**i-**

**Importar archivos CSV**

df = pd.read\_csv("diabetes.csv")

**Importar archivos de texto:**

Argumentos separadores “sep”:

sep=”,” -> Coma

sep=”\s” -> Espacio en blanco

sep = "\t" -> El tabulador

sep = “:” -> Los dos puntos

df = pd.read\_csv("diabetes.txt", sep="\s")

**Importar archivos Excel (una sola hoja)**

df = pd.read\_excel('diabetes.xlsx')

**Importar archivos Excel (varias hojas)**

df = pd.read\_excel('diabetes\_multi.xlsx', sheet\_name=1)

**Importar archivo JSON**

df = pd.read\_json("diabetes.json")

**Salida de datos en pandas**

**Pasar un DataFrame a un archivo CSV**

df.to\_csv("diabetes\_out.csv", index=False)

**Pasar un DataFrame a un archivo JSON**

df.to\_json("diabetes\_out.json")

**Pasar un DataFrame a un fichero de texto**

df.to\_csv('diabetes\_out.txt', header=df.columns, index=None, sep=' ')

**Pasar un DataFrame a un fichero Excel**

df.to\_excel("diabetes\_out.xlsx", index=False)

**Visualización y comprensión de DataFrames con pandas**

**Cómo ver los datos mediante .head() y .tail()**

df.head()

df.tail(n = 10)

**Comprender los datos con .describe()**

df.describe()

df.describe(percentiles=[0.3, 0.5, 0.7])

**aislar y excluir tipos de datos específicos**

df.describe(include=[int])

df.describe(exclude=[int])

df.describe().T

**Comprender los datos con .info()**

df.info(show\_counts=True, memory\_usage=True, verbose=True)

**Comprender tus datos con .shape**

df.shape # Get the number of rows and columns

df.shape[0] # Get the number of rows only

df.shape[1] # Get the number of columns only

**Obtener todas las columnas y nombres de columnas**

df.columns

**Comprobación de valores perdidos en pandas con .isnull()**

.copy() -> hace una copia del DataFrame original. para garantizar que cualquier cambio en la copia no se refleje en el DataFrame original.

.loc -> puedes establecer las filas

df2 = df.copy()

df2.loc[2:5,'Pregnancies'] = None

df2.head(7)

**Puedes comprobar si falta algún elemento de un DataFrame utilizando el método .isnull().**

df2.isnull().head(7)

**Dado que a menudo es más útil saber cuántos datos faltan, puedes combinar .isnull() con .sum() para contar el número de nulos de cada columna.**

df2.isnull().sum()

**También puedes hacer una suma doble para obtener el número total de nulos del DataFrame**

df2.isnull().sum().sum()

**Rebanar y extraer datos en pandas**

**Aislar una columna con [ ]**

df['Outcome']

**Aislar dos o más columnas con [[ ]]**

df[['Pregnancies', 'Outcome']]

**Aislar una fila con [ ]**

df[df.index==1]

**Aislar dos o más filas con [ ]**

df[df.index.isin(range(2,10))]

**Utilizar .loc[] y .iloc[] para recuperar filas**

.loc[] -> utiliza una etiqueta para señalar una fila

.iloc[] ->  utiliza la posición numérica.

df2.index = range(1,769)

df2.loc[1] ->  devuelve un pandas Series en lugar de un DataFrame. El 1 representa el índice de la fila (etiqueta)

df2.iloc[1] -> mientras que el 1 en .iloc[] es la posición de la fila (primera fila).

**También puedes obtener varias filas indicando un rango entre corchetes**

df2.loc[100:110]

**También puedes subconjuntar con .loc[] y .iloc[] utilizando una lista en lugar de un rango.**

df2.loc[[100, 200, 300]] #100, 200, 300

df2.iloc[[100, 200, 300]] #101, 201, 301

**También puedes seleccionar columnas concretas junto con las filas. Aquí es donde .iloc[] es diferente de .loc[]: requiere la ubicación de las columnas y no las etiquetas de las columnas.**  
df2.loc[100:110, ['Pregnancies', 'Glucose', 'BloodPressure']]

df2.iloc[100:110, :3]

**Puedes actualizar/modificar determinados valores utilizando el operador de asignación =**

df2.loc[df['Age']==81, ['Age']] = 80

**Corte condicional (que se ajusta a determinadas condiciones)**

df[df.NombreDeLaColumna == 122]

df.loc[df['BloodPressure'] > 100, ['Pregnancies', 'Glucose', 'BloodPressure']]

**Limpieza de datos con pandas**

**Técnica n.º 1 para tratar los datos que faltan: Eliminación de valores perdidos**

df3 = df2.copy()

df3 = df3.dropna()

df3.shape

(764, 9) # this is 4 rows less than df2

df3 = df2.copy()

df3.dropna(inplace=True, axis=1)

df3.head()

También puedes eliminar tanto las filas como las columnas con valores perdidos estableciendo el argumento how como 'all'

df3 = df2.copy()

df3.dropna(inplace=True, how='all')

### **Técnica n.º 2 para tratar los datos que faltan: Sustitución de valores perdidos**

df3 = df2.copy()

# Get the mean of Pregnancies

mean\_value = df3['Pregnancies'].mean()

# Fill missing values using .fillna()

df3 = df3.fillna(mean\_value)

**Cómo tratar los datos duplicados**

df3 = pd.concat([df2, df2]) -> para unir dos veces el mismo DataFrame

df3.shape

df3 = df3.drop\_duplicates() -> Para quitar duplicados

df3.shape

**Renombrar columnas**

df3.rename(columns = {'DiabetesPedigreeFunction':'DPF'}, inplace = True)

df3.head()

**Análisis de datos en pandas**

**Operadores de resumen (media, moda, mediana)**

Como has visto antes, puedes obtener la media de cada valor de columna utilizando el método .mean().

df.mean()

Un modo puede calcularse de forma similar utilizando el método .mode().

df.mode()

Del mismo modo, la mediana de cada columna se calcula con el método .median()

df.median()

**Crear nuevas columnas basadas en columnas existentes**

df2['Glucose\_Insulin\_Ratio'] = df2['Glucose']/df2['Insulin']

df2.head()

**Contar utilizando .value\_counts()**

df['Outcome'].value\_counts()

**Si añades el argumento normalize, obtendrás proporciones en lugar de recuentos absolutos**

df['Outcome'].value\_counts(normalize=True)

Desactiva la ordenación automática de los resultados mediante el argumento sort (True de forma predeterminada). El orden predeterminado se basa en los recuentos en orden descendente.

df['Outcome'].value\_counts(sort=False)

También puedes aplicar .value\_counts() a un objeto DataFrame y a columnas concretas dentro de él, en lugar de solo a una columna. Aquí, por ejemplo, estamos aplicando value\_counts() en df con el argumento subconjunto, que recibe una lista de columnas.

df.value\_counts(subset=['Pregnancies', 'Outcome'])

**Agregar datos con .groupby() en pandas**

df.groupby('Outcome').mean()

**.groupby() permite agrupar por más de una columna pasando una lista de nombres de columnas, como se muestra a continuación.**

df.groupby(['Pregnancies', 'Outcome']).mean()

Se puede utilizar cualquier método de resumen junto con .groupby(), incluido .min(), .max(), .mean(), .median(), .sum(), .mode() y más.

**Tablas dinámicas**

pd.pivot\_table(df, values="BMI", index='Pregnancies',

columns=['Outcome'], aggfunc=np.mean)

**Visualización de datos en pandas**

**Gráficos de líneas en pandas**

df[['BMI', 'Glucose']].plot.line()

Puedes seleccionar la elección de colores utilizando el argumento color.

df[['BMI', 'Glucose']].plot.line(figsize=(20, 10),

color={"BMI": "red", "Glucose": "blue"})

Todas las columnas de df también pueden representarse en diferentes escalas y ejes utilizando el argumento subplots.

df.plot.line(subplots=True)

**Gráficos de barras en pandas**

df['Outcome'].value\_counts().plot.bar()

**Gráficos de caja en pandas**

df.boxplot(column=['BMI'], by='Outcome')