrame. Como `axis = 0`, lo aplica a lo largo de las columnas.
- `lambda x: sum(x.isnull())`: la función lambda toma una columna x y calcula la suma de sus valores nulos, es decir, devuelve la cantidad de valores nulos de dicha columna.

Tratamiento de datos duplicados

Quitar filas duplicadas:

```
# Devuelve los valores duplicados de columna1
df[df.duplicated(subset=['columna1'])]

# Borrando duplicados de las columnas columna1 y columna2 de
↪ df
df_sin_duplicados = df.drop_duplicates(subset=['columna1',
↪ 'columna2'])
```

Es posible tratar los datos faltantes rellenando con la media o la mediana:

```
# Rellenando con la media
df['columna'].fillna(df['columna'].mean(), inplace=True)
```

Tratamiento de datos atípicos

Una forma de limpiar datos atípicos de una muestra es mediante el método del rango intercuartílico (IQR), el cual consiste en eliminar los datos que estén fuera de $Q3 + 1.5 * IQR$ o de $Q1 - 1.5 * IQR$, donde IQR se define como el cuartil 3 (Q3) menos el cuartil 1 (Q1). Los valores fuera de este rango se consideran atípicos.

En Python, esto se puede hacer de la siguiente forma:

```
# Filtrando valores atipicos
data = df['columna']
Q1 = np.percentile(data, 25)
Q3 = np.percentile(data, 75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR
var_num=var_num.iloc[np.where((data>= lower_bound) * (data <=
↪ upper_bound)))]

df.reset_index(drop=True, inplace=True)
```

Es posible visualizar los datos atípicos mediante un boxplot.

Filtros

Con Pandas se pueden filtrar datos por condiciones, columnas, filas.

Para seleccionar datos de un tipo, como strings o int64 o booleanos, se usa:

```
# Seleccionando solo columnas de tipo object del df
df.select_dtypes(['object'])
```

Filtrando por columnas:

```
# Devuelve los valores de la columna ventas de df
df['ventas']
```

```
# Devuelve las columnas mes y ventas
df[['mes', 'ventas']]
```

Filtrando por filas:

```
# POR INDICE
# Devuelve la fila con indice 0
df.iloc[0]

# Devuelve desde la primera fila hasta la 4ta
df.iloc[0:5]

# Devuelve la fila con indice 0 y la 2
df.iloc[[0,2]]

# POR ETIQUETAS
# Seleccionando las etiquetas ventas y annos
df.loc[['ventas', 'anno']]
```

Filtrando por condiciones:

```
# Devuelve los datos del df donde las ventas son mayores a
↳ 1000
df[df['ventas']>1000]

# Devuelve los datos del df donde las ventas son mayores a
↳ 1000 y el mes es igual a abril
df[(df['ventas']>1000) & (df['mes'] == 'abril')]

# Devuelve los meses que contienen la subcadena a, como
↳ marzo, abril, mayo, agosto
df[df['mes'].str.contains('a')]
```

Orden

Para ordenar los datos de en DataFrame, se usa la función `sort_values`, seguida de la columna la cual se quiere ordenar y luego si se quiere ordenar de forma creciente o decreciente.

```
df.sort_values(by = 'columna_a_ordenar', ascending=True,
↳ inplace=False)
```

Agrupación de datos

En Pandas se utiliza `groupby()` para agrupar datos de una columna para aplicarle funciones como la media, suma, etc.

	pais	seguidores	menciones
0	colombia	21	2
1	chile	16	54
2	argentina	37	5

Figura 5.2: DataFrame

El DataFrame anterior muestra cada país con su número de seguidores y menciones. Se que quiere agrupar el promedio de ambas columnas según su país, se debe aplicar el siguiente comando:

```
# Devuelve el promedio de los datos agrupados por sus paises
df.groupby('pais').mean()
```

Y devolverá la siguiente tabla:

	seguidores	menciones
pais		
argentina	37.0	5.0
chile	16.0	54.0
colombia	21.0	2.0

Figura 5.3: DataFrame agrupado según su país

Por otro lado, si se necesitan aplicar más funciones de agregación, se debe añadir el comando `agg` con un diccionario de las columnas y las funciones:

```
# Devuelve la suma de seguidores y el promedio de menciones
↳ de df agreupados por pais
df.groupby('pais').agg({'seguidores': 'sum', 'menciones':
↳ 'mean'})
```

	seguidores	menciones
pais		
argentina	37	5.0
chile	16	54.0
colombia	21	2.0

Figura 5.4: DataFrame agrupado según su país

5.4.7. Cruce de tablas

Con la librería Pandas, se pueden cruzar dos tablas mediante `merge()`, donde la primera tabla será la tabla derecha (tabla1) y la segunda la izquierda (tabla2):

```
# Cruce entre tabla1 y tabla2
df_merged = df_tabla1.merge(df_tabla2, left_on = 'col1_t1',
    ↪ right_on = 'col1_t2', how='inner', validate =
    ↪ 'many_to_one')
```

Sus parámetros son:

- `left_on`: Columnas a cruzar por la izquierda.
- `right_on`: Columnas a cruzar por la derecha.
- `on`: En el caso de que ambas columnas tengan el mismo nombre, se usa `on` y se escribe el nombre de la columna.
- `how`: Es el tipo de cruce. Puede ser `left`, `right`, `inner`, `outer` o `cross`.
- `validate`: Comprueba si el cruce es del tipo indicado. Puede ser `one_to_one`, `one_to_many`, `many_to_one`, `many_to_many`.

5.5. Probabilidad y estadística

5.5.1. Estadística

Tipos de datos en estadística

En estadística, existen diferentes tipos de datos que se clasifican en:

1. Variables cualitativas ordinales: son variables que pueden ser ordenadas o poseen un tipo de jerarquía.
2. Variables cualitativas nominales: son variables que no pueden ser ordenadas. Se utilizan para clasificar elementos.
3. Variables cuantitativas discretas: son variables que representan un conteo donde los valores forman un conjunto finito.
4. Variables cuantitativas continuas: son las variables que representan una medición en escalas continuas.

Medidas de tendencia central

Las medidas de tendencia central son medidas estadísticas que se utilizan para resumir un conjunto de valores. Las medidas más comunes son:

- Media aritmética
- Moda
- Mediana

En Python, es posible calcular las medidas de tendencia central mediante las siguientes fórmulas:

```
# Media
media = df['columna'].mean()

# Moda
moda = df['columna'].mode()

# Mediana
mediana = df['columna'].median()
```

Indicadores de posición

Mientras que los indicadores de posición son indicadores estadísticos que se utilizan para resumir los valores en uno solo o dividirlos en intervalos del mismo tamaño. Dentro de ellos se encuentran:

- Percentiles
- Cuartiles
- Quintiles
- Deciles

En Python, se pueden calcular usando `quantile` y escribiendo el percentil. Por ejemplo, si se quiere obtener un percentil en específico:

```
# Percentiles
percentil = df['columna'].quantile(0.30)
```

O los quintiles de una columna:

```
# Quintiles
quintil = df['columna'].quantile([0, 0.2, 0.4, 0.6, 0.8, 1])
```

Lo mismo con los cuartiles:

```
# Cuartiles
cuartil = df['columna'].quantile([0, 0.25, 0.50, 0.75, 1])
```

Medidas de dispersión

A diferencia de las medidas de tendencia central que entregan una idea sobre la ubicación central de los datos, las medidas de dispersión se utilizan para describir el cambio o dispersión de un conjunto. Algunas son:

- Rango
- Rango intercuartil
- Desviación media
- Desviación estándar
- Varianza
- Coeficiente de variación

En Python, se pueden obtener de la siguiente forma:

```
# Rango
rango = df['columna'].max() - df['columna'].min()

# Rango intercuartil
rango_inter = df['columna'].quantile(0.75) -
↳ df['columna'].quantile(0.25)

# Desviacion estandar
des_estandar = df['columna'].std()

# Devuelve indicadores estadisticos
df.describe()
```

5.5.2. Probabilidad

La probabilidad es una medida que determina qué tan posible es que ocurra un evento. Para calcularla, se utilizan las siguientes definiciones básicas:

1. Experimento aleatorio: Es un experimento cuyo resultado está determinado por el azar.
2. Espacio muestral (Ω): Es el conjunto de todos los resultados posibles de un experimento.
3. Evento (A, B, \dots): Es un conjunto de resultados posibles, es decir, un subconjunto del espacio muestral.
4. Probabilidad ($P(A), P(B), \dots$): Es un valor que se le asigna a un evento entre 0 y 1, donde 0 es imposible y 1 es seguro de que ocurra. Se obtiene dividiendo el número de casos favorables por el número total de resultados posibles.
5. Complemento (A^c): Es la probabilidad de que un evento no ocurra. Si $P(A)$ es la probabilidad de que ocurra un evento A , entonces $P(A^c)=1-(P(A))$ es la probabilidad de que no ocurra A .
6. Intersección ($A \cap B$): Son todos los elementos que son de un evento A y un evento B .
7. Unión ($A \cup B$): Son todos los elementos de A y todos los elementos de B .
8. Probabilidad clásica: Se basa en que todos los resultados del espacio muestral tienen la misma probabilidad.
9. Probabilidad frecuencial: Se calcula dividiendo el número de veces que ocurre un evento y el total de repeticiones del experimento.

Probabilidad condicional

La probabilidad condicional se refiere a que suceda un evento (A) sabiendo que sucede otro (B). Se expresa como:

$$P(A|B) \quad (5.1)$$

Y se lee la probabilidad de A dado B , la cual se puede calcular por medio del Teorema de Bayes:

$$P(A \cap B) = P(B) * P(A/B) \quad (5.2)$$

$$\Rightarrow P(A|B) = \frac{P(A \cap B)}{P(B)} \quad (5.3)$$

En Python, se pueden calcular probabilidades por medio de los siguientes comandos:

```
\\ conteo
conteo = df['columna1'].value_counts()['valor1']

\\ Conteo de cada registro
conteo_grupal = df['columna1'].value_counts()

\\ Conteo con condiciones
condicion1 = df['columna1'] == 'valor1'
condicion2 = df['columna2'] == 'valor2'

interseccion = (condicion1 & condicion2).sum()
union = (condicion1 | condicion2).sum()
```

Variable aleatoria

Se define variable aleatoria a una función que asigna un valor numérico a los resultados de un experimento aleatorio. Se clasifican en dos tipos:

1. Variables aleatorias continuas: pueden tomar cualquier valor dentro de un rango continuo de números reales.
2. Variables aleatorias discretas: son aquellas que pueden tomar un número finito o infinito numerable de valores posibles.

Distribución de probabilidad

La distribución de probabilidad de una variable aleatoria es una función que asigna a cada evento la probabilidad de que este ocurra. Describe cómo se distribuyen las probabilidades de los valores que puede tomar la variable.

Para una variable aleatoria continua, puede tomar cualquier valor dentro de un cierto rango. Se definen por medio de una función de probabilidad:

$$P(a \leq X \leq b) = \int_a^b f(x)dx \quad (5.4)$$

La cual representa el área debajo de la curva de $f(x)$ en el intervalo (a,b) .

Distribución binomial

La distribución binomial es una distribución de probabilidad que describe el número de veces que un evento aparece en una cantidad de experimentos. Los experimentos deben ser independientes entre sí, solo deben tener dos posibles resultados (éxito y fracaso) y la probabilidad de éxito debe ser constante para todos los experimentos.

Si se repite el experimento n veces, la probabilidad de tener k éxitos es de:

$$P(k) = \frac{n!}{k!(n-k)!} p^k (1-p)^{n-k} \quad (5.5)$$

donde:

- $P(k)$: es la probabilidad de que ocurra k éxitos.
- n : número total de experimentos.
- k : número de éxitos deseados.
- p : probabilidad de éxito en cada experimento.
- $1-p$: probabilidad de fracaso.

Por ejemplo, si se lanza una moneda al aire 10 veces y la probabilidad de tener sello es de 0,5, la probabilidad de tener 5 caras es:

$$P(5) = \frac{10!}{5!(10-5)!} 0,5^5 (1-0,5)^{10-5} \quad (5.6)$$

$$P(5) = \frac{10!}{5!(10-5)!} 0,5^5 (0,5)^5 \quad (5.7)$$

$$0,2373 \quad (5.8)$$

Distribución normal

La distribución normal es una curva que tiene forma de campana, donde la mayoría de los valores de la variable aleatoria se encuentran agrupados cerca de la media, y pocos valores en los extremos de ésta.

La distribución normal se puede utilizar para calcular la probabilidad de que una variable aleatoria continua tome un valor dentro de un cierto rango. Se caracteriza por su función de densidad de probabilidad, la cual describe la curva y la probabilidad de que un valor ocurra:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-0,5\left(\frac{x-\mu}{\sigma}\right)^2} \quad (5.9)$$

Una distribución normal posee simetría, donde la media, la mediana y la moda son iguales. Además, tiene una forma unimodal, los extremos son infinitos y posee parámetros de media y desviación estándar.

Ley de los grandes números

La ley de los grandes números establece que a medida que aumenta el número de repeticiones de un experimento (aumenta el tamaño de la muestra), la media tiende a acercarse a la media teórica de la distribución de probabilidad.

Ley débil: indica que la media muestral se acerca a la media poblacional cuando el tamaño de la muestra aumenta.

Ley fuerte: la muestra tiende a la media poblacional en la medida que el tamaño de la muestra tiende a infinito.

Teorema del límite central

El teorema del límite central establece que la distribución de la suma o media de una gran cantidad de variables aleatorias tiende a aproximarse a una distribución normal. Para esto, las variables aleatorias deben ser independientes entre sí, tener la misma distribución (media y varianza) y el tamaño de la muestra debe ser lo suficientemente grande.

5.6. Estadística inferencial

La estadística inferencial busca describir por medio de una muestra cómo puede ser la población. Posee parámetros, los cuales son indicadores que corresponden a la población, y estimadores, los que están asociados a la muestra:

Nombre	Parámetro	Estimador
Media	μ	\bar{x}
Varianza	σ^2	S^2
Desviación	σ	S
Proporción	p	\hat{p}

A través de los estadísticos de prueba, se puede relacionar un indicador estadístico con su respectivo estimador, lo cual permite hacer inferencia estadística de una muestra de datos.

Si una variable aleatoria X es tal que $X \sim N(\mu, \sigma)$ y x_1, x_2, \dots, x_n es una muestra aleatoria de X , entonces:

Si se conoce el valor de σ :

$$(\bar{x} - \mu) \frac{\sqrt{n}}{\sigma} \sim N(0, 1) \quad (5.10)$$

donde $\bar{x} - \mu$ es el error muestral definido como la diferencia entre los resultados de una muestra y los resultados que se habrían obtenido si se hubiera analizado toda la población.

De lo contrario, se puede usar el estimador S:

$$(\bar{x} - \mu) \frac{\sqrt{n}}{S} \sim t_{n-1} \quad (5.11)$$

Si X es una variable aleatoria y x_1, x_2, \dots, x_n es una muestra aleatoria de X ($n > 30$):

Si X tiene una distribución desconocida y se conoce S:

$$(\bar{x} - \mu) \frac{\sqrt{n}}{S} \approx N(0, 1) \quad (5.12)$$

Si X tiene distribución Bernouilli con probabilidad p:

$$\frac{\hat{p} - p}{\sqrt{p(1-p)}} \sim N(0, 1) \quad (5.13)$$

Cuando la muestra es menor que 30 o la desviación estándar de la población es desconocida, se utiliza la distribución t, o distribución de Student.

$$t = (\bar{X} - \mu) \frac{\sqrt{n}}{S} \quad (5.14)$$

donde t es el estadístico de prueba t que se utiliza para evaluar la diferencia entre la media muestral y la media poblacional.

5.6.1. Prueba de hipótesis

Una prueba de hipótesis se utiliza para evaluar si una afirmación sobre una población es compatible con la evidencia observada. Para realizar una prueba de hipótesis se siguen los siguientes pasos:

1. Formular la hipótesis nula (H_0), afirmación inicial que se somete a prueba, y alternativa (H_1), afirmación alternativa a la nula.
2. Establecer el nivel de significancia (α), el cual se refiere a la probabilidad de rechazar la hipótesis nula cuando es verdadera. Se usan valores como 0,05 o 0,01. Con esto, se puede calcular el nivel de confianza definido como $1 - \alpha$.

3. Elegir y calcular el estadístico de prueba dependiendo de los valores conocidos. En este paso, se calcula los valores p .
4. Determinación de la región crítica, la cual se refiere al área de la distribución de probabilidad donde se encuentran los valores que llevarían a rechazar la hipótesis nula.
5. Tomar una decisión comparando el valor obtenido del estadístico de prueba y los valores de la región crítica, es decir, el valor p y la significancia.

Para decidir si se rechaza H_0 se usa el siguiente criterio:

- Si valor $p >$ significancia, no se rechaza H_0 .
- Si valor $p \leq$ significancia, se rechaza H_0 .

Sin embargo, cuando el estadístico de prueba cae dentro de la región crítica se rechaza H_0 , sugiriendo que hay evidencia suficiente para respaldar H_1 .

5.7. Regresión

La regresión es un proceso estadístico que se usa para modelar la relación entre una variable dependiente (y) y una o dos independientes (x).

5.7.1. Coeficiente de correlación

La correlación es una medida que evalúa la relación entre dos o más variables.

Si los puntos parecen seguir una recta en un gráfico, podría corresponder a una correlación lineal, la cual se cuantifica mediante el coeficiente de correlación de Pearson. Este coeficiente toma valores entre -1 y 1, donde:

- -1: indica una correlación negativa perfecta. Señala que una variable aumenta mientras la otra disminuye.
- 0: indica que no hay relación lineal entre las variables.
- 1: indica una correlación positiva perfecta. Señala que ambas variables aumentan o disminuyen.

Un coeficiente de correlación cercano a 1 o -1 indicará que el modelo se ajusta a los datos.

Si la relación entre las variables no es lineal, se debe utilizar otro tipo de correlación, como el coeficiente de correlación de Spearman o el de Kendall.

5.7.2. Coeficiente de determinación

El coeficiente de correlación R^2 es una medida estadística que dice qué tanto se ajusta el modelo a los datos. En otras palabras, muestra qué tan cerca están los datos de la línea de tendencia. su valor va de 0 a 100 %, donde:

- 0 % indica que el modelo no explica la variabilidad de los datos.
- 100 % indica que el modelo explica toda la variabilidad de los datos.

La variabilidad, en estadística, se refiere a la distancia entre los datos y la media del conjunto de datos. Se puede cuantificar mediante la varianza, desviación estándar y el rango intercuartil. Una alta variabilidad en los datos significa que hay una alta dispersión en ellos, donde la diferencia entre los datos y la media del conjunto es mayor. Por el contrario, una baja variabilidad indica que los datos están más agrupados alrededor de la media del conjunto y la dispersión es menor.

5.7.3. Regresión lineal

La regresión lineal se utiliza para encontrar la mejor línea recta que se ajusta a los datos para luego predecir los valores de la variable dependiente en función de la variable independiente. Mediante esta regresión, se busca predecir un número.

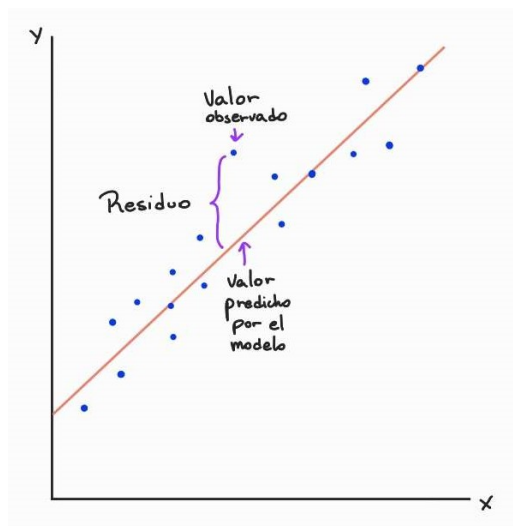


Figura 5.5: Gráfico

Se modela mediante la ecuación de la recta, la cual tiene la siguiente forma:

$$y = b_0 + b_1 * x \pm c \quad (5.15)$$

donde:

- y : variable dependiente que se intenta predecir. Variable target.
- x : variable independiente.
- b_0 : intersección de la recta con el eje Y, llamada intercepto.
- b_1 : pendiente de la recta.
- c : término de error.

El objetivo de la regresión lineal es encontrar los valores de b_0 y b_1 que minimicen la suma de los cuadrados de errores, más conocido como el método de los mínimos cuadrados.

Otros parámetro de la regresión lineal son los valores p de los coeficientes. Un valor p bajo indica que la variable independiente tiene un efecto significativo sobre la variable dependiente.

En Python, se puede realizar una regresión lineal simple y calcular sus parámetros por medio del siguiente código:

```
from scipy import stats

# Antes de realizar la regresion, se deben seleccionar las
#  ↪  columnas
datos= df[["variable_independiente", "variable_target"]]

# Regresion lineal simple
sns.lmplot(x="variable_independiente", y="variable_target",
#  ↪  data=datos)
plt.title("Regresion lineal entre variable_independiente y
#  ↪  variable_target")
plt.xlabel("variable_independiente")
plt.ylabel("variable_target")

# Calculando los parametros para evaluar el modelo
slope, intercept, r_value, p_value, std_err =
#  ↪  stats.linregress(datos["variable_independiente"],
#  ↪  datos["variable_target"])

# Mostrando el grafico completo
plt.show()

# Imprimiendo los parametros
print(f"Pendiente: {slope}")
```

```
print(f"Intercepto: {intercept}")
print(f"Error estandar: {std_err}")
print(f"Coef. de correlacion: {r_value}")
```

Es posible hacer un modelo de regresión lineal con Statsmodels mediante la función `OLS()`, la cual busca minimizar los cuadrados de los residuos.

```
import statsmodels.api as sm

# Definiendo las variables predictoras (X) y la variable de
↪ respuesta (y)
X = df["variable_independiente"]
y = df["variable_target"]

# Agregando una constante al predictor (intercepto)
X = sm.add_constant(X)

# Creando el modelo de regresion lineal
model = sm.OLS(y, X).fit()

# Obteniendo los resultados del modelo
results = model.summary()

# Imprimiendo los resultados
print(results)

# Graficando la regresion lineal
plt.scatter(df["variable_independiente"],
↪ df["variable_target"], label="Datos")
plt.plot(df["variable_independiente"], model.predict(X),
↪ color='red', label="Regresion Lineal")
plt.title("Regresion lineal entre variable_independiente y
↪ variable target")
plt.xlabel("variable_independiente")
plt.ylabel("variable_target")
plt.legend()
plt.show()
```

Los resultados de esta regresión entrega los siguientes estadísticos:

1. r-squared: r cuadrado. Varía entre 0 y 1, y corresponde a la variabilidad de la variable dependiente. Un r cuadrado alto indica que el modelo se ajusta bien a los datos.

2. Adj. r-squared: r cuadrado ajustado. Penaliza (disminuye con respecto al r cuadrado) por añadir variables que no contribuyen significativamente a explicar la variabilidad de la variable dependiente.
3. F-statistic: F estadístico. Un alto valor del estadístico F con un valor p bajo implica que al menos una de las variables independientes tiene un efecto significativo en la variable dependiente.
4. coef: coeficientes. Representan las pendientes de las variables independientes.

Nota: La variabilidad describe la dispersión de los puntos alrededor de una medida central. La significancia se refiere a la probabilidad de que los resultados de una prueba estadística sean improbables bajo la suposición de que la hipótesis nula (H_0) sea cierta.

Regresión lineal múltiple

Para aplicar Statsmodels y obtener un modelo de regresión para dos o más variables independientes, se transforma una variable categórica en numérica usando `get.dummies`:

```
# Paso 1: Definiendo variables predictoras (X) y variable de
↳ respuesta (y)
X = df[['variable_independiente', 'variable_categorica']] #
↳ Largo de la aleta y sexo como variables independientes
y = df['variable_target'] # Peso como variable dependiente

# Codificando la variable categorica usando get dummies
X_encoded = pd.get_dummies(X,
↳ columns=['variable_categorica'],
↳ prefix=['variable_categorica'])

# Paso 2: Agregando una constante (intercepto) a las
↳ variables predictoras
X_encoded = sm.add_constant(X_encoded)

# Paso 3: Creando el modelo de regresion lineal
model = sm.OLS(y, X_encoded).fit()

# Paso 4: Obteniendo resultados del modelo
results = model.summary()
```

```

predictions = model.predict(X_encoded)

# Calculando el Error Cuadratico Medio (MSE)
mse = ((y - predictions) ** 2).mean()

# Calculando el Error Cuadratico Medio Explicado (EMSE)
emse = np.sqrt(mse)

# Calculando el valor R cuadrado del modelo
r2 = model.rsquared

print(results)
print(f"MSE: {mse:.2f}")
print(f"EMSE: {emse:.2f}")
print(f"R cuadrado: {r2:.2f}")

```

5.7.4. Residuos

Los residuos son la diferencia entre los valores y los valores predichos por el modelo. Evalúan la calidad del ajuste del modelo. Si los residuos son menores, el modelo es más preciso.

Para graficar los residuos, se utiliza el siguiente código:

```

residuals = datos["variable_target"] - (slope *
↪  datos["variable_independiente"] + intercept)

# Mostrando el grafico de los residuos
plt.figure()
plt.scatter(datos["variable_independiente"], residuals)
plt.axhline(y=0, color='r', linestyle='--')
plt.title("Grafico de Residuos")
plt.xlabel("variable_independiente")
plt.ylabel("Residuos")
plt.show()

```

El gráfico de residuos puede darse de las siguientes formas:

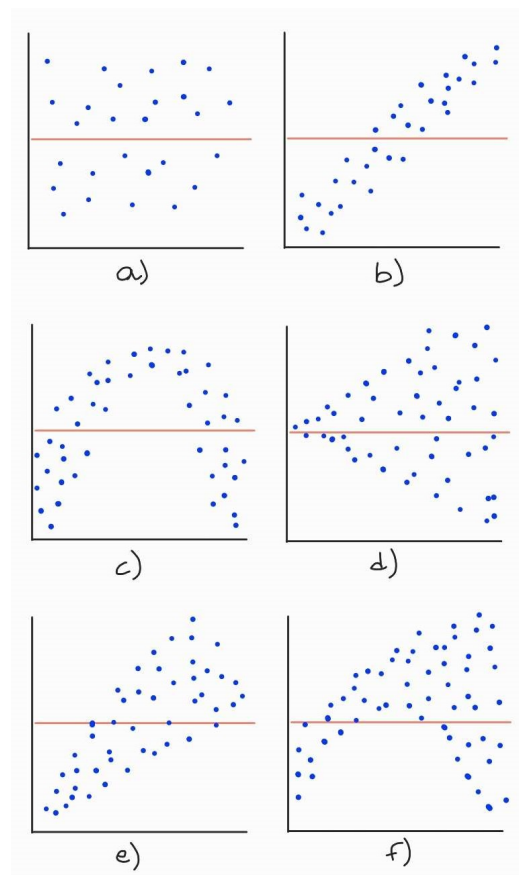


Figura 5.6: Gráficas de residuos

De las figuras anteriores, la figura a presenta un patrón de residuos al azar donde no hay tendencias. La b y c presentan tendencias, lo que podría indicar que la relación entre las variables no es la indicada. La d, e y f tienen una dispersión irregular, donde la varianza de los residuos aumenta conforme aumentan los valores, lo cual indica que aumenta la variabilidad y su media.

5.8. Gráficas con matplotlib y seaborn

Para crear visualizaciones en Python, por lo general se usan las librerías matplotlib y seaborn. Se deben seguir los siguientes pasos para poder hacer un gráfico:

1. Importar las bibliotecas.
2. Crear la figura.
3. Agregar los datos.

4. Agregar ejes.

Un ejemplo de un gráfico de líneas en matplotlib:

```
# Importando la libreria
import matplotlib.pyplot as plt

# Datos
x = [1, 2, 3, 4, 5]
y = [2, 4, 6, 12, 20]

# Creando el grafico de lineas
plt.plot(x, y, marker='o', label = 'Linea')

# Agregando los nombres de los ejes
plt.xlabel('Eje X') # Eje x
plt.ylabel('Eje Y') # Eje y
plt.title('Grafico de Ejemplo') # Titulo

# Para agregar una leyenda
plt.legend()

# Mostrando el grafico
plt.show()
```

Se verá de la siguiente forma:

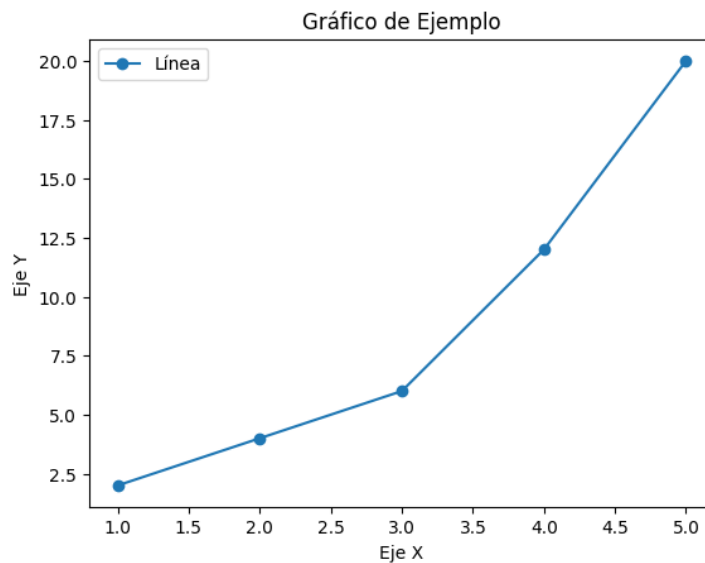


Figura 5.7: Gráfica de ejemplo

Parámetros:

Agrega cuadrícula de fondo

```
plt.grid(True)
```

marker: da opciones de la forma de los puntos

```
plt.plot(x, y, marker='o', label = 'Linea')
```

```
plt.plot(x, y, marker='-', label = 'Linea')
```

markersize: grosor de los puntos

```
plt.plot(x, y, marker='o', markersize=6, label = 'Linea')
```

linestyle: configura el tipo de linea

```
plt.plot(x, y, marker='o', linestyle='-' label = 'Linea')
```

color: agrega color a la linea

```
plt.plot(x, y, color='b', label = 'Linea') # linea azul
```

```
plt.plot(x, y, color='r', label = 'Linea') # linea roja
```

marker también posee:

- .: punto.

- .: píxel.

- \wedge : triángulo hacia arriba.
- v: triángulo hacia abajo.
- s: cuadrado.
- +: cruz.
- *: estrella.
- x: cruz diagonal.
- D: diamante.
- p: pentágono.

Es posible configurar el tamaño de la figura con:

```
# Configurando el tamaño a 8 x 6 pulgadas  
plt.figure(figsize=(8,6))
```

Gráfico de líneas

En seaborn, un gráfico de líneas sería de la siguiente forma:

```
import seaborn as sns  
import matplotlib.pyplot as plt  
  
# Datos  
x = [1, 2, 3, 4, 5]  
y = [2, 4, 6, 12, 20]  
  
# Creando el grafico de linea  
sns.lineplot(x=x, y=y, label='Linea')  
  
# Agregando etiquetas y titulo  
plt.xlabel('Eje X')  
plt.ylabel('Eje Y')  
plt.title('Grafico de linea')  
  
# Mostrando el grafico  
plt.show()
```

Mostrará el siguiente gráfico:

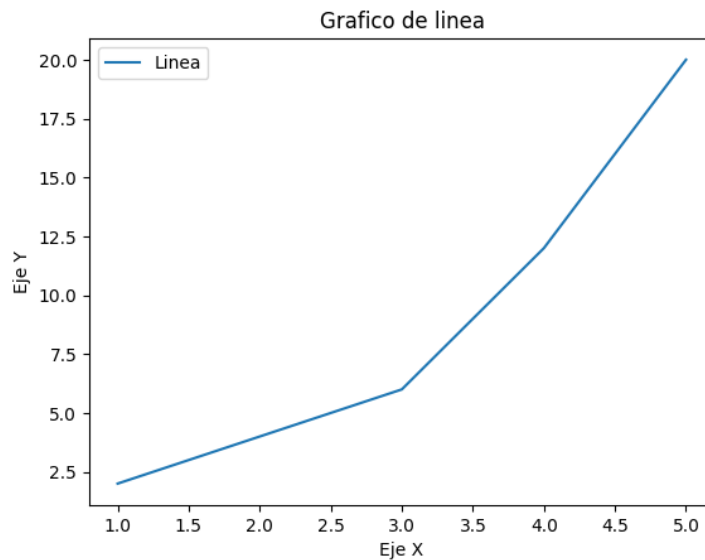


Figura 5.8: Gráfico de líneas con sns

Histograma

Con matplotlib, se puede construir un histograma de la siguiente forma:

```
import matplotlib.pyplot as plt

# Creando un histograma con 10 bins de color azul
↳ transparente
plt.hist(x, color='blue', alpha=.4, bins=10, label='x')
# Agregando titulo
plt.title('Histograma de la variable x')
# Leyenda
plt.legend()
```

Para construir un histograma en seaborn:

```
import seaborn as sns
import matplotlib.pyplot as plt

# Construyendo el histograma
sns.histplot(data=df, bins=10, color='skyblue')
# Agregando etiquetas y titulo
```

```
plt.title('Histograma con sns.histplot()')
plt.xlabel('Valores')
plt.ylabel('Frecuencia')
# Mostrando el grafico
plt.show()
```

Donde:

- data: datos.
- bins: cantidad de barras.

Con displot:

```
import seaborn as sns
import matplotlib.pyplot as plt

sns.displot(data, color='skyblue', bins=10)
# Agregando etiquetas y titulo
plt.title('Histograma con sns.displot()')
plt.xlabel('Valores')
plt.ylabel('Frecuencia')
# Mostrando el grafico
plt.show()
```

displot permite utilizar otros parámetros, como por ejemplo permite construir dos histogramas en uno:

```
import seaborn as sns
import matplotlib.pyplot as plt

# Creando un objeto en el cual grafique los datos de data
↳ correspondientes a mean_perimeter distinguiendo por el
↳ valor de diagnosis
ax = sns.displot(data=df, x='mean_perimeter',
↳ hue='diagnosis', fill=True)

# Agregando etiquetas y titulo
ax.set(xlabel='Media del Perimetro por
↳ Diagnostico', ylabel='Densidad', title='Perimetro v/s
↳ densidad')
# Mostrando el grafico
plt.show()
```


Distribución normal

Para crear una gráfica que represente una distribución normal, primero se debe crear un arreglo de valores equidistantes entre el valor mínimo y el máximo de los datos con `linspace`. Luego, se crea una función llamada `pdf` que calcula el valor de la función de distribución de probabilidad normal para finalmente construir el histograma.

```
from scipy.stats import norm
import seaborn as sns
import matplotlib.pyplot as plt

# Calculando el maximo y el minimo de los datos
minimo = df['columna'].min()
maximo = df['columna'].max()
mu = df['columna'].mean()
sigma = df['columna'].std()

# Arreglo de valores equidistantes
x = np.linspace(minimo, maximo)

# Crando la funcion pdf
# 'sigma' funciona como 'escala'
pdf = norm.pdf(x, loc=mu, scale=sigma)

# Construyendo un histograma
# 'density=True' construye el histograma haciendo que la suma
→ de todas las areas de las barras sea igual a 1, para
→ representar las probabilidades
plt.hist(x, bins=100, density=True, alpha=0.5,
→ label='Muestra') #alpha=0.5 da un 50% de transparencia

# Graficando la distribucion normal
plt.plot(x, pdf, color='red', label='PDF')
# Agregando etiquetas y titulo
plt.xlabel('x')
plt.ylabel('Densidad de probabilidad normalizada')
plt.title('Distribucion')
# Leyenda
plt.legend()
# Mostrando el grafico
```

```
plt.show()
```

La gráfica construirá una función representada por una línea roja sobre un histograma, resultando en una distribución normal:

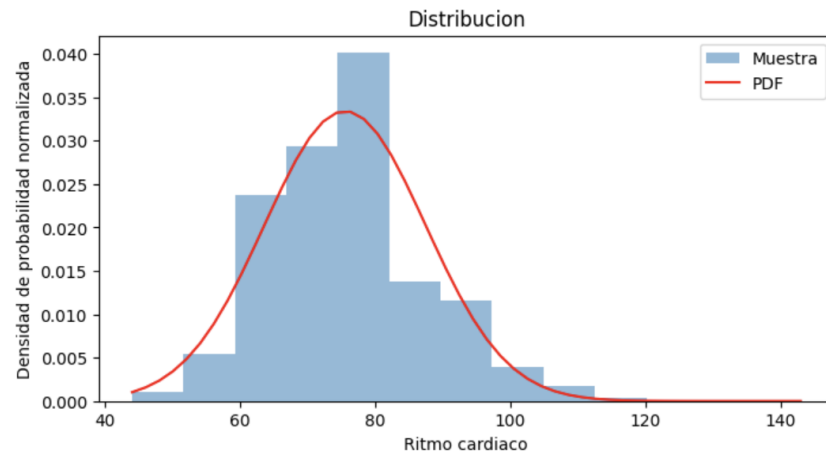


Figura 5.9: Gráfico de una distribución normal

También, se puede hacer con la librería seaborn:

```
# Histograma
sns.histplot(x='x', data=df, kde=True)
plt.show()
```

Boxplot

Anteriormente, se mencionó que se pueden visualizar de mejor manera los datos atípicos de una muestra con un boxplot.

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

# df
data = {
    'Grupo': ['A', 'A', 'B', 'B', 'B', 'C', 'C', 'C',
              ↪ 'C'],
    'Valor': [5, 8, 12, 15, 20, 7, 10, 14, 18]
}
```

```
df = pd.DataFrame(data)

# Creaando un boxplot
sns.boxplot(x='Grupo', y='Valor', data=df)

# Agregando etiquetas y titulo
plt.xlabel('Grupo')
plt.ylabel('Valor')
plt.title('Boxplot por Grupo')

# Mostrando el grafico
plt.show()
```

Se verá de la siguiente forma:

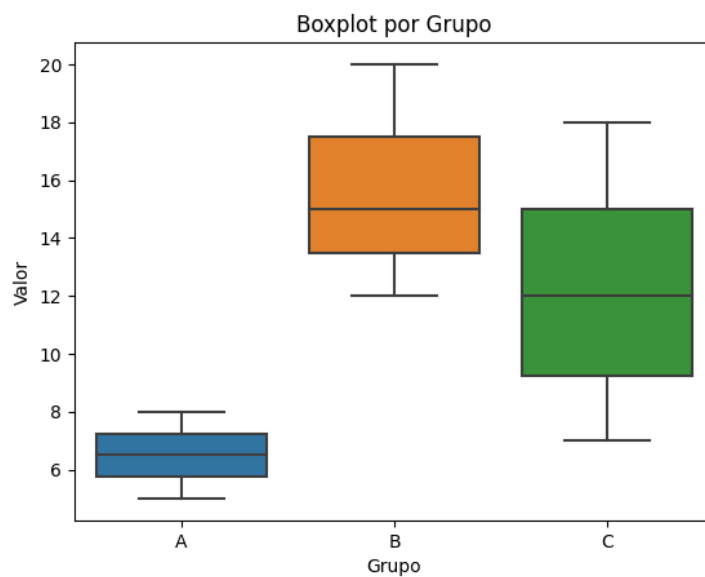


Figura 5.10: Boxplot

Donde un boxplot con valores atípicos se vería de la siguiente forma:

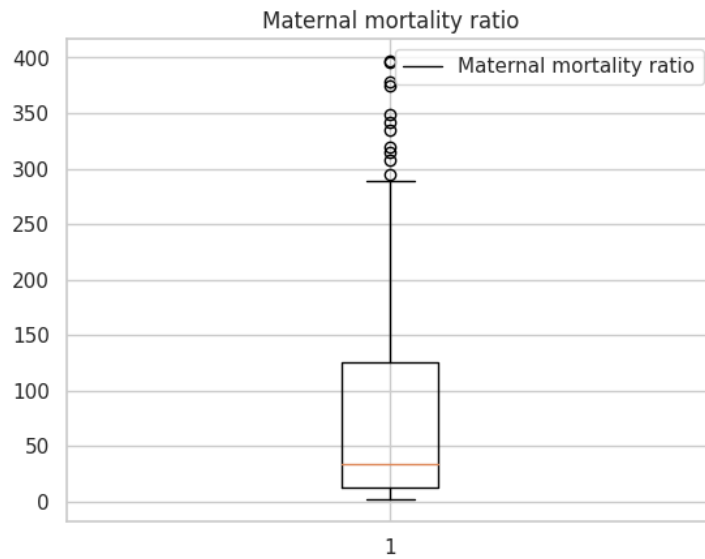


Figura 5.11: Boxplot con valores atípicos

Gráfico de barras

```
import seaborn as sns
import matplotlib.pyplot as plt

# Creando el grafico de barras
sns.barplot(x=grouped_data.index, y=grouped_data.values)

# Agregando etiquetas y titulo
plt.xlabel('x')
plt.ylabel('y')
plt.title('Grafica de barras')

# Etiquetas a las barras
custom_labels = ['Barra 1', 'Barra 2']
plt.xticks(range(len(custom_labels)), custom_labels)
# Mostrando el grafico
plt.show()
```

Scatterplot

```
import seaborn as sns
import matplotlib.pyplot as plt
```

```
sns.set(style='whitegrid')
# Creando el grafico
sns.scatter(x='x', y='y', hue='columna', data=df,
↪ palette='Set1')
plt.xlabel('x')
plt.ylabel('y')
plt.title('Grafica scatterplot')
# Mostrando el grafico
plt.show()
```

El parámetro `hue` especifica que se utilizará una columna del DataFrame para colorear los puntos de la gráfica. `palette='Set1'` especifica la paleta de colores que se utilizará para colorearla.

Se le puede añadir una línea recta para relacionar dos variable añadiendo el siguiente código:

```
# Linea recta
plt.plot(df['x'], df['y'], color='red')
```


Capítulo 6

R

6.1. Notas previas...

- 1.

6.2. Paquetes

Los paquetes en R incluyen funciones, documentación sobre estas mismas, muestras de conjuntos de datos y pruebas para verificar el código. R incluye un conjunto de paquetes denominados Base R.

Existe una colección de paquetes llamado Tidyverse que contiene los siguientes paquetes:

1. ggplot2: se usa para visualizar datos y crear visualizaciones.
2. tidyr: se usa para la limpieza de datos.
3. readr: se usa para importar datos.
4. dplyr: ofrece funciones que ayudan a la manipulación de datos.

Otros paquetes:

1. here
2. skimr: facilita el resumen de los datos.
3. janitor

Para ver los paquetes instalados:

```
installed.packages()
```

Para cargar paquetes:

```
library(paquete)
```

6.3. Limpieza

6.3.1. Resúmenes de datos

Funciones para obtener resúmenes de los marcos de datos:

```
#Muestra el tipo de datos
str(df)
#Devuelve el min,max,media,mediana,1er y 3r cuartil. skimr
summary(df)
# skimr
skim_without_charts(df)
# Skimr
skim(df)
# Devuelve el numero de filas y columnas.
glimpse(df)
# Muestra solo las primeras seis filas.
head(df)
```

6.4. Orden

```
# Se usa para elegir qué variable se quiere ordenar.
arrange()
# Para ordenar de forma descendente.
arrange(-)
# Agrupa.
group_by()
# Filtra los datos.
filter()
# Excluye valores NA.
drop_na()
```


6.5. Transformación de datos

```
# Divide datos en columnas separadas.
separate(df, columna-a-separar, into=c('nom-col-nueva',
  ↪ 'nom-col-nueva-2'), sep= 'tipo-de-separador')
# Permite fusionar columnas entre sí.
unite(df, 'nom-col', col-a-combinar, col-a-combinar-2,
  ↪ sep='separador')
# Se puede usar para añadir columnas con cálculos.
mutate()
```

6.6. Cálculos matemáticos

```
# Media aritmética.
mean()
# Valor máximo.
max()
# Desviación estándar.
sd()
# Correlación.
cor(x=var-x, y=var-y, method='pearson o kendall o spearman')
```

6.7. Visualización

Se utiliza la función ggplot para crear gráficas:

```
ggplot((data=df) + geom_función(mapping=aes(x=var-x,
  ↪ y=var-y, color=var1, shape=var2, size=var3,
  ↪ alpha=var4))), (color= color-para-todos-los-puntos) +
  ↪ labs() + annotate(text)
```

Conceptos de la función ggplot:

```
# Se inicia un diagrama y se le pueden agregar capas con +.
ggplot()
# Dataset.
data
# Usa puntos para crear diagramas (figura geométrica).
geom_función()
# Define cómo se aplican las variables a las propiedades
  ↪ visuales.
```

```

mapping # Va junto a la función aes()
# Especifica qué variables aplicar a los ejes x e y, como el
  ↪ color (color),
# color de relleno (fill), forma de puntos (shape), tipo de
  ↪ línea (linetype)
# y tamaño (size).
aes()
# Grado de transparencia del color.
alpha()
# Se pueden introducir etiquetas como
  ↪ title="", subtitle="", caption="".
labs
# Se utiliza para agregar texto dentro de la cuadrícula.
annotate(text)
# Más información en
  ↪ https://ggplot2.tidyverse.org/reference/annotate.html

```

Funciones geom:

```

# Crea diagramas de dispersion.
geom_point()
# Crea graficos de barra.
geom_bar()
# Gráfico de líneas.
geom_line()
# Suavizado. Crea una línea de tendencia.
geom_smooth()
# Crea un histograma.
geom_histogram()
# Crea un diagrama de dispersión y agrega una pequeña
  ↪ cantidad de ruido aleatorio para lidiar con la
  ↪ superposición de puntos.
geom_jitter()

```

Cuando se usa la función `geom_bar`, R cuenta automáticamente cuantas veces aparece cada valor `x` en los datos y muestra los recuentos en el eje `y`. Por esta razón, se puede poner en el código solo el eje `x`. Se puede usar `fill` para llenar de color las barras.

En el caso de `geom_histogram`, los argumentos de esta función son `bins=numero`, el cual hace referencia al número de intervalos y `binwidth=numero` es la amplitud de los intervalos.

6.7.1. Suavizado

El suavizado permite detectar una tendencia de datos aun cuando no se pueda notar con facilidad una tendencia en los puntos de datos graficados. En ggplot2, suma una línea de suavizado como otra capa en un diagrama. Ayuda a que los diagramas sean más legibles.

1. El suavizado LOESS es óptimo para suavizar diagramas con menos de 1000 puntos.

```
ggplot(data, aes(x=, y=))+ geom_point() +  
  ↪ geom_smooth(method="loess")
```

2. El suavizado GAM es útil para suavizar diagramas con un gran numero de puntos.

```
ggplot(data, aes(x=, y=))+ geom_point() +  
  ↪ geom_smooth(method="gam", formula = y ~ s(x))
```

6.7.2. Estética y facetas

Una faceta es una cara o sección de un objeto. Sirve para comparar datos. Permiten mostrar grupos más pequeños, o subconjuntos, de datos.

```
# Facetar el diagrama con una variable.  
facet_wrap(~variable)  
# Facetar el diagrama con dos variables.  
facet_grid(var1~var2)
```

6.7.3. Guardar gráficas

Para guardar una gráfica se puede exportar desde RStudio o usar la función:

```
ggsave(\nombre-grafico.tipo-archivo", width=ancho,  
  ↪ height=largo) # del paquete ggplot2. Ejemplo:  
ggsave(\Pinguinos.png")
```


Capítulo 7

Markdown

Markdown es un lenguaje usado en documentos donde existen dos tipos de celdas: una de texto y otra de código. Las celdas de texto están formateadas con este lenguaje, lo que lo hace fácil y simple de escribir.

Tanto Colab como R permiten insertar ecuaciones usando la notación de LaTeX y personalizar la escritura con código HTML.

Se le puede dar formato a la escritura con los siguientes comandos:

1. *Cursiva*: `*texto*`
2. **Negrita**: `**texto**`
3. Tachado: `~texto~`
4. [Enlaces](#): `[Clic aquí](aquiVaElEnlace)`
5. Monoespaciado: ``texto``
6. Imagen: `![Una imagen](linkDeLaImagen)`

Los encabezados en Markdown se inician con un numeral. Mientras más numerales tenga, más pequeño es el encabezado.

Capítulo 1: **Capítulo 1:**

Sección 1: **Sección 1:**

Subsección: **Subsección:**

Se pueden insertar bloques de código en las celdas de texto usando las comillas

“

“python

import pandas as pd “

De esta forma, el código en la celda tendrá un formato especial:

```
import pandas as pd
```

Se pueden crear listas ordenadas:

1. Propiedad 1
2. Propiedad 2
3. Propiedad 3

Y listas desordenadas con guiones (-) o asteriscos (*):

- Propiedad 1
- Propiedad 2
- Propiedad 3

Es posible insertar líneas horizontales con tres asteriscos (***) o tres guiones (—).

Capítulo 8

Machine Learning

El aprendizaje automático (machine learning en inglés) permite que los computadores puedan identificar patrones y hacer predicciones por medio de modelos que analizan datos. En este contexto, un modelo toma una muestra como datos de entrada y entrega una predicción como salida.

Las librerías más utilizadas en machine learning son:

1. Scikit-learn: ofrece una amplia gama de herramientas para la selección de características, construcción y evaluación de modelos.
2. TensorFlow: se utiliza para construir y entrenar redes neuronales profundas en entornos distribuidos.
3. Keras: se ejecuta sobre TensorFlow. Proporciona una API simple para construir y entrenar redes neuronales.
4. PyTorch: proporciona una API dinámica y flexible. Se utiliza esencialmente para calcular gradientes de forma automática.
5. XGBoost.

8.1. Conceptos previos

Algunos conceptos de aprendizaje automático:

- El entrenamiento se refiere a a que el modelo aprende por medio de ejemplos etiquetados.
- Los ejemplos o instancias se refieren a un caso dentro de este conjunto de datos en el que se entrena o evalúa el modelo. Se compone por varias características. Puede ser pensada como una fila en una tabla donde cada

columna representa una característica. Los ejemplos pueden ser etiquetados o sin etiqueta.

- Los ejemplos de entrada se refieren a las variables independientes, predictores o atributo, mientras que los de salida son las etiquetas.
- Las características son las propiedades que se utilizan para representar cada instancia o ejemplo en el set de datos sobre el que se entrena un modelo. Se refiere a los datos de entrada que el modelo usa para hacer sus predicciones.

Antes de entrar en los algoritmos del aprendizaje supervisado, es importante tener en cuenta lo que es el overfitting y el underfitting, la búsqueda de hiperparámetros y las métricas de evaluación de los modelos.

1. Overfitting: El sobreajuste ocurre cuando un modelo se ajusta de forma excesiva a los datos de entrenamiento. No aprende patrones para poder aplicarlo a datos nuevos. Su rendimiento es muy bueno en el conjunto de entrenamiento, pero pésimo en conjunto de prueba. Se puede identificar el sobreajuste cuando el rendimiento del modelo en el conjunto de entrenamiento es mejor que en el de conjunto de prueba.
2. Underfitting: El subajuste ocurre cuando el modelo no logra capturar los patrones en los datos. Se ajusta bien a los datos de entrenamiento y tiene un rendimiento deficiente en el conjunto de entrenamiento y en el de prueba. Se puede identificar el subajuste cuando el modelo en el conjunto de entrenamiento y de prueba es bajo.

En el aprendizaje automático, existen cuatro tipos de aprendizaje que se verán con detalle en las siguientes secciones:

1. Aprendizaje supervisado.
2. Aprendizaje no supervisado.
3. Aprendizaje reforzado.
4. Inteligencia artificial generativa.

8.2. Aprendizaje supervisado

Los modelos de aprendizaje supervisado se entrenan con conjuntos de datos etiquetados, es decir, ejemplos de entradas con sus correspondientes salidas. De esta forma, el modelo aprende a asociar los datos de entradas con sus correspondientes salidas para luego predecir la salida cuando se le presenten nuevos datos

de entrada. Finalmente, se evalúa el modelo usando un conjunto de prueba, el cual también está conformado por datos etiquetados.

Este aprendizaje se divide en dos categorías:

1. Clasificación: se utiliza para predecir la probabilidad de que algo pertenezca a una categoría (puede ser binaria o multiclase).
2. Regresión: predice un valor numérico.

Antes de entrar en los algoritmos del aprendizaje supervisado, es importante tener en cuenta lo que es el overfitting y el underfitting, la búsqueda de hiperparámetros y las métricas de evaluación de los modelos.

1. Overfitting: El sobreajuste ocurre cuando un modelo se ajusta de forma excesiva a los datos de entrenamiento. No aprende patrones para poder aplicarlo a datos nuevos. Su rendimiento es muy bueno en el conjunto de entrenamiento, pero pésimo en conjunto de prueba. Se puede identificar el sobreajuste cuando el rendimiento del modelo en el conjunto de entrenamiento es mejor que en el de conjunto de prueba.
2. Underfitting: El subajuste ocurre cuando el modelo no logra capturar los patrones en los datos. Se ajusta bien a los datos de entrenamiento y tiene un rendimiento deficiente en el conjunto de entrenamiento y en el de prueba. Se puede identificar el subajuste cuando el modelo en el conjunto de entrenamiento y de prueba es bajo.

Búsqueda de hiperparámetros

Los hiperparámetros son parámetros que no se aprenden del conjunto de datos durante el entrenamiento de un modelo. Estos se establecen antes de entrenar el modelo. Mediante la búsqueda de hiperparámetros, se encuentra la combinación óptima (maximizar el rendimiento del modelo) de valores para los hiperparámetros de dicho modelo.

Aquí se muestra un ejemplo de la búsqueda de los hiperparámetros *penalty* y *C* de una regresión logística:

```
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression

# Cargando el conjunto de datos
X = df.drop('target', axis=1)
y = df['target']
```

```

# Dividiendo los datos en conjuntos de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y,
    ↪ test_size=0.3, random_state=42)

# Definiendo los hiperparametros a ajustar
param_grid = {'C': [0.001, 0.01, 0.1, 1, 10, 100], 'penalty':
    ↪ ['l1', 'l2']}

# Creando una instancia del modelo
model = LogisticRegression()

# Creando una instancia de GridSearchCV
grid_search = GridSearchCV(model, param_grid, cv=5)

# Ajustando GridSearchCV al conjunto de entrenamiento
grid_search.fit(X_train, y_train)

# Imprimiendo los mejores hiperparametros encontrados
print("Mejores hiperparametros:", grid_search.best_params_)

# Obteniendo el mejor modelo
best_model = grid_search.best_estimator_

# Evaluando el mejor modelo
test_score = best_model.score(X_test, y_test) # Precision
print("Puntuacion en el conjunto de prueba:", test_score)

```

SMOTE

La librería SMOTE (Synthetic Minority Over-sampling Technique) se usa para el problema de desbalanceo de clases. Genera muestras sintéticas de la clase minoritaria para aumentar el número de muestras de dicha clase.

```

# SMOTE
#agregamos la librería (las anteriores las supondremos
    ↪ cargadas
from imblearn.over_sampling import SMOTE
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression

```

```
# Características
X = df.drop('target', axis=1).values
# Target
y = df['target'].values
# Aplicando SMOTE para balancear las clases
smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X, y)
# Estandarizando las características (opcional pero
→ recomendado)
scaler = StandardScaler()
X_resampled = scaler.fit_transform(X_resampled)
# Creando el modelo de Regresión Logística
logistic_regression_model =
→ LogisticRegression(random_state=42)
# Ajustando el modelo
logistic_regression_model.fit(X_resampled, y_resampled)
# Predicciones
y_pred = logistic_regression_model.predict(X_resampled)
```

Métricas de evaluación

1. Exactitud (Accuracy): Absoluto del porcentaje que se predijo correctamente. Mide la proporción de las predicciones que son correctas.
2. Precisión (Precision): Se busca minimizar los falsos positivos. Mide cuando algo positivo sea realmente positivo. Minimiza los falsos positivos.
3. Sensibilidad (Recall): Evalúa cuántos de los casos positivos reales que fueron identificados de forma correcta por el modelo.
4. Puntuación F1 (F1-score): Promedio armónico entre precisión y sensibilidad. Es útil cuando se busca un balance entre ambas métricas.

Cuando se quiere optimizar sobre precisión, se aumenta el umbral. Se quiere estar seguro de que sea positivo.

Cuando se quiere optimizar sobre la sensibilidad, se baja el umbral. Esto se hace para capturar a todas las clases positivas.

Otras métricas de evaluación que usualmente se usan en modelos de regresión son:

- Error medio absoluto (MAE): Calcula el promedio de las diferencias absolutas entre los valores predichos y los valores reales. Mientras menor el MAE, mejor será el ajuste del modelo.

- Error cuadrático medio (MSE): Calcula el promedio de los cuadrados de las diferencias entre los valores predichos y los valores reales. Mientras más bajo sea el MSE, mejor será el ajuste.
- Raíz del error cuadrático medio (RMSE): Es la raíz cuadrada del MSE.

En Python se calculan como:

```
import numpy as np
from sklearn.metrics import r2_score, mean_squared_error,
    ↪ mean_absolute_percentage_error

# Calculando el Error Cuadrático Medio (MSE)
mse = mean_squared_error(y_predict, y_test)

# Calculando el Error Cuadrático Medio Explicado (MAPE)
mape = mean_absolute_percentage_error(y_predict, y_test)

# Calculando el valor R cuadrado del modelo
r2 = r2_score(y_predict, y_test)

print(results)
print(f"MSE: {mse:.2f}")
print(f"MAPE: {mape:.2f}")
print(f"R cuadrado: {r2:.2f}")

from sklearn.metrics import accuracy_score, precision_score,
    ↪ recall_score, classification_report

# Exactitud
accuracy = accuracy_score(y, y_pred)

# Precision
precision = precision_score(y, y_pred)

# Sensibilidad
recall = recall_score(y, y_pred)

# Reporte que incluye precision, sensibilidad y f1
reporte = classification_report(y, y_pred)
```

También, se puede evaluar el modelo mediante la matriz de confusión y las curvas ROC-AUC y PRC-AUC, las cuales se verán a continuación.

Matriz de confusión

La matriz de confusión proporciona una matriz resumida las predicciones correctas e incorrectas. Cuantifica los casos falsos positivos, falsos negativos, verdaderos positivos y verdaderos negativos.

	Predicción		
Actual		Positivo	Negativo
	Positivo	Verdaderos positivos	Falsos negativos
	Negativo	Falsos positivos	Verdaderos Negativos

Figura 8.1: Matriz de confusión

donde:

- Verdaderos positivos: son los casos en que los datos reales son 1 (Verdadero) y la predicción también es 1.
- Verdaderos negativos: son los casos en que los datos reales son 0 (Falso) y la predicción también es 0.
- Falsos positivos: son los casos en que los datos reales son 0 y la predicción es 1. El modelo ha pronosticado incorrectamente.
- Falsos negativos: son los casos en que los datos reales son 1 y la predicción es 0. El modelo ha pronosticado incorrectamente.

El escenario ideal es que el modelo no de falsos positivos ni falsos negativos.

En Python, la matriz de confusión se hace de la siguiente forma:

```
from sklearn.metrics import confusion_matrix,
    ConfusionMatrixDisplay

# Matriz de confusion
confusion = confusion_matrix(y, y_pred)

# Visualizando la matriz de confusion
vis = ConfusionMatrixDisplay(confusion)
vis.plot()

# Tambien, se puede visualizar de la siguiente forma
ConfusionMatrixDisplay.from_predictions(y, y_pred)
```

La siguiente gráfica muestra una la matriz de confusión. En ella, se puede apreciar que los verdaderos positivos están en la esquina inferior derecha y los verdaderos negativos en la esquina superior izquierda.

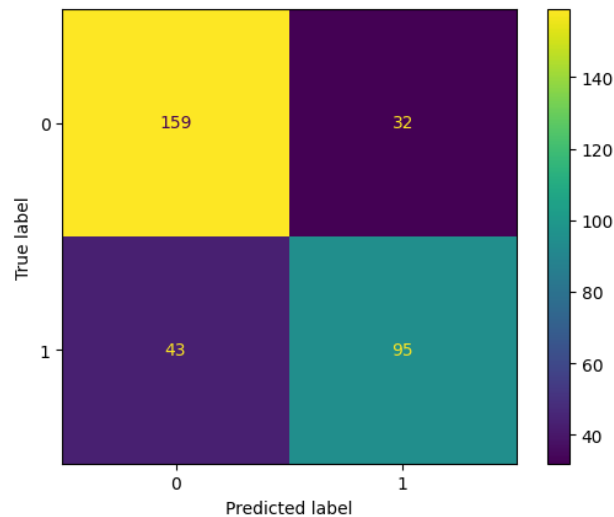


Figura 8.2: Ejemplo de matriz de confusión

Para evitar confusiones, es posible cambiar el nombre de los ejes a verdadero y falso, o true y false de la siguiente manera:

```
from sklearn.metrics import confusion_matrix,
    ↪ ConfusionMatrixDisplay

# Matriz de confusion
confusion = confusion_matrix(y, y_pred)

# Visualizando la matriz de confusion cambiando el nombre de
    ↪ los ejes
vis = ConfusionMatrixDisplay(confusion, display_labels
    ↪ =[False, True])
vis.plot()
```

Resultando en:

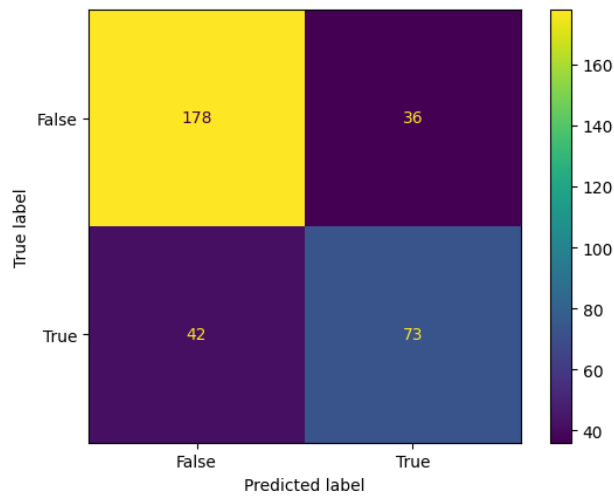


Figura 8.3: Ejemplo de matriz de confusión

Curva ROC-AUC

ROC (Receiver Operating Characteristic, o característica operativa del receptor) es una gráfica que contrasta las tasas de falsos positivos y falsos negativos.

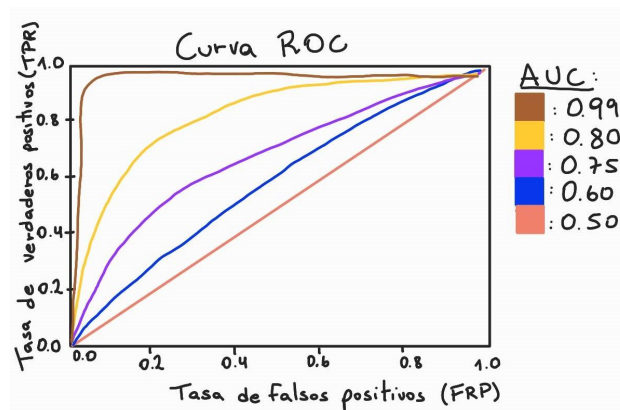


Figura 8.4: Gráfica de la curva ROC

El área bajo la curva (AUC) se utiliza como resumen del rendimiento del modelo. El AUC-ROC varía entre 0 y 1, donde un valor más cercano a 1 indica un mejor rendimiento.

Curva PRC

La curva PRC (Precision-Recall, o precisión-sensibilidad) contrasta la precisión con la sensibilidad para diferentes umbrales. El AUC-PRC varía entre 0 y 1, donde un valor más cercano a 1 indica un mejor rendimiento. Si el valor es 1, significa que el modelo puede lograr una precisión perfecta y una sensibilidad perfecta.

División del set de datos

En el aprendizaje supervisado, los algoritmos deben ser aplicados posteriormente a la división del set de datos en un conjunto de entrenamiento (train) y en un conjunto de prueba (test), con `train.test.split()`, donde sus argumentos son:

1. Características (X).
2. Etiquetas (y).
3. `test_size`: indica el porcentaje de los datos que se utilizará en el conjunto de prueba.
4. `random_state`: se utiliza para controlar la aleatoriedad en el modelo y obtener el mismo resultado al repetir la separación.

Mientras que los datos de salida del comando son:

1. `X_train`: contiene las variables independientes (características) del conjunto de entrenamiento, los cuales se utilizan para entrenar el modelo.
2. `X_test`: contiene las variables independientes del conjunto de prueba, el cual se utiliza para predecir.
3. `y_train`: contiene la variable dependiente (u objetivo) correspondiente a `X_train`. El modelo usa este conjunto para aprender de la relación entre las variables dependientes e independientes en el entrenamiento.
4. `y_test`: contiene la variable dependiente correspondiente a `X_test`. Se utiliza este conjunto para comparar las predicciones de `X_test` y evaluar qué tan precisas son.

Luego, se crea una instancia del objeto de regresión lineal para entrenar el modelo con `X_train` e `y_train` y predecir con `X_test`, guardando las predicciones en `y_pred`. Finalmente, se evalúa el modelo mediante las métricas de evaluación usando `y_test` e `y_pred`.

Regresión lineal

En aprendizaje supervisado, la regresión lineal es un algoritmo que tiene como objetivo optimizar la capacidad de predicción del modelo sobre nuevos datos.

Se debe tener en cuenta que al aplicar este algoritmo, se asume que las variables predictora y dependiente presentan una relación lineal.

```
import numpy as np
from sklearn import linear_model
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score

# Definiendo las características
X = df['col_1']
# Definiendo las etiquetas
y = df['col_2']

# Dividiendo los datos con train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,
    ↪ test_size= 0.2, random_state=42)

# Instanciando el objeto de la regresion lineal
regr = linear_model.LinearRegression()

# Entrenando el objeto
regr.fit(X_train, y_train)

# Prediciendo
y_pred = regr.predict(X_test)

# Coeficientes del modelo
y_pred = regr.predict(X_test)
print('Coeficientes: \n', regr.coef_)

# Evaluando el modelo
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print(f'Error cuadratico medio: {mse}')
print(f'r2: {r2}')
```

Se pueden verificar las predicciones creando un scatterplot que considere datos reales y predichos por el modelo:

```
import matplotlib.pyplot as plt
plt.scatterplot(y_test, y_pred)
plt.xlabel('Datos reales')
plt.ylabel('Predicciones')
plt.title('Predicciones vs datos reales')
plt.show()
```

La siguiente gráfica mostrará qué tan cercanas son las predicciones hechas por el modelo (puntos azules) a los valores reales dependiendo de la alineación de los puntos con la recta:

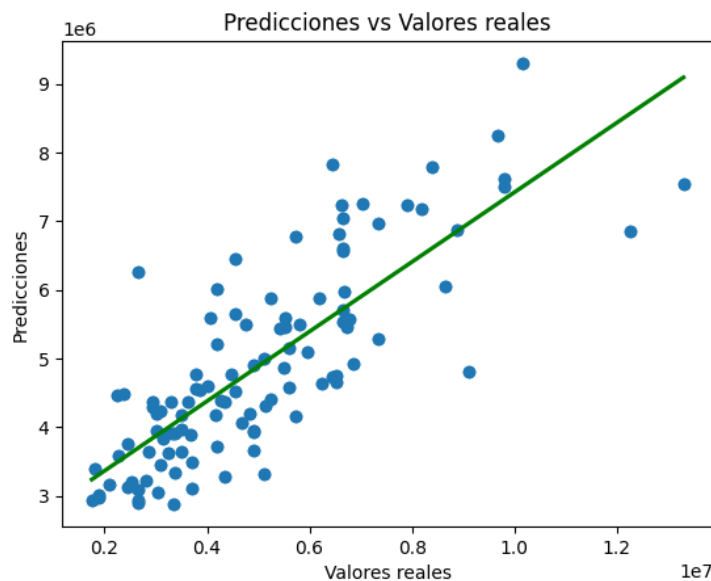


Figura 8.5: Ejemplo de gráfica de predicciones vs datos reales

Sin embargo, la regresión lineal puede sufrir problemas como el overfitting. Para solucionar esto, se usa la regularización en la regresión lineal, la cual penaliza a los coeficientes del modelo reduciendo la complejidad del mismo. Existen dos normas principales de penalización: la norma L1 y la norma L2.

1. Regularización de Lasso (L1): en esta técnica, se agrega un término de penalización proporcional al valor absoluto de los coeficientes del modelo. Tiende a reducir algunos coeficientes a cero. Ayuda a simplificar el modelo y eliminar variables irrelevantes.
2. Regularización de Ridge (L2): esta norma penaliza los coeficientes usando el cuadrado de sus valores. Tiende a mantener sus coeficientes pequeños, lo que es útil cuando todas las características son relevantes.

3. ElasticNet: combina las normas L1 y L2, el cual busca un equilibrio entre la selección de características y la reducción de coeficientes.

La regularización también se puede utilizar en la regresión logística, modelos de regresión polinomial, no lineal o de clasificación, como SVM, árboles de decisión y random forests.

Regresión logística

La regresión logística se emplea para determinar la probabilidad de que un evento pertenezca a una de dos o más categorías. Se busca relacionar un conjunto de variables independientes con una dependiente, usando coeficientes que determinan un modelo lineal. Esta regresión busca predecir categorías.

Por medio de la función logarítmica o logit, transforma la salida en una escala log-odds con la función Sigmoide:

$$f(x) = \frac{1}{1 + e^{-x}} \quad (8.1)$$

La función logística devuelve valores entre 0 (no/falso) y 1 (sí/verdadero) para la variable dependiente. Los valores por sobre 0.5 se pueden interpretar como una probabilidad mayoritaria de que ocurra un evento (clase 1), mientras que por debajo de 0.5 indica una probabilidad mayoritaria de que no ocurra (clase 0). Se define un umbral de decisión para definir una categoría como positiva y otra negativa.

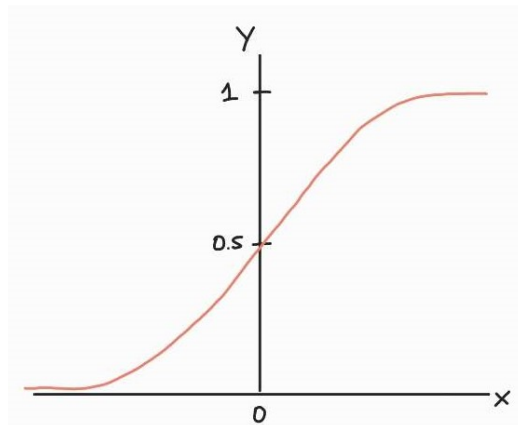


Figura 8.6: Gráfico de la función logística

Existen dos tipos de regresión logística, la binaria y multinomial:

1. La regresión logística binaria es un modelo estadístico que se utiliza para problemas de clasificación binaria donde la variable dependiente tiene dos

categorías posibles, etiquetados como 0 y 1, sí y no, etc. La función Sigmoide usa la función logística para modelar la probabilidad de que la variable dependiente pertenezca a la categoría 1 (clase positiva).

2. Por otro lado, la regresión logística multinomial se utiliza para problemas de clasificación con más de dos categorías mutuamente excluyentes. La función Softmax en la regresión logística multinomial, se usa para modelar la probabilidad de pertenecer a cada una de las categorías. Asigna probabilidades a cada categoría, dando la suma de ellas igual a 1. Puede predecir varias categorías distintas. Se utiliza en problemas de clasificación multiclase.

Para aplicar este modelo, se debe dividir el dataset en conjunto de entrenamiento y prueba. Se recomienda escalar las características (X) con `fit_transform()`. Luego, se eligen dos características para el gráfico

```
# Librerias
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler

# Variables predictoras
X = df[['var_1', 'var_2', 'var_3']]
# Variables objetivo (1 o 0 en el caso de la clasificacion
↳ binaria)
y = df[['var_y']]

# Escalando X
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Seleccionando dos características para el plot
caracteristica1 = 0
caracteristica2 = 1
X_2d = X_scaled[:, [caracteristica1, caracteristica2]]

# Dividiendo los datos en conjunto de prueba y entrenamiento
X_train, X_test, y_train, y_test = train_test_split(X_2d, y,
↳ test_size=0.2, random_state=42)
```

```

# Entrenando el modelo de regresion logistica
LogReg = LogisticRegression()
LogReg.fit(X_train, y_train)

# Obteniendo los coeficientes y el intercepto del modelo
coeficientes = LogReg.coef_
intercepto = LogReg.intercept_

Crear un meshgrid para el plot
x_min, x_max = X_2d[:, 0].min() - 1, X_2d[:, 0].max() + 1
y_min, y_max = X_2d[:, 1].min() - 1, X_2d[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.1),
np.arange(y_min, y_max, 0.1))

# Prediciendo las clases para cada punto del meshgrid
Z = LogReg.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

# Grafico de las regiones de decision y los puntos de
↪ entrenamiento
plt.figure(figsize=(8, 6))
plt.contourf(xx, yy, Z, alpha=0.6, cmap=plt.cm.coolwarm)
plt.scatter(X_train[:, 0], X_train[:, 1], c=y_train,
↪ cmap=plt.cm.coolwarm,
edgecolors='k')
plt.xlabel(caracteristica1)
plt.ylabel(caracteristica2)
plt.title('Regresion Logistica - Regiones de Decision')
plt.colorbar(label='Clase')
plt.show()

```

Support Vector Machine (SVM)

El SVM es un algoritmo utilizado para la clasificación y regresión. Busca maximizar la separabilidad entre clases en un espacio de características mediante los hiperplanos óptimos. Puede abordar problemas lineales y no lineales.

Para aplicar el modelo, se deben cargar los datos, dividir el conjunto de de entrenamiento y prueba, escalar y entrenar el modelo. Al entrenarlo, se debe usar un kernel, el cual puede ser rbf (predeterminado), linear, poly o sigmoid.

```

import pandas as pd
import numpy as np

```

```
import matplotlib.pyplot as plt
from matplotlib.gridspec import GridSpec
import seaborn as sns
from sklearn.datasets import make_blobs
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.metrics import classification_report

# Variables predictoras
X = df[['var_1', 'var_2', 'var_3']]
# Variables objetivo (1 o 0 en el caso de la clasificacion
→ binaria)
y = df[['var_y']]

# Dividiendo los datos en conjunto de prueba y entrenamiento
X_train, X_test, y_train, y_test = train_test_split(X, y,
→ test_size=0.2, random_state=42)

# Escalando las características
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Entrenando el modelo
svc_model = SVC(kernel='linear')
svc_model.fit(X_train, y_train)

# Prediciendo
svc_model.predict(X_test)
```

El algoritmo SVM posee tres hiperparámetros principales, el kernel, costo c y γ .

1. Kernel: es una función que realizan una transformación no lineal de los datos. Abordan problemas de clasificación no lineales. Los más comunes son lineal (linear), polinómico (poly), radial (rbf), sigmoide.
2. Costo c : determina cuánto se penalizan los errores en la clasificación de las muestras de entrenamiento. Un valor alto de c busca clasificar todos

los ejemplos de entrenamiento de forma correcta, lo que puede llevar a un overfitting en el conjunto de entrenamiento. Mientras que un bajo valor de c permite que el modelo tenga un margen más amplio y un mayor número de errores en la clasificación de las muestras de entrenamiento, lo que puede resultar en un underfitting en el conjunto de entrenamiento.

3. Gamma: este parámetro es específico del kernel radial. Define hasta qué punto llega la influencia de un solo ejemplo de entrenamiento. Los valores bajos de gamma indican que el alcance de influencia de un punto de entrenamiento es más amplio, donde los puntos se afectan entre sí en un área más grande, lo que podría llevar a un overfitting. Por otro lado, un valor alto de gamma indica que el alcance de influencia de un punto de entrenamiento es más estrecho, donde los puntos afectan solo a los puntos cercanos en el espacio de las características.

Para visualizar las diferencias de los kernels mediante gráficas:

```
# Variables predictoras
X = df[['var_1', 'var_2', 'var_3']]
# Variables objetivo (1 o 0 en el caso de la clasificacion
↳ binaria)
y = df[['var_y']]

# Entrenando el modelo lineal
svc_model = SVC(kernel='linear', gamma=1)
svc_model.fit(X, y)

# Prediciendo las clases para todo el espacio de
↳ características
h = 0.02 # Tamanno del paso en el meshgrid
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
↳ np.arange(y_min, y_max, h))
Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
# Plot del resultado
Z = Z.reshape(xx.shape)
plt.contourf(xx, yy, Z, cmap=plt.cm.Paired, alpha=0.8)
plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.Paired,
↳ edgecolors='k')
plt.xlabel('Caracteristica 1')
plt.ylabel('Caracteristica 2')
```

```
plt.title('Support Vector Machine con Kernel Lineal')

plt.subplot(1,2, 2)
# Entrenando el modelo rbf
model = SVC(kernel='rbf', gamma=1)
model.fit(X, y)

# Predeciendo las clases para todo el espacio de
↪ características
h = 0.02 # Tamanno del paso en el meshgrid
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
↪ np.arange(y_min, y_max, h))
Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
# Grafico del resultado
Z = Z.reshape(xx.shape)
plt.contourf(xx, yy, Z, cmap=plt.cm.Paired, alpha=0.8)
plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.Paired,
↪ edgecolors='k')
plt.xlabel('Caracteristica 1')
plt.ylabel('Caracteristica 2')
plt.title('Support Vector Machine con Kernel RBF')
plt.show()
```

KNN (K-Nearest Neighbors)

Antes de usar KNN, es importante preparar los datos dividiendo los conjuntos de entrenamiento y prueba, manipulando los datos faltantes y normalizando las características. Luego, se entrena el modelo KNN ajustando el número de vecinos (`n_neighbors`). Con el modelo entrenado, se pueden realizar predicciones sobre datos nuevos y evaluar su rendimiento.

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

# Dividiendo los datos en características (X) y target (y)
X = datos.drop('target', axis=1)
y = datos['target']
```



```
# Dividiendo los datos en conjuntos de entrenamiento (70) y
↳ prueba (30)
X_train, X_test, y_train, y_test = train_test_split(X, y,
↳ test_size=0.3, random_state=42)

# Normalizando las características (recomendado para KNN)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Creando el modelo KNN
knn = KNeighborsClassifier(n_neighbors=5)

# Entrenando el modelo
knn.fit(X_train_scaled, y_train)

# Predicciones
y_pred = knn.predict(X_test_scaled)

# Evaluando el modelo
accuracy = accuracy_score(y_test, y_pred)
print(f'Precision: {accuracy}')
```

Se debe tener en cuenta que la elección del número de vecinos (`n_neighbors`) es importante. Un número bajo puede hacer que el modelo sea sensible al ruido de los datos. Por otro lado, un número alto puede hacer que el modelo sea demasiado general.

Árboles de decisión

Los árboles de decisión se utilizan para la clasificación como para la regresión.

Como en KNN, se deben preparar los datos y dividir los conjuntos de entrenamiento y prueba para entrenar el modelo y hacer predicciones.

```
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

# Dividiendo los datos en características (X) y target (y)
X = datos.drop('target', axis=1)
y = datos['target']
```

```
# Dividiendo los datos en conjuntos de entrenamiento (70) y
↳ prueba (30)
X_train, X_test, y_train, y_test = train_test_split(X, y,
↳ test_size=0.3, random_state=42)

# Creando el modelo
modelo_arbol = DecisionTreeClassifier(random_state=42)

# Entrenando el modelo
modelo_arbol.fit(X_train, y_train)

# Predicciones
predicciones = modelo_arbol.predict(X_test)

# Evaluando el modelo
accuracy = accuracy_score(y_test, predicciones)
```

Árboles de regresión

Los árboles de regresión se utilizan para predecir valores numéricos. Divide el espacio de características en regiones rectangulares donde a cada región se le asocia con una predicción numérica. Estas divisiones se realizan repetidamente a las características de entrada creando un árbol de decisiones donde cada nodo interno representa una pregunta sobre una características y cada hoja un valor de predicción.

Son útiles cuando existe la sospecha de que la relación entre las características y la variable de respuesta son no lineales.

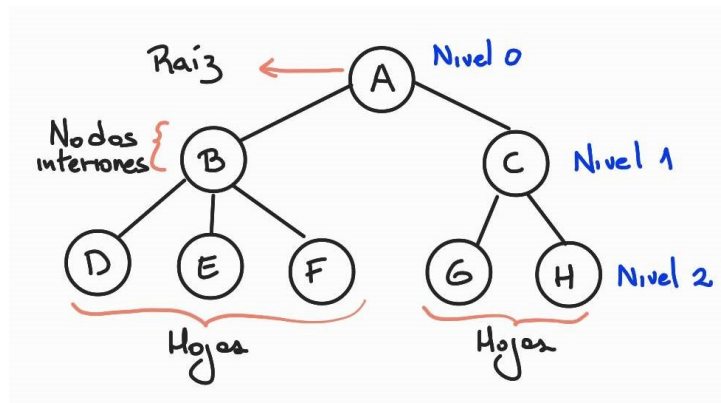


Figura 8.7: Visualización de un árbol de decisión

El árbol inicia con un nodo raíz que contiene todos los datos de entrenamiento. En los siguientes pasos, se selecciona una característica y un cierto umbral que dividirá los datos en grupos. Luego, se calcula la predicción para cada región.

El siguiente código muestra cómo aplicar el modelo a un set de datos junto al gráfico del árbol de regresión:

```

import seaborn as sns
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor
from sklearn.tree import plot_tree
from sklearn.metrics import mean_squared_error

# Convirtiendo las variables categoricas a numericas
df = pd.get_dummies(df, drop_first=True)

# Datos
X = df.drop(['target'], axis=1)
y = df['target']

# Dividiendo el conjunto de datos en entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y,
    ↪ test_size=0.2, random_state=42)

# Creando y entrenando el modelo de arbol de regresion
# Se fija la profundidad en 2 (max_depth)
  
```

```
reg_tree = DecisionTreeRegressor(max_depth=2,  
    ↪ random_state=42)  
reg_tree.fit(X_train, y_train)  
  
# Predicciones  
y_pred = reg_tree.predict(X_test)  
  
# MSE  
mse = mean_squared_error(y_test, y_pred)  
print("Error Cuadratico Medio (MSE):", mse)  
  
# Graficando el arbol de regresion  
plt.figure(figsize=(15, 8))  
plot_tree(reg_tree, feature_names=X.columns, filled=True,  
    ↪ rounded=True)  
plt.show()
```

Los árboles de decisión poseen cinco hiperparámetros importantes:

1. Profundidad máxima (max_depth): controla la cantidad de divisiones que puede tener el árbol. Si la profundidad es demasiado grande, puede ser más propenso al sobreajuste, aunque también puede capturar relaciones más complejas en los datos.
2. Número mínimo de muestras en una hoja (min_samples_leaf): establece el número mínimo de muestras que deben estar en una hoja. Si el valor que se establece es alto, el árbol tendrá hojas con más muestras. Un valor bajo puede llevar al overfitting.
3. Número mínimo de muestras para dividir (min_samples_split): define el número mínimo de muestras para que un nodo pueda dividirse. Un valor alto evita divisiones en regiones con pocas muestras, previniendo el overfitting.
4. Número máximo de nodos (max_nodes): define el número máximo de nodos en el árbol.
5. Función de criterio (criterion): mide la calidad de una división. Los dos criterios más comunes son el error cuadrático medio (MSE) y la desviación absoluta media (MAE).

Naive Bayes

Naive Bayes es un conjunto de algoritmos de clasificación basados en el Teorema de Bayes. Se usa comúnmente para clasificar texto, como filtrado de spam.

Al igual que en los modelos anteriores, se deben preparar los datos, dividir los conjuntos, crear y entrenar el modelo para hacer predicciones. Scikit-learn ofrece GaussianNB, MultinomialNB, BernoulliNB, entre otros. La elección depende de los datos. Por ejemplo, GaussianNB se usa para características con una distribución continua. MultinomialNB y BernoulliNB se usan para datos discretos.

```
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score

# Dividiendo los datos en características (X) y target (y)
X = datos.drop('target', axis=1)
y = datos['target']

# Dividiendo los datos en conjuntos de entrenamiento (70) y
  ↳ prueba (30)
X_train, X_test, y_train, y_test = train_test_split(X, y,
  ↳ test_size=0.3, random_state=42)

# Naive Bayes
modelo_nb = GaussianNB()

# Entrenando el modelo
modelo_nb.fit(X_train, y_train)

predicciones = modelo_nb.predict(X_test)

# Evaluando el modelo
accuracy = accuracy_score(y_test, predicciones)

Además de las métricas de evaluación accuracy, precision, recall, f1-score,
también se puede usar la matriz de confusión, la cual debe llevar como argumento
y_test y y_pred:

# Matriz de confusion
confusion = confusion_matrix(y_test, y_pred)
# Visualizando la matriz de confusion cambiando el nombre de
  ↳ los ejes
vis = ConfusionMatrixDisplay(confusion, display_labels
  ↳ =['False', 'True'])
vis.plot()
```

Lazy predict

Lazy predict es una herramienta que aplica varios modelos a un conjunto de datos y calcula su exactitud, la curva ROC-AUC, el puntaje F1. No es tan preciso como un modelo de aprendizaje automático dado que se usa para predecir valores en un conjunto de datos sin necesidad de entrenar un modelo.

```
# Clasificacion
from lazypredict.Supervised import LazyClassifier
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split

X = df
y = df['target']
X_train, X_test, y_train, y_test = train_test_split(X, y,
    ↪ test_size=5, random_state=123)
clf = LazyClassifier(verbose=0, ignore_warnings=True,
    ↪ custom_metric=None)
models, prediccions = clf.fit(X_train, X_test, y_train,
    ↪ y_test)
models

# Regression
from lazypredict.Supervised import LazyRegressor
from sklearn import datasets
from sklearn.utils import shuffle
import numpy as np

X = df
y = df['target']

X_train, X_test, y_train, y_test = train_test_split(X, y,
    ↪ test_size=5, random_state=123)
reg = LazyRegressor(verbose=0, ignore_warnings=False,
    ↪ custom_metric=None )
models, predictions = reg.fit(X_train, X_test, y_train,
    ↪ y_test)
models
```

Series de tiempo

En el análisis de datos, existen tres categorías importantes de datos: Cross-sectional data, Time series data y Panel data. Las principales características de cada uno son:

1. Cross-sectional: en este conjunto de datos se suelen recopilar en un punto en específico en el tiempo sobre diferentes variables o unidades, donde cada observación corresponde a una entidad distinta.
2. Time series: son un tipo de datos en el que se recopilan observaciones en un orden cronológico y se registran datos en intervalos de tiempos regulares, los cuales permiten visualizar cómo cambian las variables a lo largo del tiempo.
3. Panel: estos datos contienen observaciones de múltiples identidades en diferentes momentos temporales, los cuales se utilizan para estudiar relaciones a lo largo del tiempo.

En específico, las series de tiempo son una secuencia de puntos de datos organizados en el tiempo de forma cronológica. Los datos que son recopilados de forma irregular no son considerados como una serie de tiempo, aunque se pueden trabajar en intervalos regulares.

Las series de tiempo tienen 3 componentes principales:

1. Tendencia: captura la dirección general de la serie temporal. Puede ser ascendente, descendente o constante.
2. Estacionalidad: captura los efectos que ocurren con frecuencia específica. Las estadísticas no cambian con el tiempo, como la media, la varianza y la autocorrelación.
3. Residuos: son fluctuaciones aleatorias que no se pueden atribuir a ninguna tendencia.

Existen dos modelos esenciales de series de tiempo: los modelos autorregresivos (AR) y los modelos de medias móviles (MA). Ambos capturan patrones en los datos secuenciales.

Los modelos autoregresivos modelan la relación entre una observación actual y un conjunto de observaciones anteriores. Por otro lado, los modelos de medias móviles buscan una relación entre una observación y un conjunto de valores pasados de un término de error estocástico.

Para el modelado de series de tiempo, es importante el análisis de autocorrelación (ACF) y la función de autocorrelación parcial (PACF). Estas funciones

proporcionan información sobre las correlaciones entre valores presentes y antiguos.

- La función de autocorrelación muestra la correlación entre un punto de tiempo y valores anteriores en intervalos de tiempo diferentes.
- La función de autocorrelación parcial muestra la correlación entre un valor en un punto de tiempo y valores anteriores en un solo intervalo.

Para determinar si una serie es estacionaria o no, se utiliza el Test de Dickey-Fuller. Plantea una hipótesis nula (H_0) que asume a no estacionalidad y la hipótesis alternativa que asume estacionalidad. El test proporciona una estadística conocida como ADF Static. Mientras más negativa sea la estadística, estará más en favor de la estacionalidad. Este test también proporciona un valor p , que se usa para evaluar la significancia estadística del resultado. Si el valor p es menor que un umbral definido, se rechaza la hipótesis nula de que la serie es no estacionaria.

El modelo más ampliamente utilizado en el análisis de series de tiempo para modelar y predecir datos es ARIMA (Autoregressive Integrated Moving Average). Combina los componentes de PACF (p) y ACF (q) junto con la diferenciación (d) para manejar tendencia en los datos. ARIMA se debe utilizar con series de tiempo estacionarias, por lo que antes de aplicar el modelo a un set de datos se debe hacer el Test de Dickey-Fuller. En caso de que no sea estacionaria, se debe diferenciar.

Aplicando todos los conceptos anteriores, el código tendría la siguiente estructura. Luego de cargar los datos, se deben transformar el tipo de dato a `datetime` para las fechas con `pd.to_datetime` y se deben indexar las fechas con `set_index()`. Para poder visualizar la serie de tiempo, se puede crear un gráfico o descomponer la serie en tendencia, estacionalidad y residuo.

```
from date import datetime
import matplotlib.pyplot as plt
import seaborn as sns
from statsmodels.tsa.seasonal import seasonal_decompose

# Suponiendo que df es el dataframe ya cargado
# Si es necesario, cambiar el tipo de dato a datetime
df['Fecha'] = pd.to_datetime(df['Fecha'])

# Indexando las fechas
df = df.set_index('Fecha')
# Se puede crear una columna por mes:
# df['mes'] = df.index.month
# 0 por días de la semana:
```



```
# df['dia_semana'] = df.index.day_of_week + 1

# Visualizacion general de los datos
plt.plot(df)
plt.title('Serie de tiempo')
plt.xlabel('Fecha')
plt.ylabel('Ventas')
plt.show()

# Descomponiendo la serie de tiempo
result = seasonal_decompose(df, model='additive')
trend = result.trend
seasonal = result.seasonal
residual = result.resid

# Mostrando los componentes descompuestos
plt.figure(figsize=(12, 8))
# Original
plt.subplot(411)
plt.plot(df, label='Original')
plt.legend(loc='upper left')
# Tendencia
plt.subplot(412)
plt.plot(trend, label='Tendencia')
plt.legend(loc='upper left')
# Estacionalidad
plt.subplot(413)
plt.plot(seasonal, label='Estacionalidad')
plt.legend(loc='upper left')
# Residuo
plt.subplot(414)
plt.plot(residual, label='Residuo')
plt.legend(loc='upper left')
plt.tight_layout()
plt.show()
```

Posteriormente, el Test de Dickey-Fuller entregará un valor p señalando si la serie de tiempo es estacionaria o no. Si el resultado es menor o igual al umbral (0.05 en este caso), la hipótesis nula H_0 es rechazada, por lo tanto los datos son estacionarios. De lo contrario, si el resultado es mayor al umbral establecido, la hipótesis nula H_0 no puede ser rechazada y, por lo tanto, los datos no son

estacionarios.

```
from statsmodels.tsa.stattools import adfuller

# Test de Dickey-Fuller
result = adfuller(df['Ventas'])
print('ADF Statistic:', result[0])
print('p-value:', result[1])
print('Critical Values:', result[4])
# Resultado <= 0.05: H0 rechazada, los datos son estacionarios
# Resultado > 0.05: H0 no rechazada, los datos no son
  ↪ estacionarios
```

Si los datos son no estacionarios, se puede diferenciar la serie. Se le aplica el Test nuevamente para saber si los datos ahora son estacionarios. Esto se debe iterar hasta que se cumpla la estacionalidad.

```
# Diferenciando
df['Ventas_diff'] = df['Ventas'].diff()
```

El siguiente paso es visualizar la autocorrelación y la autocorrelación parcial de los datos mediante gráficos.

```
# Autocorrelacion ACF
fig, axes = plt.subplots(1, 2, figsize=(10, 4))
plot_acf(df, lags=20, title='ACF - Estacionaria', alpha=0.05,
  ↪ ax=axes[0])

# Autocorrelacion parcial PACF
plot_pacf(df, lags=20, title='PACF - Estacionaria',
  ↪ alpha=0.05, ax=axes[1])

# Ajustando el espaciado entre subplots
plt.tight_layout()

# Mostrando el grafico
plt.show()
```

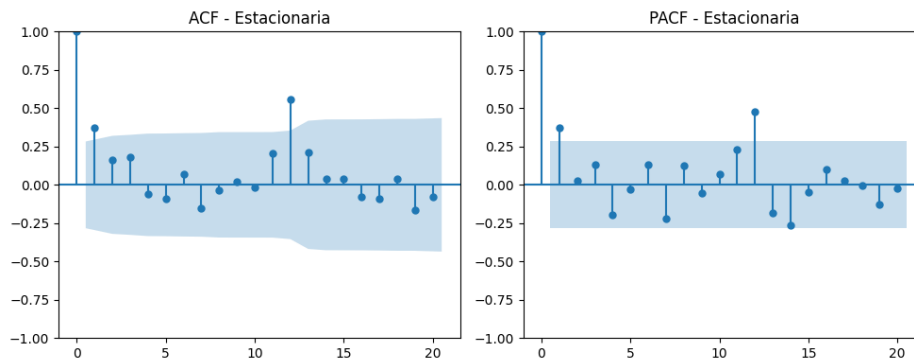


Figura 8.8: Ejemplo de gráfica de ACF y PACF

Las bandas azules alrededor del cero representan los intervalos de confianza. Si un valor de autocorrelación cae fuera de estas bandas tanto en el gráfico de ACF como en el ed PACF, se considera estadísticamente significativo en esos puntos de tiempo. Estos valores pueden ayudar a identificar el retardo óptimo de series de tiempo. En el caso de los gráficos anteriores, los valores pueden ser 12 en cada uno.

Con esto, se tienen los parámetros p , q y d , los cuales corresponden a PACF, ACF y la diferenciación. Para poder modelar ARIMA se deben usar estos valores, sin embargo, antes se deben dividir los datos en un set de prueba y entrenamiento. Finalmente, se evalúa su rendimiento con las métricas de evaluación.

```
from statsmodels.tsa.arima.model import ARIMA
from sklearn.metrics import mean_squared_error,
    ↪ mean_absolute_percentage_error

# Separando los datos en entrenamiento y prueba
train_size = int(len(df) * 0.8)
train_data, test_data = df[:train_size], df[train_size:]

# Aplicando ARIMA
model_arima = ARIMA(train_data['Ventas'], order=(p, d, q))
model_arima_fit = model_arima.fit()

# Predicciones
forecast_arima =
    ↪ model_arima_fit.forecast(steps=len(test_data))

# Visualizando resultados
plt.figure(figsize=(12, 6))
```

```

# Datos reales
plt.plot(test_data.index, test_data['Ventas'], label='Real
↳ Data')
# Predicciones con ARIMA
plt.plot(test_data.index, forecast_arima, label='ARIMA
↳ Forecast')
plt.xlabel('Fecha')
plt.ylabel('Ventas')
plt.title('Comparacion de Forecasting en Serie de Tiempo')
plt.legend()
plt.show()

# Metricas de ARIMA
mse = mean_squared_error(forecast_arima, test_data['Ventas'])
mae = mean_absolute_percentage_error(forecast_arima,
↳ test_data['Ventas'])
print(f'MSE ARIMA: {mse:.2f}')
print(f'MAE ARIMA: {mae:.2f}')

```

También, se puede modelar con Auto-ARIMA, donde el parámetro *m* se refiere al número de periodo para cada estacionalidad:

- *m*=7: días.
- *m*=12: meses.
- *m*=52: semanas.
- *m*=4: trimestral.
- *m*=1: anual.

```

# pip install pmdarima
from pmdarima import auto_arima
from sklearn.metrics import mean_squared_error,
↳ mean_absolute_percentage_error

# Separando los datos en entrenamiento y prueba
train_size = int(len(df) * 0.8)
train_data, test_data = df[:train_size], df[train_size:]

# auto arima

```

```

model_auto_arima = auto_arima(train_data['Ventas'],
    ↪ seasonal=True, m=12)

# Predicciones
forecast_auto_arima =
    ↪ model_auto_arima.predict(n_periods=len(test_data))

# Visualizando resultados
plt.figure(figsize=(12, 6))
# Datos reales
plt.plot(test_data.index, test_data['Ventas'], label='Real
    ↪ Data')
# Predicciones con auto ARIMA
plt.plot(test_data.index, forecast_auto_arima, label='AUTO
    ↪ ARIMA Forecast')
plt.xlabel('Fecha')
plt.ylabel('Ventas')
plt.title('Comparacion de Forecasting en Serie de Tiempo')
plt.legend()
plt.show()

# Metricas de AUTO ARIMA
mse = mean_squared_error(forecast_auto_arima,
    ↪ test_data['Ventas'])
mae = mean_absolute_percentage_error(forecast_auto_arima,
    ↪ test_data['Ventas'])
print(f'MSE ARIMA: {mse:.2f}')
print(f'MAE ARIMA: {mae:.2f}')

```

Otra biblioteca para el análisis y predicción de series de tiempo es Prophet. Para aplicarlo, las columnas de fecha deben llamarse ds y los valores de la serie y.

```

# pip install fbprophet
from prophet import Prophet

# Renombrando las columnas
df.rename(columns={'Fechas' : 'ds',
    ↪ 'Ventas' : 'y'}, inplace=True)

# Creando y ajustando el modelo
m = Prophet()

```

```
m.fit(df)

# Generando fechas futuras para las predicciones
future = m.make_future_dataframe(periods=365) # Periods es el
↪ numero de dias futuros a predecir
future.tail()

# Prediciendo
forecast = m.predict(future)
forecast[['ds', 'yhat', 'yhat_lower', 'yhat_upper']].tail()

# Grafico general
fig1 = m.plot(forecast)
# Grafico descompuesto
fig2 = m.plot_components(forecast)
```

8.2.1. Aprendizaje no supervisado

Este modelo hace predicciones a partir de conjuntos de datos que no tienen una salida o datos etiquetados. La etiqueta es la respuesta o valor que se quiere que el modelo prediga. Su objetivo es identificar patrones en los datos de entrada. En lugar de predecir una salida, busca agrupar los datos en función de características comunes.

Posee dos categorías:

1. Clustering (segmentación): es una técnica que tiene como objetivo agrupar datos similares en grupos llamados clusters, donde un cluster es un conjunto de objetos que comparten características similares entre sí. Algunos algoritmos de clustering son K-means, Fuzzy CMean, DBSCAN o clustering jerárquico.
2. Reducción de dimensionalidad: utilizan técnicas para reducir la cantidad de variables en un conjunto.

Antes de entrar en detalle a los algoritmos de aprendizaje no supervisado, se debe tener en cuenta las métricas de evaluación para estos modelos.

Métricas de evaluación

Las métricas de evaluación para los modelos de aprendizaje no supervisados consta del puntaje de Silhouette y el de Davies Bouldin.

```
from sklearn.metrics import silhouette_score
from sklearn.metrics import davies_bouldin_score
# Metricas de evaluacion
# Puntuacion de silhouette
silhouette_scores = silhouette_score(X, labels)
# Indice de David Boulding
dunn_index = davies_bouldin_score(X, labels)
print(f'Silhouutte Score : {silhouette_scores}')
print(f'Davies Bouldin Score : {dunn_index}')
```

Para validar un modelo de clustering, se usan los siguientes métodos:

- Distancia Intra Cluster (SSE): mide la coherencia de los objetos dentro de un mismo grupo. Cuanto menor sea la distancia intracluster, mayor la cohesión dentro de este.
- Coeficiente de Silhouette: mide la cohesión y separación de los clusters. Devuelve un valor entre -1 y 1, donde 1 indica una buena separación entre los clusters y alta cohesión.
- Índice de Davies-Bouldin: mide la similitud entre los clusters y distancia entre los centroides de estos. Un valor más bajo indica una mejor separación y cohesión de los clusters.
- Validación de expertos: se refiere a la participación de personas con experiencia en el dominio de los datos para evaluar los resultados.

K-means

El algoritmo K-means busca agrupar conjuntos de datos similares en clusters. El proceso de este algoritmo es el siguiente:

1. Inicialización: Se selecciona el número K de clusters y se inicializan los centroides de los K clusters de forma aleatoria o usando un método.
2. Asignación: A cada punto de datos se le asigna al cluster cuyo centroide esté más cerca.
3. Actualización: Se calculan nuevos centroides de los clusters.
4. Se repiten los pasos 2 y 3 con los nuevos centroides.

Por medio del método del codo se puede determinar el número óptimo de clusters (K) en el algoritmo. El objetivo es identificar el valor de K donde se produce un cambio en la variabilidad explicada por los clusters.

El procedimiento de este método consta de ejecutar el algoritmo K-means para un rango de valores de K para posteriormente calcular la suma de las distancias al cuadrado dentro de los clusters para cada valor de K. Luego, se grafica el valor de K y la métrica intra cluster. Con esto, se puede buscar el punto en el gráfico donde se produce una disminución significativa en la métrica intracluster, el cual indica el número óptimo de clusters.

El siguiente código muestra el método del codo para identificar el número óptimo de clusters y el algoritmo K-means usando el conjunto de datos Iris:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.cluster import KMeans
import seaborn as sns
from sklearn.metrics import silhouette_score
from sklearn.metrics import davies_bouldin_score
from sklearn.metrics import pairwise_distances

import warnings
warnings.filterwarnings('ignore')

# Cargando los datos
iris = load_iris()
X = pd.DataFrame(iris.data)
X

# Metodo del codo
# Definiendo una lista de valores de k
k_values = range(2, 8)
# Inicializando listas para almacenar las metricas
inertia_values = []

# Realizar clustering con diferentes valores de k y calcular
↳ las métricas
for k in k_values:
    kmeans = KMeans(n_clusters=k, random_state=42) #random_state
    ↳ es una semilla para valores aleatorios
    kmeans.fit(X)
    labels = kmeans.labels_
# Calcular la inercia
```



```

inertia_values.append(kmeans.inertia_)

# Graficando el metodo del codo utilizando la inercia
plt.plot(k_values, inertia_values, 'bo-')
plt.xlabel('Numero de clusters (k)')
plt.ylabel('Inercia')
plt.title('Metodo del Codo')
plt.show()

```

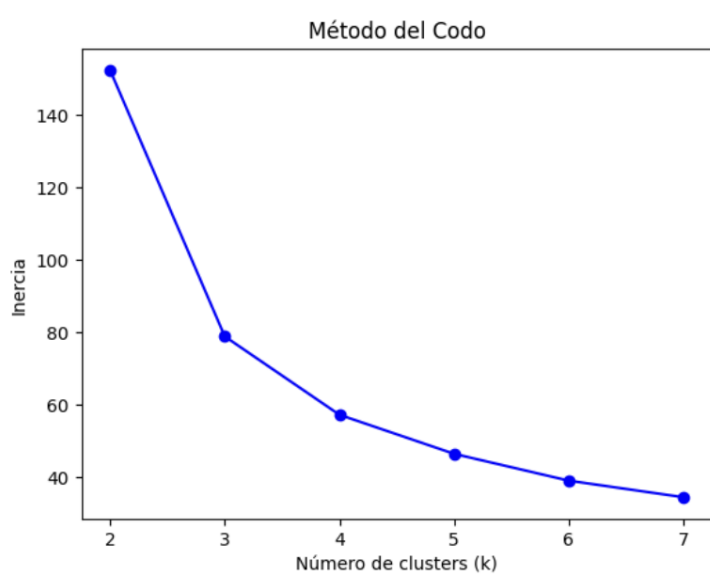


Figura 8.9: Método del codo

El gráfico anterior muestra que el número óptimo de k es 3, por lo tanto:

```

optimal_k = 3
kmeans = KMeans(n_clusters=optimal_k, random_state=42)
# Aplicando kmeans
kmeans.fit(X)
# Etiquetando los resultados
labels = kmeans.labels_
# Insertando una columna con las etiquetas de cluster
X['clusters'] = labels

# Graficando los clusters en el espacio reducido por PCA
# Se utiliza la segunda y la tercera columna
plt.scatter(X.iloc[:, 1], X.iloc[:, 2], c=labels)

```

```

plt.xlabel('Componente Principal 1')
plt.ylabel('Componente Principal 2')
plt.title('Clusters K-means (k=3)')
plt.show()

# Métricas de evaluación
# Puntuación de silhouette
silhouette_scores = silhouette_score(X, labels)
# Índice de David Bouldin
dunn_index = davies_bouldin_score(X, labels)
print(f'Silhouette Score : {silhouette_scores}')
print(f'Davies Bouldin Score : {dunn_index}')

```

Fuzzy C-Means (FCM)

Esta técnica de clustering asigna a cada dato un grado de membresía difuso. Como consecuencia, un dato puede pertenecer a múltiples clusters.

El algoritmo Fuzzy C-Means requiere de la especificación del parámetro fuzziness (m), el cual controla la difusión de las asignaciones de membresía. Si m es igual a 1, los datos pertenecen a un cluster o a otro. Si es mayor a 1, la pertenencia es difusa, lo que implica que los puntos pueden tener grados de pertenencia parciales a múltiples clusters. Mientras mayor sea m , más difusas son las asignaciones.

El procedimiento es el siguiente:

1. Inicialización: Se selecciona el número K de clusters y se inicializan los centroides de los K clusters de forma aleatoria y los grados de pertenencia para cada dato. Estos grados de pertenencia son valores entre 0 y 1.
2. Cálculo de los centroides: Se calculan los centroides ponderados usando los grados de pertenencia.
3. Actualización de los grados de pertenencia: Se actualizan los grados de pertenencia de cada punto usando una función de pertenencia que usa la distancia de los centroides.
4. Iteración: Se repiten los pasos 2 y 3 hasta que se alcance un criterio de convergencia.

El siguiente código muestra la aplicación del algoritmo Fuzzy C-Means:

```

# Instalar fuzzy
# !pip install scikit-fuzzy
import skfuzzy as fuzz

```

```
# Definiendo una lista de posibles valores de m
m_values = np.arange(1.1, 3.1, 0.1)

# Inicializando listas para almacenar las metricas
silhouette_scores_fuzzy = []
davies_bouldin_scores_fuzzy = []

# Clustering fuzzy
for m in m_values:
    cntr, u, u0, d, jm, p, fpc = fuzz.cluster.cmeans(X.T, 3, m,
    ↪ error=0.005, maxiter=1000)
    labels = np.argmax(u, axis=0)

# Silhouette
silhouette_scores_fuzzy.append(silhouette_score(X, labels))

# Davies bouldin
dunn_index = davies_bouldin_score(X, labels)
davies_bouldin_scores_fuzzy.append(dunn_index)

# Buscando el valor optimo de m basado en silhouette
optimal_index = np.argmax(silhouette_scores_fuzzy)
optimal_m = m_values[optimal_index]

# Clustering con el valor optimo
optimal_cntr, optimal_u, optimal_u0, optimal_d, optimal_jm,
    ↪ optimal_p, optimal_fpc = fuzz.cluster.cmeans(X.T, 3,
    ↪ optimal_m, error=0.005, maxiter=1000)
optimal_labels = np.argmax(optimal_u, axis=0)

# Graficando los resultados
plt.figure(figsize=(10, 4))
plt.subplot(121)
plt.scatter(X.iloc[:, 1], X.iloc[:, 2], c=wine.target,
    ↪ cmap='viridis')
plt.title('Clasificación real')
plt.subplot(122)
plt.scatter(X.iloc[:, 1], X.iloc[:, 2], c=optimal_labels,
    ↪ cmap='viridis')
plt.title('Clustering Fuzzy C-means')
```

```
plt.show()

# Imprimiendo el valor optimo de m
print(f"Valor optimo de m: {optimal_m}")

#Graficando Silhouette y Davies bouldin
plt.figure(figsize=(8, 4))
plt.plot(m_values, silhouette_scores_fuzzy, 'bo-',
        ↪ label='Puntuación de silueta')
plt.plot(m_values, davies_bouldin_scores_fuzzy, 'ro-',
        ↪ label='Indice de Dunn')
plt.xlabel('Valor de m')
plt.ylabel('Metrica')
plt.title('Evaluacion de Clustering Fuzzy C-means')
plt.legend()
plt.show()
```

Clustering jerárquico

Es un método de agrupamiento que crea una jerarquía de clusters, donde se agrupan en subgrupos más pequeños o se fusionan en grupos más grandes. No necesita un número de clusters. El algoritmo de clustering jerárquico puede ser de dos tipos: aglomerativo y divisivo.

1. Clustering jerárquico aglomerativo: Inicia con cada punto de dato como un cluster individual. Luego, fusiona los clusters más cercanos. Repite este proceso hasta que todos los puntos estén en un solo cluster o se cumpla un criterio de detención.
2. Clustering jerárquico divisivo: Comienza con todos los puntos en un solo cluster para luego dividirlo en subclusters más pequeños. Esto se repite hasta que cada dato esté en su propio cluster o se cumpla un criterio de detención.

```
from sklearn.cluster import AgglomerativeClustering
from scipy.cluster.hierarchy import linkage, dendrogram

# Clustering jerarquico con diferentes metodos de enlace.
↪ Para ello se crea una lista
linkage_methods = ['ward', 'complete', 'average', 'single']

plt.figure(figsize=(12, 8))
for i, method in enumerate(linkage_methods):
```

```

# Realizando clustering jerárquico con el metodo de enlace
↪ actual
clustering = AgglomerativeClustering(n_clusters=3,
↪ linkage=method)
labels = clustering.fit_predict(X)
sc = silhouette_score(X, labels)

# Graficando los puntos en un diagrama de dispersion para las
↪ dos primeras columnas
plt.subplot(2, 2, i+1)
plt.scatter(X.iloc[:, 0], X.iloc[:, 1], c=labels,
↪ cmap='viridis')
plt.xlabel('Característica 1')
plt.ylabel('Característica 2')
plt.title(f'Clustering jerárquico - Enlace: {method} -
↪ s-score: {sc:.2f}')

plt.tight_layout()
plt.show()

```

El criterio de enlace determina cómo se mide la distancia entre los clusters. Algunos de ellos son:

- Enlace completo (Complete linkage): mide la distancia máxima entre todos los pares de puntos de datos de distintos clusters.
- Enlace simple (Single linkage): mide la distancia mínima entre todos los pares de puntos de diferentes clusters.
- Enlace promedio (Average linkage): devuelve la distancia promedio entre todos los pares de puntos de diferentes clusters.
- Enlace de Ward (Ward's linkage): minimiza la suma de las diferencias cuadradas dentro del cluster al fusionar dos clusters.

Dendograma

Un dendograma es una gráfica que muestra la estructura de los clusters formados durante el proceso de agrupamiento.

Se puede graficar un dendograma mediante el siguiente código:

```

# Graficando los dendrogramas
plt.figure(figsize=(12, 4))

```

```

for i, method in enumerate(linkage_methods):
# Clustering jerarquico con el metodo de enlace actual
Z = linkage(X, method)

# Convirtiendo la matriz de enlace a tipo float
Z = Z.astype(float)

# Graficando el dendrograma
plt.subplot(1, 4, i+1)
dendrogram(Z, labels=wine.target)
plt.xlabel('Indice de muestra')
plt.ylabel('Distancia')
plt.title(f'Dendrograma - Enlace: {method}')

plt.tight_layout()
plt.show()

```

Finalmente, se puede analizar el mejor cluster obtenido:

```

clustering = AgglomerativeClustering(n_clusters=3,
↪ linkage='ward')
labels = clustering.fit_predict(X)

# Analizando los clusters obtenidos
for cluster in range(3):
    cluster_indices = np.where(labels == cluster)[0]
    cluster_samples = wine.data[cluster_indices]
    cluster_target = wine.target[cluster_indices]
    cluster_name = wine.target_names[cluster]
    print(f'Cluster {cluster}: {cluster_name}')
    print(f'Numero de muestras: {len(cluster_samples)}')
    print(f'Caracteristicas mas repres:
↪ {np.mean(cluster_samples,
axis=0)}')
    print(f'Etiquetas reales en cluster:
↪ {np.unique(cluster_target)}')
    print('---')

```

Reducción de dimensionalidad

8.2.2. Aprendizaje reforzado

En este modelo se realizan predicciones mediante recompensas o penalizaciones dependiendo de las acciones de un agente dentro de un entorno. Estas retroalimentaciones le permiten aprender a tomar mejores decisiones, creando un conjunto de reglas o política que define la mejor estrategia para obtener recompensas.

8.2.3. Inteligencia artificial generativa

Es una clase de modelos que crea contenido por medio de las entradas de un usuario. Ejemplo: crear imágenes, contar chistes, resumir artículos. Estos modelos se entrenan inicialmente con un enfoque no supervisado, donde aprende a imitar datos con los que se entrena. En algunas ocasiones, el modelo se entrena más mediante el aprendizaje supervisado o por refuerzo dependiendo de las tareas que se le pida realizar.

8.3. Ensamblés

En Machine Learning, los ensambles son métodos que mezclan modelos para obtener una mejor predicción. Los ensambles toman como input los datos de entrenamiento de un dataset para someterlos a múltiples modelos, donde cada uno entrega su estimación y, a partir de esto, determinar la clasificación para cada observación del conjunto de entrenamiento por medio de una votación por mayoría. Existen varios tipos de ensambles, pero solo se revisarán dos:

1. Ensamblés Bagging
2. Ensamblés Random Forest

En el contexto de los ensambles, un clasificador es un tipo de modelo que se usa para predecir la categoría a la que pertenecen los datos, con el objetivo de asignar una etiqueta de clase a cada instancia. Por otro lado, un estimador es un modelo que se utiliza para predecir un valor numérico.

8.3.1. Ensamblés Bagging

También conocido como Agregación Bootstrap, Bagging es una técnica que extrae varias muestras llamadas bootstrap del conjunto de entrenamiento de los datos, realizado de forma aleatoria. Luego, los modelos se entrenan por separado

para hacer sus predicciones sobre el conjunto de prueba y combinarlas para obtener una predicción final.

Bagging se desempeña mejor cuando los modelos generan problemas de alta varianza.

El siguiente código muestra un ejemplo de un modelo Bagging:

```
from sklearn.ensemble import BaggingRegressor

RSTATE = 23124
model = BaggingRegressor(base_estimator=LinearRegression(),
    ↪ n_estimators=100, oob_score=True, random_state=RSTATE,
    ↪ n_jobs=-1)
```

donde:

- **base_estimator**: estimador. Por defecto, los estimadores base son árboles de decisión.
- **n_estimators=100**: señala el número de estimadores (modelos base) en el conjunto. En este caso, se utilizan 100 estimadores base.
- **oob_score=True**: calcula el rendimiento OOB (out-of-bag score).
- **random_state=RSTATE**: se establece la semilla para la reproducibilidad del modelo y obtener los mismos resultados en cada ejecución.
- **n_jobs=-1**: indica el número de núcleos de CPU que se utilizarán para entrenar los modelos base de forma paralela. En este caso, se utiliza -1 para utilizar todos los núcleos disponibles.

De esta forma, la estimación de R^2 con datos OOB se obtendría mediante:

```
model.oob_score_
```

8.3.2. Ensamblas Random Forest

Random Forest está compuesto únicamente por estimadores de árboles de decisión. Este modelo sigue la misma estructura que Bagging, sin embargo, agrega un componente de aleatoriedad en el que se selecciona aleatoriamente un subconjunto de atributos en cada nodo de un árbol.

Random Forest supera el problema de Bagging de estimadores altamente correlacionados.

Tanto en Random Forest como en Bagging, se genera una bolsa de observaciones no seleccionadas en los procesos de muestreo bootstrap para cada clasificador,

llamados OOB (Out Of Bag). Estos datos se utilizan para estimar el error cometido en cada clasificador. Este error del ensamble se obtiene promediando todos los errores de cada clasificador.

El siguiente código muestra un ejemplo de un modelo Random Forest:

```
from sklearn.ensemble import RandomForestRegressor
```

```
model = RandomForestRegressor(max_features='sqrt',  
    ↪ random_state=RSTATE, oob_score=True, n_jobs=-1)
```

el cual posee los mismos argumentos que Bagging, menos `max_features='sqrt'`. Este controla la cantidad máxima de características igual a la raíz cuadrada del total de características de los datos. Otros valores que puede tomar es `None`, donde el algoritmo considera todas las características en cada división.

Capítulo 9

Redes neuronales

Las redes neuronales se inspiran en el sistema neuronal biológico del cerebro. Una neurona biológica es una célula que recibe señales por medio de las dendritas, un núcleo que procesa la información y transmite los impulsos por el axón a otras neuronas.

9.1. Perceptrón

El perceptrón es una de las primeras formas de red neuronal de aprendizaje supervisado. Consiste en un algoritmo que toma un conjunto de entradas (X), las que multiplica por sus pesos (w) y calcula una suma ponderada junto el bias (u), para luego someter el resultado a una función de activación ($f(u)$) que asigna un valor de 1 si la suma ponderada entre pesos y entradas supera un umbral, o 0 en el caso contrario. La función de activación siempre estará al final de la red neuronal y corresponde a la salida del perceptrón.



Figura 9.1: Esquema de un perceptrón

En las redes neuronales, se diferencian las capas de entrada, ocultas y de salida por su posición dentro del modelo.

1. La capa de entrada (input layer) es la que recibe los datos y se determina con el parámetro `Input(shape=())`. No tiene neuronas ni función de activación, solo especifica la forma de los datos.
2. En las capas ocultas (hidden layer) reciben los valores de la capa de entrada, ponderándose por los pesos. Se definen las neuronas y las funciones de activación, y se sitúan entre la capa de entrada y la de salida.
3. La última capa de la red es la de salida (output layer), la cual produce la salida final del modelo, es decir, genera la predicción.

Por lo tanto, luego de recibir los datos en la capa de entrada, las capas ocultas harán una suma ponderada de las entradas y los pesos más el bias, representada en la siguiente ecuación:

$$u = \left(\sum_{i=0}^N x_i w_i \right) + b_0 \quad (9.1)$$

Luego, este valor pasa por la función de activación, que lo transforma en un valor de salida. Las funciones más conocidas son:

1. Función escalón
2. Función sigmoideal: se usa principalmente para problemas de clasificación binaria.

3. Función rectificadora (ReLU): activa la neurona solo si la entrada está por sobre cero. Si este valor es menor a cero, la salida será cero. De lo contrario, aumentará la salida de forma lineal.
4. Función tangente hiperbólica
5. Función lineal
6. Función logística multiclase (softmax): se usa para problemas de clasificación multiclase.

La función de pérdida evalúa qué tan bien realiza la predicción el modelo comparando los datos de entrenamiento con los reales. Esto se usa para ajustar los pesos en la red. La elección de la función de pérdida depende del problema:

1. Error cuadrático medio: se usa para problemas de regresión, es decir, en la predicción de valores numéricos.
2. Entropía cruzada binaria: se usa para problemas de clasificación binaria.
3. Entropía cruzada categórica: se usa para problemas de clasificación multiclase.

Los perceptrones conectados a otros perceptrones que forman capas se le conoce como red neuronal:

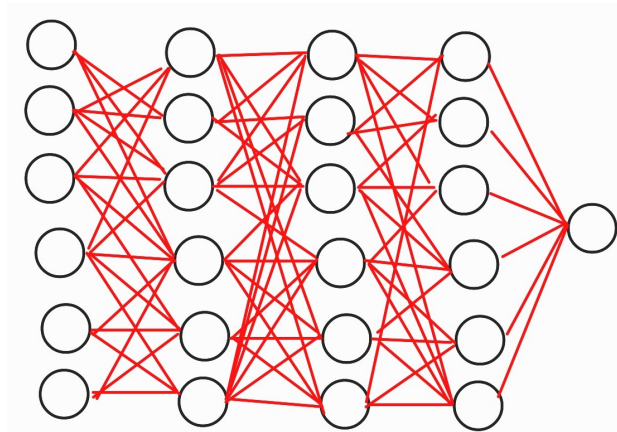


Figura 9.2: Red neuronal

Ejemplo de código de una red neuronal feedforward o secuencial:

```

import tensorflow as tf
import keras
from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import SGD # Gradiente Descendente
↳ estocastico

X = df.drop('target', axis=1)
y = df['target']
# Dividiendo el set
X_train, X_test, y_train, y_test = train_test_split(X, y,
↳ test_size=.33, random_state=23)

# Creando un modelo secuencial
model_sec = Sequential()
# Agregando capa de entrada tipo densa (las neuronas se
↳ conectan con todas las neuronas de la siguiente capa)
model_sec.add(
    Dense(
        units=1, # Neuronas
        activation='linear', # Funcion de activacion
        use_bias=True # Considerando un bias
    )
)
# Resumen
model_sec.summary()

# Compilando con optimizador descenso del gradiente
↳ estocastico y funcion de perdida error cuadratico medio
model_sec.compile(optimizer=SGD(learning_rate=0.01),
↳ loss='mean_squared_error')

# Entrenando el modelo con 100 epocas
model_sec.fit(X_train, y_train, epochs=100, verbose=0)

# Midiendo el rendimiento
y_pred = model_sec.predict(X_test).ravel()

print('\nMétricas conjunto de test')
print(f'r2 = {r2_score(y_test, y_pred)}')
```

```

print(f'RMSE = {mean_squared_error(y_test, y_pred,
    ↪ squared=False)}')
print()
# Graficando
plt.scatter(X_test, y_test, label='Datos reales')
plt.plot(X_test, y_pred, ls='--', color='gray', lw=1.5,
    ↪ label='Prediccion')
plt.xlabel('Rut')
plt.ylabel('Edad')
plt.legend();

```

Ejemplo de una red neuronal con una capa oculta:

```

# Modelo red neuronal feedforward=Sequencial
model_sec = Sequential()

# Agregando capa de entrada
model_sec.add(keras.Input(shape=(X_train.shape[1],)))

# Agregando capa oculta
model_sec.add(Dense(
    units=20,
    activation='sigmoid',
    use_bias=True
))

# Agregando capa de salida
model_sec.add(Dense(
    units=1,
    activation='sigmoid',
    use_bias=True
))

```

9.2. Resumen

Machine Learning				
Aprendizaje supervisado		Aprendizaje no supervisado		
Clasificación	Regresión	Clustering	Reducción de dimensionalidad	Asociación
Regresión logística	Regresión lineal	K-Means	Análisis de componentes principales (PCA)	Apriori
Árboles de decisión	Regresión Ridge	DBSCAN	Linear Discriminant Analysis (LDA)	FP-Growth
SVM	Regresión Lasso	Cluster jerárquico	Non-negative Matrix Factorization (NMF)	
KNN	Regresión ElasticNet			
Naive Bayes	Árboles de regresión			
Métricas de evaluación				
Accuracy (Exactitud)	MSE	Coeficiente de Silhouette	Varianza explicada	Soporte
Precision (Precisión)	MAE	Índice de Davies-Bouldin		Confianza
Recall	R cuadrado			Lift
F1-Score				
Matriz de confusión				
Curva ROC AUC				

Capítulo 10

Git y GitHub

10.1. Git

Se puede hacer correr desde la cmd con el comando `git`.

Lista de comandos:

- `git config --global user.name "nom-usuario"`: se configura el nombre de usuario que va entre las comillas. Para configurar el email se debe poner `git config --global user.email` seguido del email entre comillas.
- `git config user.name`: para ver el nombre de usuario. Para ver el email es `git config user.email`

10.1.1. Repositorios

Para abrir Git en el proyecto, se debe hacer clic derecho Git Bash Here en la carpeta donde se ubica dicho proyecto. Una vez abierto, se puede crear un nuevo repositorio con `git init` en la rama Master o main.

Con el comando `git add nom-archivo.extension` se puede enviar archivos al repositorio. Si se quieren subir todos los archivos, se debe usar el comando `git add ..`

Una vez que se hayan realizado todos los cambios en los archivos, se le puede añadir un comentario con `git commit -m "Mensaje"` y se suben con `git push origin nom-rama`.

Si los archivos fueron modificados en GitHub, se pueden bajar al computador con `git pull`.

Lista de comandos:

1. `git status`: ver el estado del proyecto.

2. `git commit -m "mensaje"`: se utiliza para tomar una instantánea del proyecto y dejar un mensaje con los cambios realizados en este.
3. `git log`: para ver todos los mensajes y cambios que han realizado los autores.
4. `git log --stat`: sirve para visualizar los commits y muestra un pequeño resumen de lo modificado, como la cantidad de líneas e inserciones.
5. `git log --oneline`: muestra un código que identifica al commit.
6. `git checkout codigo-commit`: actualiza el archivo para que coincida con el commit señalado en el código-commit. Una vez que se vuelve a un commit anterior, los otros que están en medio se borran. Por esta razón se recomienda trabajar con ramas.
7. `git rm --cached nom-archivo.extension`: deja de rastrear un archivo sin eliminarlo del directorio de trabajo. Si se quiere hacer con todos los archivos: `git rm --cached ..`
8. `git clone direccion-del-repositorio`: clonar un repositorio en el computador.
9. `touch nom_archivo.extension`: crea un archivo.
10. `clear`: limpiar la pantalla.

10.1.2. .gitignore

El archivo `.gitignore` se crea con fines de que Git ignore ciertos archivos. Para crearlo, se puede hacer mediante la consola de Git con `touch .gitignore`. Luego de crearlo, se puede abrir con cualquier aplicación y escribir los archivos que se desean ignorar. Se deben escribir los nombres de los proyectos, como `mi_proyecto.txt`, o se pueden ignorar todas las extensiones, como `*.txt`.

10.1.3. Ramas

Una rama en Git es una versión del código de un proyecto, las cuales ayudan a mantener el orden en el control de versiones. El control de versiones es la práctica de gestionar los cambios que se realizan sobre un código. Cuando se habla del estado en el que se encuentra, se refiere a la versión, revisión o edición de este. Cada repositorio en Git comienza con una rama principal predeterminada llamada `master`.

Git almacena una serie de instantáneas al ejecutar el comando commit junto con los metadatos de quién lo haya modificado.

Comandos:

- `git branch`: devuelve la cantidad de ramas que tiene un proyecto.
- `git checkout rama`: cambia de rama.
- `git checkout -b nombre-rama`: crear una rama.
- `git merge nombre-rama`: unir ramas. El código de nombre-rama se une al de la rama en la que se está trabajando.
- `git branch -d nombre-rama`: eliminar una rama solo si la rama se ha fusionado. De lo contrario, usar `-D` que elimina la rama independientemente de su estado de fusión.
- `git branch -d nombre-remoto/nombre-rama`: eliminar una rama remota de VSCode. Generalmente, `nombre-remoto` es `origin`.

Capítulo 11

Glosario de términos de programación

La siguiente lista muestra las traducciones al español de los términos de programación en inglés más usados.

1. Array: Arreglo, listas.
2. Dataset: Conjunto de datos.
3. Debugging: Depuración.
4. Default: Predeterminado.
5. Delete: Borrar.
6. Drop: Borrar.
7. Loop: Bucle.
8. Return: Retornar.
9. String: Cadena.
10. Database: Base de datos.
11. Null: Nulo.
12. Fork: Bifurcación, clonación de un programa.

Bibliografía

- [1] <https://www.javatpoint.com/postgresql-tutorial>
- [2] <https://git-scm.com/doc>
- [3] <https://git-scm.com/book/es/v2>
- [4] https://help.tableau.com/current/pro/desktop/es-es/dashboards_create.htm
- [5] <https://docs.python.org/es/3/index.html>
- [6] https://facebook.github.io/prophet/docs/quick_start.html#python-api