5.4 Resumen

Resumen de lo aprendido en el `groupby`

Método `groupby`	Descripción	Ejemplo en Python
`.groupby()`	Agrupa un DataFrame por columnas específicas.	`grupos = df.groupby('columna')`
`.agg()`	Realiza agregaciones en grupos.	`resultados = grupos['columna_agregada'].agg(['sum', 'mean'])`
`.count()`	Cuenta los elementos en cada grupo.	`conteo = grupos['columna'].count()`
`.sum()`	Calcula la suma de valores en grupos.	`suma = grupos['columna'].sum()`
`.mean()`	Calcula el promedio en grupos.	`promedio = grupos['columna'].mean()`
`.max()`	Encuentra el valor máximo en grupos.	`maximo = grupos['columna'].max()`
`.min()`	Encuentra el valor mínimo en grupos.	`minimo = grupos['columna'].min()`
`.std()`	Calcula la desviación estándar en grupos.	`desviacion = grupos['columna'].std()`
`.median()`	Calcula la mediana en grupos.	`mediana = grupos['columna'].median()`
`.ngroups`	Devuelve el número de grupos creados.	`num_grupos = grupos.ngroups`

- Usos:
  - Analiza datos de algunas categorías
  - Divide, aplica (un estadístico) y agrupa (en función de la columna o columnas que especifiquemos)
  - Si no le pasamos un estadístico nos devuelve un objeto de tipo groupby
  - Filtra los datos
- Para convertir el resultado del groupby a un DataFrame:
  - EL_RESULTADO_DEL_GROUPBY.reset_index()
  - pd.DataFrame(EL_RESULTADO_DEL_GROUPBY)
- Estadísticos:
  - Se puede sacar un estadístico para todas las columnas o para una sola columna:

```
# PARA TODAS LAS COLUMNAS
df.groupby("gender").count()
```

```
# PARA UNA COLUMNA
df.groupby("gender")["num_polices"].count()
```

  - También podemos incluir múltiples condiciones

```
# ESTAMOS HACIENDO UNA AGRUPACIÓN POR GÉNERO Y TAMAÑO DE VEHÍCULO
df.groupby(["gender", "vehicule_size"])["num_polices"].count()
```

- Si quiero calcular más de un estadístico == `.agg(TIENE QUE IR EN FORMATO LISTA)`

```
df.groupby("gender")["col1"].agg(["mean", "std"])
```

- Nulos:
  - Tenemos el parámetro `dropna`.
  - El groupby por defecto **LOS IGNORA**, es decir, `dropna = True`.
  - En caso de que queramos que **nos incluya** los nulos tendremos que establecer `dropna = False`.

Apply

- Utilizamos el metodo `.apply()` para aplicar una misma función a las filas o columnas de un objeto DataFrame de Pandas.
- Sintaxis general: `df['nombre_columna'].apply(funcion_a_aplicar)`
- Utilizamos el metodo apply con lambdas, cuando queremos aplicar la misma funcion a varias columnas o utilizar funciones con más de un argumento.
  - Sintaxis general: `df.apply(lambda x:funcion_a_aplicar(argumento1,argumento2))`
    - Si no tenemos una funcion definida previamente podemos aplicarlo directamente. Eg: `df.apply(lambda x:df[columna_1]/df[columna_2])`
- Funcion que recibe un parametro: SOLO APPLY

```
df[col].apply(mifuncion)
```

  - OJO!!! QUE SI MI FUNCIÓN RECIBE UN PARAMETRO PERO EN EL RETURN TENEMOS MAS DE UN VALOR ENTONCES APPLY + LAMBDA

```
df.apply(lambda lola: mifuncion(lola[col1]), axis = 1,
result_type = "expand")
```

- Funcion que recibe mas de un parametro: APPLY + LAMBDA

```
df.apply(lambda lolo: mifuncion(lolo[col1], lolo[col2]), axis = 1)
```

Repaso de lo aprendido en el apply y métodos de limpieza

Método	Descripción	Ejemplo en Python
`max`	Encuentra el valor máximo en una serie o DataFrame.	`df['columna'].max()`
`min`	Encuentra el valor mínimo en una serie o DataFrame.	`df['columna'].min()`
`dtypes`	Devuelve los tipos de datos de las columnas.	`df.dtypes`
`apply`	Aplica una función a lo largo de una serie o DataFrame.	`df['columna'].apply(funcion)`
`map`	Reemplaza los valores en una serie utilizando un mapeo.	`df['columna'].map(mapeo)`
`replace`	Reemplaza valores específicos en una serie o DataFrame.	`df['columna'].replace({valor_original: valor_nuevo})`