Introduccion a la visualizacion con Matplotlib Seaborn y Folium

July 17, 2024

1 Introducción a la visualización de datos con Matplotlib, Seaborn y Folium

Elaboración: Gabriel Armando Landín Alvarado

1.1 Contenido

- ¿Qué es la visualización?
- ¿Qué es Matplotlib?
- ¿Qué es Seaborn?
- ¿Qué es Folium?
- Preparación de los datos
- Primeros pasos con Matpltlib
- Personalización del gráfico
- Primeros pasos con seaborn
- Primeros pasos con Folium

1.2 ¿Qué es la visualización?

La visualización es una de las tareas más importantes en el análisis de datos, es parte del proceso de exploración, siendo de gran utilidad para identificar outliers o datos atípicos, asimismo, es de utilidad para la transformación de los datos, o bien, en la generación de ideas para formular modelos, lo anterior, se puede conseguir a través de la representación de gráficos de línea, barras, puntos, histogramas, mapas, etc. La visualización también tiene como objetivo representar a los datos mediante imágenes para comprender la historia detrás de éstos.

1.3 ¿Qué es Matplotlib?

Matplotlib es una libreria potente de Python, muy empleada en la elaboración y publicación de gráficos, principalmente de 2 dimensiones. La representación de los datos con Matplotlib se da a través de tablas y gráficos, estos últimos, son diversos para cumplir con las necesidades y especificaciones de los usuarios, pues poseen un alto grado de personalización.

1.4 ¿Qué es Seaborn?

Seaborn es una libreria de visualización de datos de Python basada en matplotlib. Proporciona una interfaz de alto nivel para dibujar gráficos principalmente enfocados al análisis estadístico, los cuales son muy atractivos e informativos.

1.5 ¿Qué es Folium?

Folium es una libreria de visualización de datos en Python, la cual se creó principalmente para visualizar datos geoespaciales. Con Folium se pueden crear mapas de cualquier lugar del mundo gracias a los diversos mosaicos o Tiles que se proporcionan como plantillas URL. Aunque Folium, cabe aclarar que es en realidad un contenedor de Python para leaflet.js, pues es una libreria de JavaScript para trazar mapas interactivos.

1.6 Preparación de los datos

Para este ejercicio se hace uso de los datos abiertos de comercio exterior que proporciona la **Agencia Nacional de Aduanas de México (ANAM)**, los datos a visualizar son los correspondientes al número de operaciones y el total de recaudación (MXN) mensual, estos desagregados por aduana, los cuales incluyen entre sus características, nombre de aduana, la clave, la entidad federartiva y el tipo de aduana, se observa que en algunos casos caos casos se tienen datos desde 2006 y en otros desde 2018. Asimsimo, para la visualización geográfica o geoespacial se obtuvieron las cordenadas de longitud (x) y latitud (y) a partir de google maps, lo que implica una posible inexactitud.

Importar las librerias necesarias para la lectura de los datos

[1]:
```
import pandas as pd
import numpy as np
```

Carga de los datos de operaciones en un dataframe a partir de un archivo excel(.xlsx) con el uso de la función read_excel de pandas, asignando el nombre de la hoja (operacion) con los datos mediante el parámetro **sheet_name='operacion'**.

[2]:
```
df_ope = pd.read_excel('../Data/operaciones_comercio_exterior_ene-2012_oct_2023.
↪xlsx', sheet_name='operacion')
df_ope
```

[2]:

	fecha	cve_aduana	ope_imp	ope_exp	tot_ope	val_ope_imp	\
0	2012-01-31	1	0	16	16	4.446800e+04	
1	2012-01-31	2	2474	2012	4486	1.092726e+09	
2	2012-01-31	5	53	590	643	7.547948e+06	
3	2012-01-31	6	14	118	132	2.422538e+06	
4	2012-01-31	7	67330	53587	120917	4.741879e+10	
…	…	…	…	…	…	…	
7083	2023-10-31	81	18903	11139	30042	4.696339e+10	
7084	2023-10-31	82	911	5429	6340	2.456200e+09	
7085	2023-10-31	83	8	49	57	6.580586e+08	
7086	2023-10-31	84	1039	1295	2334	6.127381e+08	
7087	2023-10-31	85	40769	8906	49675	3.402402e+10	

	val_ope_exp	val_tot_ope
0	7.120369e+08	7.120813e+08
1	1.147747e+09	2.240473e+09
2	1.051023e+08	1.126503e+08
3	4.043483e+10	4.043725e+10

```
4     4.900728e+10  9.642608e+10
…              …           …
7083  3.701648e+10  8.397987e+10
7084  6.526085e+08  3.108809e+09
7085  2.172299e+10  2.238105e+10
7086  1.259251e+09  1.871990e+09
7087  1.147614e+10  4.550016e+10

[7088 rows x 8 columns]
```

Como se observa se tienen ocho columnas, correspondientes a la fecha, clave de aduana, operaciones de importación, operaciones de exportación, total de operaciones, valor de las operaciones de importación, valor de las operaciones de exportación y valor total de las operaciones.

A continuación, se muestra la información general del dataframe, total de registros, total de columnas, nombre de las columnas , la existencia o no de valores nulos, así como el tipo de dato.

[3]:
```
df_ope.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7088 entries, 0 to 7087
Data columns (total 8 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   fecha        7088 non-null   datetime64[ns]
 1   cve_aduana   7088 non-null   int64
 2   ope_imp      7088 non-null   int64
 3   ope_exp      7088 non-null   int64
 4   tot_ope      7088 non-null   int64
 5   val_ope_imp  7088 non-null   float64
 6   val_ope_exp  7088 non-null   float64
 7   val_tot_ope  7088 non-null   float64
dtypes: datetime64[ns](1), float64(3), int64(4)
memory usage: 443.1 KB
```

A fin de visualizar los datos para el total de aduanas (50) a la fecha de elaboración de este ejercicio, se filtran los datos a partir del campo fecha, para este caso los datos de enero a octubre de 2023, pues es la última actualización que se proporciona en la página de la ANAM en la sección Estadísticas hasta fecha de elaboración de este documento.

[4]:
```
df_ope_2023 = df_ope[ (df_ope['fecha'] >= '2023-01-01') ]
df_ope_2023
```

[4]:
```
          fecha  cve_aduana  ope_imp  ope_exp  tot_ope   val_ope_imp  \
6588 2023-01-31           1        3        0        3  1.672294e+08
6589 2023-01-31           2     1360     1849     3209  1.467799e+09
6590 2023-01-31           5       50      970     1020  1.966268e+07
6591 2023-01-31           6       20       17       37  8.667868e+06
6592 2023-01-31           7    74100    67531   141631  9.911240e+10
```

```
…          …          …       …       …       …           …
7083 2023-10-31          81    18903    11139    30042  4.696339e+10
7084 2023-10-31          82      911     5429     6340  2.456200e+09
7085 2023-10-31          83        8       49       57  6.580586e+08
7086 2023-10-31          84     1039     1295     2334  6.127381e+08
7087 2023-10-31          85    40769     8906    49675  3.402402e+10

       val_ope_exp   val_tot_ope
6588  0.000000e+00  1.672294e+08
6589  2.021909e+09  3.489708e+09
6590  2.638391e+08  2.835018e+08
6591  1.135490e+10  1.136357e+10
6592  1.067136e+11  2.058260e+11
…              …             …
7083  3.701648e+10  8.397987e+10
7084  6.526085e+08  3.108809e+09
7085  2.172299e+10  2.238105e+10
7086  1.259251e+09  1.871990e+09
7087  1.147614e+10  4.550016e+10

[500 rows x 8 columns]
```

Se tienen un total de 500 filas y 8 columnas en el dataframe filtrado.

Se asigna un nuevo índice (index) a partir de un rango con la ayuda de la función arange de numpy, el rango va desde 0 hasta 500, en pasos de 1.

[5]:
```
df_ope_2023.set_index(np.arange(0, 500, 1), inplace=True)
df_ope_2023
```

[5]:
```
          fecha  cve_aduana  ope_imp  ope_exp  tot_ope   val_ope_imp  \
0    2023-01-31           1        3        0        3  1.672294e+08
1    2023-01-31           2     1360     1849     3209  1.467799e+09
2    2023-01-31           5       50      970     1020  1.966268e+07
3    2023-01-31           6       20       17       37  8.667868e+06
4    2023-01-31           7    74100    67531   141631  9.911240e+10
..          …          …       …        …       …           …
495 2023-10-31          81    18903    11139    30042  4.696339e+10
496 2023-10-31          82      911     5429     6340  2.456200e+09
497 2023-10-31          83        8       49       57  6.580586e+08
498 2023-10-31          84     1039     1295     2334  6.127381e+08
499 2023-10-31          85    40769     8906    49675  3.402402e+10

      val_ope_exp   val_tot_ope
0    0.000000e+00  1.672294e+08
1    2.021909e+09  3.489708e+09
2    2.638391e+08  2.835018e+08
3    1.135490e+10  1.136357e+10
```

```
4    1.067136e+11  2.058260e+11
..            …           …
495  3.701648e+10  8.397987e+10
496  6.526085e+08  3.108809e+09
497  2.172299e+10  2.238105e+10
498  1.259251e+09  1.871990e+09
499  1.147614e+10  4.550016e+10

[500 rows x 8 columns]
```

Ahora toca la carga de los datos de recaudación igualmente a partir de un archivo excel con el uso de la función read_excel de pandas asignando el nombre de la hoja con los datos mediante el parámetro **sheet_name='recaudacion'**.

[6]:
```
df_rec = pd.read_excel('../Data/recaudacion_comercio_exterior_ene-2018_oct_2023.
 ↪xlsx', sheet_name='recaudacion')
df_rec
```

[6]:
```
          fecha  cve_aduana         iva        igi        dta       ieps  \
0    2018-01-31           1     9745040          0     196413   11583569
1    2018-01-31           2   102488214    2440155    1001256      12917
2    2018-01-31           5      246343       4872     259674          0
3    2018-01-31           6      112630       5271      28099          0
4    2018-01-31           7  1752961157   73953470   57139357  655910671
…           …         …          …          …          …
3443 2023-10-31          81  4038678834  372181316   97399246   19740102
3444 2023-10-31          82   405053959          0      14791          0
3445 2023-10-31          83   109451668       8489      13710  161378418
3446 2023-10-31          84    42693789    2827459    1531536     167974
3447 2023-10-31          85  3284093139  329853387  151145471   17824912

       isan       otros     total_rec
0         0       960.0  2.152598e+07
1         0    513346.0  1.064559e+08
2         0    276034.0  7.869230e+05
3         0     28752.0  1.747520e+05
4         0  14846438.0  2.554811e+09
…     …          …           …
3443  22500  29465094.0  4.557487e+09
3444      0    541108.0  4.056099e+08
3445      0    218588.0  2.710709e+08
3446      0    925966.0  4.814672e+07
3447      0  18681519.0  3.801598e+09

[3448 rows x 9 columns]
```

Se filtran los datos a partir de una fecha, nuevamente solo los datos de 2023 para unir los dataframes.

[7]:
```
df_rec_2023 = df_rec[ (df_rec['fecha'] >= '2023-01-01') ]
df_rec_2023
```

[7]:

	fecha	cve_aduana	iva	igi	dta	ieps	isan	otros	total_rec
2948	2023-01-31	1	33304355	0	1337836	39584086	0	326222.0	7.455250e+07
2949	2023-01-31	2	153123551	821147	2206187	2355	2500	2569986.0	1.587257e+08
2950	2023-01-31	5	969148	294858	653314	0	0	287496.0	2.204816e+06
2951	2023-01-31	6	5622452	303870	286062	0	0	2941079.0	9.153463e+06
2952	2023-01-31	7	1991114014	79418473	53411452	480373370	50000	34776868.0	2.639144e+09
…	…	…	…	…	…	…	…	…	…
3443	2023-10-31	81	4038678834	372181316	97399246	19740102	22500	29465094.0	4.557487e+09
3444	2023-10-31	82	405053959	0	14791	0	0	541108.0	4.056099e+08
3445	2023-10-31	83	109451668	8489	13710	161378418	0	218588.0	2.710709e+08
3446	2023-10-31	84	42693789	2827459	1531536	167974	0	925966.0	4.814672e+07
3447	2023-10-31	85	3284093139	329853387	151145471	17824912	0	18681519.0	3.801598e+09

[500 rows x 9 columns]

Se asigna un nuevo índice a las 500 filas a partir de un rango con la ayuda de la función arange de numpy.

[8]:
```
df_rec_2023.set_index(np.arange(0, 500, 1), inplace=True)
df_rec_2023
```

[8]:

	fecha	cve_aduana	iva	igi	dta	ieps
0	2023-01-31	1	33304355	0	1337836	39584086
1	2023-01-31	2	153123551	821147	2206187	2355
2	2023-01-31	5	969148	294858	653314	0
3	2023-01-31	6	5622452	303870	286062	0
4	2023-01-31	7	1991114014	79418473	53411452	480373370
..	…	…	…	…	…	…
495	2023-10-31	81	4038678834	372181316	97399246	19740102
496	2023-10-31	82	405053959	0	14791	0
497	2023-10-31	83	109451668	8489	13710	161378418
498	2023-10-31	84	42693789	2827459	1531536	167974

```
499 2023-10-31          85  3284093139  329853387  151145471   17824912

      isan       otros     total_rec
0        0    326222.0  7.455250e+07
1     2500   2569986.0  1.587257e+08
2        0    287496.0  2.204816e+06
3        0   2941079.0  9.153463e+06
4    50000  34776868.0  2.639144e+09
..     …         …           …
495  22500  29465094.0  4.557487e+09
496      0    541108.0  4.056099e+08
497      0    218588.0  2.710709e+08
498      0    925966.0  4.814672e+07
499      0  18681519.0  3.801598e+09

[500 rows x 9 columns]
```

Se eliminan las columnas de clave de aduana o cve_aduana y fecha en el dataframe de operaciones para evitar duplicidad al momento de unir con el dataframe de recaudación, el parámetro axis=1 hace referencia a columnas.

[9]:
```
df_ope_2023 = df_ope_2023.drop(['cve_aduana', 'fecha'], axis=1)
df_ope_2023
```

[9]:
```
     ope_imp  ope_exp  tot_ope   val_ope_imp   val_ope_exp   val_tot_ope
0          3        0        3  1.672294e+08  0.000000e+00  1.672294e+08
1       1360     1849     3209  1.467799e+09  2.021909e+09  3.489708e+09
2         50      970     1020  1.966268e+07  2.638391e+08  2.835018e+08
3         20       17       37  8.667868e+06  1.135490e+10  1.136357e+10
4      74100    67531   141631  9.911240e+10  1.067136e+11  2.058260e+11
..       …        …       …           …             …             …
495    18903    11139    30042  4.696339e+10  3.701648e+10  8.397987e+10
496      911     5429     6340  2.456200e+09  6.526085e+08  3.108809e+09
497        8       49       57  6.580586e+08  2.172299e+10  2.238105e+10
498     1039     1295     2334  6.127381e+08  1.259251e+09  1.871990e+09
499    40769     8906    49675  3.402402e+10  1.147614e+10  4.550016e+10

[500 rows x 6 columns]
```

Lo siguiente es concatenar o unir los dataframes de recaudacion y operaciones con ayuda de la función **conact()** de pandas a partir de columnas (axis=1).

[10]:
```
df_com_ext = pd.concat([df_rec_2023, df_ope_2023], axis=1)
df_com_ext
```

[10]:
```
         fecha  cve_aduana         iva        igi        dta       ieps  \
0   2023-01-31           1    33304355          0    1337836   39584086
1   2023-01-31           2   153123551     821147    2206187       2355
```

```
2   2023-01-31           5      969148     294858     653314          0
3   2023-01-31           6     5622452     303870     286062          0
4   2023-01-31           7  1991114014   79418473   53411452  480373370
..         …         …         …         …         …         …
495 2023-10-31          81  4038678834  372181316   97399246   19740102
496 2023-10-31          82   405053959          0      14791          0
497 2023-10-31          83   109451668       8489      13710  161378418
498 2023-10-31          84    42693789    2827459    1531536     167974
499 2023-10-31          85  3284093139  329853387  151145471   17824912

      isan       otros     total_rec  ope_imp  ope_exp  tot_ope   val_ope_imp  \
0        0    326222.0  7.455250e+07        3        0        3  1.672294e+08
1     2500   2569986.0  1.587257e+08     1360     1849     3209  1.467799e+09
2        0    287496.0  2.204816e+06       50      970     1020  1.966268e+07
3        0   2941079.0  9.153463e+06       20       17       37  8.667868e+06
4    50000  34776868.0  2.639144e+09    74100    67531   141631  9.911240e+10
..     …         …           …      …      …        …          …
495  22500  29465094.0  4.557487e+09    18903    11139    30042  4.696339e+10
496      0    541108.0  4.056099e+08      911     5429     6340  2.456200e+09
497      0    218588.0  2.710709e+08        8       49       57  6.580586e+08
498      0    925966.0  4.814672e+07     1039     1295     2334  6.127381e+08
499      0  18681519.0  3.801598e+09    40769     8906    49675  3.402402e+10

      val_ope_exp   val_tot_ope
0    0.000000e+00  1.672294e+08
1    2.021909e+09  3.489708e+09
2    2.638391e+08  2.835018e+08
3    1.135490e+10  1.136357e+10
4    1.067136e+11  2.058260e+11
..            …           …
495  3.701648e+10  8.397987e+10
496  6.526085e+08  3.108809e+09
497  2.172299e+10  2.238105e+10
498  1.259251e+09  1.871990e+09
499  1.147614e+10  4.550016e+10

[500 rows x 15 columns]
```

Información general del dataframe unido.

[11]:
```
df_com_ext.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 500 entries, 0 to 499
Data columns (total 15 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   fecha        500 non-null    datetime64[ns]
```

```
 1   cve_aduana   500 non-null    int64
 2   iva          500 non-null    int64
 3   igi          500 non-null    int64
 4   dta          500 non-null    int64
 5   ieps         500 non-null    int64
 6   isan         500 non-null    int64
 7   otros        500 non-null    float64
 8   total_rec    500 non-null    float64
 9   ope_imp      500 non-null    int64
 10  ope_exp      500 non-null    int64
 11  tot_ope      500 non-null    int64
 12  val_ope_imp  500 non-null    float64
 13  val_ope_exp  500 non-null    float64
 14  val_tot_ope  500 non-null    float64
dtypes: datetime64[ns](1), float64(5), int64(9)
memory usage: 60.5 KB
```

Ahora toca cargar la información de las aduanas, la cual se localiza en un archivo csv.

[12]:
```
df_aduanas = pd.read_csv('../Data/aduanas.csv')
df_aduanas.head()
```

[12]:

	cve_aduana	tipo_aduana	entidad_federativa	nombre_aduana	latitud_y	longitud_x
0	1	Marítima	Guerrero	Acapulco	16.84864	-99.90517
1	2	Fronteriza	Sonora	Agua Prieta	31.33370	-109.56082
2	73	Interior	Aguascalientes	Aguascalientes	22.00755	-102.25071
3	47	Interior	Ciudad de México	AICM	19.44654	-99.07066
4	85	Interior	Estado de México	AIFA	19.75307	-99.00563

Se concatena el dataframe de aduanas con el de operaciones y recaudación a partir de un campo común: cve_aduana, uniendo los datos con el parámetro how='left'.

[13]:
```
df_com_ext_2023 = pd.merge(df_com_ext, df_aduanas, on='cve_aduana', how='left')
df_com_ext_2023.head()
```

[13]:

	fecha	cve_aduana	iva	igi	dta	ieps	isan
0	2023-01-31	1	33304355	0	1337836	39584086	0
1	2023-01-31	2	153123551	821147	2206187	2355	2500
2	2023-01-31	5	969148	294858	653314	0	0
3	2023-01-31	6	5622452	303870	286062	0	0
4	2023-01-31	7	1991114014	79418473	53411452	480373370	50000

```
        otros     total_rec  ope_imp  ope_exp  tot_ope   val_ope_imp  \
0    326222.0  7.455250e+07        3        0        3  1.672294e+08
1   2569986.0  1.587257e+08     1360     1849     3209  1.467799e+09
2    287496.0  2.204816e+06       50      970     1020  1.966268e+07
3   2941079.0  9.153463e+06       20       17       37  8.667868e+06
4  34776868.0  2.639144e+09    74100    67531   141631  9.911240e+10

    val_ope_exp   val_tot_ope tipo_aduana entidad_federativa   nombre_aduana  \
0  0.000000e+00  1.672294e+08    Marítima           Guerrero        Acapulco
1  2.021909e+09  3.489708e+09  Fronteriza             Sonora     Agua Prieta
2  2.638391e+08  2.835018e+08  Fronteriza       Quintana Roo    Subte. López
3  1.135490e+10  1.136357e+10    Marítima           Campeche  Cd. Del Carmen
4  1.067136e+11  2.058260e+11  Fronteriza          Chihuahua      Cd. Juárez

   latitud_y  longitud_x
0   16.84864   -99.90517
1   31.33370  -109.56082
2   18.49097   -88.38474
3   18.65070   -91.84000
4   31.76268  -106.45177
```

Se genera una nueva columna con el número del mes a partir de la columna fecha con el apoyo de la función **DatetimeIndex** de pandas.

[14]:
```
df_com_ext_2023['mes'] = pd.DatetimeIndex(df_com_ext_2023['fecha']).month
# se contabiliza el número de registros o filas para cada mes, en este caso␣
 ↪corresponden a las 50 aduanas
df_com_ext_2023['mes'].value_counts()
```

[14]:
```
mes
1     50
2     50
3     50
4     50
5     50
6     50
7     50
8     50
9     50
10    50
Name: count, dtype: int64
```

Se crea una nueva columna (nombre_mes) con el nombre del mes a partir de la columna mes, para este caso se hace uso de la función .map(), donde a partir de un diccionario se asignan los valores de acuerdo con la clave o key que corresponde al número de mes.

[15]:
```
meses = {1:'Enero', 2:'Febrero', 3:'Marzo', 4:'Abril', 5:'Mayo', 6:'Junio', 7:
 ↪'Julio', 8:'Agosto', 9:'Septiembre', 10:'Octubre'}
```

```
df_com_ext_2023['nombre_mes'] = df_com_ext_2023['mes'].map(meses)
df_com_ext_2023['nombre_mes'].value_counts()
```

[15]:
```
nombre_mes
Enero         50
Febrero       50
Marzo         50
Abril         50
Mayo          50
Junio         50
Julio         50
Agosto        50
Septiembre    50
Octubre       50
Name: count, dtype: int64
```

[16]:
```
df_com_ext_2023.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 500 entries, 0 to 499
Data columns (total 22 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   fecha               500 non-null    datetime64[ns]
 1   cve_aduana          500 non-null    int64
 2   iva                 500 non-null    int64
 3   igi                 500 non-null    int64
 4   dta                 500 non-null    int64
 5   ieps                500 non-null    int64
 6   isan                500 non-null    int64
 7   otros               500 non-null    float64
 8   total_rec           500 non-null    float64
 9   ope_imp             500 non-null    int64
 10  ope_exp             500 non-null    int64
 11  tot_ope             500 non-null    int64
 12  val_ope_imp         500 non-null    float64
 13  val_ope_exp         500 non-null    float64
 14  val_tot_ope         500 non-null    float64
 15  tipo_aduana         500 non-null    object
 16  entidad_federativa  500 non-null    object
 17  nombre_aduana       500 non-null    object
 18  latitud_y           500 non-null    float64
 19  longitud_x          500 non-null    float64
 20  mes                 500 non-null    int32
 21  nombre_mes          500 non-null    object
dtypes: datetime64[ns](1), float64(7), int32(1), int64(9), object(4)
memory usage: 84.1+ KB
```

1.7 Primeros pasos con Matpltlib

Importar las librerias necesarias y definir algunas configuraciones.

[17]:
```
import matplotlib.pyplot as plt
# método magico que indica que los gráficos generados con Matplotlib se␣
 ↪muestran directamente en el notebook,
# justo debajo de la celda de código que los produjo. Esto significa que no␣
 ↪necesitas llamar explícitamente plt.show() para visualizar las gráficas.
%matplotlib inline
plt.style.use('ggplot') # con esta linea de código se específica que se use el␣
 ↪estilo ggplot
```

Matplotlib se compone de las siguientes capas:

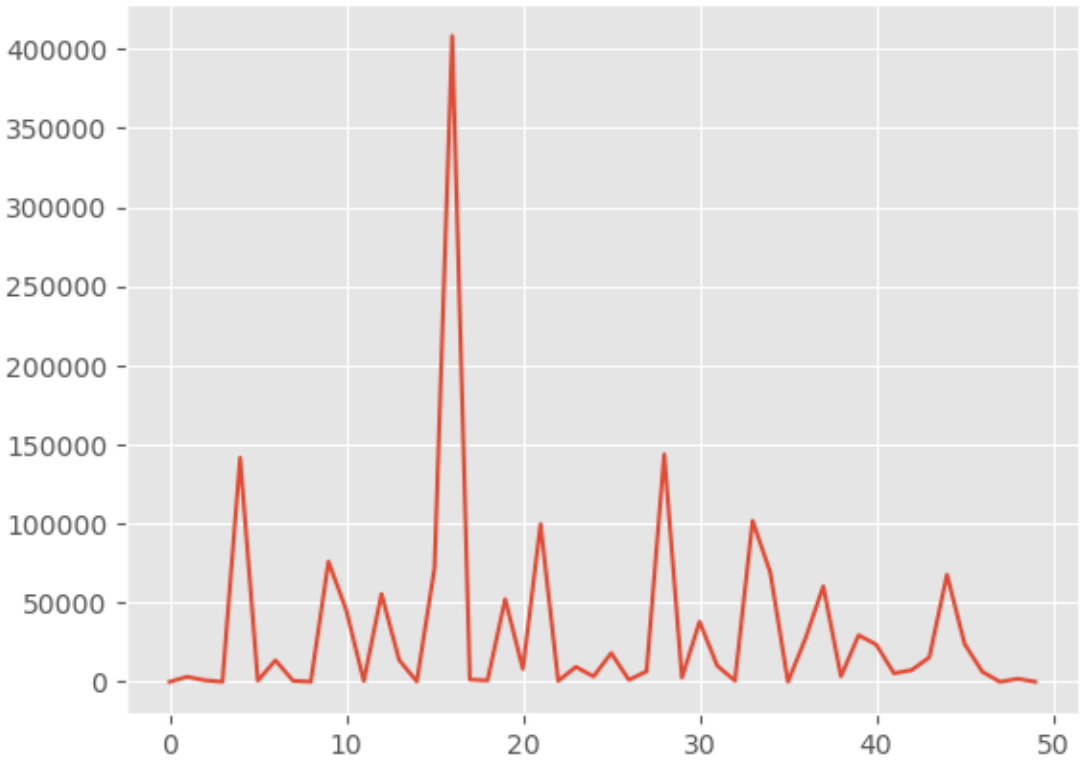
- **Capa de Scripting o Scripting Layer:** Esta es la capa más" ala"a y la qu see utilizs más comúnment, es decir, por quienes no poseen un conocimeinto amplio de programacióne En esta se , interactos con Matplotlib a través de funciones como plt.plot(), plt.xlabel(, plt.title(), et . Es ideal para tareas rápidas y sencillas de visualizaci
- **Capa de Artistas o Artist Layer:** En esta capa, todo lo visible en la figura esde un "artista" Los artistas incluyen objetos como líneas (Line2D), texto (Text), parches (Patch), et Se puede es personalizas directame a través de funciones como set_xlabel, set_title, etc.ntasimismoplse puede o, camrndo colores o estil
- **Capa de Ejes o Axes Layer:** Un "Axes" es un" artist"a adjunto a una figura que contiene una región(n filas, n columnas) para traza datos Cada "Axes" tiene dos (o tres en el caso de gráficos 3D) objetos "Axis" que proporcionan marcas y etiquetas para los eje . Aquí configuramos la mayoría de las partes del gráfico como agregar datos, controlar la escala y/o límites, así como agregar etiquetas. Para más información se puede consultar la documentación.

A partir de este momento ya podemos crear gráficos con la función plot() de matplotlib (plt).

En el siguiente ejemplo, se crea un gráfico de línea simple con los primeros 50 datos de la columna total de operaciones (tot_ope), podrá observar que encima del gráfico se proporciona información del objeto creado y el uso en la memoria, esto puede ocultarse empleando la función show().

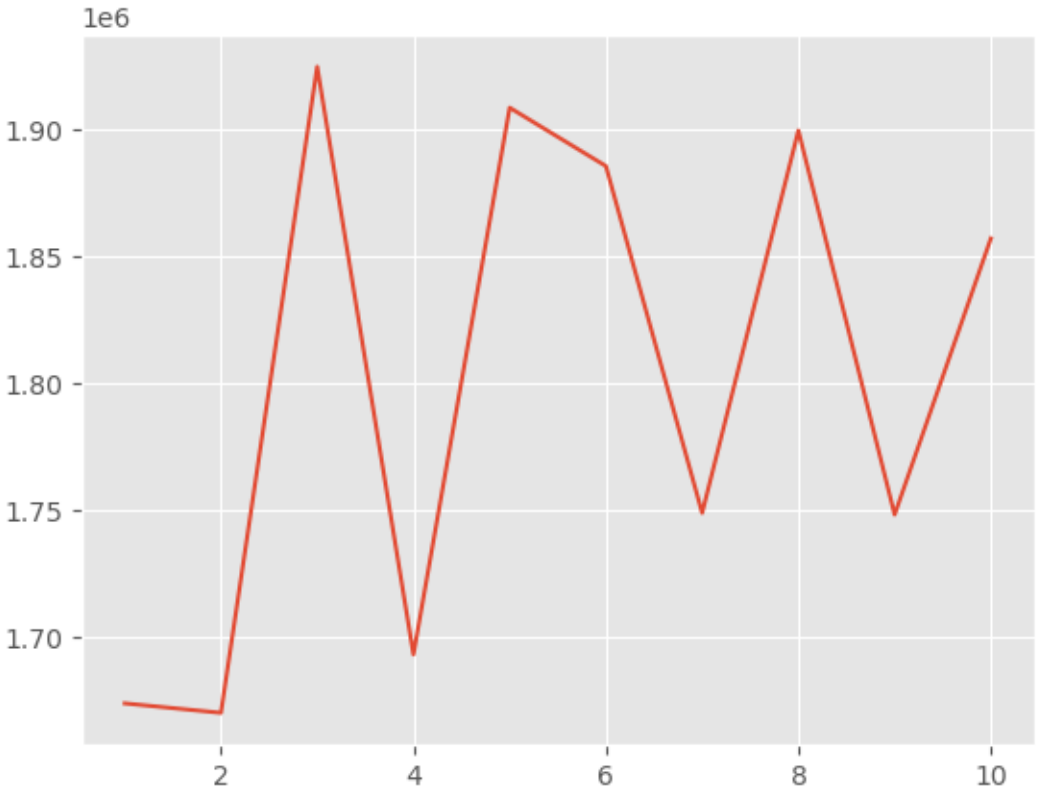
[18]:
```
plt.plot(df_com_ext_2023['tot_ope'].iloc[:50])
#plt.show()
```

[18]:
```
[<matplotlib.lines.Line2D at 0x18e9618b620>]
```



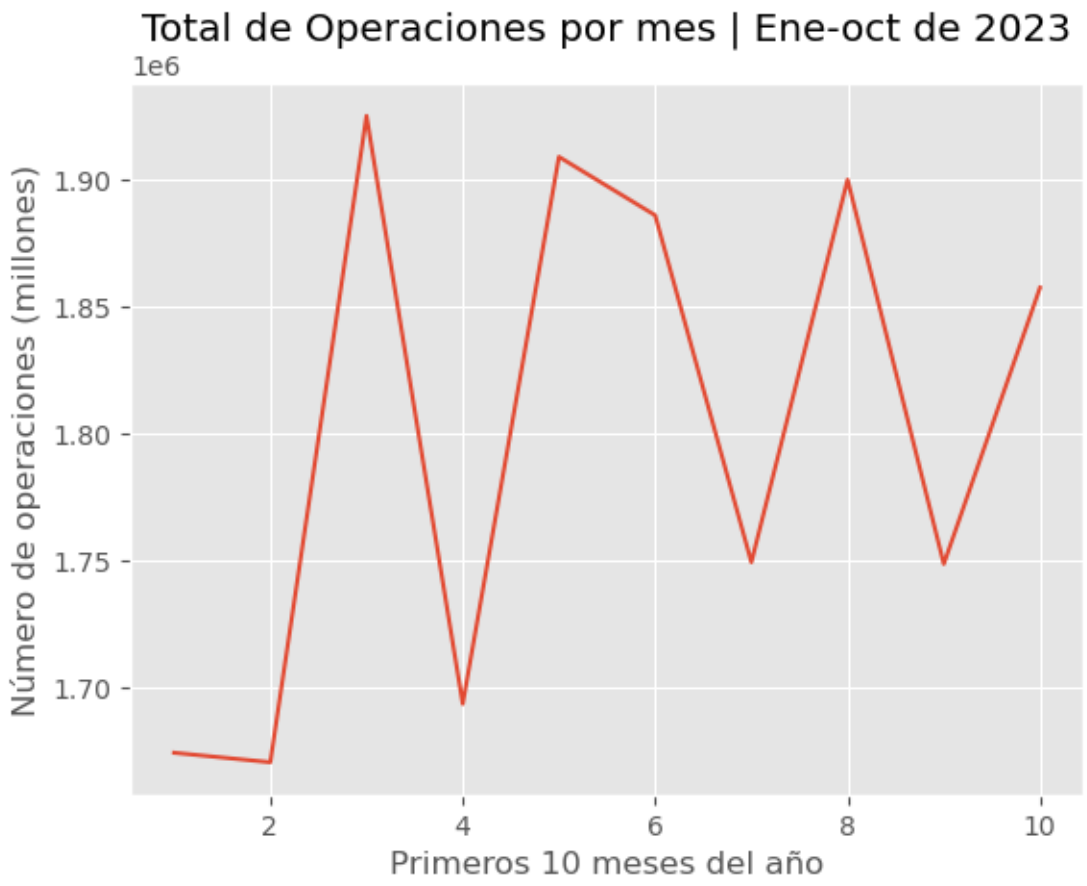
Para visualizar la suma del total de las operaciones por mes se hace uso de la función **groupby()** con el parámetro de la columna mes, y la suma de la columna total de operaciones (tot_ope).

[19]:
```
df_group_mes_ope = df_com_ext_2023.groupby('mes')[['tot_ope']].sum()
plt.plot(df_group_mes_ope)
plt.show()
```



Al gráfico anterior se agregan algunos elementos como: etiquetas de los ejes (X, Y) y titulo.

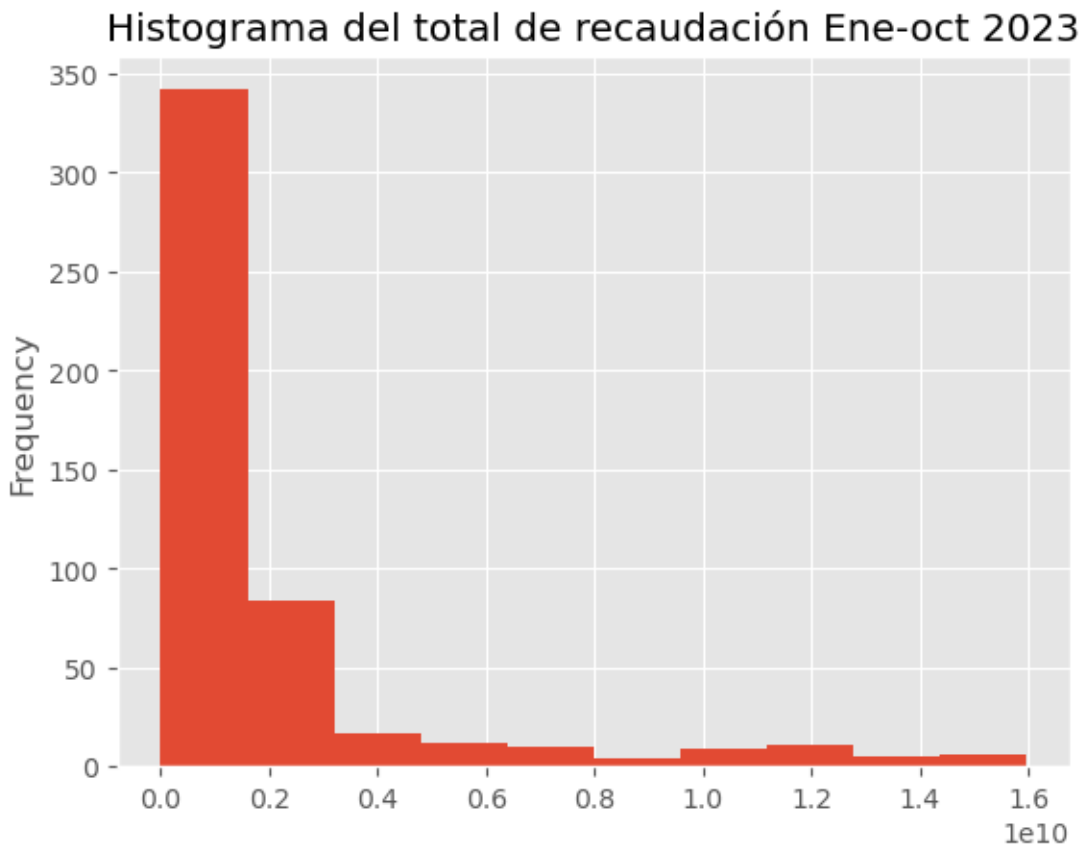
[20]:
```
plt.plot(df_group_mes_ope)
plt.xlabel('Primeros 10 meses del año') # Etiqueta del eje X
plt.ylabel('Número de operaciones (millones)') # Etiqueta del eje Y
plt.title('Total de Operaciones por mes | Ene-oct de 2023') # Titulo del gráfico
plt.show()
```



Más adelante se muestra como mejorar el gráfico con la capa Artist Layer.

Con el parámetro **kind=''** de la función plot() podemos asignar un tipo de gráfico, para el caso siguiente será un histograma con el total de la recaudación, pero pueden ser otros más, algunos se veran más adelante.

[21]:
```
df_com_ext_2023['total_rec'].plot(kind="hist")
plt.title("Histograma del total de recaudación Ene-oct 2023")
plt.show()
```



Para mostrar un gráfico de tipo área, se realiza una agrupación de datos por fecha, con los diferentes tipos de impuesto de la recaudación mediante una función de agregación de suma.

[22]:
```
df_group_fecha_rec = df_com_ext_2023.groupby('fecha')[['iva', 'igi', 'dta',␣
↪'ieps', 'isan', 'otros']].sum()
df_group_fecha_rec
```

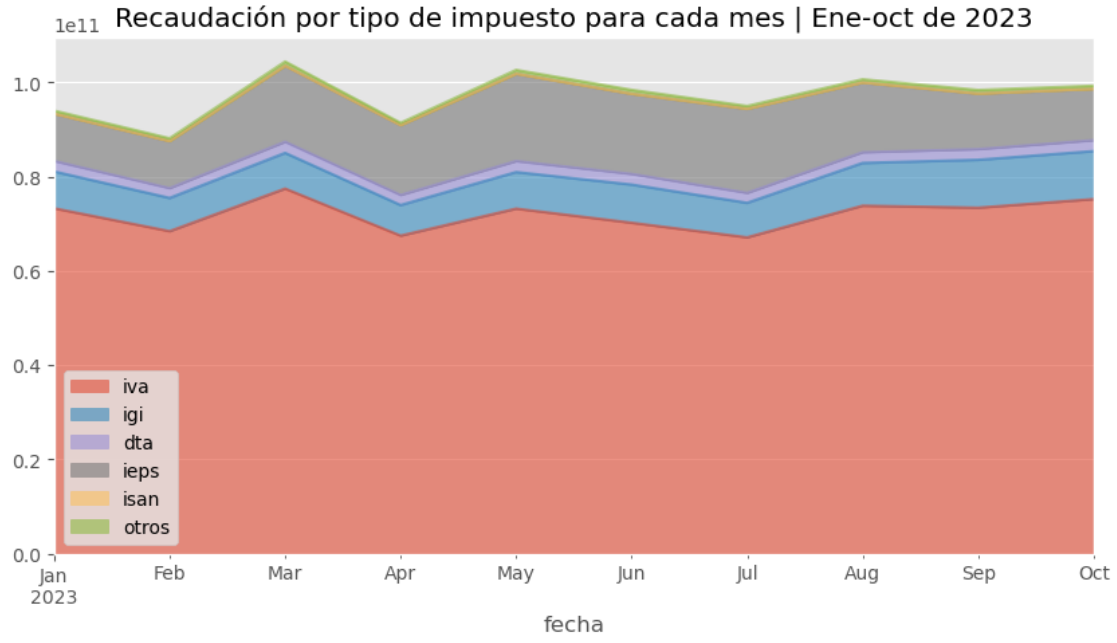
[22]:

fecha	iva	igi	dta	ieps	isan
2023-01-31	73240560482	7819901115	2229656025	10020273099	3249310
2023-02-28	68384554001	7058177383	2080309298	9974612522	4566140
2023-03-31	77423067360	7580069876	2363051546	16106319231	5559172
2023-04-30	67434729700	6497181191	2137277523	14820651953	2108212
2023-05-31	73170186674	7761569911	2338810351	18515106413	3766373
2023-06-30	70173567139	8106943495	2260934017	16985527381	5214726
2023-07-31	67073629479	7328308510	2112426403	17840293479	3076524
2023-08-31	73795161233	9075994684	2271289554	14788445112	3745578
2023-09-30	73368222391	10168432615	2264193718	11773501385	5022426
2023-10-31	75187177952	10187720568	2315240297	10881441255	3167082

```
                 otros
fecha
2023-01-31  664052482.0
2023-02-28  601043353.0
2023-03-31  920050358.0
2023-04-30  485908897.0
2023-05-31  829449191.0
2023-06-30  851900675.0
2023-07-31  598819963.0
2023-08-31  694088024.0
2023-09-30  773778582.0
2023-10-31  682318544.0
```

El siguiente gráfico se define mediante la función **subplots()**, esto quiere decir que se puede asignar varios subgráficos a una figura (fig) mediante una grilla; la figura se define con un tamaño de 10x5 pulgadas, para este caso solo contendra un gráfico instanciado con el nombre de **ax**, el cual se asigna con el parámetro del mismo nombre (ax=) en la función plot(). Más adelante se muestra un ejemplo con 4 gráficos dentro de una misma figura para su mejor comprensión.

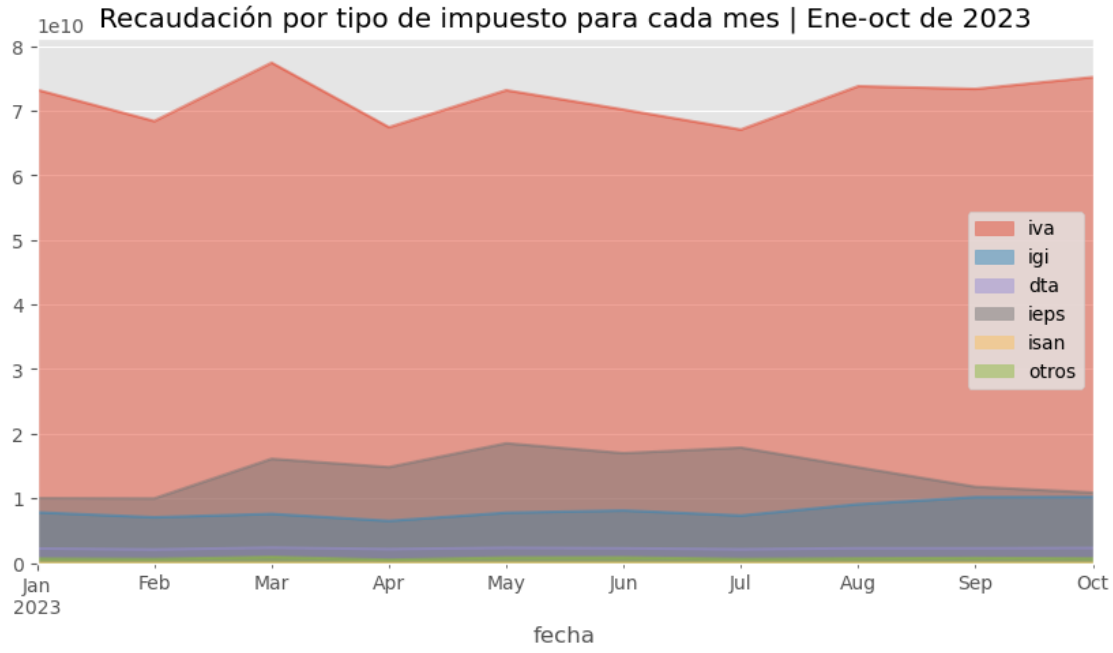
[23]:
```
fig, ax = plt.subplots(figsize=(10, 5)) # tamaño de la figura
df_group_fecha_rec.plot(kind='area', ax=ax, alpha=0.6) # tipo de gráfico,␣
 ↪asignación del subgrafico y transparencia
plt.legend() # leyenda o simbología
plt.title("Recaudación por tipo de impuesto para cada mes | Ene-oct de 2023")
plt.show()
```



Como se puede observar, en el gráfico los datos se muestran de forma apilada por defecto, esto

puede cambiarse con el parámetro **stacked=** en False.

[24]:
```
fig, ax = plt.subplots(figsize=(10, 5))
df_group_fecha_rec.plot(kind='area', ax=ax, alpha=0.5, stacked=False)
plt.legend()
plt.title("Recaudación por tipo de impuesto para cada mes | Ene-oct de 2023")
plt.show()
```

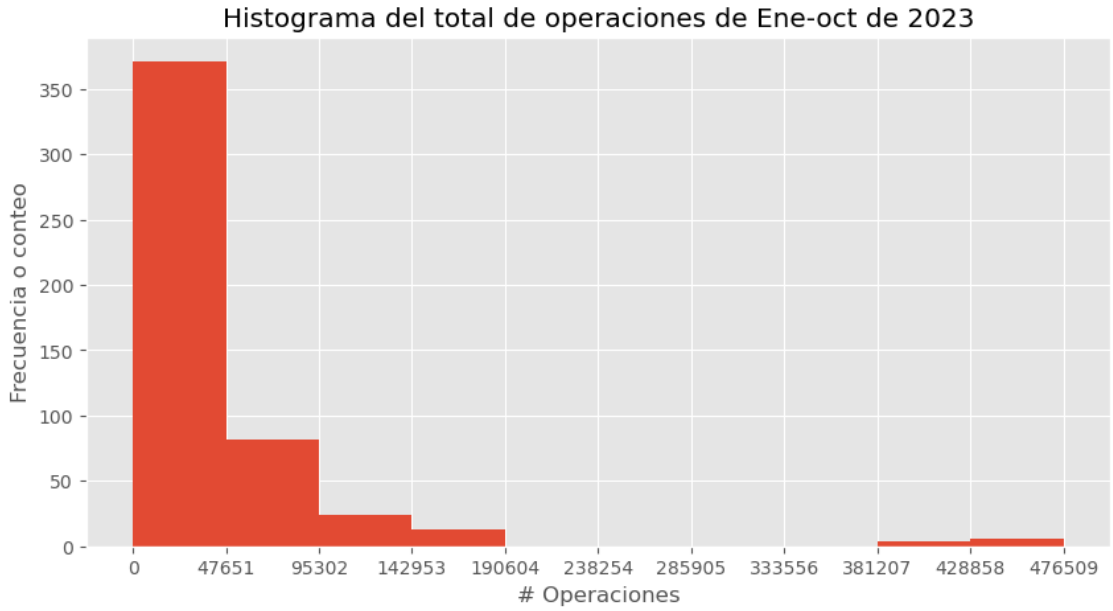


Enseguida se muestra un histograma realizado con el apoyo de numpy, en este se hace uso de los conteos o frecuencias (count) y de los contenedores (bins) así como de los bordes (edges) que proporciona la función, esto nos sirve para personalizar el gráfico, veamos el código:

[25]:
```
fig, ax = plt.subplots(figsize=(10, 5))
count, bin_edges = np.histogram(df_com_ext_2023['tot_ope'])
print("Contenedores o bins:", bin_edges)
print("Conteos:", count)
print("\n")
# creación del gráfico
df_com_ext_2023['tot_ope'].plot(kind='hist', xticks=bin_edges, ax=ax)
plt.title("Histograma del total de operaciones de Ene-oct de 2023")
plt.xlabel('# Operaciones')
plt.ylabel('Frecuencia o conteo')

plt.show()
```

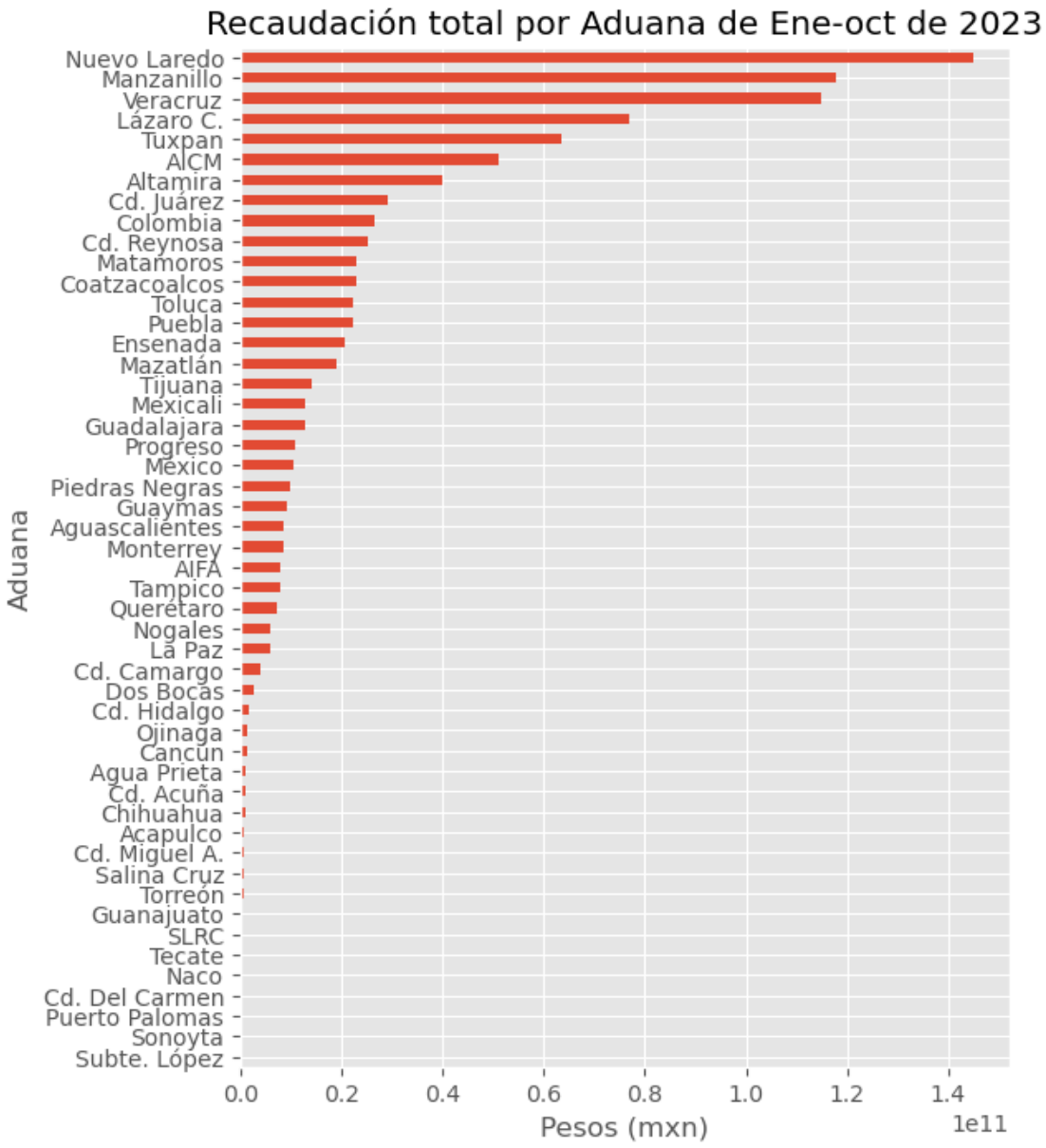
```
Contenedores o bins: [     0.   47650.9  95301.8 142952.7 190603.6 238254.5
285905.4 333556.3
```

```
 381207.2 428858.1 476509. ]
Conteos: [371  82  24  13   0   0   0   0   4   6]
```



El siguiente ejemplo corresponde al gráfico de tipo barra, para este caso se muestra de forma horizontal por la necesidad de que visualizar de mejor manera el nombre de las aduanas. En este caso se agrupa por aduana la suma del total de la recaudación y se ordenan los valores de forma descendente con la función **sort_values()** con el parámetro **ascending=** en False.

[26]:
```
# agrupar los datos de la recaudación total por aduana y ordenar de manera␣
 ↪descendente
df_rec_por_aduana = df_com_ext_2023.groupby('nombre_aduana')['total_rec'].sum().
 ↪sort_values(ascending=True)
# crear el gráfico
fig, ax = plt.subplots(figsize=(6, 8))
df_rec_por_aduana.plot(kind='barh', ax=ax)
plt.title('Recaudación total por Aduana de Ene-oct de 2023')
plt.ylabel('Aduana')
plt.xlabel('Pesos (mxn)')
plt.show()
```

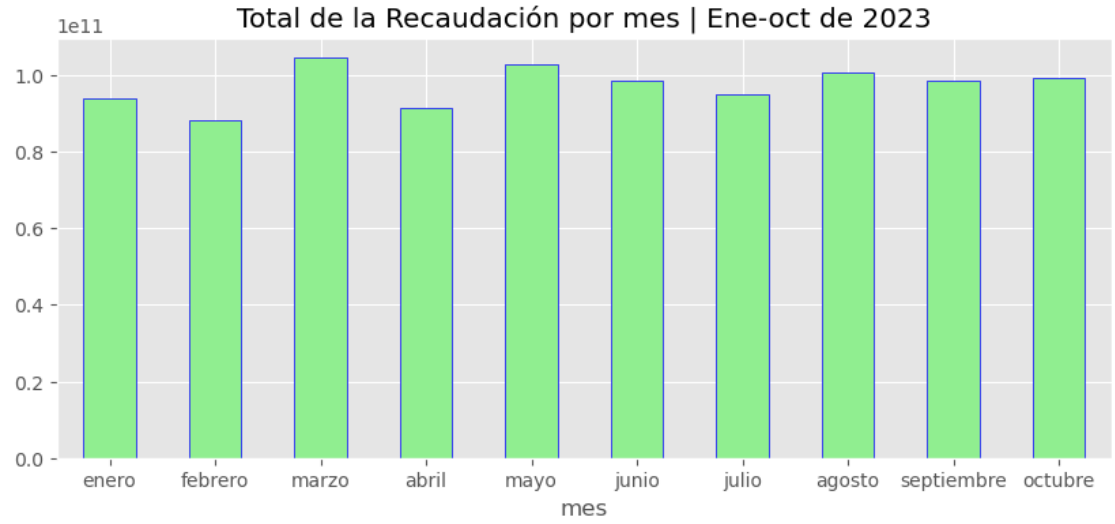


Ahora se muestra un ejemplo de cómo personalizar un gráfico de barras cambiando el número de mes con el nombre, asimismo, la asignación de colores a las barras y la rotación de etiquetas.

En inicio se realiza la agrupación por mes y se agrega la suma del total de recaudación dentro de un objeto o variable. Con el parámetro **edgecolor=** se define el color de contorno de las barras y el relleno mediante el parámetro **color=**, con la función **set_xticklabels()** se asigna mediante una lista o tupla los nombres a cada barra, con el parámetro **rotation=** se define el giro en grados de esta etiqueta, en este caso es cero porque no se requiere una rotación.

[27]:
```
df_rec_por_mes = df_com_ext_2023.groupby('mes')['total_rec'].sum()
```

```
fig, ax = plt.subplots(figsize=(10, 4))
df_rec_por_mes.plot(kind='bar', ax=ax, edgecolor='blue', color='lightgreen')
ax.set_xticklabels(['enero', 'febrero', 'marzo', 'abril', 'mayo', 'junio',␣
 ↪'julio', 'agosto', 'septiembre', 'octubre'], rotation=0)

plt.title('Total de la Recaudación por mes | Ene-oct de 2023')

plt.show()
```

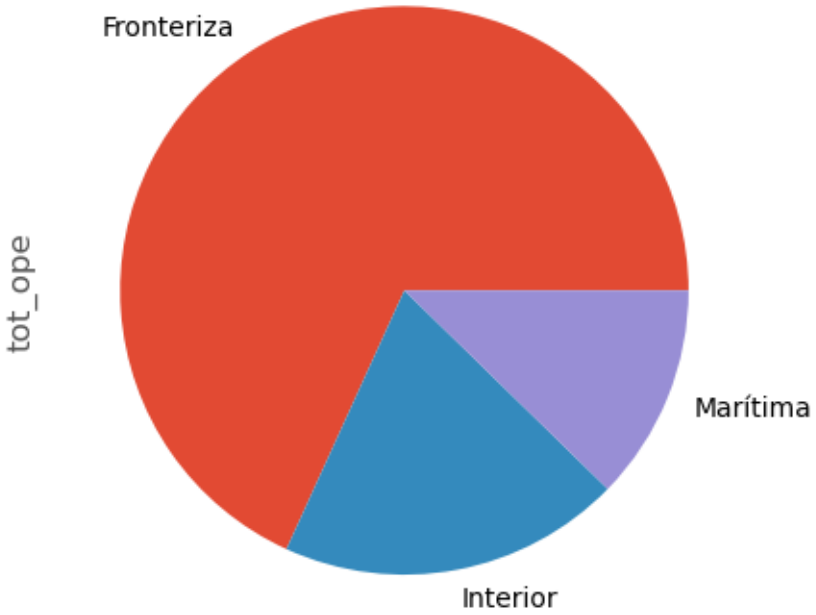


Nota: Para obtener una lista completa de los colores disponibles en Matplotlib puede correr el siguiente código:

[28]:
```
#import matplotlib
#for name, hex in matplotlib.colors.cnames.items():
#    print(name, hex)
```

A continuación, se muestra un gráfico de pastel o pay muy sencillo con la función plot() para el total de operaciones por tipo de aduana.

[29]:
```
df_com_ext_2023.groupby('tipo_aduana')['tot_ope'].sum().plot(kind='pie')
plt.show()
```



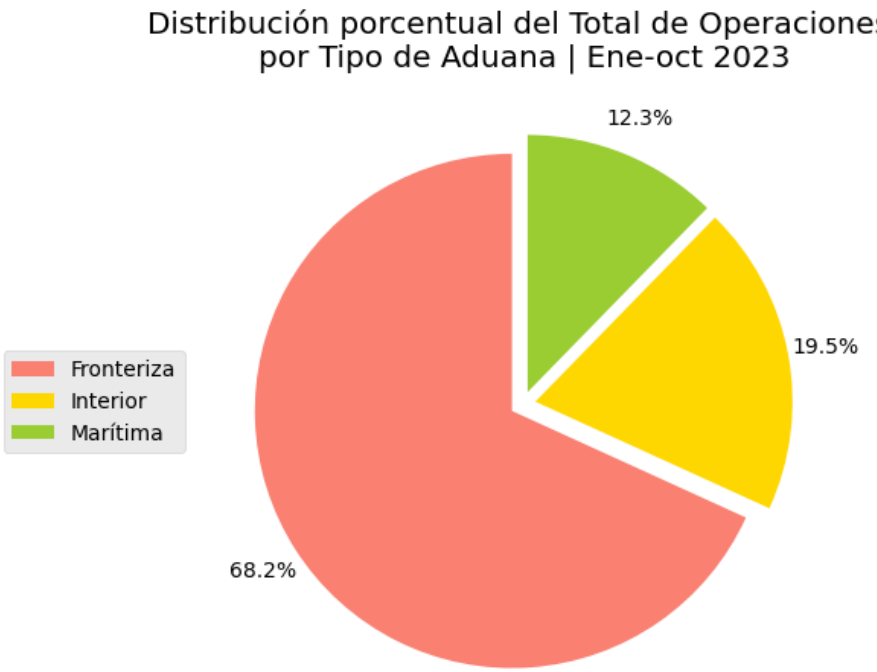
Si bien, los gráficos de pastel no son recomendables en el análisis de datos, ya que resulta muy confuso distinguir las proporciones de manera clara, pese a esto, a continuación, se ejemplifica la personalización de un gráfico de este tipo, lo primero es agrupar y agregar la suma de los datos a una variable, posteriormente se muestra el código y los parámetros necesarios.

[30]:
```
df_tipo_aduana = df_com_ext_2023.groupby(by='tipo_aduana')['tot_ope'].sum()
df_tipo_aduana
```

[30]:
```
tipo_aduana
Fronteriza    12283471
Interior       3512797
Marítima       2216413
Name: tot_ope, dtype: int64
```

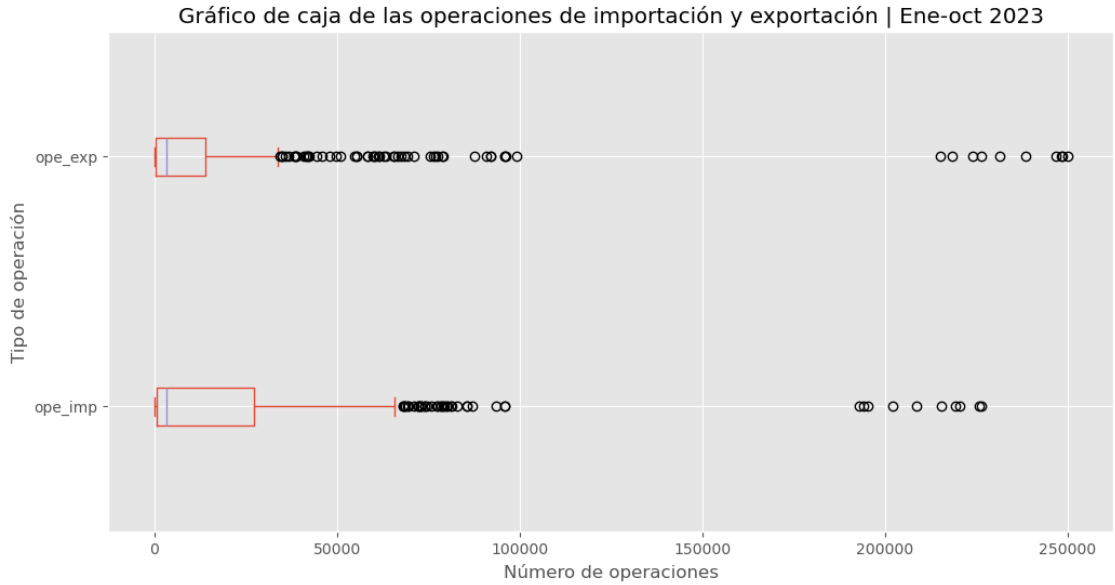
[31]:
```
df_tipo_aduana.plot(kind='pie',# Indica que queremos crear un gráfico de pastel
                    figsize=(9, 5), # Define el tamaño de la figura en pulgadas␣
 ↪(ancho, alto)
                    startangle=90, # Establece el ángulo inicial para la primera␣
 ↪porción del pastel (en grados)
                    autopct='%1.1f%%', # Muestra el porcentaje en cada porción␣
 ↪con un formato específico (1 decimal)
                    shadow=False, # Desactiva las sombras en las porciones del␣
 ↪pastel
```

```
                    labels=None, # No muestra etiquetas en las porciones
                    pctdistance=1.15, # Controla la distancia de los porcentajes␣
 ↪desde el centro del pastel
                    explode=[0.05, 0.05, 0.05], # Hace que las porciones se␣
 ↪separen ligeramente del centro (efecto de "explosión")
                    colors=['salmon', 'gold', 'yellowgreen'] # Define los␣
 ↪colores para cada porción
                    )

plt.title('Distribución porcentual del Total de Operaciones \npor Tipo de␣
 ↪Aduana | Ene-oct 2023', y=1.05) # y=1.05 ajusta la posición vertical del␣
 ↪título
plt.legend(labels=df_tipo_aduana.index, loc='center left') # La ubicación se␣
 ↪establece en la parte izquierda del gráfico
plt.ylabel(' ') # Elimina la etiqueta del eje y (vertical)
plt.axis('equal') # Hace que el gráfico sea circular (proporciones iguales en␣
 ↪ambos ejes)

plt.show()
```



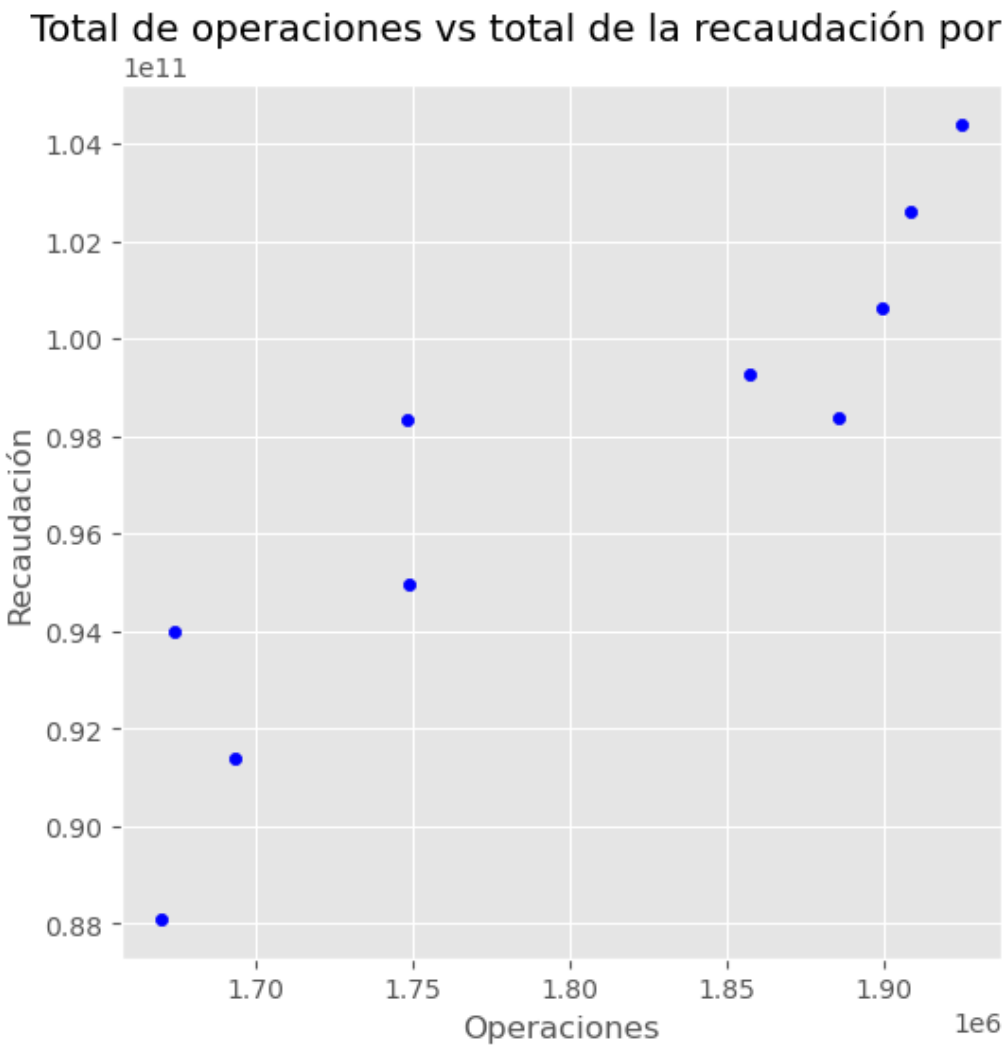
El siguiente gráfico es el boxplot o gráfico de caja y bigote, muy útil para visualizar la distribución de los datos y los outliers o datos atípicos, asimismo, para identificar 5 estadísticos como son: valor mínimo, 1er cuartil, 2do cuartil o mediana, 3er cuartil y valor máximo.

[32]:
```
df_com_ext_2023[['ope_imp', 'ope_exp']].plot(kind='box', figsize=(12, 6),␣
↪vert=False) # con vert=False se define los gráficos horizontales
plt.title('Gráfico de caja de las operaciones de importación y exportación |␣
↪Ene-oct 2023')
plt.xlabel('Número de operaciones')
plt.ylabel('Tipo de operación')

plt.show()
```



Otro gráfico importante para realizar un análisis estadístico es el gráfico de puntos o scatterplot, este gráfico nos sirve para encontrar relaciones entre dos variables, es decir, si hay una correlación. Para este ejemplo se gráfica la variable del total de operaciones en el eje X vs el total de la recaudación en el eje Y, lo anterior, agrupado por mes.

[33]:
```
df_ope_rec_mes = df_com_ext_2023.groupby('mes')[['total_rec', 'tot_ope']].sum()

df_ope_rec_mes.plot(kind='scatter',
                    x='tot_ope',
                    y='total_rec',
                    color='blue',
                    figsize=(6, 6)
                    )
plt.title('Total de operaciones vs total de la recaudación por mes')
plt.ylabel('Recaudación')
plt.xlabel('Operaciones')

plt.show()
```



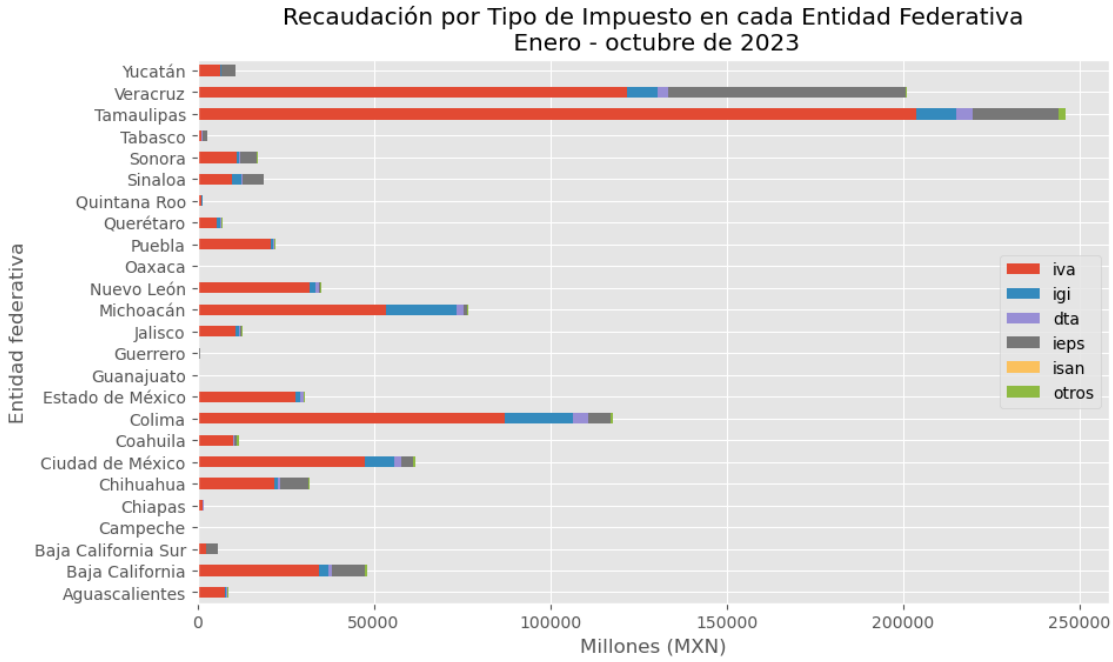
Otro gráfico muy útil es el de densidad de probabilidad, para crear este se proporciona la función **density()**, para nuestro ejemplo se usa la variable valor total de las operaciones (val_tot_ope) dividida en millones.

[34]:
```
(df_com_ext_2023['val_tot_ope']/1000000).plot.density(figsize=(7, 5))
plt.title('Densidad de probabilidad del Valor de las Operaciones')
plt.show()
```

El siguiente ejemplo será un gráfico nuevamente de barras, sin embargo, a diferencia del anterior se usará la función **barh()**; para la construcción de este gráfico se crea un daraframe con los datos agrupados por entidad federativa y la suma de cada tipo de impuesto dividido en millones, los datos serán apilados en cada barra.

[35]:
```
df_rec_ent_fed = df_com_ext_2023.groupby('entidad_federativa')[['iva', 'igi',␣
↪'dta', 'ieps', 'isan', 'otros']].sum()/1000000
```

[36]:
```
df_rec_ent_fed.plot.barh(figsize=(10, 6), stacked=True)
plt.ylabel('Entidad federativa')
plt.xlabel('Millones (MXN)')
plt.title('Recaudación por Tipo de Impuesto en cada Entidad Federativa\n Enero␣
↪- octubre de 2023')
plt.legend(loc='center right')

plt.show()
```



1.8 Personalización del gráfico

En los ejemplos anteriores se ha hecho uso en su mayoria de la capa de secuencias de comandos o Scripting layer, a continuación, se muestra el uso de la capa de artista o Artist Layer, un elemento muy importante para el uso de ésta es la instancia **Axes**(ax).

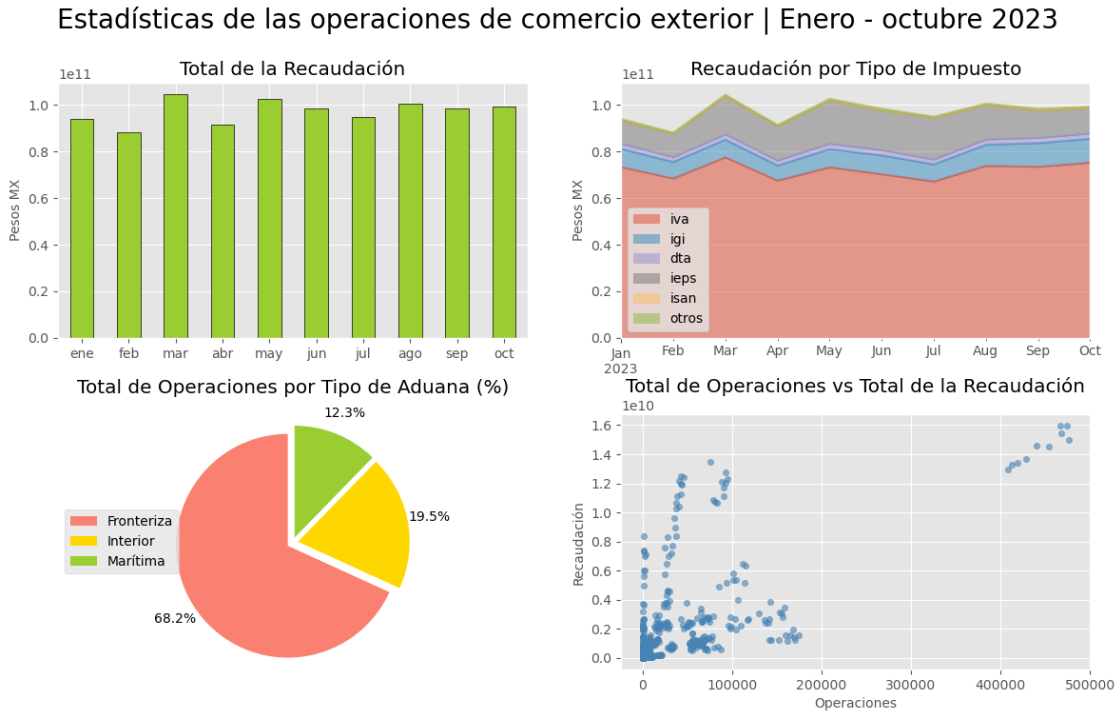
Algunas veces se necesitará crear multiples gráficos dentro de una misma figura para comparar algunos datos, para poder realizar ésto, lo primero es crear la figura y agregar a la misma los **subplots**, es decir, definir una grilla de acuerdo con la cantidad de filas y columnas que se deseen crear para los gráficos a visualizar.

Lo primero es crear la figura a partir de instanciar un objeto de nombre fig con el método **figure()** de matplotlib.pyplot (plt), después dividir y agregar los subplots con la función **add_subplot()**, asignar a una variable cada subplot, así como su posición mediante un número (tercer parámetro), para esto se pasa como parámetros el número de filas y columnas (primer y segundo parámetro respectivamente). Cada instancia Axes (ax) sigue una numeración inicial de izquierda-derecha y de arriba hacia abajo.

Veamos el ejemplo:

[37]:
```
# crear la figura
fig = plt.figure(figsize=(14, 8))
plt.subplots_adjust(hspace=0.3) # tamaño del espacio hori¿zontal entre gráficos
# asignación de los subplots a las variables
ax0 = fig.add_subplot(2, 2, 1) # 2 filas, 2 columnas, subplot número 1
ax1 = fig.add_subplot(2, 2, 2) # 2 filas, 2 columnas, subplot número 2
ax2 = fig.add_subplot(2, 2, 3) # 2 filas, 2 columnas, subplot número 3
```

```
ax3 = fig.add_subplot(2, 2, 4) # 2 filas, 2 columnas, subplot número 4
# subplot 1 gráfico de barras para el total de la recaudación por mes
df_rec_por_mes.plot(kind='bar', ax=ax0, edgecolor='k', color='yellowgreen')
ax0.set_title('Total de la Recaudación')
ax0.set_ylabel('Pesos MX', size=10)
ax0.set_xlabel(' ')
ax0.set_xticklabels(['ene', 'feb', 'mar', 'abr', 'may', 'jun', 'jul', 'ago',␣
 ↪'sep', 'oct'], rotation=0)
# subplot 2 gráfico de area de la recaudación por tipó de impuesto
df_group_fecha_rec.plot(kind='area', ax=ax1,  alpha=0.5)
ax1.set_title("Recaudación por Tipo de Impuesto")
ax1.set_xlabel(' ')
ax1.set_ylabel('Pesos MX', size=10)
#ax1.set_xticklabels(['ene', 'feb', 'mar', 'abr', 'may', 'jun', 'jul', 'ago',␣
 ↪'sep', 'oct'], rotation=0)
# subplot 3 gráfico de pay de la distribución porcentual de las operaciones por␣
 ↪tio de aduana
explode_1 = [0.05, 0.05, 0.05]
colors_1 = ['salmon', 'gold', 'yellowgreen']
df_tipo_aduana.plot(kind='pie', ax=ax2, startangle=90, autopct='%1.1f%%',␣
 ↪shadow=False, labels=None, pctdistance=1.2, explode=explode_1,␣
 ↪colors=colors_1)
ax2.set_title('Total de Operaciones por Tipo de Aduana (%)', y=1.05)
ax2.legend(labels=df_tipo_aduana.index, loc='center left')
ax2.set_ylabel(' ')
ax2.axis('equal')
# subplot 4 gráfico scatter del total de operaciones vs el total de la␣
 ↪recaudación
df_com_ext_2023.plot(kind='scatter', ax=ax3, x='tot_ope', y='total_rec',␣
 ↪color='steelblue', alpha=0.6)
ax3.set_title('Total de Operaciones vs Total de la Recaudación')
ax3.set_ylabel('Recaudación', size=10)
ax3.set_xlabel('Operaciones', size=10)

fig.suptitle('Estadísticas de las operaciones de comercio exterior | Enero -␣
 ↪octubre 2023', size=20)

plt.show()
```



1.9 Primeros pasos con seaborn

Seaborn es una libreria basada en matplotlib, la cual está enfocada principalmente al análisis estadístico, las visualizaciones creadas con esta libreria están bastante optimizadas, los gráficos creados con seaborn son muy atractivos visualmente, asimismo, el código es más sencillo si se compara con los gráficos anteriores creados con matplotlib.

A continuación, se ejemplifican algunos gráficos básicos para un análisis exploratorio del dataframe de las estadísticas de comercio exterior.

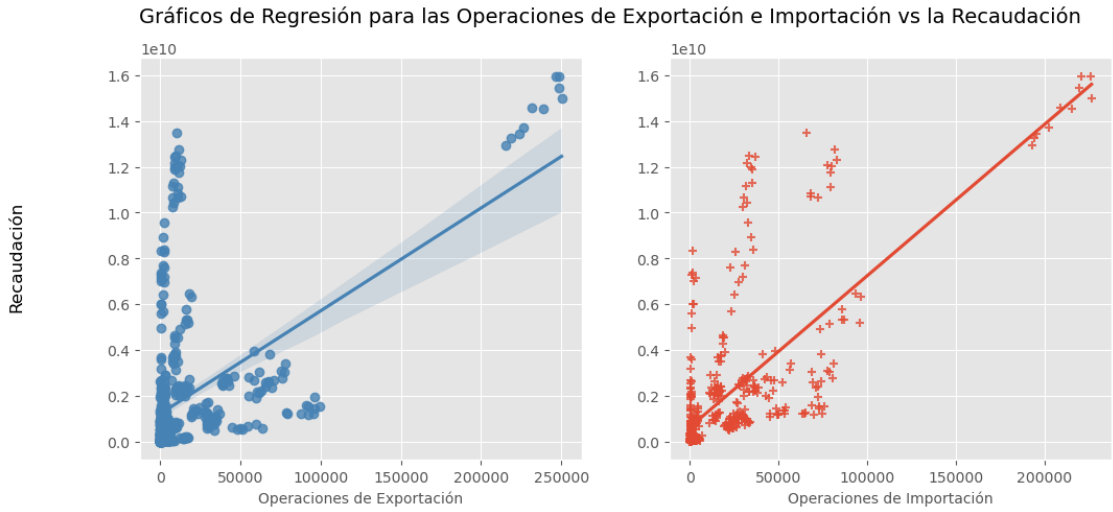
[38]:
```
# importar la libreria
import seaborn as sns
```

El ejemplo siguiente consiste en crear dos gráficos a partir de datos bivariados, estos son el número de operaciones de exportación y de importación vs la recaudación, el gráfico a mostrar es uno de regresión líneal, en éste, se muestra la relación entre variables, la línea de ajuste y el intervalo de confianza.

En el primer subgráfico se observa el número de operaciones de exportación en X como variable independiente y en Y la recaudación como variable dependiente, en el segundo en X las operaciones de importación y en Y se mantiene la recaudación, es este último no se muestra el intervalo de confianza.

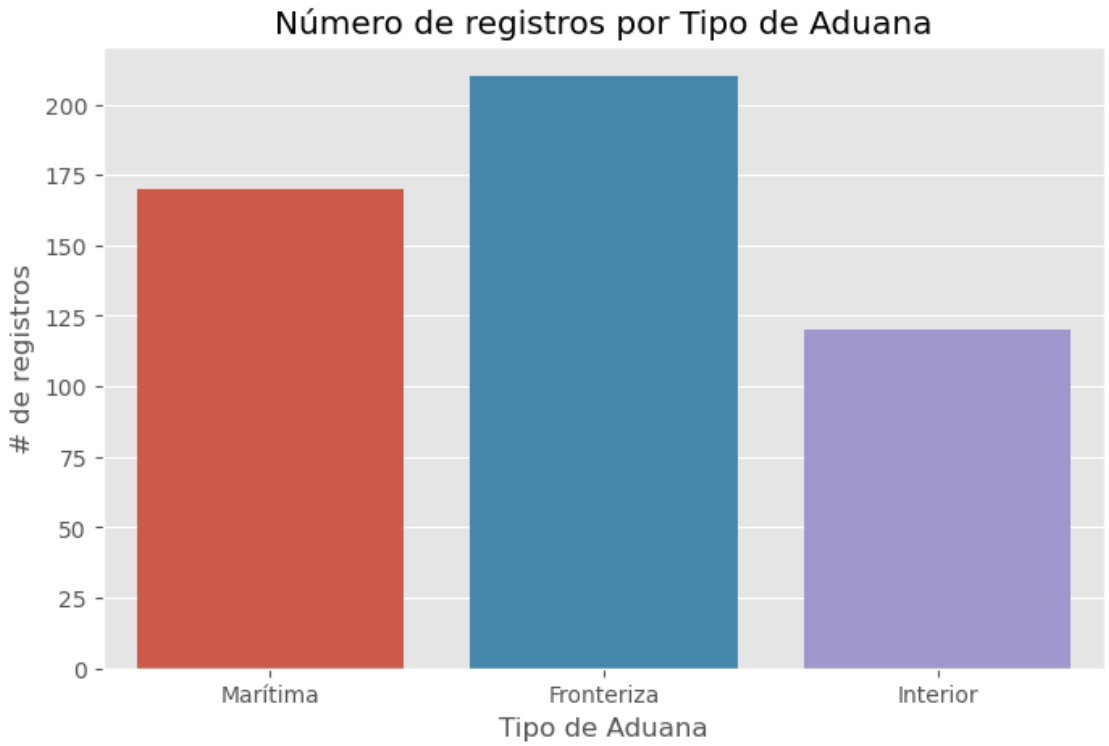
[39]:
```
fig = plt.figure(figsize=(12, 5))

ax1 = fig.add_subplot(1, 2, 1)
```

```
ax2 = fig.add_subplot(1, 2, 2)

sns.regplot(x='ope_exp', y='total_rec', data=df_com_ext_2023, ax=ax1,␣
 ↪color='steelblue')
ax1.set_xlabel('Operaciones de Exportación', size=10)
ax1.set_ylabel(None)

sns.regplot(x='ope_imp', y='total_rec', data=df_com_ext_2023, ax=ax2,␣
 ↪marker='+', ci=None)
ax2.set_xlabel('Operaciones de Importación', size=10)
ax2.set_ylabel(None)

fig.suptitle('Gráficos de Regresión para las Operaciones de Exportación e␣
 ↪Importación vs la Recaudación', size=14)
fig.supylabel('Recaudación')

plt.show()
```



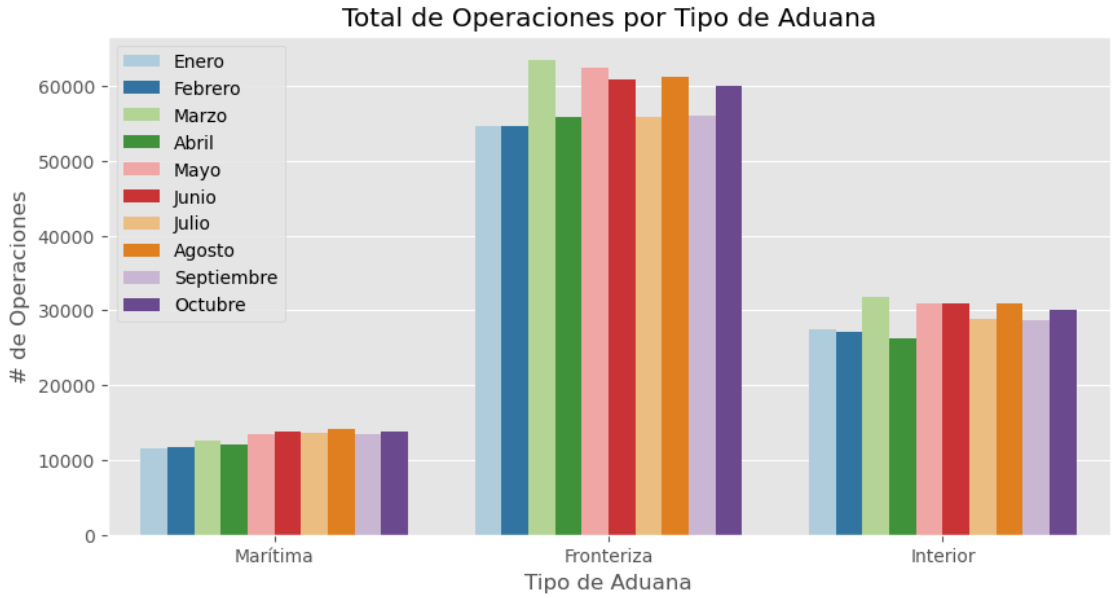
El siguiente gráfico se crea con la función **countplot()**, esta función crea un gráfico de barras con la frecuencia o conteo de una variable categórica, en este caso, la variable 'tipo de aduana' en relación al número de registros o filas del dataframe.

[40]:
```
fig = plt.figure(figsize=(8, 5))
ax = sns.countplot(x='tipo_aduana', data=df_com_ext_2023, hue='tipo_aduana')
ax.set_xlabel('Tipo de Aduana')
ax.set_title('Número de registros por Tipo de Aduana')
ax.set_ylabel('# de registros')

plt.show()
```



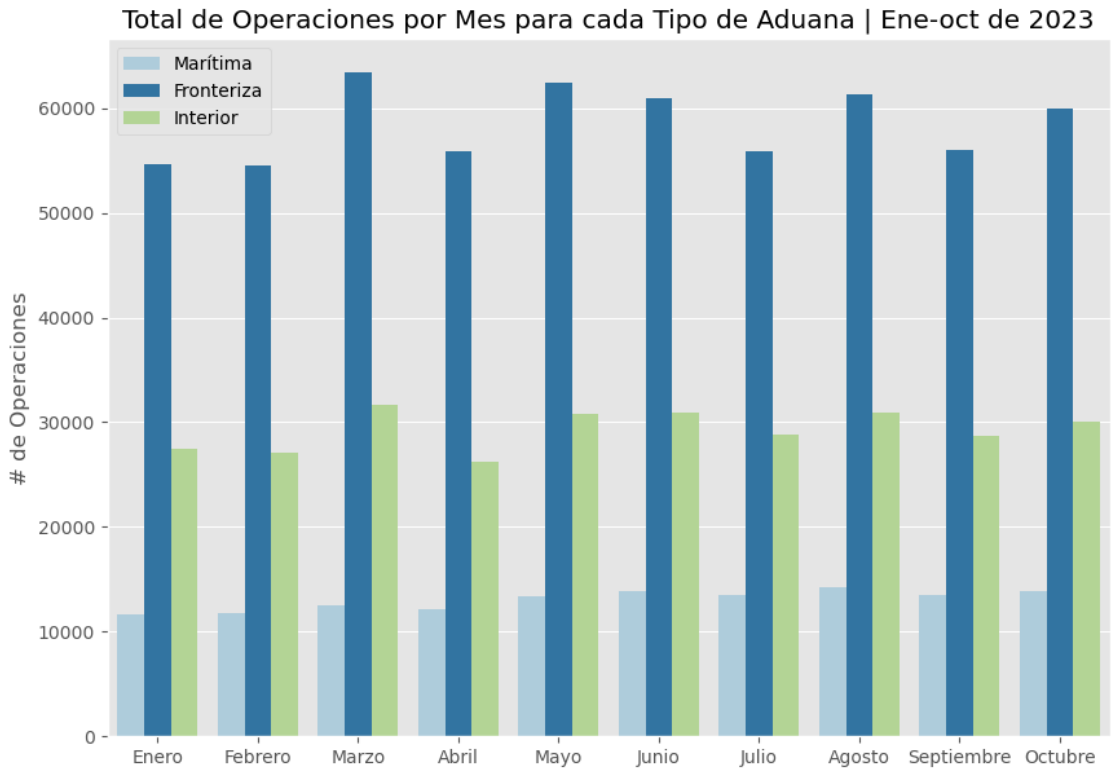
El siguiente gráfico es uno de barra creado con la función **barplot()**, muestra el total de operaciones por tipo de aduana pero se desagregan los datos por mes mediante el parámetro **hue=**, este gráfico aporta más información y detalle comparado con el anterior.

[41]:
```
fig = plt.figure(figsize=(10, 5))
ax = sns.barplot(x='tipo_aduana', y='tot_ope', hue='nombre_mes',␣
 ↪data=df_com_ext_2023, errorbar=None, palette="Paired")# palette asigna una␣
 ↪nueva paleta de colores
ax.set_title('Total de Operaciones por Tipo de Aduana')
ax.set_xlabel('Tipo de Aduana')
ax.set_ylabel('# de Operaciones')
ax.legend(loc='upper left')

plt.show()
```



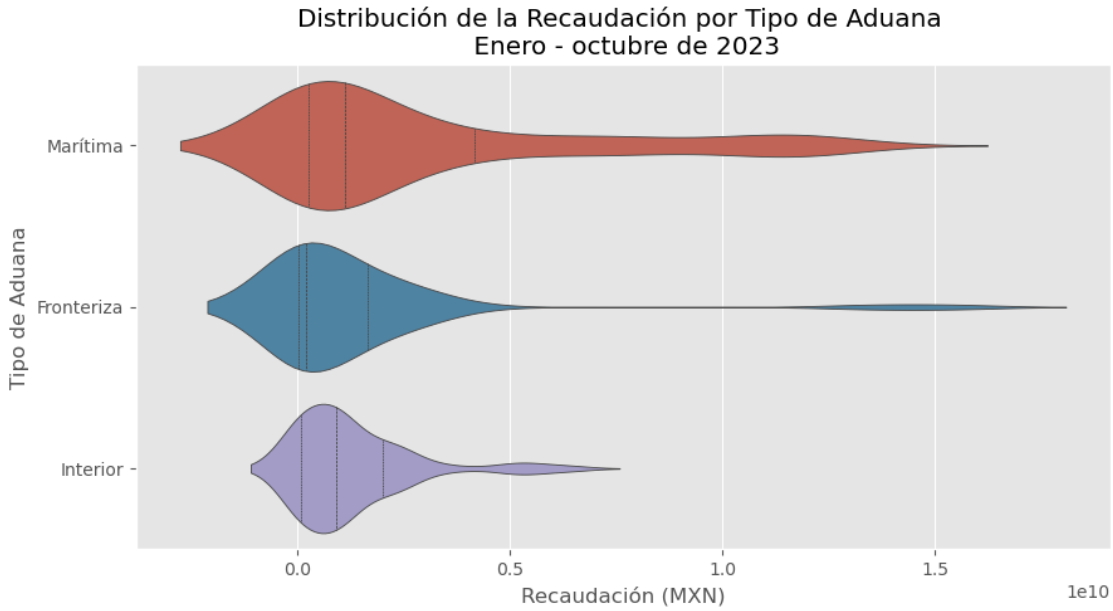
Una variante para el gráfico anterior con la misma función es el siguiente, se gráfica en el eje X los meses, en Y el total de operaciones, pero se hace una segmentación por tipo de aduana.

[42]:
```
fig = plt.figure(figsize=(10, 7))
ax = sns.barplot(x='nombre_mes', y='tot_ope', hue='tipo_aduana',␣
 ↪data=df_com_ext_2023, errorbar=None, palette="Paired")
ax.set_title('Total de Operaciones por Mes para cada Tipo de Aduana | Ene-oct␣
 ↪de 2023')
ax.set_xlabel(None)
ax.set_ylabel('# de Operaciones')
ax.legend()

plt.show()
```



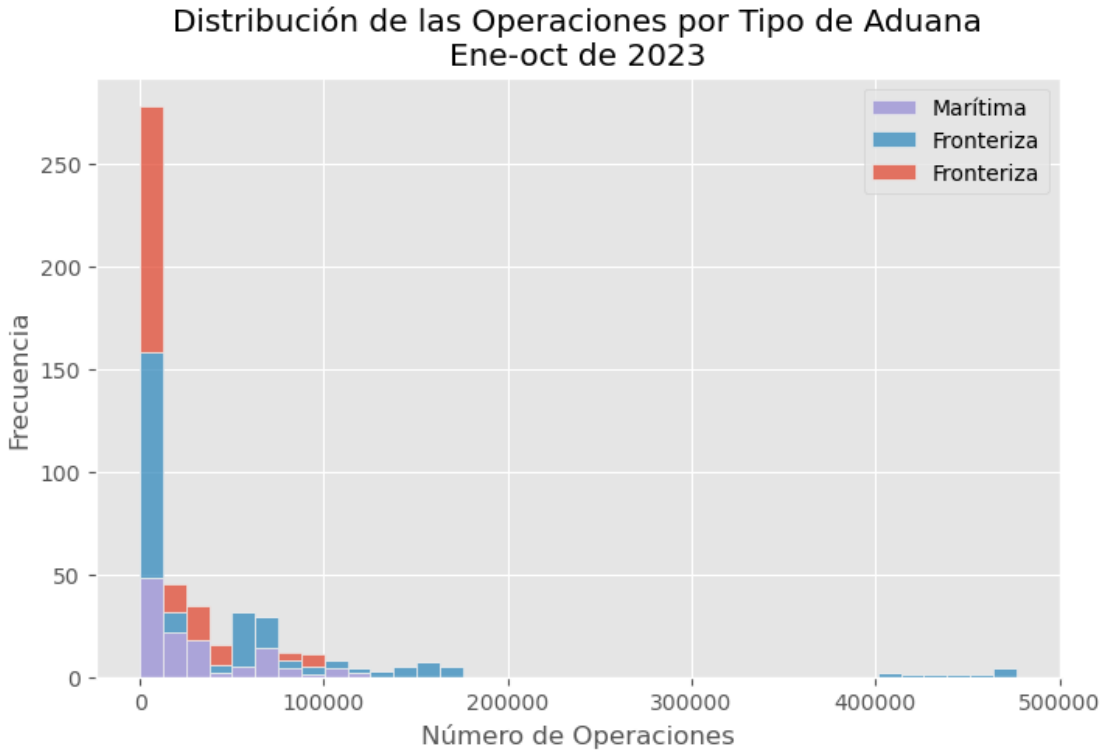
Un gráfico interesante y el cual aporta información comparable entre las distribuciones de datos cuantitativos es el gráfico de violin, veamos el ejemplo con la distribución de la recaudación por tipo de aduana.

[43]:
```
fig = plt.figure(figsize=(10, 5))
ax = sns.violinplot(data=df_com_ext_2023,
                    x="total_rec",
                    y="tipo_aduana",
                    inner="quart",
                    fill=True,
                    split=False,
                    hue="tipo_aduana",
                    saturation=0.6,
                   )
ax.set_xlabel('Recaudación (MXN)')
ax.set_ylabel('Tipo de Aduana')

ax.set_title('Distribución de la Recaudación por Tipo de Aduana \n Enero -␣
 ↪octubre de 2023')

plt.show()
```



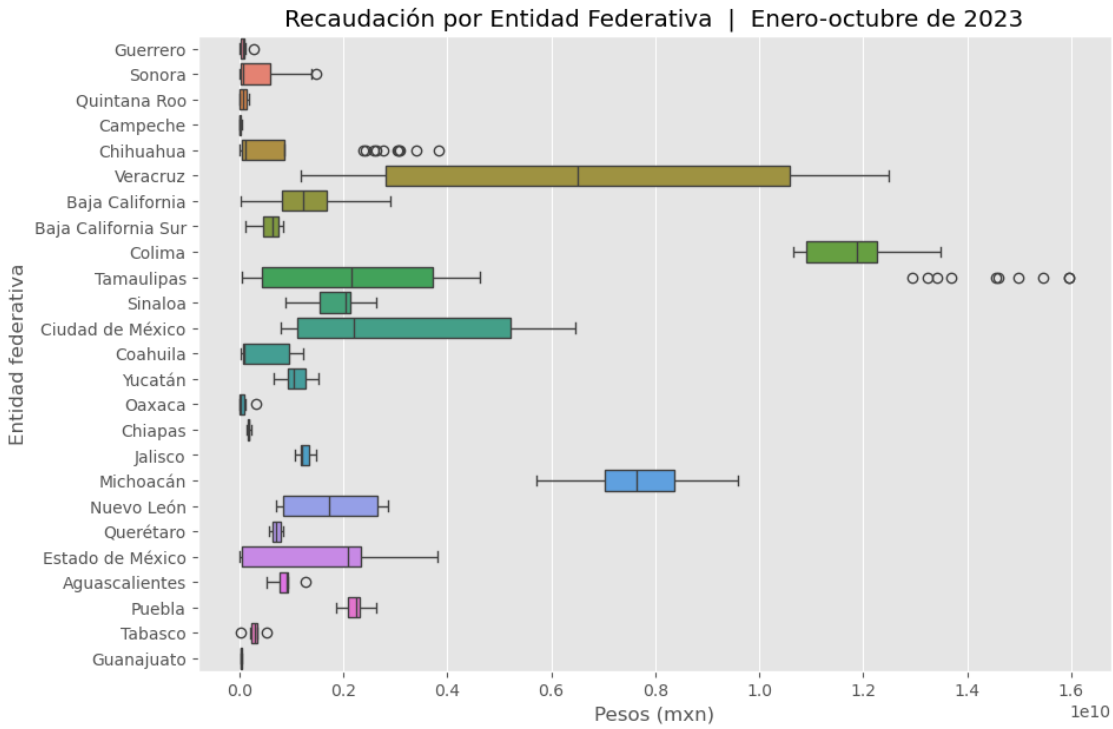
Un gráfico interesante es aquel que se crea con la función **histplot**, a diferencia del histograma visto, podemos pasar el parámetro **hue=** para ver la distribución por una variable categórica, en este ejemplo vemos la distribución de las operaciones por tipo de aduana.

[44]:
```
fig = plt.figure(figsize=(8, 5))

ax = sns.histplot(
                df_com_ext_2023,
                x="tot_ope",
                hue="tipo_aduana",
                multiple="stack"
                )
ax.legend(df_com_ext_2023.tipo_aduana)
ax.set_xlabel('Número de Operaciones')
ax.set_ylabel('Frecuencia')

plt.title('Distribución de las Operaciones por Tipo de Aduana\nEne-oct de 2023')
plt.show()
```



Finalmente para este apartado se crea un gráfico de caja y bigote para el total de la recaudación por entidad federativa con la dfunción **boxplot()**.

[45]:
```
fig= plt.figure(figsize=(10, 7))
ax = sns.boxplot(
                data=df_com_ext_2023,
                x="total_rec",
                y="entidad_federativa",
                hue='entidad_federativa'
                )

ax.set_xlabel("Pesos (mxn)")
ax.set_ylabel('Entidad federativa')
plt.title('Recaudación por Entidad Federativa  |  Enero-octubre de 2023')

plt.show()
```



Puede consultar muchos más tipos de graficos en la documentación oficial.

1.10 Primeros pasos con folium

Como se mencionó al inicio, Folium es una poderosa librería de visualización de datos geoespaciales en Python. Con Folium, se puede crear mapas de cualquier lugar del mundo y en diferentes estilos gracias a la variedad de mosaicos provistos en la siguiente liga. Folium es en realidad un "envoltura" de python para leaflet.js, misma que es una libreria de javascript para crear mapas interactivos.

[46]:
```
# importar la librería y módulos necesarios
import folium
from folium import plugins
from folium.plugins import MarkerCluster
```

Lo primero será crear un mapa el cual se asigna a un objeto de nombre 'mex_map_aduanas', pues en este se visualizarán los puntos que representan las 50 aduanas de México, el parámetro **location=** recibe como valor una lista con las coordenadas del lugar central a visualizar, en este caso el territorio nacional de México, el siguiente parámetro **zoom_start=** recibe un valor número con el nivel de acercamiento.

[47]:
```
mex_map_aduanas = folium.Map(
    location=[23.52, -102.52],
    zoom_start=5
    )
#folium.Marker(location=[19.2188, -99.1265], popup='Tecalipa').add_to(mex_map)
```

```
mex_map_aduanas
```

[47]: <folium.folium.Map at 0x18e9bf96660>

El siguiente proceso es importar la librería geopandas, con ella es posible leer archivos de datos geoespaciales, en este caso se trata de un archivo 'geopackage' creado en QGIS.

[48]:
```
import geopandas as gpd
```

[49]:
```
aduanas_geo = gpd.read_file('../Data/aduanas_mx.gpkg')
aduanas_geo.head()
```

[49]:

	cve_aduana	tipo_aduana	entidad_federativa	nombre_aduana	latitud_y	longitud_x	geometry
0	1	Marítima	Guerrero	Acapulco	16.84864	-99.90517	POINT (-99.90517 16.84864)
1	2	Fronteriza	Sonora	Agua Prieta	31.33370	-109.56082	POINT (-109.56082 31.33370)
2	73	Interior	Aguascalientes	Aguascalientes	22.00755	-102.25071	POINT (-102.25071 22.00755)
3	47	Interior	Ciudad de México	AICM	19.44654	-99.07066	POINT (-99.07066 19.44654)
4	85	Interior	Estado de México	AIFA	19.75307	-99.00563	POINT (-99.00563 19.75307)

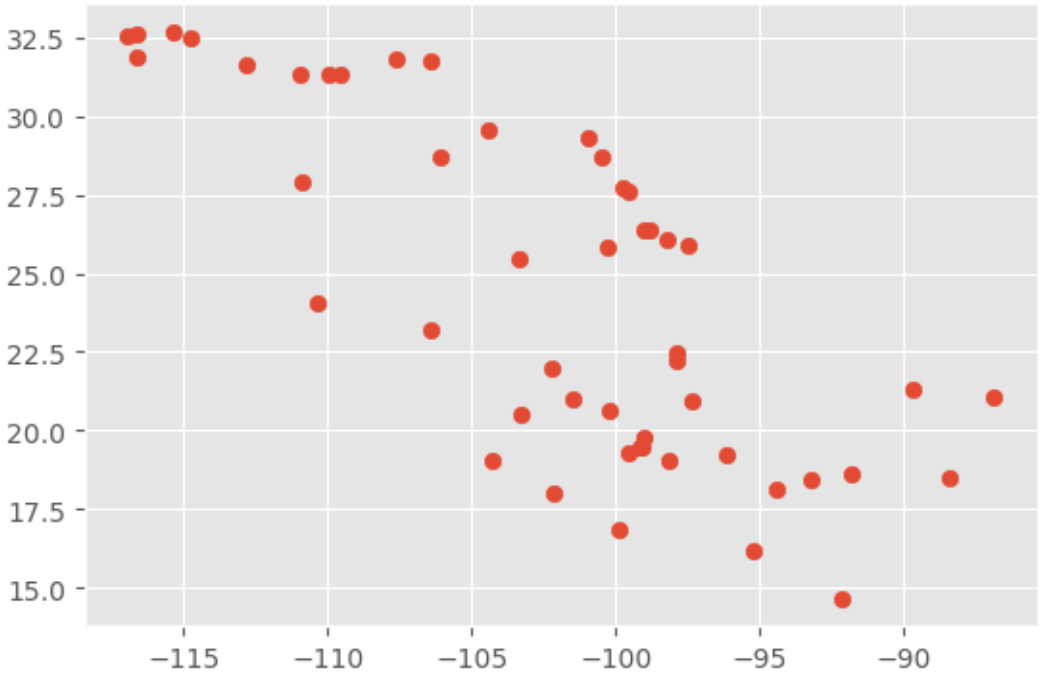
[50]:
```
aduanas_geo.info()
```

```
<class 'geopandas.geodataframe.GeoDataFrame'>
RangeIndex: 50 entries, 0 to 49
Data columns (total 7 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   cve_aduana          50 non-null     int64
 1   tipo_aduana         50 non-null     object
 2   entidad_federativa  50 non-null     object
 3   nombre_aduana       50 non-null     object
 4   latitud_y           50 non-null     float64
 5   longitud_x          50 non-null     float64
 6   geometry            50 non-null     geometry
dtypes: float64(2), geometry(1), int64(1), object(3)
memory usage: 2.9+ KB
```

Gracias a que geopandas crea una columna 'geometry' es posible representar en el espacio con un marco de referencia geográfico los puntos o coordenadas de cada aduana.

[51]:
```
aduanas_geo.plot()
```

[51]: <Axes: >



Ahora se creará un dataframe agrupado por la clave de aduana con la suma de las variables de recaudación, operaciones y valor de las operaciones con base en el dataframe usado en las secciones anteriores.

[52]:
```
df_group_cve_aduana = df_com_ext_2023.groupby('cve_aduana')[['total_rec',␣
↪'ope_imp', 'ope_exp', 'tot_ope', 'val_ope_imp', 'val_ope_exp',␣
↪'val_tot_ope']].sum()
```

[53]:
```
df_group_cve_aduana.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 50 entries, 1 to 85
Data columns (total 7 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   total_rec    50 non-null     float64
 1   ope_imp      50 non-null     int64
 2   ope_exp      50 non-null     int64
 3   tot_ope      50 non-null     int64
 4   val_ope_imp  50 non-null     float64
 5   val_ope_exp  50 non-null     float64
 6   val_tot_ope  50 non-null     float64
dtypes: float64(4), int64(3)
```

```
memory usage: 3.1 KB
```

[54]:
```
df_group_cve_aduana.head()
```

[54]:

cve_aduana	total_rec	ope_imp	ope_exp	tot_ope	val_ope_imp	val_ope_exp	val_tot_ope
1	8.423561e+08	30	0	30	1.573719e+09	0.000000e+00	1.573719e+09
2	1.214492e+09	14399	19035	33434	1.528610e+10	3.114999e+10	4.643609e+10
5	2.599892e+07	656	8323	8979	5.485813e+08	3.544482e+09	4.093063e+09
6	1.533195e+08	492	310	802	1.784228e+10	8.219237e+10	1.000347e+11
7	2.923040e+10	747201	705553	1452754	1.337743e+12	1.461420e+12	2.799163e+12

Lo siguiente será unir los datos agrupados del dataframe anterior al geodataframe de las aduanas, esto se logra con la función **merge()** del mismo objeto de geopandas, para la coincidencia de registros se usa la columna clave de aduana (cve_aduana).

[55]:
```
df_group_merge_cve = aduanas_geo.merge(df_group_cve_aduana, on='cve_aduana')
```

[56]:
```
df_group_merge_cve.info()
```

```
<class 'geopandas.geodataframe.GeoDataFrame'>
RangeIndex: 50 entries, 0 to 49
Data columns (total 14 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   cve_aduana          50 non-null     int64
 1   tipo_aduana         50 non-null     object
 2   entidad_federativa  50 non-null     object
 3   nombre_aduana       50 non-null     object
 4   latitud_y           50 non-null     float64
 5   longitud_x          50 non-null     float64
 6   geometry            50 non-null     geometry
 7   total_rec           50 non-null     float64
 8   ope_imp             50 non-null     int64
 9   ope_exp             50 non-null     int64
 10  tot_ope             50 non-null     int64
 11  val_ope_imp         50 non-null     float64
 12  val_ope_exp         50 non-null     float64
 13  val_tot_ope         50 non-null     float64
dtypes: float64(6), geometry(1), int64(4), object(3)
memory usage: 5.6+ KB
```

[57]:
```
df_group_merge_cve.head()
```

[57]:

	cve_aduana	tipo_aduana	entidad_federativa	nombre_aduana	latitud_y	longitud_x	geometry	total_rec	ope_imp	ope_exp
0	1	Marítima	Guerrero	Acapulco	16.84864	-99.90517	POINT (-99.90517 16.84864)	8.423561e+08	30	0
1	2	Fronteriza	Sonora	Agua Prieta	31.33370	-109.56082	POINT (-109.56082 31.33370)	1.214492e+09	14399	19035
2	73	Interior	Aguascalientes	Aguascalientes	22.00755	-102.25071	POINT (-102.25071 22.00755)	8.734347e+09	34059	39776
3	47	Interior	Ciudad de México	AICM	19.44654	-99.07066	POINT (-99.07066 19.44654)	5.102574e+10	808467	149937
4	85	Interior	Estado de México	AIFA	19.75307	-99.00563	POINT (-99.00563 19.75307)	7.928747e+09	127130	21693

	tot_ope	val_ope_imp	val_ope_exp	val_tot_ope
0	30	1.573719e+09	0.000000e+00	1.573719e+09
1	33434	1.528610e+10	3.114999e+10	4.643609e+10
2	73835	9.433582e+10	1.009644e+11	1.953002e+11
3	958404	5.428277e+11	2.768450e+11	8.196726e+11
4	148823	6.767349e+10	2.066624e+10	8.833973e+10

Para tener una mayor de referencia a nivel entidad federativa en la visualización de las aduanas, se lee un archivo con las geometrías de las entidades federativas a través de geopandas.

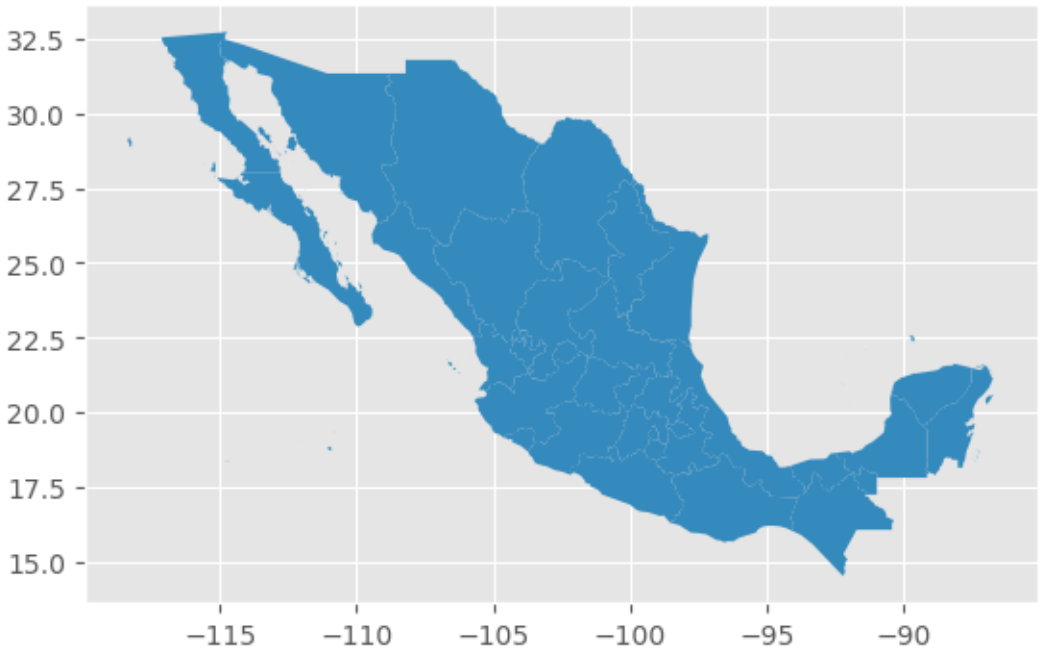
[58]:
```
ent_fed_geo = gpd.read_file('../Data/entidades_fed_simplif.gpkg')
ent_fed_geo.head()
```

[58]:

	CVEGEO	CVE_ENT	NOMGEO	geometry
0	01	01	Aguascalientes	MULTIPOLYGON (((-102.29739 22.45527, -102.2532…
1	02	02	Baja California	MULTIPOLYGON (((-114.12174 28.09477, -114.1171…
2	03	03	Baja California Sur	MULTIPOLYGON (((-109.89425 22.87494, -109.8946…
3	04	04	Campeche	MULTIPOLYGON (((-91.95774 20.19674, -91.95547 …
4	05	05	Coahuila de Zaragoza	MULTIPOLYGON (((-102.31549 29.88000, -102.0722…

[59]:
```
ent_fed_geo.plot()
```

[59]:
```
<Axes: >
```



Lo siguiente será cargar el archivo de las entidades al mapa base, para ello se le asigna un estilo, donde no tendra color de relleno y el cortorno de cada entidad será en color gris, para asignar capa de entidades al mapa se hace uso de la función **GeoJson()** de folium, esta recibe como parámetro el nombre del archivo o capa, así como el estilo, finalmente con la función **add_to()** se agregar al mapa que se desea.

[60]:
```
# Función de estilo para los polígonos sin relleno
style_function = lambda x: {
    'fillColor': 'none',
    'color': 'gray'
}

# Agregar la capa de polígonos al mapa
folium.GeoJson(ent_fed_geo, style_function=style_function).
 ↪add_to(mex_map_aduanas)

mex_map_aduanas
```

[60]:
```
<folium.folium.Map at 0x18e9bf96660>
```

Lo siguiente será cargar los puntos de las aduanas, para esto nos apoyaremos del método **FeatureGroup()** de la función **map()** de folium, con esta podemos asignar un grupo de características a ser visualizadas, es decir, las 50 aduanas con n columnas; mediante un bucle for asignamos valores a 8 variables, los valores son: latitud, longitud, nombre de la aduana, tipo de aduana, entidad federativa, recaudación, operaciones y valor de las operaciones. Los valores extraídos sirven de parámetro para su representación e interacción con el usuario, el cual podrá ver las caractrerísticas

extraídas.

[75]:
```
loc_adua = folium.map.FeatureGroup()

for lat, lng, nombre, tipo, entidad, recaudacion, operaciones, valor in␣
 ↪zip(df_group_merge_cve.latitud_y, df_group_merge_cve.longitud_x,␣
 ↪df_group_merge_cve.nombre_aduana, df_group_merge_cve.tipo_aduana,␣
 ↪df_group_merge_cve.entidad_federativa, df_group_merge_cve.total_rec,␣
 ↪df_group_merge_cve.tot_ope, df_group_merge_cve.val_tot_ope):
    loc_adua.add_child(
        folium.features.CircleMarker(
            [lat, lng],
            radius=6,
            color='steelblue',
            fill=True,
            fill_color='gold',
            fill_opacity=0.7,
            popup=f'Tipo de Aduana:{tipo} | \nEntidad Federativa:{entidad} |␣
 ↪\nRecaudación:{recaudacion} | \nNúmero de Operaciones:{operaciones} |␣
 ↪\nValor de las Operaciones:{valor}',
            tooltip=nombre
        )
    )

mex_map_aduanas.add_child(loc_adua)
```

[75]:
```
<folium.folium.Map at 0x18e9bf96660>
```

Como se mencionó, existen diversos mosaicos o estilos de mapas, en el ejemplo siguiente usaremos un mapa satelital, para crear dicho mapa se necesita agreagr dos parámetros a la función **Map()** de folium, éstos son: **attr** y **tiles**, el código siguiente muestra lo explicado.

[76]:
```
attr = ('Tiles &copy; Esri &mdash; Source: Esri, i-cubed, USDA, USGS, AEX,␣
 ↪GeoEye, Getmapping, Aerogrid, IGN, IGP, UPR-EGP, and the GIS User␣
 ↪Community')#attribution
tiles = 'https://server.arcgisonline.com/ArcGIS/rest/services/World_Imagery/
 ↪MapServer/tile/{z}/{y}/{x}' #L.tileLayer
mex_map_sat = folium.Map(location=[23.52, -102.52], zoom_start=4, tiles=tiles,␣
 ↪attr=attr)

mex_map_sat
```

[76]:
```
<folium.folium.Map at 0x18ea0944230>
```

Nuevamente para tener una referencia de la localización de las aduanas en las entidades federativas se agrega al mapa la capa de entidades.

[77]:
```
# Función de estilo para los polígonos sin relleno
style_function = lambda x: {
    'fillColor': 'none',
    'color': 'white'
}

# Agregar la capa de polígonos al mapa
folium.GeoJson(ent_fed_geo, style_function=style_function).add_to(mex_map_sat)

mex_map_sat
```

[77]:
```
<folium.folium.Map at 0x18ea0944230>
```

Para este ejemplo se visualizará un mapa de cluster por localización, este tipo de mapa resulta muy útil para visualizar agrupamientos, muy usado en la identificación de incidenias delictivas.

[79]:
```
# instanciar objeto de marcador de cluster
clust_aduanas = plugins.MarkerCluster().add_to(mex_map_sat)

# Iterar sobre los datos del geodataframe (aduanas)
for index, row in aduanas_geo.iterrows():
    lat, lng, nombre, tipo = row['latitud_y'], row['longitud_x'],␣
 ↪row['nombre_aduana'], row['tipo_aduana']  # Obtener latitud, longitud,␣
 ↪nombre y tipo
    folium.Marker(
        location=[lat, lng],
        popup=tipo,  # Asignar el nombre como texto del popup
        icon=None,
        tooltip=nombre
    ).add_to(clust_aduanas)

# Mostrar el mapa
mex_map_sat
```

[79]:
```
<folium.folium.Map at 0x18ea0944230>
```

Finalmente con la función **save()** podemos guardar el archivo en formato html para ser integrado a un sitio web o aplicación.

[80]:
```
#mex_map_sat.save('mapa_aduanas.html')
```