Cargar Datos y unirlos en un sólo dataframe

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
from scipy.stats import chi2_contingency
from scipy.stats import contingency

pd.__version__
```

```
'1.5.3'
```

```
df1= pd.read_csv('https://raw.githubusercontent.com/kevingonzal/cienciaDatos/main/file.csv')
df2= pd.read_csv('https://raw.githubusercontent.com/kevingonzal/cienciaDatos/main/file2.csv')
```

```
df1
```

	PLANTAORIGEN	FECHADESPACHO	C_ID_VIAJE	FECHAVIAJE	C_CLIENTE	C_CLIENTE.1	ID_
0	Pesqueria	01/10/2022 01:36:16	46227308	30/09/2022 23:59:00	N000003805	CLIENTE27	

df2

	PLANTAORIGEN	FECHADESPACHO	C_ID_VIAJE	FECHAVIAJE	C_CLIENTE	D_CLIENTE	ID_E
0	Guerrero	11/06/2022 23:19:19	47103405	05-11-2022 15:16:00	NaN	CLIENTE58	
1	Guerrero	11/16/2022 14:58:16	47322566	19-11-2022 10:27:00	NaN	CLIENTE132	
2	Guerrero	11/18/2022 07:14:21	47358720	18-11-2022 00:20:00	NaN	CLIENTE58	
3	Guerrero	11/19/2022 07:46:41	47369172	18-11-2022 12:18:00	NaN	CLIENTE58	
4	Guerrero	11/23/2022 15:13:51	47480917	23-11-2022 23:59:00	H000232500	CLIENTE171	
...	...	...	...	...	...	...	
9149	Guerrero	11/29/2022 14:34:37	47610758	06-12-2022 23:59:00	H000756702	CLIENTE6	
9150	Guerrero	11/29/2022 08:11:46	47599580	28-11-2022 23:59:00	N000100732	CLIENTE219	
9151	Guerrero	11/29/2022 08:11:46	47599580	28-11-2022 23:59:00	N000100723	CLIENTE219	
9152	Guerrero	11/29/2022 08:11:46	47599580	28-11-2022 23:59:00	N000100723	CLIENTE219	
9153	Guerrero	11/23/2022 16:37:44	47468966	25-11-2022 01:16:00	NaN	CLIENTE38	

9154 rows × 34 columns

```
df1.columns
```

```
Index(['PLANTAORIGEN', 'FECHADESPACHO', 'C_ID_VIAJE', 'FECHAVIAJE',
       'C_CLIENTE', 'C_CLIENTE.1', 'ID_ESTADO', 'ESTADO',
       'D_EMPRESA_TRANSPORTISTA', 'C_ID_CONDUCTOR', 'NOM_APE_COND',
       'TIPOCAMION', 'TIPOTRANSPORTE', 'TIPO_PRODUCTO', 'TIPO_FORMA',
       'Q_CANTIDAD', 'F_LLEGADANAVE', 'F_EGRESONAVE', 'F_PRESENTACION',
       'F_INGRESOPLANTA', 'F_PESAJEENTRADA', 'F_ASIGVIAJE',
       'F_InicioCargaBulto', 'F_FINCARGA', 'F_PESAJESALIDA', 'TIPO_ASIGNACION',
       'TIPO_SERVICIO', 'C_PLANTA', 'D_PLANTA', 'TIPO_PERMISO', 'PESO_NETO',
```

```
       'CAP_MAXIMA', 'ZONA_DESTINO', 'CUIDAD'],
      dtype='object')
```

```
df2.columns
```

```
Index(['PLANTAORIGEN', 'FECHADESPACHO', 'C_ID_VIAJE', 'FECHAVIAJE',
       'C_CLIENTE', 'D_CLIENTE', 'ID_ESTADO', 'ESTADO',
       'D_EMPRESA_TRANSPORTISTA', 'C_ID_CONDUCTOR', 'NOM_APE_COND',
       'TIPO_SERVICIO', 'TIPOCAMION', 'TIPOTRANSPORTE', 'TIPO_PRODUCTO',
       'TIPO_FORMA', 'CANT_PROGRAMADA', 'F_PRESENTACION', 'F_LLEGADANAVE',
       'F_EGRESONAVE', 'F_INGRESOPLANTA', 'F_PESAJEENTRADA', 'F_ASIGVIAJE',
       'F_INICIOCARGABULTO', 'F_FINCARGA', 'F_PESAJESALIDA', 'TIPO_ASIGNACION',
       'C_PLANTA', 'D_PLANTA', 'TIPO_PERMISO', 'PESO_NETO', 'CAP_MAXIMA',
       'ZONA_DESTINO', 'CUIDAD'],
      dtype='object')
```

```
df1 = df1.rename(columns={'C_CLIENTE.1': 'D_CLIENTE','Q_CANTIDAD': 'CANT_PROGRAMADA','F_Inici
```

```
df1 = df1[['PLANTAORIGEN', 'FECHADESPACHO', 'C_ID_VIAJE', 'FECHAVIAJE',
       'C_CLIENTE', 'D_CLIENTE', 'ID_ESTADO', 'ESTADO',
       'D_EMPRESA_TRANSPORTISTA', 'C_ID_CONDUCTOR', 'NOM_APE_COND',
       'TIPO_SERVICIO', 'TIPOCAMION', 'TIPOTRANSPORTE', 'TIPO_PRODUCTO',
       'TIPO_FORMA', 'CANT_PROGRAMADA', 'F_PRESENTACION', 'F_LLEGADANAVE',
       'F_EGRESONAVE', 'F_INGRESOPLANTA', 'F_PESAJEENTRADA', 'F_ASIGVIAJE',
       'F_INICIOCARGABULTO', 'F_FINCARGA', 'F_PESAJESALIDA', 'TIPO_ASIGNACION',
       'C_PLANTA', 'D_PLANTA', 'TIPO_PERMISO', 'PESO_NETO', 'CAP_MAXIMA',
       'ZONA_DESTINO', 'CUIDAD']]
```

```
df1.columns
```

```
Index(['PLANTAORIGEN', 'FECHADESPACHO', 'C_ID_VIAJE', 'FECHAVIAJE',
       'C_CLIENTE', 'D_CLIENTE', 'ID_ESTADO', 'ESTADO',
       'D_EMPRESA_TRANSPORTISTA', 'C_ID_CONDUCTOR', 'NOM_APE_COND',
       'TIPO_SERVICIO', 'TIPOCAMION', 'TIPOTRANSPORTE', 'TIPO_PRODUCTO',
       'TIPO_FORMA', 'CANT_PROGRAMADA', 'F_PRESENTACION', 'F_LLEGADANAVE',
       'F_EGRESONAVE', 'F_INGRESOPLANTA', 'F_PESAJEENTRADA', 'F_ASIGVIAJE',
       'F_INICIOCARGABULTO', 'F_FINCARGA', 'F_PESAJESALIDA', 'TIPO_ASIGNACION',
       'C_PLANTA', 'D_PLANTA', 'TIPO_PERMISO', 'PESO_NETO', 'CAP_MAXIMA',
       'ZONA_DESTINO', 'CUIDAD'],
      dtype='object')
```

```
df=  pd.concat([df1,df2])
```

Limpieza de dataframe

```
df
```

	PLANTAORIGEN	FECHADESPACHO	C_ID_VIAJE	FECHAVIAJE	C_CLIENTE	D_CLIENTE	ID_E
0	Pesqueria	01/10/2022 01:36:16	46227308	30/09/2022 23:59:00	N000003805	CLIENTE27	
1	Pesqueria	01/10/2022 05:38:00	46248480	01/10/2022 14:13:00	H000030812	CLIENTE198	
2	Pesqueria	01/10/2022 04:04:55	46260526	01/10/2022 23:59:00	NaN	CLIENTE231	
3	Pesqueria	01/10/2022 04:59:28	46252417	30/09/2022 17:44:00	H000269818	CLIENTE232	
4	Pesqueria	01/10/2022 01:31:28	46246049	01/10/2022 11:47:00	N000100336	CLIENTE233	
...	...	...	...	...	...	...	
9149	Guerrero	11/29/2022 14:34:37	47610758	06-12-2022 23:59:00	H000756702	CLIENTE6	
9150	Guerrero	11/29/2022 08:11:46	47599580	28-11-2022 23:59:00	N000100732	CLIENTE219	
9151	Guerrero	11/29/2022 08:11:46	47599580	28-11-2022 23:59:00	N000100723	CLIENTE219	
9152	Guerrero	11/29/2022 08:11:46	47599580	28-11-2022 23:59:00	N000100723	CLIENTE219	
9153	Guerrero	11/23/2022 16:37:44	47468966	25-11-2022 01:16:00	NaN	CLIENTE38	

14440 rows × 34 columns

```
columnas_interes = df.columns
tablas_frecuencias = []

for columna in columnas_interes:
    tabla_frecuencia = df[columna].value_counts().reset_index()
    tabla_frecuencia.columns = ['Valor', f'Frecuencia_{columna}']
    tablas_frecuencias.append(tabla_frecuencia)


tablas_frecuencias
```

```
[       Valor  Frecuencia_PLANTAORIGEN
 0   Guerrero                     9154
 1  Pesqueria                     5286,
                     Valor  Frecuencia_FECHADESPACHO
 0      10/28/2022 18:45:43                        31
 1      10/07/2022 20:16:57                        24
 2      10/08/2022 14:34:29                        24
 3      10/24/2022 04:07:20                        23
```

```
4      11/12/2022 00:31:36                        22
...                    ...                       ...
12777  16/10/2022 03:24:14                         1
12778  16/10/2022 07:01:44                         1
12779  16/10/2022 02:16:45                         1
12780  16/10/2022 05:01:09                         1
12781  11/23/2022 16:37:44                         1

[12782 rows x 2 columns],
          Valor  Frecuencia_C_ID_VIAJE
0      46927909                     31
1      46461286                     24
2      46440893                     24
3      46810998                     23
4      47231837                     22
...         ...                    ...
12792  46640559                      1
12793  46643920                      1
12794  46639639                      1
12795  46646802                      1
12796  47468966                      1

[12797 rows x 2 columns],
                    Valor  Frecuencia_FECHAVIAJE
0     13/10/2022 23:59:00                    217
1     16-11-2022 23:59:00                    204
2     14/10/2022 23:59:00                    183
3     06/10/2022 23:59:00                    162
4     08/10/2022 23:59:00                    160
...                   ...                    ...
4140  28-11-2022 16:25:00                      1
4141  29-11-2022 07:25:00                      1
4142  10/10/2022 17:05:00                      1
4143  03-11-2022 00:30:00                      1
4144  25-11-2022 01:16:00                      1

[4145 rows x 2 columns],
          Valor  Frecuencia_C_CLIENTE
0    H000269647                   783
1    H000126500                   596
2    H010005034                   582
3    H000122204                   459
4    H000041807                   421
..          ...                   ...
227  N000191639                     1
228  N000001967                     1
229  N000000687                     1
230  N000003863                     1
231  H000148817                     1
```

```
df = df.rename(columns={'CUIDAD': 'CIUDAD'})
```

```
df['ESTADO'] = df['ESTADO'].str.upper()
df['CIUDAD'] = df['CIUDAD'].str.upper()


reemplazos_Estados = {'SAN LUIS POTOSO': 'SAN LUIS POTOSI', 'ESTADO DE MIXICO': 'ESTADO DE ME
            'CIUDAD DE MOXICO': 'CIUDAD DE MEXICO', 'CIUDAD DE MIXICO': 'CIUDAD DE MEXICO',
            'QUEROTARO': 'QUERETARO', 'YUCATON': 'YUCATAN', 'MICHOACON': 'MICHOACAN',}


df = df.replace(reemplazos_Estados)


reemplazos_TipoTransporte = {'Plataforma 3 ejes Neumotica': 'Plataforma 3 ejes Neumatica', 'P
            'Plataforma 3 Ejes Neumotica Corta': 'Plataforma 3 ejes Neumatica Corta', 'Plata
            'Plataforma 3 ejes Neumotica Cortina': 'Plataforma 3 ejes Neumatica Cortina', 'P
                        'Plataforma 3 Ejes Neumotica Ultra Ligera': 'Plataforma 3 ejes N
                       'Plataforma 3 Ejes Neumitica Corta Ligera': 'Plataforma 3 ejes Ne
                       'Plataforma 3 ejes Neumotica Portacintas': 'Plataforma 3 ejes Neu


df = df.replace(reemplazos_TipoTransporte)


reemplazos_Ciudad = {'S NICOLAS DE LOS GZA': 'SAN NICOLAS DE LOS G', 'SAN NICOLAS DE LOS GARZ
            'GRAL. ESCOBEDO': 'GRAL ESCOBEDO', 'ESCOBEDO': 'GRAL ESCOBEDO', 'ESCOBEDO , NUEV
            'CIUDAD FRONTERA': 'FRONTERA', 'APODACA, N. L.': 'APODACA', 'APODACA, N.L.': 'AP
                        'SAN JOSE ITURBIDE': 'SAN JOSE DE ITURBIDE', 'CASTAIOS': 'CASTAN
                       'CIUDAD JUAREZ': 'CD JUAREZ', 'TEPEJI DEL RIO': 'TEPEJI DEL RIO D
                       'MORIDA': 'MERIDA', 'CUAUTITLAN': 'CUAUTITLAN IZCALLI', 'SALTILLO


df = df.replace(reemplazos_Ciudad)


df_filtrado1 = df.query('ESTADO == "NUEVO LEON" & TIPO_SERVICIO == "FO"')


print(df_filtrado1)

     Empty DataFrame
     Columns: [PLANTAORIGEN, FECHADESPACHO, C_ID_VIAJE, FECHAVIAJE, C_CLIENTE, D_CLIENTE, ID_
     Index: []

     [0 rows x 34 columns]


df_filtrado2 = df.query('ESTADO != "NUEVO LEON" & TIPO_SERVICIO == "LO"')


print(df_filtrado2)

          PLANTAORIGEN        FECHADESPACHO  C_ID_VIAJE           FECHAVIAJE  \
     6         Guerrero  11/28/2022 02:35:46    47578889  30-11-2022 21:00:00
     19        Guerrero  04-10-2022 20:11:50    46360434  04-10-2022 17:37:00
```

```
20       Guerrero  04-10-2022 20:11:50    46360434  04-10-2022 17:37:00
179      Guerrero  11/25/2022 00:38:10    47504715  24-11-2022 12:35:00
180      Guerrero  11/29/2022 04:26:52    47597900  28-11-2022 23:59:00
...           ...                  ...         ...                  ...
9112     Guerrero  11/17/2022 20:13:16    47325163  16-11-2022 23:59:00
9113     Guerrero  11/18/2022 18:10:08    47359390  18-11-2022 23:59:00
9114     Guerrero  11/18/2022 18:10:08    47359390  18-11-2022 23:59:00
9115     Guerrero  11/19/2022 17:20:48    47319752  16-11-2022 23:59:00
9117     Guerrero  11/28/2022 15:07:53    47578888  30-11-2022 21:00:00

       C_CLIENTE   D_CLIENTE  ID_ESTADO            ESTADO  \
6            NaN   CLIENTE41         24   SAN LUIS POTOSI
19     N000100516  CLIENTE110          5          COAHUILA
20     N000100516  CLIENTE110          5          COAHUILA
179    H000833200   CLIENTE13         24   SAN LUIS POTOSI
180    H000126300  CLIENTE174         24   SAN LUIS POTOSI
...           ...         ...        ...               ...
9112   H000148800  CLIENTE223          9  CIUDAD DE MEXICO
9113   N000001678  CLIENTE160          1    AGUASCALIENTES
9114   N000001678  CLIENTE160          1    AGUASCALIENTES
9115   N000002693  CLIENTE184         30          VERACRUZ
9117          NaN   CLIENTE41         24   SAN LUIS POTOSI

     D_EMPRESA_TRANSPORTISTA  C_ID_CONDUCTOR  ...           F_FINCARGA  \
6                TRANPORTE12          106040  ...  28-11-2022 01:36:57
19               TRANPORTE47          110231  ...  04-10-2022 19:15:52
20               TRANPORTE47          110231  ...  04-10-2022 19:15:52
179              TRANPORTE50          109845  ...  25-11-2022 00:22:58
180              TRANPORTE50          109845  ...  29-11-2022 03:53:31
...                      ...             ...  ...                  ...
9112             TRANPORTE61          109770  ...  17-11-2022 19:40:01
9113             TRANPORTE61          109770  ...  18-11-2022 17:46:13
9114             TRANPORTE61          109770  ...  18-11-2022 17:46:13
9115             TRANPORTE61          109770  ...  19-11-2022 17:08:23
9117             TRANPORTE61          109770  ...  28-11-2022 14:49:44

          F_PESAJESALIDA TIPO_ASIGNACION C_PLANTA  D_PLANTA  \
6     28-11-2022 02:35:46             NaN      GUE  Guerrero
19    04-10-2022 20:11:50          MANUAL      GUE  Guerrero
20    04-10-2022 20:11:50          MANUAL      GUE  Guerrero
179   25-11-2022 00:38:10          MANUAL      GUE  Guerrero
180   29-11-2022 04:26:52             NaN      GUE  Guerrero
...                   ...             ...      ...       ...
9112  17-11-2022 20:13:16          MANUAL      GUE  Guerrero
9113  18-11-2022 18:10:08          MANUAL      GUE  Guerrero
9114  18-11-2022 18:10:08          MANUAL      GUE  Guerrero
9115  19-11-2022 17:20:48          MANUAL      GUE  Guerrero
9117  28-11-2022 15:07:53          MANUAL      GUE  Guerrero

          TIPO_PERMISO  PESO_NETO CAP_MAXIMA ZONA_DESTINO           CIUDAD
6     Traslado Externo      36830       54.0            0  SAN LUIS POTOSI
19       Cliente ENVIA      12720       54.0            0     RAMOS ARIZPE
20       Cliente ENVIA      12720       54.0            0     RAMOS ARIZPE
179      Cliente ENVIA      19860       54.0            0  SAN LUIS POTOSI
180      Cliente ENVIA      18480       54.0            0  SAN LUIS POTOSI
```

```
df.loc[(df['ESTADO'] != "NUEVO LEON") & (df['TIPO_SERVICIO'] == 'LO'), 'TIPO_SERVICIO'] = "FO
```

```
df_filtrado2 = df.query('ESTADO != "NUEVO LEON" & TIPO_SERVICIO == "LO"')
```

```
print(df_filtrado2)
```

```
Empty DataFrame
Columns: [PLANTAORIGEN, FECHADESPACHO, C_ID_VIAJE, FECHAVIAJE, C_CLIENTE, D_CLIENTE, ID_
Index: []

[0 rows x 34 columns]
```

```
df_dirty = df
```

```
df_dirty.dtypes
```

```
PLANTAORIGEN               object
FECHADESPACHO              object
C_ID_VIAJE                  int64
FECHAVIAJE                 object
C_CLIENTE                  object
D_CLIENTE                  object
ID_ESTADO                   int64
ESTADO                     object
D_EMPRESA_TRANSPORTISTA    object
C_ID_CONDUCTOR              int64
NOM_APE_COND               object
TIPO_SERVICIO              object
TIPOCAMION                 object
TIPOTRANSPORTE             object
TIPO_PRODUCTO              object
TIPO_FORMA                 object
CANT_PROGRAMADA           float64
F_PRESENTACION             object
F_LLEGADANAVE              object
F_EGRESONAVE               object
F_INGRESOPLANTA            object
F_PESAJEENTRADA            object
F_ASIGVIAJE                object
F_INICIOCARGABULTO         object
F_FINCARGA                 object
F_PESAJESALIDA             object
TIPO_ASIGNACION            object
C_PLANTA                   object
D_PLANTA                   object
TIPO_PERMISO               object
PESO_NETO                   int64
CAP_MAXIMA                float64
ZONA_DESTINO               object
CIUDAD                     object
dtype: object
```

```
#El porcentaje de nulos por columnas
total_of_all = df_dirty.isnull().sum()
percent_of_all = (df_dirty.isnull().sum()/df_dirty.isnull().count())
missing_data_test = pd.concat([total_of_all, percent_of_all*100], axis=1, keys=['Total Nulos'
missing_data_test
```

	Total Nulos	Percent
PLANTAORIGEN	0	0.000000
FECHADESPACHO	0	0.000000
C_ID_VIAJE	0	0.000000
FECHAVIAJE	0	0.000000
C_CLIENTE	7538	52.202216
D_CLIENTE	18	0.124654
ID_ESTADO	0	0.000000
ESTADO	0	0.000000

```
# obtener el número de puntos de datos faltantes por columna
missing_values_count = df_dirty.isnull().sum()
```

```
# mira el número de puntos que faltan
missing_values_count[0:]
```

```
PLANTAORIGEN                0
FECHADESPACHO               0
C_ID_VIAJE                  0
FECHAVIAJE                  0
C_CLIENTE                7538
D_CLIENTE                  18
ID_ESTADO                   0
ESTADO                      0
D_EMPRESA_TRANSPORTISTA     0
C_ID_CONDUCTOR              0
NOM_APE_COND                1
TIPO_SERVICIO               0
TIPOCAMION                  0
TIPOTRANSPORTE              0
TIPO_PRODUCTO               0
TIPO_FORMA                  0
CANT_PROGRAMADA             0
F_PRESENTACION              0
F_LLEGADANAVE               0
F_EGRESONAVE               55
F_INGRESOPLANTA             1
F_PESAJEENTRADA           627
F_ASIGVIAJE                12
F_INICIOCARGABULTO          0
F_FINCARGA                  0
F_PESAJESALIDA            634
TIPO_ASIGNACION          6631
C_PLANTA                    0
D_PLANTA                    0
TIPO_PERMISO                0
PESO_NETO                   0
CAP_MAXIMA                  0
```

```
ZONA_DESTINO                0
CIUDAD                      0
dtype: int64
```

```
total_cells = np.product(df_dirty.shape)
total_missing = missing_values_count.sum()


# por ciento de datos que faltan
percentage_missing_values = (total_missing/total_cells) * 100
print("Hay un porcentaje total de:",percentage_missing_values,"% de valores faltantes")
```

```
Hay un porcentaje total de: 3.160542610395959 % de valores faltantes
```

```
df_dirty
```

	PLANTAORIGEN	FECHADESPACHO	C_ID_VIAJE	FECHAVIAJE	C_CLIENTE	D_CLIENTE	ID_E
0	Pesqueria	01/10/2022 01:36:16	46227308	30/09/2022 23:59:00	N000003805	CLIENTE27	
1	Pesqueria	01/10/2022 05:38:00	46248480	01/10/2022 14:13:00	H000030812	CLIENTE198	
2	Pesqueria	01/10/2022 04:04:55	46260526	01/10/2022 23:59:00	NaN	CLIENTE231	
3	Pesqueria	01/10/2022 04:59:28	46252417	30/09/2022 17:44:00	H000269818	CLIENTE232	
4	Pesqueria	01/10/2022 01:31:28	46246049	01/10/2022 11:47:00	N000100336	CLIENTE233	
...	...	...	...	...	...	...	
9149	Guerrero	11/29/2022 14:34:37	47610758	06-12-2022 23:59:00	H000756702	CLIENTE6	
9150	Guerrero	11/29/2022 08:11:46	47599580	28-11-2022 23:59:00	N000100732	CLIENTE219	
9151	Guerrero	11/29/2022 08:11:46	47599580	28-11-2022 23:59:00	N000100723	CLIENTE219	
9152	Guerrero	11/29/2022 08:11:46	47599580	28-11-2022 23:59:00	N000100723	CLIENTE219	
9153	Guerrero	11/23/2022 16:37:44	47468966	25-11-2022 01:16:00	NaN	CLIENTE38	

14440 rows × 34 columns

```
df_dirty.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 14440 entries, 0 to 9153
Data columns (total 34 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   PLANTAORIGEN             14440 non-null  object 
 1   FECHADESPACHO            14440 non-null  object 
 2   C_ID_VIAJE               14440 non-null  int64  
 3   FECHAVIAJE               14440 non-null  object 
 4   C_CLIENTE                6902 non-null   object 
 5   D_CLIENTE                14422 non-null  object 
 6   ID_ESTADO                14440 non-null  int64  
 7   ESTADO                   14440 non-null  object 
 8   D_EMPRESA_TRANSPORTISTA  14440 non-null  object 
 9   C_ID_CONDUCTOR           14440 non-null  int64  
 10  NOM_APE_COND             14439 non-null  object 
 11  TIPO_SERVICIO            14440 non-null  object 
 12  TIPOCAMION               14440 non-null  object 
 13  TIPOTRANSPORTE           14440 non-null  object 
 14  TIPO_PRODUCTO            14440 non-null  object 
 15  TIPO_FORMA               14440 non-null  object 
 16  CANT_PROGRAMADA          14440 non-null  float64
 17  F_PRESENTACION           14440 non-null  object 
 18  F_LLEGADANAVE            14440 non-null  object 
 19  F_EGRESONAVE             14385 non-null  object 
 20  F_INGRESOPLANTA          14439 non-null  object 
 21  F_PESAJEENTRADA          13813 non-null  object 
 22  F_ASIGVIAJE              14428 non-null  object 
 23  F_INICIOCARGABULTO       14440 non-null  object 
 24  F_FINCARGA               14440 non-null  object 
 25  F_PESAJESALIDA           13806 non-null  object 
 26  TIPO_ASIGNACION          7809 non-null   object 
 27  C_PLANTA                 14440 non-null  object 
 28  D_PLANTA                 14440 non-null  object 
 29  TIPO_PERMISO             14440 non-null  object 
 30  PESO_NETO                14440 non-null  int64  
 31  CAP_MAXIMA               14440 non-null  float64
 32  ZONA_DESTINO             14440 non-null  object 
 33  CIUDAD                   14440 non-null  object 
dtypes: float64(2), int64(4), object(28)
memory usage: 3.9+ MB
```

```
df_dirty.duplicated()
```

```
0       False
1       False
2       False
3       False
4       False
        ...
9149    False
9150    False
9151    False
```

```
9152     True
9153    False
Length: 14440, dtype: bool
```

```
#Eliminación de los registros duplicados parciales
#Fechas
df_dirty_no_duplicates = df_dirty.drop_duplicates(subset=df_dirty.columns[[1,2,12,13,14,15,16
df_dirty_no_duplicates
```

	PLANTAORIGEN	FECHADESPACHO	C_ID_VIAJE	FECHAVIAJE	C_CLIENTE	D_CLIENTE	ID_E
0	Pesqueria	01/10/2022 01:36:16	46227308	30/09/2022 23:59:00	N000003805	CLIENTE27	
1	Pesqueria	01/10/2022 05:38:00	46248480	01/10/2022 14:13:00	H000030812	CLIENTE198	
2	Pesqueria	01/10/2022 04:04:55	46260526	01/10/2022 23:59:00	NaN	CLIENTE231	
3	Pesqueria	01/10/2022 04:59:28	46252417	30/09/2022 17:44:00	H000269818	CLIENTE232	
4	Pesqueria	01/10/2022 01:31:28	46246049	01/10/2022 11:47:00	N000100336	CLIENTE233	
...	...	...	...	...	...	...	
9147	Guerrero	11/28/2022 04:55:19	47581582	28-11-2022 00:05:00	H000041807	CLIENTE93	
9148	Guerrero	11/29/2022 02:28:35	47593553	28-11-2022 23:59:00	NaN	CLIENTE58	
9149	Guerrero	11/29/2022 14:34:37	47610758	06-12-2022 23:59:00	H000756702	CLIENTE6	
9150	Guerrero	11/29/2022 08:11:46	47599580	28-11-2022 23:59:00	N000100732	CLIENTE219	
9153	Guerrero	11/23/2022 16:37:44	47468966	25-11-2022 01:16:00	NaN	CLIENTE38	

12797 rows × 34 columns

```
#Corregir valores erroneos
df_dirty_no_duplicates.isnull().sum()
```

```
PLANTAORIGEN                0
FECHADESPACHO               0
C_ID_VIAJE                  0
FECHAVIAJE                  0
C_CLIENTE                7495
```

```
D_CLIENTE                    15
ID_ESTADO                     0
ESTADO                        0
D_EMPRESA_TRANSPORTISTA       0
C_ID_CONDUCTOR                0
NOM_APE_COND                  1
TIPO_SERVICIO                 0
TIPOCAMION                    0
TIPOTRANSPORTE                0
TIPO_PRODUCTO                 0
TIPO_FORMA                    0
CANT_PROGRAMADA               0
F_PRESENTACION                0
F_LLEGADANAVE                 0
F_EGRESONAVE                 52
F_INGRESOPLANTA               1
F_PESAJEENTRADA             627
F_ASIGVIAJE                  11
F_INICIOCARGABULTO            0
F_FINCARGA                    0
F_PESAJESALIDA              634
TIPO_ASIGNACION            6498
C_PLANTA                      0
D_PLANTA                      0
TIPO_PERMISO                  0
PESO_NETO                     0
CAP_MAXIMA                    0
ZONA_DESTINO                  0
CIUDAD                        0
dtype: int64
```

```
#Corregimos la variable usando replace
#Columna Tipo
df_coerce = df_dirty_no_duplicates.copy()


df_coerce['NOM_APE_COND']=df_coerce['NOM_APE_COND'].replace('','NaN')

#Columna Operacion
df_coerce['F_EGRESONAVE']=df_coerce['F_EGRESONAVE'].replace(' ','NaN')

df_coerce['F_INGRESOPLANTA']=df_coerce['F_INGRESOPLANTA'].replace(' ','NaN')

#Columna Precio_venta
df_coerce['F_PESAJEENTRADA']=df_coerce['F_PESAJEENTRADA'].replace(' ','NaN')

#Columna Edad_vendedor
#Columna Super
df_coerce['F_ASIGVIAJE']=df_coerce['F_ASIGVIAJE'].replace('','NaN')

#Columna Precio_venta
df_coerce['F_PESAJESALIDA']=df_coerce['F_PESAJESALIDA'].replace(' ','NaN')
```

```
df_coerce
```

	PLANTAORIGEN	FECHADESPACHO	C_ID_VIAJE	FECHAVIAJE	C_CLIENTE	D_CLIENTE	ID_E
0	Pesqueria	01/10/2022 01:36:16	46227308	30/09/2022 23:59:00	N000003805	CLIENTE27	
1	Pesqueria	01/10/2022 05:38:00	46248480	01/10/2022 14:13:00	H000030812	CLIENTE198	
2	Pesqueria	01/10/2022 04:04:55	46260526	01/10/2022 23:59:00	NaN	CLIENTE231	
3	Pesqueria	01/10/2022 04:59:28	46252417	30/09/2022 17:44:00	H000269818	CLIENTE232	
4	Pesqueria	01/10/2022 01:31:28	46246049	01/10/2022 11:47:00	N000100336	CLIENTE233	
...	...	...	...	...	...	...	
9147	Guerrero	11/28/2022 04:55:19	47581582	28-11-2022 00:05:00	H000041807	CLIENTE93	
9148	Guerrero	11/29/2022 02:28:35	47593553	28-11-2022 23:59:00	NaN	CLIENTE58	
9149	Guerrero	11/29/2022 14:34:37	47610758	06-12-2022 23:59:00	H000756702	CLIENTE6	
9150	Guerrero	11/29/2022 08:11:46	47599580	28-11-2022 23:59:00	N000100732	CLIENTE219	
9153	Guerrero	11/23/2022 16:37:44	47468966	25-11-2022 01:16:00	NaN	CLIENTE38	

12797 rows × 34 columns

```
#Eliminamos los valores NaNs de la tabla
df_clean = df_coerce.dropna(axis=0)
df_clean
```

	PLANTAORIGEN	FECHADESPACHO	C_ID_VIAJE	FECHAVIAJE	C_CLIENTE	D_CLIENTE	ID_E
0	Pesqueria	01/10/2022 01:36:16	46227308	30/09/2022 23:59:00	N000003805	CLIENTE27	
1	Pesqueria	01/10/2022 05:38:00	46248480	01/10/2022 14:13:00	H000030812	CLIENTE198	
3	Pesqueria	01/10/2022 04:59:28	46252417	30/09/2022 17:44:00	H000269818	CLIENTE232	
4	Pesqueria	01/10/2022 01:31:28	46246049	01/10/2022 11:47:00	N000100336	CLIENTE233	
5	Pesqueria	01/10/2022 01:42:42	46252415	01/10/2022 17:44:00	N000100266	CLIENTE190	
...	...	...	...	...	...	...	
9125	Guerrero	11/05/2022 01:37:45	47082392	05-11-2022 23:59:00	N000003215	CLIENTE139	
9136	Guerrero	11/16/2022 13:59:59	47307962	15-11-2022 15:25:00	H000126500	CLIENTE177	

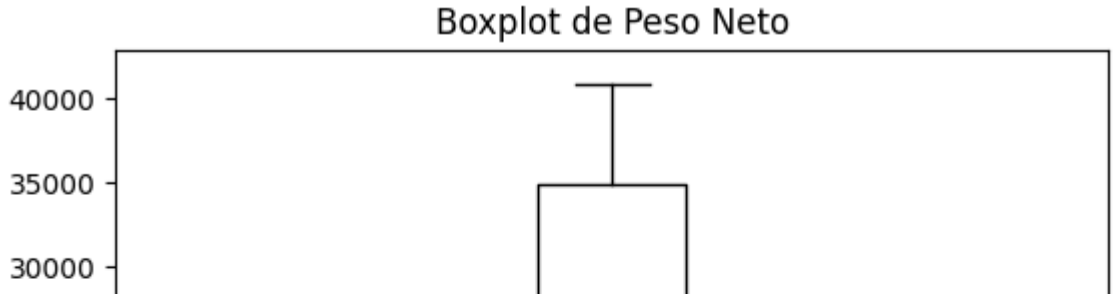
```
#Identificamos vacíos usando isna
df_clean.isna().any(0)
```

```
<ipython-input-44-90131866f2af>:2: FutureWarning: In a future version of pandas all argu
  df_clean.isna().any(0)
PLANTAORIGEN               False
FECHADESPACHO              False
C_ID_VIAJE                 False
FECHAVIAJE                 False
C_CLIENTE                  False
D_CLIENTE                  False
ID_ESTADO                  False
ESTADO                     False
D_EMPRESA_TRANSPORTISTA    False
C_ID_CONDUCTOR             False
NOM_APE_COND               False
TIPO_SERVICIO              False
TIPOCAMION                 False
TIPOTRANSPORTE             False
TIPO_PRODUCTO              False
TIPO_FORMA                 False
CANT_PROGRAMADA            False
F_PRESENTACION             False
F_LLEGADANAVE              False
F_EGRESONAVE               False
F_INGRESOPLANTA            False
F_PESAJEENTRADA            False
F_ASIGVIAJE                False
F_INICIOCARGABULTO         False
F_FINCARGA                 False
F_PESAJESALIDA             False
```

```
TIPO_ASIGNACION           False
C_PLANTA                  False
D_PLANTA                  False
TIPO_PERMISO              False
PESO_NETO                 False
CAP_MAXIMA                False
ZONA_DESTINO              False
CIUDAD                    False
dtype: bool
```

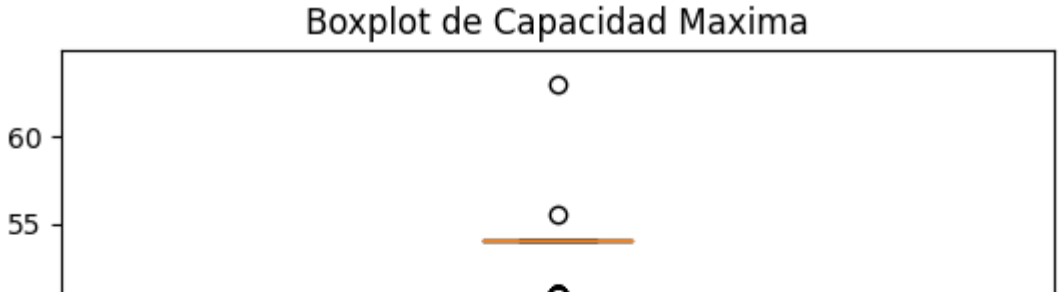
```
# Calcular la media y la desviación estándar
mean = np.mean(df_clean['PESO_NETO'])
std_dev = np.std(df_clean['PESO_NETO'])

# Definir un umbral para identificar outliers
threshold = 3

# Identificar los valores outliers
outliers = []
for value in df_clean['PESO_NETO']:
    z_score = (value - mean) / (std_dev)
    if np.abs(z_score) > threshold:
        outliers.append(value)

# Imprimir los valores outliers
print("Valores outliers encontrados:", outliers)
# Crear el gráfico de boxplot
plt.boxplot(df_clean['PESO_NETO'])
plt.title('Boxplot de Peso Neto')
plt.xlabel('Peso Neto')
plt.ylabel('Unidades')
plt.scatter(x=[1]*len(outliers), y=outliers, c='red', marker='o')
plt.show()
```

```
Valores outliers encontrados: []
```




```
# Calcular la media y la desviación estándar
mean = np.mean(df_clean['CAP_MAXIMA'])
std_dev = np.std(df_clean['CAP_MAXIMA'])

# Definir un umbral para identificar outliers
threshold = 3

# Identificar los valores outliers
outliers = []
for value in df_clean['CAP_MAXIMA']:
    z_score = (value - mean) / (std_dev)
    if np.abs(z_score) > threshold:
        outliers.append(value)

# Imprimir los valores outliers
print("Valores outliers encontrados:", outliers)
# Crear el gráfico de boxplot
plt.boxplot(df_clean['CAP_MAXIMA'])
plt.title('Boxplot de Capacidad Maxima')
plt.xlabel('Capacidad Maxima')
plt.ylabel('Unidades')
plt.scatter(x=[1]*len(outliers), y=outliers, c='red', marker='o')
plt.show()
```

```
Valores outliers encontrados: [24.5, 24.5, 24.5, 24.5, 24.5, 24.5, 24.5, 24.5, 24.5, 24
```



Cálculo de estadísticas variables cualitativas y cuantitativas

```
print("\nCAP_MAXIMA:")
print("Media:",df['CAP_MAXIMA'].mean())
print("Mediana:",df['CAP_MAXIMA'].median())
print("Máximo:",df['CAP_MAXIMA'].max())
print("Mínimo:",df['CAP_MAXIMA'].min())
print("Varianza:",df['CAP_MAXIMA'].var())
print("Desviación Estándar:",df['CAP_MAXIMA'].std())
print("Moda:",df['CAP_MAXIMA'].mode())
```

```
CAP_MAXIMA:
Media: 53.244563711911354
Mediana: 54.0
Máximo: 63.0
Mínimo: 24.5
Varianza: 14.792719250043978
Desviación Estándar: 3.846130425511332
Moda: 0    54.0
Name: CAP_MAXIMA, dtype: float64
```

```
print("\nPESO_NETO:")
print("Media:",df['PESO_NETO'].mean())
print("Mediana:",df['PESO_NETO'].median())
print("Máximo:",df['PESO_NETO'].max())
print("Mínimo:",df['PESO_NETO'].min())
print("Varianza:",df['PESO_NETO'].var())
print("Desviación Estándar:",df['PESO_NETO'].std())
print("Moda:",df['PESO_NETO'].mode())
```

```
PESO_NETO:
Media: 27319.924515235456
Mediana: 28840.0
Máximo: 41200
Mínimo: 0
Varianza: 91847479.07872579
Desviación Estándar: 9583.709046017924
```

```
Moda: 0    0
Name: PESO_NETO, dtype: int64
```

```
print("CANT_PROGRAMADA:")
print("Media:",df['CANT_PROGRAMADA'].mean())
print("Mediana:",df['CANT_PROGRAMADA'].median())
print("Máximo:",df['CANT_PROGRAMADA'].max())
print("Mínimo:",df['CANT_PROGRAMADA'].min())
print("Varianza:",df['CANT_PROGRAMADA'].var())
print("Desviación Estándar:",df['CANT_PROGRAMADA'].std())
print("Moda:",df['CANT_PROGRAMADA'].mode())
```

```
CANT_PROGRAMADA:
Media: 28.70895768698061
Mediana: 29.7635
Máximo: 41.446
Mínimo: 0.818
Varianza: 60.49343800284969
Desviación Estándar: 7.77775276046042
Moda: 0    25.929
Name: CANT_PROGRAMADA, dtype: float64
```

```
print('La moda de "TIPO_FORMA" es:', df['TIPO_FORMA'].mode()[0])
print('La moda de "PLANTAORIGEN" es:', df['PLANTAORIGEN'].mode()[0])
print('La moda de "FECHAVIAJE" es:', df['FECHAVIAJE'].mode()[0])
print('La moda de "C_CLIENTE" es:', df['C_CLIENTE'].mode()[0])
print('La moda de "ESTADO" es:', df['ESTADO'].mode()[0])
print('La moda de "D_EMPRESA_TRANSPORTISTA" es:', df['D_EMPRESA_TRANSPORTISTA'].mode()[0])
print('La moda de "C_ID_CONDUCTOR" es:', df['C_ID_CONDUCTOR'].mode()[0])
print('La moda de "TIPO_SERVICIO" es:', df['TIPO_SERVICIO'].mode()[0])
print('La moda de "TIPOCAMION" es:', df['TIPOCAMION'].mode()[0])
print('La moda de "TIPO_PRODUCTO" es:', df['TIPO_PRODUCTO'].mode()[0])
print('La moda de "TIPO_ASIGNACION" es:', df['TIPO_ASIGNACION'].mode()[0])
print('La moda de "C_PLANTA" es:', df['C_PLANTA'].mode()[0])
print('La moda de "TIPO_PERMISO" es:', df['TIPO_PERMISO'].mode()[0])
print('La moda de "ZONA_DESTINO" es:', df['ZONA_DESTINO'].mode()[0])
print('La moda de "CUIDAD" es:', df['CIUDAD'].mode()[0])
```

```
La moda de "TIPO_FORMA" es: PLANOS
La moda de "PLANTAORIGEN" es: Guerrero
La moda de "FECHAVIAJE" es: 13/10/2022 23:59:00
La moda de "C_CLIENTE" es: H000269647
La moda de "ESTADO" es: NUEVO LEON
La moda de "D_EMPRESA_TRANSPORTISTA" es: TRANPORTE68
La moda de "C_ID_CONDUCTOR" es: 92866
La moda de "TIPO_SERVICIO" es: LO
La moda de "TIPOCAMION" es: Plataforma 3 ejes Neumatica
La moda de "TIPO_PRODUCTO" es: ROLLO
La moda de "TIPO_ASIGNACION" es: MANUAL
La moda de "C_PLANTA" es: GUE
La moda de "TIPO_PERMISO" es: Traslado Externo
```

```
La moda de "ZONA_DESTINO" es: 0
La moda de "CUIDAD" es: APODACA
```

Gráficas pruebas dependencia de variables

```
planta_origen =df.PLANTAORIGEN.value_counts().plot(kind="pie")
planta_origen.set_title('Planta de origen')
planta_origen
```

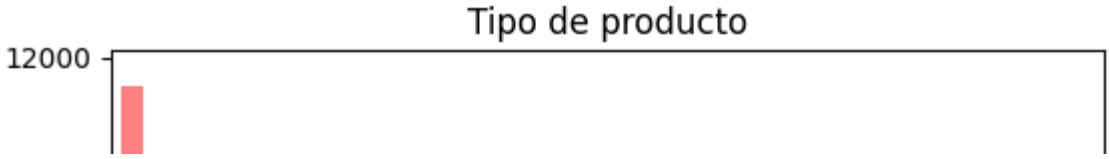
```
<Axes: title={'center': 'Planta de origen'}, ylabel='PLANTAORIGEN'>
```


```
planta_origen = df.TIPO_SERVICIO.value_counts().plot(kind="bar",color=['red', 'blue'],alpha=0
planta_origen.set_title('Tipo de servicio')
planta_origen
```

```
<Axes: title={'center': 'Tipo de servicio'}>
```

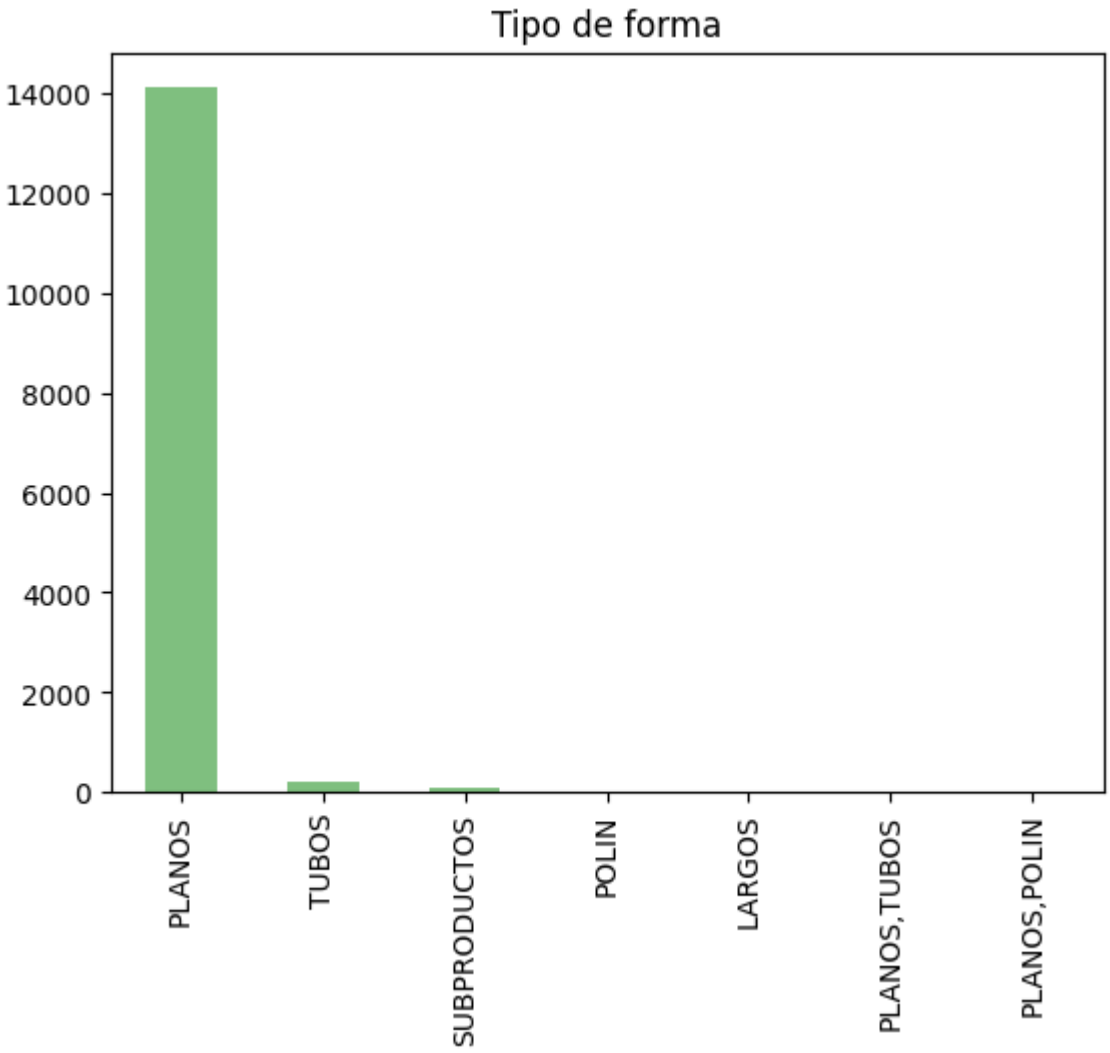

```
planta_origen = df.TIPO_PRODUCTO.value_counts().plot(kind="bar",color="red",alpha=0.5)
planta_origen.set_title('Tipo de producto')
planta_origen
```

```
<Axes: title={'center': 'Tipo de producto'}>
```




```
planta_origen = df.TIPO_FORMA.value_counts().plot(kind="bar",color="green",alpha=0.5)
planta_origen.set_title('Tipo de forma')
planta_origen
```

```
<Axes: title={'center': 'Tipo de forma'}>
```

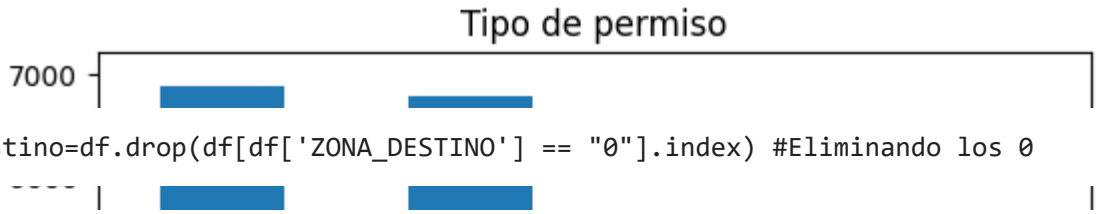



```
planta_origen = df.TIPO_ASIGNACION.value_counts().plot(kind="pie")
planta_origen.set_title('Tipo de asignación')
planta_origen
```

```
<Axes: title={'center': 'Tipo de asignación'}, ylabel='TIPO_ASIGNACION'>
```


```
planta_origen = df.TIPO_PERMISO.value_counts().plot(kind="bar")
planta_origen.set_title('Tipo de permiso')
planta_origen
```

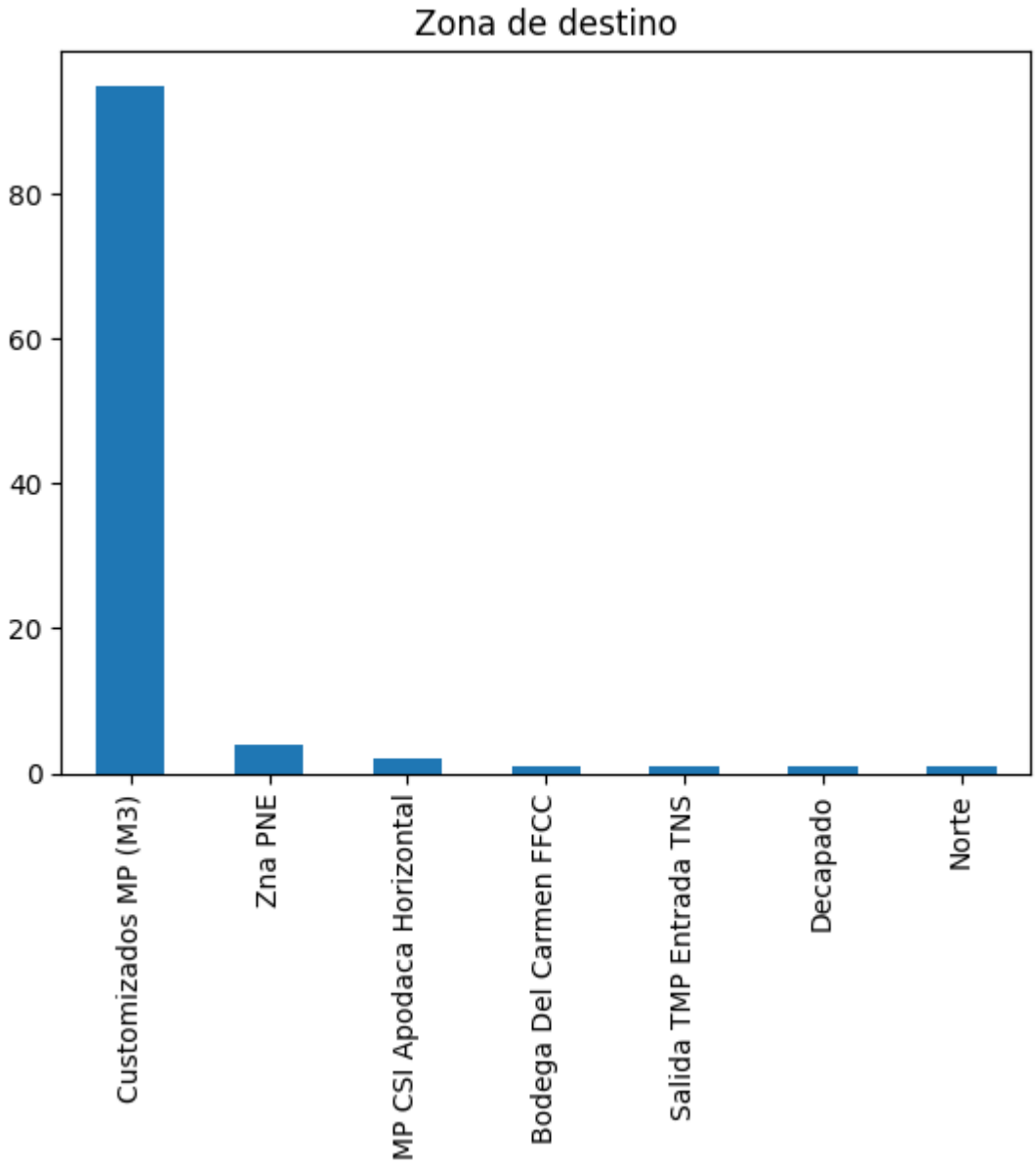
```
<Axes: title={'center': 'Tipo de permiso'}>
```




```
df_destino=df.drop(df[df['ZONA_DESTINO'] == "0"].index) #Eliminando los 0
```

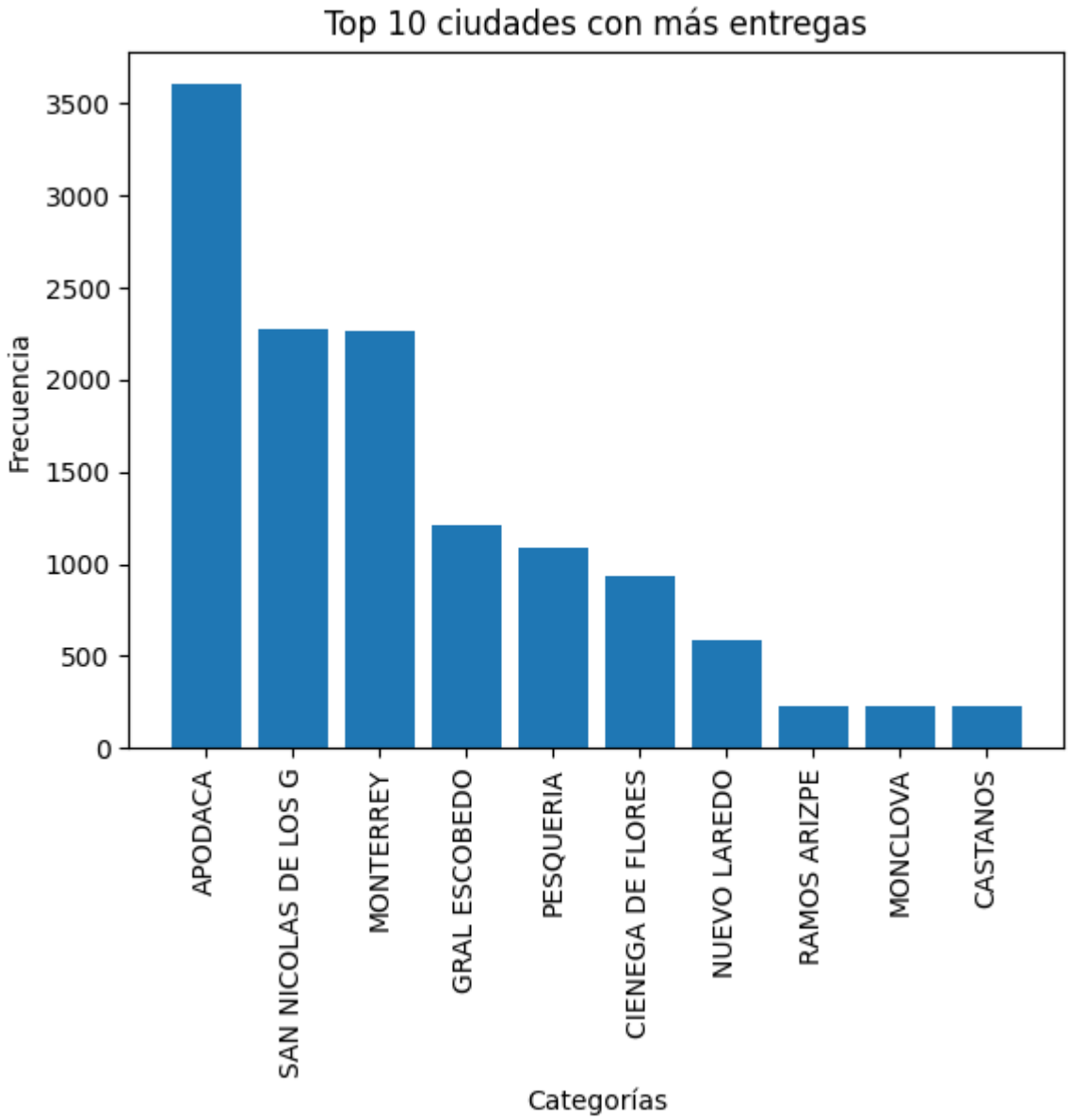
```
planta_origen = df_destino.ZONA_DESTINO.value_counts().plot(kind="bar")
planta_origen.set_title('Zona de destino')
planta_origen
```

```
<Axes: title={'center': 'Zona de destino'}>
```

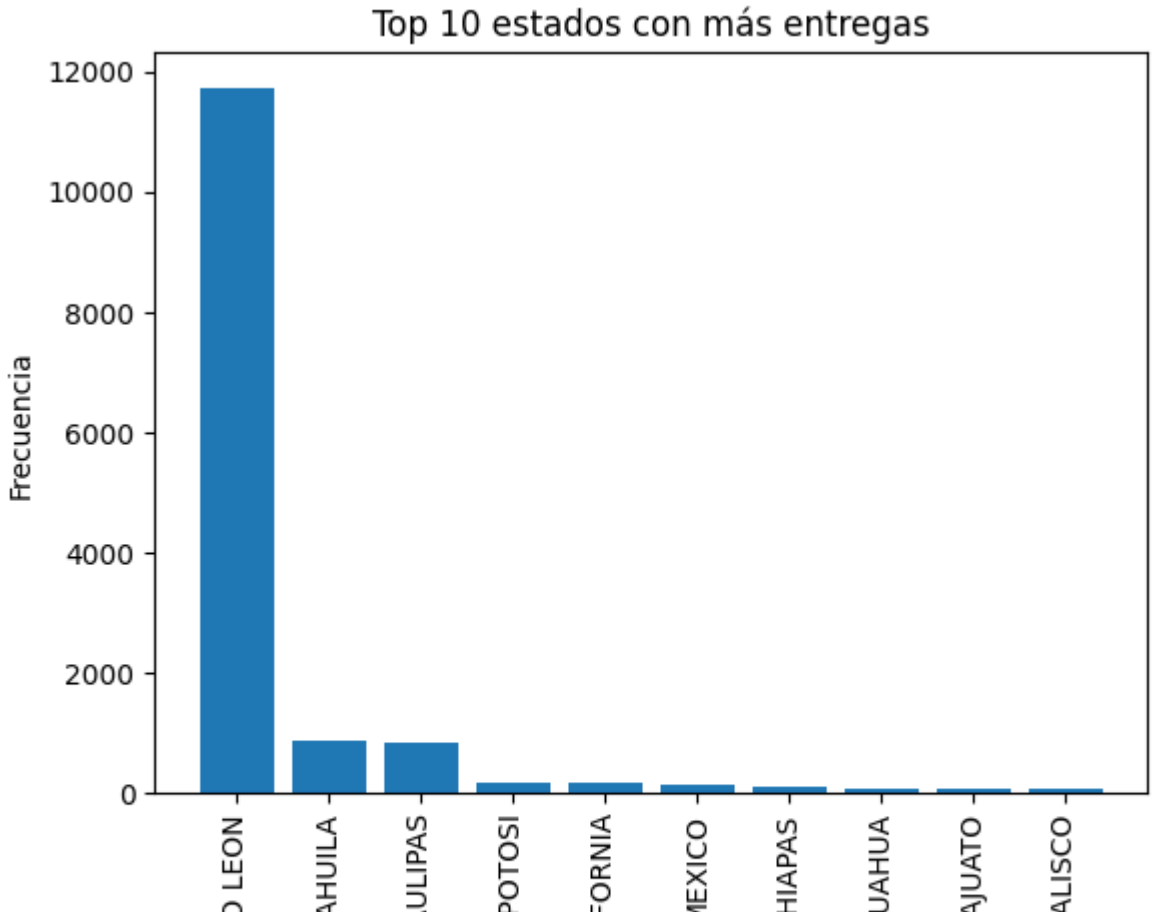



```
counts = df['CIUDAD'].value_counts()
counts = df['CIUDAD'].value_counts()
plt.bar(counts.index[:10], counts.values[:10])
plt.title('Top 10 ciudades con más entregas')
plt.xlabel('Categorías')
```

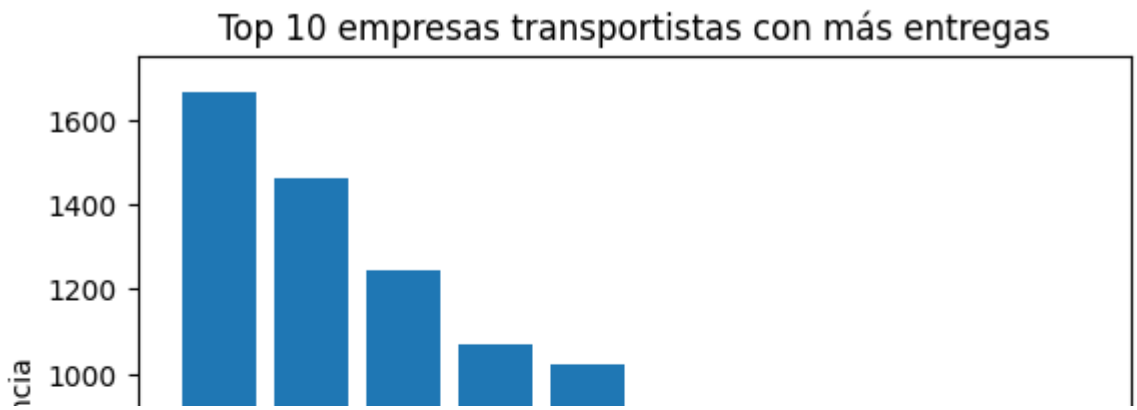
```
plt.ylabel('Frecuencia')
plt.xticks(rotation=90)
plt.show()
```




```
counts = df['ESTADO'].value_counts()
plt.bar(counts.index[:10], counts.values[:10])
plt.title('Top 10 estados con más entregas')
plt.xlabel('Categorías')
plt.ylabel('Frecuencia')
plt.xticks(rotation=90)
plt.show()
```




```
counts = df['D_EMPRESA_TRANSPORTISTA'].value_counts()
plt.bar(counts.index[:10], counts.values[:10])
plt.title('Top 10 empresas transportistas con más entregas')
plt.xlabel('Categorías')
plt.ylabel('Frecuencia')
plt.xticks(rotation=90)
plt.show()
```




```
df_num= df.drop(['C_ID_VIAJE','ID_ESTADO','C_ID_CONDUCTOR'], axis=1)

df_num.corr(method='pearson')
```

```
<ipython-input-62-76d53264c5bc>:3: FutureWarning: The default value of numeric_only in D
  df_num.corr(method='pearson')
```

	CANT_PROGRAMADA	PESO_NETO	CAP_MAXIMA
CANT_PROGRAMADA	1.000000	0.716711	0.338885
PESO_NETO	0.716711	1.000000	0.246934
CAP_MAXIMA	0.338885	0.246934	1.000000

```
correlation_matrix = df_num.corr().round(2)
sns.heatmap(data=correlation_matrix, annot=True)
plt.title('Correlación entre variables');
```

```
<ipython-input-63-0e9ccc8040c4>:1: FutureWarning: The default value of numeric_only in D
  correlation_matrix = df_num.corr().round(2)
```


```
#Tabla de distribución de frecuencias en variables categóricas
columnas_interes = ['TIPO_FORMA', 'PLANTAORIGEN', 'C_CLIENTE', 'ESTADO', 'D_EMPRESA_TRANSPORT
tablas_frecuencias = []

for columna in columnas_interes:
    tabla_frecuencia = df[columna].value_counts().reset_index()
    tabla_frecuencia.columns = ['Valor', f'Frecuencia_{columna}']
    tablas_frecuencias.append(tabla_frecuencia)

tablas_frecuencias
```

```
[          Valor  Frecuencia_TIPO_FORMA
 0        PLANOS                  14114
 1         TUBOS                    200
 2  SUBPRODUCTOS                     92
 3         POLIN                     13
 4        LARGOS                     10
 5  PLANOS,TUBOS                      8
 6  PLANOS,POLIN                      3,
       Valor  Frecuencia_PLANTAORIGEN
 0   Guerrero                   9154
 1  Pesqueria                   5286,
          Valor  Frecuencia_C_CLIENTE
 0    H000269647                  783
 1    H000126500                  596
 2    H010005034                  582
 3    H000122204                  459
 4    H000041807                  421
 ..          ...                  ...
 227  N000191639                    1
 228  N000001967                    1
 229  N000000687                    1
 230  N000003863                    1
 231  H000148817                    1

 [232 rows x 2 columns],
               Valor  Frecuencia_ESTADO
 0         NUEVO LEON              11714
 1           COAHUILA                859
 2         TAMAULIPAS                850
 3    SAN LUIS POTOSI                176
 4    BAJA CALIFORNIA                169
 5             MEXICO                121
 6            CHIAPAS                117
 7          CHIHUAHUA                 77
 8         GUANAJUATO                 69
 9            JALISCO                 54
 10  ESTADO DE MEXICO                 53
```

```
11         QUERETARO                 42
12  CIUDAD DE MEXICO                 28
13            SONORA                 20
14           DURANGO                 19
15           HIDALGO                 19
16    AGUASCALIENTES                 15
17          VERACRUZ                  9
18            PUEBLA                  8
19          TLAXCALA                  6
20           YUCATAN                  6
21           MORELOS                  3
22           SINALOA                  2
23         MICHOACAN                  2
24           TABASCO                  1
25            OAXACA                  1,
          Valor  Frecuencia_D_EMPRESA_TRANSPORTISTA
0   TRANPORTE68                                1666
1   TRANPORTE53                                1462
2   TRANPORTE38                                1242
3   TRANPORTE59                                1069
4   TRANPORTE63                                1021
```

```
#Medidas de posición no-central para variables cuantitativas: Cuartiles

# Lista de columnas para los cuartiles
columnas_interes_1 = ['CANT_PROGRAMADA', 'PESO_NETO', 'CAP_MAXIMA']

# Calcular los cuartiles solo para las columnas de interés
cuartiles = df[columnas_interes_1].quantile([0.25, 0.5, 0.75])

# Obtener los valores de los cuartiles
cuartil_25 = cuartiles.loc[0.25]
cuartil_50 = cuartiles.loc[0.5]
cuartil_75 = cuartiles.loc[0.75]

# Imprimir los valores de los cuartiles
print("Cuartil 25:")
print(cuartil_25)
print("\nCuartil 50 (Mediana):")
print(cuartil_50)
print("\nCuartil 75:")
print(cuartil_75)
```

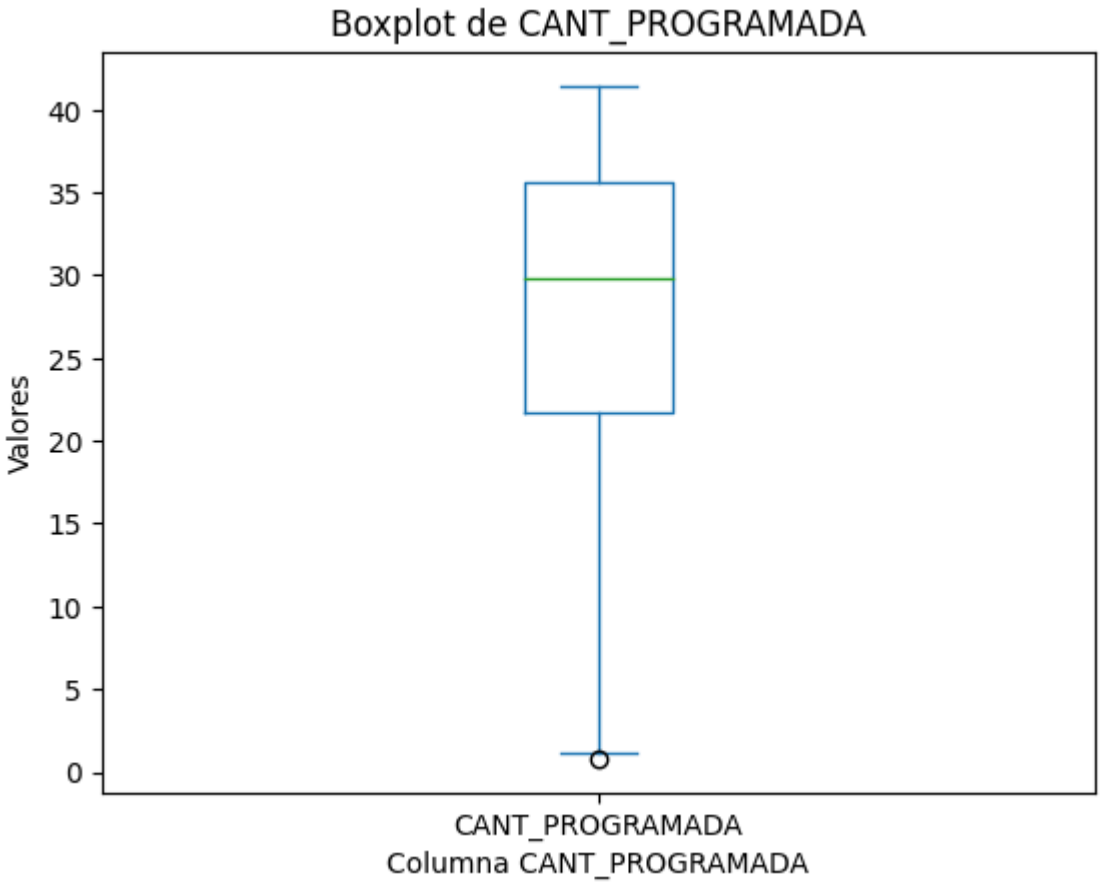
```
Cuartil 25:
CANT_PROGRAMADA       21.7395
PESO_NETO          21407.5000
CAP_MAXIMA            54.0000
Name: 0.25, dtype: float64

Cuartil 50 (Mediana):
CANT_PROGRAMADA       29.7635
PESO_NETO          28840.0000
CAP_MAXIMA            54.0000
Name: 0.5, dtype: float64
```

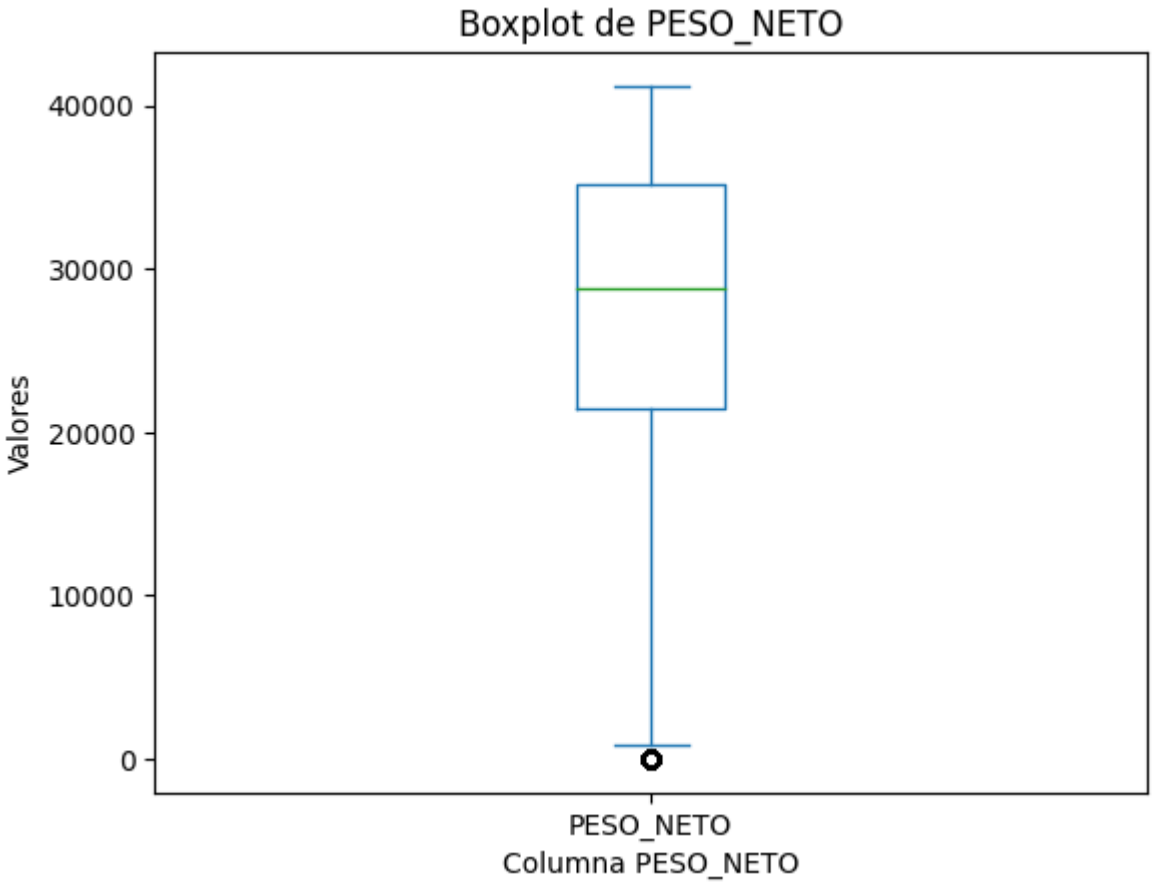
```
Cuartil 75:
CANT_PROGRAMADA       35.66075
PESO_NETO          35210.00000
CAP_MAXIMA            54.00000
Name: 0.75, dtype: float64
```

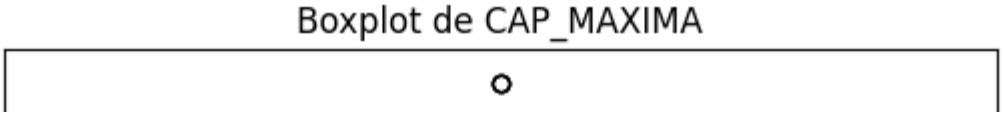
```
#Medidas de posición no-central para variables cuantitativas: Valores atípicos

# Calcular la media y desviación estándar de las columnas de interés
media = df[columnas_interes_1].mean()
desviacion_estandar = df[columnas_interes_1].std()

# Calcular los valores Z para cada valor en las columnas de interés
valores_z = (df[columnas_interes_1] - media) / desviacion_estandar

# Definir umbral para identificar valores atípicos (por ejemplo, Z > 3 o Z < -3)
umbral = 3

# Identificar valores atípicos basados en los valores Z
valores_atipicos = df[valores_z.abs() > umbral][columnas_interes_1]
valores_atipicos = valores_atipicos.dropna()

# Imprimir los valores atípicos
print(valores_atipicos)
```

```
Empty DataFrame
Columns: [CANT_PROGRAMADA, PESO_NETO, CAP_MAXIMA]
Index: []
```

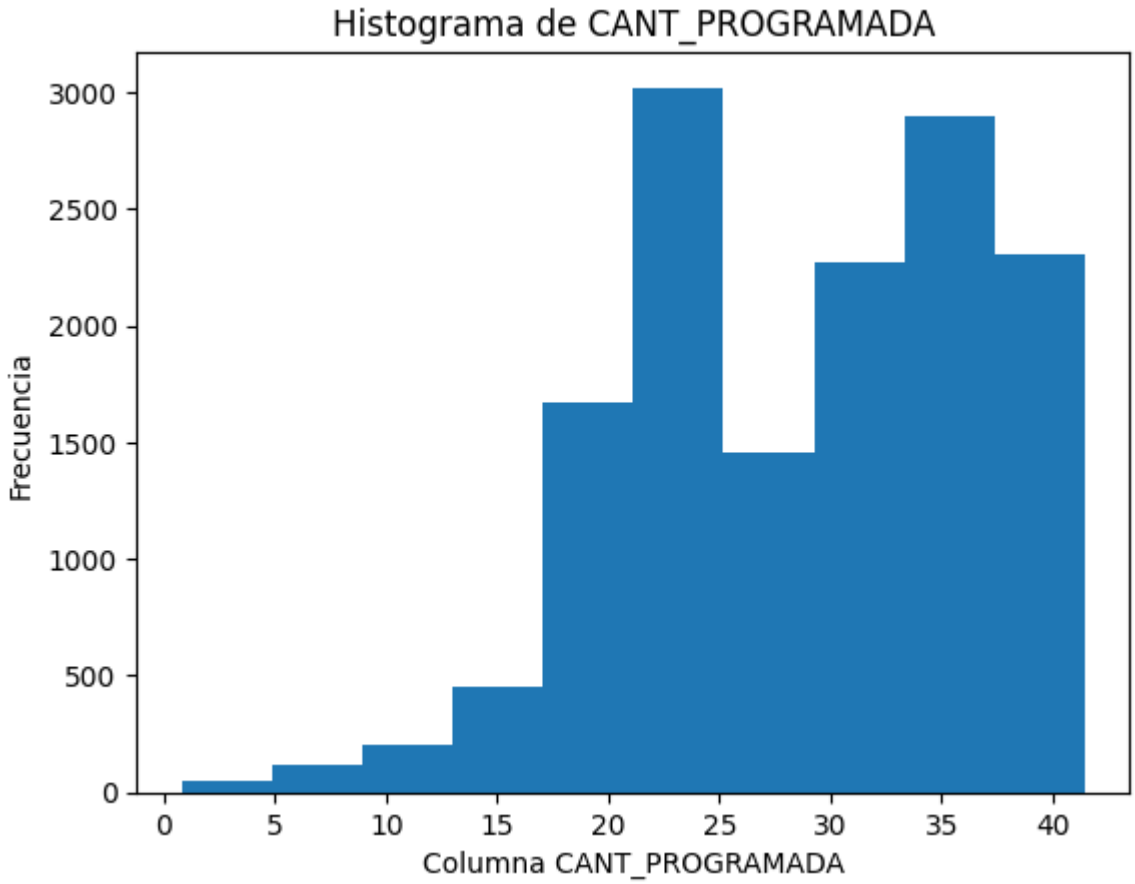
```
#Medidas de posición no-central para variables cuantitativas: Boxplots

# Crear los boxplots
for col in columnas_interes_1:
    plt.figure()
    df[col].plot(kind='box')
    plt.xlabel('Columna {}'.format(col))
    plt.ylabel('Valores')
    plt.title('Boxplot de {}'.format(col))

plt.show()
```

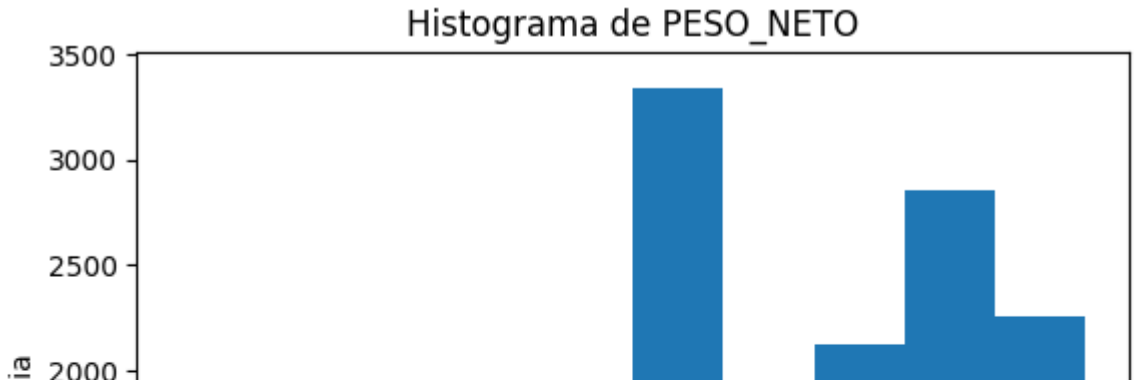

Boxplot de CANT_PROGRAMADA
Valores
40
35
30
25
20
15
10
5
0
CANT_PROGRAMADA
Columna CANT_PROGRAMADA


Boxplot de PESO_NETO
Valores
40000
30000
20000
10000
0
PESO_NETO
Columna PESO_NETO


Boxplot de CAP_MAXIMA

```
#Análisis de distribución de datos (Histogramas), Identificar si tiene forma simétrica o asim

#Crear histogramas separados para cada columna de valores cuantitativos
for col in columnas_interes_1:
    plt.figure()
    df[col].plot(kind='hist', bins=10)
    plt.xlabel('Columna {}'.format(col))
    plt.ylabel('Frecuencia')
    plt.title('Histograma de {}'.format(col))

plt.show()
```

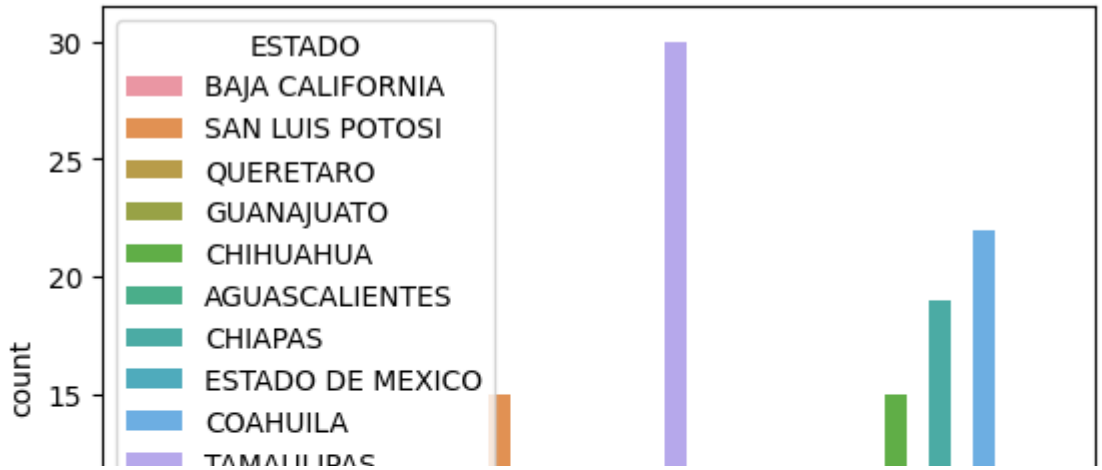




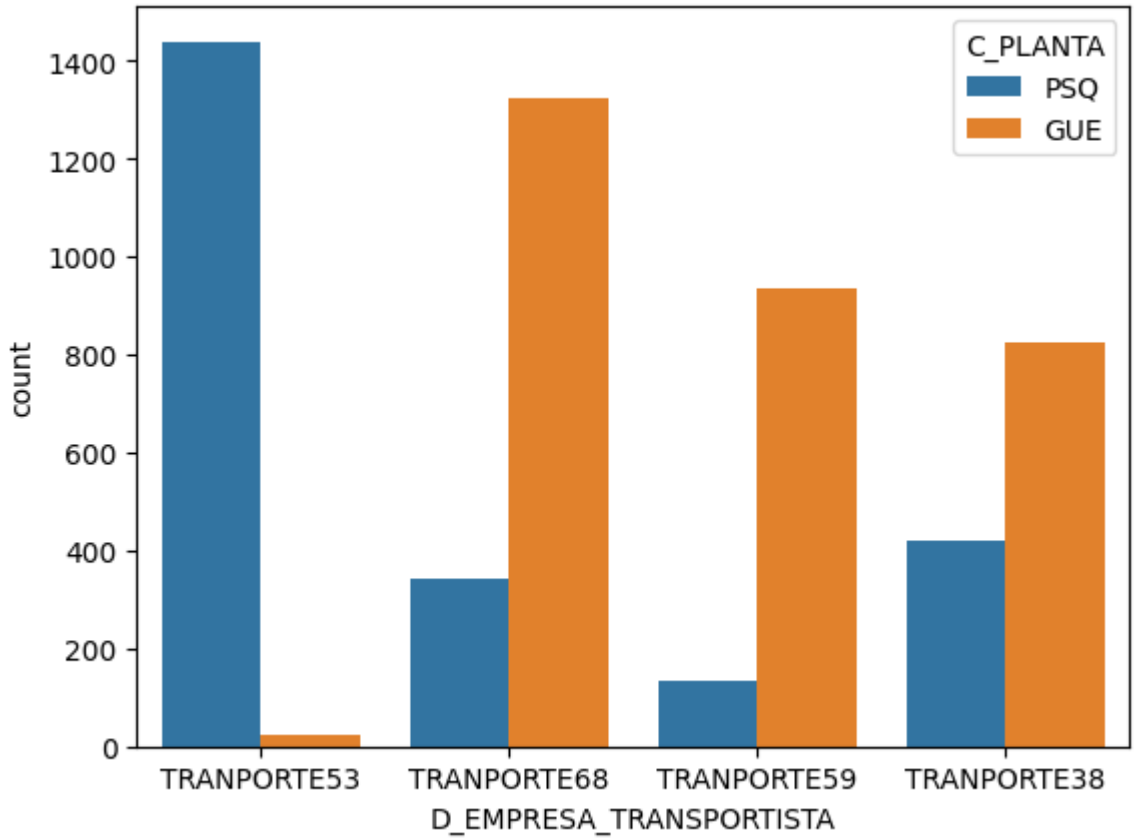

```
# Obtener los 10 valores más comunes en la columna "D_EMPRESA_TRANSPORTISTA"
top_3 = df['D_EMPRESA_TRANSPORTISTA'].value_counts().nlargest(4).index.tolist()

# Seleccionar las filas del DataFrame que contienen los valores más comunes
df_top3 = df[df['D_EMPRESA_TRANSPORTISTA'].isin(top_3)]

df_top3
```

	PLANTAORIGEN	FECHADESPACHO	C_ID_VIAJE	FECHAVIAJE	C_CLIENTE	D_CLIENTE	ID_E
5	Pesqueria	01/10/2022 01:42:42	46252415	01/10/2022 17:44:00	N000100266	CLIENTE190	
8	Pesqueria	01/10/2022 01:47:33	46247529	30/09/2022 13:11:00	H000122205	CLIENTE95	
9	Pesqueria	01/10/2022 01:28:20	46252420	01/10/2022 17:44:00	H000269818	CLIENTE232	
17	Pesqueria	01/10/2022 00:01:49	46251419	30/09/2022 16:57:00	H000756702	CLIENTE6	
21	Pesqueria	01/10/2022 02:18:00	46251518	30/09/2022 23:59:00	N000100834	CLIENTE236	
...	...	...	...	...	...	...	
9028	Guerrero	11/12/2022 08:27:07	47152009	07-11-2022 22:24:00	NaN	CLIENTE224	
9029	Guerrero	11/12/2022 22:16:01	47249070	12-11-2022 15:24:00	NaN	CLIENTE58	
9030	Guerrero	11/18/2022 21:46:42	47372259	18-11-2022 23:59:00	NaN	CLIENTE58	

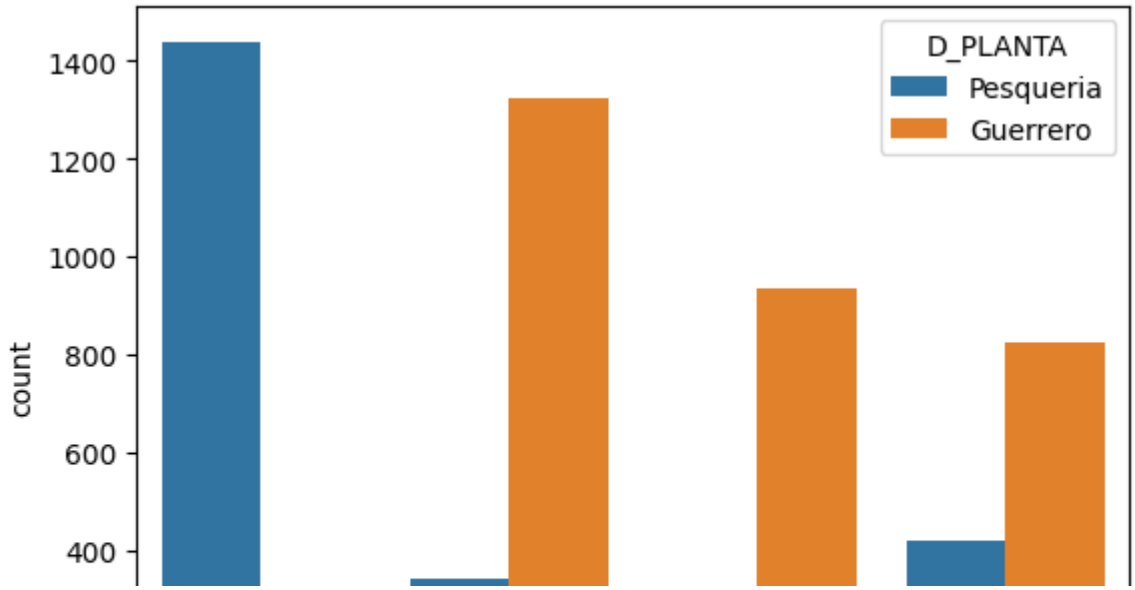
```
top_5 = df['D_EMPRESA_TRANSPORTISTA'].value_counts().nlargest(5).index.tolist()

# Seleccionar las filas del DataFrame que contienen los valores más comunes
df_top5 = df[df['D_EMPRESA_TRANSPORTISTA'].isin(top_5)]



filas_a_eliminar = df_top5[df_top5['ESTADO'] =="NUEVO LEON"].index

# eliminar las filas seleccionadas del DataFrame
df_NL = df_top5.drop(filas_a_eliminar)

CrosstabResult=pd.crosstab(df_NL['D_EMPRESA_TRANSPORTISTA'],df_NL['ESTADO'])
sns.countplot(x="D_EMPRESA_TRANSPORTISTA", hue="ESTADO", data=df_NL);
```



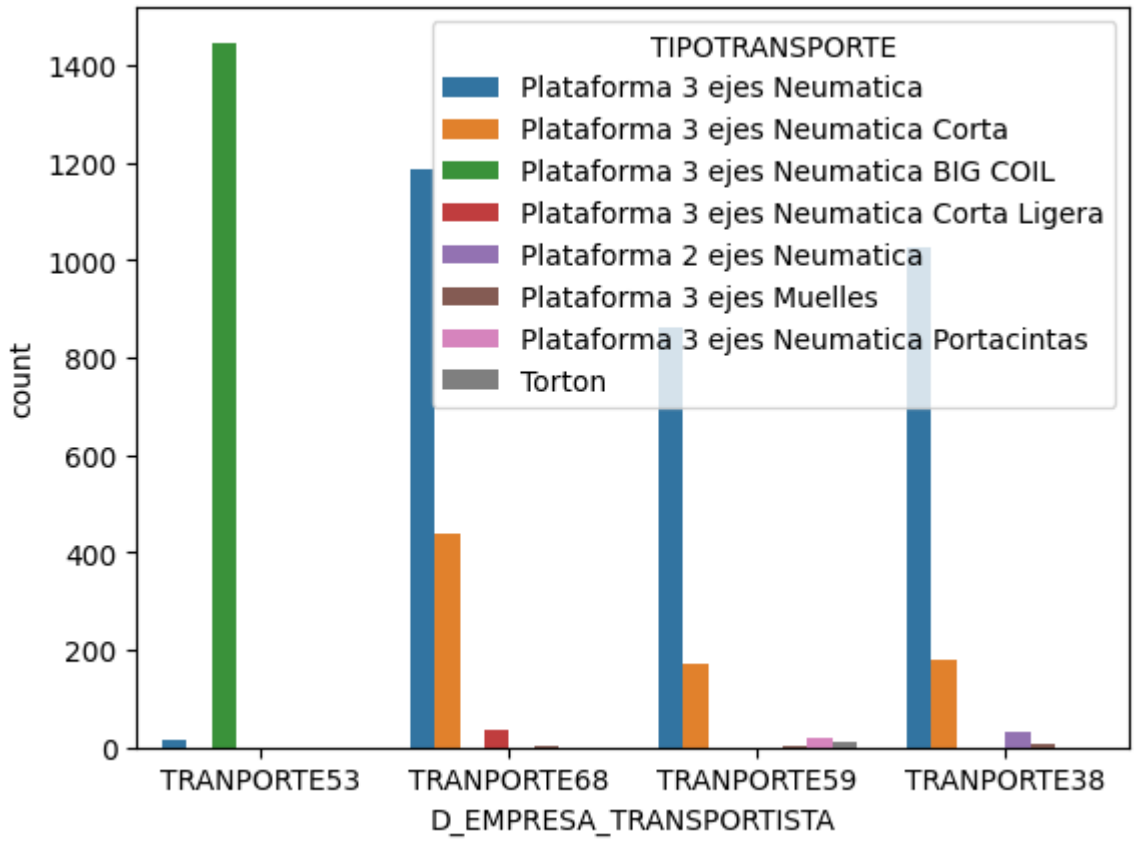

```
CrosstabResult=pd.crosstab(df_top3['D_EMPRESA_TRANSPORTISTA'],df_top3['C_PLANTA'])
sns.countplot(x="D_EMPRESA_TRANSPORTISTA", hue="C_PLANTA", data=df_top3);
```




```
CrosstabResult=pd.crosstab(df_top3['D_EMPRESA_TRANSPORTISTA'],df_top3['D_PLANTA'])
sns.countplot(x="D_EMPRESA_TRANSPORTISTA", hue="D_PLANTA", data=df_top3);
```

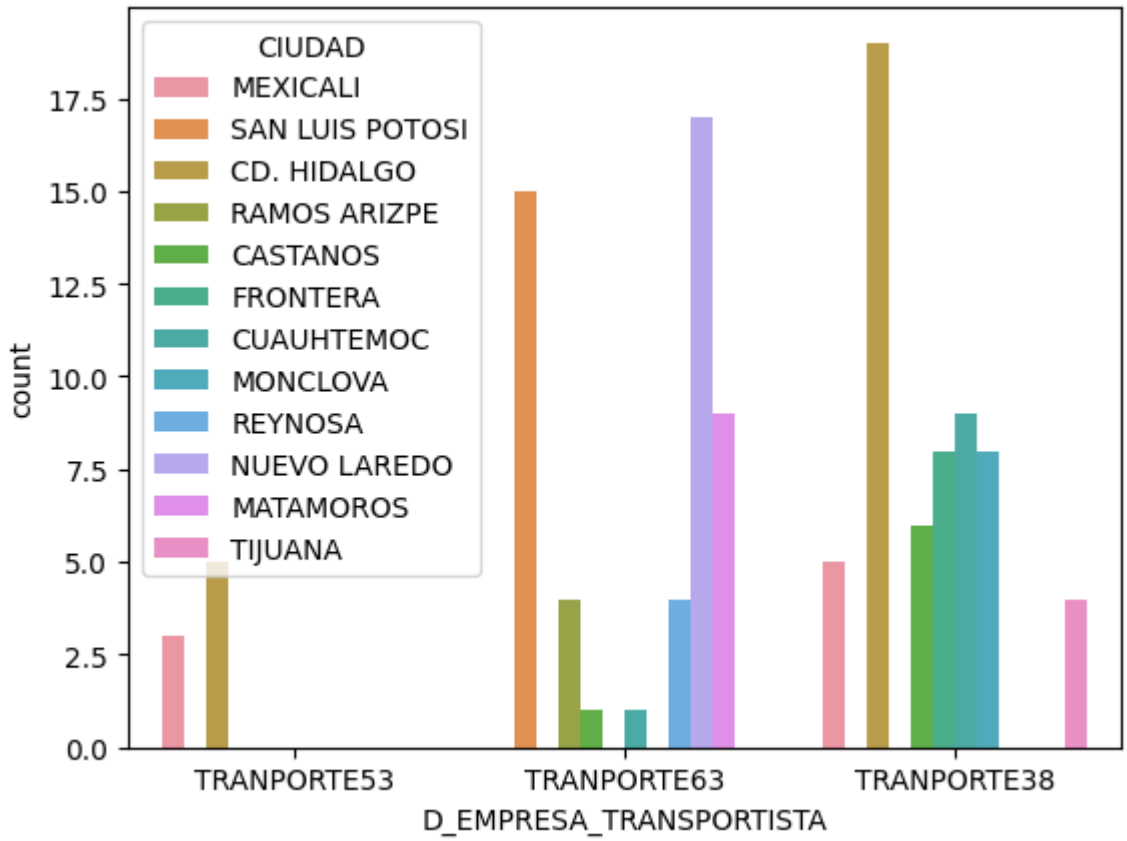



```
CrosstabResult=pd.crosstab(df_top3['D_EMPRESA_TRANSPORTISTA'],df_top3['TIPOTRANSPORTE'])
sns.countplot(x="D_EMPRESA_TRANSPORTISTA", hue="TIPOTRANSPORTE", data=df_top3);
```

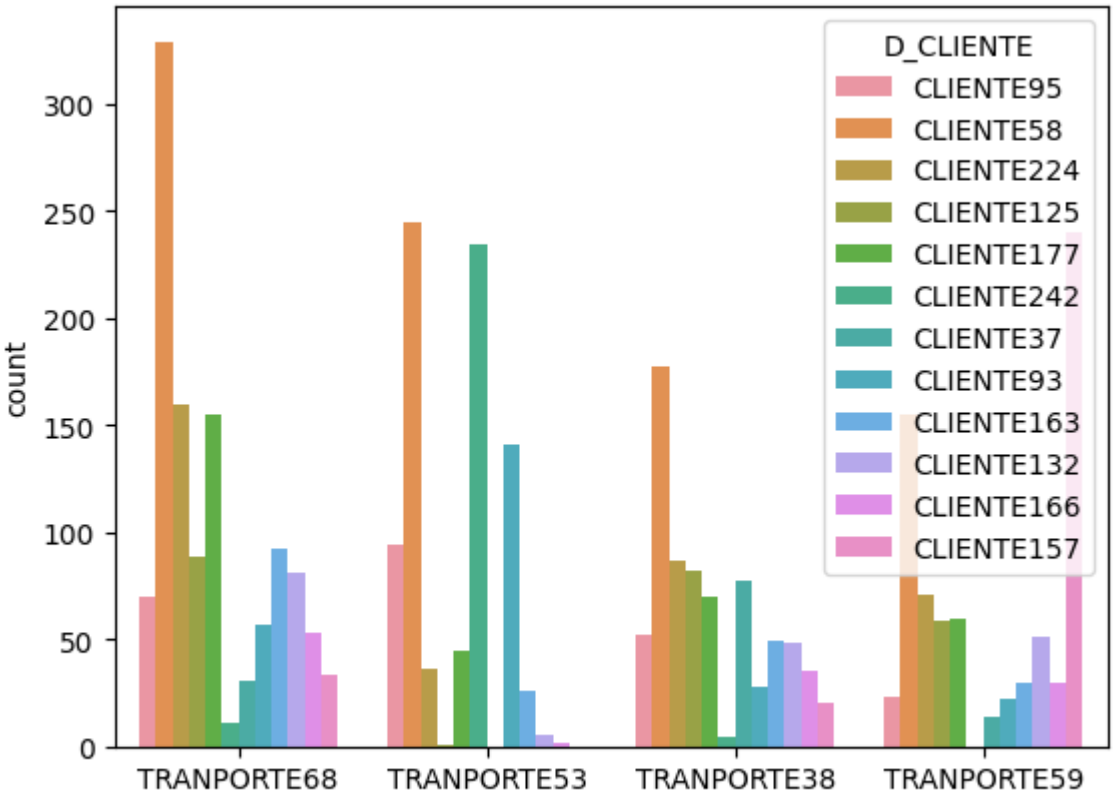



```
top_5 = df_NL['D_EMPRESA_TRANSPORTISTA'].value_counts().nlargest(5).index.tolist()

# Seleccionar las filas del DataFrame que contienen los valores más comunes
df_top5 = df_NL[df_NL['D_EMPRESA_TRANSPORTISTA'].isin(top_5)]

ciudad_5 = df_top5['CIUDAD'].value_counts().nlargest(12).index.tolist()
```

```
# Seleccionar las filas del DataFrame que contienen los valores más comunes
ciudad_top5 = df_top5[df_top5['CIUDAD'].isin(ciudad_5)]

filas_a_eliminar = ciudad_top5[ciudad_top5['CIUDAD'] =="APODACA"].index

# eliminar las filas seleccionadas del DataFrame
df_ciu = ciudad_top5.drop(filas_a_eliminar)

CrosstabResult=pd.crosstab(df_ciu['D_EMPRESA_TRANSPORTISTA'],df_ciu['CIUDAD'])
sns.countplot(x="D_EMPRESA_TRANSPORTISTA", hue="CIUDAD", data=df_ciu);
```




```
top_cl = df_top3['D_CLIENTE'].value_counts().nlargest(12).index.tolist()

# Seleccionar las filas del DataFrame que contienen los valores más comunes
df_cl = df_top3[df_top3['D_CLIENTE'].isin(top_cl)]

filas_a_eliminar = df_cl[df_cl['D_CLIENTE'] =="CLIENTE58"].index

# eliminar las filas seleccionadas del DataFrame
df_ciu2 = df_cl.drop(filas_a_eliminar)

CrosstabResult=pd.crosstab(df_cl['D_EMPRESA_TRANSPORTISTA'],df_cl['D_CLIENTE'])
sns.countplot(x="D_EMPRESA_TRANSPORTISTA", hue="D_CLIENTE", data=df_cl);
```

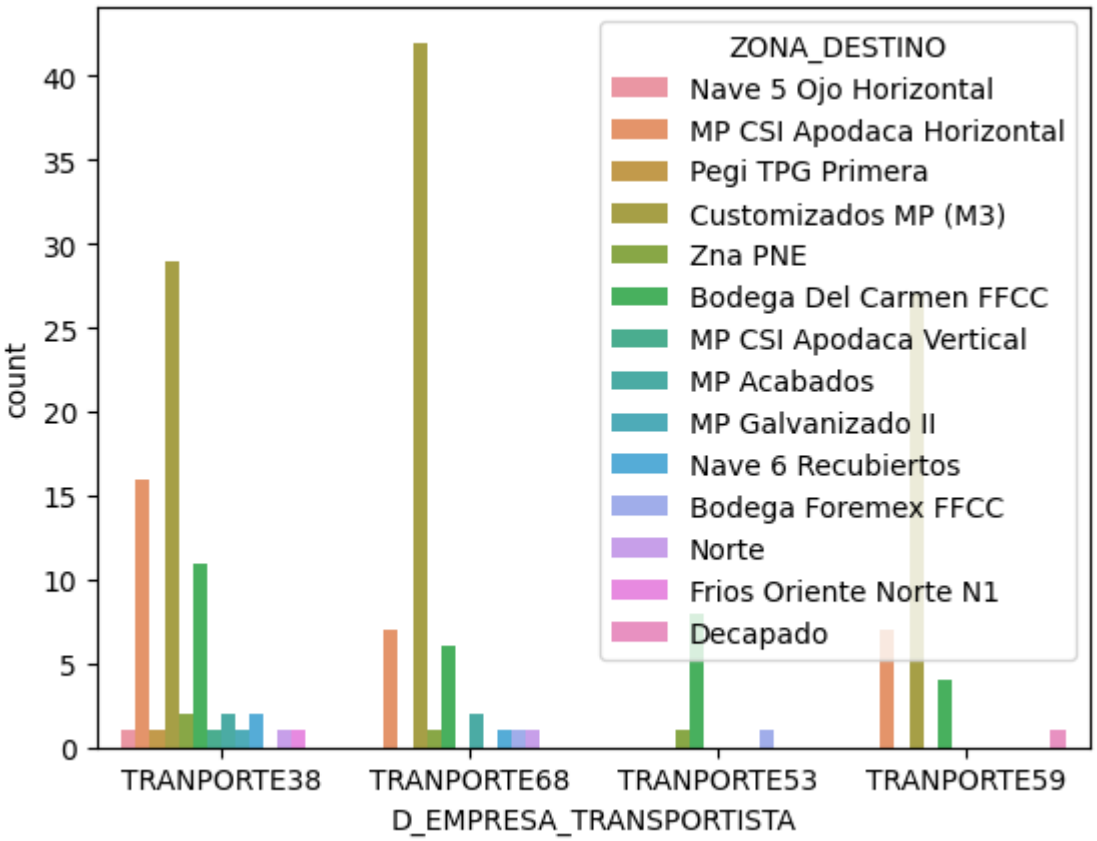



```
df_top3['ZONA_DESTINO'] = df_top3['ZONA_DESTINO'].replace("0", pd.np.nan)

# Eliminar las filas que contienen valores nulos solo en la columna 'ZONA_DESTINO'
df_top3 = df_top3.dropna(subset=['ZONA_DESTINO'])

df_top3
```

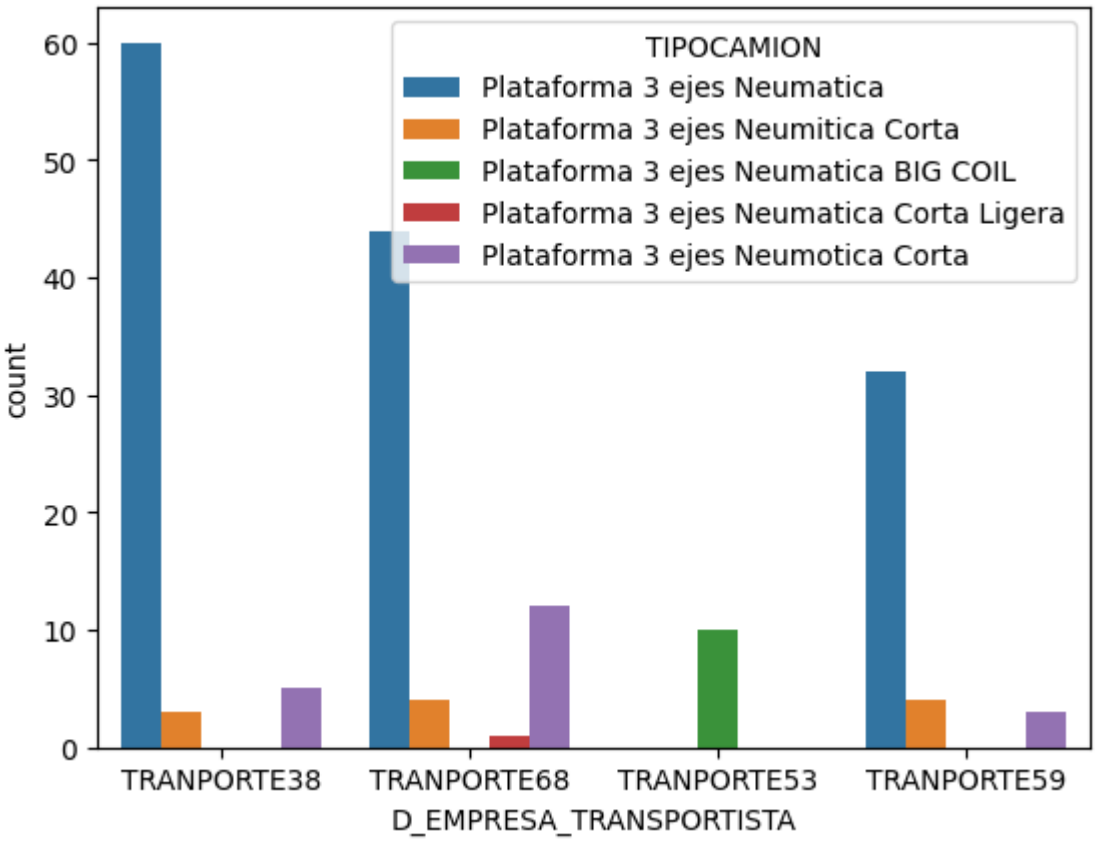
```
<ipython-input-76-7cc77455d3a0>:1: FutureWarning: The pandas.np module is deprecated and
  df_top3['ZONA_DESTINO'] = df_top3['ZONA_DESTINO'].replace("0", pd.np.nan)
<ipython-input-76-7cc77455d3a0>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
  df_top3['ZONA_DESTINO'] = df_top3['ZONA_DESTINO'].replace("0", pd.np.nan)
```

	PLANTAORIGEN	FECHADESPACHO	C_ID_VIAJE	FECHAVIAJE	C_CLIENTE	D_CLIENTE	ID_ES
26	Pesqueria	01/10/2022 05:12:41	46257534	03/10/2022 23:14:00	NaN	CLIENTE125	
27	Pesqueria	01/10/2022 05:13:59	46256288	02/10/2022 22:10:00	NaN	CLIENTE237	
72	Pesqueria	01/10/2022 06:25:41	46264732	01/10/2022 23:59:00	NaN	CLIENTE237	
84	Pesqueria	01/10/2022 09:05:49	46265791	01/10/2022 23:59:00	NaN	CLIENTE237	

```
CrosstabResult=pd.crosstab(df_top3['D_EMPRESA_TRANSPORTISTA'],df_top3['ZONA_DESTINO'])
sns.countplot(x="D_EMPRESA_TRANSPORTISTA", hue="ZONA_DESTINO", data=df_top3);
```



```
CrosstabResult=pd.crosstab(df_top3['D_EMPRESA_TRANSPORTISTA'],df_top3['TIPOCAMION'])
sns.countplot(x="D_EMPRESA_TRANSPORTISTA", hue="TIPOCAMION", data=df_top3);
```




```
frecuencia = df['PESO_NETO'].value_counts()
frecuencia = frecuencia.sort_values(ascending=False)
top10 = frecuencia[:10]
df_peso_top10 = df[df['PESO_NETO'].isin(top10.index)]

df_peso_top10 = df_peso_top10.drop(df_peso_top10[df_peso_top10['PESO_NETO'] == 0].index)

df_peso_top10
```

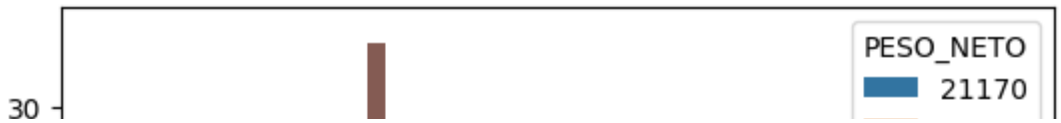
	PLANTAORIGEN	FECHADESPACHO	C_ID_VIAJE	FECHAVIAJE	C_CLIENTE	D_CLIENTE	ID_E
320	Pesqueria	02/10/2022 10:09:18	46272867	15/10/2022 23:59:00	NaN	CLIENTE58	
1032	Pesqueria	05/10/2022 02:27:40	46366486	05/10/2022 23:59:00	N000100732	CLIENTE220	
1135	Pesqueria	05/10/2022 11:13:56	46376805	05/10/2022 23:59:00	H000122205	CLIENTE95	
1515	Pesqueria	06/10/2022 20:59:19	46408884	06/10/2022 12:09:00	NaN	CLIENTE125	
1707	Pesqueria	07/10/2022 09:41:35	46405335	06/10/2022 09:11:00	NaN	CLIENTE58	
...	...	...	...	...	...	...	
8754	Guerrero	11/08/2022 20:40:10	47168516	08-11-2022 17:53:00	H010000377	CLIENTE188	

```
# Obtener la frecuencia de cada string en el dataframe
frecuencia2 = df_peso_top10['TIPOCAMION'].value_counts()

# Seleccionar las 10 strings más comunes
strings_comunes = frecuencia2[:5].index.tolist()

# Filtrar las filas que contienen las strings más comunes
df_camion_top10 = df_peso_top10[df_peso_top10['TIPOCAMION'].isin(strings_comunes)]
df_camion_top10
```

	PLANTAORIGEN	FECHADESPACHO	C_ID_VIAJE	FECHAVIAJE	C_CLIENTE	D_CLIENTE	ID_E
320	Pesqueria	02/10/2022 10:09:18	46272867	15/10/2022 23:59:00	NaN	CLIENTE58	
1032	Pesqueria	05/10/2022 02:27:40	46366486	05/10/2022 23:59:00	N000100732	CLIENTE220	
1707	Pesqueria	07/10/2022 09:41:35	46405335	06/10/2022 09:11:00	NaN	CLIENTE58	
1747	Pesqueria	07/10/2022 12:14:08	46431092	10/10/2022 07:09:00	NaN	CLIENTE242	
1862	Pesqueria	07/10/2022	46433017	10/10/2022	NaN	CLIENTE242	

```
CrosstabResult=pd.crosstab(df_camion_top10['TIPOCAMION'],df_camion_top10['PESO_NETO'])
sns.countplot(x="TIPOCAMION", hue="PESO_NETO", data=df_camion_top10);
plt.xticks(rotation=90)
```

```
(array([0, 1, 2, 3, 4]),
 [Text(0, 0, 'Plataforma 3 ejes Neumatica BIG COIL'),
  Text(1, 0, 'Plataforma 3 ejes Neumatica'),
  Text(2, 0, 'Plataforma 3 ejes Neumotica Corta'),
  Text(3, 0, 'Plataforma 2 ejes Neumatica'),
  Text(4, 0, 'Plataforma 3 ejes Neumatica Corta')])
```



Pruebas de dependencia chi-cuadrado

```
df = df.drop(['C_CLIENTE', 'FECHADESPACHO', 'C_ID_VIAJE', 'FECHAVIAJE', 'ID_ESTADO', 'C_ID_CO
              'F_PRESENTACION', 'F_LLEGADANAVE', 'F_EGRESONAVE', 'F_INGRESOPLANTA', 'F_PESAJEE
              'F_PESAJESALIDA', 'TIPO_ASIGNACION', 'C_PLANTA', 'D_PLANTA', 'ZONA_DESTINO'], ax
```

```
from scipy.stats import chi2_contingency

# Crear una lista con todas las columnas del DataFrame
columnas = df.columns

# Utilizar un bucle for anidado para evaluar todas las combinaciones de columnas
for col1 in columnas:
    for col2 in columnas:
        if col1 != col2:  # Evitar comparar una columna consigo misma
            tabla_contingencia = pd.crosstab(df[col1], df[col2])
            chi2, p, dof, expected = chi2_contingency(tabla_contingencia)
            print(f'\n\nPrueba de chi-cuadrado entre {col1} y {col2}:')
            print(f'Estadístico de prueba: {chi2}')
            print(f'Valor p: {p}')
            print(f'Grados de libertad: {dof}')
```

```
Prueba de chi-cuadrado entre PLANTAORIGEN y D_CLIENTE:
Estadístico de prueba: 7981.573673582509
Valor p: 0.0
Grados de libertad: 272


Prueba de chi-cuadrado entre PLANTAORIGEN y ESTADO:
Estadístico de prueba: 935.2092324334516
Valor p: 3.162774466085186e-181
Grados de libertad: 25


Prueba de chi-cuadrado entre PLANTAORIGEN y D_EMPRESA_TRANSPORTISTA:
Estadístico de prueba: 6757.945127367248
Valor p: 0.0
Grados de libertad: 90
```

```
Prueba de chi-cuadrado entre PLANTAORIGEN y TIPO_SERVICIO:
Estadístico de prueba: 50.76034846343704
Valor p: 1.0436087182764713e-12
Grados de libertad: 1


Prueba de chi-cuadrado entre PLANTAORIGEN y TIPOTRANSPORTE:
Estadístico de prueba: 3847.7959239961956
Valor p: 0.0
Grados de libertad: 13


Prueba de chi-cuadrado entre PLANTAORIGEN y TIPO_PRODUCTO:
Estadístico de prueba: 2055.954883336751
Valor p: 0.0
Grados de libertad: 22


Prueba de chi-cuadrado entre PLANTAORIGEN y TIPO_FORMA:
Estadístico de prueba: 204.59438178866083
Valor p: 1.9956400294280227e-41
Grados de libertad: 6


Prueba de chi-cuadrado entre PLANTAORIGEN y TIPO_PERMISO:
Estadístico de prueba: 933.2684926215687
Valor p: 5.379349058187976e-202
Grados de libertad: 3


Prueba de chi-cuadrado entre PLANTAORIGEN y PESO_NETO:
Estadístico de prueba: 5781.089515826718
Valor p: 3.0477889337452374e-238
Grados de libertad: 2624


Prueba de chi-cuadrado entre PLANTAORIGEN y CIUDAD:
Estadístico de prueba: 2345.220800216277
```

Transformación y análisis de datos

```
cols_to_encode = ['PLANTAORIGEN', 'TIPO_SERVICIO', 'TIPO_PRODUCTO', 'TIPO_FORMA', 'TIPO_PERMI

# Aplicar codificación one-hot binaria solo a las columnas seleccionadas
df_encoded = pd.get_dummies(df, columns=cols_to_encode, drop_first=True)

# Mostrar el DataFrame codificado
df_encoded
```

	D_CLIENTE	ESTADO	D_EMPRESA_TRANSPORTISTA	TIPOTRANSPORTE	PESO_NETO	
0	CLIENTE27	BAJA CALIFORNIA	TRANPORTE31	Plataforma 3 ejes Neumatica Cortina	30670	M
1	CLIENTE198	GUANAJUATO	TRANPORTE63	Plataforma 3 ejes Neumatica	36890	
2	CLIENTE231	NUEVO LEON	TRANPORTE52	Plataforma 3 ejes Neumatica BIG COIL	23300	D
3	CLIENTE232	BAJA CALIFORNIA	TRANPORTE72	Plataforma 3 ejes Neumatica	36330	M
4	CLIENTE233	TAMAULIPAS	TRANPORTE26	Plataforma 3 ejes Neumatica	20480	A
...	...	...	...	...	...	
9149	CLIENTE6	NUEVO LEON	TRANPORTE63	Plataforma 3 ejes Neumatica BIG COIL	34580	ES
9150	CLIENTE219	COAHUILA	TRANPORTE48	Plataforma 3 ejes Neumatica	19090	FR
9151	CLIENTE219	COAHUILA	TRANPORTE48	Plataforma 3 ejes Neumatica	19090	FR
9152	CLIENTE219	COAHUILA	TRANPORTE48	Plataforma 3 ejes Neumatica	19090	FR
9153	CLIENTE38	CHIAPAS	TRANPORTE61	Plataforma 3 ejes Neumatica	22880	H

14440 rows × 39 columns

```
df_encoded.columns
```

```
Index(['D_CLIENTE', 'ESTADO', 'D_EMPRESA_TRANSPORTISTA', 'TIPOTRANSPORTE',
       'PESO_NETO', 'CIUDAD', 'PLANTAORIGEN_Pesqueria', 'TIPO_SERVICIO_LO',
       'TIPO_PRODUCTO_BANDA', 'TIPO_PRODUCTO_BANDA,HOJA',
       'TIPO_PRODUCTO_BANDA,ROLLO', 'TIPO_PRODUCTO_CHATARRA',
       'TIPO_PRODUCTO_CINTA', 'TIPO_PRODUCTO_CINTA,HOJA',
       'TIPO_PRODUCTO_CINTA,HOJA,ROLLO', 'TIPO_PRODUCTO_CINTA,POLIN Z',
       'TIPO_PRODUCTO_CINTA,ROLLO', 'TIPO_PRODUCTO_CUADRADO',
       'TIPO_PRODUCTO_CUADRADO,RECTANGULAR',
       'TIPO_PRODUCTO_CUADRADO,RECTANGULAR,REDONDO',
       'TIPO_PRODUCTO_CUADRADO,REDONDO',
       'TIPO_PRODUCTO_CUADRADO,REDONDO,ROLLO', 'TIPO_PRODUCTO_CUADRADO,ROLLO',
       'TIPO_PRODUCTO_HOJA', 'TIPO_PRODUCTO_HOJA,ROLLO',
       'TIPO_PRODUCTO_POLIN Z', 'TIPO_PRODUCTO_RECTANGULAR',
       'TIPO_PRODUCTO_RECTANGULAR,REDONDO', 'TIPO_PRODUCTO_REDONDO',
       'TIPO_PRODUCTO_ROLLO', 'TIPO_FORMA_PLANOS', 'TIPO_FORMA_PLANOS,POLIN',
```

```
            'TIPO_FORMA_PLANOS,TUBOS', 'TIPO_FORMA_POLIN',
            'TIPO_FORMA_SUBPRODUCTOS', 'TIPO_FORMA_TUBOS', 'TIPO_PERMISO_Despacho',
            'TIPO_PERMISO_Traslado Externo',
            'TIPO_PERMISO_Traslado Externo sin Pesada'],
           dtype='object')


df_encoded.loc[(df_encoded['TIPO_FORMA_PLANOS,POLIN'] == 1), 'TIPO_FORMA_PLANOS'] = 1
df_encoded.loc[(df_encoded['TIPO_FORMA_PLANOS,POLIN'] == 1), 'TIPO_FORMA_POLIN'] = 1


df_encoded.loc[(df_encoded['TIPO_FORMA_PLANOS,TUBOS'] == "PLANOS,TUBOS"), 'TIPO_FORMA_PLANOS'
df_encoded.loc[(df_encoded['TIPO_FORMA_PLANOS,TUBOS'] == "PLANOS,TUBOS"), 'TIPO_FORMA_TUBOS']

df_encoded = df_encoded.drop(['TIPO_FORMA_PLANOS,POLIN', 'TIPO_FORMA_PLANOS,TUBOS'], axis=1)
df_encoded.columns

     Index(['D_CLIENTE', 'ESTADO', 'D_EMPRESA_TRANSPORTISTA', 'TIPOTRANSPORTE',
            'PESO_NETO', 'CIUDAD', 'PLANTAORIGEN_Pesqueria', 'TIPO_SERVICIO_LO',
            'TIPO_PRODUCTO_BANDA', 'TIPO_PRODUCTO_BANDA,HOJA',
            'TIPO_PRODUCTO_BANDA,ROLLO', 'TIPO_PRODUCTO_CHATARRA',
            'TIPO_PRODUCTO_CINTA', 'TIPO_PRODUCTO_CINTA,HOJA',
            'TIPO_PRODUCTO_CINTA,HOJA,ROLLO', 'TIPO_PRODUCTO_CINTA,POLIN Z',
            'TIPO_PRODUCTO_CINTA,ROLLO', 'TIPO_PRODUCTO_CUADRADO',
            'TIPO_PRODUCTO_CUADRADO,RECTANGULAR',
            'TIPO_PRODUCTO_CUADRADO,RECTANGULAR,REDONDO',
            'TIPO_PRODUCTO_CUADRADO,REDONDO',
            'TIPO_PRODUCTO_CUADRADO,REDONDO,ROLLO', 'TIPO_PRODUCTO_CUADRADO,ROLLO',
            'TIPO_PRODUCTO_HOJA', 'TIPO_PRODUCTO_HOJA,ROLLO',
            'TIPO_PRODUCTO_POLIN Z', 'TIPO_PRODUCTO_RECTANGULAR',
            'TIPO_PRODUCTO_RECTANGULAR,REDONDO', 'TIPO_PRODUCTO_REDONDO',
            'TIPO_PRODUCTO_ROLLO', 'TIPO_FORMA_PLANOS', 'TIPO_FORMA_POLIN',
            'TIPO_FORMA_SUBPRODUCTOS', 'TIPO_FORMA_TUBOS', 'TIPO_PERMISO_Despacho',
            'TIPO_PERMISO_Traslado Externo',
            'TIPO_PERMISO_Traslado Externo sin Pesada'],
           dtype='object')


df_encoded.loc[(df_encoded['TIPO_PRODUCTO_CINTA,HOJA'] == 1), 'TIPO_PRODUCTO_CINTA'] = 1
df_encoded.loc[(df_encoded['TIPO_PRODUCTO_CINTA,HOJA'] == 1), 'TIPO_PRODUCTO_HOJA'] = 1

df_encoded.loc[(df_encoded['TIPO_PRODUCTO_CINTA,ROLLO'] == 1), 'TIPO_PRODUCTO_CINTA'] = 1
df_encoded.loc[(df_encoded['TIPO_PRODUCTO_CINTA,ROLLO'] == 1), 'TIPO_PRODUCTO_ROLLO'] = 1

df_encoded.loc[(df_encoded['TIPO_PRODUCTO_CINTA,POLIN Z'] == 1), 'TIPO_PRODUCTO_CINTA'] = 1
df_encoded.loc[(df_encoded['TIPO_PRODUCTO_CINTA,POLIN Z'] == 1), 'TIPO_PRODUCTO_POLIN Z'] = 1

df_encoded.loc[(df_encoded['TIPO_PRODUCTO_CINTA,HOJA,ROLLO'] == 1), 'TIPO_PRODUCTO_CINTA'] =
df_encoded.loc[(df_encoded['TIPO_PRODUCTO_CINTA,HOJA,ROLLO'] == 1), 'TIPO_PRODUCTO_HOJA'] = 1
df_encoded.loc[(df_encoded['TIPO_PRODUCTO_CINTA,HOJA,ROLLO'] == 1), 'TIPO_PRODUCTO_ROLLO'] =

df_encoded.loc[(df_encoded['TIPO_PRODUCTO_BANDA,HOJA'] == 1), 'TIPO_PRODUCTO_BANDA'] = 1
df_encoded.loc[(df_encoded['TIPO_PRODUCTO_BANDA,HOJA'] == 1), 'TIPO_PRODUCTO_HOJA'] = 1
```

```
df_encoded.loc[(df_encoded['TIPO_PRODUCTO_BANDA,ROLLO'] == 1), 'TIPO_PRODUCTO_BANDA'] = 1
df_encoded.loc[(df_encoded['TIPO_PRODUCTO_BANDA,ROLLO'] == 1), 'TIPO_PRODUCTO_ROLLO'] = 1

df_encoded.loc[(df_encoded['TIPO_PRODUCTO_HOJA,ROLLO'] == 1), 'TIPO_PRODUCTO_ROLLO'] = 1
df_encoded.loc[(df_encoded['TIPO_PRODUCTO_HOJA,ROLLO'] == 1), 'TIPO_PRODUCTO_HOJA'] = 1

df_encoded.loc[(df_encoded['TIPO_PRODUCTO_CUADRADO,REDONDO'] == 1), 'TIPO_PRODUCTO_CUADRADO']
df_encoded.loc[(df_encoded['TIPO_PRODUCTO_CUADRADO,REDONDO'] == 1), 'TIPO_PRODUCTO_REDONDO']

df_encoded.loc[(df_encoded['TIPO_PRODUCTO_CUADRADO,RECTANGULAR'] == 1), 'TIPO_PRODUCTO_CUADRA
df_encoded.loc[(df_encoded['TIPO_PRODUCTO_CUADRADO,RECTANGULAR'] == 1), 'TIPO_PRODUCTO_RECTAN

df_encoded.loc[(df_encoded['TIPO_PRODUCTO_CUADRADO,ROLLO'] == 1), 'TIPO_PRODUCTO_CUADRADO'] =
df_encoded.loc[(df_encoded['TIPO_PRODUCTO_CUADRADO,ROLLO'] == 1), 'TIPO_PRODUCTO_ROLLO'] = 1

df_encoded.loc[(df_encoded['TIPO_PRODUCTO_RECTANGULAR,REDONDO'] == 1), 'TIPO_PRODUCTO_RECTANG
df_encoded.loc[(df_encoded['TIPO_PRODUCTO_RECTANGULAR,REDONDO'] == 1), 'TIPO_PRODUCTO_REDONDO

df_encoded.loc[(df_encoded['TIPO_PRODUCTO_CUADRADO,RECTANGULAR,REDONDO'] == 1), 'TIPO_PRODUCT
df_encoded.loc[(df_encoded['TIPO_PRODUCTO_CUADRADO,RECTANGULAR,REDONDO'] == 1), 'TIPO_PRODUCT
df_encoded.loc[(df_encoded['TIPO_PRODUCTO_CUADRADO,RECTANGULAR,REDONDO'] == 1), 'TIPO_PRODUCT

df_encoded.loc[(df_encoded['TIPO_PRODUCTO_CUADRADO,REDONDO,ROLLO'] == 1), 'TIPO_PRODUCTO_CUAD
df_encoded.loc[(df_encoded['TIPO_PRODUCTO_CUADRADO,REDONDO,ROLLO'] == 1), 'TIPO_PRODUCTO_ROLL
df_encoded.loc[(df_encoded['TIPO_PRODUCTO_CUADRADO,REDONDO,ROLLO'] == 1), 'TIPO_PRODUCTO_REDO

df_encoded = df_encoded.drop(['TIPO_PRODUCTO_CINTA,HOJA', 'TIPO_PRODUCTO_CINTA,ROLLO', 'TIPO_
                              'TIPO_PRODUCTO_CINTA,HOJA,ROLLO', 'TIPO_PRODUCTO_BANDA,HOJA', 'T
                              'TIPO_PRODUCTO_HOJA,ROLLO', 'TIPO_PRODUCTO_CUADRADO,REDONDO', 'T
                              'TIPO_PRODUCTO_CUADRADO,ROLLO', 'TIPO_PRODUCTO_RECTANGULAR,REDON
                              'TIPO_PRODUCTO_CUADRADO,REDONDO,ROLLO'], axis=1)
df_encoded.columns
```

```
Index(['D_CLIENTE', 'ESTADO', 'D_EMPRESA_TRANSPORTISTA', 'TIPOTRANSPORTE',
       'PESO_NETO', 'CIUDAD', 'PLANTAORIGEN_Pesqueria', 'TIPO_SERVICIO_LO',
       'TIPO_PRODUCTO_BANDA', 'TIPO_PRODUCTO_CHATARRA', 'TIPO_PRODUCTO_CINTA',
       'TIPO_PRODUCTO_CUADRADO', 'TIPO_PRODUCTO_HOJA', 'TIPO_PRODUCTO_POLIN Z',
       'TIPO_PRODUCTO_RECTANGULAR', 'TIPO_PRODUCTO_REDONDO',
       'TIPO_PRODUCTO_ROLLO', 'TIPO_FORMA_PLANOS', 'TIPO_FORMA_POLIN',
       'TIPO_FORMA_SUBPRODUCTOS', 'TIPO_FORMA_TUBOS', 'TIPO_PERMISO_Despacho',
       'TIPO_PERMISO_Traslado Externo',
       'TIPO_PERMISO_Traslado Externo sin Pesada'],
      dtype='object')
```

```
df_encoded.insert(loc=df_encoded.columns.get_loc('TIPO_PRODUCTO_ROLLO'), column='TIPO_PRODUCT
df_encoded.insert(loc=df_encoded.columns.get_loc('TIPO_FORMA_TUBOS'), column='TIPO_FORMA_LARG
df_encoded.insert(loc=df_encoded.columns.get_loc('TIPO_PERMISO_Traslado Externo'), column='TI
df_encoded.columns
```

```
Index(['D_CLIENTE', 'ESTADO', 'D_EMPRESA_TRANSPORTISTA', 'TIPOTRANSPORTE',
       'PESO_NETO', 'CIUDAD', 'PLANTAORIGEN_Pesqueria', 'TIPO_SERVICIO_LO',
       'TIPO_PRODUCTO_BANDA', 'TIPO_PRODUCTO_CHATARRA', 'TIPO_PRODUCTO_CINTA',
```

```
            'TIPO_PRODUCTO_CUADRADO', 'TIPO_PRODUCTO_HOJA', 'TIPO_PRODUCTO_POLIN Z',
            'TIPO_PRODUCTO_RECTANGULAR', 'TIPO_PRODUCTO_REDONDO',
            'TIPO_PRODUCTO_ALAMBRON', 'TIPO_PRODUCTO_ROLLO', 'TIPO_FORMA_PLANOS',
            'TIPO_FORMA_POLIN', 'TIPO_FORMA_SUBPRODUCTOS', 'TIPO_FORMA_LARGOS',
            'TIPO_FORMA_TUBOS', 'TIPO_PERMISO_Despacho',
            'TIPO_PERMISO_Cliente ENVIA', 'TIPO_PERMISO_Traslado Externo',
            'TIPO_PERMISO_Traslado Externo sin Pesada'],
           dtype='object')


df_encoded.loc[(df_encoded['TIPO_PRODUCTO_BANDA'] == 0) & (df_encoded['TIPO_PRODUCTO_CHATARRA
      & (df_encoded['TIPO_PRODUCTO_CHATARRA'] == 0) & (df_encoded['TIPO_PRODUCTO_CHATARRA']
      & (df_encoded['TIPO_PRODUCTO_CUADRADO'] == 0) & (df_encoded['TIPO_PRODUCTO_HOJA'] == 0
      & (df_encoded['TIPO_PRODUCTO_RECTANGULAR'] == 0) & (df_encoded['TIPO_PRODUCTO_REDONDO'
      , 'TIPO_PRODUCTO_ALAMBRON'] = 1
df_encoded.loc[(df_encoded['TIPO_FORMA_PLANOS'] == 0) & (df_encoded['TIPO_FORMA_POLIN'] == 0)
       & (df_encoded['TIPO_FORMA_TUBOS'] == 0), 'TIPO_FORMA_LARGOS'] = 1
df_encoded.loc[(df_encoded['TIPO_PERMISO_Despacho'] == 0) & (df_encoded['TIPO_PERMISO_Traslad
       , 'TIPO_PERMISO_Cliente ENVIA'] = 1

df_encoded.head()
```

	D_CLIENTE	ESTADO	D_EMPRESA_TRANSPORTISTA	TIPOTRANSPORTE	PESO_NETO	CIU
0	CLIENTE27	BAJA CALIFORNIA	TRANPORTE31	Plataforma 3 ejes Neumatica Cortina	30670	MEXIC
1	CLIENTE198	GUANAJUATO	TRANPORTE63	Plataforma 3 ejes Neumatica	36890	LE
2	CLIENTE231	NUEVO LEON	TRANPORTE52	Plataforma 3 ejes Neumatica BIG COIL	23300	S NICO DE LO
3	CLIENTE232	BAJA CALIFORNIA	TRANPORTE72	Plataforma 3 ejes Neumatica	36330	MEXIC
4	CLIENTE233	TAMAULIPAS	TRANPORTE26	Plataforma 3 ejes Neumatica	20480	ALTAM

5 rows × 27 columns

```
from sklearn.preprocessing import LabelEncoder

# Crear un codificador de etiquetas
label_encoder = LabelEncoder()

# Aplicar la codificación numérica
df_encoded['D_CLIENTE'] = label_encoder.fit_transform(df_encoded['D_CLIENTE'])

# Crear un codificador de etiquetas
```

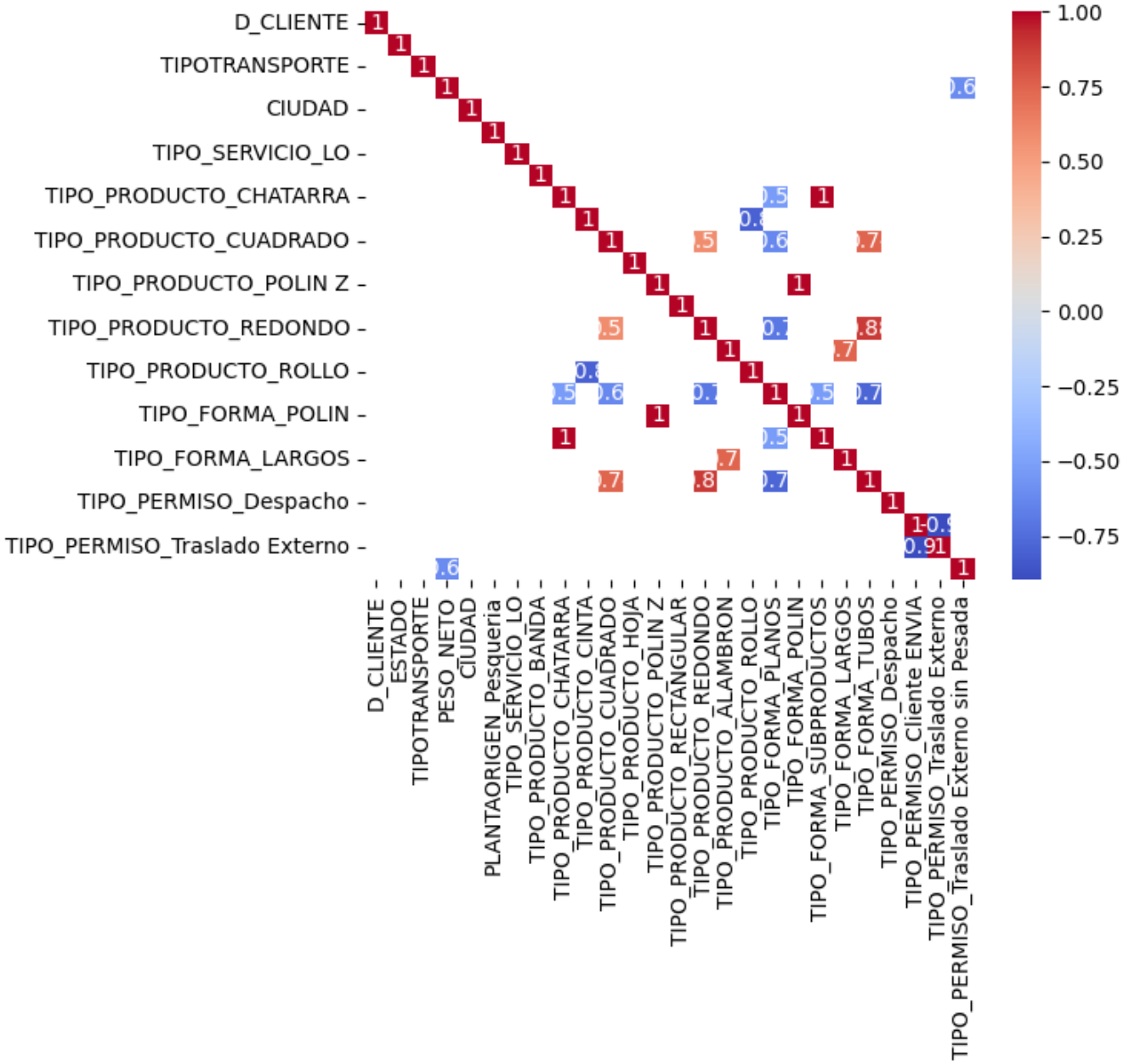
```
label_encoder = LabelEncoder()

# Aplicar la codificación numérica
df_encoded['ESTADO'] = label_encoder.fit_transform(df_encoded['ESTADO'])

# Crear un codificador de etiquetas
label_encoder = LabelEncoder()

# Aplicar la codificación numérica
df_encoded['CIUDAD'] = label_encoder.fit_transform(df_encoded['CIUDAD'])

# Crear un codificador de etiquetas
label_encoder = LabelEncoder()

# Aplicar la codificación numérica
df_encoded['TIPOTRANSPORTE'] = label_encoder.fit_transform(df_encoded['TIPOTRANSPORTE'])

df_encoded['D_EMPRESA_TRANSPORTISTA'] = df_encoded['D_EMPRESA_TRANSPORTISTA'].str.replace('TR

df_encoded.head()
```

	D_CLIENTE	ESTADO	D_EMPRESA_TRANSPORTISTA	TIPOTRANSPORTE	PESO_NETO	CIUDAD	PLANTA
0	187	1	TRANSPORTE31	9	30670	39	
1	107	8	TRANSPORTE63	5	36890	35	
2	145	14	TRANSPORTE52	6	23300	61	
3	146	1	TRANSPORTE72	5	36330	39	
4	147	22	TRANSPORTE26	5	20480	1	

5 rows × 27 columns

```
# Calcular la matriz de correlación
correlation_matrix = df_encoded.corr()

# Establecer un umbral de correlación
umbral = 0.5

# Filtrar la matriz de correlación
correlation_filtered = correlation_matrix[abs(correlation_matrix) >= umbral]

# Crear el mapa de calor
sns.heatmap(correlation_filtered, annot=True, cmap='coolwarm')

# Mostrar el mapa de calor
plt.show()
```

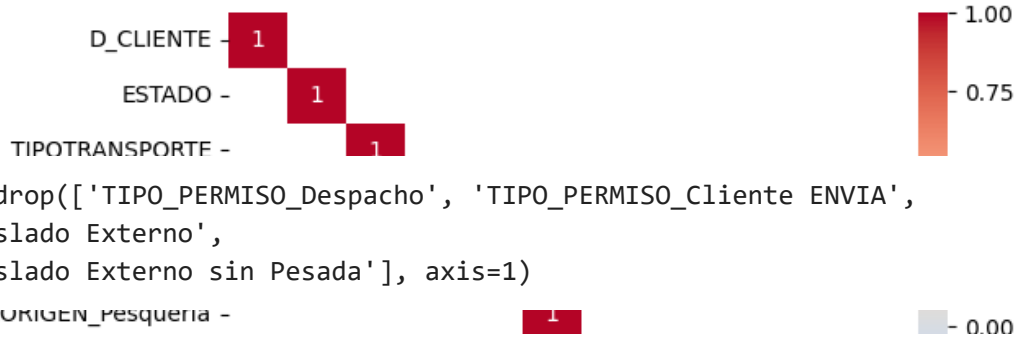
```
<ipython-input-91-61a77f26b6b2>:2: FutureWarning: The default value of numeric_only in D
  correlation_matrix = df_encoded.corr()
```




```
df_encoded = df_encoded.drop([
       'TIPO_PRODUCTO_BANDA', 'TIPO_PRODUCTO_CHATARRA', 'TIPO_PRODUCTO_CINTA',
       'TIPO_PRODUCTO_CUADRADO', 'TIPO_PRODUCTO_HOJA', 'TIPO_PRODUCTO_POLIN Z',
       'TIPO_PRODUCTO_RECTANGULAR', 'TIPO_PRODUCTO_REDONDO',
       'TIPO_PRODUCTO_ALAMBRON', 'TIPO_PRODUCTO_ROLLO', 'TIPO_FORMA_PLANOS',
       'TIPO_FORMA_POLIN', 'TIPO_FORMA_SUBPRODUCTOS', 'TIPO_FORMA_LARGOS',
       'TIPO_FORMA_TUBOS'], axis=1)
```

```
columnas_seleccionadas = df_encoded.corr()

# Calcular la matriz de correlación
matriz_correlacion = columnas_seleccionadas.corr()

# Establecer un umbral de correlación
umbral = 0.5

# Filtrar la matriz de correlación
correlation_filtered = matriz_correlacion[abs(matriz_correlacion) >= umbral]

# Crear el mapa de calor
sns.heatmap(correlation_filtered, annot=True, cmap='coolwarm')

# Mostrar el mapa de calor
plt.show()
```

```
<ipython-input-93-c82676257c51>:1: FutureWarning: The default value of numeric_only in D
  columnas_seleccionadas = df_encoded.corr()
```




```
df_encoded = df_encoded.drop(['TIPO_PERMISO_Despacho', 'TIPO_PERMISO_Cliente ENVIA',
       'TIPO_PERMISO_Traslado Externo',
       'TIPO_PERMISO_Traslado Externo sin Pesada'], axis=1)
```

```
import scipy.stats as stats

# Definir la variable cualitativa y las variables numéricas
variable_cualitativa = "D_EMPRESA_TRANSPORTISTA"
variables_numericas = list(df_encoded.select_dtypes(exclude=['object']).columns)  # Lista de

# Realizar el análisis de varianza (ANOVA) para cada variable numérica
for variable_numerica in variables_numericas:
    anova_result = stats.f_oneway(*(df_encoded[df_encoded[variable_cualitativa] == cat][varia
    f_statistic = anova_result[0]  # Estadístico F
    p_value = anova_result[1]  # Valor p
    print(f"Variable numérica: {variable_numerica}")
    print("Estadístico F:", f_statistic)
    print("Valor p:", p_value)
```

```
Variable numérica: D_CLIENTE
Estadístico F: 20.1554955393527
Valor p: 3.922824732775374e-298
Variable numérica: ESTADO
Estadístico F: 110.05706441919925
Valor p: 0.0
Variable numérica: TIPOTRANSPORTE
Estadístico F: 102.98030148612476
Valor p: 0.0
Variable numérica: PESO_NETO
Estadístico F: 47.86030348734494
Valor p: 0.0
Variable numérica: CIUDAD
Estadístico F: 30.093453339297202
Valor p: 0.0
Variable numérica: PLANTAORIGEN_Pesqueria
Estadístico F: 140.25436370915446
Valor p: 0.0
Variable numérica: TIPO_SERVICIO_LO
Estadístico F: 453.1968709758653
Valor p: 0.0
```

Selección de variables de interés y modelos de Aprendizaje

[ ] ↳ *13 celdas ocultas*

Balanceo de datos

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from numpy import std
from numpy import mean
from scipy import stats
from sklearn import datasets
from sklearn import metrics
from scipy.stats import chi2_contingency
from scipy.stats import contingency
from imblearn.over_sampling import SMOTE
from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from matplotlib import pyplot
from collections import Counter
from pandas import read_csv
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix

df= pd.read_csv('https://raw.githubusercontent.com/kevingonzal/cienciaDatos/main/df_listo_mod
df = df.loc[:, [ "PLANTAORIGEN_Pesqueria", "D_CLIENTE",  "PESO_NETO","ESTADO","CIUDAD", "TIPO
le = LabelEncoder()
df['D_EMPRESA_TRANSPORTISTA'] = le.fit_transform(df['D_EMPRESA_TRANSPORTISTA'])
df.rename(columns={"D_EMPRESA_TRANSPORTISTA": "target"}, inplace=True)
df.to_csv('df_modelos.csv', index=False)

df
```

	PLANTAORIGEN_Pesqueria	D_CLIENTE	PESO_NETO	ESTADO	CIUDAD	TIPOTRANSPORTE	tar
0	1	187	30670	1	39	9	
1	1	107	36890	8	35	5	
2	1	145	23300	14	61	6	
3	1	147	20480	22	1	5	
4	1	99	36980	1	39	5	
...	...	...	...	...	...	...	

```
df.info()
```

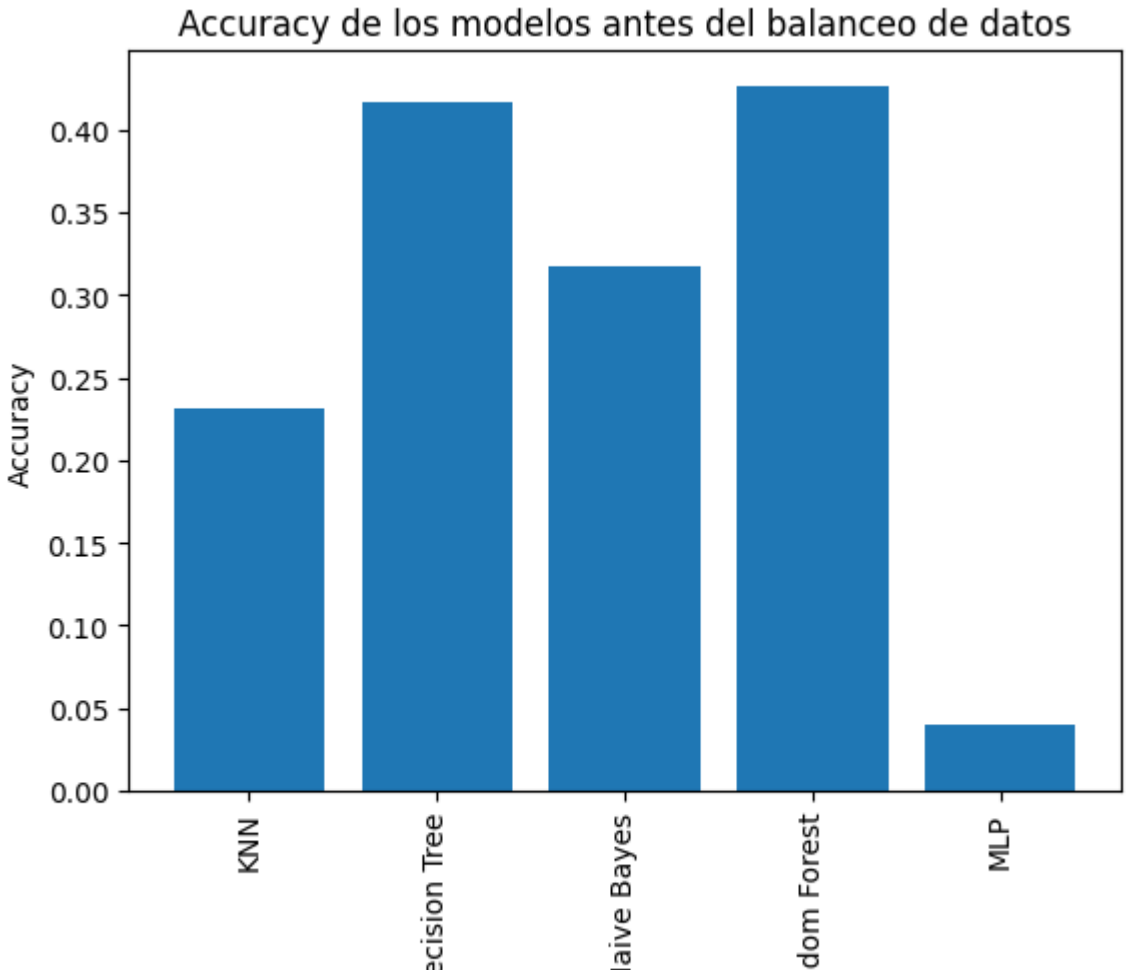
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14408 entries, 0 to 14407
Data columns (total 7 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   PLANTAORIGEN_Pesqueria  14408 non-null  int64
 1   D_CLIENTE               14408 non-null  int64
 2   PESO_NETO               14408 non-null  int64
 3   ESTADO                  14408 non-null  int64
 4   CIUDAD                  14408 non-null  int64
 5   TIPOTRANSPORTE          14408 non-null  int64
 6   target                  14408 non-null  int64
dtypes: int64(7)
memory usage: 788.1 KB
```

```
frecuencias = df["target"].value_counts()

print(frecuencias)
```

```
54    1666
40    1462
25    1242
45    1069
50    1021
      ...
59       6
46       6
74       6
10       5
42       5
Name: target, Length: 76, dtype: int64
```

```
# Importar las bibliotecas necesarias
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier
```

```
from sklearn.neural_network import MLPClassifier
import pandas as pd
import matplotlib.pyplot as plt


# Separar las características (features) y etiquetas (labels)
X = df.iloc[:, :-1].values
y = df.iloc[:, -1].values

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

# Crear los modelos y entrenarlos con los datos de entrenamiento
knn = KNeighborsClassifier()
knn.fit(X_train, y_train)

dt = DecisionTreeClassifier()
dt.fit(X_train, y_train)

nb = GaussianNB()
nb.fit(X_train, y_train)

rf = RandomForestClassifier()
rf.fit(X_train, y_train)

mlp = MLPClassifier()
mlp.fit(X_train, y_train)

# Evaluar el rendimiento de los modelos en los datos de prueba
acc_knn = knn.score(X_test, y_test)
acc_dt = dt.score(X_test, y_test)
acc_nb = nb.score(X_test, y_test)
acc_rf = rf.score(X_test, y_test)
acc_mlp = mlp.score(X_test, y_test)

# Crear una gráfica que muestre los valores de accuracy de cada modelo
models = ['KNN', 'Decision Tree', 'Naive Bayes', 'Random Forest', 'MLP']
accuracies_antes = [acc_knn, acc_dt, acc_nb, acc_rf, acc_mlp]

plt.bar(models, accuracies_antes)
plt.title('Accuracy de los modelos antes del balanceo de datos')
plt.xlabel('Modelos')
plt.ylabel('Accuracy')
plt.xticks(rotation=90)

plt.show()
```




```
data = df.values
# split into input and output elements
X, y = data[:, :-1], data[:, -1]
# label encode the target variable
y = LabelEncoder().fit_transform(y)
# summarize distribution
counter = Counter(y)
for k,v in counter.items():
 per = v / len(y) * 100
 print('Clase=%d, n=%d (%.3f%%)' % (k, v, per))
# plot the distribution

pyplot.bar(counter.keys(), counter.values())
pyplot.title('Datos no balanceados')
pyplot.xlabel('Empresa transportista')
pyplot.ylabel('Frecuencia')


pyplot.show()
```

```
Clase=20, n=108 (0.750%)
Clase=50, n=1021 (7.086%)
Clase=39, n=426 (2.957%)
Clase=15, n=10 (0.069%)
Clase=40, n=1462 (10.147%)
Clase=41, n=427 (2.964%)
Clase=56, n=80 (0.555%)
Clase=54, n=1666 (11.563%)
Clase=33, n=214 (1.485%)
Clase=52, n=280 (1.943%)
Clase=59, n=6 (0.042%)
Clase=60, n=59 (0.409%)
Clase=4, n=13 (0.090%)
Clase=45, n=1069 (7.419%)
Clase=9, n=255 (1.770%)
Clase=25, n=1242 (8.620%)
Clase=12, n=344 (2.388%)
Clase=26, n=9 (0.062%)
Clase=61, n=64 (0.444%)
Clase=48, n=199 (1.381%)
Clase=57, n=820 (5.691%)
Clase=7, n=566 (3.928%)
Clase=36, n=58 (0.403%)
Clase=44, n=393 (2.728%)
Clase=62, n=27 (0.187%)
Clase=63, n=68 (0.472%)
Clase=13, n=49 (0.340%)
Clase=66, n=98 (0.680%)
Clase=64, n=23 (0.160%)
Clase=65, n=19 (0.132%)
Clase=67, n=9 (0.062%)
Clase=68, n=30 (0.208%)
Clase=49, n=197 (1.367%)
Clase=43, n=98 (0.680%)
Clase=29, n=149 (1.034%)
Clase=69, n=56 (0.389%)
Clase=70, n=12 (0.083%)
Clase=75, n=27 (0.187%)
Clase=30, n=19 (0.132%)
Clase=32, n=358 (2.485%)
Clase=71, n=12 (0.083%)
Clase=58, n=174 (1.208%)
Clase=34, n=110 (0.763%)
Clase=72, n=14 (0.097%)
Clase=19, n=46 (0.319%)
Clase=73, n=8 (0.056%)
Clase=37, n=34 (0.236%)
Clase=23, n=43 (0.298%)
Clase=74, n=6 (0.042%)
Clase=16, n=21 (0.146%)
Clase=6, n=56 (0.389%)
Clase=31, n=209 (1.451%)
Clase=18, n=128 (0.888%)
Clase=35, n=332 (2.304%)
Clase=3, n=53 (0.368%)
Clase=5, n=161 (1.117%)
```

```
Clase=17, n=65 (0.451%)
```

```
data = df.values
# split into input and output elements
X, y = data[1:, :-1], data[1:, -1]
# label encode the target variable
y = LabelEncoder().fit_transform(y)
# transform the dataset
oversample = SMOTE(k_neighbors=4)
X, y = oversample.fit_resample(X, y)
# summarize distribution
counter = Counter(y)
for k,v in counter.items():
 per = v / len(y) * 100
 print('Class=%d, n=%d (%.3f%%)' % (k, v, per))
# plot the distribution
pyplot.bar(counter.keys(), counter.values())
pyplot.title('Datos balanceados')
pyplot.xlabel('Empresa transportista')
pyplot.ylabel('Frecuencia')
pyplot.show()
```

```
Class=50, n=1666 (1.316%)
Class=39, n=1666 (1.316%)
Class=15, n=1666 (1.316%)
Class=40, n=1666 (1.316%)
Class=41, n=1666 (1.316%)
Class=56, n=1666 (1.316%)
Class=54, n=1666 (1.316%)
Class=33, n=1666 (1.316%)
Class=52, n=1666 (1.316%)
Class=59, n=1666 (1.316%)
Class=60, n=1666 (1.316%)
Class=4, n=1666 (1.316%)
Class=45, n=1666 (1.316%)
Class=9, n=1666 (1.316%)
Class=25, n=1666 (1.316%)
Class=12, n=1666 (1.316%)
Class=26, n=1666 (1.316%)
Class=61, n=1666 (1.316%)
Class=48, n=1666 (1.316%)
Class=57, n=1666 (1.316%)
Class=7, n=1666 (1.316%)
Class=36, n=1666 (1.316%)
Class=44, n=1666 (1.316%)
Class=62, n=1666 (1.316%)
Class=63, n=1666 (1.316%)
Class=13, n=1666 (1.316%)
Class=66, n=1666 (1.316%)
Class=64, n=1666 (1.316%)
Class=65, n=1666 (1.316%)
Class=67, n=1666 (1.316%)
Class=68, n=1666 (1.316%)
Class=49, n=1666 (1.316%)
Class=43, n=1666 (1.316%)
Class=29, n=1666 (1.316%)
Class=69, n=1666 (1.316%)
Class=70, n=1666 (1.316%)
Class=75, n=1666 (1.316%)
Class=30, n=1666 (1.316%)
Class=32, n=1666 (1.316%)
Class=71, n=1666 (1.316%)
Class=20, n=1666 (1.316%)
Class=58, n=1666 (1.316%)
Class=34, n=1666 (1.316%)
Class=72, n=1666 (1.316%)
Class=19, n=1666 (1.316%)
Class=73, n=1666 (1.316%)
Class=37, n=1666 (1.316%)
Class=23, n=1666 (1.316%)
Class=74, n=1666 (1.316%)
Class=16, n=1666 (1.316%)
Class=6, n=1666 (1.316%)
Class=31, n=1666 (1.316%)
Class=18, n=1666 (1.316%)
Class=35, n=1666 (1.316%)
Class=3, n=1666 (1.316%)
Class=5, n=1666 (1.316%)
```

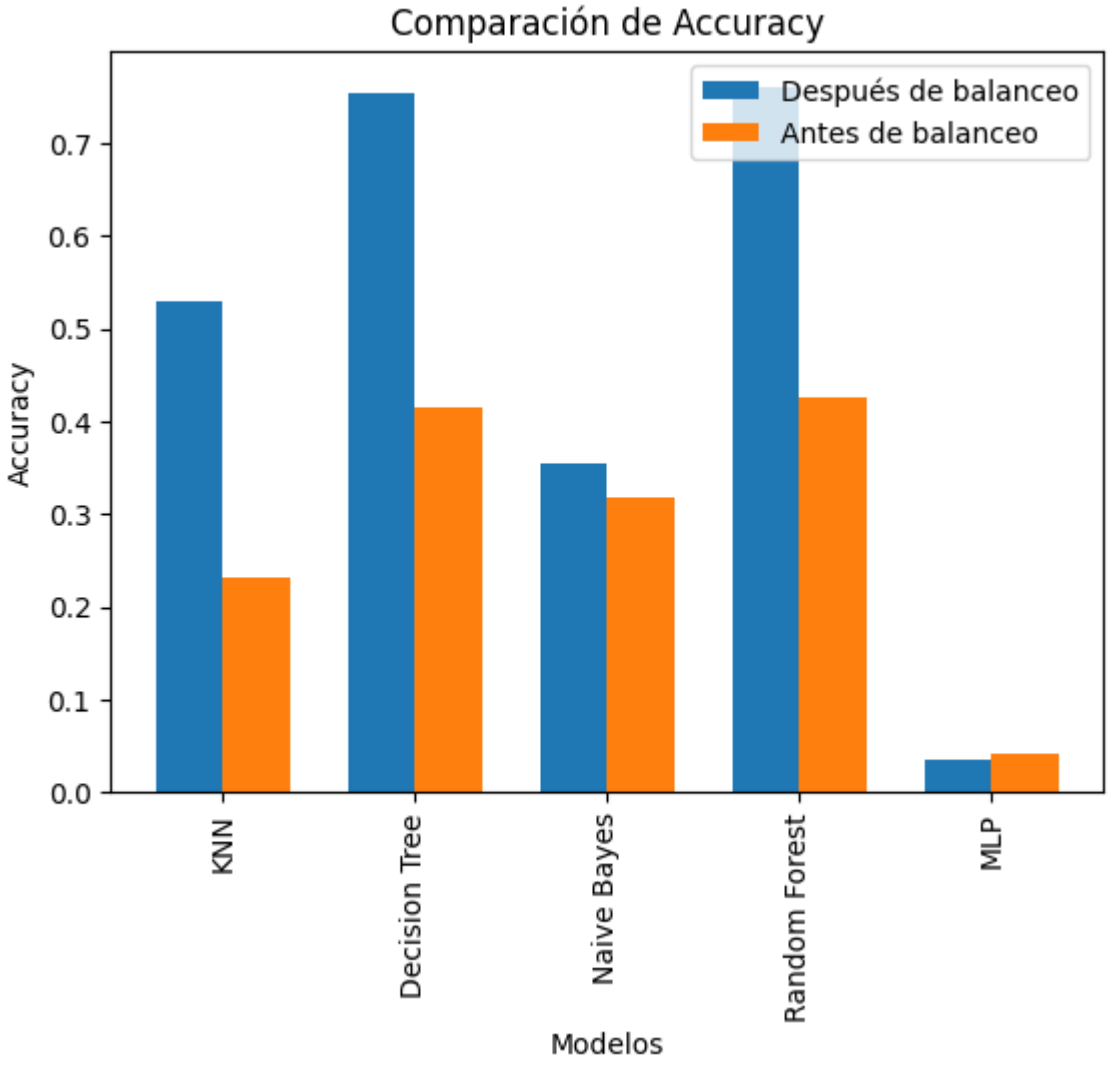
```
Class=17, n=1666 (1.316%)


Class=47, n=1666 (1.316%)
```

Modelos con datos balanceados

```
Class=51, n=1666 (1.316%)
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
# Crear los modelos y entrenarlos con los datos de entrenamiento
knn = KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski', p = 2)
knn.fit(X_train, y_train)

dt = DecisionTreeClassifier(criterion = 'entropy')
dt.fit(X_train, y_train)

nb = GaussianNB()
nb.fit(X_train, y_train)

rf = RandomForestClassifier(n_estimators = 100, criterion = 'entropy')
rf.fit(X_train, y_train)

mlp = MLPClassifier(hidden_layer_sizes=(8,8,8), activation='relu', solver='adam', max_iter=50
mlp.fit(X_train, y_train)

# Evaluar el rendimiento de los modelos en los datos de prueba
acc_knn = knn.score(X_test, y_test)
acc_dt = dt.score(X_test, y_test)
acc_nb = nb.score(X_test, y_test)
acc_rf = rf.score(X_test, y_test)
acc_mlp = mlp.score(X_test, y_test)

# Crear una gráfica que muestre los valores de accuracy de cada modelo
models = ['KNN', 'Decision Tree', 'Naive Bayes', 'Random Forest', 'MLP']
accuracies = [acc_knn, acc_dt, acc_nb, acc_rf, acc_mlp]

# Definir la anchura de las barras y la posición en el eje X para cada grupo
bar_width = 0.35
x_pos = np.arange(len(models))

# Crear una figura y un eje
fig, ax = plt.subplots()

# Dibujar las barras
ax.bar(x_pos, accuracies, width=bar_width, label='Después de balanceo')
ax.bar(x_pos + bar_width, accuracies_antes, width=bar_width, label='Antes de balanceo')

# Agregar etiquetas de los ejes, título y leyenda
ax.set_xlabel('Modelos')
ax.set_ylabel('Accuracy')
ax.set_title('Comparación de Accuracy')
```

```
ax.set_xticks(x_pos + bar_width / 2)
ax.set_xticklabels(models)
ax.legend()
plt.xticks(rotation=90)
plt.show()
```

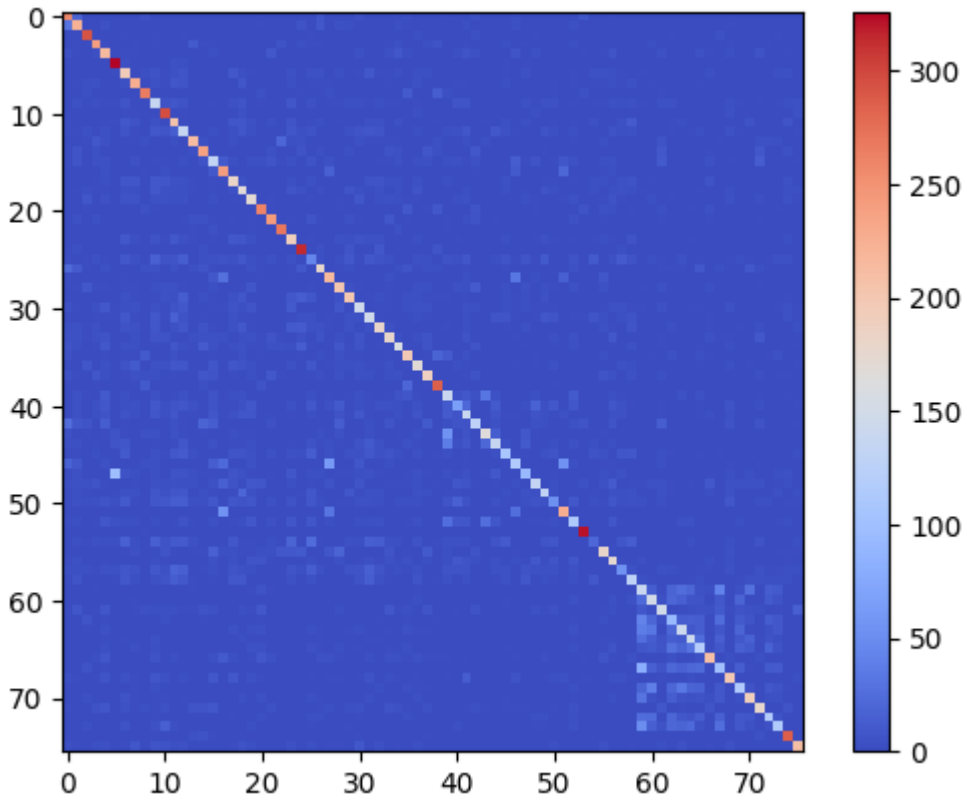


Gráficas de matriz de confusión

```
#K VECINOS MÁS CERCANOS

#Defino el algoritmo a utilizar
from sklearn.neighbors import KNeighborsClassifier
algoritmo = KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski', p = 2)
#Entreno el modelo
algoritmo.fit(X_train, y_train)
#Realizo una predicción
y_pred = algoritmo.predict(X_test)

#Calculo la exactitud del modelo
```

```
accuracy = metrics.accuracy_score(y_test, y_pred)
print('Exactitud del modelo K-NN:')
print(accuracy)
print('\n\nMatriz de confusión: ')

matriz=metrics.confusion_matrix(y_test,y_pred)
print(matriz)


heatmap = plt.imshow(matriz, cmap='coolwarm')

# Añadimos una barra de colores para indicar los valores de la matriz
plt.colorbar(heatmap)

# Mostramos el mapa de calor
plt.show()
```

```
Exactitud del modelo K-NN:
0.5291818038224609


Matriz de confusión:
[[256  18   1 ...   0   0   1]
 [ 30 223   0 ...   0   0   0]
 [  0   0 292 ...   2   0   1]
 ...
 [  0   0   3 ... 114   0   0]
 [  0   0   0 ...   0 286   0]
 [  3   2   0 ...   1   0 209]]
```



```
#Naive Bayes
from sklearn.naive_bayes import GaussianNB
algoritmo = GaussianNB()
#Entreno el modelo
algoritmo.fit(X_train, y_train)
#Realizo predicción
y_pred = algoritmo.predict(X_test)

#Calculo la exactitud del modelo
accuracy2 = metrics.accuracy_score(y_test, y_pred)
print('Exactitud del modelo Naive Bayes:')
print(accuracy2)
print('\n\nMatriz de confusión: ')

matriz=metrics.confusion_matrix(y_test,y_pred)
print(matriz)


heatmap = plt.imshow(matriz, cmap='coolwarm')

# Añadimos una barra de colores para indicar los valores de la matriz
plt.colorbar(heatmap)

# Mostramos el mapa de calor
plt.show()
```

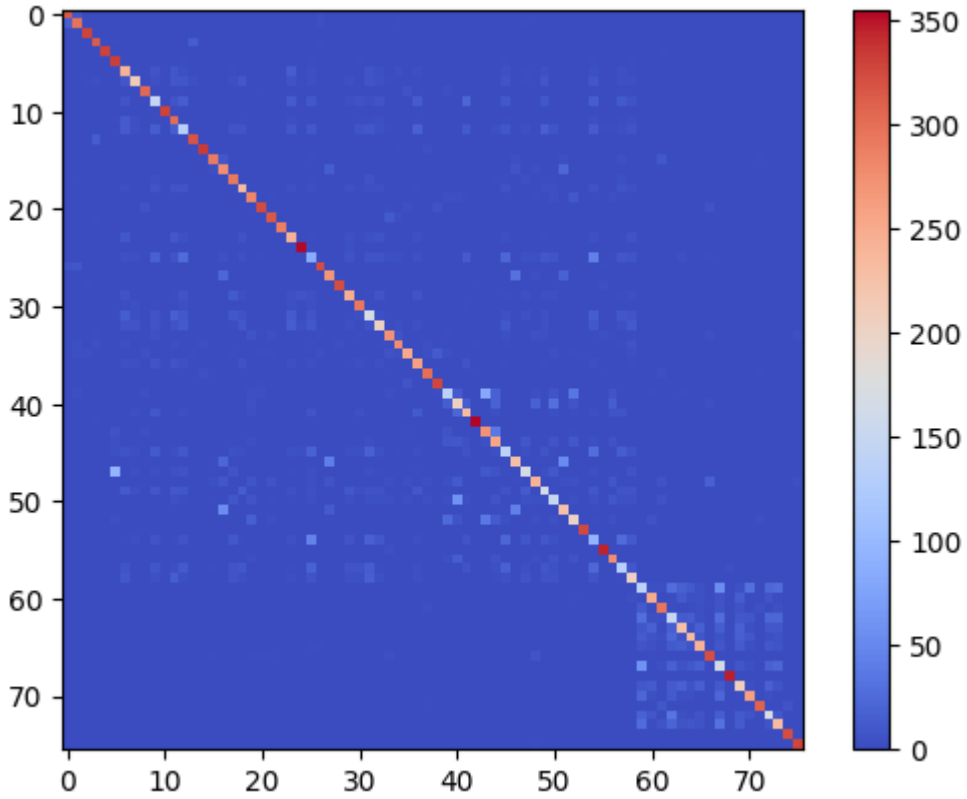
```
Exactitud del modelo Naive Bayes:
0.3536171220976149


Matriz de confusión:
[[325   0   0 ...   0   0   0]
 [185  79   0 ...   0   0   0]
 [  0   0 251 ...   0   0   0]
 ...
```

```
#Arboles de decisión
from sklearn.tree import DecisionTreeClassifier
algoritmo = DecisionTreeClassifier(criterion = 'entropy')
#Entreno el modelo
algoritmo.fit(X_train, y_train)
#Realizo una predicción
y_pred = algoritmo.predict(X_test)
#Calculo la exactitud del modelo
accuracy = metrics.accuracy_score(y_test, y_pred)
print('Exactitud del modelo árboles de decisión:')
print(accuracy)

print('\n\nMatriz de confusión: ')
print(metrics.confusion_matrix(y_test,y_pred))

matriz=metrics.confusion_matrix(y_test,y_pred)
print(matriz)


heatmap = plt.imshow(matriz, cmap='coolwarm')

# Añadimos una barra de colores para indicar los valores de la matriz
plt.colorbar(heatmap)

# Mostramos el mapa de calor
plt.show()
```

```
Exactitud del modelo árboles de decisión:
0.7533564997630706


Matriz de confusión:
[[311  14   0 ...   0   0   0]
 [ 19 289   0 ...   0   0   0]
 [  0   0 327 ...   0   0   0]
 ...
 [  0   0   0 ... 230   0   0]
 [  0   0   0 ...   0 323   0]
 [  0   0   0 ...   0   0 316]]
[[311  14   0 ...   0   0   0]
 [ 19 289   0 ...   0   0   0]
 [  0   0 327 ...   0   0   0]
 ...
 [  0   0   0 ... 230   0   0]
 [  0   0   0 ...   0 323   0]
 [  0   0   0 ...   0   0 316]]
```


```
#Bosques Aleatorios
from sklearn.ensemble import RandomForestClassifier
algoritmo = RandomForestClassifier(n_estimators = 100, criterion = 'entropy')
#Entreno el modelo
algoritmo.fit(X_train, y_train)
#Realizo una predicción
y_pred = algoritmo.predict(X_test)
#Calculo la exactitud del modelo
accuracy = metrics.accuracy_score(y_test, y_pred)
print('Exactitud del modelo Bosques Aleatorios:')
print(accuracy)

print('\n\nMatriz de confusión: ')

matriz=metrics.confusion_matrix(y_test,y_pred)
print(matriz)


heatmap = plt.imshow(matriz, cmap='coolwarm')

# Añadimos una barra de colores para indicar los valores de la matriz
plt.colorbar(heatmap)

# Mostramos el mapa de calor
plt.show()
```

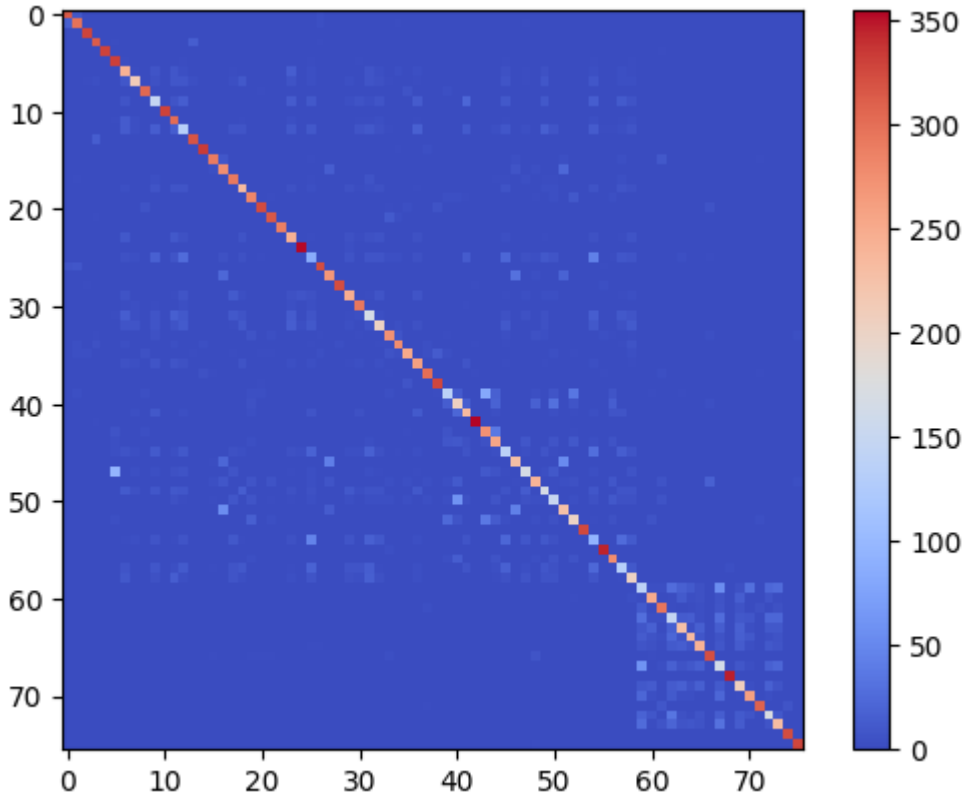
```
Exactitud del modelo Bosques Aleatorios:
0.761135681566893


Matriz de confusión:
[[314  11   0 ...   0   0   0]
 [ 16 297   0 ...   0   0   0]
 [  0   0 328 ...   0   0   0]
 ...
 [  0   0   0 ... 231   0   0]
 [  0   0   0 ...   0 323   0]
 [  0   0   0 ...   0   0 326]]
```




```
#Redes neuronales
from sklearn.neural_network import MLPClassifier

mlp = MLPClassifier(hidden_layer_sizes=(8,8,8), activation='relu', solver='adam', max_iter=50

mlp.fit(X_train,y_train.ravel())

predict_train = mlp.predict(X_train)
predict_test = mlp.predict(X_test)

accuracy = metrics.accuracy_score(y_test, y_pred)
print('Exactitud del modelo Redes neuronales:')
print(accuracy)
print('\n\nMatriz de confusión: ')

matriz=metrics.confusion_matrix(y_test,y_pred)
print(matriz)
```

```
heatmap = plt.imshow(matriz, cmap='coolwarm')

# Añadimos una barra de colores para indicar los valores de la matriz
plt.colorbar(heatmap)

# Mostramos el mapa de calor
plt.show()
```

```
Exactitud del modelo Redes neuronales:
0.761135681566893


Matriz de confusión:
[[314  11   0 ...   0   0   0]
 [ 16 297   0 ...   0   0   0]
 [  0   0 328 ...   0   0   0]
 ...
 [  0   0   0 ... 231   0   0]
 [  0   0   0 ...   0 323   0]
 [  0   0   0 ...   0   0 326]]
```



Accuracy modelos con tres variables

```
#Bosques Aleatorios
modelo =  RandomForestClassifier(n_estimators = 100, criterion = 'entropy')
```

```
modelo.fit(X_train, y_train)

# Realizar predicciones para las instancias de prueba
y_pred_probs = modelo.predict_proba(X_test)  # Probabilidades de predicción para cada clase
y_pred_classes = modelo.classes_[y_pred_probs.argsort(axis=1)[:, -3:]] #Obtener las 3 clases


# Calcular el accuracy global para verificar si alguna de las 3 opciones coincide con la clas
aciertos = 0

# Imprimir las 3 mejores opciones de predicción para cada instancia de prueba
for i in range(len(y_pred_classes)):
    print (f'Instancia {i+1}: {y_pred_classes[i]} - Resultado Real:{y_test[i]}')

for i in range(len(y_pred_classes)):
    if y_test[i] in y_pred_classes[i]:
        aciertos += 1

#Evaluar el modelo
accuracy = aciertos / len(y_pred_classes)
m4=accuracy
print(f'Accuracy global: {accuracy:.2f}')
```

Se truncaron las últimas líneas 5000 del resultado de transmisión.

```
Instancia 20326: [48 40 56] - Resultado Real:56
Instancia 20327: [54 36 57] - Resultado Real:57
Instancia 20328: [47 17 29] - Resultado Real:29
Instancia 20329: [27 75 41] - Resultado Real:41
Instancia 20330: [ 9 23  6] - Resultado Real:23
Instancia 20331: [34 33 19] - Resultado Real:19
Instancia 20332: [25 52 71] - Resultado Real:71
Instancia 20333: [27 25 36] - Resultado Real:36
Instancia 20334: [75 63 64] - Resultado Real:64
Instancia 20335: [32 25 36] - Resultado Real:36
Instancia 20336: [44 47  5] - Resultado Real:5
Instancia 20337: [27 37 14] - Resultado Real:14
Instancia 20338: [69 65 63] - Resultado Real:62
Instancia 20339: [55 28  2] - Resultado Real:2
Instancia 20340: [27 75 68] - Resultado Real:68
Instancia 20341: [26 34  1] - Resultado Real:1
Instancia 20342: [ 9  6 11] - Resultado Real:11
Instancia 20343: [43 47  5] - Resultado Real:5
Instancia 20344: [27 54  2] - Resultado Real:2
Instancia 20345: [20 27 46] - Resultado Real:46
Instancia 20346: [40 56 19] - Resultado Real:56
Instancia 20347: [ 9 25 57] - Resultado Real:9
Instancia 20348: [16 46 51] - Resultado Real:51
Instancia 20349: [12  9 54] - Resultado Real:12
Instancia 20350: [27 75 39] - Resultado Real:39
Instancia 20351: [27 75 10] - Resultado Real:10
Instancia 20352: [26 75 58] - Resultado Real:58
Instancia 20353: [28 55  2] - Resultado Real:2
Instancia 20354: [75 52 63] - Resultado Real:60
Instancia 20355: [23 45 58] - Resultado Real:23
```

```
Instancia 20356: [32 29 24] - Resultado Real:24
Instancia 20357: [20 27  0] - Resultado Real:0
Instancia 20358: [72 65 60] - Resultado Real:60
Instancia 20359: [75 32 24] - Resultado Real:24
Instancia 20360: [20 27 75] - Resultado Real:75
Instancia 20361: [19 44 41] - Resultado Real:19
Instancia 20362: [45 25 57] - Resultado Real:57
Instancia 20363: [36 49  6] - Resultado Real:6
Instancia 20364: [12  6 58] - Resultado Real:54
Instancia 20365: [27 75 35] - Resultado Real:20
Instancia 20366: [58  9 17] - Resultado Real:17
Instancia 20367: [ 3 34 21] - Resultado Real:21
Instancia 20368: [73 64 69] - Resultado Real:73
Instancia 20369: [75 29 24] - Resultado Real:24
Instancia 20370: [37 26 14] - Resultado Real:14
Instancia 20371: [57 18 15] - Resultado Real:15
Instancia 20372: [27 75 22] - Resultado Real:22
Instancia 20373: [ 7  9 23] - Resultado Real:9
Instancia 20374: [57 45  9] - Resultado Real:9
Instancia 20375: [44 40 52] - Resultado Real:52
Instancia 20376: [66 35 61] - Resultado Real:66
Instancia 20377: [ 6 23  7] - Resultado Real:7
Instancia 20378: [16 48 27] - Resultado Real:27
Instancia 20379: [27 75 42] - Resultado Real:42
Instancia 20380: [56 52 39] - Resultado Real:52
Instancia 20381: [26 75 61] - Resultado Real:61
Instancia 20382: [28 75  7] - Resultado Real:7
```

```
#Redes neuronales

modelo =  MLPClassifier(hidden_layer_sizes=(8,8,8), activation='relu', solver='adam', max_ite
modelo.fit(X_train, y_train)

# Realizar predicciones para las instancias de prueba
y_pred_probs = modelo.predict_proba(X_test)  # Probabilidades de predicción para cada clase
y_pred_classes = modelo.classes_[y_pred_probs.argsort(axis=1)[:, -3:]]  # Obtener las 3 clase

# Calcular el accuracy global para verificar si alguna de las 3 opciones coincide con la clas
aciertos = 0

# Imprimir las 3 mejores opciones de predicción para cada instancia de prueba
for i in range(len(y_pred_classes)):
    print (f'Instancia {i+1}: {y_pred_classes[i]} - Resultado Real:{y_test[i]}')

for i in range(len(y_pred_classes)):
    if y_test[i] in y_pred_classes[i]:
        aciertos += 1

accuracy = aciertos / len(y_pred_classes)
m5=accuracy
print(f'Accuracy global: {accuracy:.2f}')
```

```
Se truncaron las últimas líneas 5000 del resultado de transmisión.
Instancia 20326: [68 34 16] - Resultado Real:56
```

```
Instancia 20327: [67 69 60] - Resultado Real:57
Instancia 20328: [29 53 24] - Resultado Real:29
Instancia 20329: [68 34 16] - Resultado Real:41
Instancia 20330: [72 69 16] - Resultado Real:23
Instancia 20331: [69 68 16] - Resultado Real:19
Instancia 20332: [59 67 70] - Resultado Real:71
Instancia 20333: [68 14 34] - Resultado Real:36
Instancia 20334: [72 60 69] - Resultado Real:64
Instancia 20335: [68 16 34] - Resultado Real:36
Instancia 20336: [44 47  5] - Resultado Real:5
Instancia 20337: [ 2 53 29] - Resultado Real:14
Instancia 20338: [60 72 69] - Resultado Real:62
Instancia 20339: [29 53 24] - Resultado Real:2
Instancia 20340: [68 34 16] - Resultado Real:68
Instancia 20341: [32  2 29] - Resultado Real:1
Instancia 20342: [32  2 29] - Resultado Real:11
Instancia 20343: [44 47  5] - Resultado Real:5
Instancia 20344: [ 2 32 29] - Resultado Real:2
Instancia 20345: [68 34 14] - Resultado Real:46
Instancia 20346: [34 68 16] - Resultado Real:56
Instancia 20347: [72 69 16] - Resultado Real:9
Instancia 20348: [60 72 69] - Resultado Real:51
Instancia 20349: [68 16 34] - Resultado Real:12
Instancia 20350: [29 53 24] - Resultado Real:39
Instancia 20351: [32  2 29] - Resultado Real:10
Instancia 20352: [29 53 24] - Resultado Real:58
Instancia 20353: [29 53 24] - Resultado Real:2
Instancia 20354: [60 72 69] - Resultado Real:60
Instancia 20355: [72 69 16] - Resultado Real:23
Instancia 20356: [29 53 24] - Resultado Real:24
Instancia 20357: [29 53 24] - Resultado Real:0
Instancia 20358: [60 72 69] - Resultado Real:60
Instancia 20359: [29 53 24] - Resultado Real:24
Instancia 20360: [29 53 24] - Resultado Real:75
Instancia 20361: [34 68 16] - Resultado Real:19
Instancia 20362: [32 34 14] - Resultado Real:57
Instancia 20363: [72 69 16] - Resultado Real:6
Instancia 20364: [60 72 69] - Resultado Real:54
Instancia 20365: [29 53 24] - Resultado Real:20
Instancia 20366: [68 34 16] - Resultado Real:17
Instancia 20367: [72 60 69] - Resultado Real:21
Instancia 20368: [59 67 70] - Resultado Real:73
Instancia 20369: [29 53 24] - Resultado Real:24
Instancia 20370: [ 2 32 29] - Resultado Real:14
Instancia 20371: [29 53 24] - Resultado Real:15
Instancia 20372: [32  2 29] - Resultado Real:22
Instancia 20373: [29 53 24] - Resultado Real:9
Instancia 20374: [68 14 34] - Resultado Real:9
Instancia 20375: [60 70 67] - Resultado Real:52
Instancia 20376: [ 2 32 29] - Resultado Real:66
Instancia 20377: [16 68 34] - Resultado Real:7
Instancia 20378: [60 72 69] - Resultado Real:27
Instancia 20379: [69 68 16] - Resultado Real:42
Instancia 20380: [34 68 16] - Resultado Real:52
Instancia 20381: [29 53 24] - Resultado Real:61
Instancia 20382: [32  2 29] - Resultado Real:7
```

```
#Arboles de decisión


modelo =  DecisionTreeClassifier(criterion = 'entropy')
modelo.fit(X_train, y_train)

# Realizar predicciones para las instancias de prueba
y_pred_probs = modelo.predict_proba(X_test)  # Probabilidades de predicción para cada clase
y_pred_classes = modelo.classes_[y_pred_probs.argsort(axis=1)[:, -3:]]  # Obtener las 3 clase

# Calcular el accuracy global para verificar si alguna de las 3 opciones coincide con la clas
#y_test_array = y_test.values  # Convertir la serie de pandas a un array numpy
aciertos = 0

# Imprimir las 3 mejores opciones de predicción para cada instancia de prueba
for i in range(len(y_pred_classes)):
    print (f'Instancia {i+1}: {y_pred_classes[i]} - Resultado Real:{y_test[i]}')
for i in range(len(y_pred_classes)):
    if y_test[i] in y_pred_classes[i]:
        aciertos += 1

accuracy = aciertos / len(y_pred_classes)
m2=accuracy
print(f'Accuracy global: {accuracy:.2f}')
```

```
Se truncaron las últimas líneas 5000 del resultado de transmisión.
Instancia 20326: [19 26 56] - Resultado Real:56
Instancia 20327: [19 26 57] - Resultado Real:57
Instancia 20328: [27 75 29] - Resultado Real:29
Instancia 20329: [27 75 41] - Resultado Real:41
Instancia 20330: [21 75  9] - Resultado Real:23
Instancia 20331: [20 27 19] - Resultado Real:19
Instancia 20332: [27 75 71] - Resultado Real:71
Instancia 20333: [20 27 36] - Resultado Real:36
Instancia 20334: [26 75 64] - Resultado Real:64
Instancia 20335: [20 27 36] - Resultado Real:36
Instancia 20336: [44 47  5] - Resultado Real:5
Instancia 20337: [27 75 14] - Resultado Real:14
Instancia 20338: [26 75 63] - Resultado Real:62
Instancia 20339: [28 75  2] - Resultado Real:2
Instancia 20340: [27 75 68] - Resultado Real:68
Instancia 20341: [28 75  1] - Resultado Real:1
Instancia 20342: [27 75 11] - Resultado Real:11
Instancia 20343: [43 47  5] - Resultado Real:5
Instancia 20344: [28 75  2] - Resultado Real:2
Instancia 20345: [20 27 46] - Resultado Real:46
Instancia 20346: [19 26 56] - Resultado Real:56
Instancia 20347: [27 75 30] - Resultado Real:9
Instancia 20348: [16 46 51] - Resultado Real:51
Instancia 20349: [20  9 54] - Resultado Real:12
Instancia 20350: [27 75 39] - Resultado Real:39
Instancia 20351: [27 75 10] - Resultado Real:10
```

```
Instancia 20352: [26 75 58] - Resultado Real:58
Instancia 20353: [28 75  2] - Resultado Real:2
Instancia 20354: [26 75 63] - Resultado Real:60
Instancia 20355: [75 45 58] - Resultado Real:23
Instancia 20356: [27 75 24] - Resultado Real:24
Instancia 20357: [20 27  0] - Resultado Real:0
Instancia 20358: [18 60 65] - Resultado Real:60
Instancia 20359: [27 75 24] - Resultado Real:24
Instancia 20360: [20 27 75] - Resultado Real:75
Instancia 20361: [27 75 41] - Resultado Real:19
Instancia 20362: [19 26 57] - Resultado Real:57
Instancia 20363: [28 75  6] - Resultado Real:6
Instancia 20364: [26 75 58] - Resultado Real:54
Instancia 20365: [27 75 35] - Resultado Real:20
Instancia 20366: [20 27 17] - Resultado Real:17
Instancia 20367: [27 75 21] - Resultado Real:21
Instancia 20368: [27 75 69] - Resultado Real:73
Instancia 20369: [27 75 24] - Resultado Real:24
Instancia 20370: [27 75 14] - Resultado Real:14
Instancia 20371: [27 75 15] - Resultado Real:15
Instancia 20372: [27 75 22] - Resultado Real:22
Instancia 20373: [28 75  7] - Resultado Real:9
Instancia 20374: [21 75  9] - Resultado Real:9
Instancia 20375: [26 75 52] - Resultado Real:52
Instancia 20376: [26 75 61] - Resultado Real:66
Instancia 20377: [28 75  7] - Resultado Real:7
Instancia 20378: [20 75 27] - Resultado Real:27
Instancia 20379: [27 75 42] - Resultado Real:42
Instancia 20380: [27 75 39] - Resultado Real:52
Instancia 20381: [26 75 61] - Resultado Real:61
Instancia 20382: [28 75  7] - Resultado Real:7
```

```
#Naive Bayes

modelo = GaussianNB()

modelo.fit(X_train, y_train)

# Realizar predicciones para las instancias de prueba
y_pred_probs = modelo.predict_proba(X_test)  # Probabilidades de predicción para cada clase
y_pred_classes = modelo.classes_[y_pred_probs.argsort(axis=1)[:, -3:]]  # Obtener las 3 clase

# Calcular el accuracy global para verificar si alguna de las 3 opciones coincide con la clas
#y_test_array = y_test.values  # Convertir la serie de pandas a un array numpy
aciertos = 0

# Imprimir las 3 mejores opciones de predicción para cada instancia de prueba
for i in range(len(y_pred_classes)):
    print (f'Instancia {i+1}: {y_pred_classes[i]} - Resultado Real:{y_test[i]}')

for i in range(len(y_pred_classes)):
    if y_test[i] in y_pred_classes[i]:
        aciertos += 1
```

```
accuracy = aciertos / len(y_pred_classes)
m3=accuracy
print(f'Accuracy global: {accuracy:.2f}')
```

Se truncaron las últimas líneas 5000 del resultado de transmisión.

```
Instancia 20326: [41 44 40] - Resultado Real:56
Instancia 20327: [30  9 36] - Resultado Real:57
Instancia 20328: [49 45 29] - Resultado Real:29
Instancia 20329: [36  9 41] - Resultado Real:41
Instancia 20330: [11 23  6] - Resultado Real:23
Instancia 20331: [27 34 55] - Resultado Real:19
Instancia 20332: [50 52 71] - Resultado Real:71
Instancia 20333: [ 9 36 41] - Resultado Real:36
Instancia 20334: [63 60 64] - Resultado Real:64
Instancia 20335: [41  9 36] - Resultado Real:36
Instancia 20336: [44 47  5] - Resultado Real:5
Instancia 20337: [26  1  0] - Resultado Real:14
Instancia 20338: [73 65 72] - Resultado Real:62
Instancia 20339: [26  2 28] - Resultado Real:2
Instancia 20340: [57 30 68] - Resultado Real:68
Instancia 20341: [14 26  1] - Resultado Real:1
Instancia 20342: [ 6 23  9] - Resultado Real:11
Instancia 20343: [44  5 47] - Resultado Real:5
Instancia 20344: [23 11  2] - Resultado Real:2
Instancia 20345: [58 46 21] - Resultado Real:46
Instancia 20346: [44 41 40] - Resultado Real:56
Instancia 20347: [11  6 23] - Resultado Real:9
Instancia 20348: [46 16 51] - Resultado Real:51
Instancia 20349: [12 41  9] - Resultado Real:12
Instancia 20350: [35 20  8] - Resultado Real:39
Instancia 20351: [32 29 10] - Resultado Real:10
Instancia 20352: [23 18 29] - Resultado Real:58
Instancia 20353: [55 75 28] - Resultado Real:2
Instancia 20354: [63 60 69] - Resultado Real:60
Instancia 20355: [11 23  6] - Resultado Real:23
Instancia 20356: [32 29 24] - Resultado Real:24
Instancia 20357: [35  1  0] - Resultado Real:0
Instancia 20358: [69 65 72] - Resultado Real:60
Instancia 20359: [32 29 24] - Resultado Real:24
Instancia 20360: [27 46 75] - Resultado Real:75
Instancia 20361: [44 41 40] - Resultado Real:19
Instancia 20362: [29 45 17] - Resultado Real:57
Instancia 20363: [11 23  6] - Resultado Real:6
Instancia 20364: [31  6 23] - Resultado Real:54
Instancia 20365: [35 20 38] - Resultado Real:20
Instancia 20366: [11 23  6] - Resultado Real:17
Instancia 20367: [14 21  3] - Resultado Real:21
Instancia 20368: [73 62 67] - Resultado Real:73
Instancia 20369: [32 29 24] - Resultado Real:24
Instancia 20370: [37 26  1] - Resultado Real:14
Instancia 20371: [45 57 18] - Resultado Real:15
Instancia 20372: [32 43 22] - Resultado Real:22
Instancia 20373: [47 23 11] - Resultado Real:9
```

```
Instancia 20374: [31  6 11] - Resultado Real:9
Instancia 20375: [41 44 40] - Resultado Real:52
Instancia 20376: [37 75 61] - Resultado Real:66
Instancia 20377: [11 23  6] - Resultado Real:7
Instancia 20378: [46 16 51] - Resultado Real:27
Instancia 20379: [56 42 33] - Resultado Real:42
Instancia 20380: [43 52 39] - Resultado Real:52
Instancia 20381: [19 61 75] - Resultado Real:61
```

```
#K VECINOS MÁS CERCANOS
modelo = KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski', p = 2)
modelo.fit(X_train, y_train)

# Realizar predicciones para las instancias de prueba
y_pred_probs = modelo.predict_proba(X_test)  # Probabilidades de predicción para cada clase
y_pred_classes = modelo.classes_[y_pred_probs.argsort(axis=1)[:, -3:]]  # Obtener las 3 clase

# Calcular el accuracy global para verificar si alguna de las 3 opciones coincide con la clas
#y_test_array = y_test.values  # Convertir la serie de pandas a un array numpy
aciertos = 0

# Imprimir las 3 mejores opciones de predicción para cada instancia de prueba
for i in range(len(y_pred_classes)):
    print (f'Instancia {i+1}: {y_pred_classes[i]} - Resultado Real:{y_test[i]}')

for i in range(len(y_pred_classes)):
    if y_test[i] in y_pred_classes[i]:
        aciertos += 1

accuracy = aciertos / len(y_pred_classes)
m1=accuracy
print(f'Accuracy global: {accuracy:.2f}')
```

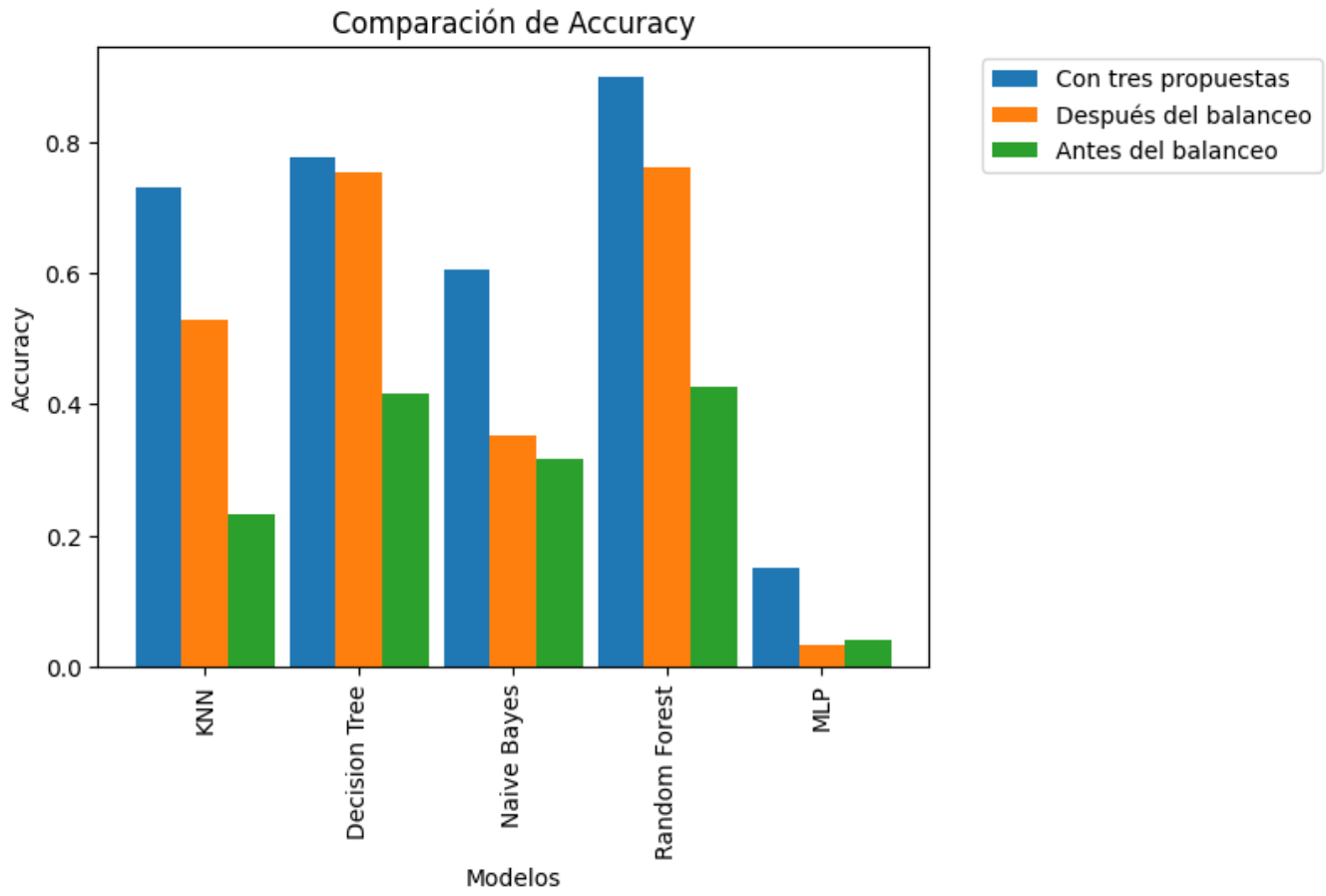
```
Se truncaron las últimas líneas 5000 del resultado de transmisión.
Instancia 20326: [27  4 56] - Resultado Real:56
Instancia 20327: [75 43 36] - Resultado Real:57
Instancia 20328: [60  7 29] - Resultado Real:29
Instancia 20329: [27 75 41] - Resultado Real:41
Instancia 20330: [18  6 23] - Resultado Real:23
Instancia 20331: [20 27 19] - Resultado Real:19
Instancia 20332: [43 39 41] - Resultado Real:71
Instancia 20333: [48 36 40] - Resultado Real:36
Instancia 20334: [60 73 64] - Resultado Real:64
Instancia 20335: [20 27 36] - Resultado Real:36
Instancia 20336: [28 75  5] - Resultado Real:5
Instancia 20337: [19 14 74] - Resultado Real:14
Instancia 20338: [60 63 65] - Resultado Real:62
Instancia 20339: [75 28  2] - Resultado Real:2
Instancia 20340: [15 19 68] - Resultado Real:68
Instancia 20341: [52 46  1] - Resultado Real:1
Instancia 20342: [11 54 68] - Resultado Real:11
Instancia 20343: [75 47  5] - Resultado Real:5
```

```
Instancia 20344: [67 70  2] - Resultado Real:2
Instancia 20345: [ 4 18 15] - Resultado Real:46
Instancia 20346: [44 39 56] - Resultado Real:56
Instancia 20347: [75 45  9] - Resultado Real:9
Instancia 20348: [16 46 51] - Resultado Real:51
Instancia 20349: [54 50  9] - Resultado Real:12
Instancia 20350: [27 75 39] - Resultado Real:39
Instancia 20351: [48 61 71] - Resultado Real:10
Instancia 20352: [26 75 58] - Resultado Real:58
Instancia 20353: [28 75  2] - Resultado Real:2
Instancia 20354: [67 63 69] - Resultado Real:60
Instancia 20355: [11 50 58] - Resultado Real:23
Instancia 20356: [29 32 24] - Resultado Real:24
Instancia 20357: [35  0  8] - Resultado Real:0
Instancia 20358: [18 65 60] - Resultado Real:60
Instancia 20359: [27 75 24] - Resultado Real:24
Instancia 20360: [20 27 75] - Resultado Real:75
Instancia 20361: [44 18 41] - Resultado Real:19
Instancia 20362: [12 50 31] - Resultado Real:57
Instancia 20363: [75 12  6] - Resultado Real:6
Instancia 20364: [75 58  6] - Resultado Real:54
Instancia 20365: [28 20  1] - Resultado Real:20
Instancia 20366: [26 58 17] - Resultado Real:17
Instancia 20367: [75 47 21] - Resultado Real:21
Instancia 20368: [68 69 64] - Resultado Real:73
Instancia 20369: [27 75 24] - Resultado Real:24
Instancia 20370: [75 42 71] - Resultado Real:14
Instancia 20371: [75 15 17] - Resultado Real:15
Instancia 20372: [27 75 22] - Resultado Real:22
Instancia 20373: [ 7 33 20] - Resultado Real:9
Instancia 20374: [45 57  9] - Resultado Real:9
Instancia 20375: [40 17 52] - Resultado Real:52
Instancia 20376: [66  9 53] - Resultado Real:66
Instancia 20377: [27 74  7] - Resultado Real:7
Instancia 20378: [26 48 27] - Resultado Real:27
Instancia 20379: [39 42 21] - Resultado Real:42
Instancia 20380: [42 35 36] - Resultado Real:52
Instancia 20381: [48 75 61] - Resultado Real:61
Instancia 20382: [28 75  7] - Resultado Real:7
```

Gráfica comparación de accuracy entre modelos y técnicas

```
accuracies_multi=[m1,m2,m3,m4,m5]

# Definir la anchura de las barras y la posición en el eje X para cada grupo
bar_width = 0.3
x_pos = np.arange(len(models))

# Crear una figura y un eje
fig, ax = plt.subplots()

# Dibujar las barras
```

```
ax.bar(x_pos, accuracies_multi, width=bar_width, label='Con tres propuestas')
ax.bar(x_pos + bar_width, accuracies, width=bar_width, label='Después del balanceo')
ax.bar(x_pos + bar_width * 2, accuracies_antes, width=bar_width, label='Antes del balanceo')

# Agregar etiquetas de los ejes, título y leyenda
ax.set_xlabel('Modelos')
ax.set_ylabel('Accuracy')
ax.set_title('Comparación de Accuracy')
ax.set_xticks(x_pos + bar_width)
ax.set_xticklabels(models)
plt.legend(loc='upper left', bbox_to_anchor=(1.05, 1))

plt.xticks(rotation=90)

plt.show()
```



Análisis de modelo Random Forest

```
#ajuste de hiperparámetros

from sklearn.model_selection import GridSearchCV

rfc = RandomForestClassifier()

# Definir los hiperparámetros que queremos ajustar
param_grid = {'n_estimators': [50, 100,
              'criterion': ['gini', 'entropy']}

# Crear el objeto GridSearchCV
grid_search = GridSearchCV(estimator = rfc, param_grid = param_grid,
                           cv = 5, n_jobs = -1, verbose = 2)

# Ajustar el objeto GridSearchCV a los datos
grid_search.fit(X, y)

# Imprimir los mejores parámetros encontrados
print(grid_search.best_params_)
```

```
       File "<ipython-input-3-c6e76e725167>", line 8
         param_grid = {'n_estimators': [50, 100]
                                               ^
    SyntaxError: invalid syntax. Perhaps you forgot a comma?
```

SEARCH STACK OVERFLOW

```
from sklearn.model_selection import RandomizedSearchCV

# Definir el modelo
rfc = RandomForestClassifier()

# Definir los hiperparámetros y sus rangos de valores posibles
param_dist = {'n_estimators': [10,50,100],
              'criterion': ['gini', 'entropy']}

# Crear el objeto RandomizedSearchCV
random_search = RandomizedSearchCV(estimator = rfc, param_distributions = param_dist,
                                   n_iter = 10, cv = 5, n_jobs = -1, verbose = 2)

# Ajustar el objeto RandomizedSearchCV a los datos
random_search.fit(X, y)

# Imprimir los mejores parámetros encontrados
print(random_search.best_params_)
```