

| Python Cheat Sheet 1 |
|--|
| <div><div><div><div><div></div></div><div><div></div></div></div><div><div><div></div></div><div><div></div></div></div><div><div><div></div></div><div><div></div></div></div></div></div> <div>Variables ampliadas por text (CONCATENATION)</div> |
| <div><div><div><div><div></div></div><div><div></div></div></div><div><div><div></div></div><div><div></div></div></div><div><div><div></div></div><div><div></div></div></div></div></div> <div>Para encadenar texto</div> <div><pre>categorial = "verde" color_detalle = categorial + ' ' + 'oscuro' print(categorial + ' oscuro') print(categorial, 'oscuro')</pre></div> |
| <div><div><div><div><div></div></div><div><div></div></div></div><div><div><div></div></div><div><div></div></div></div><div><div><div></div></div><div><div></div></div></div></div></div> <div>type() and isinstance()</div> <div><pre>float/int/str(variable) cambia el tipo de data/type type(variable) devuelve: class 'float/int/str' isinstance(variable, float/int/str) comprobar el tipo de dato (devuelve True/False)</pre></div> |
| <div><div><div><div><div></div></div><div><div></div></div></div><div><div><div></div></div><div><div></div></div></div><div><div><div></div></div><div><div></div></div></div></div></div> <div>Operaciones Algebraicas</div> <div><div><div><div><div></div></div><div><div></div></div></div><div><div><div></div></div><div><div></div></div></div><div><div><div></div></div><div><div></div></div></div><div><div><div></div></div><div><div></div></div></div></div><div><div><div><div></div></div><div><div></div></div></div><div><div><div></div></div><div><div></div></div></div></div><div><div><div><div></div></div><div><div></div></div></div><div><div><div></div></div><div><div></div></div></div></div></div> <div><div><div><div></div></div><div><div></div></div></div><div><div><div></div></div><div><div></div></div></div></div> <div><div><div><div></div></div><div><div></div></div></div><div><div><div></div></div><div><div></div></div></div></div> |

| Python Cheat Sheet 4 |
|---|
| |
| Pandas |
| |
| Union de datos |
| <div><div><div><div><div><div></div></div></div><div><div><div></div></div><div><div></div></div></div><div><div><div></div></div><div><div></div></div></div><div><div><div></div></div><div><div></div></div></div></div></div></div> <div>.concat() unir dataframes con columnas en comun df_union = pd.concat([df1, df2, df3], axis=b, join = 'inner/outer', ignore_index = True/False) parametros: axis = 0 une por columnas - los dataframes van uno encima del otro; las columnas tienen que ser de formatos compatible axis = 1 une por filas - los dataframes van uno al lado del otro; los datos deben ser relacionados para que tenga sentido join = 'inner' solo se quedan elementos que aparecen en todos los dataframes join = 'outer' se queda todo los datos de todos los dataframes ignore_index = True/False por defecto es False; si es True no usa las índices para la union (por ejemplo para union por el axis 0) .merge() unir las columnas de un dataframe a otro df_nuevo = df1.merge(df2, on = 'columna') inner merge df_nuevo = pd.merge(left = df1, right = df2, how='left', left_on = 'columna_df1', right_on = 'columna_df2') left merge parametros: how = 'left' 'right' 'outer' 'inner' 'cross' on = columna [columna1, columna2, etc] si las columnas se llaman igual en los dos dataframes left_on = columna_df1 right_on = columna_df2 para especificar por donde hacer el merge suffixes = ['left', 'right'] por defecto nada, el sufijo que apareciera en columnas duplicadas .join() unir dataframes por los índices df_nuevo = df1.join(df2, on = 'columna', how = 'left') inner merge parametros: how = 'left' 'right' 'outer' 'inner' por defecto left on = columna la columna o índice por el que queremos hacer el union; tienen que tener el mismo nombre en los dos dataframes lsuffix = 'string' rsuffix = 'string' por defecto nada, el sufijo que apareciera en columnas duplicadas</div> |
| Group By |
| <div><div><div><div><div><div></div></div></div><div><div><div></div></div><div><div></div></div></div><div><div><div></div></div><div><div></div></div></div></div></div></div> <div>df_groupby = df.groupby("columna_categoria") crea un objeto DataFrameGroupBy; agrupa los valores segun las categorías de los valores de la columna indicada (o múltiples columnas en una lista) df_groupby.ngroups devuelve el numero de grupos df_groupby.groups devuelve un diccionario donde los keys son las categorías y los valores son listas de los índices de cada elemento en la categoría df_grupo1 = df_groupby.get_group("grupo1") devuelve un dataframe con los resultados de un grupo (la categoria indicada como grupo1) Cálculos con groupby: df_nuevo = df.groupby("columna_categoria").mean() devuelve un dataframe con la media de todas las columnas de valores numéricos, por categoría df_nuevo = df.groupby("columna_categoria")["columna1"].mean() devuelve un dataframe con la media de la columna especificada count() número de observaciones no nulas describe() resumen de los principales estadísticos sum() suma de todos los valores mean() media de los valores df_nuevo = df.groupby("columna_categoria", dropna = False) ["columna_valores"].agg([nombre_columna = 'estadistico1', nombre_columna2 = 'estadistico2']) añade columnas con los cálculos de los estadísticos especificados dropna = False para tener en cuenta los Nan en los cálculos (por defecto es True)</div> |

| Subsets: loc e iloc |
|---|
| <div><div><div><div><div><div></div></div></div><div><div><div></div></div><div><div></div></div></div><div><div><div></div></div><div><div></div></div></div></div></div></div> <div>df.loc["etiqueta_fila", "etiqueta_columna"] devuelve el contenido de un campo en una columna de una fila df.loc["etiqueta_fila",:] devuelve los valores de todas las columnas de una fila df.loc[:, "etiqueta_columna"] devuelve los valores de todas las filas de una columna df.iloc[indice_fila, indice_columna] devuelve el contenido de un campo en una columna de una fila df.iloc[indice_fila, :] devuelve los valores de todas las columnas de una fila df.iloc[:, indice_columna] devuelve el contenido de un campo en una columna de una fila df.loc[[lista_etiquetas_filas], [lista_etiquetas_columnas]] devuelve el contenido de varias filas / varias columnas df.loc[[lista_indices_filas], [lista_indices_columnas]] devuelve el contenido de varias filas / varias columnas - se puede usar los índices/rangos de las listas [start:stop:step] dentro de los loc/iloc df.loc[df.etiqueta > x] seleccionar datos basado en una condición usando operadores comparativos df.loc[(df.etiqueta > x) & (df.etiqueta == y)] seleccionar datos que tienen que cumplir las dos condiciones (and) df.loc[(df.etiqueta > x) (df.etiqueta == y)] seleccionar datos que tienen que deben cumplir una de las dos condiciones (or) df.iloc[list(df.etiqueta > x), :] iloc no acepta una Serie booleana; hay que convertirla en lista variable_df.head(n) devuelve las n primeras filas del df, o 5 por defecto</div> |
| Filtrados de datos |
| <div><div><div><div><div><div></div></div></div><div><div><div></div></div><div><div></div></div></div><div><div><div></div></div><div><div></div></div></div></div></div></div> <div>Filtrado por una columna con operadores de comparación df_filtrado = df[df["nombre_columna"] == valor] extrae las filas donde el valor de la columna igual al valor dado Filtrado por multiples columnas con operadores logicos df_filtrado = df[(df["columna1"] == valor) & (df["columna2"] == valor) & (df["columna3"] > n valor)] extrae las filas donde los valores de las columnas cumplan las condiciones en parentesis df_filtrado = df[(df["columna1"] == valor) (df["columna1"] == valor) extrae las filas donde los valores de las columnas cumplan con una condición u otra df_filtrado = ~(df[df["columna1"] == valor]) extrae las filas donde los valores de las columnas NO cumplan con la condición</div> |

| Filtrados de datos |
|--|
| <div><div><div><div><div><div></div></div></div><div><div><div></div></div><div><div></div></div></div><div><div><div></div></div><div><div></div></div></div></div></div></div> <div>Metodos de pandas de filtrar df_filtrado = df[df["nombre_columna"].isin(iterable)] extrae las filas cuyas valores de la columna nombrada están en el iterable (una lista, serie, dataframe o diccionario) df_filtrado= df[df["nombre_columna"].str.contains (patron, regex = True, na = False)] extrae las filas cuyas valores de la columna nombrada contienen el patron de regex df_filtrado = df[df["nombre_columna"].str.contains ("substring", case = False, regex = False)] extrae las filas cuyas valores de la columna nombrada contienen el substring, no siendo case sensitive df_filtrado = df[df["nombre_columna"].str.contains ("substring", case = False, regex = False)] extrae las filas cuyas valores de la columna nombrada contienen el substring, no siendo case sensitive df[pd.notnull(df["nombre_columna"])] devuelve las filas que no tiene valores nulos en la columna especificada</div> |
| Cambiar columnas |
| <div><div><div><div><div><div></div></div></div><div><div><div></div></div><div><div></div></div></div><div><div><div></div></div><div><div></div></div></div></div></div></div> <div>lista_columnas = df.columns.to_list() crea una lista de los nombres de las columnas del dataframe df.set_index(["nombre_columna"], inplace = True) establece el índice utilizando uno o mas columnas; puede sustituir o ampliar un índice existente inplace = True los cambios sobrescriben sobre el df * cuando una columna se cambia a índice ya no es columna * df.reset_index(inplace = True) quitar una columna como índice para que vuelva a ser columna; crea un dataframe de una serie Renombrar columnas df.rename(columns = {"nombre_columna": "nombre_nueva"}, inplace = True) cambia los nombres de una o mas columnas ejemplo de dict comprehension para crear diccionario sobre las columnas existentes de un dataframe: diccionario = {col : col.upper() for col in df.columns} df.rename(columns = diccionario, inplace = True) cambia los nombres de las columnas según el diccionario Eliminar columnas df.drop(columns = ["columna1", "columna2"], axis = b, inplace=True) eliminar una o mas columnas o filas segun lo que especificamos Reordenar columnas df = df.reindex(columns = lista_reordenada) cambia el orden de las columnas del dataframe segun el orden de la lista reordenada</div> |

| Crear columnas |
|---|
| <div><div><div><div><div><div></div></div></div><div><div><div></div></div><div><div></div></div></div><div><div><div></div></div><div><div></div></div></div></div></div></div> <div>Creacion de ratios df["columna?ratio"] = df.apply(lambda df: df["columna1"] / df["columna2"], axis = 1) Creacion de porcentajes def porcentaje(columna1, columna2): return (columna1 * 100) / columna2 df["columna_%"] = df.apply(lambda df: porcentaje(df["columna1"], datos["columna2"]), axis = 1) df["nueva_columna"] = np.where(df["nombre_columna"] > n, "categoria_if_true", "categoria_if_false") crea una nueva columna basada en una condición df["nueva_columna"] = np.select(lista_de_condiciones, lista_de_opciones) crea una nueva columna con los valores basados en multiples condiciones df["columna_nueva"] = pd.cut(x = df["nombre_columna"], bins = [n,m,l..], labels = ['a', 'b', 'c']) separa los elementos de un dataframe en diferentes intervalos (n-m, m-l, etc), creando una columna nueva que indica en cual intervalo cae el valor; con labels se puede asignar un string a cada intervalo Crear columnas df["nueva_columna"] = (df["etiqueta_columna"] + x) crea una nueva columna basada en otra df = df.assign(nueva_columna= df["etiqueta_columna"] + x) crea una nueva basada en otra df = df.assign(nueva_columna= [lista_valores]) crea una nueva columna de una lista de valores *tiene que ser de la misma longitud como el número de filas del dataframe* df.insert(indice_nueva_columna, "nombre_columna", valores) crea una nueva columna en la indice indicada allow_duplicates = True parametro cuando queremos permitir columnas duplicadas (por defecto es False) Apply apply() toma una función como argumento y la aplica a lo largo de un eje del DataFrame df['columna_nueva'] = df['col_1'].apply(función) crea una columna nueva con los valores de otra columna transformados según la función indicada df['columna_nueva'] = df['col_1'].apply(lambda x: x.método() if x > 1) crea una columna nueva con los valores de otra columna transformados según la lambda indicada df['columna_nueva'] = df.apply(lambda nombre: función(nombre['columna1'], nombre['columna2']), axis = b) crea una columna nueva usando una función que coge dos parámetros (columna 1 y columna2) df.applymap(funcion, na_action=None, **kwargs) acepta y devuelve un escalar a cada elemento de un dataframe; se tiene que aplicar a todo el DataFrame df['columna'] = df['columna'].map(mapa, na_action = 'ignore') reemplaza valores de la columna según el mapa, que puede ser un diccionario o una serie; solo se puede aplicar a una columa en particular. apply() con datetime df['columna_fecha'] = df['columna_fecha'] .apply(pd.to_datetime) cambia una columna de datos tipo fecha en el formato datetime def sacar_año(x): return x.strftime("%Y") df['columna_año'] = (df['columna_fecha'] .apply (sacar_año) crea una columna nueva del año solo usando un método de la libreria datetime; ("%B") para meses</div> |

| Cambiar valores |
|---|
| <div><div><div><div><div><div></div></div></div><div><div><div></div></div><div><div></div></div></div><div><div><div></div></div><div><div></div></div></div></div></div></div> <div>Reemplazar valores basados en índices y condiciones: indices_filtrados = df.index[df["columna"] == "valor"] for indice in indices_filtrados: df["nombre_columna"].iloc[indice] = "valor_nuevo" Reemplazar valores basados en metodos NumPy: df.replace(to_replace = valor, value = valor_nuevo, inplace = True) reemplaza cierto valor por otro que especificamos df["nombre_columna"].replace(to_replace = valor, value = valor_nuevo, inplace = True) reemplaza cierto valor en una columna por otro que especificamos df[["columna1", "columna2"]] = df[["columna1", "columna2"]].replace(r"string", "string", regex=True) cambiar un patron/string por otro en multiples columnas df["nombre_columna"] = df["nombre_columna"] + x reemplaza los valores de la columna por el valor + x (o otro valor que indicamos)</div> |
| datetime |
| <div><div><div><div><div><div></div></div></div><div><div><div></div></div><div><div></div></div></div><div><div><div></div></div><div><div></div></div></div></div></div></div> <div>import datetime datetime.now() devuelve la fecha actual timedelta(n) representa una duración la diferencia entre dos instancias; n es un numero de días datetime.strptime(variable_fecha, '%Y-%m-%d') formatea la fecha al formato indicado ayer = datetime.now() - timedelta(1) ayer = datetime.strptime(ayer, '%Y-%m-%d') df["fecha"] = ayer crea una columna con la fecha de ayer</div> |

Python Cheat Sheet 5

Matplotlib

Gráficas

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

```
plt.rcParams["figure.figsize"] = (10,8)
```

```
plt.figure(figsize = (n,m))
```

inicia una grafica dibujando el marco de la figura; n es la anchura y m es la altura, en pulgadas

```
plt.show()
```

muestra la figura

Gráficas básicas

Bar plot

```
plt.bar(df["columna1"], df["columna2"])
```

crea un diagrama de barras donde los ejes son: columna1 – x, columna2 – y

Horizontal bar plot

```
plt.barh(df["columna1"], df["columna2"])
```

crea una diagramma de barras horizontales donde los ejes son: columna1 – x, columna2 – y

Stacked bar plot

```
plt.bar(x, y, label = 'etiqueta')
```

```
plt.bar(x2, y2, bottom = y, label = 'etiqueta2')
```

crea una diagrama de barras apiladas para visualizar dos variables juntas; y indica la barra de referencia

Scatter plot

```
plt.scatter(df["columna1"], df["columna2"])
```

crea una gráfica de dispersión donde los ejes son: columna1 – x, columna2 – y

Gráficas estadísticas

Histogram

```
plt.hist(x = df['columna1'], bins = n)
```

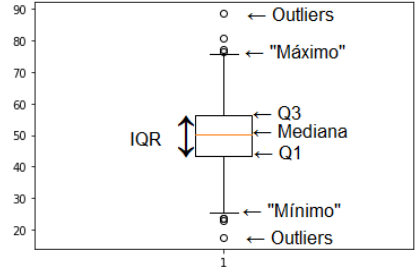
crea una histograma que muestra la frecuencias de una distribución de datos; donde x es la variable de interés y n es el número de barras

Box Plot

```
plt.boxplot(x = df['columna1'])
```

crea un diagrama de cajas para estudiar las características de una variable numerica; x es la variable de interés

- el mínimo es lo mismo que Q1 - 1.5 * IQR
- el máximo es lo mismo que Q3 + 1.5 * IQR



Pie Chart

```
plt.pie(x, labels = categorias, radius = n)
```

crea un gráfico de sectores donde x es la variable de interés (debe esta agrupado por categorias); n es el tamaño

Violin Plot

```
plt.violinplot(x, showmedians = True, showmeans = True)
```

crea un diagrama de violin donde x es la variable de interés y muestra la mediana y la media

Seaborn gráficas

Line plot

```
fig = sns.lineplot(x = 'columna1', y = 'columna2', data = df, ci = None)
```

crea una gráfica lineal donde los ejes son: columna1 – x, columna2 – y

ci = None para que no muestra el intervalo de confianza de los datos

hue = columna opcional; muestra lineas en diferentes colores por categorias segun una variable

Scatter plot

```
fig = sns.scatterplot(x = 'columna1', y = 'columna2', data = df, hue = 'columna')
```

crea una gráfica de dispersión

Swarm plot

```
fig = sns.swarmplot(x = 'columna1', y = 'columna2', data = df, hue = 'columna')
```

crea una gráfica de dispersión donde los marcadores no se solapan

Count plot

```
fig = sns.countplot(x = 'columna1', data = df, hue = 'columna')
```

crea una gráfica de barras con la cuenta de una variable categórica; se puede especificar solo una variable en la eje x o y, mas una variable opcional con hue

Histogram

```
fig = sns.histplot(x = 'columna1', data = df, hue = 'columna3', kde = True, bins = n)
```

crea una histograma que muestra la frecuencias de una distribución de datos; donde x es la variable de interés y n es el número de barras

kde = True muestra una curva de la distribucion

Box Plot

```
fig = sns.boxplot(x = 'columna1', data = df, hue = 'columna')
```

crea un diagrama de cajas; x es la variable de interés; por defecto se muestra con orientación horizontal

– usar eje y para orientación vertical

Catplot

```
fig = sns.catplot(x = 'columna1', y = 'columna2', data = df, hue = 'columna', kind = 'tipo')
```

crea una gráfica que muestra la relacion entre una variable categorica y una variable numerica

kind = 'box' | 'bar' | 'violín' | 'boxen' | 'point' por defecto es strip plot

Pairplot

```
fig = sns.pairplot(data = df, hue = 'columna', kind = 'tipo')
```

crea los histogramas y diagramas de dispersión de todas las variables numéricas de las que disponga el dataset con el que estemos trabajando; hue es opcional

kind = 'scatter' | 'kde' | 'hist' | 'reg' | 'point' por defecto es scatter

Heatmap

```
sns.heatmap(df.corr(), cmap = 'color_palette', annot = True, vmin = -1, vmax = 1)
```

crea un heatmap con una escala de colores que refleja los valores de correlacion

annot = True para que aparezcan los valores

vmin/vmax establecen la escala de color

Regplot

```
fig = sns.regplot(x = 'columna1', y = 'columna2', data = df, scatter_kws = {'color': 'blue'}, line_kws = {'color': 'blue'})
```

crea un scatterplot mas la línea de regresión; nos permite encontrar la mejor función de la recta que permite predecir el valor de una variable sabiendo los valores de otra variable

Jointplot

```
sns.jointplot(x = 'columna1', y = 'columna2', data = df, color = 'blue', kind = 'tipo')
```

crea un scatterplot o regplot con histogramas pegados en los lados para cada variable

Exportar figuras

```
plt.savefig('nombre_de_la_figura.extension')
```

Multigráficas

```
fig, ax = plt.subplots(numero_filas, numero_columnas)
```

crear una figura con multiples graficas; fig es la figura y ax es un array con subplots como elementos

se establece como es cada grafica con los indices:

```
ax[indice].tipo_grafica(detalles de la grafica)
```

```
ax[indice].set_title('titulo')
```

```
ax[indice].set_xlabel('xlabel')
```

```
ax[indice].set_ylabel('ylabel')
```

```
ax[indice].set_xlim(min, max)
```

```
ax[indice].set_ylim(min, max)
```

```
ax[indice].set_xticklabels(labels = df['column'], rotation = n)
```

para cambiar los nombres y/o la rotacion de las etiquetas de los valores en los ejes

Crear subplots en un for loop

```
fig, axes = plt.subplots(numero_filas, numero_columnas, figsize = (n, m))
```

```
axes = axes.flatten()
```

```
for col in df.columns:
```

```
fig = sns.plot(x=col, data=df, ax=axes[i])
```

Usos de los tipos de gráficas

Datos categóricos

Barras

- muestra la relación entre una variable numérica y categórica
- barplot si tienes una variable numérica
- countplot para contar registros/filas por categoría

Pie chart/quesitos

- determinación de frecuencias

Datos numéricos

Líneas

- tendencias/evolución de una o más variables numéricas (normalmente sobre un período de tiempo)

Histograma

- distribución de una variable numérica

Boxplot

-representación de las medidas de posición más usadas: mediana, IQR, outliers

Scatterplot

- muestra la relación entre dos variables numéricas

Regplot

- scatterplot con una línea de regresión

Swarmplot

- tipo de gráfica de dispersión para representar variables categóricas; evita que se solapan los marcadores

Violinplot

- para visualizar la distribución de los datos y su densidad de probabilidad

Pairplot

- para representar múltiples relaciones entre dos variables

Heatmap

- evaluar la correlación entre las variables en una matriz de correlación

Personalización

Titulos

```
plt.title(label = "titulo")
```

asignar un titulo a la gráfica

Ejes

```
plt.xlabel("etiqueta_eje_x")
```

asignar nombre al eje x

```
plt.ylabel("etiqueta_eje_y")
```

asignar nombre al eje y

```
plt.xlim([n,m])
```

establece el rango del eje x; donde n es el mínimo y m es el máximo

```
plt.ylim([n,m])
```

establece el rango del eje y; donde n es el mínimo y m es el máximo

```
fig.set(xlabel = 'etiqueta_eje_x', ylabel = 'etiqueta_eje_y')
```

asignar nombre a los ejes

```
fig.set_title('titulo')
```

asignar un titulo a la gráfica

```
fig.set_xlabel(xlabel = "etiqueta_eje_x", fontsize = n)
```

```
fig.set_ylabel(ylabel = "etiqueta_eje_y", fontsize = n)
```

```
fig.set(xticks = [1, 2, 3])
```

```
fig.set(yticks = [1, 2, 3, 4, 5])
```

```
fig.set(xticklabels = ['0%', '20%', '40%', '60%', '80%', '100%'])
```

```
fig.set(yticklabels = ['cat1', 'cat2', 'cat3'])
```

```
fig.set_xticklabels(labels = [0, 500, 1000, 1500], size=n)
```

```
fig.set_yticklabels(labels = fig.get_yticklabels(), size=n)
```

Para poner etiquetas encima de las barras

```
for indice, valor in enumerate(df ["col"]):
```

```
plt.text(valor+1, indice, valor,
```

```
horizontalalignment='left', fontsize= 16)
```

```
order = df.sort_values('columnay', ascending=False) ['columnax']
```

```
sns.set(font_scale=2)
```

```
plt.rcParams.update({'font.size': 22})
```

font size general

Legendas

```
plt.legend(labels = ['label1', 'label2', etc])
```

muestra la leyenda cuando mostramos la figura

```
plt.legend(bbox_to_anchor = (1, 1))
```

coloca la leyenda en relación con los ejes

Quitar bordes

```
fig.spines[["top", "right"]].set_visible(False)
```

Linea de tres desviaciones estandares:

```
fig.axvline(x=valor, c='color', label='valor')
```

```
fig.axvline(x=valor, c='color', label='valor')
```

Cuadrícula

```
plt.grid()
```

crea una cuadrícula al fondo de la figura; coge los parámetros:

color = "color"

linestyle = "solid" | "dashed" | "dashdot" | "dotted"

linewidth = n establece la anchura de la linea

Personalización

Colores

color = "color" establece el color de la grafica

facecolor = "color" establece el color del relleno

edgecolor = "color" establece el color de los bordes

Colores en Scatter Plots:

```
c= df['columna'].map(diccionario)
```

```
diccionario = {"valor1": "color1", "valor1": "color1"}
```

Lista de colores

Paletas Seaborn:

Accent', 'Accent_r', 'Blues', 'Blues_r', 'BrBG', 'BrBG_r', 'BuGn', 'BuGn_r', 'BuPu', 'BuPu_r', 'CMRmap', 'CMRmap_r', 'Dark2', 'Dark2_r', 'GnBu', 'GnBu_r', 'Greens', 'Greens_r', 'Greys', 'Greys_r', 'OrRd', 'OrRd_r', 'Oranges', 'Oranges_r', 'PRGn', 'PRGn_r', 'Paired', 'Paired_r', 'Pastell1', 'Pastell1_r', 'Pastel2', 'Pastel2_r', 'PiYG', 'PiYG_r', 'PuBu', 'PuBuGn', 'PuBuGn_r', 'PuBu_r', 'PuOr', 'PuOr_r', 'PuRd', 'PuRd_r', 'Purples', 'Purples_r', 'RdBu', 'RdBu_r', 'RdGy', 'RdGy_r', 'RdPu', 'RdPu_r', 'RdYlBu', 'RdYlBu_r', 'RdYlGn', 'RdYlGn_r', 'Reds', 'Reds_r', 'Set1', 'Set1_r', 'Set2', 'Set2_r', 'Set3', 'Set3_r', 'Spectral', 'Spectral_r', 'Wistia', 'Wistia_r', 'YlGn', 'YlGnBu', 'YlGnBu_r', 'YlGn_r', 'YlOrBr', 'YlOrBr_r', 'YlOrRd', 'YlOrRd_r', 'afmhot', 'afmhot_r', 'autumn', 'autumn_r', 'binary', 'binary_r', 'bone', 'bone_r', 'brg', 'brg_r', 'bwr', 'bwr_r', 'cividis', 'cividis_r', 'cool', 'cool_r', 'coolwarm', 'coolwarm_r', 'copper', 'copper_r', 'crest', 'crest_r', 'cubehelix', 'cubehelix_r', 'flag', 'flag_r', 'flare', 'flare_r', 'gist_earth', 'gist_earth_r', 'gist_gray', 'gist_gray_r', 'gist_heat', 'gist_heat_r', 'gist_ncar', 'gist_ncar_r', 'gist_rainbow', 'gist_rainbow_r', 'gist_stern', 'gist_stern_r', 'gist_yarg', 'gist_yarg_r', 'gnuplot', 'gnuplot2', 'gnuplot2_r', 'gnuplot_r', 'gray', 'gray_r', 'hot', 'hot_r', 'hsv', 'hsv_r', 'icefire', 'icefire_r', 'inferno', 'inferno_r', 'jet', 'jet_r', 'magma', 'magma_r', 'mako', 'mako_r', 'nipy_spectral', 'nipy_spectral_r', 'ocean', 'ocean_r', 'pink', 'pink_r', 'plasma', 'plasma_r', 'prism', 'prism_r', 'rainbow', 'rainbow_r', 'rocket', 'rocket_r', 'seismic', 'seismic_r', 'spring', 'spring_r', 'summer', 'summer_r', 'tab10', 'tab10_r', 'tab20', 'tab20_r', 'tab20b', 'tab20b_r', 'tab20c', 'tab20c_r', 'terrain', 'terrain_r', 'turbo', 'turbo_r', 'twilight', 'twilight_r', 'twilight_shifted', 'twilight_shifted_r', 'viridis', 'viridis_r', 'vlag', 'vlag_r', 'winter', 'winter_r'

```
palette='light:nombre_paleta'|'dark:nombre_paleta'
```

Marcadores

marker = 'tipo' establece el tipo de marcador; se usa con plt.scatter y plt.plot

| | |
|-------------------------|-------------------|
| "." Punto | "P" Más (relleno) |
| "," Pixel | "*" Estrella |
| "o" Circulo | "h" Hexágono 1 |
| "v" Triángulo abajo | "H" Hexágono 2 |
| "^" Triángulo arriba | "+" Más |
| "<" Triángulo izquierda | "x" x |
| ">" Triángulo derecha | "X" x (relleno) |
| "8" Octágono | "D" Diamante |
| "s" Cuadrado | "d" Diamante fino |
| "p" Pentágono | |

| NumPy (Numerical Python) |
|--|
| |
| Crear arrays |
| |
| Crear arrays de listas array = np.array(lista, dtype= tipo) crea un array unidimensional de una lista array = np.array([lista1, lista2]) crea un array bidimensional de dos listas array = np.array([listadelistas1, listadelistas2]) crea un array bidimensional de dos listas |
| |
| Crear otros tipos de arrays array = np.arange(valor_inicio, valor_final, saltos) crea un array usando el formato [start:stop:step] array = np.ones(z,y,x) crea un array de todo unos de la forma especificada array2 = np.ones_like(array1) crea un array de todo unos de la forma basada en otra array array = np.zeros(z,y,x) crea un array de todo zeros de la forma especificada array2 = np.zeros_like(array1) crea un array de todo zeros de la forma basada en otra array array = np.empty((z,y,x), tipo) crea un array vacio con datos por defecto tipo float array2 = np.empty_like(array1) crea un array vacia con la forma basada en otra array array = np.eye(z,y,x, k = n) crea un array con unos en diagonal empezando en la posicion k array = np.identity(x) crea una matriz de identidad con ceros en filas y unos en la diagonal, de forma cuadrada |

NumPy Random

np.random.seed(x) establece la semilla aleatoria del generador de números aleatorios, para que las funciones random que van después siempre cogerán los mismos valores "aleatorios"

Crear arrays con valores aleatorios

array = np.random.randint(inicio, final, forma_matriz)
crea un array de números aleatorios entre dos valores;
forma_matriz: (z,y,x)
z: número de arrays
y: número de filas
x: número de columnas
array = np.random.randint(inicio, final) devuelve un número aleatorio en el rango
array = np.random.rand(z,y,x) crea un array de floats aleatorias con la forma que le especificemos; por defecto genera números aleatorios entre 0-1
array = np.random.random_sample((z,y,x)) crea un array de floats aleatorias con la forma que le especificemos; por defecto genera números aleatorios entre 0-0.9999999...
array = np.random.z,y,x=None) devuelve un número aleatorio en 0 y 0.9999999999999...
np.round(np.random.rand(z,y,x), n) crear array con floats de n decimales
np.random.uniform(n,m, size = (z,y,x)) genera muestras aleatorias de una distribución uniforme en el intervalo entre n y m
np.random.binomial(n,m, size = (z,y,x)) genera muestras con una distribución binomial; n es el numero total de pruebas; m es la probabilidad de éxito
np.random.normal(loc = n, scale = m, size = (z,y,x)) genera números aleatorios de una distribución normal (curva de campana); loc es la media; scale es la desviación estándar
np.random.permutation(array) devuelve un array con los mismos valores mezclados aleatoriamente

Indices, Subsets, Metodos de Arrays

Indices de arrays

array[i] devuelve la indice i; las indices de los arrays unidimensionales funcionan igual que las listas
array[i, j] o **array[i][j]** devuelve el elemento de la columna j de la fila i
array[:,n] seleccionar todas las filas y las columnas hasta n-1
array[h, i, j] o **array[h][i][j]** devuelve el elemento de la columna j de la fila i del array h
array[h][i][j] = n cambiar el valor del elemento en esta posicion al valor n

Subsets

array > n devuelve la forma del array con True o False según si el elemento cumple con la condición o no
array[array > n] devuelve un subset: todos los valores que cumplen la condición en una lista dentro de un array
array[(array > n) & (array < m)] devuelve un subset: todos los valores que cumplen las condiciones en una lista dentro de un array; se puede usar | para "or"

Metodos de arrays

nuevo_array = array.copy() crea un a copia del array
np.transpose(array_bidimensional) cambia los filas del array a columnas y las columnas a filas
np.transpose(array_multidimensional) cambia el número de columnas al número de arrays y viceversa; el número de filas no cambia
np.transpose(array_multidimensional, (z,y,x)) hace la transposicion segun lo que especifecemos usando las posiciones de la tupla (0,1,2) de la forma original
array = np.arange(n).reshape((y,x)) crea un array usando reshape para definir la forma
array = np.reshape(array, (z,y,x)) crea un array con los valores de otro array usando reshape para definir la forma
array = np.swapaxes(array, posicion, posicion) intercambia dos ejes de una matriz usando las posiciones (z=0,y=1,x=2) de la forma original

Otras operaciones

np.sort(array) devuelve un array con los valores de cada fila ordenados en orden ascendente por defecto
np.sort(array, axis = 0) devuelve un array con los valores de cada columna ordenados en orden ascendente
np.sort(-array) devuelve un array con los valores de cada fila ordenados en orden descendente
np.round(array, decimals = x) devuelve un array con los valores del array redondeados a x decimales
np.round(array, decimals = x) devuelve un array con los valores del array redondeados a x decimales
np.where(array > x) devuelve los indices de los valores que cumplan la condición, por fila y columna

Operaciones con arrays

np.add(array1, array2) suma dos arrays
np.subtract(array1, array2) resta el array2 del array1
np.multiply(array1, array2) multiplica dos arrays
np.divide(array1, array2) divide el array1 por el array2
array + n, n * array, etc. - operadores algebraicos

Operaciones estadísticas y matemáticas

Operaciones estadísticas y matemáticas

El parametro axis en arrays bidimensionales:
axis = 0 columnas
axis = 1 filas
- si especificamos el axis, la operación devuelve el resultado por cada fila o columna.
Por ejemplo:
np.sum(array, axis = 0) devuelve un array con la suma de cada fila

El parametro axis en arrays multidimensionales:
axis = 0 dimensión
axis = 1 columnas
axis = 2 filas
- si especificamos el axis, la operación devuelve el resultado por cada dimensión, fila o columna.
Por ejemplo:
np.sum(array_3D, axis = 0) devuelve un array de una matriz con la suma de todas las matrices
np.sum(array_3D, axis = 1) devuelve un array donde las filas contienen las sumas de las columnas de cada matriz

Operaciones con parámetro del axis:
np.sum(array_3D) devuelve la suma de todos los elementos de los matrices
np.mean(array) devuelve la media de todo el array
np.std(array) devuelve la desviación estándar de todo
np.var(array) devuelve la varianza de valores de todo
np.min(array) devuelve el valor mínimo del array
np.max(array) devuelve el valor máximo del array
np.sum(array) devuelve la suma de los elementos del array
np.cumsum(array) devuelve un array con la suma acumulada de los elementos a lo largo del array
np.cumprod(array) devuelve un array con la multiplicación acumulada de los elementos a lo largo del array

Operaciones sin parámetro del axis:
np.sqrt(array) devuelve un array con la raíz cuadrada no negativa de cada elemento del array
np.exp(array) devuelve un array con el exponencial de cada elemento del array
np.mod(array1, array2) devuelve un array con el resto de la división entre dos arrays
np.mod(array1, n) devuelve un array con el resto de la división entre el array y el valor de n
np.cos(array) devuelve un array con el coseno de cada elemento del array
np.sin(array) devuelve un array con el seno de cada elemento del array
np.sin(array) devuelve un array con la tangente de cada elemento del array

Operaciones de comparación en arrays bidimensionales
np.any(array > n) devuelve True o False segun si cualquier valor del array cumpla con la condicion
np.any(array > n, axis = b) devuelve un array con True o False por cada columna o fila según si algún valor de la fila o columna cumpla con la condición
np.all(array > n) devuelve True o False segun si todos los valores del array cumpla con la condicion
np.all(array > n, axis = b) devuelve un array con True o False por cada columna o fila según si todos los valores de la fila o columna cumplan con la condición

Funciones de conjuntos

np.unique(array) devuelve un array con los valores únicos del array ordenados
np.unique(array, return_index=True) devuelve un array con los valores únicos del array ordenados y un array con la posición de la primera instancia de cada valor
np.unique(array, return_inverse=True) devuelve un array con los valores únicos del array ordenados y un array con las posiciones de cada elemento de cada valor
np.unique(array, return_counts=True) devuelve un array con los valores únicos del array ordenados y un array con el número de veces que aparece cada valor
np.unique(array, axis = b) devuelve un array con los valores únicos ordenados de las filas o columnas

Funciones para arrays unidimensionales

np.intersect1d(array1, array2) devuelve un array con los valores únicos de los elementos en común de dos arrays
np.intersect1d(array1, array2, return_indices=True) devuelve un array con los valores únicos de los elementos en común de dos arrays y arrays con los índices de cada valor, por array
np.union1d(array1, array2) devuelve un array ordenado con los elementos resultantes de unir dos arrays (valores únicos)
np.in1d(array1, array2) devuelve un array con True o False por cada elemento de array1 según si aparece el mismo valor en array2
np.setdiff1d(array1, array2) devuelve un array ordenado con los valores únicos que están en array1 pero no en array2
np.setxor1d(array1, array2) devuelve un array ordenado con los valores únicos que NO están en común de los dos arrays

Estadística

Medidas de dispersión

Desviación respecto a la media
la diferencia en valor absoluto entre cada valor de los datos y su media aritmética
diferencias = df['columna'] - df['columna'].mean()
desviación_media = np.abs(diferencias)

Varianza
medida de dispersión; la variabilidad respecto a la media
df['columna'].var()
Desviación estándar o desviación típica
la raíz cuadrada de la varianza; cuanto mayor sea, mayor será la dispersión o variabilidad en nuestros datos
df['columna'].std()

Robustez
- cuanto más cantidad de datos, más robustos
1/n donde n es el numero de registros
Coefficiente de variación
el cociente entre la desviación típica y la media; cuanto mayor sea, mayor será la dispersión en nuestros datos
df['columna'].std() / df['columna'].mean()

Percentiles
divide datos ordenados de menor a mayor en cien partes; muestra la proporción de datos por debajo de su valor
percentil_n = np.percentile(df['columna'], n) saca el valor en el percentil n

Rangos intercuartílicos
medida de dispersión: diferencia entre cuartiles 75 y 25
q3, q1 = np.percentile(df["columna"], [75, 25]) saca los tercer y primer cuartiles
rango_intercuartílico = q3 - q1

Estadística

Tablas de frecuencias

Frecuencias absolutas
el número de veces que se repite un número en un conjunto de datos
df = df.groupby('columna').count().reset_index()
Frecuencias relativas
las veces que se repite un número o categoría en un conjunto de datos respecto al total, en porcentajes
df_group_sin_str = df_group.drop('columna_str', axis=1)
frecuencia_relativa = df_group_sin_str / df.shape[0] * 100
columnas = df_group_sin_strings.columns
df_group[columnas] = frecuencia_relativa

Tablas de contingencia
tabla de frecuencias que cuenta todas las combinaciones posibles de cada pareja de valores de las columnas que estamos intentando comparar
df_crosstab = pd.crosstab(df['columna1'], df['columna2'], normalize = True, margins = True)
normalize muestra los valores en porcentajes (por uno)
margins muestra los totales y subtotales

Coefficiente de correlación de Pearson
- nos permite conocer la intensidad y dirección de la relación entre las dos variables
- coeficiente > 0: correlación positiva
- coeficiente < 0: correlación negativa
- coeficiente = 1 o -1: correlación total
- coeficiente = 0: no existe relación lineal
df['columna1'].corr(df['columna2']) calcula la correlacion entre dos variables
matriz_correlacion = df.corr() crea una matriz mostrando las correlaciones entre todos los variables
sns.heatmap(df.corr()[['column1', 'column2']], cmap = 'color_palette', annot = True, vmin = -1, vmax = 1)
crea una grafica heatmap de la matriz de correlaciones

Sesgos (skewness)
medida de la asimetría de la distribución de los valores de una variable alrededor de su valor medio
- valor de sesgo positivo: sesgado a la derecha
- valor de sesgo negativo: sesgado a la izquierda
- valor de sesgo igual a 0: valores simetricos
sns.displot(df['columna'], kde = True) crea un histograma que muestra la distribution de los valores
import scipy.stats import skew
skew(df['columna']) muestra el valor del sesgo de una variable

Intervalos de confianza
describe la variabilidad entre la medida obtenida en un estudio y la medida real de la población (el valor real)
import scipy.stats as st
st.t.interval(alpha = n, df = len(df['columna'])-1, loc = np.mean(df['columna']), scale = st.sem(df['columna']))
devuelve el rango de valores para lo cual hay un n% de probabilidad que un valor real cae en ese rango
alpha: porcentaje de confianza (p.ej. 90%, 95%, o 99%)
df: los datos
loc: la media
scale: la desviación estándar

| EDA Exploratory Data Analysis | ETL: Extract, Transform, Load | Machine Learning: Preparación | Tests estadísticos | Normalización |
|---|---|--|--|--|
| <div>Análisis exploratorio de datos</div> <div>El Análisis Exploratorio de Datos se refiere al proceso de realizar una serie de investigaciones inciales sobre los datos que tenemos para poder descubrir patrones, detectar anomalías, probar hipótesis y comprobar suposiciones con la ayuda de estadísticas y representaciones gráficas.</div> | <div>Extraccion<div>- obtener datos crudos y almacenarlos<ul style="list-style-type: none">- Tablas de bases de datos SQL o NoSQL- Ficheros de texto plano- Emails- Información de páginas web- Hojas de cálculo- Ficheros obtenidos de API's</div></div> <div>Transformación<div>- procesar los datos, unificarlos, limpiarlos, validarlos, filtrarlos, etc.<ul style="list-style-type: none">- Formetear fechas- Reordenar filas o columnas- Unir o separar datos- Combinar las fuentes de datos- Limpiar y estandarizar los datos- Verificar y validar los datos- Eliminar duplicados o datos erroneos- Filtrado, realización de calculos o agrupaciones</div></div> <div>Carga<div>- cargar los datos en su formato de destino, el tipo de lo cual dependerá de la naturaleza, el tamaño y la complejidad de los datos. Los sistemas más comunes suelen ser:<ul style="list-style-type: none">- Ficheros csv- Ficheros json- Bases de datos- Almacenes de datos (Data Warehouse)- Lagos de datos (Data Lakes)</div></div> | <div>Hipotesis Nula y Errores Tipo I y II</div> <div>Hipótesis nula (H0)<div>- en general es la afirmación contraria a la que queremos probar</div></div> <div>Hipótesis alternativa (H1)<div>- en general la afirmación que queremos comprobar</div></div> <div>p-valor<div>- medida de la probabilidad de que una hipótesis nula sea cierta</div><div>- valor entre 0 y 1</div><div>- si *p-valor* < 0.05 ❌ Rechazamos la hipótesis nula.</div><div>- si *p-valor* > 0.05 ✅ Aceptamos la hipótesis nula.</div></div> <div>Error Tipo I:<div>- rechazar la hipótesis nula cuando es verdadera</div></div> <div>Error Tipo II:<div>- aceptar la hipótesis nula cuando es falsa</div></div> | <div>Independencia entre variables predictoras</div> <div>- las variables predictoras tienen que ser independientes para poder crear un modelo de regresión lineal</div> <div>Variables numéricas: Correlaciones<div>- pairplot</div><div>sns.pairplot(df)</div><div>- covarianza</div><div>df_numéricas.cov()</div><div>- correlación de Pearson (relación lineal)</div><div>df_numéricas.corr()</div><div>- correlación de Spearman (relación no lineal)</div><div>df_numéricas.corr(method = 'spearman')</div><div>- correlación de Kendall (datos numéricos pero categóricos y ordinales)</div><div>df_numéricas.corr(method = 'kendall')</div></div> <div>Variables categóricas: Chi-cuadrado<div>- V-Cramer: varía entre 0 y 1<ul style="list-style-type: none">- más cerca a 1 más dependientes</div><div>- resultado < 0,7 para hacer ML ✅</div><div>import researchpy as rp</div><div>crosstab, test_results, expected = rp.crosstab(df["col1"], df["col2"], test= "chi-square", expected_freqs= True, prop= "cell")</div><div>test_results devuelve los resultados del test en un dataframe</div></div> <div>Homocedasticidad (homogeneidad de varianzas)<div>- las variables predictoras tienen que tener homogeneidad de varianzas en comparación con la variable respuesta</div></div> <div>Visualmente:<div>- violinplot</div><div>- boxplot</div><div>- regplot (columnas numéricas vs variable respuesta)</div></div> <div>Metodos analiticos:<div>- test de Levene (más robusto ante falta de normalidad) o Bartlett</div><div>from scipy import stats</div><div>from scipy.stats import levene</div><div>Variables categóricas:<div>- hay que crear un dataframe para cada valor único de las columnas categóricas</div><div>df_valor1 = df[df['col1'] == 'valor1']['col_VR']</div><div>df_valor2 = df[df['col1'] == 'valor2']['col_VR']</div><div>levene_test = stats.levene(df_valor1, df_valor2, center='median')</div><div>bartlett_test = stats.bartlett(df_valor1, df_valor2, center='median')</div></div><div>Variables numéricas:<div>- hay que crear un dataframe de las columnas numéricas sin la variable respuesta</div><div>for col in df_numericas.columns:<div>statistic, p_val = levene(df[col], df['col_VR'], center='median')</div><div>resultados[col] = p_val</div></div></div><div>devuelve los p-valores en un diccionario</div><div>- p-valor del test > 0.05: varianzas iguales, homocedasticidad ✅</div><div>- p-valor del test < 0.05: varianzas diferentes, heterocedasticidad</div></div> | <div>Método manual</div> <div>df["col_norm"] = (df["col_VR"] - df["col_VR"].media()) / (df["col_VR"].max() - df["col_VR"].min())</div> <div>Método logarítmica</div> <div>*no se puede hacer si algún valor sea 0*</div> <div>df["col_norm"] = df["col_VR"].apply(lambda x: np.log(x) if x > 0 else 0)</div> <div>Método raiz cuadrada</div> <div>import math</div> <div>df["col_norm"] = df["col_VR"].apply(lambda x: math.sqrt(x))</div> <div>Método stats.boxcox()</div> <div>from scipy import stats</div> <div>df["col_norm"], lambda ajustada = stats.boxcox(df["col_VR"])</div> <div>Método MinMaxScaler</div> <div>from sklearn.preprocessing import MinMaxScaler</div> <div>modelo = MinMaxScaler()</div> <div>modelo.fit(df["col_VR"])</div> <div>datos_normalizados = modelo.transform(df["col_VR"])</div> <div>df_datos_norm = pd.DataFrame(datos_normalizados, columns = ['col_norm'])</div> <div>df['col_norm'] = df_datos_norm</div> |
| <div>1. Entender las variables</div> <div>- que variables temenos</div> <div>.head(), .tail(), .describe(), .info(), .shape</div> <div>- que tipos de datos</div> <div>.dtypes(), .info()</div> <div>- si temenos nulos o duplicados</div> <div>.isnull().sum()</div> <div>.duplicated().sum()</div> <div>- que valores unicos temenos</div> <div>.unique(), .value_counts()</div> | | | | |
| <div>2. Limpiar el dataset</div> <div>- quitar duplicados (filas o columnas)</div> <div>- cambiar nombres de columnas</div> <div>- cambiar tipo de datos de columnas</div> <div>- ordenar columnas</div> <div>- separar columna en dos con str.split()</div> <div>- crear intervalos con pd.cut()</div> <div>- crear porcentajes o ratios</div> <div>- decidir como tratar outliers: mantenerlos, eliminarlos, o reemplazarlos con la media, mediana o moda; o aplicar una imputacion</div> <div>- decidir como tratar nulos:<ul style="list-style-type: none">- eliminar filas o columnas con nulos drop.na()- imputar valores perdidos:<ul style="list-style-type: none">- reemplazarlos con la media, mediana o moda usando .fillna() o .replace()- imputer con metodos de machine learning usando la libreria sklearn: Simple-Imputer, Iterative-Imputer, o KNN Imputer</div> | | | | |
| <div>3. Analizar relaciones entre variables</div> <div>Analizar relaciones entre las variables</div> <div>- para encontrar patrones, relaciones o anomalias</div> <div>Relaciones entre dos variables numéricas:<div>- scatterplot</div><div>- regplot - scatterplot con línea de regresion</div><div>- matriz de correlación y heatmap</div><div>- joinplot - permite emparejar dos gráficas - una histograma con scatter o reg plot por ejemplo</div></div> <div>Relaciones entre dos variables categóricas:<div>- countplot</div></div> <div>Relaciones entre variables numéricas y categóricas:<div>- swarmplot</div><div>- violinplot</div><div>- pointplot</div><div>- boxplot</div></div> | <div>APIs</div> <div>import requests</div> <div>libreria para realizar petitions HTTP a una URL, para hacer web scraping</div> <div>url = 'enlace'</div> <div>el enlace de la que queremos extraer datos</div> <div>header = {}</div> <div>opcional; contiene informacion sobre las peticiones realizadas (tipo de ficheros, credenciales)</div> <div>response = requests.get(url=url, header = header)</div> <div>pedimos a la API que nos de los datos</div> <div>variables = {'parametro1':'valor1', 'parametro2':'valor2'}</div> <div>response = request.get(url=url, params=variables)</div> <div>pedimos a la API que nos de los datos con los parametros segun el diccionario de parametros que le pasamos</div> <div>response.status_code</div> <div>devuelve el status de la peticion</div> <div>response.reason</div> <div>devuelve el motive de codigo de estado</div> <div>response.text</div> <div>devuelve los datos en formato string</div> <div>response.json()</div> <div>devuelve los datos en formato json</div> <div>df = pd.json_normalize(response.json)</div> <div>devuelve los datos en un dataframe</div> <div>Codigos de respuesta de HTTP</div> <div>1XX informa de una respuesta correcta</div> <div>2XX codigo de exito</div> <div>200 OK</div> <div>201 creado</div> <div>202 aceptado</div> <div>204 sin contenido</div> <div>3XX redireccion</div> <div>4XX error durante peticion</div> <div>401 peticion incorrecta</div> <div>402 sin autorizacion</div> <div>403 prohibido</div> <div>404 no encontrado</div> <div>5XX error del servidor</div> <div>501 error interno del servidor</div> <div>503 servicio no disponible</div> | | | |

| Machine Learning: Preparación | Regresión Lineal | Regresión Lineal: Decision Tree | Imbalanced Data |
|--|--|---|--|
| <div>ANOVA</div> <div><pre>import statsmodels.api as sm from statsmodels.formula.api import ols lm = ols('col_VR ~ col_VP1 + col_VP2 + col_VP3', data=df).fit()</pre>devuelve un dataframe de los resultados: df (degrees of freedom): número de observaciones en los datos que pueden variar libremente al estimar los parámetros estadísticos; para variables categóricas será el número de valores únicos menos 1; para variables numéricas será siempre 1 sum_sq: una medida de variación o desviación de la media; suma de los cuadrados de las diferencias con respecto a la media mean_sq: es el resultado de dividir la suma de cuadrados entre el número de grados de libertad. F: un test que se utiliza para evaluar la capacidad explicativa que tiene la variable predictora sobre la variación de la variable respuestae - PR(>F): si el p-valor < 0.05 es una variable significativa; que puede afectar a la VR lm.summary() devuelve una resumen de los resultados: coef: el coeficiente que representa los cambios medios en la variable respuesta para una unidad de cambio en la variable predictora mientras se mantienen constantes el resto de las VP; los signos nos indican si esta relación es positiva o negativa std err: el error estándar del coeficiente que se usa para medir la precisión de la estimación del coeficiente; cuanto menor sea el error estándar, más precisa será la estimación t: es el resultado de dividir el coeficiente entre su error estándar</div> | <div>1. separar los datos de las variables predictoras (x) de la variable respuesta (y) x = df.drop('col_VR', axis=1) y = df['col_VR'] 2. dividimos los datos en datos de entrenamiento y datos de test con train_test_split() x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2, random_state = 42) 3. Ajustamos el modelo lr = LinearRegression(n_jobs=-1) lr.fit(x_train, y_train) 4. Hacemos las predicciones y_predict_train = lr.predict(x_train) y_predict_test = lr.predict(x_test) 5. Guardamos los resultados en dataframes y los concatenamos train_df = pd.DataFrame({'Real': y_train, 'Predicted': y_predict_train, 'Set': ['Train']*len(y_train)}) test_df = pd.DataFrame({'Real': y_test, 'Predicted': y_predict_test, 'Set': ['Test']*len(y_test)}) resultados = pd.concat([train_df,test_df], axis = 0) 6. creamos una columna de los residuos: la diferencia entre los valores observados y los de la predicción resultados['residuos'] = resultados['Real'] - resultados['Predicted'] Cross-validation cv_scores = cross_val_score(estimator = LinearRegression(), X = X, y = y, scoring = 'neg_root_mean_squared_error', cv = 10) cv_scores.mean() calcula la media de los resultados de validación de una métrica cv_scores = cross_validate(estimator = LinearRegression(), X = X, y = y, scoring = 'r2', 'neg_root_mean_squared_error', cv = 10) cv_scores["test_r2"].mean() cv_scores["test_neg_root_mean_squared_error"].mean() calcula las medias de los resultados de validación de múltiples métricas Métricas R2: una medida estadística que representa la proporción de la varianza que puede ser explicada por las variables independientes (o predictoras) del modelo de regresión r2_score(y_train,y_predict_train) r2_score(y_test,y_predict_test) MAE (Mean absolute error): una medida de la diferencia entre los valores predichos frente a los reales. A menor MAE, mejor es capaz de ajustar los datos del modelo que hemos creado. mean_absolute_error(y_train,y_predict_train) mean_absolute_error(y_test,y_predict_test) MSE (Mean Squared Error): mide el promedio(media) de los errores al cuadrado. A menor MSE, mejor es capaz de ajustar los datos del modelo que hemos creado. mean_squared_error(y_train,y_predict_train) mean_squared_error(y_test,y_predict_test) RMSE (Root Mean Squared Error): nos muestra la distancia promedio entre los valores predichos y los valores reales del dataset. A menor RMSE, mejor es capaz de ajustarse el modelo obtenido. np.sqrt(mean_squared_error(y_train,y_predict_train)) np.sqrt(mean_squared_error(y_test,y_predict_test))</div> | <div><pre>from sklearn.model_selection import train_test_split from sklearn.ensemble import DecisionTreeRegressor from sklearn import tree from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error from sklearn.model_selection import GridSearchCV</pre>1. separar los datos de las variables predictoras (x) de la variable respuesta (y) x = df.drop('col_VR', axis=1) y = df['col_VR'] 2. dividimos los datos en datos de entrenamiento y datos de test con train_test_split() x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2, random_state = 42) 3. Ajustamos el modelo arbol = DecisionTreeRegressor(random_state=0) arbol.fit(x_train, y_train) Para dibujar el árbol: fig = plt.figure(figsize = (10,6)) tree.plot_tree(arbol, feature_names = x_train.columns, filled = True) plt.show() 4. Hacemos las predicciones y_predict_train = arbol.predict(x_train) y_predict_test = arbol.predict(x_test) 5. Examinamos lás métricas de los resultados - si temenos overfitting hay que reducir la profundidad del modelo - si temenos underfitting hay que aumentar la profundidad del modelo max_features = np.sqrt(len(x_train.columns)) podemos calcular el valor de max_features siendo la raíz cuadrada del número de variables predictoras arbol.tree_.max_depth nos muestra el max depth usado por defecto, para poder ajustarlo param = {"max_depth": [n,m,l], "max_features": [a,b,c,d], "min_samples_split": [10, 50, 100], "min_samples_leaf": [10, 50, 100]} definimos un diccionario de los hiperparámetros 6. Iniciamos el modelo con GridSearch gs = GridSearchCV(estimator = DecisionTreeRegressor(), param_grid = param, cv=10, verbose=-1, return_train_score = True, scoring = "neg_mean_squared_error") - GridSearch ejecuta todas las posibles combinaciones de hiperparámetros 7. Ajustamos el modelo en el GridSearch gs.fit(x_train, y_train) 8. Aplicamos el método de best_estimator_ mejor_modelo = gs.best_estimator_ devuelve la mejor combinación de hiperparámetros 9. Volvemos a sacar las predicciones y_pred_test_dt2 = mejor_modelo.predict(x_test) y_pred_train_dt2 = mejor_modelo.predict(x_train)</div> <div>Regresión Lineal: Random Forest seguir los mismos pasos como para el Decision Tree pero con RandomForestRegressor() from sklearn.ensemble import RandomForestRegressor</div> | <div>Downsampling ajustar la cantidad de datos de la categoría mayoritaria a la minoritaria Método manual df_minoritaria = df[df['col'] == valor_min] df_muestra = df[df['col'] == valor_max].sample(num_minoritarios, random_state = 42) df_balanceado = pd.concat([df_minoritaria, df_muestra],axis = 0) Método RandomUnderSample import imblearn X = df.drop('col_VR', axis=1) y = df['col_VR'] down_sampler = RandomUnderSampler() X_down, y_down = down_sampler.fit_resample(X,y) df_balanceado = pd.concat([X_down, y_down], axis = 1) Método Tomek x = df.drop('col_VR', axis=1) y = df['col_VR'] x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2, random_state = 42) tomek_sampler = SMOTETomek() X_train_res, y_train_res = tomek_sampler.fit_resample(X_train, y_train) Upsampling ajustar la cantidad de datos de la categoría minoritaria a la mayoritaria Método manual df_mayoritaria = df[df['col'] == valor_may] df_muestra = df[df['col'] == valor_min].sample(num_mayoritarias, random_state = 42) df_balanceado = pd.concat([df_mayoritaria, df_muestra],axis = 0) Método RandomOverSample import imblearn X = df.drop('col_VR', axis=1) y = df['col_VR'] down_sampler = RandomUnderSampler() X_down, y_down = down_sampler.fit_resample(X,y) df_balanceado = pd.concat([X_down, y_down], axis = 1)</div> |