validacion-de-la-informacionipynb

November 23, 2023

#Herramientas para la visualización de datos masivos

Objetivo: El objetivo de este cuaderno es cargar y realizar la exploración inicial de los datos utilizando el lenguaje de programación Python.

Exploración de Datos

Indice

Cargar los Datos

Visualizar los Datos

Tipos de datos

Visualizar las estadísticas

Identificar datos faltantes

Explorar relaciones entre los datos

Graficar las estadísticas

Exportar los datos

##Cargar los datos

Existen varios formatos para un conjunto de datos, .csv, .json, .xlsx, etc. Los datos pueden ser almacenados en distintos lugares, ya sea localmente o en línea. En estas sección aprenderá a cargar un conjunto de datos en su cuaderno de python. En nuestro caso el conjunto de datos Automobile es de una fuente en línea en formato CSV (valores separados por coma). Usemos este conjunto como ejemplo para practicar la lectura de datos.

fuente de datos: https://archive.ics.uci.edu/ml/machine-learning-databases/autos/imports-85.data

tipo de datos: csv

Vamos a utilizar la librería Pandas de Python para realizar la lectura de archivos. Le ponemos un alias pd para que sea más fácil utilizarla:

[ ]:
```
# Importar libreria requerida
import pandas as pd
import numpy as np
import os
```

Después del comando para importar, ahora tenemos acceso a una gran cantidad de clases y funciones predefinidas. Una forma en que pandas le permite trabajar con datos es con dataframes. Repasemos el proceso para pasar de un archivo de valores separados por comas (.csv) a un dataframe. Esta variable csv_path almacena la ruta de .csv, que se utiliza como argumento para la función read_csv. El resultado se almacena en el objeto df, esta es una forma corta común que se usa para una variable que se refiere a un dataframe de Pandas.

[ ]:
```
# ruta de archivos
files = os.listdir("c:\\archivos\proyecto")
os.chdir(r'C:\archivos\proyecto')

csv_path = 'ind_urgencias_final_2023_filtrado.txt'

# Read data from CSV file
df = pd.read_csv(csv_path,sep=";",header= None)
```

```
C:\Users\Victor.Gomez\AppData\Local\Temp\ipykernel_14576\2968785239.py:8:
DtypeWarning: Columns (10,12,17,18,19,21,24) have mixed types. Specify dtype
option on import or set low_memory=False.
  df = pd.read_csv(csv_path,sep=";",header= None)
```

##Visualizar los datos

Podemos utilizar el método dataframe.head() para examinar las primeras cinco filas del dataframe, se utiliza cuando el conjuntos de datos es muy grande y no queremos cargar todo:

[ ]:
```
# Imprimir las primeras cinco filas de un dataframe
df.head()
```

[ ]:

	0	1	2
0	FECHA_LLEGADA	FECHA_TRIAGE	FECHA_INGRESO
1	2023-01-01 01:20:23.853	2023-01-01 01:28:01.847	2023-01-01 01:29:41.210
2	2023-01-01 01:29:46.050	2023-01-01 01:48:03.070	2023-01-01 01:49:40.973
3	2023-01-01 03:15:35.623	2023-01-01 03:23:01.990	2023-01-01 03:23:39.793
4	2023-01-01 05:54:53.563	2023-01-01 06:00:07.943	2023-01-01 06:02:07.320

	3	4	5
0	FECHA_ATENCION	TIEMPO_DGTURNO_A_TRIAGE	TIEMPO_TRIAGE_A_INGRESO
1	2023-01-01 02:00:07.590	0:07:38	0:01:40
2	2023-01-01 02:02:53.663	0:18:17	0:01:37
3	2023-01-01 03:30:21.233	0:07:26	0:00:38
4	2023-01-01 06:26:17.050	0:05:14	0:02:00

	6	7	8
0	TIEMPO_INGRESO_A_CONSULTA	TIEMPO_TOTAL	Tiempo_Minutos_Total
1	0:30:26	0:39:44	39,73
2	0:13:13	0:33:07	33,12
3	0:06:42	0:14:46	14,77
4	0:24:10	0:31:24	31,40

```
                9   …                15               16      17    18   19  \
0  CENTRO_ATENCION  …  RÉGIMEN PACIENTE  NOMBRE_ENTIDAD   MEDICO   AÑO  MES
1               TN  …        SUBSIDIADO           EPSS34   11065  2023    1
2               ME  …        SUBSIDIADO           EPSS34    8861  2023    1
3               UC  …      CONTRIBUTIVO           EPS002    5855  2023    1
4               UC  …        SUBSIDIADO           EPSS34   11072  2023    1

           20    21      22                      23   24
0  DIA_SEMANA  HOUR  Turnos                    TIME  DIA
1     DOMINGO     1   NOCHE  2023-01-01 01:20:23.853    1
2     DOMINGO     1   NOCHE  2023-01-01 01:29:46.050    1
3     DOMINGO     3   NOCHE  2023-01-01 03:15:35.623    1
4     DOMINGO     5   NOCHE  2023-01-01 05:54:53.563    1

[5 rows x 25 columns]
```

Después de leer el conjunto de datos podemos utilizar el método dataframe.head(n) para revisar las primeras n filas del dataframe; donde n es un entero. Al contrario de dataframe.head(n), dataframe.tail(n) mostrará las n filas del final del dataframe.

AHORA TÚ:

Revise las ultimas 10 filas del dataframe “df”:

[ ]:
```
# Escribe tu código aquí y presiona Shift+Enter para ejecutar
print("The last 10 rows of the dataframe\n")
df.tail(10)
```

```
The last 10 rows of the dataframe
```

[ ]:
```
                           0                       1   \
82407  2023-09-17 18:12:40.660  2023-09-17 18:43:16.430
82408  2023-09-17 18:52:28.640  2023-09-17 19:48:17.070
82409  2023-09-17 19:57:56.033  2023-09-17 20:15:56.793
82410  2023-09-17 20:22:04.970  2023-09-17 20:36:16.703
82411  2023-09-17 21:22:59.137  2023-09-17 21:42:20.273
82412  2023-09-18 04:44:41.970  2023-09-18 04:53:22.553
82413  2023-09-18 06:17:00.573  2023-09-18 06:28:43.040
82414  2023-09-18 06:21:37.273  2023-09-18 07:00:57.420
82415  2023-09-18 06:25:33.483  2023-09-18 06:42:02.883
82416  2023-09-18 07:14:58.180  2023-09-18 07:30:50.643

                           2                       3        4        5   \
82407  2023-09-17 18:48:36.403  2023-09-17 19:40:11.800  0:30:36  0:05:20
82408  2023-09-17 19:49:47.657  2023-09-17 20:32:54.070  0:55:49  0:01:30
82409  2023-09-17 20:28:57.590  2023-09-17 20:46:29.250  0:18:00  0:13:01
```

```
82410  2023-09-17 20:41:12.783  2023-09-17 22:00:50.287  0:14:12  0:04:56
82411  2023-09-17 21:51:04.433  2023-09-17 22:43:13.710  0:19:21  0:08:44
82412  2023-09-18 05:05:51.423  2023-09-18 06:09:35.867  0:08:41  0:12:29
82413  2023-09-18 06:35:38.213  2023-09-18 07:40:45.957  0:11:43  0:06:55
82414  2023-09-18 07:16:45.907  2023-09-18 08:27:27.337  0:39:20  0:15:48
82415  2023-09-18 06:51:35.970  2023-09-18 07:28:28.290  0:16:29  0:09:33
82416  2023-09-18 07:34:08.370  2023-09-18 07:49:18.440  0:15:52  0:03:18

            6        7       8   9   …            15      16     17    18  \
82407  0:51:35  1:27:31   87,52  ME  …  CONTRIBUTIVO  EPS005    120  2023
82408  0:43:07  1:40:26  100,43  ME  …     SUBSIDIADO  EPSS17    120  2023
82409  0:17:32  0:48:33   48,55  UC  …     SUBSIDIADO  EPSC34   3818  2023
82410  1:19:38  1:38:46   98,77  ME  …     SUBSIDIADO  EPSS02  10786  2023
82411  0:52:09  1:20:14   80,23  ME  …      VINCULADO   11000  10786  2023
82412  1:03:44  1:24:54   84,90  ME  …  CONTRIBUTIVO  EPS017   7844  2023
82413  1:05:07  1:23:45   83,75  ME  …     SUBSIDIADO  EPSC34   6204  2023
82414  1:10:42  2:05:50  125,83  ME  …  CONTRIBUTIVO  EPS041   9951  2023
82415  0:36:53  1:02:55   62,92  UB  …     SUBSIDIADO  EPSC34   4030  2023
82416  0:15:10  0:34:20   34,33  TN  …     SUBSIDIADO  EPSC34   1239  2023

      19       20  21      22                      23  24
82407  9  DOMINGO  18   TARDE  2023-09-17 18:12:40.660  17
82408  9  DOMINGO  18   TARDE  2023-09-17 18:52:28.640  17
82409  9  DOMINGO  19   TARDE  2023-09-17 19:57:56.033  17
82410  9  DOMINGO  20   NOCHE  2023-09-17 20:22:04.970  17
82411  9  DOMINGO  21   NOCHE  2023-09-17 21:22:59.137  17
82412  9    LUNES   4   NOCHE  2023-09-18 04:44:41.970  18
82413  9    LUNES   6   NOCHE  2023-09-18 06:17:00.573  18
82414  9    LUNES   6   NOCHE  2023-09-18 06:21:37.273  18
82415  9    LUNES   6   NOCHE  2023-09-18 06:25:33.483  18
82416  9    LUNES   7  MAÑANA  2023-09-18 07:14:58.180  18

[10 rows x 25 columns]
```

###Añadir cabeceras

Observe el conjunto de datos; Pandas automaticamente establece la cabecera en un entero a partir de 0.
Para describir mejor nuestros datos podemos agregarle una cabecera, esta información esta disponible en: https://archive.ics.uci.edu/ml/datasets/Automobile

De este modo debemos agregar las cabeceras manualmente.
Primero creamos una lista headers que incluya todos los nombres de columna en orden. Despues usamos dataframe.columns = headers para reemplazar las cabeceras por la lista que hemos creado.

[ ]:
```
# crear la lista headers
headers =␣
  ↪["FECHA_LLEGADA","FECHA_TRIAGE","FECHA_INGRESO","FECHA_ATENCION","TIEMPO_DGTURNO_A_TRIAGE",
```

```
␣
↪"CENTRO_ATENCION","CLASIFICACION_TRIAGE","PACIENTE_#_DOCUMENTO","EDAD","EDAD_RANGO","SEXO",
↪PACIENTE","NOMBRE_ENTIDAD","MEDICO","AÑO","MES","DIA_SEMANA","HOUR","Turnos","TIME","DIA"]
print("headers\n", headers)
```

```
headers
 ['FECHA_LLEGADA', 'FECHA_TRIAGE', 'FECHA_INGRESO', 'FECHA_ATENCION',
'TIEMPO_DGTURNO_A_TRIAGE', 'TIEMPO_TRIAGE_A_INGRESO',
'TIEMPO_INGRESO_A_CONSULTA', 'TIEMPO_TOTAL', 'Tiempo_Minutos_Total',
'CENTRO_ATENCION', 'CLASIFICACION_TRIAGE', 'PACIENTE_#_DOCUMENTO', 'EDAD',
'EDAD_RANGO', 'SEXO', 'RÉGIMEN PACIENTE', 'NOMBRE_ENTIDAD', 'MEDICO', 'AÑO',
'MES', 'DIA_SEMANA', 'HOUR', 'Turnos', 'TIME', 'DIA']
```

Remplazamos las cabeceras y volvemos a revisar nuestro dataframe:

[ ]:
```
df.columns = headers
df.head()
```

Acceder a una columna y ver sus valores

Se accede a una columna especificando el nombre de la misma. Por ejemplo, puedes acceder a la columna symboling y a la columna body-style. Cada una de estas columnas es una serie de Pandas.

[ ]:
```
x=df[["PACIENTE_#_DOCUMENTO"]]
x
```

[ ]:
```
       PACIENTE_#_DOCUMENTO
0      PACIENTE_#_DOCUMENTO
1                1007228378
2                1000003681
3                1007454009
4                1022997183
…                       …
82412            1010242518
82413              93357619
82414              41372387
82415            1000807249
82416               4497020

[82417 rows x 1 columns]
```

[ ]:
```
y=df[["Tiempo_Minutos_Total"]]
y
```

[ ]:
```
       Tiempo_Minutos_Total
0      Tiempo_Minutos_Total
1                     39,73
2                     33,12
3                     14,77
```

```
4                    31,40
…                    …
82412                84,90
82413                83,75
82414               125,83
82415                62,92
82416                34,33

[82417 rows x 1 columns]
```

##Tipos de datos

Los datos se encuentran en una variedad de tipos. Los tipos principales almacenados en dataframes de Pandas son object, float, int, bool y datetime64. Para aprender mejor acerca de cada atributo es mejor para nosotros saber el tipo de dato de cada columna.

[ ]:
```
#La función dtypes genera una tabla con el tipo de dato de cada columna
df.dtypes
```

[ ]:
```
FECHA_LLEGADA                object
FECHA_TRIAGE                 object
FECHA_INGRESO                object
FECHA_ATENCION               object
TIEMPO_DGTURNO_A_TRIAGE      object
TIEMPO_TRIAGE_A_INGRESO      object
TIEMPO_INGRESO_A_CONSULTA    object
TIEMPO_TOTAL                 object
Tiempo_Minutos_Total         object
CENTRO_ATENCION              object
CLASIFICACION_TRIAGE         object
PACIENTE_#_DOCUMENTO         object
EDAD                         object
EDAD_RANGO                   object
SEXO                         object
RÉGIMEN PACIENTE             object
NOMBRE_ENTIDAD               object
MEDICO                       object
AÑO                          object
MES                          object
DIA_SEMANA                   object
HOUR                         object
Turnos                       object
TIME                         object
DIA                          object
dtype: object
```

Tipo de dato de una columna específica

De esta forma podemos consultar cuál es el tipo de dato de una columna específica:

[ ]:
```
#Separamos la columna en una dataframe llamado df_column
df_column=df[['Turnos']]
df_column.dtypes
```

[ ]:
```
Turnos    object
dtype: object
```

Cambiar el tipo de dato de una columna específica

¿Cómo cambiar el tipo de dato de una columna específica? Cambiemos el tipo de datos de la columna Price que fue identificado como object y es un float.

[ ]:
```
#utilizamos errors='coerce' para ignorar los datos faltantes
# df["price"] = pd.to_numeric(df["price"],errors='coerce')

df['Turnos'] = df['Turnos'].astype('category')
df['DIA_SEMANA'] = df['DIA_SEMANA'].astype('category')
df['CENTRO_ATENCION'] = df['CENTRO_ATENCION'].astype('category')
df.dtypes
```

[ ]:
```
FECHA_LLEGADA                 object
FECHA_TRIAGE                  object
FECHA_INGRESO                 object
FECHA_ATENCION                object
TIEMPO_DGTURNO_A_TRIAGE       object
TIEMPO_TRIAGE_A_INGRESO       object
TIEMPO_INGRESO_A_CONSULTA     object
TIEMPO_TOTAL                  object
Tiempo_Minutos_Total          object
CENTRO_ATENCION             category
CLASIFICACION_TRIAGE          object
PACIENTE_#_DOCUMENTO          object
EDAD                          object
EDAD_RANGO                    object
SEXO                          object
RÉGIMEN PACIENTE              object
NOMBRE_ENTIDAD                object
MEDICO                        object
AÑO                           object
MES                           object
DIA_SEMANA                  category
HOUR                          object
Turnos                      category
TIME                          object
DIA                           object
dtype: object
```

Como se muestra, se observa claramente que el tipo de dato de symboling y curb-weight es int64,

normalized-losses es object pero debería ser de tipo numérico, al igual que bore, etc. Estos tipos de datos pueden modificarse.

AHORA TÚ:

Cambie el tipo de datos de la columna "stroke":

[ ]:
```
# Escribe tu código aquí y presiona Shift+Enter para ejecutar

df['Tiempo_Total'] = df['Tiempo_Minutos_Total'].str.replace(',', '.',␣
↪regex=True)
df['Tiempo_Total'] = pd.to_numeric(df['Tiempo_Total'], errors='coerce')
```

##Visualizar las estadísticas

Este conjunto de datos es pequeño, pero si se quisiera saber la cantidad de atributos y de elementos que se tienen en el conjunto de datos, se puede utilizar la función dataframe.shape. Esta función visualiza primero el número de elementos y luego el número de atributos.

[ ]:
```
df.shape
```

[ ]:
```
(82417, 26)
```

Vamos a utilizar la función dataframe.describe para visualizar las estadísticas del conjunto de datos. Por defecto, la función dataframe.describe muestra las filas y columnas que contienen números.

Esto mostrará:

el recuento de esa variable

la media

la desviación estándar (std)

el valor mínimo

el IQR (rango intercuartil: 25%, 50% y 75%)

el valor máximo

[ ]:
```
df.describe()
```

[ ]:
```
       Tiempo_Total
count  82416.000000
mean     108.346035
std      132.899154
min        8.570000
25%       60.600000
50%       84.200000
75%      117.672500
max     2684.830000
```

Si se quisiera calcular la mediana de una variable en específico se puede de la siguiente manera:

[ ]:
```
import matplotlib.pyplot as plt
import seaborn as sns
sns.distplot(df["Tiempo_Total"])
```

[ ]:
```
df.head()
```

[ ]:
```
             FECHA_LLEGADA             FECHA_TRIAGE            FECHA_INGRESO  \
1  2023-01-01 01:20:23.853  2023-01-01 01:28:01.847  2023-01-01 01:29:41.210
2  2023-01-01 01:29:46.050  2023-01-01 01:48:03.070  2023-01-01 01:49:40.973
3  2023-01-01 03:15:35.623  2023-01-01 03:23:01.990  2023-01-01 03:23:39.793
4  2023-01-01 05:54:53.563  2023-01-01 06:00:07.943  2023-01-01 06:02:07.320
5  2023-01-01 06:37:27.237  2023-01-01 07:52:31.687  2023-01-01 07:52:37.717

            FECHA_ATENCION TIEMPO_DGTURNO_A_TRIAGE TIEMPO_TRIAGE_A_INGRESO  \
1  2023-01-01 02:00:07.590                0:07:38                 0:01:40
2  2023-01-01 02:02:53.663                0:18:17                 0:01:37
3  2023-01-01 03:30:21.233                0:07:26                 0:00:38
4  2023-01-01 06:26:17.050                0:05:14                 0:02:00
5  2023-01-01 09:31:15.597                1:15:04                 0:00:06

  TIEMPO_INGRESO_A_CONSULTA TIEMPO_TOTAL Tiempo_Minutos_Total CENTRO_ATENCION  \
1                   0:30:26      0:39:44               39,73             TN
2                   0:13:13      0:33:07               33,12             ME
3                   0:06:42      0:14:46               14,77             UC
4                   0:24:10      0:31:24               31,40             UC
5                   1:38:38      2:53:48              173,80             TN

   … NOMBRE_ENTIDAD MEDICO   AÑO MES DIA_SEMANA HOUR Turnos  \
1  …         EPSS34  11065  2023   1    DOMINGO    1  NOCHE
2  …         EPSS34   8861  2023   1    DOMINGO    1  NOCHE
3  …         EPS002   5855  2023   1    DOMINGO    3  NOCHE
4  …         EPSS34  11072  2023   1    DOMINGO    5  NOCHE
5  …         EPSS34   1239  2023   1    DOMINGO    6  NOCHE

                     TIME DIA Tiempo_Total
1  2023-01-01 01:20:23.853   1        39.73
2  2023-01-01 01:29:46.050   1        33.12
3  2023-01-01 03:15:35.623   1        14.77
4  2023-01-01 05:54:53.563   1        31.40
5  2023-01-01 06:37:27.237   1       173.80

[5 rows x 26 columns]
```

[ ]:
```
df = df.drop(0)
```

[ ]:
```
#Muestra la mediana para los atributos "length" y "compression-ratio"
median= df[['Tiempo_Total']].median()
```

```
median
```

```
Tiempo_Total    84.2
dtype: float64
```

Por defecto la función solo muestra atributos que son numéricos. Es posible hacer que la función describe funcione también para las columnas de tipo object. Para permitir un resumen de todas las columnas, podríamos añadir un argumento include="all" entre los paréntesis de la función describe.

```
#unique, top y frequency ("único, superior y frecuencia").
#df.describe(include="object")
df.describe(include="all")
```

```
                  FECHA_LLEGADA             FECHA_TRIAGE  \
count                     82416                    82416
unique                    80382                    80388
top     2023-08-06 14:31:20.657  2023-08-04 11:13:29.903
freq                          4                        4
mean                        NaN                      NaN
std                         NaN                      NaN
min                         NaN                      NaN
25%                         NaN                      NaN
50%                         NaN                      NaN
75%                         NaN                      NaN
max                         NaN                      NaN

                  FECHA_INGRESO           FECHA_ATENCION  \
count                     82416                    82416
unique                    80387                    82414
top     2023-05-05 16:30:25.897  2023-03-29 21:49:44.440
freq                          4                        2
mean                        NaN                      NaN
std                         NaN                      NaN
min                         NaN                      NaN
25%                         NaN                      NaN
50%                         NaN                      NaN
75%                         NaN                      NaN
max                         NaN                      NaN

       TIEMPO_DGTURNO_A_TRIAGE TIEMPO_TRIAGE_A_INGRESO  \
count                    82416                   82416
unique                    5981                    3433
top                     0:10:06                 0:01:08
freq                         72                     218
mean                        NaN                     NaN
std                         NaN                     NaN
min                         NaN                     NaN
```

```
25%                          NaN                    NaN
50%                          NaN                    NaN
75%                          NaN                    NaN
max                          NaN                    NaN

       TIEMPO_INGRESO_A_CONSULTA TIEMPO_TOTAL Tiempo_Minutos_Total  \
count                      82416        82416                82416
unique                     12074        13866                14141
top                      0:33:27      1:13:04                73,07
freq                          37           28                   28
mean                         NaN          NaN                  NaN
std                          NaN          NaN                  NaN
min                          NaN          NaN                  NaN
25%                          NaN          NaN                  NaN
50%                          NaN          NaN                  NaN
75%                          NaN          NaN                  NaN
max                          NaN          NaN                  NaN

       CENTRO_ATENCION  …  NOMBRE_ENTIDAD   MEDICO      AÑO      MES  \
count            82416  …           82416  82416.0  82416.0  82416.0
unique               9  …              78    378.0      2.0     14.0
top                 ME  …          EPSC34   3826.0   2023.0      7.0
freq             41221  …           45973   1464.0  49649.0  10108.0
mean               NaN  …             NaN      NaN      NaN      NaN
std                NaN  …             NaN      NaN      NaN      NaN
min                NaN  …             NaN      NaN      NaN      NaN
25%                NaN  …             NaN      NaN      NaN      NaN
50%                NaN  …             NaN      NaN      NaN      NaN
75%                NaN  …             NaN      NaN      NaN      NaN
max                NaN  …             NaN      NaN      NaN      NaN

       DIA_SEMANA     HOUR  Turnos                       TIME      DIA  \
count       82416  82416.0   82416                      82416  82416.0
unique          7     48.0       3                      80382     62.0
top         LUNES     10.0  MAÑANA  2023-08-06 14:31:20.657      8.0
freq        13390   3911.0   40363                          4   2040.0
mean          NaN      NaN     NaN                        NaN      NaN
std           NaN      NaN     NaN                        NaN      NaN
min           NaN      NaN     NaN                        NaN      NaN
25%           NaN      NaN     NaN                        NaN      NaN
50%           NaN      NaN     NaN                        NaN      NaN
75%           NaN      NaN     NaN                        NaN      NaN
max           NaN      NaN     NaN                        NaN      NaN

        Tiempo_Total
count   82416.000000
unique           NaN
```

```
top              NaN
freq             NaN
mean      108.346035
std       132.899154
min         8.570000
25%        60.600000
50%        84.200000
75%       117.672500
max      2684.830000

[11 rows x 26 columns]
```

Contar Valores

Una forma de resumir los datos categóricos es usando la función value_counts. Por ejemplo, en nuestro conjunto de datos, tenemos el lugar del motor (engine-location) como una variable categórica de frontal y trasero.

[ ]:
```
drive_wheels_counts = df['Turnos'].value_counts().to_frame()
drive_wheels_counts
```

[ ]:
```
         count
Turnos
MAÑANA   40363
TARDE    25052
NOCHE    17001
Turnos       0
```

Examinar los recuentos de valores de la ubicación del motor no sería una buena variable predictiva del precio. Esto se debe a que solo tenemos tres autos con motor trasero y 202 con motor delantero, este resultado es sesgado. Por lo tanto, no podemos sacar ninguna conclusión sobre la ubicación del motor.

AHORA TÚ:

Puede seleccionar las columna de un dataframe indicando el nombre de cada una, por ejemplo, puede seleccionar tres columnas de la siguiente manera:

dataframe[[‘column 1’,column 2’, ‘column 3’]]

Donde “column” es el nombre de la columna se puede aplicar el método “.describe()” para obtener las estadísticas de aquellas columnas de la siguiente manera:

dataframe[[’ column 1 ‘,column 2’, ‘column 3’]].describe()

Aplicar el método “.describe()” a las columnas ‘length’ y ‘compression-ratio’.

[ ]:
```
# Escribe tu código aquí y presiona Shift+Enter para ejecutar
df[['DIA_SEMANA', 'Turnos','Tiempo_Total']].describe()
```

[ ]:
```
       Tiempo_Total
count  82416.000000
mean     108.346035
std      132.899154
min        8.570000
25%       60.600000
50%       84.200000
75%      117.672500
max     2684.830000
```

##Identificar datos faltantes

Debemos visualizar nuestros datos e identificar el valor(es) que se está utilizando para los datos faltantes. Los valores faltantes pueden ser espacios vacíos, NA, n/a, –, 0, o cualquier otro valor que no es considerado correcto en esa columna.

[ ]:
```
#Identifique cual es el valor que se está utilizando para los datos faltantes␣
↪en el set de datos:
df.head(20)
```

[ ]:
```
              FECHA_LLEGADA             FECHA_TRIAGE            FECHA_INGRESO  \
1   2023-01-01 01:20:23.853  2023-01-01 01:28:01.847  2023-01-01 01:29:41.210
2   2023-01-01 01:29:46.050  2023-01-01 01:48:03.070  2023-01-01 01:49:40.973
3   2023-01-01 03:15:35.623  2023-01-01 03:23:01.990  2023-01-01 03:23:39.793
4   2023-01-01 05:54:53.563  2023-01-01 06:00:07.943  2023-01-01 06:02:07.320
5   2023-01-01 06:37:27.237  2023-01-01 07:52:31.687  2023-01-01 07:52:37.717
6   2023-01-01 07:09:46.950  2023-01-01 07:18:01.200  2023-01-01 07:20:03.720
7   2023-01-01 07:20:31.113  2023-01-01 07:27:36.230  2023-01-01 07:34:17.667
8   2023-01-01 07:53:52.963  2023-01-01 08:01:10.640  2023-01-01 08:03:13.710
9   2023-01-01 08:05:21.230  2023-01-01 08:53:07.870  2023-01-01 08:56:10.277
10  2023-01-01 09:24:15.530  2023-01-01 09:52:05.463  2023-01-01 09:59:19.673
11  2023-01-01 10:57:35.917  2023-01-01 11:26:37.567  2023-01-01 11:30:19.197
12  2023-01-01 14:34:37.470  2023-01-01 14:49:51.770  2023-01-01 15:02:04.030
13  2023-01-01 14:54:22.407  2023-01-01 15:13:27.167  2023-01-01 15:20:55.660
14  2023-01-01 14:56:15.033  2023-01-01 15:03:35.823  2023-01-01 15:04:17.987
15  2023-01-01 16:09:09.527  2023-01-01 16:14:42.000  2023-01-01 16:16:22.767
16  2023-01-01 18:07:03.087  2023-01-01 18:23:37.837  2023-01-01 18:24:11.420
17  2023-01-01 18:13:00.330  2023-01-01 18:31:06.813  2023-01-01 18:34:41.503
18  2023-01-02 03:34:19.883  2023-01-02 03:56:47.470  2023-01-02 03:57:22.443
19  2023-01-02 04:30:07.617  2023-01-02 05:36:52.657  2023-01-02 05:38:34.253
20  2023-01-02 06:27:33.233  2023-01-02 07:12:56.400  2023-01-02 07:13:07.473

              FECHA_ATENCION TIEMPO_DGTURNO_A_TRIAGE TIEMPO_TRIAGE_A_INGRESO  \
1   2023-01-01 02:00:07.590                0:07:38                0:01:40
2   2023-01-01 02:02:53.663                0:18:17                0:01:37
3   2023-01-01 03:30:21.233                0:07:26                0:00:38
4   2023-01-01 06:26:17.050                0:05:14                0:02:00
5   2023-01-01 09:31:15.597                1:15:04                0:00:06
```

```
6   2023-01-01 07:48:42.197                0:08:15                0:02:02
7   2023-01-01 07:59:17.140                0:07:05                0:06:41
8   2023-01-01 08:43:09.917                0:07:18                0:02:03
9   2023-01-01 09:26:52.977                0:47:46                0:03:03
10  2023-01-01 10:38:12.297                0:27:50                0:07:14
11  2023-01-01 12:48:10.377                0:29:02                0:03:42
12  2023-01-01 16:33:36.400                0:15:14                0:12:13
13  2023-01-01 16:47:10.060                0:19:05                0:07:28
14  2023-01-01 16:12:09.400                0:07:20                0:00:42
15  2023-01-01 16:39:18.937                0:05:33                0:01:40
16  2023-01-01 19:33:28.253                0:16:34                0:00:34
17  2023-01-01 19:54:53.770                0:18:06                0:03:35
18  2023-01-02 04:49:21.190                0:22:28                0:00:35
19  2023-01-02 05:54:41.813                1:06:45                0:01:42
20  2023-01-02 07:21:55.423                0:45:23                0:00:11

   TIEMPO_INGRESO_A_CONSULTA TIEMPO_TOTAL Tiempo_Minutos_Total  \
1                    0:30:26      0:39:44                39,73
2                    0:13:13      0:33:07                33,12
3                    0:06:42      0:14:46                14,77
4                    0:24:10      0:31:24                31,40
5                    1:38:38      2:53:48               173,80
6                    0:28:39      0:38:56                38,93
7                    0:25:00      0:38:46                38,77
8                    0:39:56      0:49:17                49,28
9                    0:30:42      1:21:31                81,52
10                   0:38:53      1:13:57                73,95
11                   1:17:51      1:50:35               110,58
12                   1:31:32      1:58:59               118,98
13                   1:26:15      1:52:48               112,80
14                   1:07:52      1:15:54                75,90
15                   0:22:56      0:30:09                30,15
16                   1:09:17      1:26:25                86,42
17                   1:20:12      1:41:53               101,88
18                   0:51:59      1:15:02                75,03
19                   0:16:07      1:24:34                84,57
20                   0:08:48      0:54:22                54,37

   CENTRO_ATENCION  … NOMBRE_ENTIDAD MEDICO   AÑO MES DIA_SEMANA HOUR  \
1               TN  …         EPSS34  11065  2023   1    DOMINGO    1
2               ME  …         EPSS34   8861  2023   1    DOMINGO    1
3               UC  …         EPS002   5855  2023   1    DOMINGO    3
4               UC  …         EPSS34  11072  2023   1    DOMINGO    5
5               TN  …         EPSS34   1239  2023   1    DOMINGO    6
6               VB  …         EPS005  10751  2023   1    DOMINGO    7
7               UC  …         EPSS34  11072  2023   1    DOMINGO    7
8               VB  …         EPS041  10795  2023   1    DOMINGO    7
```

```
9              UB  …         ESS207   7795  2023   1    DOMINGO    8
10             ME  …         EPSS34    120  2023   1    DOMINGO    9
11             ME  …         EPSS17   7551  2023   1    DOMINGO   10
12             ME  …         ESS024   6204  2023   1    DOMINGO   14
13             VB  …         EPSS34   3826  2023   1    DOMINGO   14
14             VB  …         EPSS34  10795  2023   1    DOMINGO   14
15             TN  …         EPSC34   1239  2023   1    DOMINGO   16
16             TN  …         EPSS34  10030  2023   1    DOMINGO   18
17             TN  …         EPSS05  10030  2023   1    DOMINGO   18
18             ME  …         EPS002  10938  2023   1      LUNES    3
19             JT  …         EPSS34   8373  2023   1      LUNES    4
20             ME  …         EPSC34   4610  2023   1      LUNES    6

    Turnos                       TIME DIA Tiempo_Total
1    NOCHE  2023-01-01 01:20:23.853   1        39.73
2    NOCHE  2023-01-01 01:29:46.050   1        33.12
3    NOCHE  2023-01-01 03:15:35.623   1        14.77
4    NOCHE  2023-01-01 05:54:53.563   1        31.40
5    NOCHE  2023-01-01 06:37:27.237   1       173.80
6   MAÑANA  2023-01-01 07:09:46.950   1        38.93
7   MAÑANA  2023-01-01 07:20:31.113   1        38.77
8   MAÑANA  2023-01-01 07:53:52.963   1        49.28
9   MAÑANA  2023-01-01 08:05:21.230   1        81.52
10  MAÑANA  2023-01-01 09:24:15.530   1        73.95
11  MAÑANA  2023-01-01 10:57:35.917   1       110.58
12   TARDE  2023-01-01 14:34:37.470   1       118.98
13   TARDE  2023-01-01 14:54:22.407   1       112.80
14   TARDE  2023-01-01 14:56:15.033   1        75.90
15   TARDE  2023-01-01 16:09:09.527   1        30.15
16   TARDE  2023-01-01 18:07:03.087   1        86.42
17   TARDE  2023-01-01 18:13:00.330   1       101.88
18   NOCHE  2023-01-02 03:34:19.883   2        75.03
19   NOCHE  2023-01-02 04:30:07.617   2        84.57
20   NOCHE  2023-01-02 06:27:33.233   2        54.37

[20 rows x 26 columns]
```

Con la función isnull podemos saber cuantos datos faltantes identifica Python en nuestro set de datos.

[ ]:
```
print(df.isnull().sum())
```

```
FECHA_LLEGADA               0
FECHA_TRIAGE                0
FECHA_INGRESO               0
FECHA_ATENCION              0
TIEMPO_DGTURNO_A_TRIAGE     0
TIEMPO_TRIAGE_A_INGRESO     0
```

```
TIEMPO_INGRESO_A_CONSULTA    0
TIEMPO_TOTAL                 0
Tiempo_Minutos_Total         0
CENTRO_ATENCION              0
CLASIFICACION_TRIAGE         0
PACIENTE_#_DOCUMENTO         0
EDAD                         0
EDAD_RANGO                   0
SEXO                         0
RÉGIMEN PACIENTE             0
NOMBRE_ENTIDAD               0
MEDICO                       0
AÑO                          0
MES                          0
DIA_SEMANA                   0
HOUR                         0
Turnos                       0
TIME                         0
DIA                          0
Tiempo_Total                 0
dtype: int64
```

Todavía Python no está identificando los datos faltantes en el conjunto de datos, sino que los está tratando como un valor correcto más. Para marcar los datos faltantes se realiza los siguiente:

[ ]:
```
#Realice una lista de los valores que son identificados como datos faltantes
#No olvide al final volver a cargar las cabeceras
missing_values = ["?", "1"]
csv_path = 'ind_urgencias_final_2023_filtrado.txt'
df = pd.read_csv(csv_path,sep=";",header= None, na_values = missing_values)
df.head(20)
```

```
C:\Users\Victor.Gomez\AppData\Local\Temp\ipykernel_14576\2949932682.py:5:
DtypeWarning: Columns (10,12,17,18,19,21,24) have mixed types. Specify dtype
option on import or set low_memory=False.
  df = pd.read_csv(csv_path,sep=";",header= None, na_values = missing_values)
```

[ ]:

	0	1	2	\
0	FECHA_LLEGADA	FECHA_TRIAGE	FECHA_INGRESO	
1	2023-01-01 01:20:23.853	2023-01-01 01:28:01.847	2023-01-01 01:29:41.210	
2	2023-01-01 01:29:46.050	2023-01-01 01:48:03.070	2023-01-01 01:49:40.973	
3	2023-01-01 03:15:35.623	2023-01-01 03:23:01.990	2023-01-01 03:23:39.793	
4	2023-01-01 05:54:53.563	2023-01-01 06:00:07.943	2023-01-01 06:02:07.320	
5	2023-01-01 06:37:27.237	2023-01-01 07:52:31.687	2023-01-01 07:52:37.717	
6	2023-01-01 07:09:46.950	2023-01-01 07:18:01.200	2023-01-01 07:20:03.720	
7	2023-01-01 07:20:31.113	2023-01-01 07:27:36.230	2023-01-01 07:34:17.667	
8	2023-01-01 07:53:52.963	2023-01-01 08:01:10.640	2023-01-01 08:03:13.710	
9	2023-01-01 08:05:21.230	2023-01-01 08:53:07.870	2023-01-01 08:56:10.277	

10	2023-01-01 09:24:15.530	2023-01-01 09:52:05.463	2023-01-01 09:59:19.673
11	2023-01-01 10:57:35.917	2023-01-01 11:26:37.567	2023-01-01 11:30:19.197
12	2023-01-01 14:34:37.470	2023-01-01 14:49:51.770	2023-01-01 15:02:04.030
13	2023-01-01 14:54:22.407	2023-01-01 15:13:27.167	2023-01-01 15:20:55.660
14	2023-01-01 14:56:15.033	2023-01-01 15:03:35.823	2023-01-01 15:04:17.987
15	2023-01-01 16:09:09.527	2023-01-01 16:14:42.000	2023-01-01 16:16:22.767
16	2023-01-01 18:07:03.087	2023-01-01 18:23:37.837	2023-01-01 18:24:11.420
17	2023-01-01 18:13:00.330	2023-01-01 18:31:06.813	2023-01-01 18:34:41.503
18	2023-01-02 03:34:19.883	2023-01-02 03:56:47.470	2023-01-02 03:57:22.443
19	2023-01-02 04:30:07.617	2023-01-02 05:36:52.657	2023-01-02 05:38:34.253

	3	4	5
0	FECHA_ATENCION	TIEMPO_DGTURNO_A_TRIAGE	TIEMPO_TRIAGE_A_INGRESO
1	2023-01-01 02:00:07.590	0:07:38	0:01:40
2	2023-01-01 02:02:53.663	0:18:17	0:01:37
3	2023-01-01 03:30:21.233	0:07:26	0:00:38
4	2023-01-01 06:26:17.050	0:05:14	0:02:00
5	2023-01-01 09:31:15.597	1:15:04	0:00:06
6	2023-01-01 07:48:42.197	0:08:15	0:02:02
7	2023-01-01 07:59:17.140	0:07:05	0:06:41
8	2023-01-01 08:43:09.917	0:07:18	0:02:03
9	2023-01-01 09:26:52.977	0:47:46	0:03:03
10	2023-01-01 10:38:12.297	0:27:50	0:07:14
11	2023-01-01 12:48:10.377	0:29:02	0:03:42
12	2023-01-01 16:33:36.400	0:15:14	0:12:13
13	2023-01-01 16:47:10.060	0:19:05	0:07:28
14	2023-01-01 16:12:09.400	0:07:20	0:00:42
15	2023-01-01 16:39:18.937	0:05:33	0:01:40
16	2023-01-01 19:33:28.253	0:16:34	0:00:34
17	2023-01-01 19:54:53.770	0:18:06	0:03:35
18	2023-01-02 04:49:21.190	0:22:28	0:00:35
19	2023-01-02 05:54:41.813	1:06:45	0:01:42

	6	7	8
0	TIEMPO_INGRESO_A_CONSULTA	TIEMPO_TOTAL	Tiempo_Minutos_Total
1	0:30:26	0:39:44	39,73
2	0:13:13	0:33:07	33,12
3	0:06:42	0:14:46	14,77
4	0:24:10	0:31:24	31,40
5	1:38:38	2:53:48	173,80
6	0:28:39	0:38:56	38,93
7	0:25:00	0:38:46	38,77
8	0:39:56	0:49:17	49,28
9	0:30:42	1:21:31	81,52
10	0:38:53	1:13:57	73,95
11	1:17:51	1:50:35	110,58
12	1:31:32	1:58:59	118,98

13	1:26:15	1:52:48	112,80
14	1:07:52	1:15:54	75,90
15	0:22:56	0:30:09	30,15
16	1:09:17	1:26:25	86,42
17	1:20:12	1:41:53	101,88
18	0:51:59	1:15:02	75,03
19	0:16:07	1:24:34	84,57

	9	…	15	16	17	18	\
0	CENTRO_ATENCION	…	RÉGIMEN PACIENTE	NOMBRE_ENTIDAD	MEDICO	AÑO	
1	TN	…	SUBSIDIADO	EPSS34	11065	2023	
2	ME	…	SUBSIDIADO	EPSS34	8861	2023	
3	UC	…	CONTRIBUTIVO	EPS002	5855	2023	
4	UC	…	SUBSIDIADO	EPSS34	11072	2023	
5	TN	…	SUBSIDIADO	EPSS34	1239	2023	
6	VB	…	CONTRIBUTIVO	EPS005	10751	2023	
7	UC	…	SUBSIDIADO	EPSS34	11072	2023	
8	VB	…	CONTRIBUTIVO	EPS041	10795	2023	
9	UB	…	SUBSIDIADO	ESS207	7795	2023	
10	ME	…	SUBSIDIADO	EPSS34	120	2023	
11	ME	…	CONTRIBUTIVO	EPSS17	7551	2023	
12	ME	…	SUBSIDIADO	ESS024	6204	2023	
13	VB	…	SUBSIDIADO	EPSS34	3826	2023	
14	VB	…	SUBSIDIADO	EPSS34	10795	2023	
15	TN	…	CONTRIBUTIVO	EPSC34	1239	2023	
16	TN	…	SUBSIDIADO	EPSS34	10030	2023	
17	TN	…	SUBSIDIADO	EPSS05	10030	2023	
18	ME	…	CONTRIBUTIVO	EPS002	10938	2023	
19	JT	…	SUBSIDIADO	EPSS34	8373	2023	

	19	20	21	22	23	24
0	MES	DIA_SEMANA	HOUR	Turnos	TIME	DIA
1	1	DOMINGO	1	NOCHE	2023-01-01 01:20:23.853	1
2	1	DOMINGO	1	NOCHE	2023-01-01 01:29:46.050	1
3	1	DOMINGO	3	NOCHE	2023-01-01 03:15:35.623	1
4	1	DOMINGO	5	NOCHE	2023-01-01 05:54:53.563	1
5	1	DOMINGO	6	NOCHE	2023-01-01 06:37:27.237	1
6	1	DOMINGO	7	MAÑANA	2023-01-01 07:09:46.950	1
7	1	DOMINGO	7	MAÑANA	2023-01-01 07:20:31.113	1
8	1	DOMINGO	7	MAÑANA	2023-01-01 07:53:52.963	1
9	1	DOMINGO	8	MAÑANA	2023-01-01 08:05:21.230	1
10	1	DOMINGO	9	MAÑANA	2023-01-01 09:24:15.530	1
11	1	DOMINGO	10	MAÑANA	2023-01-01 10:57:35.917	1
12	1	DOMINGO	14	TARDE	2023-01-01 14:34:37.470	1
13	1	DOMINGO	14	TARDE	2023-01-01 14:54:22.407	1
14	1	DOMINGO	14	TARDE	2023-01-01 14:56:15.033	1
15	1	DOMINGO	16	TARDE	2023-01-01 16:09:09.527	1

```
16    1     DOMINGO    18   TARDE  2023-01-01 18:07:03.087    1
17    1     DOMINGO    18   TARDE  2023-01-01 18:13:00.330    1
18    1       LUNES     3   NOCHE  2023-01-02 03:34:19.883    2
19    1       LUNES     4   NOCHE  2023-01-02 04:30:07.617    2

[20 rows x 25 columns]
```

[ ]:
```
# crear la lista headers

df = df.drop(0)
headers = ␣
  ↪["FECHA_LLEGADA","FECHA_TRIAGE","FECHA_INGRESO","FECHA_ATENCION","TIEMPO_DGTURNO_A_TRIAGE",
          ␣
  ↪"CENTRO_ATENCION","CLASIFICACION_TRIAGE","PACIENTE_#_DOCUMENTO","EDAD","EDAD_RANGO","SEXO",
  ↪PACIENTE","NOMBRE_ENTIDAD","MEDICO","AÑO","MES","DIA_SEMANA","HOUR","Turnos","TIME","DIA"]
df.columns = headers
df.head()
```

[ ]:
```
              FECHA_LLEGADA             FECHA_TRIAGE            FECHA_INGRESO  \
1  2023-01-01 01:20:23.853  2023-01-01 01:28:01.847  2023-01-01 01:29:41.210
2  2023-01-01 01:29:46.050  2023-01-01 01:48:03.070  2023-01-01 01:49:40.973
3  2023-01-01 03:15:35.623  2023-01-01 03:23:01.990  2023-01-01 03:23:39.793
4  2023-01-01 05:54:53.563  2023-01-01 06:00:07.943  2023-01-01 06:02:07.320
5  2023-01-01 06:37:27.237  2023-01-01 07:52:31.687  2023-01-01 07:52:37.717

             FECHA_ATENCION TIEMPO_DGTURNO_A_TRIAGE TIEMPO_TRIAGE_A_INGRESO  \
1  2023-01-01 02:00:07.590                 0:07:38                 0:01:40
2  2023-01-01 02:02:53.663                 0:18:17                 0:01:37
3  2023-01-01 03:30:21.233                 0:07:26                 0:00:38
4  2023-01-01 06:26:17.050                 0:05:14                 0:02:00
5  2023-01-01 09:31:15.597                 1:15:04                 0:00:06

  TIEMPO_INGRESO_A_CONSULTA TIEMPO_TOTAL Tiempo_Minutos_Total CENTRO_ATENCION  \
1                   0:30:26      0:39:44                39,73              TN
2                   0:13:13      0:33:07                33,12              ME
3                   0:06:42      0:14:46                14,77              UC
4                   0:24:10      0:31:24                31,40              UC
5                   1:38:38      2:53:48               173,80              TN

   … RÉGIMEN PACIENTE NOMBRE_ENTIDAD MEDICO   AÑO MES DIA_SEMANA HOUR  \
1  …       SUBSIDIADO         EPSS34  11065  2023   1    DOMINGO    1
2  …       SUBSIDIADO         EPSS34   8861  2023   1    DOMINGO    1
3  …     CONTRIBUTIVO         EPS002   5855  2023   1    DOMINGO    3
4  …       SUBSIDIADO         EPSS34  11072  2023   1    DOMINGO    5
5  …       SUBSIDIADO         EPSS34   1239  2023   1    DOMINGO    6

  Turnos                     TIME DIA
```

```
1  NOCHE  2023-01-01 01:20:23.853   1
2  NOCHE  2023-01-01 01:29:46.050   1
3  NOCHE  2023-01-01 03:15:35.623   1
4  NOCHE  2023-01-01 05:54:53.563   1
5  NOCHE  2023-01-01 06:37:27.237   1

[5 rows x 25 columns]
```

Vuelva a ejectuar la sentencia print(df.isnull().sum()) para visualizar los datos faltantes

[ ]:
```
print(df.isnull().sum())
```

```
FECHA_LLEGADA                0
FECHA_TRIAGE                 0
FECHA_INGRESO                0
FECHA_ATENCION               0
TIEMPO_DGTURNO_A_TRIAGE      0
TIEMPO_TRIAGE_A_INGRESO      0
TIEMPO_INGRESO_A_CONSULTA    0
TIEMPO_TOTAL                 0
Tiempo_Minutos_Total         0
CENTRO_ATENCION              0
CLASIFICACION_TRIAGE         0
PACIENTE_#_DOCUMENTO         0
EDAD                         0
EDAD_RANGO                   0
SEXO                         0
RÉGIMEN PACIENTE             0
NOMBRE_ENTIDAD               0
MEDICO                       0
AÑO                          0
MES                          0
DIA_SEMANA                   0
HOUR                         0
Turnos                       0
TIME                         0
DIA                          0
dtype: int64
```

[ ]:
```
df.dtypes
```

[ ]:
```
FECHA_LLEGADA                object
FECHA_TRIAGE                 object
FECHA_INGRESO                object
FECHA_ATENCION               object
TIEMPO_DGTURNO_A_TRIAGE      object
TIEMPO_TRIAGE_A_INGRESO      object
```

```
TIEMPO_INGRESO_A_CONSULTA      object
TIEMPO_TOTAL                   object
Tiempo_Minutos_Total           object
CENTRO_ATENCION              category
CLASIFICACION_TRIAGE           object
PACIENTE_#_DOCUMENTO           object
EDAD                           object
EDAD_RANGO                     object
SEXO                           object
RÉGIMEN PACIENTE               object
NOMBRE_ENTIDAD                 object
MEDICO                         object
AÑO                            object
MES                             int64
DIA_SEMANA                   category
HOUR                            int64
Turnos                       category
TIME                           object
DIA                             int64
Tiempo_Total                 float64
dtype: object
```

##Explorar relaciones entre los datos

Variables numéricas continuas:

Las variables numéricas continuas son variables que pueden contener cualquier valor dentro de cierto rango. Las variables numéricas continuas pueden tener el tipo int64 o float64. Una excelente manera de visualizar estas variables es mediante el uso de diagramas de dispersión con líneas ajustadas.

Para comenzar a comprender la relación (lineal) entre una variable individual y el precio. Podemos hacer esto usando regplot, que traza el diagrama de dispersión más la línea de regresión ajustada para los datos.

Vamos a importar las librerías Matplotlib y Seaborn para la visualización de datos. Les ponemos un alias plt y sns para que sea más fácil su uso:

[ ]:
```
import matplotlib.pyplot as plt
import seaborn as sns
```

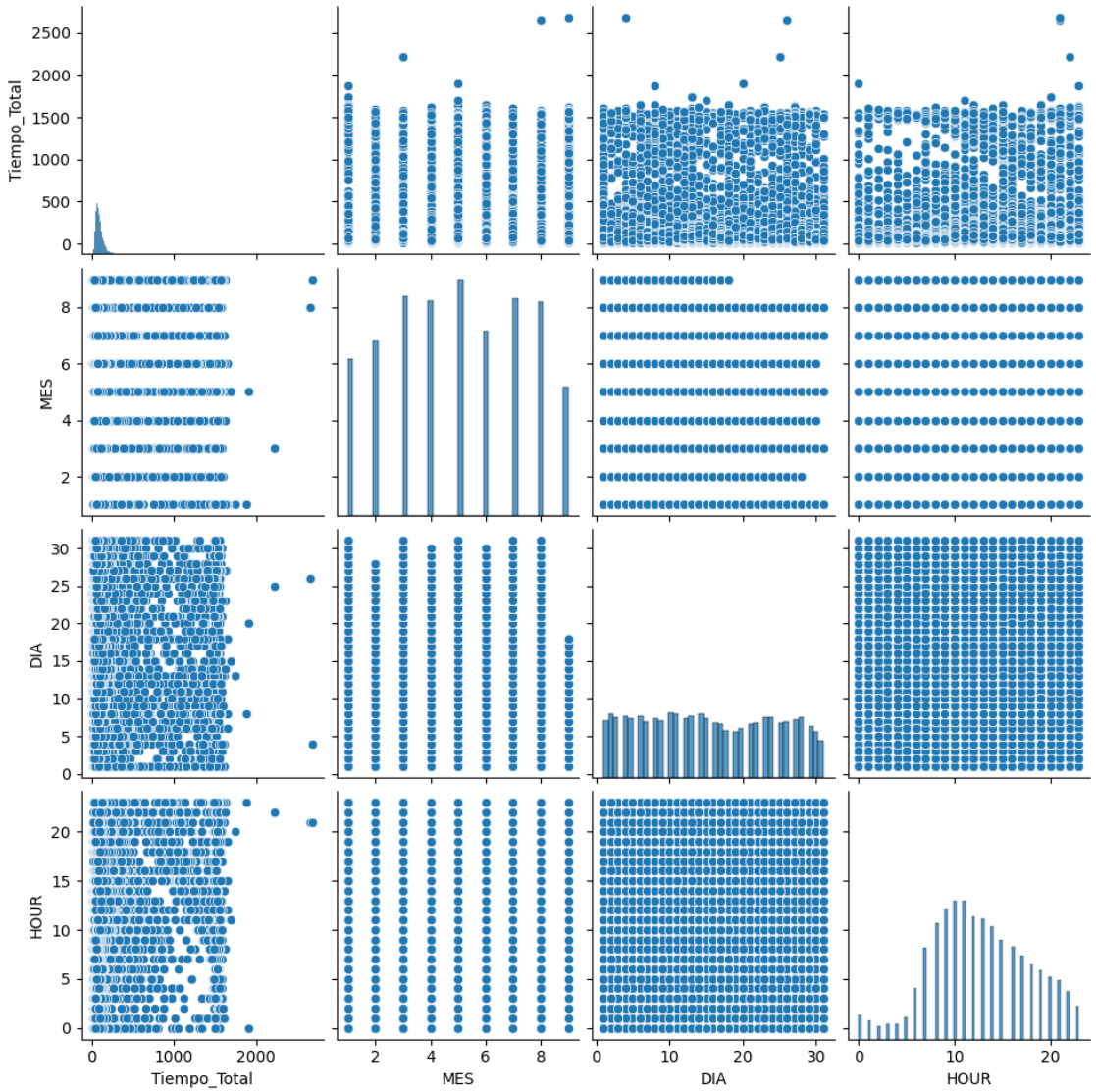
El gráfico seaborn.pairplot permite visualizar todas las variables numéricas y la combinación entre ellas. Pudiendo identificar facilmente si existen relaciones de dependencia entre algunas variables.

[ ]:
```
#sns.pairplot(df)
df['Tiempo_Total'] = df['Tiempo_Minutos_Total'].str.replace(',', '.',␣
 ↪regex=True)
df['Tiempo_Total'] = pd.to_numeric(df['Tiempo_Total'], errors='coerce')
df['Turnos'] = df['Turnos'].astype('category')
df['DIA_SEMANA'] = df['DIA_SEMANA'].astype('category')
```

```
df['CENTRO_ATENCION'] = df['CENTRO_ATENCION'].astype('category')
df['MES'] = pd.to_numeric(df['MES'], errors='coerce')
df['HOUR'] = pd.to_numeric(df['HOUR'], errors='coerce')
df['DIA'] = pd.to_numeric(df['DIA'], errors='coerce')

sub_df = df[['Tiempo_Total','MES', 'DIA','HOUR']]

sns.pairplot(sub_df)
```

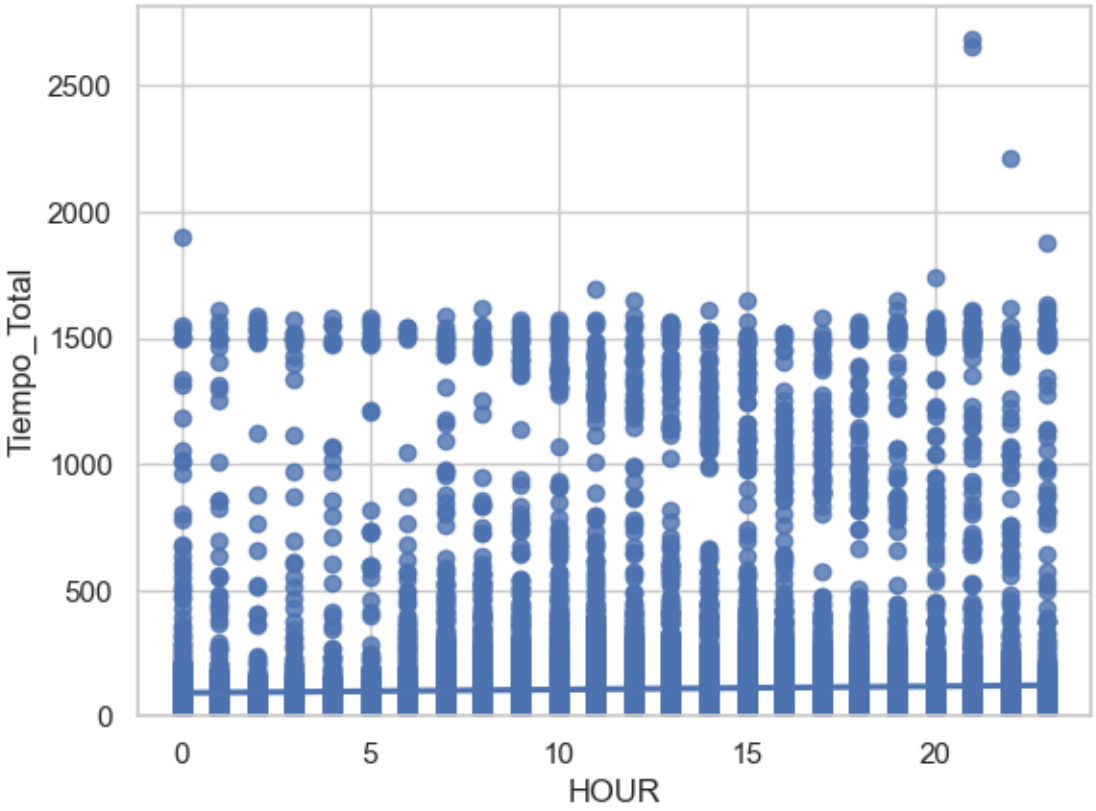
```
c:\Users\Victor.Gomez\AppData\Local\Programs\Python\Python311\Lib\site-
packages\seaborn\axisgrid.py:118: UserWarning: The figure layout has changed to
tight
  self._figure.tight_layout(*args, **kwargs)
```

[ ]: <seaborn.axisgrid.PairGrid at 0x1b6b1f7b010>

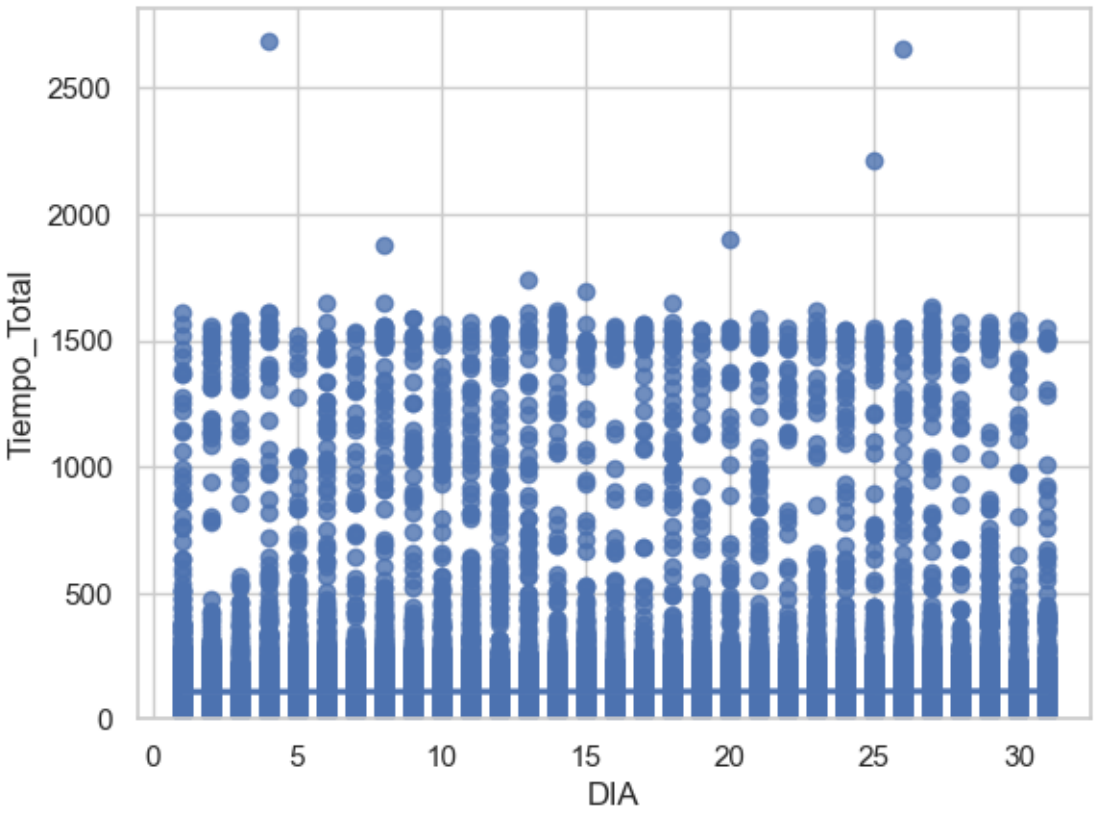


Visualicemos el diagrama de dispersión de tamaño del motor (engine-size) y precio (price)

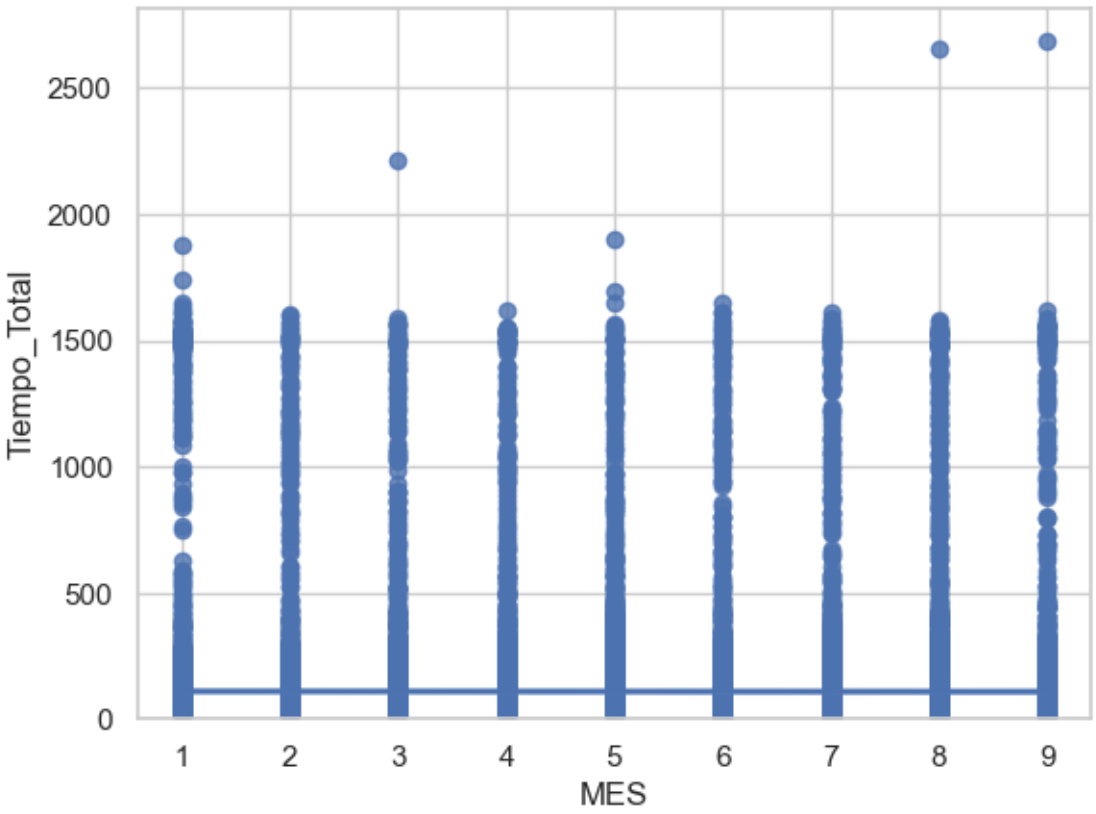
[ ]:
```
# Luego, crea el gráfico de regresión
sns.set(style="whitegrid")
sns.regplot(x="HOUR", y="Tiempo_Total", data=df)
plt.ylim(0,)
plt.show()
```



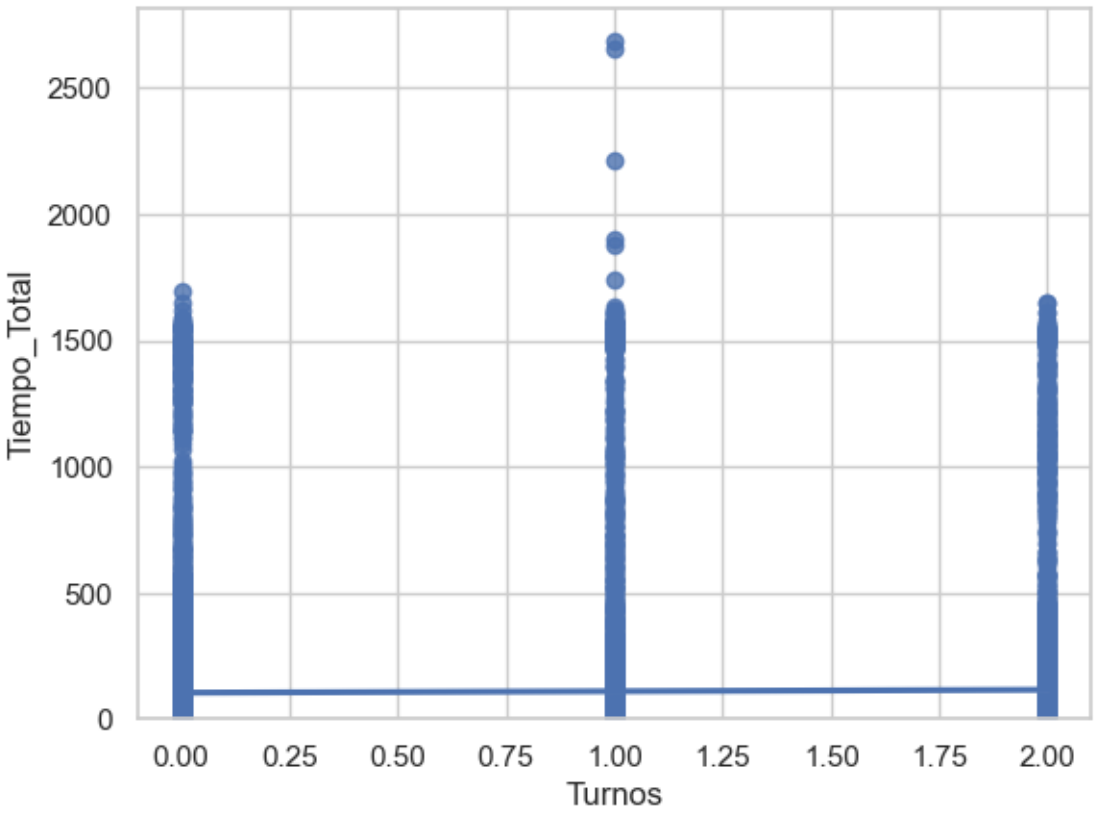
[ ]:
```
# Luego, crea el gráfico de regresión
sns.set(style="whitegrid")
sns.regplot(x="DIA", y="Tiempo_Total", data=df)
plt.ylim(0,)
plt.show()
```



[ ]:
```
# Luego, crea el gráfico de regresión
sns.set(style="whitegrid")
sns.regplot(x="MES", y="Tiempo_Total", data=df)
plt.ylim(0,)
plt.show()
```



[ ]:
```
# Primero, asegúrate de que tus variables categóricas estén ordenadas
df['Turnos'] = df['Turnos'].astype('category')
df['Turnos'] = df['Turnos'].cat.as_ordered()

# Convierte la columna categórica en una variable numérica
df['Turnos'] = df['Turnos'].cat.codes

# Luego, crea el gráfico de regresión
sns.set(style="whitegrid")
sns.regplot(x="Turnos", y="Tiempo_Total", data=df)
plt.ylim(0,)
plt.show()
```



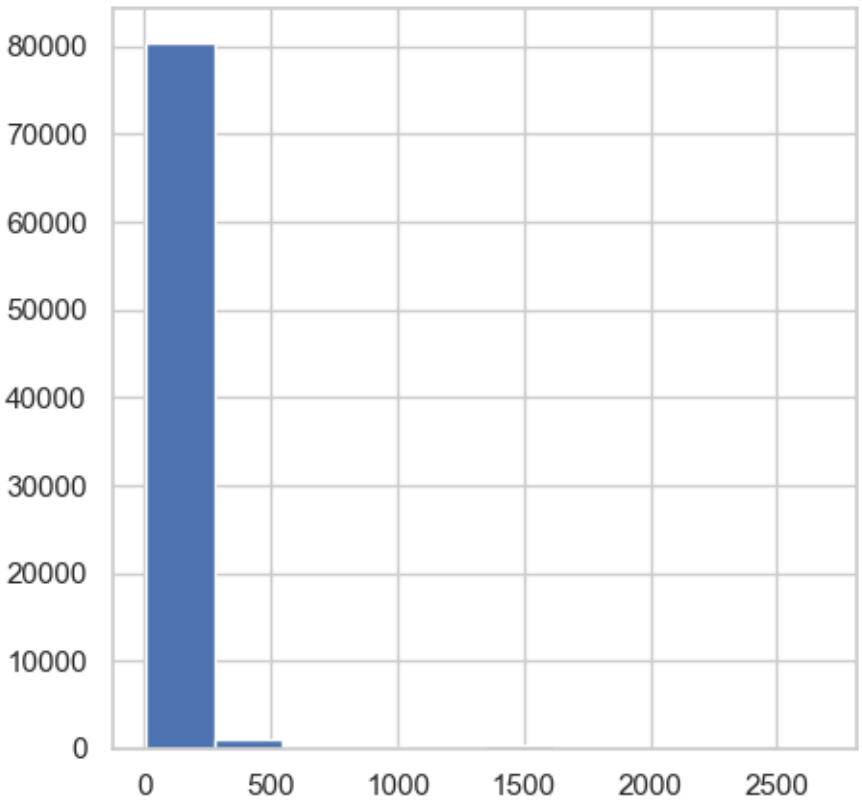
A medida que aumenta el tamaño del motor, aumenta el precio: esto indica una correlación directa positiva entre estas dos variables. El tamaño del motor parece ser un buen predictor de precio ya que la línea de regresión es casi una línea diagonal perfecta.

Visualicemos ahora highway-mpg y price. A medida que aumenta highway-mpg, el precio baja: esto indica un relación inversa/negativa entre estas dos variables. highway-mpg podría predecir el precio.

El gráfico de histograma también permite realizar un análisis sobre la variable numérica. Se utiliza el tipo de gráfico de Matplotlib hist sobre la variable age.

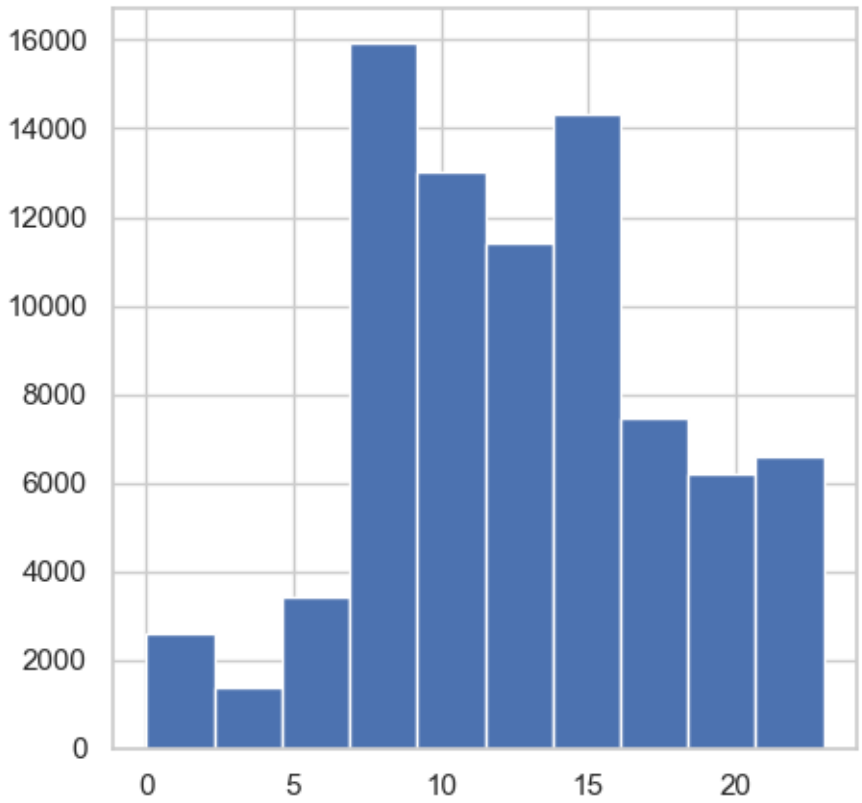
[ ]:
```
df['Tiempo_Total'].hist(figsize = (5,5))
plt.show
```

[ ]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```



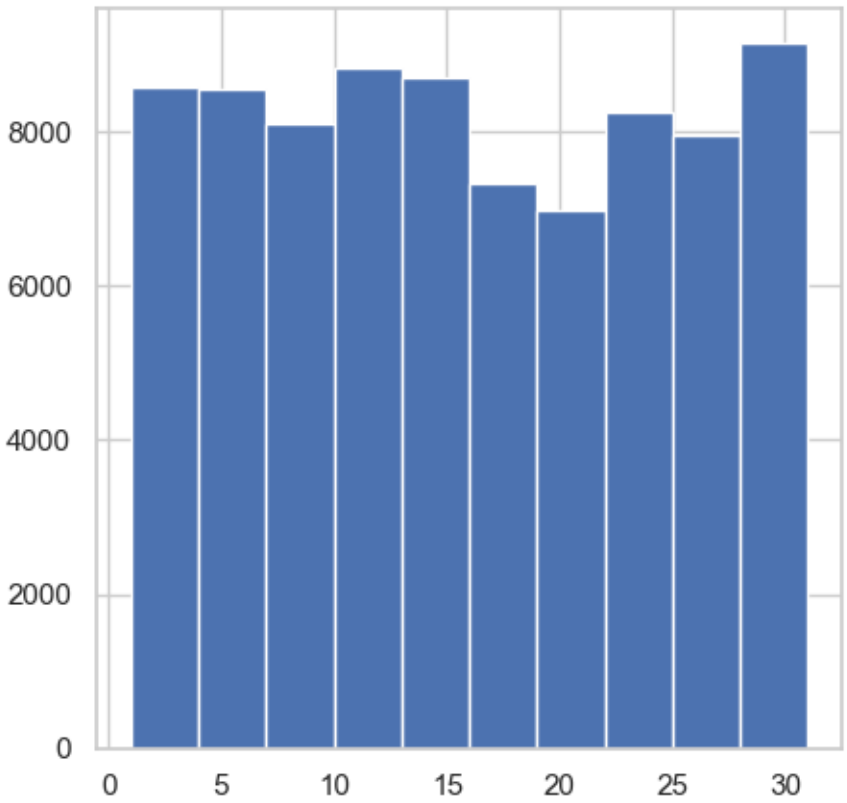
[ ]:
```
df['HOUR'].hist(figsize = (5,5))
plt.show
```

[ ]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```



[ ]:
```
df['DIA'].hist(figsize = (5,5))
plt.show
```

[ ]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```



Variables categóricas

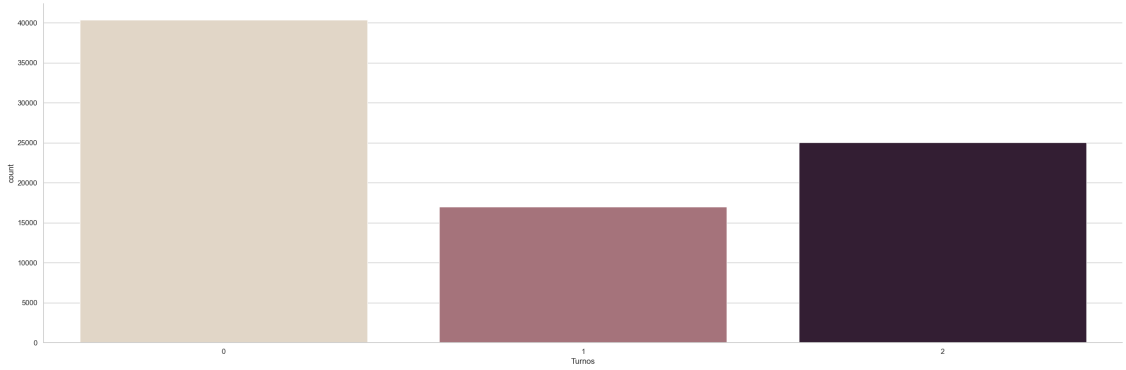
Estas son variables que describen una 'característica' de una unidad de datos y se seleccionan de un pequeño grupo de categorías. Las variables categóricas pueden tener el tipo objeto o int64. Una buena forma de visualizar variables categóricas es mediante el uso de diagramas de caja.

Mostremos un solo atributo, y contemos cuantos ejemplos hay de cada categoría. En este caso el atributo make que es la marca del carro:

[ ]:
```
sns.set(style="whitegrid")
sns.catplot(x="Turnos", kind="count", palette="ch:.25", data=df, height = 8,␣
 ↪aspect = 3)
```

```
c:\Users\Victor.Gomez\AppData\Local\Programs\Python\Python311\Lib\site-
packages\seaborn\axisgrid.py:118: UserWarning: The figure layout has changed to
tight
  self._figure.tight_layout(*args, **kwargs)
```

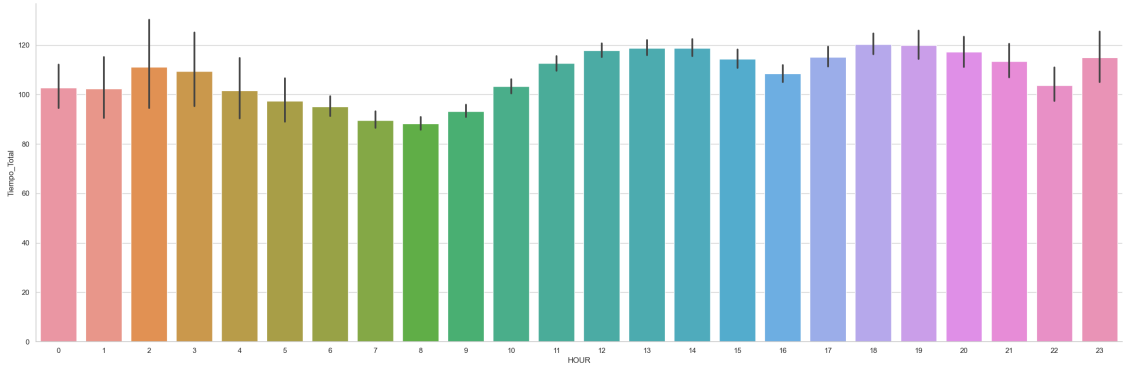
[ ]:
```
<seaborn.axisgrid.FacetGrid at 0x1b6b9ffa810>
```



[ ]:
```
sns.set(style="whitegrid")
sns.catplot(x="HOUR", y="Tiempo_Total", hue_order="class", kind="bar", data=df,␣
 ↪height = 8, aspect = 3)
```

```
c:\Users\Victor.Gomez\AppData\Local\Programs\Python\Python311\Lib\site-
packages\seaborn\axisgrid.py:118: UserWarning: The figure layout has changed to
tight
  self._figure.tight_layout(*args, **kwargs)
```

[ ]:
```
<seaborn.axisgrid.FacetGrid at 0x1b6ba008e50>
```

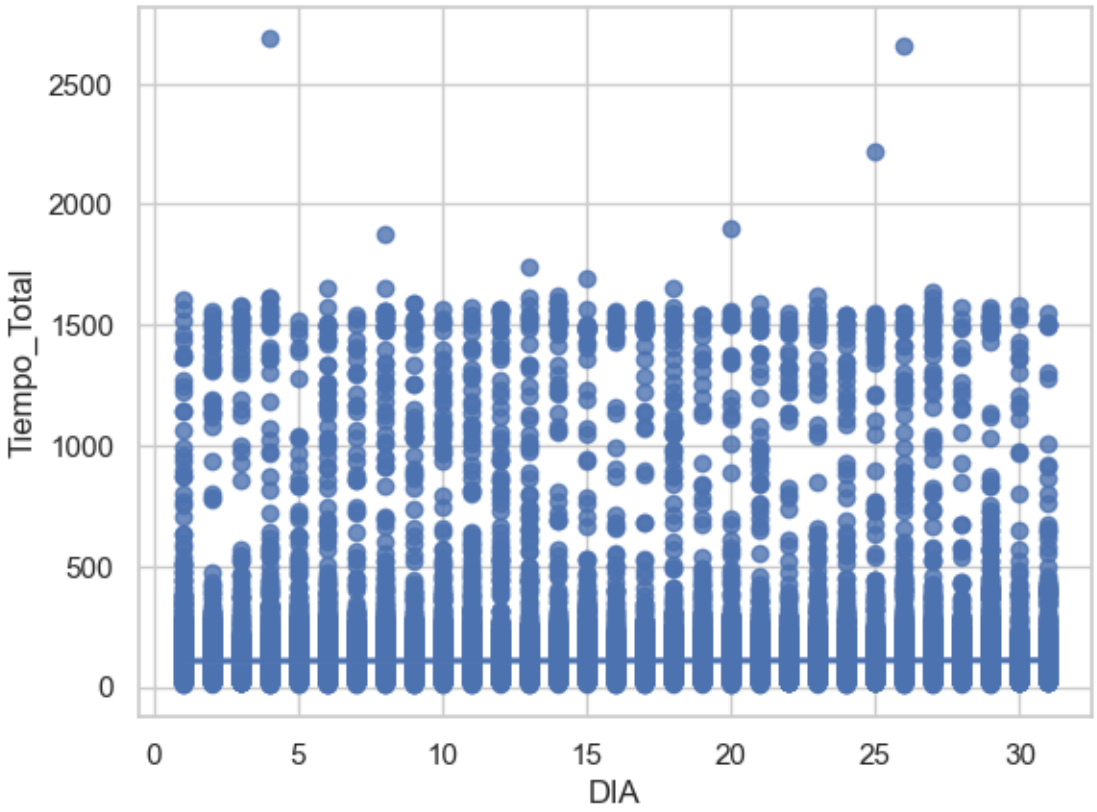


AHORA TÚ:

Visualice las variables stroke y price utilizando regplot. ¿Qué tipo de relación hay entre las variables?

[ ]:
```
# Escribe tu código aquí y presiona Shift+Enter para ejecutar

sns.set(style="whitegrid")
sns.regplot(x="DIA", y="Tiempo_Total", data=df)
```

[ ]: <Axes: xlabel='DIA', ylabel='Tiempo_Total'>



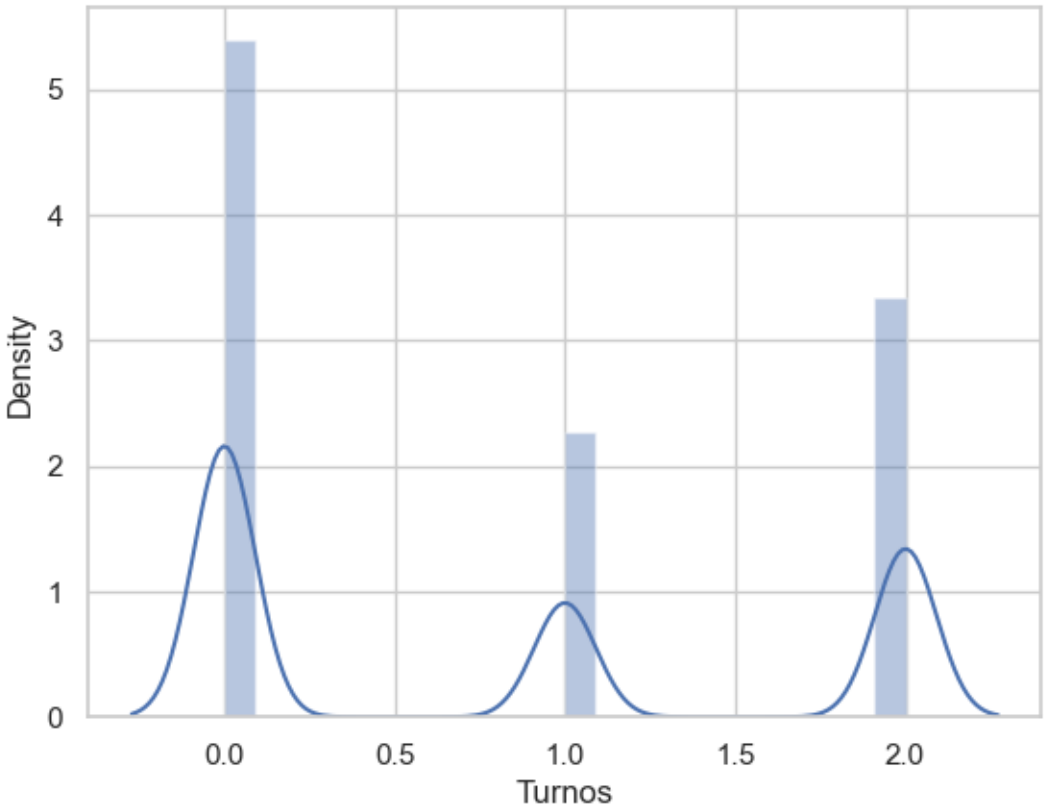
##Graficar las estadísticas

Podemos identificar la simentría de los datos utilizando gráficos de histograma:

[ ]:
```
#Histograma del atributo "length"
sns.distplot(df.Turnos)
```

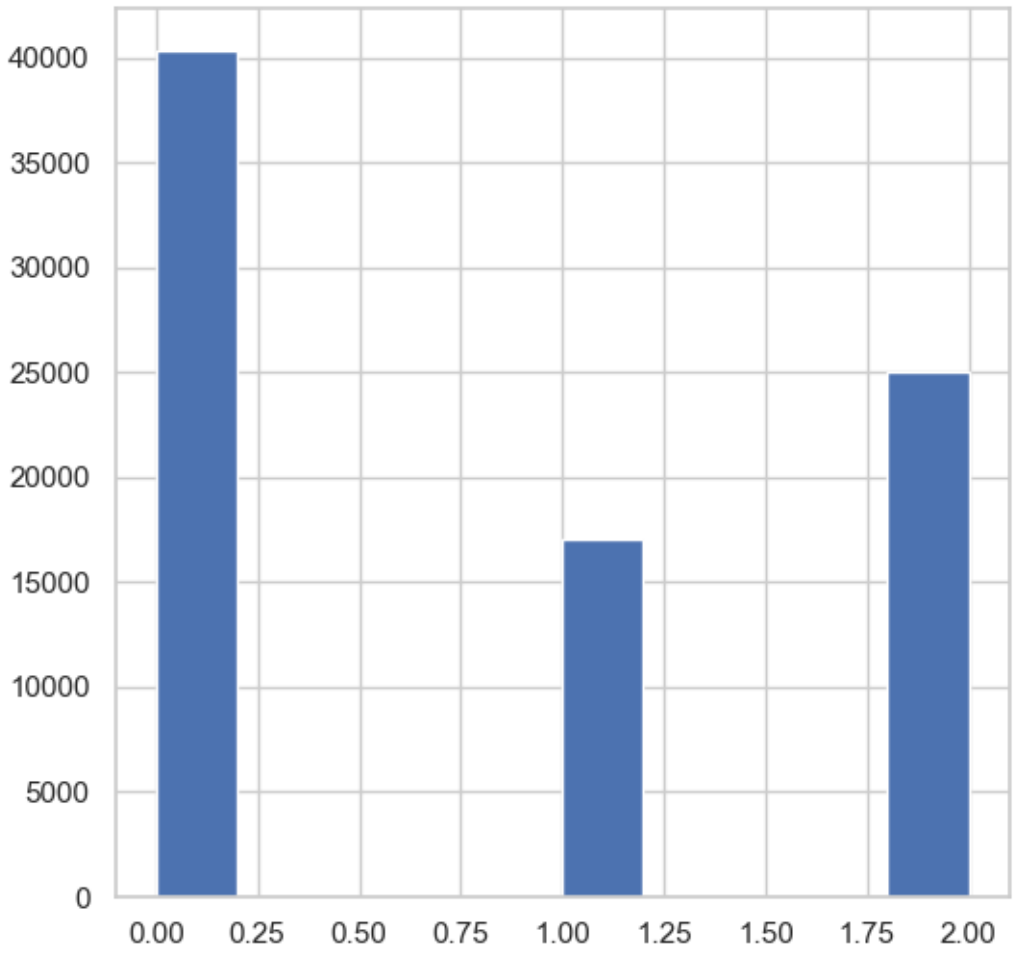
```
C:\Users\Victor.Gomez\AppData\Local\Temp\ipykernel_14576\1764428100.py:2:
UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df.Turnos)
```

[ ]: <Axes: xlabel='Turnos', ylabel='Density'>



[ ]:
```
df['Turnos'].hist(figsize = (6,6))
plt.show
```

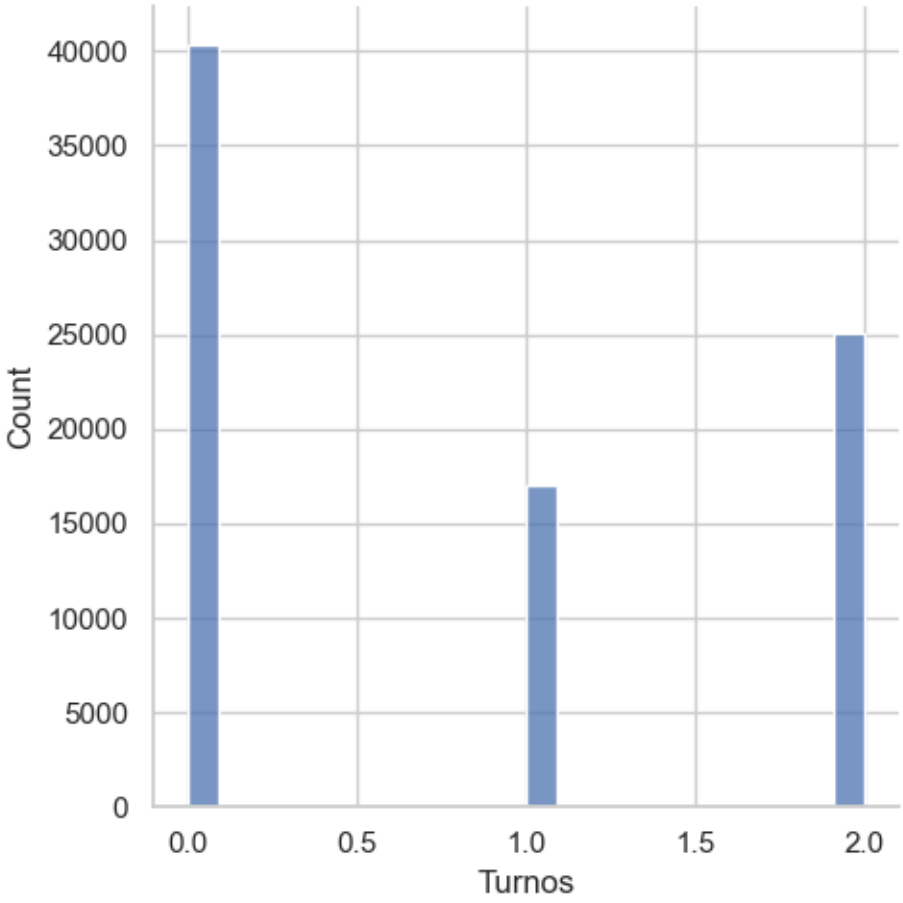
[ ]: <function matplotlib.pyplot.show(close=None, block=None)>



[ ]:
```
sns.displot(df['Turnos'])
```

```
c:\Users\Victor.Gomez\AppData\Local\Programs\Python\Python311\Lib\site-
packages\seaborn\axisgrid.py:118: UserWarning: The figure layout has changed to
tight
  self._figure.tight_layout(*args, **kwargs)
```

[ ]:
```
<seaborn.axisgrid.FacetGrid at 0x1b6ba034850>
```



[ ]:
```
mean = df['Turnos'].mean()
median = df['Turnos'].median()
mode = df['Turnos'].mode()
skew = df['Turnos'].skew()
kurt = df['Turnos'].kurt()
print("La media es:", mean)
print("La mediana es:", median)
print("La moda es:", mode)
print("El sesgo es:", skew)
print("La kurtosis es:", kurt)
```

```
La media es: 0.8142229664142885
La mediana es: 1.0
La moda es: 0    0
Name: Turnos, dtype: int8
El sesgo es: 0.3684999729645169
La kurtosis es: -1.5835310691072397
```

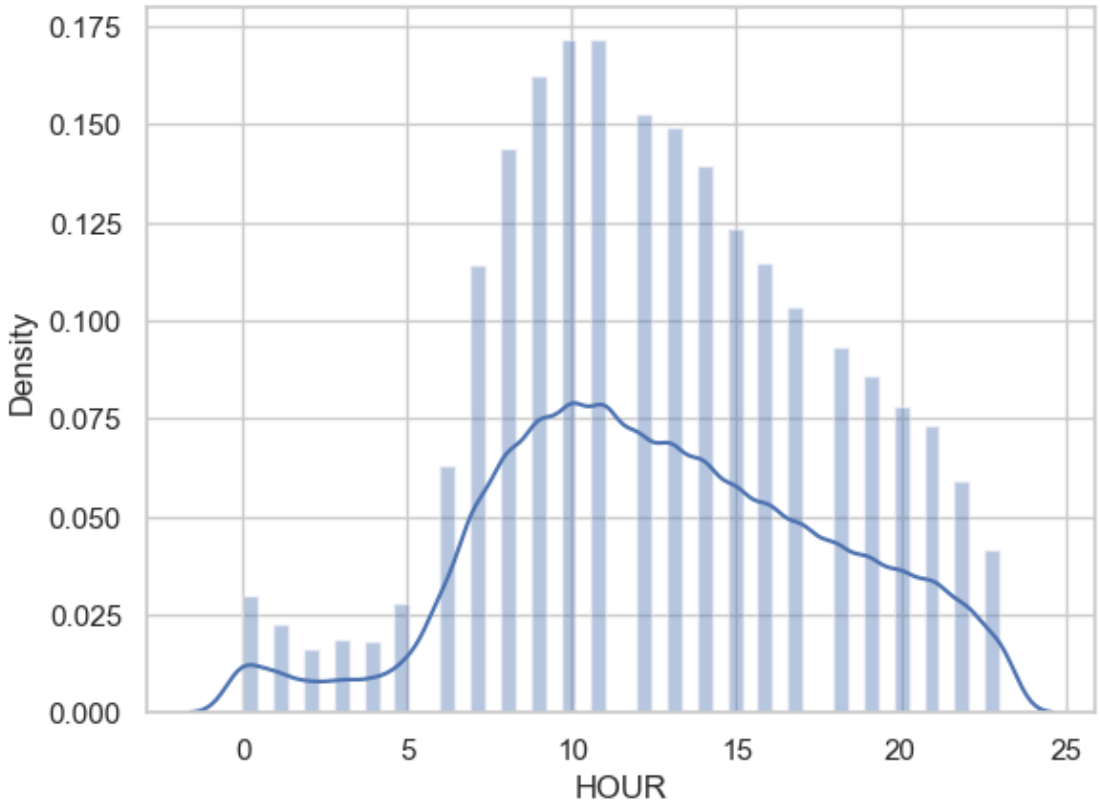
AHORA TÚ:

Seleccione una variable y visualice su histograma. ¿Qué tipo de simentría tiene?

[ ]:
```
# Escribe tu código aquí y presiona Shift+Enter para ejecutar
sns.distplot(df.HOUR)
```

```
C:\Users\Victor.Gomez\AppData\Local\Temp\ipykernel_14576\8436205.py:2:
UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df.HOUR)
```

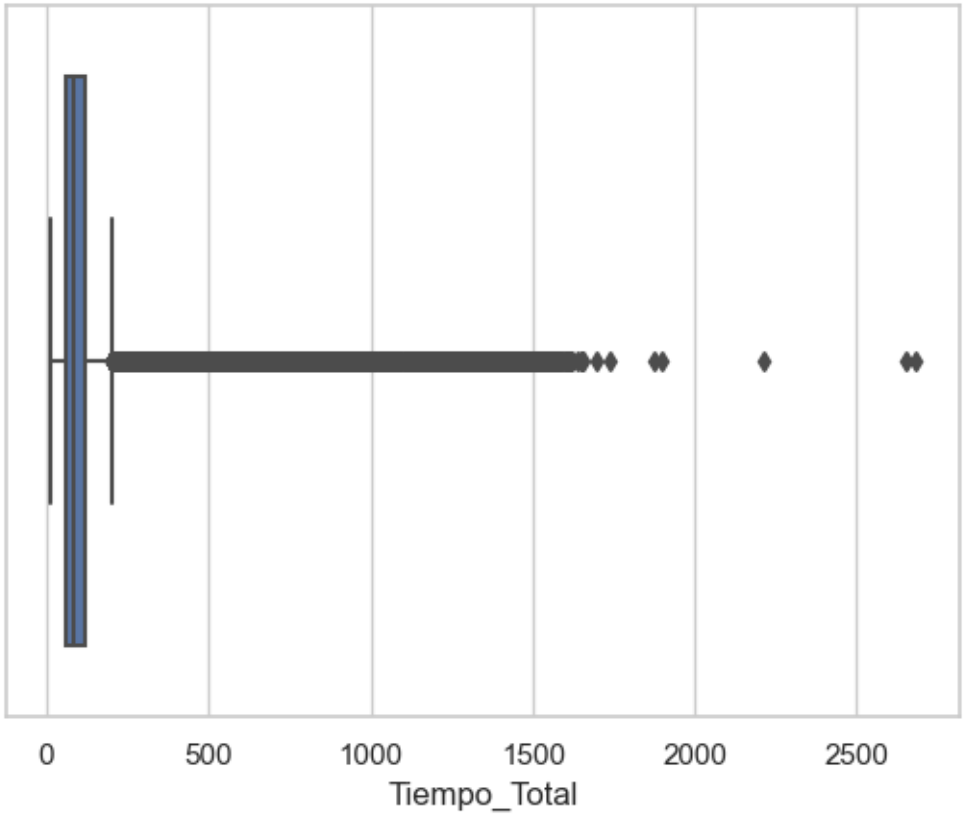
[ ]: `<Axes: xlabel='HOUR', ylabel='Density'>`



La dispersión de datos se puede comprobar también mediante los gráficos de tipo boxplot:

[ ]:
```
sns.boxplot(x=df.Tiempo_Total)
```

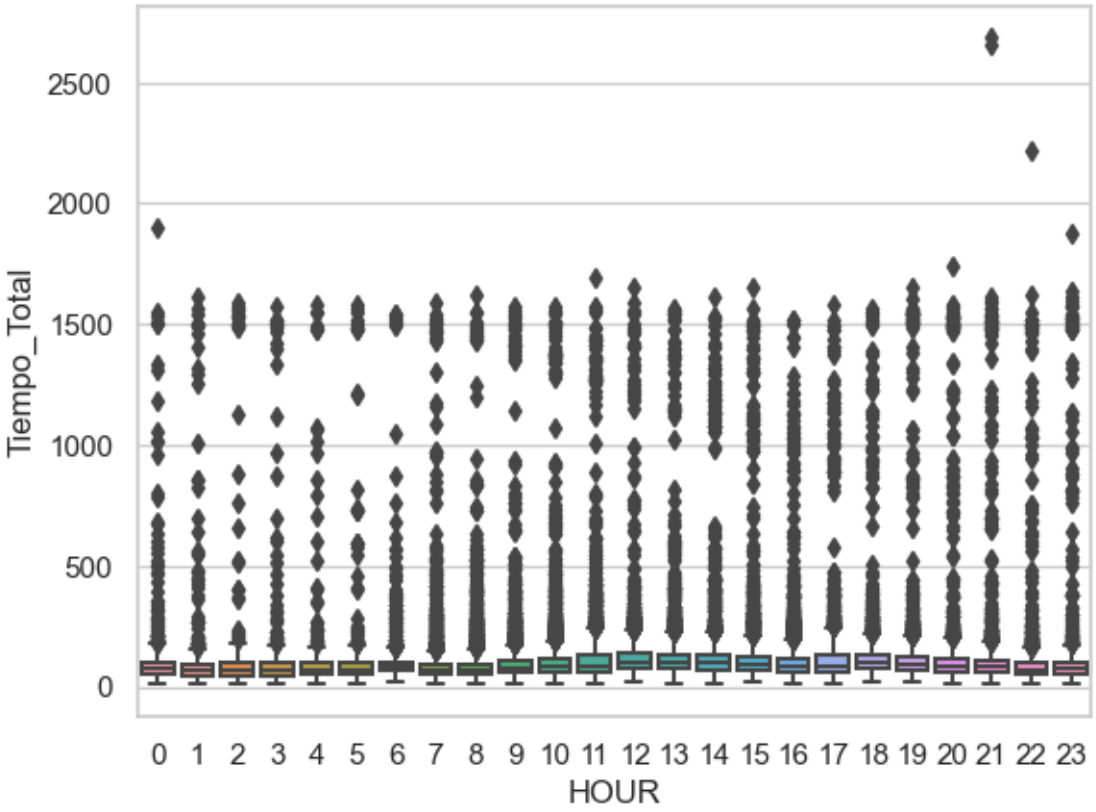
[ ]:
```
<Axes: xlabel='Tiempo_Total'>
```



Representar más de una gráfico tipo boxplot permite comparar la dispersión de los datos al poder ver los resultados de forma conjunta. Veamos la relación entre body-style y price.

[ ]:
```
sns.boxplot(x="HOUR", y="Tiempo_Total", data=df)
```

[ ]:
```
<Axes: xlabel='HOUR', ylabel='Tiempo_Total'>
```



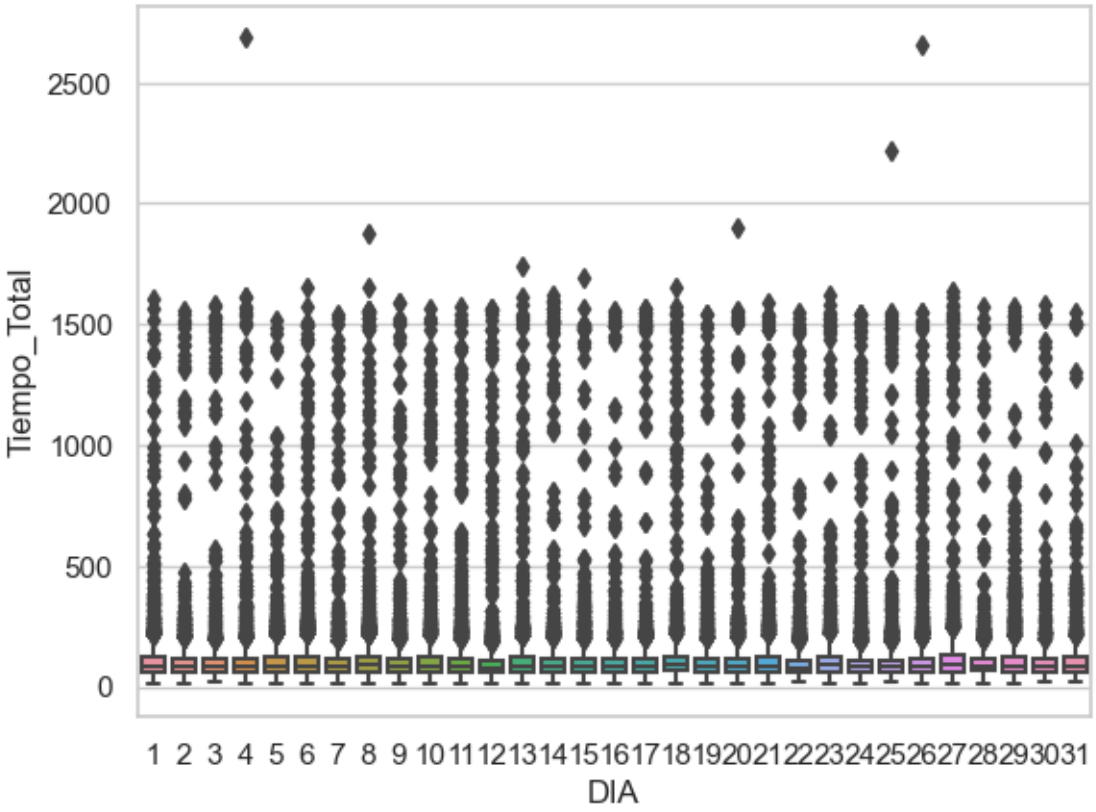
Vemos que las distribuciones de precios entre las diferentes categorías de estilo de cuerpo tienen una superposición significativa, por lo que el estilo de cuerpo no sería un buen predictor del precio.

AHORA TÚ:

Visualice las variables engine-location y price utilizando un gráfico de BoxPlot

[ ]:
```
# Escribe tu código aquí y presiona Shift+Enter para ejecutar
sns.boxplot(x="DIA", y="Tiempo_Total", data=df)
```

[ ]:
```
<Axes: xlabel='DIA', ylabel='Tiempo_Total'>
```



##Exportar los datos

De la misma forma, Pandas nos permite guardar el conjunto en formato CSV con el método dataframe.to_csv(), puede añadir la ruta al archivo y el nombre con comillas dentro de los corchetes.

Por ejemplo, si guarda el dataframe df como automobile.csv en su equipo local o en este caso dentro de Google Colab, podría usar la sintaxis siguiente:

[ ]:
```
path="URGENCIAS.csv"
df.to_csv(path)
```

Podemos leer y guardar con otros formatos y usar funciones similares a pd.read_csv() y df.to_csv() para otros formatos de datos, las funciones se muestran en la siguiente tabla:

Data Formate	Read	Save
csv	`pd.read_csv()`	`df.to_csv()`
json	`pd.read_json()`	`df.to_json()`
excel	`pd.read_excel()`	`df.to_excel()`
hdf	`pd.read_hdf()`	`df.to_hdf()`
sql	`pd.read_sql()`	`df.to_sql()`
...	...	...

##Links de ayuda interesantes:

Graficos Seaborn: https://seaborn.pydata.org/examples/index.html

Plotting with categorical data: https://seaborn.pydata.org/tutorial/categorical.html

Visualizing statistical relationships: https://seaborn.pydata.org/tutorial/relational.html

Funciones Generales y Ayuda Pandas: https://pandas.pydata.org/pandas-docs/stable/reference/general_functions.html

Ayuda de Pandas para función read_csv: https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read_csv.html?highlight=pandas%20read_csv#pandas.read_csv

How to Perform Exploratory Data Analysis with Seaborn: https://towardsdatascience.com/how-to-perform-exploratory-data-analysis-with-seaborn-97e3413e841d

de-validacion-de-datos-y-variables

November 23, 2023

[49]:
```
import pandas as pd
import os
import re
```

[50]:
```
# ruta de archivos
files = os.listdir("c:\\archivos\proyecto")
os.chdir(r'C:\archivos\proyecto')

if not os.path.exists('Errores'):
    os.makedirs('Errores')

if not os.path.exists('Buenos'):
    os.mkdir('Buenos')

df_hearth = pd.read_csv('indicadores de urgencias2.txt', sep=';')

#df_hearth = pd.read_excel("indicadores de urgencias.xlsx")
```

[51]:
```
df_hearth
# df_hearth.info()
# df_hearth.columns
```

[51]:
```
                 FECHA_LLEGADA             FECHA_TRIAGE  \
0      2023-01-01 01:20:23.853  2023-01-01 01:28:01.847
1      2023-01-01 01:29:46.050  2023-01-01 01:48:03.070
2      2023-01-01 03:15:35.623  2023-01-01 03:23:01.990
3      2023-01-01 05:54:53.563  2023-01-01 06:00:07.943
4      2023-01-01 06:37:27.237  2023-01-01 07:52:31.687
…                          …                         …
82411  2023-09-18 04:44:41.970  2023-09-18 04:53:22.553
82412  2023-09-18 06:17:00.573  2023-09-18 06:28:43.040
82413  2023-09-18 06:21:37.273  2023-09-18 07:00:57.420
82414  2023-09-18 06:25:33.483  2023-09-18 06:42:02.883
82415  2023-09-18 07:14:58.180  2023-09-18 07:30:50.643

                 FECHA_INGRESO           FECHA_ATENCION  \
0      2023-01-01 01:29:41.210  2023-01-01 02:00:07.590
```

```
1      2023-01-01 01:49:40.973  2023-01-01 02:02:53.663
2      2023-01-01 03:23:39.793  2023-01-01 03:30:21.233
3      2023-01-01 06:02:07.320  2023-01-01 06:26:17.050
4      2023-01-01 07:52:37.717  2023-01-01 09:31:15.597
…                          …                        …
82411  2023-09-18 05:05:51.423  2023-09-18 06:09:35.867
82412  2023-09-18 06:35:38.213  2023-09-18 07:40:45.957
82413  2023-09-18 07:16:45.907  2023-09-18 08:27:27.337
82414  2023-09-18 06:51:35.970  2023-09-18 07:28:28.290
82415  2023-09-18 07:34:08.370  2023-09-18 07:49:18.440

      TIEMPO_DGTURNO_A_TRIAGE TIEMPO_TRIAGE_A_INGRESO  \
0                     0:07:38                 0:01:40
1                     0:18:17                 0:01:37
2                     0:07:26                 0:00:38
3                     0:05:14                 0:02:00
4                     1:15:04                 0:00:06
…                       …                       …
82411                 0:08:41                 0:12:29
82412                 0:11:43                 0:06:55
82413                 0:39:20                 0:15:48
82414                 0:16:29                 0:09:33
82415                 0:15:52                 0:03:18

      TIEMPO_INGRESO_A_CONSULTA TIEMPO_TOTAL Tiempo_Minutos_Total  \
0                       0:30:26      0:39:44                39,73
1                       0:13:13      0:33:07                33,12
2                       0:06:42      0:14:46                14,77
3                       0:24:10      0:31:24                31,40
4                       1:38:38      2:53:48               173,80
…                         …          …                        …
82411                   1:03:44      1:24:54                84,90
82412                   1:05:07      1:23:45                83,75
82413                   1:10:42      2:05:50               125,83
82414                   0:36:53      1:02:55                62,92
82415                   0:15:10      0:34:20                34,33

                           CENTRO_ATENCION  …    EDAD_RANGO       SEXO  \
0                       TN - HOSPITAL TUNAL  …      JUVENTUD  MASCULINO
1                     ME - HOSPITAL MEISSEN  …      JUVENTUD   FEMENINO
2      UC - CENTRO DE SALUD SANTA LIBRADA I  …      JUVENTUD  MASCULINO
3      UC - CENTRO DE SALUD SANTA LIBRADA I  …        ADULTO  MASCULINO
4                       TN - HOSPITAL TUNAL  …      JUVENTUD  MASCULINO
…                                         …  …           …         …
82411                 ME - HOSPITAL MEISSEN  …      JUVENTUD   FEMENINO
82412                 ME - HOSPITAL MEISSEN  …  ADULTO MAYOR  MASCULINO
82413                 ME - HOSPITAL MEISSEN  …  ADULTO MAYOR  MASCULINO
```

```
82414             UB - CENTRO DE SALUD USME  …      JUVENTUD  MASCULINO
82415                   TN - HOSPITAL TUNAL  …   ADOLECENCIA   FEMENINO

       RÉGIMEN PACIENTE NOMBRE_ENTIDAD  MEDICO   AÑO MES  DIA_SEMANA  HOUR  \
0             SUBSIDIADO          EPSS34  11065  2023   1     DOMINGO     1
1             SUBSIDIADO          EPSS34   8861  2023   1     DOMINGO     1
2           CONTRIBUTIVO          EPS002   5855  2023   1     DOMINGO     3
3             SUBSIDIADO          EPSS34  11072  2023   1     DOMINGO     5
4             SUBSIDIADO          EPSS34   1239  2023   1     DOMINGO     6
…                    …               …     …   …  ..        …   …
82411       CONTRIBUTIVO          EPS017   7844  2023   9       LUNES     4
82412         SUBSIDIADO          EPSC34   6204  2023   9       LUNES     6
82413       CONTRIBUTIVO          EPS041   9951  2023   9       LUNES     6
82414         SUBSIDIADO          EPSC34   4030  2023   9       LUNES     6
82415         SUBSIDIADO          EPSC34   1239  2023   9       LUNES     7

             Turnos
0       TURNO_NOCHE
1       TURNO_NOCHE
2       TURNO_NOCHE
3       TURNO_NOCHE
4       TURNO_NOCHE
…               …
82411   TURNO_NOCHE
82412   TURNO_NOCHE
82413   TURNO_NOCHE
82414   TURNO_NOCHE
82415  TURNO_MAÑANA

[82416 rows x 23 columns]
```

[11]:
```
# transformar datos
df_hearth['FECHA_INGRESO'].fillna('FECHA_TRIAGE', inplace=True)
df_hearth['FECHA_ATENCION'].fillna('FECHA_INGRESO', inplace=True)
df_hearth['FECHA_TRIAGE'] = pd.to_datetime(df_hearth['FECHA_TRIAGE'],␣
 ↪format='%d/%m/%Y %H:%M')
df_hearth['FECHA_INGRESO'] = pd.to_datetime(df_hearth['FECHA_INGRESO'],␣
 ↪format='%d/%m/%Y %H:%M')
df_hearth['FECHA_LLEGADA'] = pd.to_datetime(df_hearth['FECHA_LLEGADA'],␣
 ↪format='%d/%m/%Y %H:%M')
```

[10]:
```
for i, row in df_hearth.iterrows():
    try:
        df_hearth.at[i, 'FECHA_LLEGADA'] = pd.to_datetime(row['FECHA_LLEGADA'],␣
 ↪format='%d/%m/%Y %H:%M')
    except ValueError:
```

```
        # Manejar el error, por ejemplo, reemplazar el valor incorrecto con una␣
 ↪fecha válida
        df_hearth.at[i, 'FECHA_LLEGADA'] = pd.to_datetime('01/01/2000 00:00',␣
 ↪format='%d/%m/%Y %H:%M')
```

[8]:
```
for i, row in df_hearth.iterrows():
    try:
        df_hearth.at[i, 'FECHA_INGRESO'] = pd.to_datetime(row['FECHA_INGRESO'],␣
 ↪format='%d/%m/%Y %H:%M')
    except ValueError:
        # Manejar el error, por ejemplo, reemplazar el valor incorrecto con una␣
 ↪fecha válida
        df_hearth.at[i, 'FECHA_INGRESO'] = pd.to_datetime('01/01/2000 00:00',␣
 ↪format='%d/%m/%Y %H:%M')
```

[6]:
```
df_hearth['FECHA_CONSULTA'] = pd.to_datetime(df_hearth['FECHA_CONSULTA'],␣
 ↪format='%d/%m/%Y %H:%M')
```

[ ]:
```
# Especifica el formato correcto de las fechas
df_hearth['TIME'] = pd.to_datetime(df_hearth['FECHA_LLEGADA'], format='%d/%m/%Y␣
 ↪%H:%M:%S')

# Extrae el dia
df_hearth['DIA'] = df_hearth['TIME'].dt.day
```

[7]:
```
# Especifica el formato correcto de las fechas
df_hearth['TIME'] = pd.to_datetime(df_hearth['FECHA_LLEGADA'], format='%d/%m/%Y␣
 ↪%H:%M')

# Extrae la hora
df_hearth['HOUR'] = df_hearth['TIME'].dt.hour

# Extrae el mes
df_hearth['MES'] = df_hearth['TIME'].dt.month

# Extrae el dia
df_hearth['DIA'] = df_hearth['TIME'].dt.day

# Extrae el año
df_hearth['AÑO'] = df_hearth['TIME'].dt.year
```

[8]:
```
# crea columna turnos
bins = [0, 6, 12, 18, float("inf")]
labels = ['MADRUGADA', 'MAÑANA', 'TARDE', 'NOCHE']
df_hearth['Turnos'] = pd.cut(df_hearth['HOUR'], bins=bins, labels=labels)
df_hearth['Turnos'].fillna('MADRUGADA', inplace=True)
```

[ ]:
```
# mostrar las columnas que quiero
'''
df1 = df_hearth.head(200)
numeric_columns = ['FECHA_LLEGADA', 'Turnos', 'HOUR']
df2 = df1[numeric_columns]
df2.to_csv('ind_urgencias.txt', index=False)
'''
```

[9]:
```
# Define la fecha que deseas buscar
fecha_a_buscar = pd.to_datetime('01/01/2000 00:00')

# Encuentra las filas que contienen la fecha deseada en 'FECHA_CONSULTA'
filas_a_actualizar = df_hearth[df_hearth['FECHA_CONSULTA'] == fecha_a_buscar]

# Reemplaza la fecha en 'FECHA_CONSULTA' con la fecha correspondiente en␣
 ↪'FECHA_INGRESO'
df_hearth.loc[filas_a_actualizar.index, 'FECHA_CONSULTA'] = df_hearth.
 ↪loc[filas_a_actualizar.index, 'FECHA_INGRESO']
```

[10]:
```
# Calcula la diferencia de tiempo y crea una nueva columna 'Tiempo_Consulta' en␣
 ↪minutos
df_hearth['Tiempo_Triage'] = (df_hearth['FECHA_TRIAGE'] -␣
 ↪df_hearth['FECHA_LLEGADA']).dt.total_seconds() / 60
df_hearth['Tiempo_Ingreso'] = (df_hearth['FECHA_INGRESO'] -␣
 ↪df_hearth['FECHA_TRIAGE']).dt.total_seconds() / 60
df_hearth['Tiempo_Consulta'] = (df_hearth['FECHA_CONSULTA'] -␣
 ↪df_hearth['FECHA_INGRESO']).dt.total_seconds() / 60
df_hearth['Tiempo_Minutos_Total'] = (df_hearth['Tiempo_Triage'] +␣
 ↪df_hearth['Tiempo_Ingreso'] + df_hearth['Tiempo_Consulta'])
```

[53]:
```
# trasnformar datos que no requiero
# df_hearth
df_hearth['CENTRO_ATENCION'] = df_hearth['CENTRO_ATENCION'].str.slice(0, 2)
df_hearth['CLASIFICACION_TRIAGE'] = df_hearth['CLASIFICACION_TRIAGE'].str.
 ↪slice(0, 1)
# df_hearth['Turnos'] = df_hearth['Turnos'].str.slice(3, 0)
df_hearth['Turnos'] = df_hearth['Turnos'].str[6:]
# df_hearth['PACIENTE_EDAD'] = df_hearth['PACIENTE_EDAD'].str.slice(0, 3)
# df_hearth['PACIENTE_EDAD'] = df_hearth['PACIENTE_EDAD'].apply(lambda x: re.
 ↪sub(r'\D', '', str(x)))
```

[12]:
```
df_hearth['DIA_SEMANA'] = df_hearth['TIME'].dt.day_name()
```

[46]:
```
df_hearth['Turnos'] = df_hearth['Turnos'].astype('category')
df_hearth['DIA_SEMANA'] = df_hearth['DIA_SEMANA'].astype('category')
```

[56]:
```
df_hearth.to_csv('ind_urgencias_final_2023.txt', sep=';', index=False)
```

[57]:
```
df_hearth
```

[57]:

	FECHA_LLEGADA	FECHA_TRIAGE
0	2023-01-01 01:20:23.853	2023-01-01 01:28:01.847
1	2023-01-01 01:29:46.050	2023-01-01 01:48:03.070
2	2023-01-01 03:15:35.623	2023-01-01 03:23:01.990
3	2023-01-01 05:54:53.563	2023-01-01 06:00:07.943
4	2023-01-01 06:37:27.237	2023-01-01 07:52:31.687
…	…	…
82411	2023-09-18 04:44:41.970	2023-09-18 04:53:22.553
82412	2023-09-18 06:17:00.573	2023-09-18 06:28:43.040
82413	2023-09-18 06:21:37.273	2023-09-18 07:00:57.420
82414	2023-09-18 06:25:33.483	2023-09-18 06:42:02.883
82415	2023-09-18 07:14:58.180	2023-09-18 07:30:50.643

	FECHA_INGRESO	FECHA_ATENCION
0	2023-01-01 01:29:41.210	2023-01-01 02:00:07.590
1	2023-01-01 01:49:40.973	2023-01-01 02:02:53.663
2	2023-01-01 03:23:39.793	2023-01-01 03:30:21.233
3	2023-01-01 06:02:07.320	2023-01-01 06:26:17.050
4	2023-01-01 07:52:37.717	2023-01-01 09:31:15.597
…	…	…
82411	2023-09-18 05:05:51.423	2023-09-18 06:09:35.867
82412	2023-09-18 06:35:38.213	2023-09-18 07:40:45.957
82413	2023-09-18 07:16:45.907	2023-09-18 08:27:27.337
82414	2023-09-18 06:51:35.970	2023-09-18 07:28:28.290
82415	2023-09-18 07:34:08.370	2023-09-18 07:49:18.440

	TIEMPO_DGTURNO_A_TRIAGE	TIEMPO_TRIAGE_A_INGRESO
0	0:07:38	0:01:40
1	0:18:17	0:01:37
2	0:07:26	0:00:38
3	0:05:14	0:02:00
4	1:15:04	0:00:06
…	…	…
82411	0:08:41	0:12:29
82412	0:11:43	0:06:55
82413	0:39:20	0:15:48
82414	0:16:29	0:09:33
82415	0:15:52	0:03:18

	TIEMPO_INGRESO_A_CONSULTA	TIEMPO_TOTAL	Tiempo_Minutos_Total
0	0:30:26	0:39:44	39,73
1	0:13:13	0:33:07	33,12
2	0:06:42	0:14:46	14,77

```
3                      0:24:10      0:31:24               31,40
4                      1:38:38      2:53:48              173,80
…                        …          …                      …
82411                  1:03:44      1:24:54               84,90
82412                  1:05:07      1:23:45               83,75
82413                  1:10:42      2:05:50              125,83
82414                  0:36:53      1:02:55               62,92
82415                  0:15:10      0:34:20               34,33

      CENTRO_ATENCION  …    EDAD_RANGO       SEXO  RÉGIMEN PACIENTE  \
0                  TN  …       JUVENTUD  MASCULINO        SUBSIDIADO
1                  ME  …       JUVENTUD   FEMENINO        SUBSIDIADO
2                  UC  …       JUVENTUD  MASCULINO      CONTRIBUTIVO
3                  UC  …         ADULTO  MASCULINO        SUBSIDIADO
4                  TN  …       JUVENTUD  MASCULINO        SUBSIDIADO
…                …  …            …          …                 …
82411              ME  …       JUVENTUD   FEMENINO      CONTRIBUTIVO
82412              ME  …  ADULTO MAYOR  MASCULINO        SUBSIDIADO
82413              ME  …  ADULTO MAYOR  MASCULINO      CONTRIBUTIVO
82414              UB  …       JUVENTUD  MASCULINO        SUBSIDIADO
82415              TN  …    ADOLECENCIA   FEMENINO        SUBSIDIADO

      NOMBRE_ENTIDAD  MEDICO   AÑO MES  DIA_SEMANA  HOUR  Turnos
0              EPSS34  11065  2023   1     DOMINGO     1   NOCHE
1              EPSS34   8861  2023   1     DOMINGO     1   NOCHE
2              EPS002   5855  2023   1     DOMINGO     3   NOCHE
3              EPSS34  11072  2023   1     DOMINGO     5   NOCHE
4              EPSS34   1239  2023   1     DOMINGO     6   NOCHE
…                …    …   …  ..          …    …     …
82411          EPS017   7844  2023   9       LUNES     4   NOCHE
82412          EPSC34   6204  2023   9       LUNES     6   NOCHE
82413          EPS041   9951  2023   9       LUNES     6   NOCHE
82414          EPSC34   4030  2023   9       LUNES     6   NOCHE
82415          EPSC34   1239  2023   9       LUNES     7  MAÑANA

[82416 rows x 23 columns]
```

[17]:
```
df_hearth
```

[17]:
```
        TIEMPO_LLEGADA_A_TRIAGE TIEMPO_TRIAGE_A_INGRESO  \
0                       0:00:00                0:16:13
1                       0:00:00                0:08:06
2                       0:00:00                0:33:41
3                       0:00:00                0:17:52
4                       0:00:00                0:13:57
…                         …                      …
1510868                 0:06:39                0:03:20
```

```
1510869                 1:17:38                 0:01:19
1510870                 0:24:11                 0:00:32
1510871                 0:05:44                 0:07:40
1510872                 0:09:18                 0:01:08

        TIEMPO_INGRESO_A_FOLIO TIEMPO_TOTAL       FECHA_LLEGADA  \
0                      0:08:47      0:25:00 2016-08-02 07:19:00
1                      0:01:27      0:09:33 2016-08-02 07:58:00
2                      0:04:12      0:37:53 2016-08-03 09:10:00
3                      0:02:22      0:20:14 2016-08-03 13:05:00
4                      0:21:41      0:35:38 2016-08-01 08:35:00
…                        …          …                 …
1510868                0:00:00      0:00:00 2023-09-14 19:48:00
1510869                0:13:29      1:32:26 2023-09-14 18:40:00
1510870                0:01:16      0:25:59 2023-09-14 19:37:00
1510871                0:00:00      0:00:00 2023-09-14 19:50:00
1510872                0:00:00      0:00:00 2023-09-14 19:55:00

               FECHA_TRIAGE       FECHA_INGRESO      FECHA_CONSULTA  \
0       2016-08-02 07:19:00 2016-08-02 07:36:00 2016-08-02 07:45:00
1       2016-08-02 07:58:00 2016-08-02 08:07:00 2016-08-02 08:09:00
2       2016-08-03 09:10:00 2016-08-03 09:44:00 2016-08-03 09:48:00
3       2016-08-03 13:05:00 2016-08-03 13:23:00 2016-08-03 13:26:00
4       2016-08-01 08:35:00 2016-08-01 08:49:00 2016-08-01 09:11:00
…                       …                   …                 …
1510868 2023-09-14 19:55:00 2023-09-14 19:58:00 2023-09-14 20:02:00
1510869 2023-09-14 19:57:00 2023-09-14 19:59:00 2023-09-14 20:00:00
1510870 2023-09-14 20:02:00 2023-09-14 20:02:00 2023-09-14 20:03:00
1510871 2023-09-14 19:56:00 2023-09-14 20:03:00 2023-09-14 20:11:00
1510872 2023-09-14 20:04:00 2023-09-14 20:05:00 2023-09-14 20:06:00

        CENTRO_ATENCION CLASIFICACION_TRIAGE  … HOUR MES DIA   AÑO  Turnos  \
0                    VA                    3  …    7   8   2  2016  MAÑANA
1                    VA                    3  …    7   8   2  2016  MAÑANA
2                    VA                    3  …    9   8   3  2016  MAÑANA
3                    VC                    3  …   13   8   3  2016   TARDE
4                    ME                    3  …    8   8   1  2016  MAÑANA
…                 …                      …  …  …  ..  ..   …     …
1510868              UC                    3  …   19   9  14  2023   NOCHE
1510869              JT                    3  …   18   9  14  2023   TARDE
1510870              ME                    2  …   19   9  14  2023   NOCHE
1510871              UC                    3  …   19   9  14  2023   NOCHE
1510872              ME                    3  …   19   9  14  2023   NOCHE

        Tiempo_Triage Tiempo_Ingreso Tiempo_Consulta  Tiempo_Minutos_Total  \
0                 0.0           17.0             9.0                  26.0
1                 0.0            9.0             2.0                  11.0
```

```
2                0.0          34.0            4.0                38.0
3                0.0          18.0            3.0                21.0
4                0.0          14.0           22.0                36.0
…                …            …               …                  …
1510868          7.0           3.0            4.0                14.0
1510869         77.0           2.0            1.0                80.0
1510870         25.0           0.0            1.0                26.0
1510871          6.0           7.0            8.0                21.0
1510872          9.0           1.0            1.0                11.0

          DIA_SEMANA
0            Tuesday
1            Tuesday
2          Wednesday
3          Wednesday
4             Monday
…               …
1510868     Thursday
1510869     Thursday
1510870     Thursday
1510871     Thursday
1510872     Thursday

[1510873 rows x 28 columns]
```

[25]:
```
# filtrado por año
FILTRO = 2023
df_filtrado = df_hearth[df_hearth['AÑO'] == FILTRO]
df_filtrado.to_csv('ind_urgencias_final_2023.txt', sep=';', index=False)
```

[26]:
```
df_filtrado
```

[26]:
```
         TIEMPO_LLEGADA_A_TRIAGE TIEMPO_TRIAGE_A_INGRESO  \
1373529                 0:00:00                 0:11:26
1373530                 0:00:00                 0:00:20
1373531                 0:15:58                 0:01:01
1373532                 0:00:00                 0:00:00
1373533                 0:00:00                 0:05:00
…                         …                     …
1510868                 0:06:39                 0:03:20
1510869                 1:17:38                 0:01:19
1510870                 0:24:11                 0:00:32
1510871                 0:05:44                 0:07:40
1510872                 0:09:18                 0:01:08

         TIEMPO_INGRESO_A_FOLIO TIEMPO_TOTAL       FECHA_LLEGADA  \
1373529                0:05:59      0:17:25 2023-01-01 00:04:00
```

```
1373530               0:29:25      0:29:45 2023-01-01 00:16:00
1373531               0:27:06      0:44:05 2023-01-01 00:05:00
1373532               0:00:00      0:00:00 2023-01-01 00:24:00
1373533               0:01:01      0:06:01 2023-01-01 00:24:00
…                       …          …                 …
1510868               0:00:00      0:00:00 2023-09-14 19:48:00
1510869               0:13:29      1:32:26 2023-09-14 18:40:00
1510870               0:01:16      0:25:59 2023-09-14 19:37:00
1510871               0:00:00      0:00:00 2023-09-14 19:50:00
1510872               0:00:00      0:00:00 2023-09-14 19:55:00

              FECHA_TRIAGE       FECHA_INGRESO      FECHA_CONSULTA  \
1373529 2023-01-01 00:04:00 2023-01-01 00:15:00 2023-01-01 00:27:00
1373530 2023-01-01 00:16:00 2023-01-01 00:17:00 2023-01-01 00:17:00
1373531 2023-01-01 00:21:00 2023-01-01 00:22:00 2023-01-01 00:23:00
1373532 2023-01-01 00:24:00 2023-01-01 00:24:00 2023-01-01 00:24:00
1373533 2023-01-01 00:24:00 2023-01-01 00:29:00 2023-01-01 00:34:00
…                       …                   …                 …
1510868 2023-09-14 19:55:00 2023-09-14 19:58:00 2023-09-14 20:02:00
1510869 2023-09-14 19:57:00 2023-09-14 19:59:00 2023-09-14 20:00:00
1510870 2023-09-14 20:02:00 2023-09-14 20:02:00 2023-09-14 20:03:00
1510871 2023-09-14 19:56:00 2023-09-14 20:03:00 2023-09-14 20:11:00
1510872 2023-09-14 20:04:00 2023-09-14 20:05:00 2023-09-14 20:06:00

        CENTRO_ATENCION CLASIFICACION_TRIAGE  … HOUR MES DIA   AÑO  \
1373529              JT                    3  …    0   1   1  2023
1373530              TN                    3  …    0   1   1  2023
1373531              TN                    3  …    0   1   1  2023
1373532              VB                    3  …    0   1   1  2023
1373533              VC                    3  …    0   1   1  2023
…                 …                    …  …  …   ..  ..   …
1510868              UC                    3  …   19   9  14  2023
1510869              JT                    3  …   18   9  14  2023
1510870              ME                    2  …   19   9  14  2023
1510871              UC                    3  …   19   9  14  2023
1510872              ME                    3  …   19   9  14  2023

           Turnos Tiempo_Triage Tiempo_Ingreso Tiempo_Consulta  \
1373529  MADRUGADA           0.0           11.0            12.0
1373530  MADRUGADA           0.0            1.0             0.0
1373531  MADRUGADA          16.0            1.0             1.0
1373532  MADRUGADA           0.0            0.0             0.0
1373533  MADRUGADA           0.0            5.0             5.0
…            …           …             …              …
1510868      NOCHE           7.0            3.0             4.0
1510869      TARDE          77.0            2.0             1.0
1510870      NOCHE          25.0            0.0             1.0
```

```
1510871      NOCHE           6.0            7.0             8.0
1510872      NOCHE           9.0            1.0             1.0

         Tiempo_Minutos_Total  DIA_SEMANA
1373529                  23.0      Sunday
1373530                   1.0      Sunday
1373531                  18.0      Sunday
1373532                   0.0      Sunday
1373533                  10.0      Sunday
…                         …          …
1510868                  14.0    Thursday
1510869                  80.0    Thursday
1510870                  26.0    Thursday
1510871                  21.0    Thursday
1510872                  11.0    Thursday

[137335 rows x 28 columns]
```

[15]:
```
df_filtrado.head(5)
```

[15]:
```
Empty DataFrame
Columns: [TIEMPO_LLEGADA_A_TRIAGE, TIEMPO_TRIAGE_A_INGRESO,
TIEMPO_INGRESO_A_FOLIO, TIEMPO_TOTAL, FECHA_LLEGADA, FECHA_TRIAGE,
FECHA_INGRESO, FECHA_CONSULTA, CENTRO_ATENCION, CLASIFICACION_TRIAGE,
PACIENTE_#_DOCUMENTO, PACIENTE_EDAD, EDAD_RANGO, SEXO, RÉGIMEN PACIENTE,
NOMBRE_ENTIDAD, DIAGNOSTICO, TIME, HOUR, MES, DIA, AÑO, Turnos, Tiempo_Triage,
Tiempo_Ingreso, Tiempo_Consulta, Tiempo_Minutos_Total, DIA_SEMANA]
Index: []

[0 rows x 28 columns]
```

[97]:
```
# Definir una lista de archivos y estructuras

archivos_y_estructuras = [
(f'ind_urgencias_final.txt',{ 'TIEMPO_LLEGADA_A_TRIAGE': str,␣
 ↪'TIEMPO_TRIAGE_A_INGRESO': str,
       'TIEMPO_INGRESO_A_FOLIO': str, 'TIEMPO_TOTAL': str, 'FECHA_LLEGADA': str,
       'FECHA_TRIAGE': str, 'FECHA_INGRESO': str, 'FECHA_CONSULTA': str,␣
 ↪'CENTRO_ATENCION': str,
       'CLASIFICACION_TRIAGE': str, 'PACIENTE_#_DOCUMENTO': str,␣
 ↪'PACIENTE_EDAD': str,
       'EDAD_RANGO': str, 'SEXO': str, 'RÉGIMEN PACIENTE': str,␣
 ↪'NOMBRE_ENTIDAD': str,
       'NOM_TIPO_HISTORIA': str, 'DIAGNOSTICO': str, 'NOMBRE DX': str, 'TIME':␣
 ↪str, 'HOUR': str, 'MES': str,
       'DIA': str, 'Turnos': str, 'Tiempo_Triague': str, 'Tiempo_Ingreso': str
})
```

```
                ]

for archivo, estructura in archivos_y_estructuras:
    df_buenas = []
    df_errores = []

    with open(archivo, 'r', encoding='utf-8') as file:
        # Leer la primera línea (encabezado) y omitirla
        header = next(file)
        for line in file:
            try:
                observacion = ''  # Inicializa la observación como vacía
                data = line.strip().split(';')
                if len(data) != len(estructura):
                    observacion += "La estrutura no cuadra con la original. Hay␣
 ↪caracteres malos "
                    data[-1] = observacion.strip()
                    df_errores.append(data)
                else:
                    observacion = ''  # Inicializa la observación como vacía
                    registro = pd.Series(data, index=estructura.keys())
                    if not pd.notna(registro['FECHA_CONSULTA']) or not␣
 ↪registro['FECHA_CONSULTA'].strip():
                        observacion += "Fecha consulta no puede estar vacía. "
                        data[-1] = observacion.strip()
                        df_errores.append(data)
                    elif not pd.notna(registro['FECHA_INGRESO']) or not␣
 ↪registro['FECHA_INGRESO'].strip():
                        observacion += "Fecha registro no puede estar vacía. "
                        data[-1] = observacion.strip()
                        df_errores.append(data)
                    else:
                        df_buenas.append(data)
            except Exception as e:
                df_errores.append([str(e), ''])  # Agrega una columna vacía␣
 ↪para la observación de errores

    df_errores_df = pd.DataFrame({'Errores': df_errores})
    df_buenas_df = pd.DataFrame({'Buenas': df_buenas})

    # Guardar archivos de errores y buenos
    nombre_base = archivo.split('.')[0]
    df_errores_df.to_csv(f'Validar/Errores/{nombre_base}_errores.txt',␣
 ↪index=False)
    df_buenas_df.to_csv(f'Validar/Buenos/{nombre_base}_buenos.txt', index=False)
```

ejerciciospracticos-exploracion-vd

November 23, 2023

#Herramientas para la visualización de datos masivos

Objetivo: El objetivo de este cuaderno es cargar y realizar la exploración inicial de los datos utilizando el lenguaje de programación Python.

Exploración de Datos

Indice

Cargar los Datos

Visualizar los Datos

Tipos de datos

Visualizar las estadísticas

Identificar datos faltantes

Explorar relaciones entre los datos

Graficar las estadísticas

Exportar los datos

##Cargar los datos

Existen varios formatos para un conjunto de datos, .csv, .json, .xlsx, etc. Los datos pueden ser almacenados en distintos lugares, ya sea localmente o en línea. En estas sección aprenderá a cargar un conjunto de datos en su cuaderno de python. En nuestro caso el conjunto de datos Automobile es de una fuente en línea en formato CSV (valores separados por coma). Usemos este conjunto como ejemplo para practicar la lectura de datos.

fuente de datos: https://archive.ics.uci.edu/ml/machine-learning-databases/autos/imports-85.data

tipo de datos: csv

Vamos a utilizar la librería Pandas de Python para realizar la lectura de archivos. Le ponemos un alias pd para que sea más fácil utilizarla:

[ ]:
```
# Importar libreria requerida
import pandas as pd
import numpy as np
import os
```

Después del comando para importar, ahora tenemos acceso a una gran cantidad de clases y funciones predefinidas. Una forma en que pandas le permite trabajar con datos es con dataframes. Repasemos el proceso para pasar de un archivo de valores separados por comas (.csv) a un dataframe. Esta variable csv_path almacena la ruta de .csv, que se utiliza como argumento para la función read_csv. El resultado se almacena en el objeto df, esta es una forma corta común que se usa para una variable que se refiere a un dataframe de Pandas.

[ ]:
```
# ruta de archivos
files = os.listdir("c:\\archivos\proyecto")
os.chdir(r'C:\archivos\proyecto')

csv_path = 'ind_urgencias_final_2023_filtrado.txt'

# Read data from CSV file
df = pd.read_csv(csv_path,sep=";",header= None)
```

```
C:\Users\Victor.Gomez\AppData\Local\Temp\ipykernel_14576\2968785239.py:8:
DtypeWarning: Columns (10,12,17,18,19,21,24) have mixed types. Specify dtype
option on import or set low_memory=False.
  df = pd.read_csv(csv_path,sep=";",header= None)
```

##Visualizar los datos

Podemos utilizar el método dataframe.head() para examinar las primeras cinco filas del dataframe, se utiliza cuando el conjuntos de datos es muy grande y no queremos cargar todo:

[ ]:
```
# Imprimir las primeras cinco filas de un dataframe
df.head()
```

[ ]:

	0	1	2
0	FECHA_LLEGADA	FECHA_TRIAGE	FECHA_INGRESO
1	2023-01-01 01:20:23.853	2023-01-01 01:28:01.847	2023-01-01 01:29:41.210
2	2023-01-01 01:29:46.050	2023-01-01 01:48:03.070	2023-01-01 01:49:40.973
3	2023-01-01 03:15:35.623	2023-01-01 03:23:01.990	2023-01-01 03:23:39.793
4	2023-01-01 05:54:53.563	2023-01-01 06:00:07.943	2023-01-01 06:02:07.320

	3	4	5
0	FECHA_ATENCION	TIEMPO_DGTURNO_A_TRIAGE	TIEMPO_TRIAGE_A_INGRESO
1	2023-01-01 02:00:07.590	0:07:38	0:01:40
2	2023-01-01 02:02:53.663	0:18:17	0:01:37
3	2023-01-01 03:30:21.233	0:07:26	0:00:38
4	2023-01-01 06:26:17.050	0:05:14	0:02:00

	6	7	8
0	TIEMPO_INGRESO_A_CONSULTA	TIEMPO_TOTAL	Tiempo_Minutos_Total
1	0:30:26	0:39:44	39,73
2	0:13:13	0:33:07	33,12
3	0:06:42	0:14:46	14,77
4	0:24:10	0:31:24	31,40

```
                9   …                15               16      17    18   19  \
0  CENTRO_ATENCION  …  RÉGIMEN PACIENTE  NOMBRE_ENTIDAD   MEDICO   AÑO  MES
1               TN  …        SUBSIDIADO           EPSS34   11065  2023    1
2               ME  …        SUBSIDIADO           EPSS34    8861  2023    1
3               UC  …      CONTRIBUTIVO           EPS002    5855  2023    1
4               UC  …        SUBSIDIADO           EPSS34   11072  2023    1

           20    21      22                      23   24
0  DIA_SEMANA  HOUR  Turnos                    TIME  DIA
1     DOMINGO     1   NOCHE  2023-01-01 01:20:23.853    1
2     DOMINGO     1   NOCHE  2023-01-01 01:29:46.050    1
3     DOMINGO     3   NOCHE  2023-01-01 03:15:35.623    1
4     DOMINGO     5   NOCHE  2023-01-01 05:54:53.563    1

[5 rows x 25 columns]
```

Después de leer el conjunto de datos podemos utilizar el método dataframe.head(n) para revisar las primeras n filas del dataframe; donde n es un entero. Al contrario de dataframe.head(n), dataframe.tail(n) mostrará las n filas del final del dataframe.

AHORA TÚ:

Revise las ultimas 10 filas del dataframe “df”:

[ ]:
```
# Escribe tu código aquí y presiona Shift+Enter para ejecutar
print("The last 10 rows of the dataframe\n")
df.tail(10)
```

```
The last 10 rows of the dataframe
```

[ ]:
```
                           0                       1   \
82407  2023-09-17 18:12:40.660  2023-09-17 18:43:16.430
82408  2023-09-17 18:52:28.640  2023-09-17 19:48:17.070
82409  2023-09-17 19:57:56.033  2023-09-17 20:15:56.793
82410  2023-09-17 20:22:04.970  2023-09-17 20:36:16.703
82411  2023-09-17 21:22:59.137  2023-09-17 21:42:20.273
82412  2023-09-18 04:44:41.970  2023-09-18 04:53:22.553
82413  2023-09-18 06:17:00.573  2023-09-18 06:28:43.040
82414  2023-09-18 06:21:37.273  2023-09-18 07:00:57.420
82415  2023-09-18 06:25:33.483  2023-09-18 06:42:02.883
82416  2023-09-18 07:14:58.180  2023-09-18 07:30:50.643

                           2                       3        4        5   \
82407  2023-09-17 18:48:36.403  2023-09-17 19:40:11.800  0:30:36  0:05:20
82408  2023-09-17 19:49:47.657  2023-09-17 20:32:54.070  0:55:49  0:01:30
82409  2023-09-17 20:28:57.590  2023-09-17 20:46:29.250  0:18:00  0:13:01
```

```
82410  2023-09-17 20:41:12.783  2023-09-17 22:00:50.287  0:14:12  0:04:56
82411  2023-09-17 21:51:04.433  2023-09-17 22:43:13.710  0:19:21  0:08:44
82412  2023-09-18 05:05:51.423  2023-09-18 06:09:35.867  0:08:41  0:12:29
82413  2023-09-18 06:35:38.213  2023-09-18 07:40:45.957  0:11:43  0:06:55
82414  2023-09-18 07:16:45.907  2023-09-18 08:27:27.337  0:39:20  0:15:48
82415  2023-09-18 06:51:35.970  2023-09-18 07:28:28.290  0:16:29  0:09:33
82416  2023-09-18 07:34:08.370  2023-09-18 07:49:18.440  0:15:52  0:03:18

            6        7       8   9   …            15      16     17    18  \
82407  0:51:35  1:27:31   87,52  ME  …  CONTRIBUTIVO  EPS005    120  2023
82408  0:43:07  1:40:26  100,43  ME  …     SUBSIDIADO  EPSS17    120  2023
82409  0:17:32  0:48:33   48,55  UC  …     SUBSIDIADO  EPSC34   3818  2023
82410  1:19:38  1:38:46   98,77  ME  …     SUBSIDIADO  EPSS02  10786  2023
82411  0:52:09  1:20:14   80,23  ME  …      VINCULADO   11000  10786  2023
82412  1:03:44  1:24:54   84,90  ME  …  CONTRIBUTIVO  EPS017   7844  2023
82413  1:05:07  1:23:45   83,75  ME  …     SUBSIDIADO  EPSC34   6204  2023
82414  1:10:42  2:05:50  125,83  ME  …  CONTRIBUTIVO  EPS041   9951  2023
82415  0:36:53  1:02:55   62,92  UB  …     SUBSIDIADO  EPSC34   4030  2023
82416  0:15:10  0:34:20   34,33  TN  …     SUBSIDIADO  EPSC34   1239  2023

      19       20  21      22                      23  24
82407  9  DOMINGO  18   TARDE  2023-09-17 18:12:40.660  17
82408  9  DOMINGO  18   TARDE  2023-09-17 18:52:28.640  17
82409  9  DOMINGO  19   TARDE  2023-09-17 19:57:56.033  17
82410  9  DOMINGO  20   NOCHE  2023-09-17 20:22:04.970  17
82411  9  DOMINGO  21   NOCHE  2023-09-17 21:22:59.137  17
82412  9    LUNES   4   NOCHE  2023-09-18 04:44:41.970  18
82413  9    LUNES   6   NOCHE  2023-09-18 06:17:00.573  18
82414  9    LUNES   6   NOCHE  2023-09-18 06:21:37.273  18
82415  9    LUNES   6   NOCHE  2023-09-18 06:25:33.483  18
82416  9    LUNES   7  MAÑANA  2023-09-18 07:14:58.180  18

[10 rows x 25 columns]
```

###Añadir cabeceras

Observe el conjunto de datos; Pandas automaticamente establece la cabecera en un entero a partir de 0.
Para describir mejor nuestros datos podemos agregarle una cabecera, esta información esta disponible en: https://archive.ics.uci.edu/ml/datasets/Automobile

De este modo debemos agregar las cabeceras manualmente.
Primero creamos una lista headers que incluya todos los nombres de columna en orden. Despues usamos dataframe.columns = headers para reemplazar las cabeceras por la lista que hemos creado.

[ ]:
```
# crear la lista headers
headers =␣
  ↪["FECHA_LLEGADA","FECHA_TRIAGE","FECHA_INGRESO","FECHA_ATENCION","TIEMPO_DGTURNO_A_TRIAGE",
```

```
␣
↪"CENTRO_ATENCION","CLASIFICACION_TRIAGE","PACIENTE_#_DOCUMENTO","EDAD","EDAD_RANGO","SEXO",
↪PACIENTE","NOMBRE_ENTIDAD","MEDICO","AÑO","MES","DIA_SEMANA","HOUR","Turnos","TIME","DIA"]
print("headers\n", headers)
```

```
headers
 ['FECHA_LLEGADA', 'FECHA_TRIAGE', 'FECHA_INGRESO', 'FECHA_ATENCION',
'TIEMPO_DGTURNO_A_TRIAGE', 'TIEMPO_TRIAGE_A_INGRESO',
'TIEMPO_INGRESO_A_CONSULTA', 'TIEMPO_TOTAL', 'Tiempo_Minutos_Total',
'CENTRO_ATENCION', 'CLASIFICACION_TRIAGE', 'PACIENTE_#_DOCUMENTO', 'EDAD',
'EDAD_RANGO', 'SEXO', 'RÉGIMEN PACIENTE', 'NOMBRE_ENTIDAD', 'MEDICO', 'AÑO',
'MES', 'DIA_SEMANA', 'HOUR', 'Turnos', 'TIME', 'DIA']
```

Remplazamos las cabeceras y volvemos a revisar nuestro dataframe:

[ ]:
```
df.columns = headers
df.head()
```

Acceder a una columna y ver sus valores

Se accede a una columna especificando el nombre de la misma. Por ejemplo, puedes acceder a la columna symboling y a la columna body-style. Cada una de estas columnas es una serie de Pandas.

[ ]:
```
x=df[["PACIENTE_#_DOCUMENTO"]]
x
```

[ ]:
```
       PACIENTE_#_DOCUMENTO
0      PACIENTE_#_DOCUMENTO
1                1007228378
2                1000003681
3                1007454009
4                1022997183
…                       …
82412            1010242518
82413              93357619
82414              41372387
82415            1000807249
82416               4497020

[82417 rows x 1 columns]
```

[ ]:
```
y=df[["Tiempo_Minutos_Total"]]
y
```

[ ]:
```
       Tiempo_Minutos_Total
0      Tiempo_Minutos_Total
1                     39,73
2                     33,12
3                     14,77
```

```
4                    31,40
…                    …
82412                84,90
82413                83,75
82414               125,83
82415                62,92
82416                34,33

[82417 rows x 1 columns]
```

##Tipos de datos

Los datos se encuentran en una variedad de tipos. Los tipos principales almacenados en dataframes de Pandas son object, float, int, bool y datetime64. Para aprender mejor acerca de cada atributo es mejor para nosotros saber el tipo de dato de cada columna.

[ ]:
```
#La función dtypes genera una tabla con el tipo de dato de cada columna
df.dtypes
```

[ ]:
```
FECHA_LLEGADA                object
FECHA_TRIAGE                 object
FECHA_INGRESO                object
FECHA_ATENCION               object
TIEMPO_DGTURNO_A_TRIAGE      object
TIEMPO_TRIAGE_A_INGRESO      object
TIEMPO_INGRESO_A_CONSULTA    object
TIEMPO_TOTAL                 object
Tiempo_Minutos_Total         object
CENTRO_ATENCION              object
CLASIFICACION_TRIAGE         object
PACIENTE_#_DOCUMENTO         object
EDAD                         object
EDAD_RANGO                   object
SEXO                         object
RÉGIMEN PACIENTE             object
NOMBRE_ENTIDAD               object
MEDICO                       object
AÑO                          object
MES                          object
DIA_SEMANA                   object
HOUR                         object
Turnos                       object
TIME                         object
DIA                          object
dtype: object
```

Tipo de dato de una columna específica

De esta forma podemos consultar cuál es el tipo de dato de una columna específica:

[ ]:
```
#Separamos la columna en una dataframe llamado df_column
df_column=df[['Turnos']]
df_column.dtypes
```

[ ]:
```
Turnos    object
dtype: object
```

Cambiar el tipo de dato de una columna específica

¿Cómo cambiar el tipo de dato de una columna específica? Cambiemos el tipo de datos de la columna Price que fue identificado como object y es un float.

[ ]:
```
#utilizamos errors='coerce' para ignorar los datos faltantes
# df["price"] = pd.to_numeric(df["price"],errors='coerce')

df['Turnos'] = df['Turnos'].astype('category')
df['DIA_SEMANA'] = df['DIA_SEMANA'].astype('category')
df['CENTRO_ATENCION'] = df['CENTRO_ATENCION'].astype('category')
df.dtypes
```

[ ]:
```
FECHA_LLEGADA                 object
FECHA_TRIAGE                  object
FECHA_INGRESO                 object
FECHA_ATENCION                object
TIEMPO_DGTURNO_A_TRIAGE       object
TIEMPO_TRIAGE_A_INGRESO       object
TIEMPO_INGRESO_A_CONSULTA     object
TIEMPO_TOTAL                  object
Tiempo_Minutos_Total          object
CENTRO_ATENCION             category
CLASIFICACION_TRIAGE          object
PACIENTE_#_DOCUMENTO          object
EDAD                          object
EDAD_RANGO                    object
SEXO                          object
RÉGIMEN PACIENTE              object
NOMBRE_ENTIDAD                object
MEDICO                        object
AÑO                           object
MES                           object
DIA_SEMANA                  category
HOUR                          object
Turnos                      category
TIME                          object
DIA                           object
dtype: object
```

Como se muestra, se observa claramente que el tipo de dato de symboling y curb-weight es int64,

normalized-losses es object pero debería ser de tipo numérico, al igual que bore, etc. Estos tipos de datos pueden modificarse.

AHORA TÚ:

Cambie el tipo de datos de la columna "stroke":

[ ]:
```
# Escribe tu código aquí y presiona Shift+Enter para ejecutar

df['Tiempo_Total'] = df['Tiempo_Minutos_Total'].str.replace(',', '.',␣
↪regex=True)
df['Tiempo_Total'] = pd.to_numeric(df['Tiempo_Total'], errors='coerce')
```

##Visualizar las estadísticas

Este conjunto de datos es pequeño, pero si se quisiera saber la cantidad de atributos y de elementos que se tienen en el conjunto de datos, se puede utilizar la función dataframe.shape. Esta función visualiza primero el número de elementos y luego el número de atributos.

[ ]:
```
df.shape
```

[ ]:
```
(82417, 26)
```

Vamos a utilizar la función dataframe.describe para visualizar las estadísticas del conjunto de datos. Por defecto, la función dataframe.describe muestra las filas y columnas que contienen números.

Esto mostrará:

el recuento de esa variable

la media

la desviación estándar (std)

el valor mínimo

el IQR (rango intercuartil: 25%, 50% y 75%)

el valor máximo

[ ]:
```
df.describe()
```

[ ]:
```
       Tiempo_Total
count  82416.000000
mean     108.346035
std      132.899154
min        8.570000
25%       60.600000
50%       84.200000
75%      117.672500
max     2684.830000
```

Si se quisiera calcular la mediana de una variable en específico se puede de la siguiente manera:

[ ]:
```
import matplotlib.pyplot as plt
import seaborn as sns
sns.distplot(df["Tiempo_Total"])
```

[ ]:
```
df.head()
```

[ ]:
```
             FECHA_LLEGADA             FECHA_TRIAGE            FECHA_INGRESO  \
1  2023-01-01 01:20:23.853  2023-01-01 01:28:01.847  2023-01-01 01:29:41.210
2  2023-01-01 01:29:46.050  2023-01-01 01:48:03.070  2023-01-01 01:49:40.973
3  2023-01-01 03:15:35.623  2023-01-01 03:23:01.990  2023-01-01 03:23:39.793
4  2023-01-01 05:54:53.563  2023-01-01 06:00:07.943  2023-01-01 06:02:07.320
5  2023-01-01 06:37:27.237  2023-01-01 07:52:31.687  2023-01-01 07:52:37.717

            FECHA_ATENCION TIEMPO_DGTURNO_A_TRIAGE TIEMPO_TRIAGE_A_INGRESO  \
1  2023-01-01 02:00:07.590                0:07:38                 0:01:40
2  2023-01-01 02:02:53.663                0:18:17                 0:01:37
3  2023-01-01 03:30:21.233                0:07:26                 0:00:38
4  2023-01-01 06:26:17.050                0:05:14                 0:02:00
5  2023-01-01 09:31:15.597                1:15:04                 0:00:06

  TIEMPO_INGRESO_A_CONSULTA TIEMPO_TOTAL Tiempo_Minutos_Total CENTRO_ATENCION  \
1                   0:30:26      0:39:44               39,73             TN
2                   0:13:13      0:33:07               33,12             ME
3                   0:06:42      0:14:46               14,77             UC
4                   0:24:10      0:31:24               31,40             UC
5                   1:38:38      2:53:48              173,80             TN

   … NOMBRE_ENTIDAD MEDICO   AÑO MES DIA_SEMANA HOUR Turnos  \
1  …         EPSS34  11065  2023   1    DOMINGO    1  NOCHE
2  …         EPSS34   8861  2023   1    DOMINGO    1  NOCHE
3  …         EPS002   5855  2023   1    DOMINGO    3  NOCHE
4  …         EPSS34  11072  2023   1    DOMINGO    5  NOCHE
5  …         EPSS34   1239  2023   1    DOMINGO    6  NOCHE

                     TIME DIA Tiempo_Total
1  2023-01-01 01:20:23.853   1        39.73
2  2023-01-01 01:29:46.050   1        33.12
3  2023-01-01 03:15:35.623   1        14.77
4  2023-01-01 05:54:53.563   1        31.40
5  2023-01-01 06:37:27.237   1       173.80

[5 rows x 26 columns]
```

[ ]:
```
df = df.drop(0)
```

[ ]:
```
#Muestra la mediana para los atributos "length" y "compression-ratio"
median= df[['Tiempo_Total']].median()
```

```
median
```

```
Tiempo_Total    84.2
dtype: float64
```

Por defecto la función solo muestra atributos que son numéricos. Es posible hacer que la función describe funcione también para las columnas de tipo object. Para permitir un resumen de todas las columnas, podríamos añadir un argumento include="all" entre los paréntesis de la función describe.

```
#unique, top y frequency ("único, superior y frecuencia").
#df.describe(include="object")
df.describe(include="all")
```

```
                  FECHA_LLEGADA             FECHA_TRIAGE  \
count                     82416                    82416
unique                    80382                    80388
top     2023-08-06 14:31:20.657  2023-08-04 11:13:29.903
freq                          4                        4
mean                        NaN                      NaN
std                         NaN                      NaN
min                         NaN                      NaN
25%                         NaN                      NaN
50%                         NaN                      NaN
75%                         NaN                      NaN
max                         NaN                      NaN

                  FECHA_INGRESO           FECHA_ATENCION  \
count                     82416                    82416
unique                    80387                    82414
top     2023-05-05 16:30:25.897  2023-03-29 21:49:44.440
freq                          4                        2
mean                        NaN                      NaN
std                         NaN                      NaN
min                         NaN                      NaN
25%                         NaN                      NaN
50%                         NaN                      NaN
75%                         NaN                      NaN
max                         NaN                      NaN

       TIEMPO_DGTURNO_A_TRIAGE TIEMPO_TRIAGE_A_INGRESO  \
count                    82416                   82416
unique                    5981                    3433
top                     0:10:06                 0:01:08
freq                         72                     218
mean                        NaN                     NaN
std                         NaN                     NaN
min                         NaN                     NaN
```

```
25%                          NaN                    NaN
50%                          NaN                    NaN
75%                          NaN                    NaN
max                          NaN                    NaN

       TIEMPO_INGRESO_A_CONSULTA TIEMPO_TOTAL Tiempo_Minutos_Total  \
count                      82416        82416                82416
unique                     12074        13866                14141
top                      0:33:27      1:13:04                73,07
freq                          37           28                   28
mean                         NaN          NaN                  NaN
std                          NaN          NaN                  NaN
min                          NaN          NaN                  NaN
25%                          NaN          NaN                  NaN
50%                          NaN          NaN                  NaN
75%                          NaN          NaN                  NaN
max                          NaN          NaN                  NaN

       CENTRO_ATENCION  …  NOMBRE_ENTIDAD   MEDICO      AÑO      MES  \
count            82416  …           82416  82416.0  82416.0  82416.0
unique               9  …              78    378.0      2.0     14.0
top                 ME  …          EPSC34   3826.0   2023.0      7.0
freq             41221  …           45973   1464.0  49649.0  10108.0
mean               NaN  …             NaN      NaN      NaN      NaN
std                NaN  …             NaN      NaN      NaN      NaN
min                NaN  …             NaN      NaN      NaN      NaN
25%                NaN  …             NaN      NaN      NaN      NaN
50%                NaN  …             NaN      NaN      NaN      NaN
75%                NaN  …             NaN      NaN      NaN      NaN
max                NaN  …             NaN      NaN      NaN      NaN

       DIA_SEMANA     HOUR  Turnos                       TIME      DIA  \
count       82416  82416.0   82416                      82416  82416.0
unique          7     48.0       3                      80382     62.0
top         LUNES     10.0  MAÑANA  2023-08-06 14:31:20.657      8.0
freq        13390   3911.0   40363                          4   2040.0
mean          NaN      NaN     NaN                        NaN      NaN
std           NaN      NaN     NaN                        NaN      NaN
min           NaN      NaN     NaN                        NaN      NaN
25%           NaN      NaN     NaN                        NaN      NaN
50%           NaN      NaN     NaN                        NaN      NaN
75%           NaN      NaN     NaN                        NaN      NaN
max           NaN      NaN     NaN                        NaN      NaN

        Tiempo_Total
count   82416.000000
unique           NaN
```

```
top              NaN
freq             NaN
mean      108.346035
std       132.899154
min         8.570000
25%        60.600000
50%        84.200000
75%       117.672500
max      2684.830000

[11 rows x 26 columns]
```

Contar Valores

Una forma de resumir los datos categóricos es usando la función value_counts. Por ejemplo, en nuestro conjunto de datos, tenemos el lugar del motor (engine-location) como una variable categórica de frontal y trasero.

[ ]:
```
drive_wheels_counts = df['Turnos'].value_counts().to_frame()
drive_wheels_counts
```

[ ]:
```
         count
Turnos
MAÑANA   40363
TARDE    25052
NOCHE    17001
Turnos       0
```

Examinar los recuentos de valores de la ubicación del motor no sería una buena variable predictiva del precio. Esto se debe a que solo tenemos tres autos con motor trasero y 202 con motor delantero, este resultado es sesgado. Por lo tanto, no podemos sacar ninguna conclusión sobre la ubicación del motor.

AHORA TÚ:

Puede seleccionar las columna de un dataframe indicando el nombre de cada una, por ejemplo, puede seleccionar tres columnas de la siguiente manera:

dataframe[[‘column 1’,column 2’, ‘column 3’]]

Donde “column” es el nombre de la columna se puede aplicar el método “.describe()” para obtener las estadísticas de aquellas columnas de la siguiente manera:

dataframe[[’ column 1 ‘,column 2’, ‘column 3’]].describe()

Aplicar el método “.describe()” a las columnas ‘length’ y ‘compression-ratio’.

[ ]:
```
# Escribe tu código aquí y presiona Shift+Enter para ejecutar
df[['DIA_SEMANA', 'Turnos','Tiempo_Total']].describe()
```

[ ]:
```
       Tiempo_Total
count  82416.000000
mean     108.346035
std      132.899154
min        8.570000
25%       60.600000
50%       84.200000
75%      117.672500
max     2684.830000
```

##Identificar datos faltantes

Debemos visualizar nuestros datos e identificar el valor(es) que se está utilizando para los datos faltantes. Los valores faltantes pueden ser espacios vacíos, NA, n/a, –, 0, o cualquier otro valor que no es considerado correcto en esa columna.

[ ]:
```
#Identifique cual es el valor que se está utilizando para los datos faltantes␣
↪en el set de datos:
df.head(20)
```

[ ]:
```
              FECHA_LLEGADA             FECHA_TRIAGE            FECHA_INGRESO  \
1   2023-01-01 01:20:23.853  2023-01-01 01:28:01.847  2023-01-01 01:29:41.210
2   2023-01-01 01:29:46.050  2023-01-01 01:48:03.070  2023-01-01 01:49:40.973
3   2023-01-01 03:15:35.623  2023-01-01 03:23:01.990  2023-01-01 03:23:39.793
4   2023-01-01 05:54:53.563  2023-01-01 06:00:07.943  2023-01-01 06:02:07.320
5   2023-01-01 06:37:27.237  2023-01-01 07:52:31.687  2023-01-01 07:52:37.717
6   2023-01-01 07:09:46.950  2023-01-01 07:18:01.200  2023-01-01 07:20:03.720
7   2023-01-01 07:20:31.113  2023-01-01 07:27:36.230  2023-01-01 07:34:17.667
8   2023-01-01 07:53:52.963  2023-01-01 08:01:10.640  2023-01-01 08:03:13.710
9   2023-01-01 08:05:21.230  2023-01-01 08:53:07.870  2023-01-01 08:56:10.277
10  2023-01-01 09:24:15.530  2023-01-01 09:52:05.463  2023-01-01 09:59:19.673
11  2023-01-01 10:57:35.917  2023-01-01 11:26:37.567  2023-01-01 11:30:19.197
12  2023-01-01 14:34:37.470  2023-01-01 14:49:51.770  2023-01-01 15:02:04.030
13  2023-01-01 14:54:22.407  2023-01-01 15:13:27.167  2023-01-01 15:20:55.660
14  2023-01-01 14:56:15.033  2023-01-01 15:03:35.823  2023-01-01 15:04:17.987
15  2023-01-01 16:09:09.527  2023-01-01 16:14:42.000  2023-01-01 16:16:22.767
16  2023-01-01 18:07:03.087  2023-01-01 18:23:37.837  2023-01-01 18:24:11.420
17  2023-01-01 18:13:00.330  2023-01-01 18:31:06.813  2023-01-01 18:34:41.503
18  2023-01-02 03:34:19.883  2023-01-02 03:56:47.470  2023-01-02 03:57:22.443
19  2023-01-02 04:30:07.617  2023-01-02 05:36:52.657  2023-01-02 05:38:34.253
20  2023-01-02 06:27:33.233  2023-01-02 07:12:56.400  2023-01-02 07:13:07.473

              FECHA_ATENCION TIEMPO_DGTURNO_A_TRIAGE TIEMPO_TRIAGE_A_INGRESO  \
1   2023-01-01 02:00:07.590                0:07:38                0:01:40
2   2023-01-01 02:02:53.663                0:18:17                0:01:37
3   2023-01-01 03:30:21.233                0:07:26                0:00:38
4   2023-01-01 06:26:17.050                0:05:14                0:02:00
5   2023-01-01 09:31:15.597                1:15:04                0:00:06
```

```
6   2023-01-01 07:48:42.197                0:08:15                0:02:02
7   2023-01-01 07:59:17.140                0:07:05                0:06:41
8   2023-01-01 08:43:09.917                0:07:18                0:02:03
9   2023-01-01 09:26:52.977                0:47:46                0:03:03
10  2023-01-01 10:38:12.297                0:27:50                0:07:14
11  2023-01-01 12:48:10.377                0:29:02                0:03:42
12  2023-01-01 16:33:36.400                0:15:14                0:12:13
13  2023-01-01 16:47:10.060                0:19:05                0:07:28
14  2023-01-01 16:12:09.400                0:07:20                0:00:42
15  2023-01-01 16:39:18.937                0:05:33                0:01:40
16  2023-01-01 19:33:28.253                0:16:34                0:00:34
17  2023-01-01 19:54:53.770                0:18:06                0:03:35
18  2023-01-02 04:49:21.190                0:22:28                0:00:35
19  2023-01-02 05:54:41.813                1:06:45                0:01:42
20  2023-01-02 07:21:55.423                0:45:23                0:00:11

   TIEMPO_INGRESO_A_CONSULTA TIEMPO_TOTAL Tiempo_Minutos_Total  \
1                    0:30:26      0:39:44                39,73
2                    0:13:13      0:33:07                33,12
3                    0:06:42      0:14:46                14,77
4                    0:24:10      0:31:24                31,40
5                    1:38:38      2:53:48               173,80
6                    0:28:39      0:38:56                38,93
7                    0:25:00      0:38:46                38,77
8                    0:39:56      0:49:17                49,28
9                    0:30:42      1:21:31                81,52
10                   0:38:53      1:13:57                73,95
11                   1:17:51      1:50:35               110,58
12                   1:31:32      1:58:59               118,98
13                   1:26:15      1:52:48               112,80
14                   1:07:52      1:15:54                75,90
15                   0:22:56      0:30:09                30,15
16                   1:09:17      1:26:25                86,42
17                   1:20:12      1:41:53               101,88
18                   0:51:59      1:15:02                75,03
19                   0:16:07      1:24:34                84,57
20                   0:08:48      0:54:22                54,37

   CENTRO_ATENCION  … NOMBRE_ENTIDAD MEDICO   AÑO MES DIA_SEMANA HOUR  \
1               TN  …         EPSS34  11065  2023   1    DOMINGO    1
2               ME  …         EPSS34   8861  2023   1    DOMINGO    1
3               UC  …         EPS002   5855  2023   1    DOMINGO    3
4               UC  …         EPSS34  11072  2023   1    DOMINGO    5
5               TN  …         EPSS34   1239  2023   1    DOMINGO    6
6               VB  …         EPS005  10751  2023   1    DOMINGO    7
7               UC  …         EPSS34  11072  2023   1    DOMINGO    7
8               VB  …         EPS041  10795  2023   1    DOMINGO    7
```

```
9              UB  …         ESS207   7795  2023   1    DOMINGO    8
10             ME  …         EPSS34    120  2023   1    DOMINGO    9
11             ME  …         EPSS17   7551  2023   1    DOMINGO   10
12             ME  …         ESS024   6204  2023   1    DOMINGO   14
13             VB  …         EPSS34   3826  2023   1    DOMINGO   14
14             VB  …         EPSS34  10795  2023   1    DOMINGO   14
15             TN  …         EPSC34   1239  2023   1    DOMINGO   16
16             TN  …         EPSS34  10030  2023   1    DOMINGO   18
17             TN  …         EPSS05  10030  2023   1    DOMINGO   18
18             ME  …         EPS002  10938  2023   1      LUNES    3
19             JT  …         EPSS34   8373  2023   1      LUNES    4
20             ME  …         EPSC34   4610  2023   1      LUNES    6

    Turnos                       TIME DIA Tiempo_Total
1    NOCHE  2023-01-01 01:20:23.853   1        39.73
2    NOCHE  2023-01-01 01:29:46.050   1        33.12
3    NOCHE  2023-01-01 03:15:35.623   1        14.77
4    NOCHE  2023-01-01 05:54:53.563   1        31.40
5    NOCHE  2023-01-01 06:37:27.237   1       173.80
6   MAÑANA  2023-01-01 07:09:46.950   1        38.93
7   MAÑANA  2023-01-01 07:20:31.113   1        38.77
8   MAÑANA  2023-01-01 07:53:52.963   1        49.28
9   MAÑANA  2023-01-01 08:05:21.230   1        81.52
10  MAÑANA  2023-01-01 09:24:15.530   1        73.95
11  MAÑANA  2023-01-01 10:57:35.917   1       110.58
12   TARDE  2023-01-01 14:34:37.470   1       118.98
13   TARDE  2023-01-01 14:54:22.407   1       112.80
14   TARDE  2023-01-01 14:56:15.033   1        75.90
15   TARDE  2023-01-01 16:09:09.527   1        30.15
16   TARDE  2023-01-01 18:07:03.087   1        86.42
17   TARDE  2023-01-01 18:13:00.330   1       101.88
18   NOCHE  2023-01-02 03:34:19.883   2        75.03
19   NOCHE  2023-01-02 04:30:07.617   2        84.57
20   NOCHE  2023-01-02 06:27:33.233   2        54.37

[20 rows x 26 columns]
```

Con la función isnull podemos saber cuantos datos faltantes identifica Python en nuestro set de datos.

[ ]:
```
print(df.isnull().sum())
```

```
FECHA_LLEGADA               0
FECHA_TRIAGE                0
FECHA_INGRESO               0
FECHA_ATENCION              0
TIEMPO_DGTURNO_A_TRIAGE     0
TIEMPO_TRIAGE_A_INGRESO     0
```

```
TIEMPO_INGRESO_A_CONSULTA    0
TIEMPO_TOTAL                 0
Tiempo_Minutos_Total         0
CENTRO_ATENCION              0
CLASIFICACION_TRIAGE         0
PACIENTE_#_DOCUMENTO         0
EDAD                         0
EDAD_RANGO                   0
SEXO                         0
RÉGIMEN PACIENTE             0
NOMBRE_ENTIDAD               0
MEDICO                       0
AÑO                          0
MES                          0
DIA_SEMANA                   0
HOUR                         0
Turnos                       0
TIME                         0
DIA                          0
Tiempo_Total                 0
dtype: int64
```

Todavía Python no está identificando los datos faltantes en el conjunto de datos, sino que los está tratando como un valor correcto más. Para marcar los datos faltantes se realiza los siguiente:

[ ]:
```
#Realice una lista de los valores que son identificados como datos faltantes
#No olvide al final volver a cargar las cabeceras
missing_values = ["?", "1"]
csv_path = 'ind_urgencias_final_2023_filtrado.txt'
df = pd.read_csv(csv_path,sep=";",header= None, na_values = missing_values)
df.head(20)
```

```
C:\Users\Victor.Gomez\AppData\Local\Temp\ipykernel_14576\2949932682.py:5:
DtypeWarning: Columns (10,12,17,18,19,21,24) have mixed types. Specify dtype
option on import or set low_memory=False.
  df = pd.read_csv(csv_path,sep=";",header= None, na_values = missing_values)
```

[ ]:

	0	1	2	\
0	FECHA_LLEGADA	FECHA_TRIAGE	FECHA_INGRESO	
1	2023-01-01 01:20:23.853	2023-01-01 01:28:01.847	2023-01-01 01:29:41.210	
2	2023-01-01 01:29:46.050	2023-01-01 01:48:03.070	2023-01-01 01:49:40.973	
3	2023-01-01 03:15:35.623	2023-01-01 03:23:01.990	2023-01-01 03:23:39.793	
4	2023-01-01 05:54:53.563	2023-01-01 06:00:07.943	2023-01-01 06:02:07.320	
5	2023-01-01 06:37:27.237	2023-01-01 07:52:31.687	2023-01-01 07:52:37.717	
6	2023-01-01 07:09:46.950	2023-01-01 07:18:01.200	2023-01-01 07:20:03.720	
7	2023-01-01 07:20:31.113	2023-01-01 07:27:36.230	2023-01-01 07:34:17.667	
8	2023-01-01 07:53:52.963	2023-01-01 08:01:10.640	2023-01-01 08:03:13.710	
9	2023-01-01 08:05:21.230	2023-01-01 08:53:07.870	2023-01-01 08:56:10.277	

10	2023-01-01 09:24:15.530	2023-01-01 09:52:05.463	2023-01-01 09:59:19.673
11	2023-01-01 10:57:35.917	2023-01-01 11:26:37.567	2023-01-01 11:30:19.197
12	2023-01-01 14:34:37.470	2023-01-01 14:49:51.770	2023-01-01 15:02:04.030
13	2023-01-01 14:54:22.407	2023-01-01 15:13:27.167	2023-01-01 15:20:55.660
14	2023-01-01 14:56:15.033	2023-01-01 15:03:35.823	2023-01-01 15:04:17.987
15	2023-01-01 16:09:09.527	2023-01-01 16:14:42.000	2023-01-01 16:16:22.767
16	2023-01-01 18:07:03.087	2023-01-01 18:23:37.837	2023-01-01 18:24:11.420
17	2023-01-01 18:13:00.330	2023-01-01 18:31:06.813	2023-01-01 18:34:41.503
18	2023-01-02 03:34:19.883	2023-01-02 03:56:47.470	2023-01-02 03:57:22.443
19	2023-01-02 04:30:07.617	2023-01-02 05:36:52.657	2023-01-02 05:38:34.253

	3	4	5
0	FECHA_ATENCION	TIEMPO_DGTURNO_A_TRIAGE	TIEMPO_TRIAGE_A_INGRESO
1	2023-01-01 02:00:07.590	0:07:38	0:01:40
2	2023-01-01 02:02:53.663	0:18:17	0:01:37
3	2023-01-01 03:30:21.233	0:07:26	0:00:38
4	2023-01-01 06:26:17.050	0:05:14	0:02:00
5	2023-01-01 09:31:15.597	1:15:04	0:00:06
6	2023-01-01 07:48:42.197	0:08:15	0:02:02
7	2023-01-01 07:59:17.140	0:07:05	0:06:41
8	2023-01-01 08:43:09.917	0:07:18	0:02:03
9	2023-01-01 09:26:52.977	0:47:46	0:03:03
10	2023-01-01 10:38:12.297	0:27:50	0:07:14
11	2023-01-01 12:48:10.377	0:29:02	0:03:42
12	2023-01-01 16:33:36.400	0:15:14	0:12:13
13	2023-01-01 16:47:10.060	0:19:05	0:07:28
14	2023-01-01 16:12:09.400	0:07:20	0:00:42
15	2023-01-01 16:39:18.937	0:05:33	0:01:40
16	2023-01-01 19:33:28.253	0:16:34	0:00:34
17	2023-01-01 19:54:53.770	0:18:06	0:03:35
18	2023-01-02 04:49:21.190	0:22:28	0:00:35
19	2023-01-02 05:54:41.813	1:06:45	0:01:42

	6	7	8
0	TIEMPO_INGRESO_A_CONSULTA	TIEMPO_TOTAL	Tiempo_Minutos_Total
1	0:30:26	0:39:44	39,73
2	0:13:13	0:33:07	33,12
3	0:06:42	0:14:46	14,77
4	0:24:10	0:31:24	31,40
5	1:38:38	2:53:48	173,80
6	0:28:39	0:38:56	38,93
7	0:25:00	0:38:46	38,77
8	0:39:56	0:49:17	49,28
9	0:30:42	1:21:31	81,52
10	0:38:53	1:13:57	73,95
11	1:17:51	1:50:35	110,58
12	1:31:32	1:58:59	118,98

13	1:26:15	1:52:48	112,80
14	1:07:52	1:15:54	75,90
15	0:22:56	0:30:09	30,15
16	1:09:17	1:26:25	86,42
17	1:20:12	1:41:53	101,88
18	0:51:59	1:15:02	75,03
19	0:16:07	1:24:34	84,57

	9	…	15	16	17	18	\
0	CENTRO_ATENCION	…	RÉGIMEN PACIENTE	NOMBRE_ENTIDAD	MEDICO	AÑO	
1	TN	…	SUBSIDIADO	EPSS34	11065	2023	
2	ME	…	SUBSIDIADO	EPSS34	8861	2023	
3	UC	…	CONTRIBUTIVO	EPS002	5855	2023	
4	UC	…	SUBSIDIADO	EPSS34	11072	2023	
5	TN	…	SUBSIDIADO	EPSS34	1239	2023	
6	VB	…	CONTRIBUTIVO	EPS005	10751	2023	
7	UC	…	SUBSIDIADO	EPSS34	11072	2023	
8	VB	…	CONTRIBUTIVO	EPS041	10795	2023	
9	UB	…	SUBSIDIADO	ESS207	7795	2023	
10	ME	…	SUBSIDIADO	EPSS34	120	2023	
11	ME	…	CONTRIBUTIVO	EPSS17	7551	2023	
12	ME	…	SUBSIDIADO	ESS024	6204	2023	
13	VB	…	SUBSIDIADO	EPSS34	3826	2023	
14	VB	…	SUBSIDIADO	EPSS34	10795	2023	
15	TN	…	CONTRIBUTIVO	EPSC34	1239	2023	
16	TN	…	SUBSIDIADO	EPSS34	10030	2023	
17	TN	…	SUBSIDIADO	EPSS05	10030	2023	
18	ME	…	CONTRIBUTIVO	EPS002	10938	2023	
19	JT	…	SUBSIDIADO	EPSS34	8373	2023	

	19	20	21	22	23	24
0	MES	DIA_SEMANA	HOUR	Turnos	TIME	DIA
1	1	DOMINGO	1	NOCHE	2023-01-01 01:20:23.853	1
2	1	DOMINGO	1	NOCHE	2023-01-01 01:29:46.050	1
3	1	DOMINGO	3	NOCHE	2023-01-01 03:15:35.623	1
4	1	DOMINGO	5	NOCHE	2023-01-01 05:54:53.563	1
5	1	DOMINGO	6	NOCHE	2023-01-01 06:37:27.237	1
6	1	DOMINGO	7	MAÑANA	2023-01-01 07:09:46.950	1
7	1	DOMINGO	7	MAÑANA	2023-01-01 07:20:31.113	1
8	1	DOMINGO	7	MAÑANA	2023-01-01 07:53:52.963	1
9	1	DOMINGO	8	MAÑANA	2023-01-01 08:05:21.230	1
10	1	DOMINGO	9	MAÑANA	2023-01-01 09:24:15.530	1
11	1	DOMINGO	10	MAÑANA	2023-01-01 10:57:35.917	1
12	1	DOMINGO	14	TARDE	2023-01-01 14:34:37.470	1
13	1	DOMINGO	14	TARDE	2023-01-01 14:54:22.407	1
14	1	DOMINGO	14	TARDE	2023-01-01 14:56:15.033	1
15	1	DOMINGO	16	TARDE	2023-01-01 16:09:09.527	1

```
16    1     DOMINGO    18   TARDE  2023-01-01 18:07:03.087    1
17    1     DOMINGO    18   TARDE  2023-01-01 18:13:00.330    1
18    1       LUNES     3   NOCHE  2023-01-02 03:34:19.883    2
19    1       LUNES     4   NOCHE  2023-01-02 04:30:07.617    2

[20 rows x 25 columns]
```

[ ]:
```
# crear la lista headers

df = df.drop(0)
headers = ␣
  ↪["FECHA_LLEGADA","FECHA_TRIAGE","FECHA_INGRESO","FECHA_ATENCION","TIEMPO_DGTURNO_A_TRIAGE",
          ␣
  ↪"CENTRO_ATENCION","CLASIFICACION_TRIAGE","PACIENTE_#_DOCUMENTO","EDAD","EDAD_RANGO","SEXO",
  ↪PACIENTE","NOMBRE_ENTIDAD","MEDICO","AÑO","MES","DIA_SEMANA","HOUR","Turnos","TIME","DIA"]
df.columns = headers
df.head()
```

[ ]:
```
              FECHA_LLEGADA             FECHA_TRIAGE            FECHA_INGRESO  \
1  2023-01-01 01:20:23.853  2023-01-01 01:28:01.847  2023-01-01 01:29:41.210
2  2023-01-01 01:29:46.050  2023-01-01 01:48:03.070  2023-01-01 01:49:40.973
3  2023-01-01 03:15:35.623  2023-01-01 03:23:01.990  2023-01-01 03:23:39.793
4  2023-01-01 05:54:53.563  2023-01-01 06:00:07.943  2023-01-01 06:02:07.320
5  2023-01-01 06:37:27.237  2023-01-01 07:52:31.687  2023-01-01 07:52:37.717

             FECHA_ATENCION TIEMPO_DGTURNO_A_TRIAGE TIEMPO_TRIAGE_A_INGRESO  \
1  2023-01-01 02:00:07.590                 0:07:38                 0:01:40
2  2023-01-01 02:02:53.663                 0:18:17                 0:01:37
3  2023-01-01 03:30:21.233                 0:07:26                 0:00:38
4  2023-01-01 06:26:17.050                 0:05:14                 0:02:00
5  2023-01-01 09:31:15.597                 1:15:04                 0:00:06

  TIEMPO_INGRESO_A_CONSULTA TIEMPO_TOTAL Tiempo_Minutos_Total CENTRO_ATENCION  \
1                   0:30:26      0:39:44                39,73              TN
2                   0:13:13      0:33:07                33,12              ME
3                   0:06:42      0:14:46                14,77              UC
4                   0:24:10      0:31:24                31,40              UC
5                   1:38:38      2:53:48               173,80              TN

   … RÉGIMEN PACIENTE NOMBRE_ENTIDAD MEDICO   AÑO MES DIA_SEMANA HOUR  \
1  …       SUBSIDIADO         EPSS34  11065  2023   1    DOMINGO    1
2  …       SUBSIDIADO         EPSS34   8861  2023   1    DOMINGO    1
3  …     CONTRIBUTIVO         EPS002   5855  2023   1    DOMINGO    3
4  …       SUBSIDIADO         EPSS34  11072  2023   1    DOMINGO    5
5  …       SUBSIDIADO         EPSS34   1239  2023   1    DOMINGO    6

  Turnos                     TIME DIA
```

```
1  NOCHE  2023-01-01 01:20:23.853   1
2  NOCHE  2023-01-01 01:29:46.050   1
3  NOCHE  2023-01-01 03:15:35.623   1
4  NOCHE  2023-01-01 05:54:53.563   1
5  NOCHE  2023-01-01 06:37:27.237   1

[5 rows x 25 columns]
```

Vuelva a ejectuar la sentencia print(df.isnull().sum()) para visualizar los datos faltantes

[ ]:
```
print(df.isnull().sum())
```

```
FECHA_LLEGADA                0
FECHA_TRIAGE                 0
FECHA_INGRESO                0
FECHA_ATENCION               0
TIEMPO_DGTURNO_A_TRIAGE      0
TIEMPO_TRIAGE_A_INGRESO      0
TIEMPO_INGRESO_A_CONSULTA    0
TIEMPO_TOTAL                 0
Tiempo_Minutos_Total         0
CENTRO_ATENCION              0
CLASIFICACION_TRIAGE         0
PACIENTE_#_DOCUMENTO         0
EDAD                         0
EDAD_RANGO                   0
SEXO                         0
RÉGIMEN PACIENTE             0
NOMBRE_ENTIDAD               0
MEDICO                       0
AÑO                          0
MES                          0
DIA_SEMANA                   0
HOUR                         0
Turnos                       0
TIME                         0
DIA                          0
dtype: int64
```

[ ]:
```
df.dtypes
```

[ ]:
```
FECHA_LLEGADA                object
FECHA_TRIAGE                 object
FECHA_INGRESO                object
FECHA_ATENCION               object
TIEMPO_DGTURNO_A_TRIAGE      object
TIEMPO_TRIAGE_A_INGRESO      object
```

```
TIEMPO_INGRESO_A_CONSULTA      object
TIEMPO_TOTAL                   object
Tiempo_Minutos_Total           object
CENTRO_ATENCION              category
CLASIFICACION_TRIAGE           object
PACIENTE_#_DOCUMENTO           object
EDAD                           object
EDAD_RANGO                     object
SEXO                           object
RÉGIMEN PACIENTE               object
NOMBRE_ENTIDAD                 object
MEDICO                         object
AÑO                            object
MES                             int64
DIA_SEMANA                   category
HOUR                            int64
Turnos                       category
TIME                           object
DIA                             int64
Tiempo_Total                 float64
dtype: object
```

##Explorar relaciones entre los datos

Variables numéricas continuas:

Las variables numéricas continuas son variables que pueden contener cualquier valor dentro de cierto rango. Las variables numéricas continuas pueden tener el tipo int64 o float64. Una excelente manera de visualizar estas variables es mediante el uso de diagramas de dispersión con líneas ajustadas.

Para comenzar a comprender la relación (lineal) entre una variable individual y el precio. Podemos hacer esto usando regplot, que traza el diagrama de dispersión más la línea de regresión ajustada para los datos.

Vamos a importar las librerías Matplotlib y Seaborn para la visualización de datos. Les ponemos un alias plt y sns para que sea más fácil su uso:

[ ]:
```
import matplotlib.pyplot as plt
import seaborn as sns
```

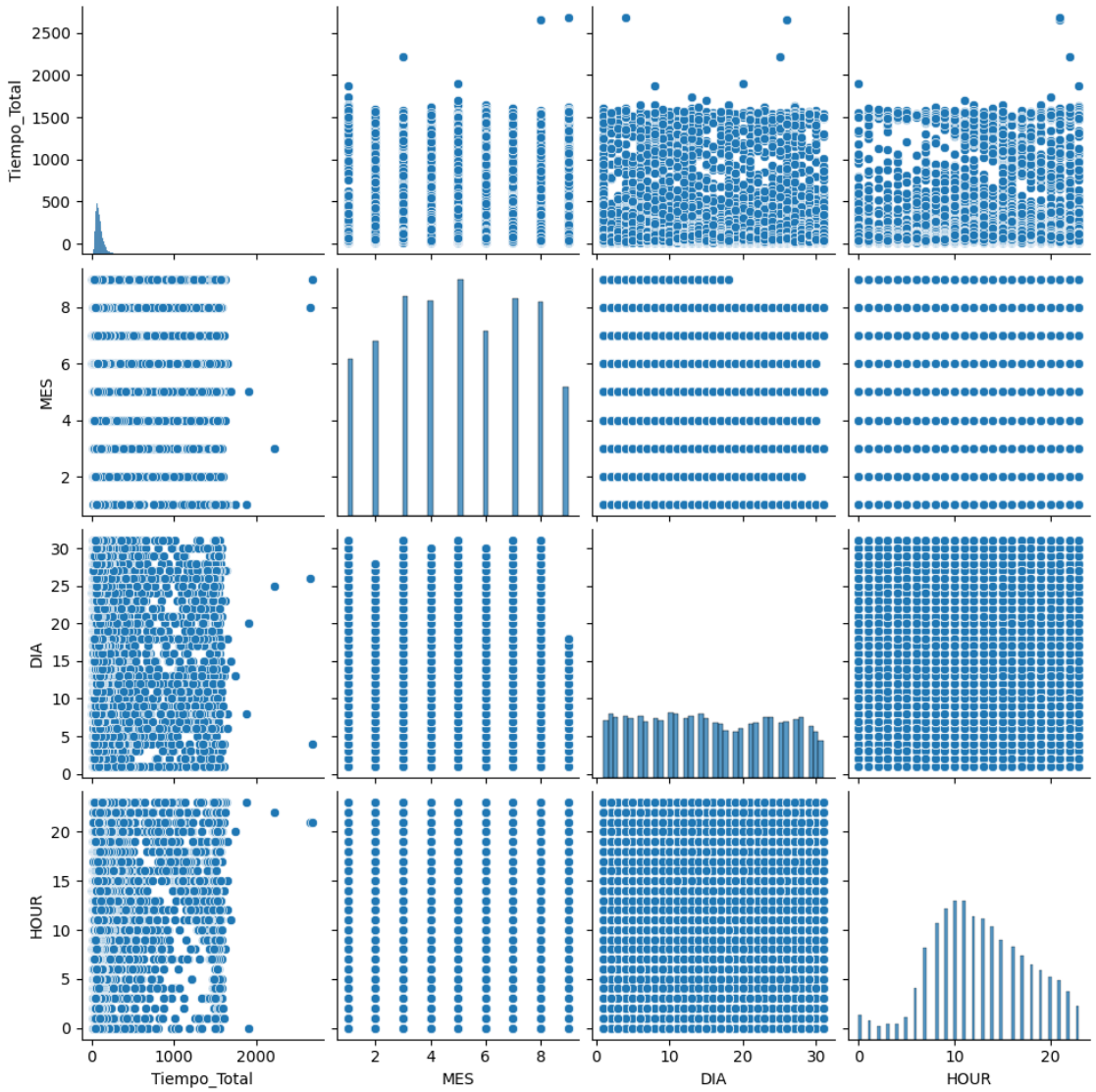
El gráfico seaborn.pairplot permite visualizar todas las variables numéricas y la combinación entre ellas. Pudiendo identificar facilmente si existen relaciones de dependencia entre algunas variables.

[ ]:
```
#sns.pairplot(df)
df['Tiempo_Total'] = df['Tiempo_Minutos_Total'].str.replace(',', '.',␣
 ↪regex=True)
df['Tiempo_Total'] = pd.to_numeric(df['Tiempo_Total'], errors='coerce')
df['Turnos'] = df['Turnos'].astype('category')
df['DIA_SEMANA'] = df['DIA_SEMANA'].astype('category')
```

```
df['CENTRO_ATENCION'] = df['CENTRO_ATENCION'].astype('category')
df['MES'] = pd.to_numeric(df['MES'], errors='coerce')
df['HOUR'] = pd.to_numeric(df['HOUR'], errors='coerce')
df['DIA'] = pd.to_numeric(df['DIA'], errors='coerce')

sub_df = df[['Tiempo_Total','MES', 'DIA','HOUR']]

sns.pairplot(sub_df)
```

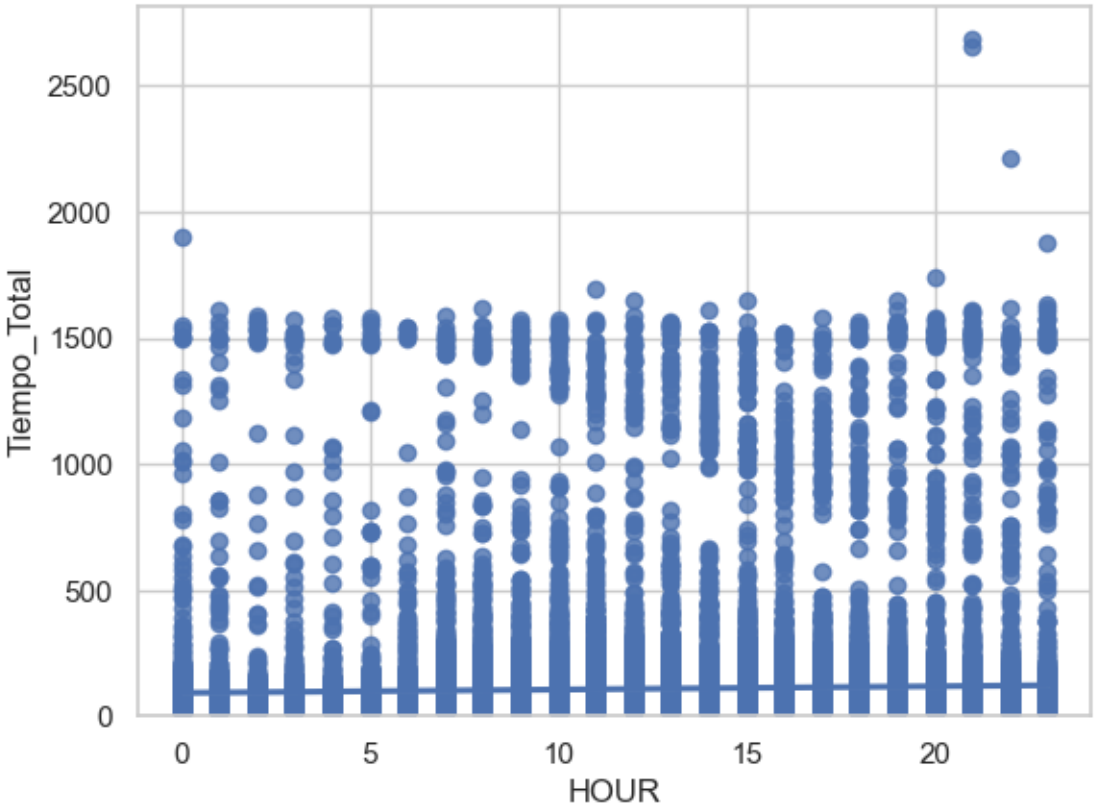
```
c:\Users\Victor.Gomez\AppData\Local\Programs\Python\Python311\Lib\site-
packages\seaborn\axisgrid.py:118: UserWarning: The figure layout has changed to
tight
  self._figure.tight_layout(*args, **kwargs)
```

[ ]: <seaborn.axisgrid.PairGrid at 0x1b6b1f7b010>

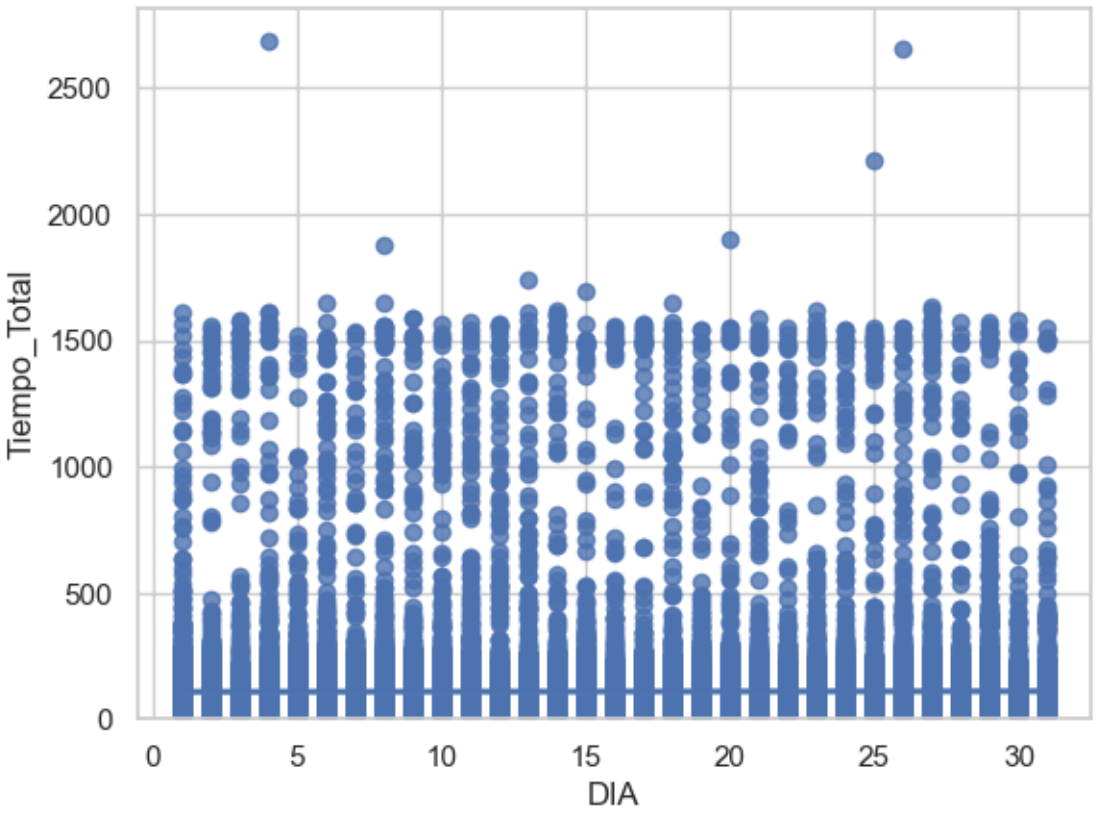


Visualicemos el diagrama de dispersión de tamaño del motor (engine-size) y precio (price)

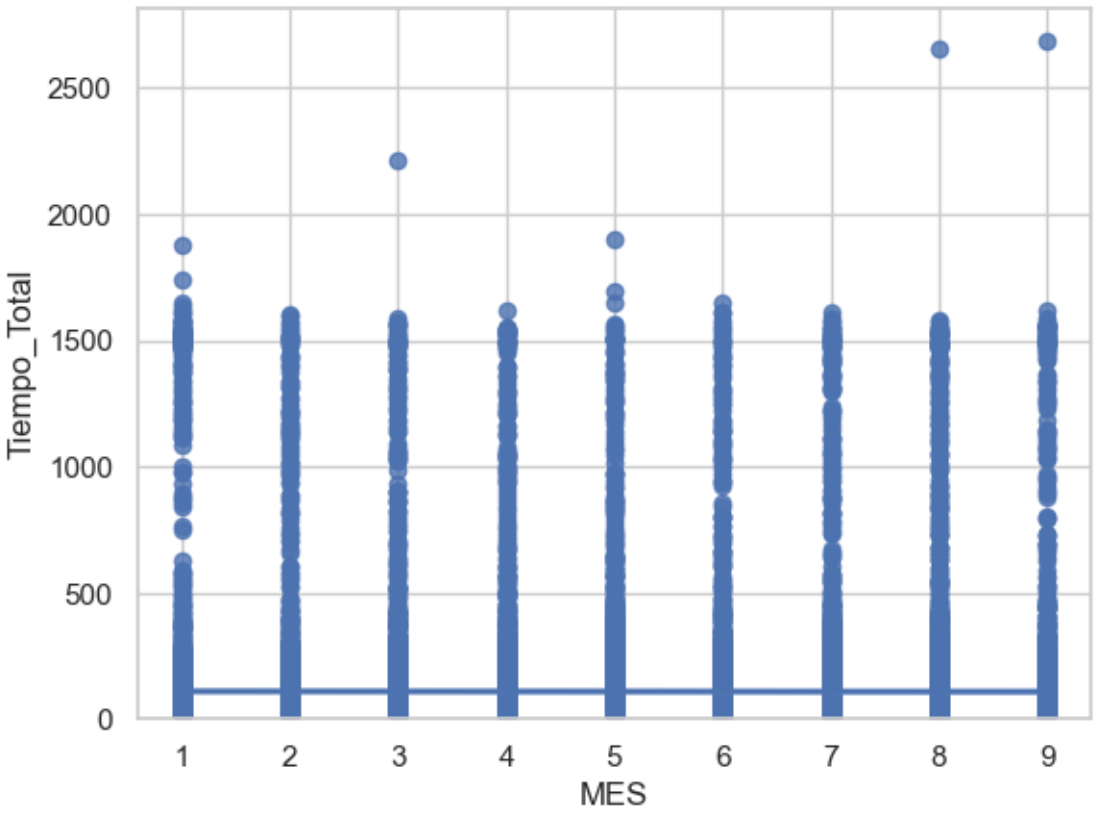
[ ]:
```
# Luego, crea el gráfico de regresión
sns.set(style="whitegrid")
sns.regplot(x="HOUR", y="Tiempo_Total", data=df)
plt.ylim(0,)
plt.show()
```



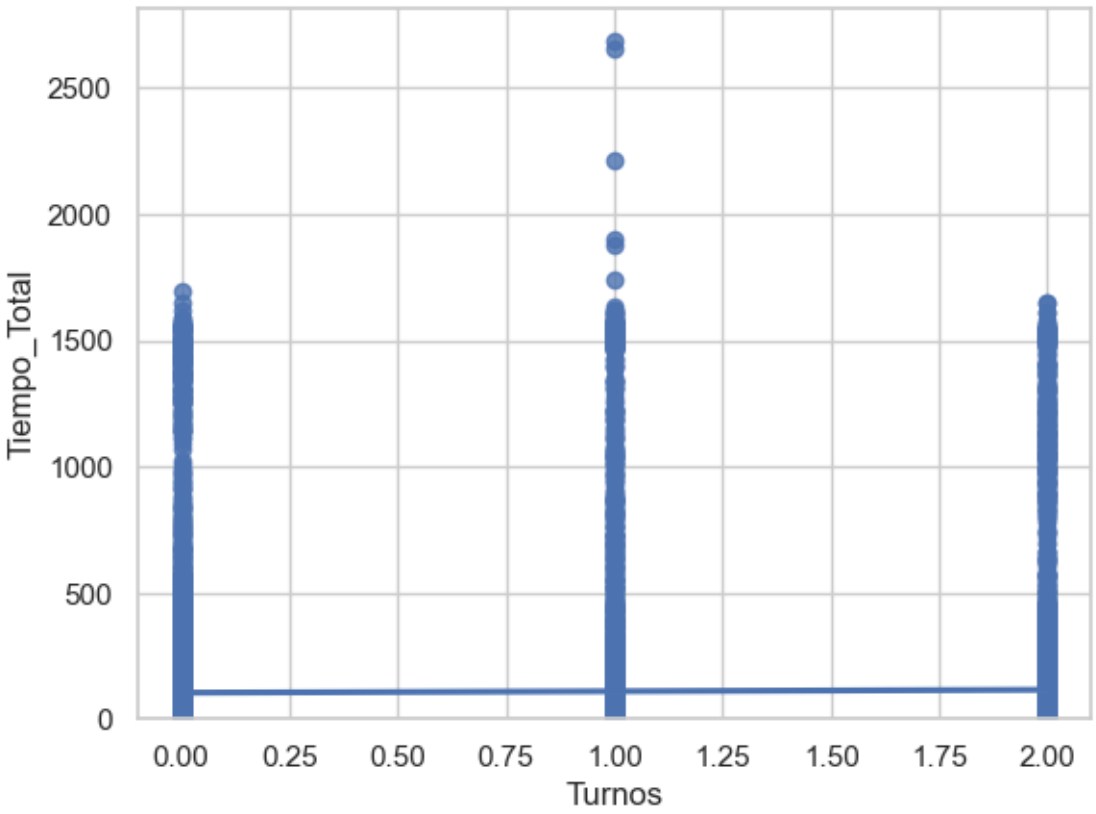
[ ]:
```
# Luego, crea el gráfico de regresión
sns.set(style="whitegrid")
sns.regplot(x="DIA", y="Tiempo_Total", data=df)
plt.ylim(0,)
plt.show()
```



[ ]:
```
# Luego, crea el gráfico de regresión
sns.set(style="whitegrid")
sns.regplot(x="MES", y="Tiempo_Total", data=df)
plt.ylim(0,)
plt.show()
```



[ ]:
```
# Primero, asegúrate de que tus variables categóricas estén ordenadas
df['Turnos'] = df['Turnos'].astype('category')
df['Turnos'] = df['Turnos'].cat.as_ordered()

# Convierte la columna categórica en una variable numérica
df['Turnos'] = df['Turnos'].cat.codes

# Luego, crea el gráfico de regresión
sns.set(style="whitegrid")
sns.regplot(x="Turnos", y="Tiempo_Total", data=df)
plt.ylim(0,)
plt.show()
```



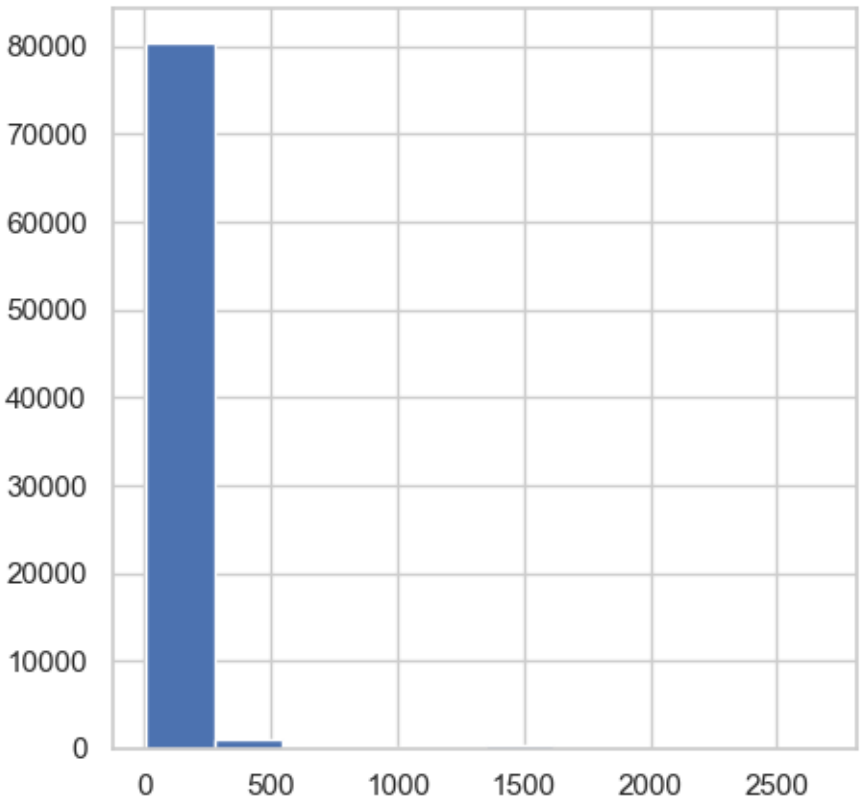
A medida que aumenta el tamaño del motor, aumenta el precio: esto indica una correlación directa positiva entre estas dos variables. El tamaño del motor parece ser un buen predictor de precio ya que la línea de regresión es casi una línea diagonal perfecta.

Visualicemos ahora highway-mpg y price. A medida que aumenta highway-mpg, el precio baja: esto indica un relación inversa/negativa entre estas dos variables. highway-mpg podría predecir el precio.

El gráfico de histograma también permite realizar un análisis sobre la variable numérica. Se utiliza el tipo de gráfico de Matplotlib hist sobre la variable age.

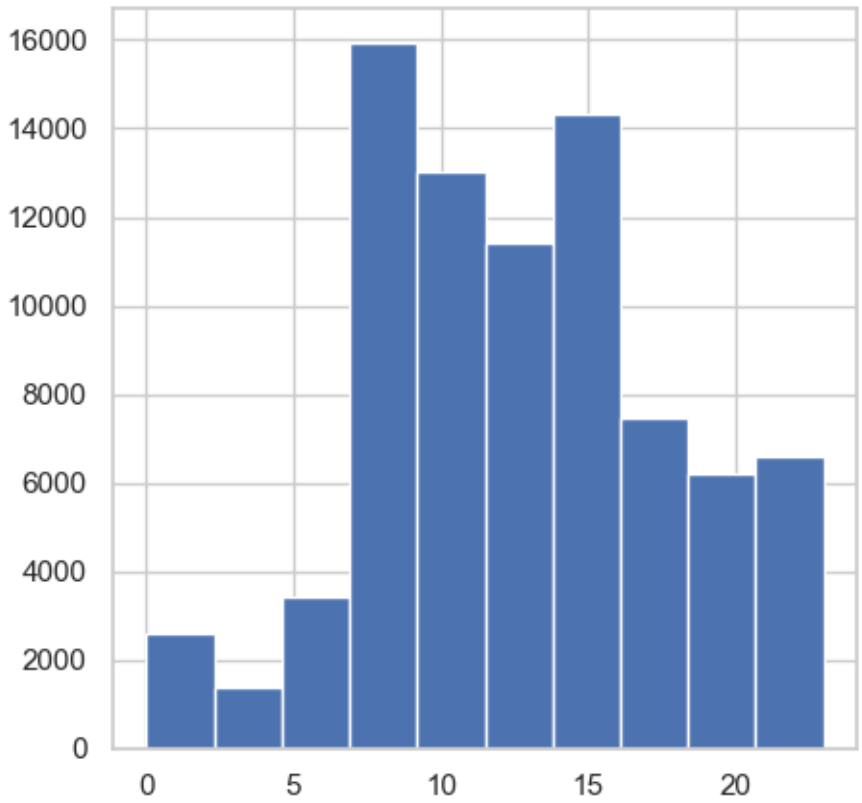
[ ]:
```
df['Tiempo_Total'].hist(figsize = (5,5))
plt.show
```

[ ]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```



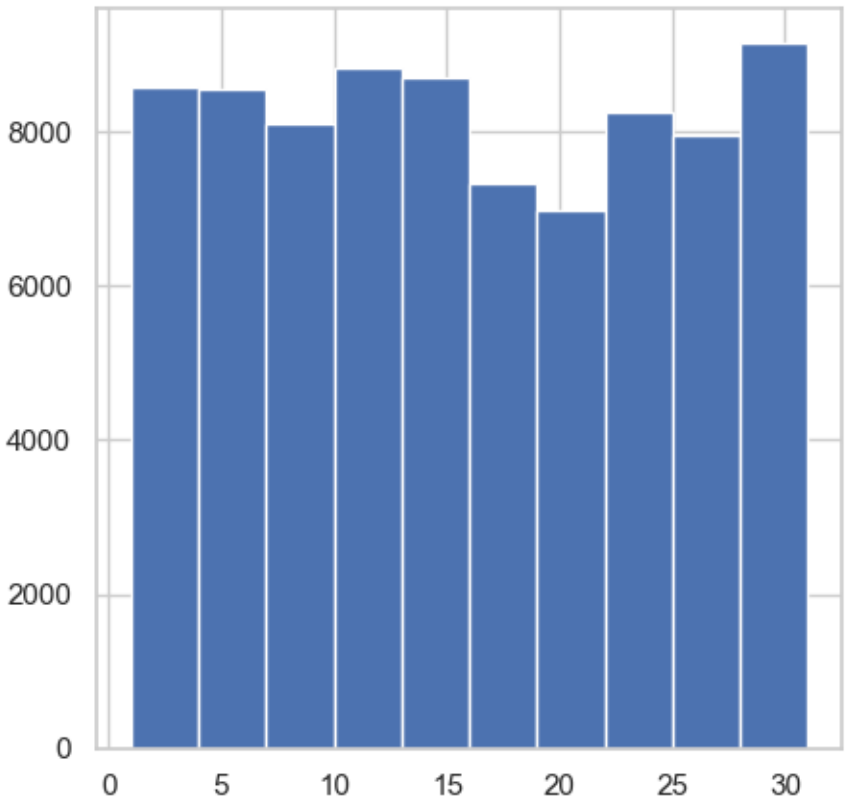
[ ]:
```
df['HOUR'].hist(figsize = (5,5))
plt.show
```

[ ]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```



[ ]:
```
df['DIA'].hist(figsize = (5,5))
plt.show
```

[ ]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```



Variables categóricas

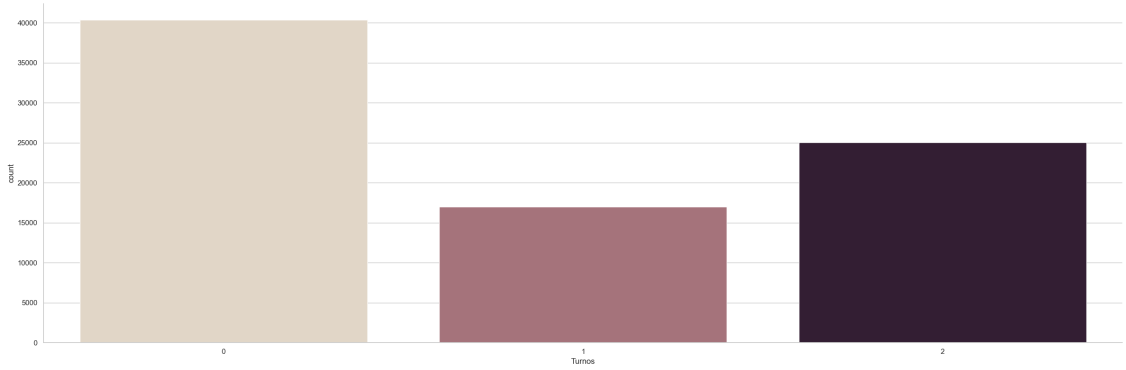
Estas son variables que describen una 'característica' de una unidad de datos y se seleccionan de un pequeño grupo de categorías. Las variables categóricas pueden tener el tipo objeto o int64. Una buena forma de visualizar variables categóricas es mediante el uso de diagramas de caja.

Mostremos un solo atributo, y contemos cuantos ejemplos hay de cada categoría. En este caso el atributo make que es la marca del carro:

[ ]:
```
sns.set(style="whitegrid")
sns.catplot(x="Turnos", kind="count", palette="ch:.25", data=df, height = 8,␣
 ↪aspect = 3)
```

```
c:\Users\Victor.Gomez\AppData\Local\Programs\Python\Python311\Lib\site-
packages\seaborn\axisgrid.py:118: UserWarning: The figure layout has changed to
tight
  self._figure.tight_layout(*args, **kwargs)
```

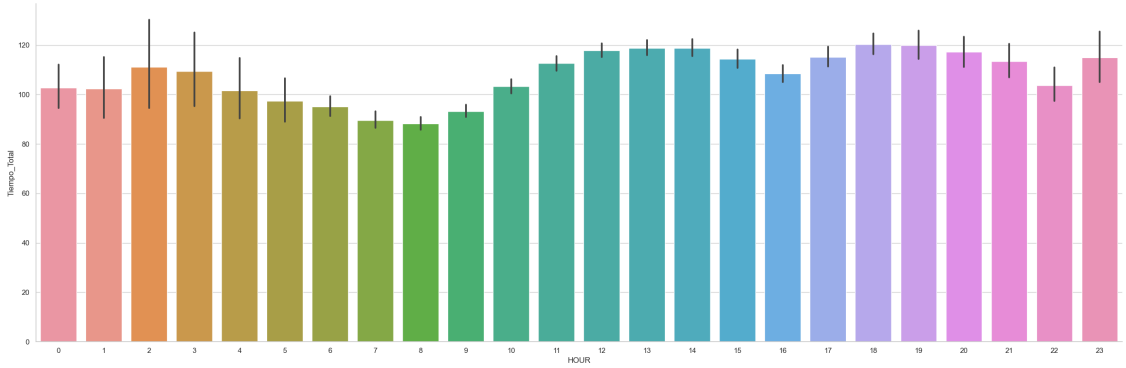
[ ]:
```
<seaborn.axisgrid.FacetGrid at 0x1b6b9ffa810>
```



[ ]:
```
sns.set(style="whitegrid")
sns.catplot(x="HOUR", y="Tiempo_Total", hue_order="class", kind="bar", data=df,␣
 ↪height = 8, aspect = 3)
```

```
c:\Users\Victor.Gomez\AppData\Local\Programs\Python\Python311\Lib\site-
packages\seaborn\axisgrid.py:118: UserWarning: The figure layout has changed to
tight
  self._figure.tight_layout(*args, **kwargs)
```

[ ]:
```
<seaborn.axisgrid.FacetGrid at 0x1b6ba008e50>
```

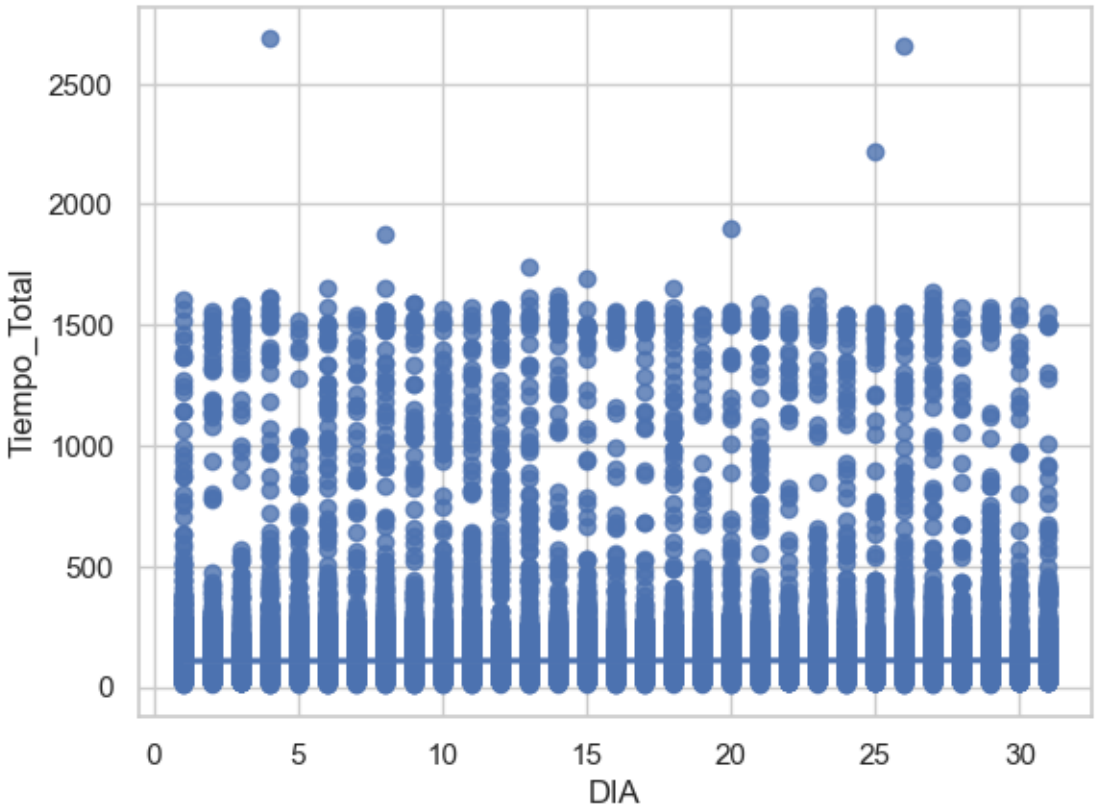


AHORA TÚ:

Visualice las variables stroke y price utilizando regplot. ¿Qué tipo de relación hay entre las variables?

[ ]:
```
# Escribe tu código aquí y presiona Shift+Enter para ejecutar

sns.set(style="whitegrid")
sns.regplot(x="DIA", y="Tiempo_Total", data=df)
```

[ ]: <Axes: xlabel='DIA', ylabel='Tiempo_Total'>



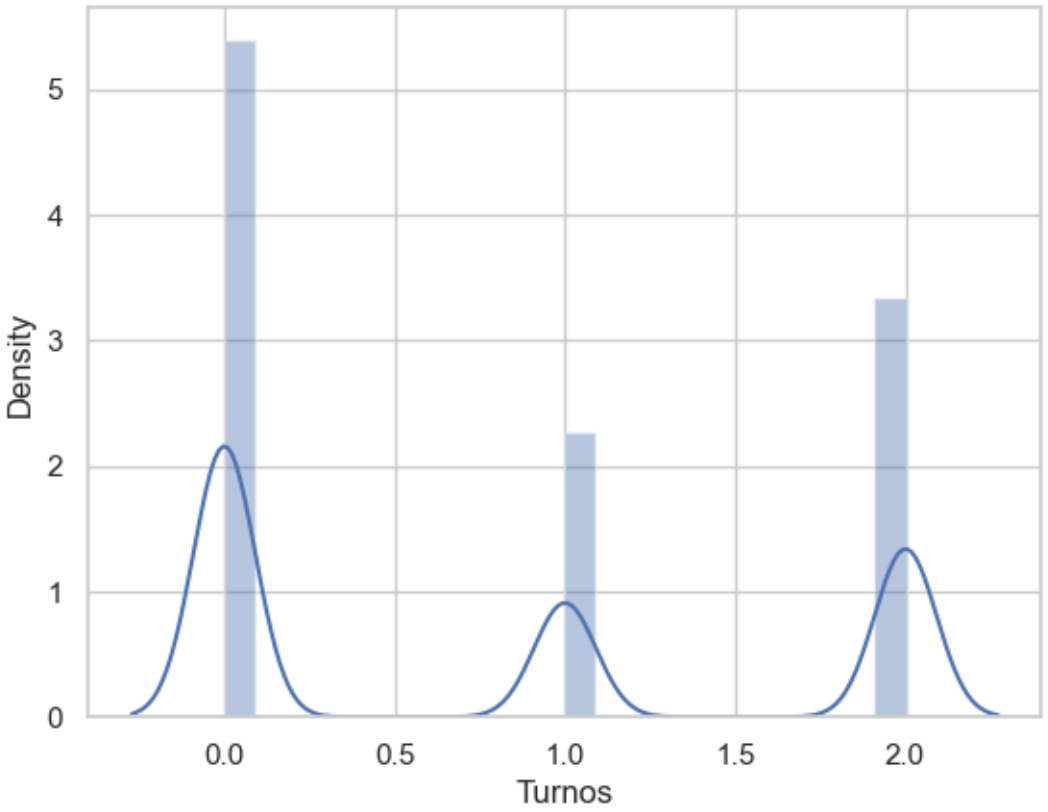
##Graficar las estadísticas

Podemos identificar la simentría de los datos utilizando gráficos de histograma:

[ ]:
```
#Histograma del atributo "length"
sns.distplot(df.Turnos)
```

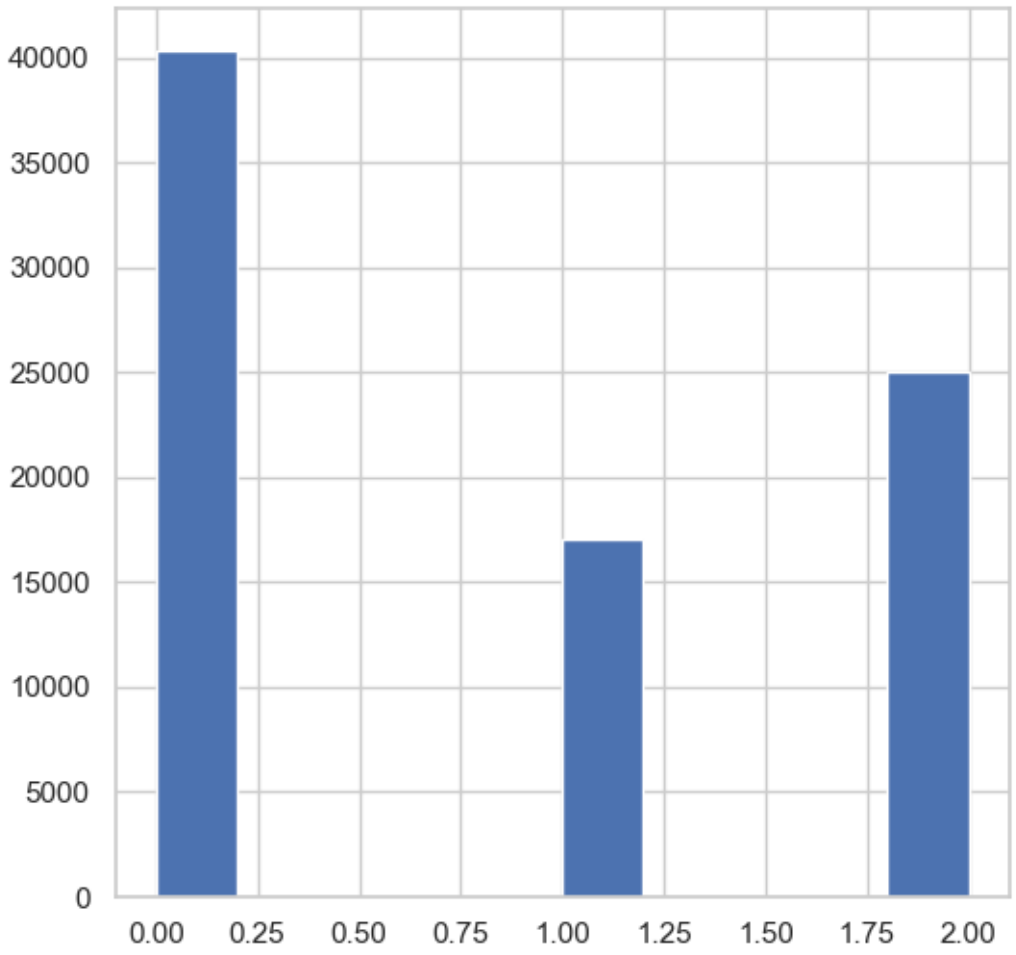
```
C:\Users\Victor.Gomez\AppData\Local\Temp\ipykernel_14576\1764428100.py:2:
UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df.Turnos)
```

[ ]: <Axes: xlabel='Turnos', ylabel='Density'>



[ ]:
```
df['Turnos'].hist(figsize = (6,6))
plt.show
```

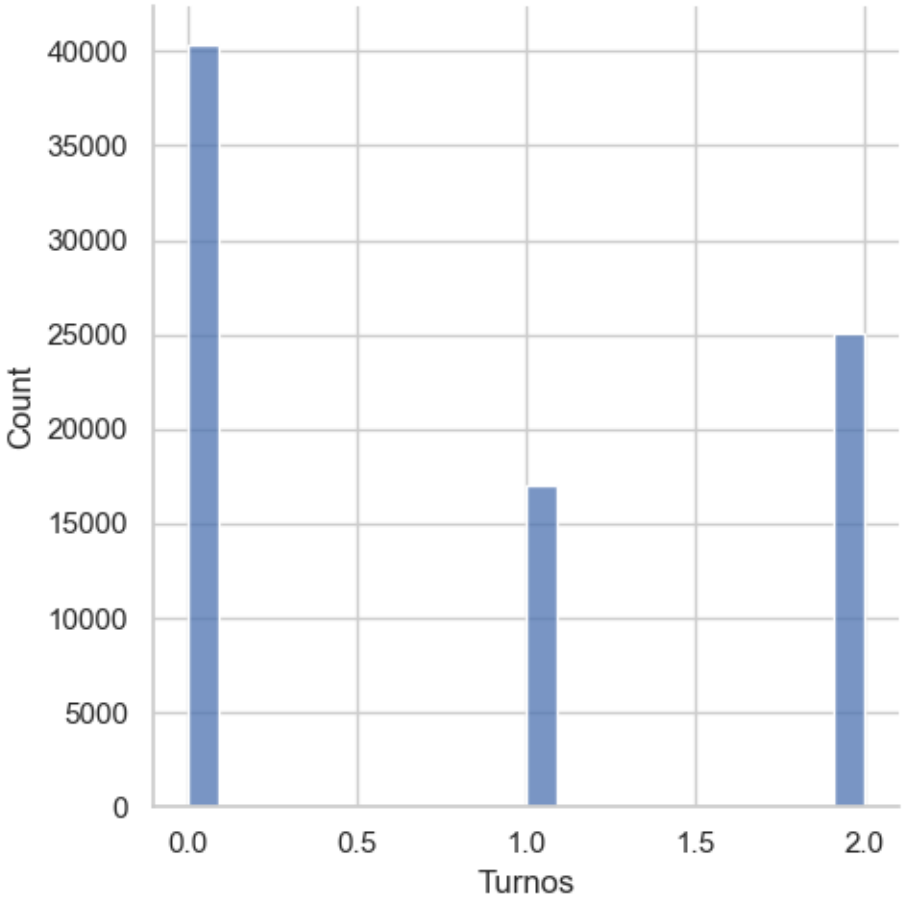
[ ]: <function matplotlib.pyplot.show(close=None, block=None)>



[ ]:
```
sns.displot(df['Turnos'])
```

```
c:\Users\Victor.Gomez\AppData\Local\Programs\Python\Python311\Lib\site-
packages\seaborn\axisgrid.py:118: UserWarning: The figure layout has changed to
tight
  self._figure.tight_layout(*args, **kwargs)
```

[ ]:
```
<seaborn.axisgrid.FacetGrid at 0x1b6ba034850>
```



[ ]:
```
mean = df['Turnos'].mean()
median = df['Turnos'].median()
mode = df['Turnos'].mode()
skew = df['Turnos'].skew()
kurt = df['Turnos'].kurt()
print("La media es:", mean)
print("La mediana es:", median)
print("La moda es:", mode)
print("El sesgo es:", skew)
print("La kurtosis es:", kurt)
```

```
La media es: 0.8142229664142885
La mediana es: 1.0
La moda es: 0    0
Name: Turnos, dtype: int8
El sesgo es: 0.3684999729645169
La kurtosis es: -1.5835310691072397
```

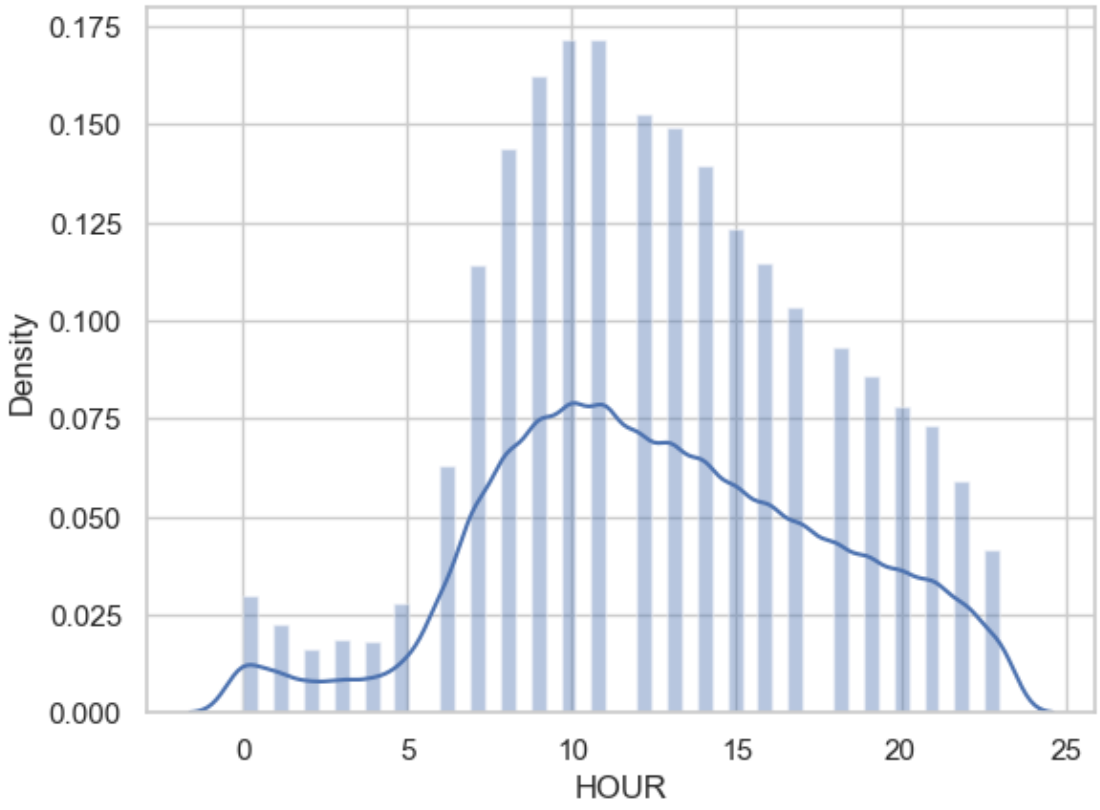
AHORA TÚ:

Seleccione una variable y visualice su histograma. ¿Qué tipo de simentría tiene?

[ ]:
```
# Escribe tu código aquí y presiona Shift+Enter para ejecutar
sns.distplot(df.HOUR)
```

```
C:\Users\Victor.Gomez\AppData\Local\Temp\ipykernel_14576\8436205.py:2:
UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df.HOUR)
```

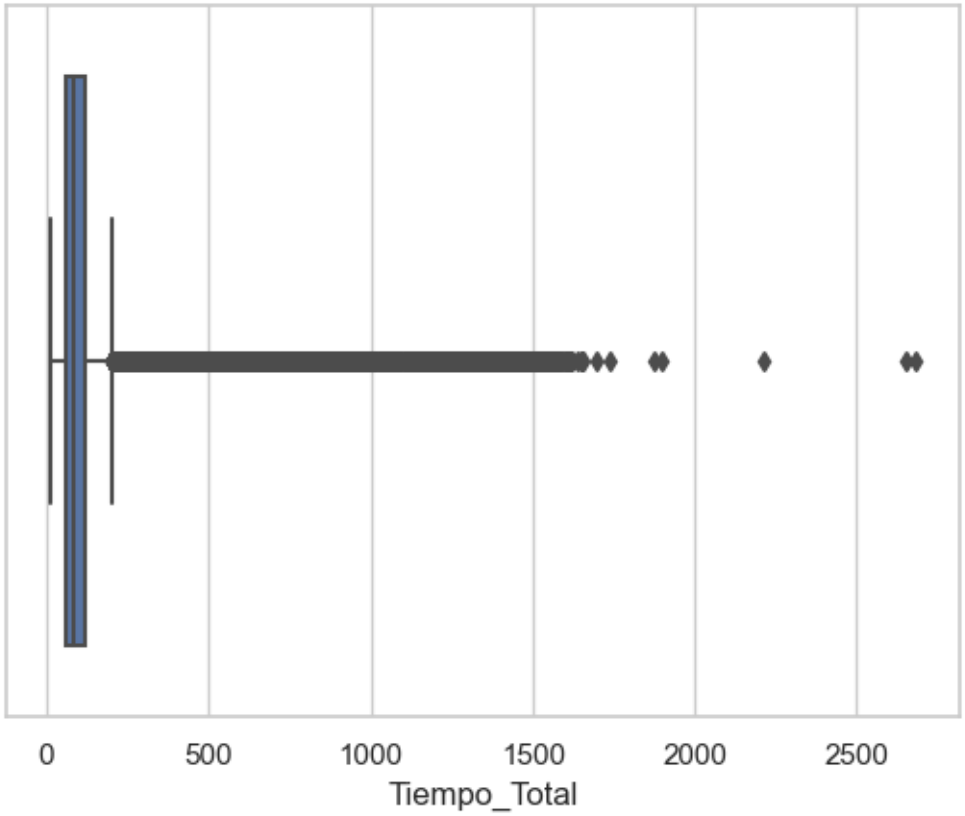
[ ]: `<Axes: xlabel='HOUR', ylabel='Density'>`



La dispersión de datos se puede comprobar también mediante los gráficos de tipo boxplot:

[ ]:
```
sns.boxplot(x=df.Tiempo_Total)
```

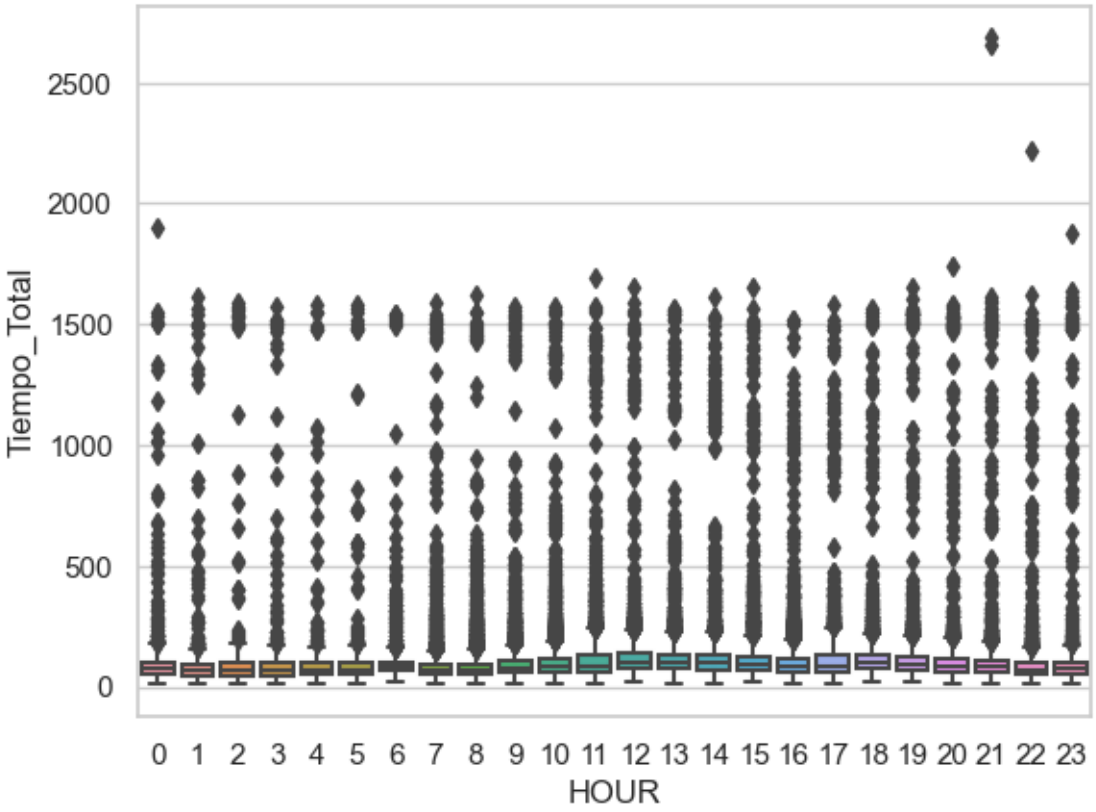
[ ]:
```
<Axes: xlabel='Tiempo_Total'>
```



Representar más de una gráfico tipo boxplot permite comparar la dispersión de los datos al poder ver los resultados de forma conjunta. Veamos la relación entre body-style y price.

[ ]:
```
sns.boxplot(x="HOUR", y="Tiempo_Total", data=df)
```

[ ]:
```
<Axes: xlabel='HOUR', ylabel='Tiempo_Total'>
```



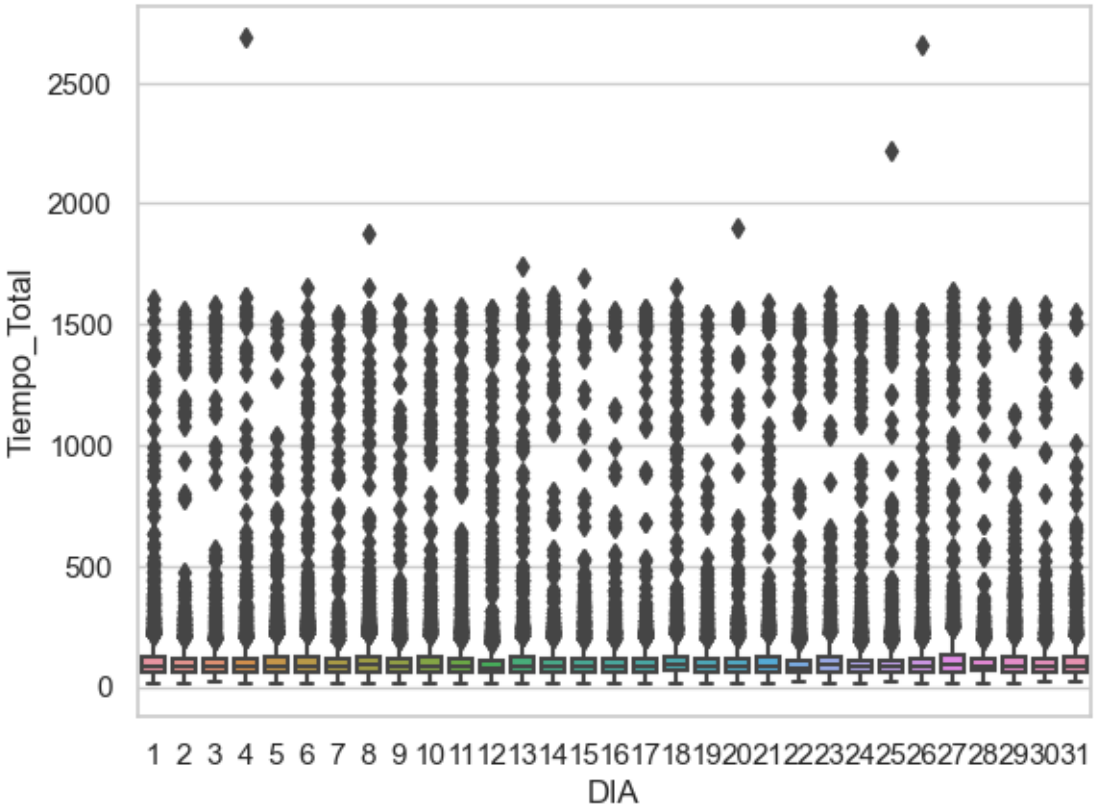
Vemos que las distribuciones de precios entre las diferentes categorías de estilo de cuerpo tienen una superposición significativa, por lo que el estilo de cuerpo no sería un buen predictor del precio.

AHORA TÚ:

Visualice las variables engine-location y price utilizando un gráfico de BoxPlot

[ ]:
```
# Escribe tu código aquí y presiona Shift+Enter para ejecutar
sns.boxplot(x="DIA", y="Tiempo_Total", data=df)
```

[ ]:
```
<Axes: xlabel='DIA', ylabel='Tiempo_Total'>
```



##Exportar los datos

De la misma forma, Pandas nos permite guardar el conjunto en formato CSV con el método dataframe.to_csv(), puede añadir la ruta al archivo y el nombre con comillas dentro de los corchetes.

Por ejemplo, si guarda el dataframe df como automobile.csv en su equipo local o en este caso dentro de Google Colab, podría usar la sintaxis siguiente:

[ ]:
```
path="URGENCIAS.csv"
df.to_csv(path)
```

Podemos leer y guardar con otros formatos y usar funciones similares a pd.read_csv() y df.to_csv() para otros formatos de datos, las funciones se muestran en la siguiente tabla:

Data Formate	Read	Save
csv	`pd.read_csv()`	`df.to_csv()`
json	`pd.read_json()`	`df.to_json()`
excel	`pd.read_excel()`	`df.to_excel()`
hdf	`pd.read_hdf()`	`df.to_hdf()`
sql	`pd.read_sql()`	`df.to_sql()`
...	...	...

##Links de ayuda interesantes:

Graficos Seaborn: https://seaborn.pydata.org/examples/index.html

Plotting with categorical data: https://seaborn.pydata.org/tutorial/categorical.html

Visualizing statistical relationships: https://seaborn.pydata.org/tutorial/relational.html

Funciones Generales y Ayuda Pandas: https://pandas.pydata.org/pandas-docs/stable/reference/general_functions.html

Ayuda de Pandas para función read_csv: https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read_csv.html?highlight=pandas%20read_csv#pandas.read_csv

How to Perform Exploratory Data Analysis with Seaborn: https://towardsdatascience.com/how-to-perform-exploratory-data-analysis-with-seaborn-97e3413e841d

modelo-de-teoria-de-colas-3-oct

November 23, 2023

[2]:
```
import pandas as pd
import os
import re
import simpy
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import seaborn as sns  # Para mejorar la apariencia de las gráficas
```

[11]:
```
# ruta de archivos
files = os.listdir("c:\\archivos\proyecto")
os.chdir(r'C:\archivos\proyecto')

if not os.path.exists('Errores'):
    os.makedirs('Errores')

if not os.path.exists('Buenos'):
    os.mkdir('Buenos')

df_total = pd.read_csv('ind_urgencias_final_2023.txt', sep=';')

# convertir a variables categoricas
df_total['Turnos'] = df_total['Turnos'].astype('category')
df_total['DIA_SEMANA'] = df_total['DIA_SEMANA'].astype('category')
df_total['CENTRO_ATENCION'] = df_total['CENTRO_ATENCION'].astype('category')

# filtros
Mes_nuevo = 4
Centros = 'JT'
dia_desde = 1
dia_hasta = 31

# Aplicar filtros múltiples al DataFrame
df_hearth = df_total[(df_total['MES'] == Mes_nuevo) &␣
 ↪(df_total['CENTRO_ATENCION'] == Centros) & (df_total['DIA'] >= dia_desde) &␣
 ↪(df_total['DIA'] <= dia_hasta)]
#df_hearth.to_csv('ind_urgencias_final_2023.txt', sep=';', index=False)
```

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

```
<class 'pandas.core.frame.DataFrame'>
Index: 329 entries, 14959 to 49545
Data columns (total 25 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   FECHA_LLEGADA               329 non-null    object
 1   FECHA_TRIAGE                329 non-null    object
 2   FECHA_INGRESO               329 non-null    object
 3   FECHA_ATENCION              329 non-null    object
 4   TIEMPO_DGTURNO_A_TRIAGE     329 non-null    object
 5   TIEMPO_TRIAGE_A_INGRESO     329 non-null    object
 6   TIEMPO_INGRESO_A_CONSULTA   329 non-null    object
 7   TIEMPO_TOTAL                329 non-null    object
 8   Tiempo_Minutos_Total        329 non-null    object
 9   CENTRO_ATENCION             329 non-null    category
 10  CLASIFICACION_TRIAGE        329 non-null    int64
 11  PACIENTE_#_DOCUMENTO        329 non-null    object
 12  EDAD                        329 non-null    int64
 13  EDAD_RANGO                  329 non-null    object
 14  SEXO                        329 non-null    object
 15  RÉGIMEN PACIENTE            329 non-null    object
 16  NOMBRE_ENTIDAD              329 non-null    object
 17  MEDICO                      329 non-null    int64
 18  AÑO                         329 non-null    int64
 19  MES                         329 non-null    int64
 20  DIA_SEMANA                  329 non-null    category
 21  HOUR                        329 non-null    int64
 22  Turnos                      329 non-null    category
 23  TIME                        329 non-null    object
 24  DIA                         329 non-null    int64
dtypes: category(3), int64(7), object(15)
memory usage: 60.9+ KB
```

[17]:
```
df_hearth
```

[17]:

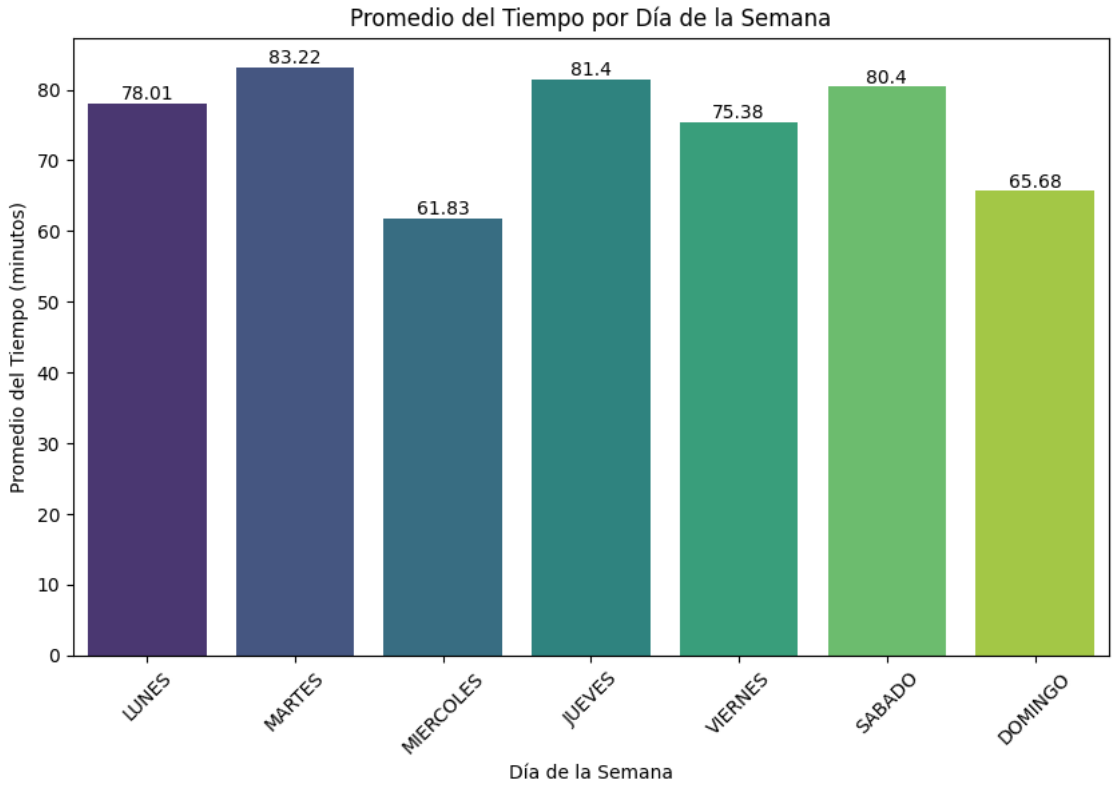
	FECHA_LLEGADA	FECHA_TRIAGE	\
14959	2023-04-01 08:35:51.963	2023-04-01 09:08:17.430	
15063	2023-04-01 12:54:41.983	2023-04-01 13:25:24.897	
15110	2023-04-02 06:05:46.703	2023-04-02 06:23:11.077	
15158	2023-04-02 08:29:50.517	2023-04-02 08:34:36.263	
15319	2023-04-02 16:30:51.517	2023-04-02 16:59:33.493	
…	…	…	
48654	2023-04-28 08:32:59.630	2023-04-28 08:59:13.710	
48674	2023-04-28 09:00:08.670	2023-04-28 09:35:01.730	
49225	2023-04-29 17:47:19.740	2023-04-29 18:40:28.527	

```
49430  2023-04-30 14:12:59.977  2023-04-30 14:26:59.890
49545  2023-04-30 20:36:39.230  2023-04-30 20:52:28.357

                FECHA_INGRESO           FECHA_ATENCION  \
14959  2023-04-01 09:09:41.733  2023-04-01 10:10:18.230
15063  2023-04-01 13:25:35.010  2023-04-01 13:46:57.037
15110  2023-04-02 06:23:58.850  2023-04-02 07:16:28.273
15158  2023-04-02 08:38:36.257  2023-04-02 09:27:41.437
15319  2023-04-02 17:18:34.363  2023-04-02 17:58:48.950
…                        …                       …
48654  2023-04-28 09:07:03.430  2023-04-28 09:27:30.830
48674  2023-04-28 09:39:46.297  2023-04-28 09:48:17.703
49225  2023-04-29 18:41:21.453  2023-04-29 19:23:08.083
49430  2023-04-30 14:30:58.183  2023-04-30 15:07:48.650
49545  2023-04-30 20:53:03.923  2023-04-30 21:12:47.130

      TIEMPO_DGTURNO_A_TRIAGE TIEMPO_TRIAGE_A_INGRESO  \
14959                 0:32:26                 0:01:24
15063                 0:30:43                 0:00:11
15110                 0:17:25                 0:00:47
15158                 0:04:46                 0:04:00
15319                 0:28:42                 0:19:01
…                       …                       …
48654                 0:26:14                 0:07:50
48674                 0:34:53                 0:04:45
49225                 0:53:09                 0:00:53
49430                 0:14:00                 0:03:59
49545                 0:15:49                 0:00:35

      TIEMPO_INGRESO_A_CONSULTA TIEMPO_TOTAL Tiempo_Minutos_Total  \
14959                   1:00:37      1:34:27                94,45
15063                   0:21:22      0:52:16                52,27
15110                   0:52:30      1:10:42                70,70
15158                   0:49:05      0:57:51                57,85
15319                   0:40:14      1:27:57                87,95
…                         …          …                    …
48654                   0:20:27      0:54:31                54,52
48674                   0:08:31      0:48:09                48,15
49225                   0:41:47      1:35:49                95,82
49430                   0:36:50      0:54:49                54,82
49545                   0:19:44      0:36:08                36,13

      CENTRO_ATENCION  …  RÉGIMEN PACIENTE NOMBRE_ENTIDAD   MEDICO   AÑO  \
14959              JT  …      CONTRIBUTIVO          EPSC34   10763  2023
15063              JT  …        SUBSIDIADO          EPSS10   11040  2023
15110              JT  …        SUBSIDIADO          EPSC34    9705  2023
15158              JT  …        SUBSIDIADO          EPSC34    5255  2023
```

```
15319              JT  …      CONTRIBUTIVO          EPS002   11040  2023
…                 …  …             …              …     …   …
48654              JT  …        SUBSIDIADO          EPSC34   11040  2023
48674              JT  …        SUBSIDIADO          EPSC34   10662  2023
49225              JT  …        SUBSIDIADO          EPSC34   10662  2023
49430              JT  …        SUBSIDIADO          EPSC34   10763  2023
49545              JT  …      CONTRIBUTIVO          EPSC34   10662  2023

      MES DIA_SEMANA HOUR  Turnos                       TIME  DIA
14959   4     SABADO    8  MAÑANA  2023-04-01 08:35:51.963    1
15063   4     SABADO   12  MAÑANA  2023-04-01 12:54:41.983    1
15110   4    DOMINGO    6   NOCHE  2023-04-02 06:05:46.703    2
15158   4    DOMINGO    8  MAÑANA  2023-04-02 08:29:50.517    2
15319   4    DOMINGO   16   TARDE  2023-04-02 16:30:51.517    2
…     ..        …  …     …                          …  …
48654   4    VIERNES    8  MAÑANA  2023-04-28 08:32:59.630   28
48674   4    VIERNES    9  MAÑANA  2023-04-28 09:00:08.670   28
49225   4     SABADO   17   TARDE  2023-04-29 17:47:19.740   29
49430   4    DOMINGO   14   TARDE  2023-04-30 14:12:59.977   30
49545   4    DOMINGO   20   NOCHE  2023-04-30 20:36:39.230   30

[329 rows x 25 columns]
```

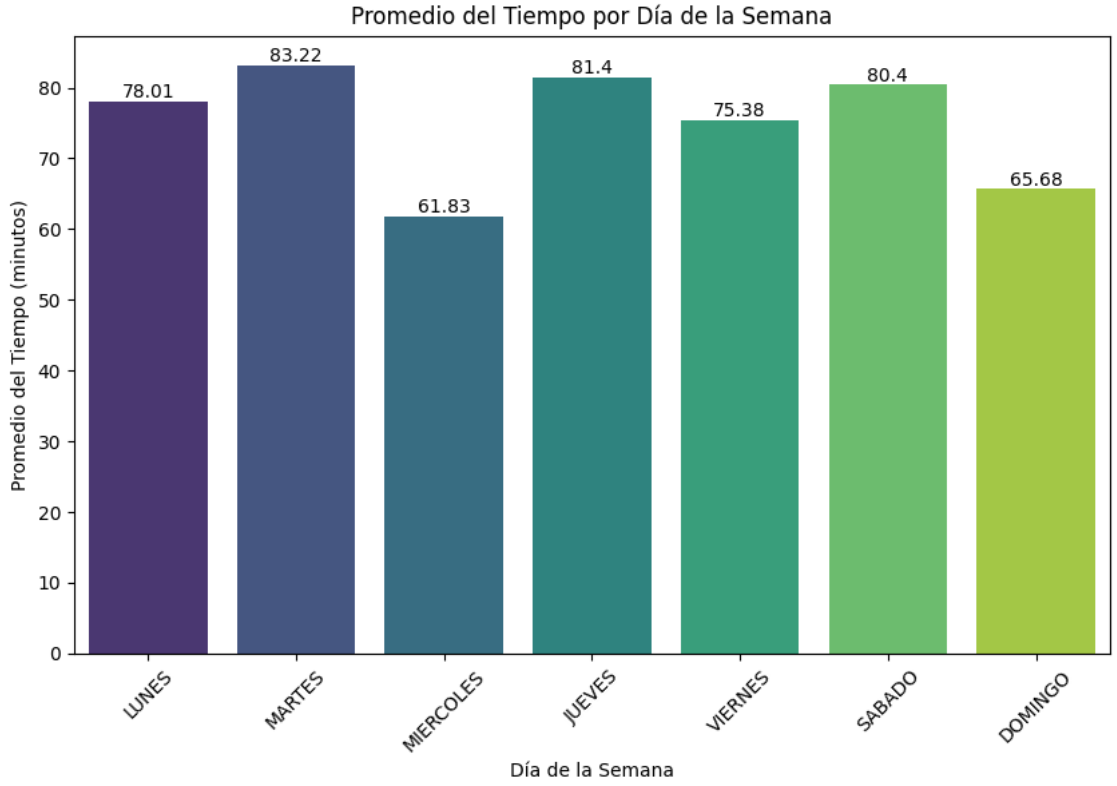
[54]:
```python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

df = df_hearth[['DIA_SEMANA', 'Tiempo_Minutos_Total']].copy()

# Definir el orden deseado de los días de la semana
orden_dias_semana = ['LUNES', 'MARTES', 'MIERCOLES', 'JUEVES', 'VIERNES',␣
 ↪'SABADO', 'DOMINGO']

# Reordenar la columna DIA_SEMANA para que coincida con el orden deseado
df.loc[:, 'DIA_SEMANA'] = pd.Categorical(df['DIA_SEMANA'],␣
 ↪categories=orden_dias_semana, ordered=True)


df['Tiempo_Minutos_Total'] = df['Tiempo_Minutos_Total'].str.replace(',', '.',␣
 ↪regex=True)
df["Tiempo_Minutos_Total"] = pd.
 ↪to_numeric(df["Tiempo_Minutos_Total"],errors='coerce')

median = df['Tiempo_Minutos_Total'].median()
df.loc[df['Tiempo_Minutos_Total'] > 420, 'Tiempo_Minutos_Total'] = median

median = df['Tiempo_Minutos_Total'].median()
```

```
df.loc[df['Tiempo_Minutos_Total'] < 0, 'Tiempo_Minutos_Total'] = median


# Agrupar por día de la semana y calcular el promedio del Tiempo_total_minutos␣
 ↪en cada grupo
promedio_tiempo_por_grupo =df.groupby(['DIA_SEMANA'])['Tiempo_Minutos_Total'].
 ↪mean().reset_index()

# Renombrar la columna del promedio
promedio_tiempo_por_grupo = promedio_tiempo_por_grupo.
 ↪rename(columns={'Tiempo_Minutos_Total': 'Promedio_Tiempo'})

plt.figure(figsize=(10, 6))
ax = sns.barplot(data=promedio_tiempo_por_grupo, x='DIA_SEMANA',␣
 ↪y='Promedio_Tiempo', palette='viridis')
plt.xlabel('Día de la Semana')
plt.ylabel('Promedio del Tiempo (minutos)')
plt.title('Promedio del Tiempo por Día de la Semana')
plt.xticks(rotation=45)

# Agregar etiquetas en las barras
for index, row in promedio_tiempo_por_grupo.iterrows():
    ax.annotate(str(round(row['Promedio_Tiempo'], 2)), (index,␣
 ↪row['Promedio_Tiempo']), ha='center', va='bottom')
plt.show()
```



[57]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

df = df_hearth[['DIA_SEMANA', 'Tiempo_Minutos_Total']].copy()

# Definir el orden deseado de los días de la semana
orden_dias_semana = ['LUNES', 'MARTES', 'MIERCOLES', 'JUEVES', 'VIERNES',␣
 ↪'SABADO', 'DOMINGO']

# Reordenar la columna DIA_SEMANA para que coincida con el orden deseado
df.loc[:, 'DIA_SEMANA'] = pd.Categorical(df['DIA_SEMANA'],␣
 ↪categories=orden_dias_semana, ordered=True)

df['Tiempo_Minutos'] = df['Tiempo_Minutos_Total']

df['Tiempo_Minutos'] = df['Tiempo_Minutos'].str.replace(',', '.', regex=True)
df["Tiempo_Minutos"] = pd.to_numeric(df["Tiempo_Minutos"],errors='coerce')

promedio = df['Tiempo_Minutos'].median()
df.loc[df['Tiempo_Minutos'] > 420, 'Tiempo_Minutos'] = promedio
df.loc[df['Tiempo_Minutos'] < 0, 'Tiempo_Minutos'] = promedio
```

```
# Agrupar por día de la semana y calcular el promedio del Tiempo_total_minutos␣
 ↪en cada grupo
promedio_tiempo_por_grupo =df.groupby(['DIA_SEMANA'])['Tiempo_Minutos'].mean().
 ↪reset_index()

# Renombrar la columna del promedio
promedio_tiempo_por_grupo = promedio_tiempo_por_grupo.
 ↪rename(columns={'Tiempo_Minutos': 'Promedio_Tiempo'})

plt.figure(figsize=(10, 6))
ax = sns.barplot(data=promedio_tiempo_por_grupo, x='DIA_SEMANA',␣
 ↪y='Promedio_Tiempo', palette='viridis')
plt.xlabel('Día de la Semana')
plt.ylabel('Promedio del Tiempo (minutos)')
plt.title('Promedio del Tiempo por Día de la Semana')
plt.xticks(rotation=45)

# Agregar etiquetas en las barras
for index, row in promedio_tiempo_por_grupo.iterrows():
    ax.annotate(str(round(row['Promedio_Tiempo'], 2)), (index,␣
 ↪row['Promedio_Tiempo']), ha='center', va='bottom')
plt.show()
```



[53]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

df = df_hearth[['DIA_SEMANA', 'Tiempo_Minutos_Total']].copy()

# Definir el orden deseado de los días de la semana
orden_dias_semana = ['LUNES', 'MARTES', 'MIERCOLES', 'JUEVES', 'VIERNES',␣
 ↪'SABADO', 'DOMINGO']

# Reordenar la columna DIA_SEMANA para que coincida con el orden deseado
df.loc[:, 'DIA_SEMANA'] = pd.Categorical(df['DIA_SEMANA'],␣
 ↪categories=orden_dias_semana, ordered=True)

# Convierte todos los valores en la columna 'Tiempo_Minutos_Total' a cadenas
# df['Tiempo_Minutos_Total'] = df['Tiempo_Minutos_Total'].astype(str)

# df['Tiempo_Minutos'] = df['Tiempo_Minutos_Total'].str.replace(',', '.',␣
 ↪regex=True)
df["Tiempo_Minutos"] = pd.to_numeric(df["Tiempo_Minutos_Total"],errors='coerce')
df
```

[53]:

	DIA_SEMANA	Tiempo_Minutos_Total	Tiempo_Minutos
14959	SABADO	94,45	NaN
15063	SABADO	52,27	NaN
15110	DOMINGO	70,70	NaN
15158	DOMINGO	57,85	NaN
15319	DOMINGO	87,95	NaN
…	…	…	…
48654	VIERNES	54,52	NaN
48674	VIERNES	48,15	NaN
49225	SABADO	95,82	NaN
49430	DOMINGO	54,82	NaN
49545	DOMINGO	36,13	NaN

[329 rows x 3 columns]

[52]:
```
df["Tiempo_Minutos"] = pd.to_numeric(df["Tiempo_Minutos_Total"],errors='coerce')
df
```

[52]:

	DIA_SEMANA	Tiempo_Minutos_Total	Tiempo_Minutos
14959	SABADO	94,45	NaN
15063	SABADO	52,27	NaN
15110	DOMINGO	70,70	NaN

```
15158    DOMINGO                57,85             NaN
15319    DOMINGO                87,95             NaN
…          …                    …                 …
48654    VIERNES                54,52             NaN
48674    VIERNES                48,15             NaN
49225     SABADO                95,82             NaN
49430    DOMINGO                54,82             NaN
49545    DOMINGO                36,13             NaN

[329 rows x 3 columns]
```

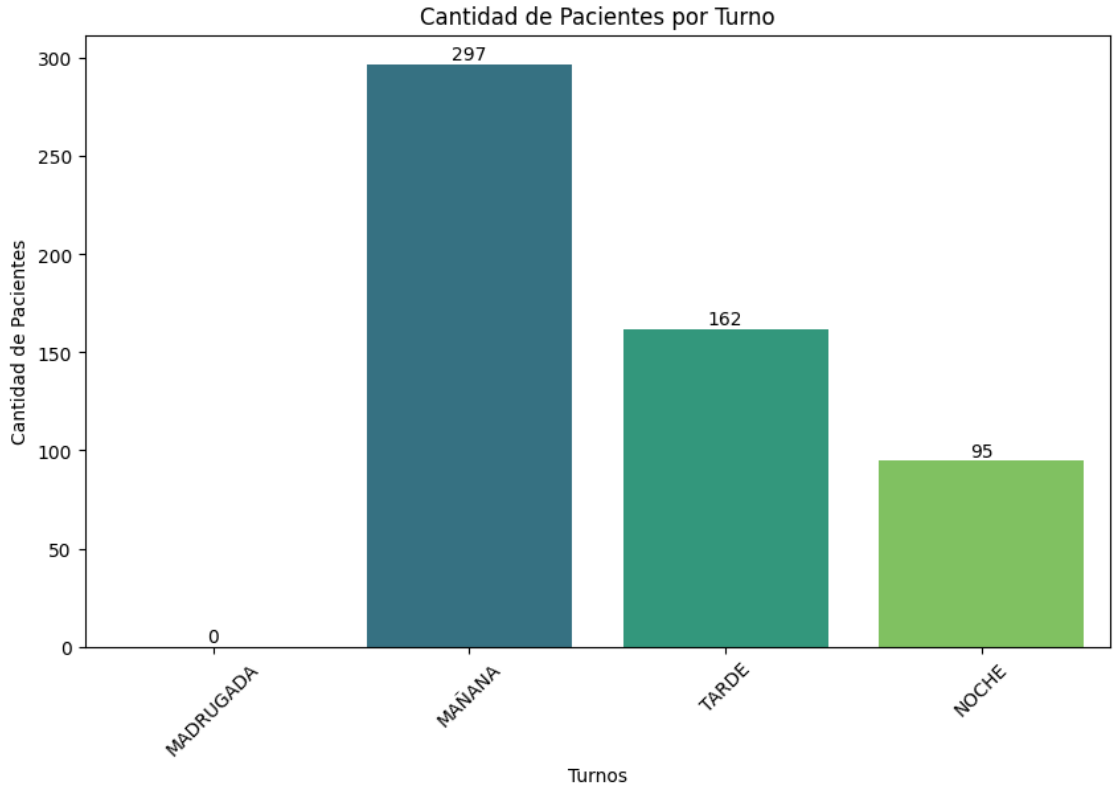
[49]:
```
promedio = df['Tiempo_Minutos'].median()
df.loc[df['Tiempo_Minutos'] > 420, 'Tiempo_Minutos'] = promedio
df.loc[df['Tiempo_Minutos'] < 0, 'Tiempo_Minutos'] = promedio
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
d:\Victor.Gomez\Downloads\modelo de teoria de colas 3.ipynb Cell 8 line 2

      <a href='vscode-notebook-cell:/d%3A/Victor.Gomez/Downloads/
 ↪modelo%20de%20teoria%20de%20colas%203.ipynb#X21sZmlsZQ%3D%3D?line=0'>1</a>␣
 ↪promedio = df['Tiempo_Minutos'].median()
----> <a href='vscode-notebook-cell:/d%3A/Victor.Gomez/Downloads/
 ↪modelo%20de%20teoria%20de%20colas%203.ipynb#X21sZmlsZQ%3D%3D?line=1'>2</a> df
 ↪loc[df['Tiempo_Minutos'] > 420, 'Tiempo_Minutos'] = promedio
      <a href='vscode-notebook-cell:/d%3A/Victor.Gomez/Downloads/
 ↪modelo%20de%20teoria%20de%20colas%203.ipynb#X21sZmlsZQ%3D%3D?line=2'>3</a> df
 ↪loc[df['Tiempo_Minutos'] < 0, 'Tiempo_Minutos'] = promedio

File c:\Users\Victor.
 ↪Gomez\AppData\Local\Programs\Python\Python311\Lib\site-packages\pandas\core\ops\common.
 ↪py:81, in _unpack_zerodim_and_defer.<locals>.new_method(self, other)
     77             return NotImplemented
     79 other = item_from_zerodim(other)
---> 81 return method(self, other)

File c:\Users\Victor.
 ↪Gomez\AppData\Local\Programs\Python\Python311\Lib\site-packages\pandas\core\arraylike.
 ↪py:56, in OpsMixin.__gt__(self, other)
     54 @unpack_zerodim_and_defer("__gt__")
     55 def __gt__(self, other):
---> 56     return self._cmp_method(other, operator.gt)

File c:\Users\Victor.
 ↪Gomez\AppData\Local\Programs\Python\Python311\Lib\site-packages\pandas\core\series.
 ↪py:6096, in Series._cmp_method(self, other, op)
   6093 rvalues = extract_array(other, extract_numpy=True, extract_range=True)
   6095 with np.errstate(all="ignore"):
-> 6096     res_values = ops.comparison_op(lvalues, rvalues, op)
```

```
   6098 return self._construct_result(res_values, name=res_name)

File c:\Users\Victor.
 ↪Gomez\AppData\Local\Programs\Python\Python311\Lib\site-packages\pandas\core\ops\array_ops.
 ↪py:293, in comparison_op(left, right, op)
    290     return invalid_comparison(lvalues, rvalues, op)
    292 elif is_object_dtype(lvalues.dtype) or isinstance(rvalues, str):
--> 293     res_values = comp_method_OBJECT_ARRAY(op, lvalues, rvalues)
    295 else:
    296     res_values = _na_arithmetic_op(lvalues, rvalues, op, is_cmp=True)

File c:\Users\Victor.
 ↪Gomez\AppData\Local\Programs\Python\Python311\Lib\site-packages\pandas\core\ops\array_ops.
 ↪py:82, in comp_method_OBJECT_ARRAY(op, x, y)
     80     result = libops.vec_compare(x.ravel(), y.ravel(), op)
     81 else:
---> 82     result = libops.scalar_compare(x.ravel(), y, op)
     83 return result.reshape(x.shape)

File c:\Users\Victor.
 ↪Gomez\AppData\Local\Programs\Python\Python311\Lib\site-packages\pandas\_libs\ops.
 ↪pyx:107, in pandas._libs.ops.scalar_compare()

TypeError: '>' not supported between instances of 'str' and 'int'
```

[6]:
```
# Grafica de Turnos
# Definir el orden deseado de los Turnos
orden_turnos = ['MADRUGADA', 'MAÑANA', 'TARDE', 'NOCHE']

# Reordenar la columna DIA_SEMANA para que coincida con el orden deseado
df_hearth.loc[:, 'Turnos'] = pd.Categorical(df_hearth['Turnos'],␣
 ↪categories=orden_turnos, ordered=True)

# Agrupar por turno y contar la cantidad de pacientes en cada grupo
pacientes_por_grupo1 = df_hearth.groupby(['Turnos'])['PACIENTE_#_DOCUMENTO'].
 ↪count().reset_index()
#pacientes_por_grupo = df_hearth.groupby(['DIA_SEMANA', 'Turnos',␣
 ↪'EDAD_RANGO'])['PACIENTE_#_DOCUMENTO'].count().reset_index()

# Renombrar la columna de conteo
pacientes_por_grupo1 = pacientes_por_grupo1.
 ↪rename(columns={'PACIENTE_#_DOCUMENTO': 'Cantidad_Pacientes'})

plt.figure(figsize=(10, 6))
ax = sns.barplot(data=pacientes_por_grupo1, x='Turnos', y='Cantidad_Pacientes',␣
 ↪palette='viridis')
plt.xlabel('Turnos')
```

```
plt.ylabel('Cantidad de Pacientes')
plt.title('Cantidad de Pacientes por Turno')
plt.xticks(rotation=45)

# Agregar etiquetas en las barras
for index, row in pacientes_por_grupo1.iterrows():
    ax.annotate(str(row['Cantidad_Pacientes']), (index,␣
 ↪row['Cantidad_Pacientes']), ha='center', va='bottom')
plt.show()

tiempo_real = sum(df_hearth['Tiempo_Minutos_Total'])
pacientes_total = df_hearth['PACIENTE_#_DOCUMENTO'].count()
tiempo_promedio_espera_mes = tiempo_real / pacientes_total if pacientes_total >␣
 ↪0 else 0
print("Datos Reales (historicos):")
print(f"Tiempo Real: {tiempo_real} pacientes por mes")
print(f"Total pacientes: {pacientes_total} pacientes por mes")
print(f"Tiempo de Espera Promedio: {tiempo_promedio_espera_mes} ")
```

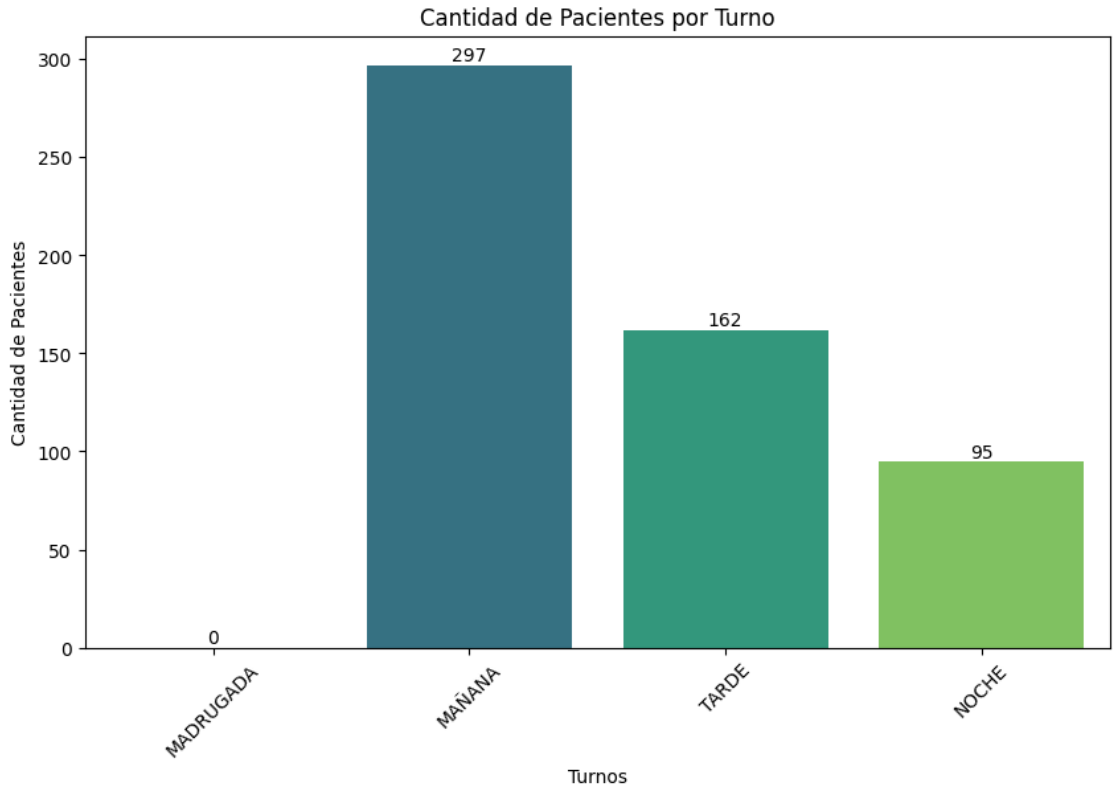



```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
```

```
d:\Victor.Gomez\Downloads\modelo de teoria de colas 3.ipynb Cell 4 line 2
      <a href='vscode-notebook-cell:/d%3A/Victor.Gomez/Downloads/
 ↪modelo%20de%20teoria%20de%20colas%203.ipynb#W3sZmlsZQ%3D%3D?line=23'>24</a>
 ↪ ax.annotate(str(row['Cantidad_Pacientes']), (index,␣
 ↪row['Cantidad_Pacientes']), ha='center', va='bottom')
      <a href='vscode-notebook-cell:/d%3A/Victor.Gomez/Downloads/
 ↪modelo%20de%20teoria%20de%20colas%203.ipynb#W3sZmlsZQ%3D%3D?line=24'>25</a>␣
 ↪plt.show()
---> <a href='vscode-notebook-cell:/d%3A/Victor.Gomez/Downloads/
 ↪modelo%20de%20teoria%20de%20colas%203.ipynb#W3sZmlsZQ%3D%3D?line=26'>27</a>␣
 ↪tiempo_real = sum(df_hearth['Tiempo_Minutos_Total'])
      <a href='vscode-notebook-cell:/d%3A/Victor.Gomez/Downloads/
 ↪modelo%20de%20teoria%20de%20colas%203.ipynb#W3sZmlsZQ%3D%3D?line=27'>28</a>␣
 ↪pacientes_total = df_hearth['PACIENTE_#_DOCUMENTO'].count()
      <a href='vscode-notebook-cell:/d%3A/Victor.Gomez/Downloads/
 ↪modelo%20de%20teoria%20de%20colas%203.ipynb#W3sZmlsZQ%3D%3D?line=28'>29</a>␣
 ↪tiempo_promedio_espera_mes = tiempo_real / pacientes_total if pacientes_total
 ↪> 0 else 0

TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

[9]:
```
# Grafica de Turnos
# Definir el orden deseado de los Turnos
orden_turnos = ['MADRUGADA', 'MAÑANA', 'TARDE', 'NOCHE']

# Reordenar la columna DIA_SEMANA para que coincida con el orden deseado
df_hearth.loc[:, 'Turnos'] = pd.Categorical(df_hearth['Turnos'],␣
 ↪categories=orden_turnos, ordered=True)

# Agrupar por turno y contar la cantidad de pacientes en cada grupo
pacientes_por_grupo1 = df_hearth.groupby(['Turnos'])['PACIENTE_#_DOCUMENTO'].
 ↪count().reset_index()
#pacientes_por_grupo = df_hearth.groupby(['DIA_SEMANA', 'Turnos',␣
 ↪'EDAD_RANGO'])['PACIENTE_#_DOCUMENTO'].count().reset_index()

# Renombrar la columna de conteo
pacientes_por_grupo1 = pacientes_por_grupo1.
 ↪rename(columns={'PACIENTE_#_DOCUMENTO': 'Cantidad_Pacientes'})

plt.figure(figsize=(10, 6))
ax = sns.barplot(data=pacientes_por_grupo1, x='Turnos', y='Cantidad_Pacientes',␣
 ↪palette='viridis')
plt.xlabel('Turnos')
plt.ylabel('Cantidad de Pacientes')
plt.title('Cantidad de Pacientes por Turno')
plt.xticks(rotation=45)

# Agregar etiquetas en las barras
```

```
for index, row in pacientes_por_grupo1.iterrows():
    ax.annotate(str(row['Cantidad_Pacientes']), (index,␣
 ↪row['Cantidad_Pacientes']), ha='center', va='bottom')
plt.show()

tiempo_real = sum(df_hearth['Tiempo_Minutos_Total'])
pacientes_total = df_hearth['PACIENTE_#_DOCUMENTO'].count()
tiempo_promedio_espera_mes = tiempo_real / pacientes_total if pacientes_total >␣
 ↪0 else 0
print("Datos Reales (historicos):")
print(f"Tiempo Real: {tiempo_real} pacientes por mes")
print(f"Total pacientes: {pacientes_total} pacientes por mes")
print(f"Tiempo de Espera Promedio: {tiempo_promedio_espera_mes} ")
```

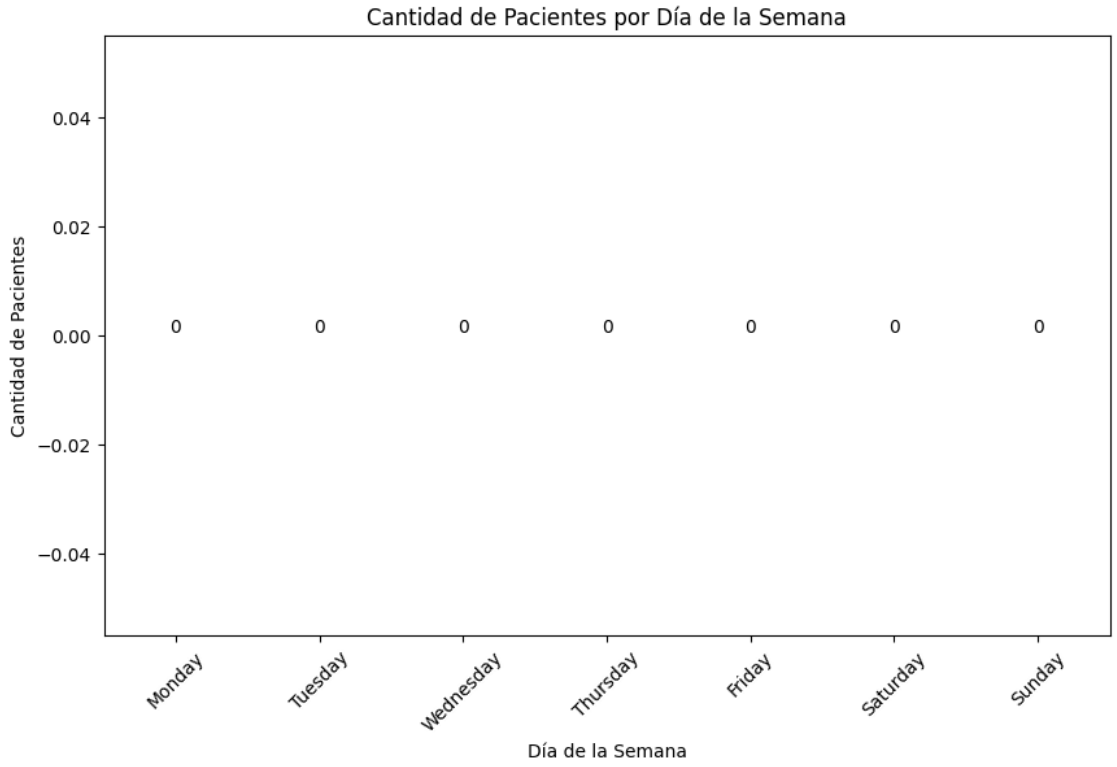



```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
d:\Victor.Gomez\Downloads\modelo de teoria de colas 3.ipynb Cell 5 line 2

    <a href='vscode-notebook-cell:/d%3A/Victor.Gomez/Downloads/
 ↪modelo%20de%20teoria%20de%20colas%203.ipynb#W4sZmlsZQ%3D%3D?line=23'>24</a>
 ↪ ax.annotate(str(row['Cantidad_Pacientes']), (index,␣
 ↪row['Cantidad_Pacientes']), ha='center', va='bottom')
```

```
    <a href='vscode-notebook-cell:/d%3A/Victor.Gomez/Downloads/
 ↪modelo%20de%20teoria%20de%20colas%203.ipynb#W4sZmlsZQ%3D%3D?line=24'>25</a>␣
 ↪plt.show()
---> <a href='vscode-notebook-cell:/d%3A/Victor.Gomez/Downloads/
 ↪modelo%20de%20teoria%20de%20colas%203.ipynb#W4sZmlsZQ%3D%3D?line=26'>27</a>␣
 ↪tiempo_real = sum(df_hearth['Tiempo_Minutos_Total'])
    <a href='vscode-notebook-cell:/d%3A/Victor.Gomez/Downloads/
 ↪modelo%20de%20teoria%20de%20colas%203.ipynb#W4sZmlsZQ%3D%3D?line=27'>28</a>␣
 ↪pacientes_total = df_hearth['PACIENTE_#_DOCUMENTO'].count()
    <a href='vscode-notebook-cell:/d%3A/Victor.Gomez/Downloads/
 ↪modelo%20de%20teoria%20de%20colas%203.ipynb#W4sZmlsZQ%3D%3D?line=28'>29</a>␣
 ↪tiempo_promedio_espera_mes = tiempo_real / pacientes_total if pacientes_total
 ↪> 0 else 0

TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

[8]:
```
# Grafica de Dia de la Semana
# Definir el orden deseado de los días de la semana
orden_dias_semana = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday',␣
 ↪'Saturday', 'Sunday']

# Reordenar la columna DIA_SEMANA para que coincida con el orden deseado
df_hearth.loc[:, 'DIA_SEMANA'] = pd.Categorical(df_hearth['DIA_SEMANA'],␣
 ↪categories=orden_dias_semana, ordered=True)

# Agrupar por día de la semana y contar la cantidad de pacientes en cada grupo
pacientes_por_grupo = df_hearth.groupby(['DIA_SEMANA'])['PACIENTE_#_DOCUMENTO'].
 ↪count().reset_index()

# Renombrar la columna de conteo
pacientes_por_grupo = pacientes_por_grupo.
 ↪rename(columns={'PACIENTE_#_DOCUMENTO': 'Cantidad_Pacientes'})

plt.figure(figsize=(10, 6))
ax = sns.barplot(data=pacientes_por_grupo, x='DIA_SEMANA',␣
 ↪y='Cantidad_Pacientes', palette='viridis')
plt.xlabel('Día de la Semana')
plt.ylabel('Cantidad de Pacientes')
plt.title('Cantidad de Pacientes por Día de la Semana')
plt.xticks(rotation=45)

# Agregar etiquetas en las barras
for index, row in pacientes_por_grupo.iterrows():
    ax.annotate(str(row['Cantidad_Pacientes']), (index,␣
 ↪row['Cantidad_Pacientes']), ha='center', va='bottom')
plt.show()

tiempo_real = sum(df_hearth['Tiempo_Minutos_Total'])
```

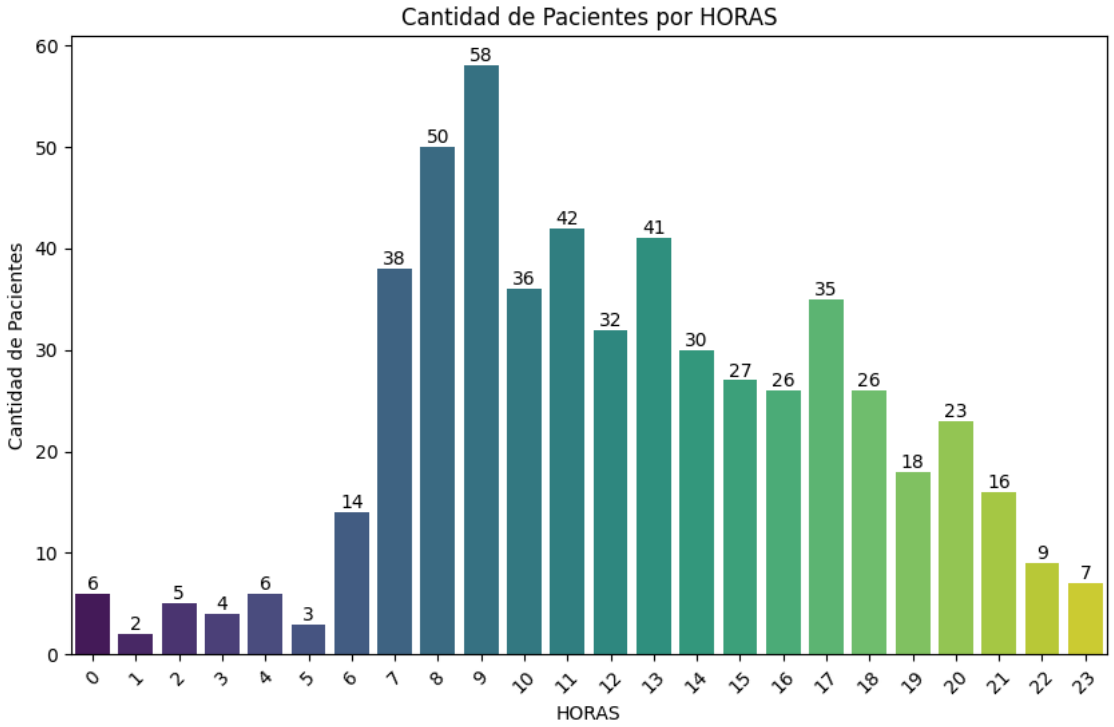
```
pacientes_total = df_hearth['PACIENTE_#_DOCUMENTO'].count()
tiempo_promedio_espera_mes = tiempo_real / pacientes_total if pacientes_total >␣
 ↪0 else 0
print("Datos Reales (historicos):")
print(f"Tiempo Real: {tiempo_real} pacientes por mes")
print(f"Total pacientes: {pacientes_total} pacientes por mes")
print(f"Tiempo de Espera Promedio: {tiempo_promedio_espera_mes} ")
```




```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
d:\Victor.Gomez\Downloads\modelo de teoria de colas 3.ipynb Cell 6 line 2

    <a href='vscode-notebook-cell:/d%3A/Victor.Gomez/Downloads/
 ↪modelo%20de%20teoria%20de%20colas%203.ipynb#W5sZmlsZQ%3D%3D?line=22'>23</a>
 ↪ ax.annotate(str(row['Cantidad_Pacientes']), (index,␣
 ↪row['Cantidad_Pacientes']), ha='center', va='bottom')
    <a href='vscode-notebook-cell:/d%3A/Victor.Gomez/Downloads/
 ↪modelo%20de%20teoria%20de%20colas%203.ipynb#W5sZmlsZQ%3D%3D?line=23'>24</a>␣
 ↪plt.show()
---> <a href='vscode-notebook-cell:/d%3A/Victor.Gomez/Downloads/
 ↪modelo%20de%20teoria%20de%20colas%203.ipynb#W5sZmlsZQ%3D%3D?line=25'>26</a>␣
 ↪tiempo_real = sum(df_hearth['Tiempo_Minutos_Total'])
    <a href='vscode-notebook-cell:/d%3A/Victor.Gomez/Downloads/
 ↪modelo%20de%20teoria%20de%20colas%203.ipynb#W5sZmlsZQ%3D%3D?line=26'>27</a>␣
 ↪pacientes_total = df_hearth['PACIENTE_#_DOCUMENTO'].count()
```

```
    <a href='vscode-notebook-cell:/d%3A/Victor.Gomez/Downloads/
 ↪modelo%20de%20teoria%20de%20colas%203.ipynb#W5sZmlsZQ%3D%3D?line=27'>28</a>␣
 ↪tiempo_promedio_espera_mes = tiempo_real / pacientes_total if pacientes_total
 ↪> 0 else 0

TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

[10]:
```
# Agrupar por turno y contar la cantidad de pacientes en cada grupo
pacientes_por_grupo1 = df_hearth.groupby(['HOUR'])['PACIENTE_#_DOCUMENTO'].
 ↪count().reset_index()

# Renombrar la columna de conteo
pacientes_por_grupo1 = pacientes_por_grupo1.
 ↪rename(columns={'PACIENTE_#_DOCUMENTO': 'Cantidad_Pacientes'})

plt.figure(figsize=(10, 6))
ax = sns.barplot(data=pacientes_por_grupo1, x='HOUR', y='Cantidad_Pacientes',␣
 ↪palette='viridis')
plt.xlabel('HORAS')
plt.ylabel('Cantidad de Pacientes')
plt.title('Cantidad de Pacientes por HORAS')
plt.xticks(rotation=45)

# Agregar etiquetas en las barras
for index, row in pacientes_por_grupo1.iterrows():
    ax.annotate(str(row['Cantidad_Pacientes']), (index,␣
 ↪row['Cantidad_Pacientes']), ha='center', va='bottom')

plt.show()
```



[104]:
```
tiempo_real = sum(df_hearth['Tiempo_Minutos_Total'])
pacientes_total = df_hearth['PACIENTE_#_DOCUMENTO'].count()
tiempo_promedio_espera_mes = tiempo_real / pacientes_total if pacientes_total >␣
↪0 else 0
print("Datos Reales (historicos):")
print(f"Tiempo Real: {tiempo_real} pacientes por mes")
print(f"Total pacientes: {pacientes_total} pacientes por mes")
print(f"Tiempo de Espera Promedio: {tiempo_promedio_espera_mes} ")
```

```
Datos Reales (historicos):
Tiempo Real: 20628.0 pacientes por mes
Total pacientes: 640 pacientes por mes
Tiempo de Espera Promedio: 32.23125
```

[26]:
```
# prueba 4

# Tasas de llegada y servicio iniciales
tasa_llegada_inicial = 5  # Tasa de llegada promedio de pacientes por hora ( )
tasa_servicio_inicial = 3  # Tasa de servicio promedio de pacientes por hora ( )
tasa_llegada = 5  # Tasa de llegada promedio de pacientes por hora ( )
tasa_servicio = 3  # Tasa de servicio promedio de pacientes por hora ( )
num_medicos = 4  # Puedes ajustar esto según tus necesidades
tiempo_espera_total = 0
```

```
clientes_atendidos = 0
tiempo_simulacion = 24  # Tiempo de simulación en horas

# Parámetros para la proyección
num_simulaciones = 2  # Número de simulaciones a realizar
tasas_llegada_proyectadas = [7, 9, 11]  # Tasas de llegada a probar
tasas_servicio_proyectadas = [5, 7, 9]  # Tasas de servicio a probar

# Listas para almacenar resultados de las simulaciones
resultados_tiempo_espera_promedio = []

# Diccionarios para almacenar métricas por turno y día de la semana
turnos = ['MADRUGADA', 'MAÑANA', 'TARDE', 'NOCHE']
meses = ['01', '02', '03', '04', '05', '06', '07','08','09','10','11','12']
dias_semana = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday',␣
 ↪'Saturday', 'Sunday']
metricas_por_turno = {turno: {'espera': 0, 'atendidos': 0} for turno in turnos}
metricas_por_dia_semana  = {dias: {'espera': 0, 'atendidos': 0} for dias in␣
 ↪dias_semana}
metricas_por_mes  = {mes: {'espera': 0, 'atendidos': 0} for mes in meses}

def llegada_paciente(env, medico, turno, dia_semana, mes, centro,␣
 ↪clasificacion):
    global tiempo_espera_total, clientes_atendidos, tiempo_real_total
    llegada = np.random.exponential(1 / tasa_llegada)
    yield env.timeout(llegada)
    # print(f"Llegó un paciente a las {env.now:.2f} horas.")
    with medico.request() as req:
        yield req
        tiempo_espera = env.now - llegada
        tiempo_espera_total += tiempo_espera
             # print(f"Paciente atendido a las {env.now:.2f} horas después de␣
 ↪esperar {tiempo_espera:.2f} horas.")
        servicio = np.random.exponential(1 / tasa_servicio)
        yield env.timeout(servicio)
        clientes_atendidos += 1

        # Actualizar métricas por turno y día de la semana
        metricas_por_turno[turno]['espera'] += tiempo_espera
        metricas_por_turno[turno]['atendidos'] += 1
        metricas_por_dia_semana[dia_semana]['espera'] += tiempo_espera
        metricas_por_dia_semana[dia_semana]['atendidos'] += 1

# Extender el tiempo de simulación a 6 meses (aproximadamente 4320 horas)
tiempo_simulacion_extendido = 4320

# Listas para almacenar resultados de las simulaciones extendidas
```

```
resultados_tiempo_espera_promedio_extendido = []

# Realizar simulación extendida
for tasa_llegada_proyectada in tasas_llegada_proyectadas:
    for tasa_servicio_proyectada in tasas_servicio_proyectadas:
        tiempo_espera_promedio_simulaciones = []

        for _ in range(num_simulaciones):
            # Configurar la simulación
            env = simpy.Environment()
            medico = simpy.Resource(env, capacity=num_medicos)
            tasa_llegada = tasa_llegada_proyectada
            tasa_servicio = tasa_servicio_proyectada
            tiempo_espera_total = 0
            clientes_atendidos = 0

            # Llamar a llegada_paciente para cada fila del DataFrame
            for index, row in df_hearth.iterrows():
                turno = row['Turnos']
                dia_semana = row['DIA_SEMANA']
                mes = row['MES']
                centro = row['CENTRO_ATENCION']
                clasificacion = row['CLASIFICACION_TRIAGE']
                env.process(llegada_paciente(env, medico, turno, dia_semana,␣
 ↪mes, centro, clasificacion))

            # Configurar tiempo de simulación basado en tus datos
            tiempo_simulacion = df_hearth['Tiempo_Minutos_Total'].max() + 60
            tiempo_real = df_hearth['Tiempo_Minutos_Total']

            # Configurar tiempo de simulación extendido
#             env.run(until=tiempo_simulacion_extendido)
            env.run(until=tiempo_simulacion)

            # Calcular métricas
            tiempo_promedio_espera = tiempo_espera_total / clientes_atendidos␣
 ↪if clientes_atendidos > 0 else 0
            tiempo_espera_promedio_simulaciones.append(tiempo_promedio_espera)

        # Calcular el promedio de tiempo de espera de todas las simulaciones␣
 ↪extendidas
        tiempo_espera_promedio = sum(tiempo_espera_promedio_simulaciones) /␣
 ↪num_simulaciones
        tiempo_espera_promedio_extendido =␣
 ↪sum(tiempo_espera_promedio_simulaciones) / num_simulaciones
```

```
        resultados_tiempo_espera_promedio_extendido.
 ↪append((tasa_llegada_proyectada, tasa_servicio_proyectada,␣
 ↪tiempo_espera_promedio_extendido))
        resultados_tiempo_espera_promedio.append((tasa_llegada, tasa_servicio,␣
 ↪tiempo_promedio_espera))

# Análisis de los resultados
mejor_proyeccion = min(resultados_tiempo_espera_promedio_extendido, key=lambda␣
 ↪x: x[2])
print("Mejor proyección:")
print(f"Tasa de Llegada Proyectada: {mejor_proyeccion[0]} pacientes por hora")
print(f"Tasa de Servicio Proyectada: {mejor_proyeccion[1]} pacientes por hora")
print(f"Tiempo de Espera Promedio: {mejor_proyeccion[2]:.2f} horas")

# Análisis de los resultados real
mejor_proyeccion2 = min(resultados_tiempo_espera_promedio, key=lambda x: x[2])
print("Mejor proyección 2:")
print(f"Tasa de Llegada : {mejor_proyeccion2[0]} pacientes por hora")
print(f"Tasa de Servicio: {mejor_proyeccion2[1]} pacientes por hora")
print(f"Tiempo de Espera Promedio: {mejor_proyeccion2[2]:.2f} horas")

# Puedes tomar decisiones basadas en estos resultados, como ajustar las tasas␣
 ↪de llegada y servicio para reducir los tiempos de espera.
```

```
Mejor proyección:
Tasa de Llegada Proyectada: 11 pacientes por hora
Tasa de Servicio Proyectada: 9 pacientes por hora
Tiempo de Espera Promedio: 23.00 horas
Mejor proyección 2:
Tasa de Llegada : 11 pacientes por hora
Tasa de Servicio: 9 pacientes por hora
Tiempo de Espera Promedio: 22.51 horas
```

modelo-de-bosque-aleatorio-18oct

November 23, 2023

[ ]:
```
# Cargue de librerías
import pandas as pd
from prophet import Prophet
import os
from sklearn.ensemble import RandomForestRegressor
import matplotlib.pyplot as plt
import hashlib
```

[ ]:
```
# ruta de archivos
files = os.listdir("c:\\archivos\proyecto")
os.chdir(r'C:\archivos\proyecto')

df_total = pd.read_csv('ind_urgencias_final_2023_filtrado.txt', sep=';')
```

[ ]:
```
df_total.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 82416 entries, 0 to 82415
Data columns (total 25 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   FECHA_LLEGADA              82416 non-null  object
 1   FECHA_TRIAGE               82416 non-null  object
 2   FECHA_INGRESO              82416 non-null  object
 3   FECHA_ATENCION             82416 non-null  object
 4   TIEMPO_DGTURNO_A_TRIAGE    82416 non-null  object
 5   TIEMPO_TRIAGE_A_INGRESO    82416 non-null  object
 6   TIEMPO_INGRESO_A_CONSULTA  82416 non-null  object
 7   TIEMPO_TOTAL               82416 non-null  object
 8   Tiempo_Minutos_Total       82416 non-null  object
 9   CENTRO_ATENCION            82416 non-null  object
 10  CLASIFICACION_TRIAGE       82416 non-null  int64
 11  PACIENTE_#_DOCUMENTO       82416 non-null  object
 12  EDAD                       82416 non-null  int64
 13  EDAD_RANGO                 82416 non-null  object
 14  SEXO                       82416 non-null  object
 15  RÉGIMEN PACIENTE           82416 non-null  object
 16  NOMBRE_ENTIDAD             82416 non-null  object
```

```
 17  MEDICO                      82416 non-null  int64
 18  AÑO                         82416 non-null  int64
 19  MES                         82416 non-null  int64
 20  DIA_SEMANA                  82416 non-null  object
 21  HOUR                        82416 non-null  int64
 22  Turnos                      82416 non-null  object
 23  TIME                        82416 non-null  object
 24  DIA                         82416 non-null  int64
dtypes: int64(7), object(18)
memory usage: 15.7+ MB
```

[ ]:
```
# convertir a variables categoricas
#df_total['Turnos'] = df_total['Turnos'].astype('category')
#df_total['DIA_SEMANA'] = df_total['DIA_SEMANA'].astype('category')
df_total['CENTRO_ATENCION'] = df_total['CENTRO_ATENCION'].astype('category')
```

[ ]:
```
# Seleccionar las columnas de interés
#df = df_hearth[['DIA_SEMANA', 'Tiempo_Minutos_Total']]

data =␣
 ↪df_total[['Tiempo_Minutos_Total','CENTRO_ATENCION','FECHA_LLEGADA','PACIENTE_#_DOCUMENTO','
```

[ ]:
```
# reemplazar datos para convertirlos enteros
data['Tiempo_Minutos_Total'] = data['Tiempo_Minutos_Total'].str.replace(',', '.
 ↪', regex=True)
data["Tiempo_Minutos_Total"] = pd.
 ↪to_numeric(data["Tiempo_Minutos_Total"],errors='coerce')
data['Turnos'] = data['Turnos'].str.replace('MADRUGADA', '4', regex=True)
data['Turnos'] = data['Turnos'].str.replace('MAÑANA', '1', regex=True)
data['Turnos'] = data['Turnos'].str.replace('TARDE', '2', regex=True)
data['Turnos'] = data['Turnos'].str.replace('NOCHE', '3', regex=True)
```

```
C:\Users\Victor.Gomez\AppData\Local\Temp\ipykernel_8704\2942042221.py:2:
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_guide/indexing.html#returning-a-view-versus-a-copy
  data['Tiempo_Minutos_Total'] = data['Tiempo_Minutos_Total'].str.replace(',',
'.', regex=True)
C:\Users\Victor.Gomez\AppData\Local\Temp\ipykernel_8704\2942042221.py:3:
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_guide/indexing.html#returning-a-view-versus-a-copy
```

```
  data["Tiempo_Minutos_Total"] =
pd.to_numeric(data["Tiempo_Minutos_Total"],errors='coerce')
C:\Users\Victor.Gomez\AppData\Local\Temp\ipykernel_8704\2942042221.py:4:
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_guide/indexing.html#returning-a-view-versus-a-copy
  data['Turnos'] = data['Turnos'].str.replace('MADRUGADA', '4', regex=True)
C:\Users\Victor.Gomez\AppData\Local\Temp\ipykernel_8704\2942042221.py:5:
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_guide/indexing.html#returning-a-view-versus-a-copy
  data['Turnos'] = data['Turnos'].str.replace('MAÑANA', '1', regex=True)
C:\Users\Victor.Gomez\AppData\Local\Temp\ipykernel_8704\2942042221.py:6:
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_guide/indexing.html#returning-a-view-versus-a-copy
  data['Turnos'] = data['Turnos'].str.replace('TARDE', '2', regex=True)
C:\Users\Victor.Gomez\AppData\Local\Temp\ipykernel_8704\2942042221.py:7:
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_guide/indexing.html#returning-a-view-versus-a-copy
  data['Turnos'] = data['Turnos'].str.replace('NOCHE', '3', regex=True)
```

[ ]:
```
# Para ver todas las filas que tienen valores faltantes -- revisar y eliminar␣
↪filas
is_NaN = data.isnull()
row_has_NaN = is_NaN.any(axis=1)
rows_with_NaN = data[row_has_NaN]
rows_with_NaN.head(100)
```

[ ]:
```
Empty DataFrame
Columns: [Tiempo_Minutos_Total, CENTRO_ATENCION, FECHA_LLEGADA,
PACIENTE_#_DOCUMENTO, Turnos, MEDICO]
Index: []
```

[ ]:
```
data
```

[ ]:
```
       Tiempo_Minutos_Total CENTRO_ATENCION            FECHA_LLEGADA  \
0                     39.73              TN  2023-01-01 01:20:23.853
1                     33.12              ME  2023-01-01 01:29:46.050
2                     14.77              UC  2023-01-01 03:15:35.623
3                     31.40              UC  2023-01-01 05:54:53.563
4                    173.80              TN  2023-01-01 06:37:27.237
…                       …               …                       …
82411                 84.90              ME  2023-09-18 04:44:41.970
82412                 83.75              ME  2023-09-18 06:17:00.573
82413                125.83              ME  2023-09-18 06:21:37.273
82414                 62.92              UB  2023-09-18 06:25:33.483
82415                 34.33              TN  2023-09-18 07:14:58.180

      PACIENTE_#_DOCUMENTO Turnos  MEDICO
0               1007228378      3   11065
1               1000003681      3    8861
2               1007454009      3    5855
3               1022997183      3   11072
4               1013671529      3    1239
…                      …    …       …
82411           1010242518      3    7844
82412             93357619      3    6204
82413             41372387      3    9951
82414           1000807249      3    4030
82415              4497020      1    1239

[82416 rows x 6 columns]
```

[ ]:
```
# Anonimizar datos sensibles
# Crear un diccionario para asignar valores únicos
unique_dict = {}
counter = 1  # Contador inicial

# Función para asignar valores únicos
def assign_unique_value(value):
    global counter  # Usa 'global' en lugar de 'nonlocal'
    if value in unique_dict:
        return unique_dict[value]
    unique_dict[value] = counter
    counter += 1
    return unique_dict[value]

# Reemplazar PACIENTE_DOCUMENTO por valores únicos
data['PACIENTE_#_DOCUMENTO'] = data['PACIENTE_#_DOCUMENTO'].
 ↪apply(assign_unique_value)
```

```
# Crear un diccionario para asignar valores únicos
unique_dict = {}
counter = 1  # Contador inicial

# Función para asignar valores únicos
def assign_unique_value(value):
    global counter  # Usa 'global' en lugar de 'nonlocal'
    if value in unique_dict:
        return unique_dict[value]
    unique_dict[value] = counter
    counter += 1
    return unique_dict[value]

# Reemplazar MEDICO por valores únicos
data['MEDICO'] = data['MEDICO'].apply(assign_unique_value)
```

```
C:\Users\Victor.Gomez\AppData\Local\Temp\ipykernel_8704\3498531685.py:16:
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_guide/indexing.html#returning-a-view-versus-a-copy
  data['PACIENTE_#_DOCUMENTO'] =
data['PACIENTE_#_DOCUMENTO'].apply(assign_unique_value)
C:\Users\Victor.Gomez\AppData\Local\Temp\ipykernel_8704\3498531685.py:32:
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_guide/indexing.html#returning-a-view-versus-a-copy
  data['MEDICO'] = data['MEDICO'].apply(assign_unique_value)
```

[ ]:
```
data
```

[ ]:

	Tiempo_Minutos_Total	CENTRO_ATENCION	FECHA_LLEGADA
0	39.73	TN	2023-01-01 01:20:23.853
1	33.12	ME	2023-01-01 01:29:46.050
2	14.77	UC	2023-01-01 03:15:35.623
3	31.40	UC	2023-01-01 05:54:53.563
4	173.80	TN	2023-01-01 06:37:27.237
…	…	…	…
82411	84.90	ME	2023-09-18 04:44:41.970
82412	83.75	ME	2023-09-18 06:17:00.573
82413	125.83	ME	2023-09-18 06:21:37.273

```
82414                  62.92              UB  2023-09-18 06:25:33.483
82415                  34.33              TN  2023-09-18 07:14:58.180

       PACIENTE_#_DOCUMENTO Turnos  MEDICO
0                         1      3       1
1                         2      3       2
2                         3      3       3
3                         4      3       4
4                         5      3       5
…                       …    …       …
82411                 62211      3      49
82412                 63445      3      11
82413                 63446      3      43
82414                 63447      3      21
82415                 63448      1       5

[82416 rows x 6 columns]
```

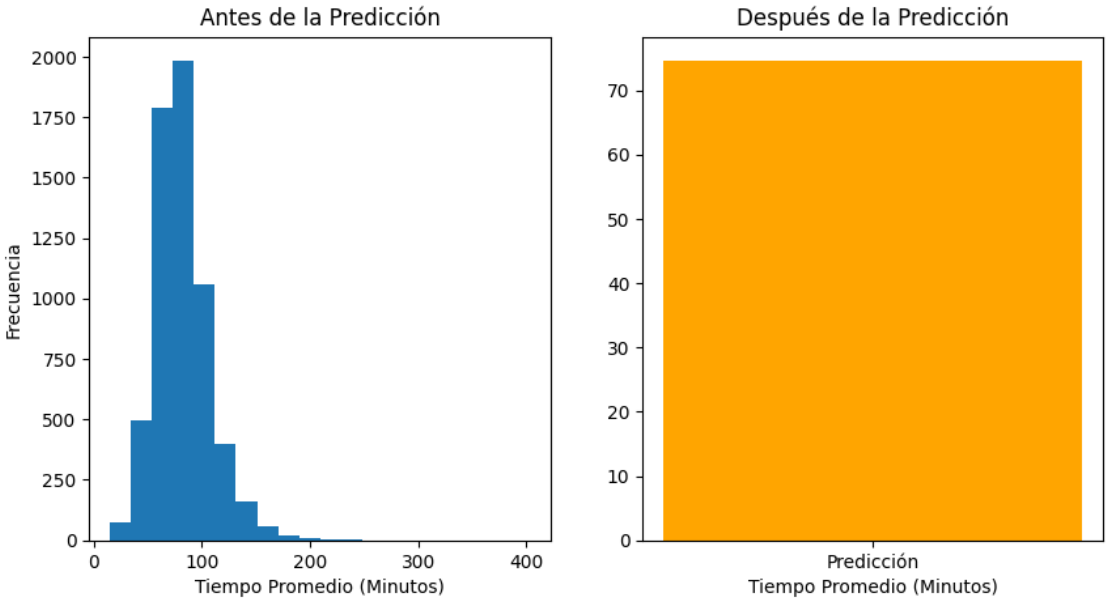
[ ]:
```python
data.to_csv('urgencias.txt', sep=';', index=False)
```

[ ]:
```python
##  ok version 1

# Preprocesar los datos
data['FECHA_LLEGADA'] = pd.to_datetime(data['FECHA_LLEGADA'])
#data['Mes'] = data['FECHA_LLEGADA'].dt.month
#data['Día'] = data['FECHA_LLEGADA'].dt.day
#data['Día de la Semana'] = data['FECHA_LLEGADA'].dt.dayofweek
#data['HOUR'] = data['FECHA_LLEGADA'].dt.hour


# Preprocesar los datos

#data.loc[:, 'FECHA_LLEGADA'] = pd.to_datetime(data['FECHA_LLEGADA'])
data.loc[:, 'Mes'] = data['FECHA_LLEGADA'].dt.month
data.loc[:, 'Día'] = data['FECHA_LLEGADA'].dt.day
data.loc[:, 'DIA_SEMANA'] = data['FECHA_LLEGADA'].dt.dayofweek
data.loc[:, 'HOUR'] = data['FECHA_LLEGADA'].dt.hour

# quitar registros malos

# Calcular la mediana
median = data['Tiempo_Minutos_Total'].median()

# Corregir valores atípicos
data.loc[data['Tiempo_Minutos_Total'] > 420, 'Tiempo_Minutos_Total'] = median
data.loc[data['Tiempo_Minutos_Total'] < 0, 'Tiempo_Minutos_Total'] = median
```

```
# Calcular el tiempo promedio por grupo
tiempo_promedio = data.groupby(['Mes', 'Día', 'DIA_SEMANA',␣
 ↪'HOUR'])['Tiempo_Minutos_Total'].median().reset_index()

# Crear un modelo de regresión (Random Forest)
X = tiempo_promedio[['Mes', 'Día', 'DIA_SEMANA', 'HOUR']]
y = tiempo_promedio['Tiempo_Minutos_Total']
model = RandomForestRegressor(n_estimators=100, random_state=42)
model.fit(X, y)

# Realizar predicciones para diferentes valores
nuevos_datos = pd.DataFrame({
    'Mes': [1,2,3,4,5,6,7],
    'Día': [10,11,12,13,14,15,16],
    'DIA_SEMANA': [1,2,3,4,5,6,7],
    'HOUR': [7,8,9,10,11,12,13]
})

tiempo_espera_predicho = model.predict(nuevos_datos)
print(f"Tiempo de espera predicho: {tiempo_espera_predicho[0]} minutos")

# Visualización (antes y después)
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.title("Antes de la Predicción")
plt.hist(tiempo_promedio['Tiempo_Minutos_Total'], bins=20)
plt.xlabel("Tiempo Promedio (Minutos)")
plt.ylabel("Frecuencia")
plt.subplot(1, 2, 2)
plt.title("Después de la Predicción")
plt.bar(["Predicción"], [tiempo_espera_predicho[0]], color='orange')
plt.xlabel("Tiempo Promedio (Minutos)")
plt.show()

print(f"Tiempo de espera predicho: {tiempo_espera_predicho[0]} minutos")
```

```
C:\Users\Victor.Gomez\AppData\Local\Temp\ipykernel_8704\4116202701.py:4:
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_guide/indexing.html#returning-a-view-versus-a-copy
  data['FECHA_LLEGADA'] = pd.to_datetime(data['FECHA_LLEGADA'])
C:\Users\Victor.Gomez\AppData\Local\Temp\ipykernel_8704\4116202701.py:14:
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_guide/indexing.html#returning-a-view-versus-a-copy
  data.loc[:, 'Mes'] = data['FECHA_LLEGADA'].dt.month
C:\Users\Victor.Gomez\AppData\Local\Temp\ipykernel_8704\4116202701.py:15:
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_guide/indexing.html#returning-a-view-versus-a-copy
  data.loc[:, 'Día'] = data['FECHA_LLEGADA'].dt.day
C:\Users\Victor.Gomez\AppData\Local\Temp\ipykernel_8704\4116202701.py:16:
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_guide/indexing.html#returning-a-view-versus-a-copy
  data.loc[:, 'DIA_SEMANA'] = data['FECHA_LLEGADA'].dt.dayofweek
C:\Users\Victor.Gomez\AppData\Local\Temp\ipykernel_8704\4116202701.py:17:
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_guide/indexing.html#returning-a-view-versus-a-copy
  data.loc[:, 'HOUR'] = data['FECHA_LLEGADA'].dt.hour

Tiempo de espera predicho: 74.59235000000004 minutos
```




```
Tiempo de espera predicho: 74.59235000000004 minutos
```

[ ]:
```
data
```

[ ]:

	Tiempo_Minutos_Total	CENTRO_ATENCION	FECHA_LLEGADA	\
0	39.73	TN	2023-01-01 01:20:23.853	
1	33.12	ME	2023-01-01 01:29:46.050	
2	14.77	UC	2023-01-01 03:15:35.623	
3	31.40	UC	2023-01-01 05:54:53.563	
4	173.80	TN	2023-01-01 06:37:27.237	
…	…	…	…	
82411	84.90	ME	2023-09-18 04:44:41.970	
82412	83.75	ME	2023-09-18 06:17:00.573	
82413	125.83	ME	2023-09-18 06:21:37.273	
82414	62.92	UB	2023-09-18 06:25:33.483	
82415	34.33	TN	2023-09-18 07:14:58.180	

	PACIENTE_#_DOCUMENTO	Turnos	MEDICO	Mes	Día	DIA_SEMANA	HOUR
0	1	3	1	1	1	6	1
1	2	3	2	1	1	6	1
2	3	3	3	1	1	6	3
3	4	3	4	1	1	6	5
4	5	3	5	1	1	6	6
…	…	…	…	…	…	…	…
82411	62211	3	49	9	18	0	4
82412	63445	3	11	9	18	0	6
82413	63446	3	43	9	18	0	6

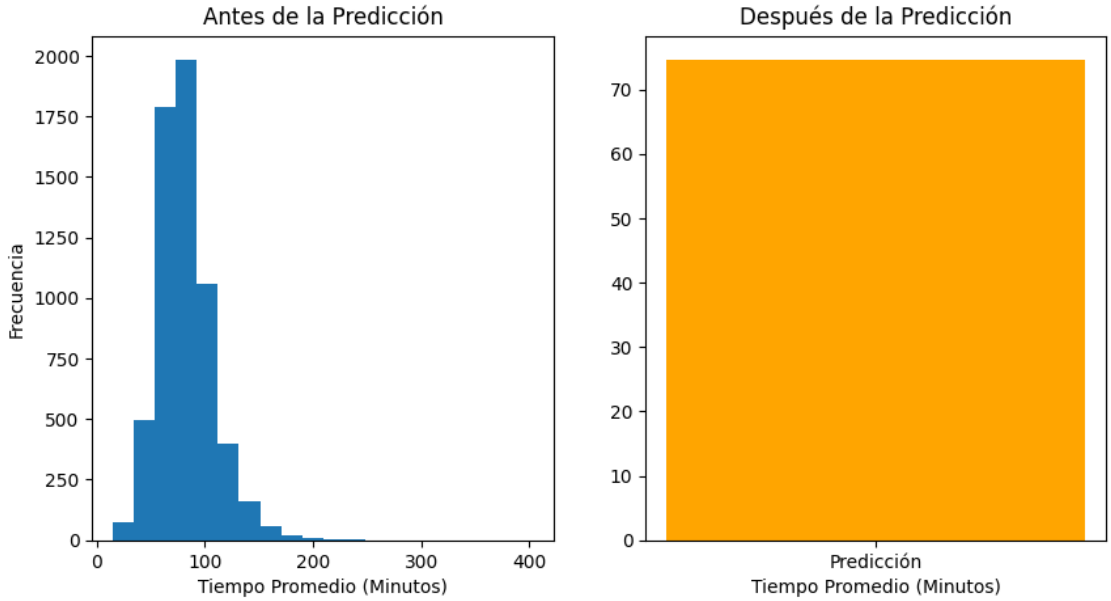
```
82414                63447      3       21    9   18           0     6
82415                63448      1        5    9   18           0     7

[82416 rows x 10 columns]
```

[ ]:
```
# version 2 - tiempo promedio
# Preprocesar los datos
# Preprocesar los datos
data['FECHA_LLEGADA'] = pd.to_datetime(data['FECHA_LLEGADA'])

#data.loc[:, 'FECHA_LLEGADA'] = pd.to_datetime(data['FECHA_LLEGADA'])
data.loc[:, 'Mes'] = data['FECHA_LLEGADA'].dt.month
data.loc[:, 'Día'] = data['FECHA_LLEGADA'].dt.day
data.loc[:, 'DIA_SEMANA'] = data['FECHA_LLEGADA'].dt.dayofweek
data.loc[:, 'HOUR'] = data['FECHA_LLEGADA'].dt.hour

# quitar registros malos

# Calcular la mediana
median = data['Tiempo_Minutos_Total'].median()

# Corregir valores atípicos
data.loc[data['Tiempo_Minutos_Total'] > 420, 'Tiempo_Minutos_Total'] = median
data.loc[data['Tiempo_Minutos_Total'] < 0, 'Tiempo_Minutos_Total'] = median

# Calcular el total medicos
medico_contador = data.groupby(['Mes', 'Día', 'DIA_SEMANA', 'HOUR'])['MEDICO'].
 ↪count().reset_index()

# Calcular el medico promedio  por grupo
medico_promedio = data.groupby(['Mes', 'Día', 'DIA_SEMANA', 'HOUR'])['MEDICO'].
 ↪nunique().reset_index()
medico_promedio['Promedio_Medicos'] = medico_contador['MEDICO'] /␣
 ↪medico_promedio['MEDICO']

# Crear un modelo de regresión (Random Forest)
X = tiempo_promedio[['Mes', 'Día', 'DIA_SEMANA', 'HOUR']]
y = tiempo_promedio['Tiempo_Minutos_Total']
model = RandomForestRegressor(n_estimators=100, random_state=42)
model.fit(X, y)

# Realizar predicciones para diferentes valores
nuevos_datos = pd.DataFrame({
    'Mes': [1,2,3,4,5,6,7],
    'Día': [10,11,12,13,14,15,16],
    'DIA_SEMANA': [1,2,3,4,5,6,7],
    'HOUR': [7,8,9,10,11,12,13]
```

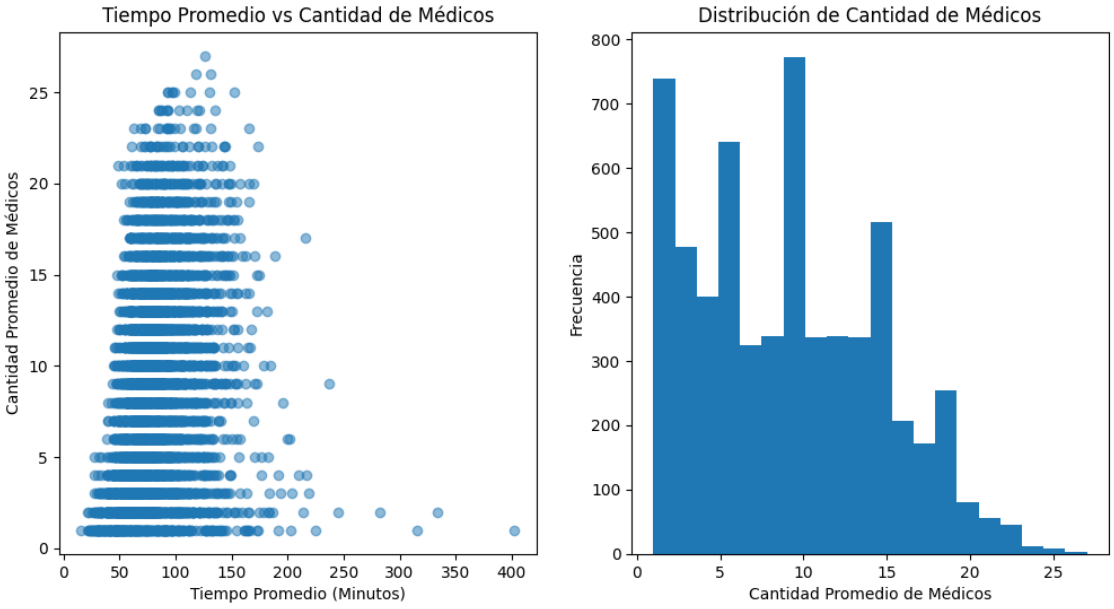
```
})

tiempo_espera_predicho = model.predict(nuevos_datos)
print(f"Tiempo de espera predicho: {tiempo_espera_predicho[0]} minutos")

# Visualización (antes y después)
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.title("Antes de la Predicción")
plt.hist(tiempo_promedio['Tiempo_Minutos_Total'], bins=20)
plt.xlabel("Tiempo Promedio (Minutos)")
plt.ylabel("Frecuencia")
plt.subplot(1, 2, 2)
plt.title("Después de la Predicción")
plt.bar(["Predicción"], [tiempo_espera_predicho[0]], color='orange')
plt.xlabel("Tiempo Promedio (Minutos)")
plt.show()

print(f"Tiempo de espera predicho: {tiempo_espera_predicho[0]} minutos")



# Comparar Tiempo_Minutos_Total y Cantidad_Medicos
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.title("Tiempo Promedio vs Cantidad de Médicos")
plt.scatter(tiempo_promedio['Tiempo_Minutos_Total'], medico_promedio['MEDICO'],␣
 ↪alpha=0.5)
plt.xlabel("Tiempo Promedio (Minutos)")
plt.ylabel("Cantidad Promedio de Médicos")

plt.subplot(1, 2, 2)
plt.title("Distribución de Cantidad de Médicos")
plt.hist(medico_promedio['MEDICO'], bins=20)
plt.xlabel("Cantidad Promedio de Médicos")
plt.ylabel("Frecuencia")

plt.show()
```

```
C:\Users\Victor.Gomez\AppData\Local\Temp\ipykernel_8704\3379508475.py:4:
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_guide/indexing.html#returning-a-view-versus-a-copy
  data['FECHA_LLEGADA'] = pd.to_datetime(data['FECHA_LLEGADA'])
```

```
Tiempo de espera predicho: 74.59235000000004 minutos
```

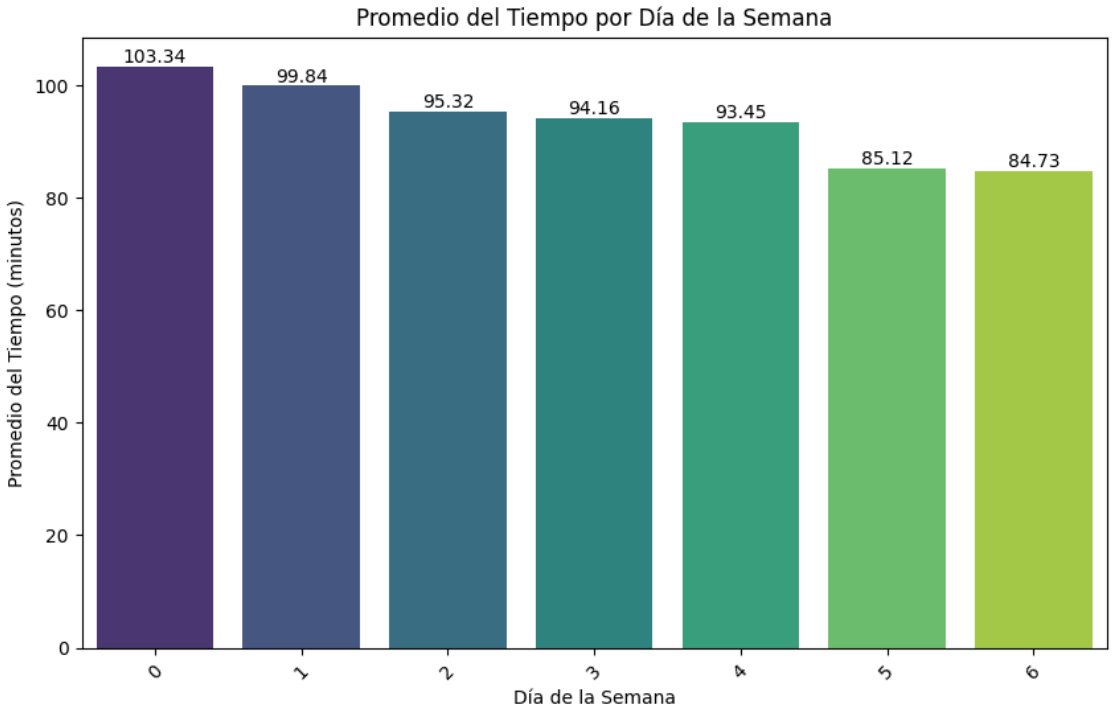



```
Tiempo de espera predicho: 74.59235000000004 minutos
```




```
[ ]: import seaborn as sns
```

```
# Agrupar por día de la semana y calcular el promedio del Tiempo_total_minutos␣
 ↪en cada grupo
promedio_tiempo_por_grupo =data.groupby(['DIA_SEMANA'])['Tiempo_Minutos_Total'].
 ↪mean().reset_index()

# Renombrar la columna del promedio
promedio_tiempo_por_grupo = promedio_tiempo_por_grupo.
 ↪rename(columns={'Tiempo_Minutos_Total': 'Promedio_Tiempo'})

plt.figure(figsize=(10, 6))
ax = sns.barplot(data=promedio_tiempo_por_grupo, x='DIA_SEMANA',␣
 ↪y='Promedio_Tiempo', palette='viridis')
plt.xlabel('Día de la Semana')
plt.ylabel('Promedio del Tiempo (minutos)')
plt.title('Promedio del Tiempo por Día de la Semana')
plt.xticks(rotation=45)

# Agregar etiquetas en las barras
for index, row in promedio_tiempo_por_grupo.iterrows():
    ax.annotate(str(round(row['Promedio_Tiempo'], 2)), (index,␣
 ↪row['Promedio_Tiempo']), ha='center', va='bottom')
plt.show()
```



[ ]:
```
print(f"Tiempo de espera predicho: {tiempo_espera_predicho[0]} minutos")
print(f"Tiempo de espera predicho: {tiempo_espera_predicho[1]} minutos")
print(f"Tiempo de espera predicho: {tiempo_espera_predicho[2]} minutos")
print(f"Tiempo de espera predicho: {tiempo_espera_predicho[3]} minutos")
print(f"Tiempo de espera predicho: {tiempo_espera_predicho[4]} minutos")
```

```
Tiempo de espera predicho: 74.59235000000004 minutos
Tiempo de espera predicho: 79.17985000000002 minutos
Tiempo de espera predicho: 85.67735000000002 minutos
Tiempo de espera predicho: 82.24480000000005 minutos
Tiempo de espera predicho: 66.15245000000002 minutos
```

odelos-de-series-de-tiempo-8-novv3

November 23, 2023

[ ]:
```
# Importar libreria requerida
import pandas as pd
import numpy as np
import os
```

[ ]:
```
# Métrica de Evaluación
from sklearn.metrics import mean_squared_error
from statsmodels.tools.eval_measures import rmse
from sklearn import metrics
```

[ ]:
```
# No presentar advertencias
import warnings
warnings.filterwarnings("ignore")
```

[ ]:
```
# Visualización de datos
import plotly.express as px
import matplotlib.pyplot as plt
%matplotlib inline
plt.style.use('ggplot')
```

[ ]:
```
#!pip install adfuller
!pip install pmdarima
```

```
Collecting pmdarima
  Downloading pmdarima-2.0.4-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86
_64.manylinux_2_28_x86_64.whl (2.1 MB)
                           2.1/2.1 MB
20.8 MB/s eta 0:00:00
Requirement already satisfied: joblib>=0.11 in
/usr/local/lib/python3.10/dist-packages (from pmdarima) (1.3.2)
Requirement already satisfied: Cython!=0.29.18,!=0.29.31,>=0.29 in
/usr/local/lib/python3.10/dist-packages (from pmdarima) (3.0.5)
Requirement already satisfied: numpy>=1.21.2 in /usr/local/lib/python3.10/dist-
packages (from pmdarima) (1.23.5)
Requirement already satisfied: pandas>=0.19 in /usr/local/lib/python3.10/dist-
packages (from pmdarima) (1.5.3)
Requirement already satisfied: scikit-learn>=0.22 in
/usr/local/lib/python3.10/dist-packages (from pmdarima) (1.2.2)
```

```
Requirement already satisfied: scipy>=1.3.2 in /usr/local/lib/python3.10/dist-
packages (from pmdarima) (1.11.3)
Requirement already satisfied: statsmodels>=0.13.2 in
/usr/local/lib/python3.10/dist-packages (from pmdarima) (0.14.0)
Requirement already satisfied: urllib3 in /usr/local/lib/python3.10/dist-
packages (from pmdarima) (2.0.7)
Requirement already satisfied: setuptools!=50.0.0,>=38.6.0 in
/usr/local/lib/python3.10/dist-packages (from pmdarima) (67.7.2)
Requirement already satisfied: packaging>=17.1 in
/usr/local/lib/python3.10/dist-packages (from pmdarima) (23.2)
Requirement already satisfied: python-dateutil>=2.8.1 in
/usr/local/lib/python3.10/dist-packages (from pandas>=0.19->pmdarima) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-
packages (from pandas>=0.19->pmdarima) (2023.3.post1)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.22->pmdarima)
(3.2.0)
Requirement already satisfied: patsy>=0.5.2 in /usr/local/lib/python3.10/dist-
packages (from statsmodels>=0.13.2->pmdarima) (0.5.3)
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages
(from patsy>=0.5.2->statsmodels>=0.13.2->pmdarima) (1.16.0)
Installing collected packages: pmdarima
Successfully installed pmdarima-2.0.4
```

[ ]:
```
# ruta de archivos
files = os.listdir("c:\\archivos\proyecto")
os.chdir(r'C:\archivos\proyecto')

csv_path = 'ind_urgencias_final_2023_filtrado.txt'

# Read data from CSV file
df = pd.read_csv(csv_path,sep=";",header= None)
```

[ ]:
```
# ruta de archivos con colab google

#csv_path = 'ind_urgencias_final_2023_filtrado.txt'
csv_path = 'indicadores de urgencias.txt'

# Read data from CSV file
df = pd.read_csv(csv_path,sep=";",header= None)
```

[ ]:
```
df
```

[ ]:
```
                            0                         1   \
0       TIEMPO_LLEGADA_A_TRIAGE  TIEMPO_TRIAGE_A_INGRESO
1                       0:00:00                   0:16:13
2                       0:00:00                   0:08:06
```

```
3                       0:00:00                  0:33:41
4                       0:00:00                  0:17:52
…                         …                      …
121256                  0:00:00                  0:41:05
121257                  0:00:00                  0:31:57
121258                  0:00:00                  0:10:19
121259                  0:00:00                  0:26:17
121260                  0:00:00                  0:06:08

                            2            3                4   \
0       TIEMPO_INGRESO_A_FOLIO  TIEMPO_TOTAL     FECHA_LLEGADA
1                       0:08:47       0:25:00    2/08/2016 7:19
2                       0:01:27       0:09:33    2/08/2016 7:58
3                       0:04:12       0:37:53    3/08/2016 9:10
4                       0:02:22       0:20:14   3/08/2016 13:05
…                         …           …              …
121256                  0:18:40       0:59:45  19/05/2017 18:05
121257                  0:11:33       0:43:30  18/05/2017 13:18
121258                  0:06:26       0:16:45  20/05/2017 11:31
121259                  0:05:49       0:32:06  25/05/2017 14:20
121260                  0:06:01       0:12:09  25/05/2017 17:39

                     5                6                7   \
0           FECHA_TRIAGE     FECHA_INGRESO    FECHA_CONSULTA
1         2/08/2016 7:19    2/08/2016 7:36    2/08/2016 7:45
2         2/08/2016 7:58    2/08/2016 8:07    2/08/2016 8:09
3         3/08/2016 9:10    3/08/2016 9:44    3/08/2016 9:48
4        3/08/2016 13:05   3/08/2016 13:23   3/08/2016 13:26
…                  …              …              …
121256  19/05/2017 18:05  19/05/2017 18:47  19/05/2017 19:05
121257  18/05/2017 13:18  18/05/2017 13:50  18/05/2017 14:02
121258  20/05/2017 11:31  20/05/2017 11:42  20/05/2017 11:48
121259  25/05/2017 14:20  25/05/2017 14:47  25/05/2017 14:52
121260  25/05/2017 17:39  25/05/2017 17:46  25/05/2017 17:52

                                                    8   \
0                                        CENTRO_ATENCION
1       VA - CENTRO DE SALUD CANDELARIA  I C. BOLÍVAR
2       VA - CENTRO DE SALUD CANDELARIA  I C. BOLÍVAR
3       VA - CENTRO DE SALUD CANDELARIA  I C. BOLÍVAR
4                       VC - CENTRO DE SALUD JERUSALÉN
…                                                  …
121256                     VB - HOSPITAL VISTA HERMOSA
121257                           ME - HOSPITAL MEISSEN
121258                  JC - CENTRO DE SALUD EL CARMEN
121259                  JC - CENTRO DE SALUD EL CARMEN
121260                      VJ - CENTRO DE SALUD MANUE
```

```
                                                  9                    10  \
0                                CLASIFICACION_TRIAGE  PACIENTE_#_DOCUMENTO
1       3 - TRIAGE III - NO DEBE SUPERAR LAS 3 HORAS               4108268
2       3 - TRIAGE III - NO DEBE SUPERAR LAS 3 HORAS            1026581403
3       3 - TRIAGE III - NO DEBE SUPERAR LAS 3 HORAS              19334228
4       3 - TRIAGE III - NO DEBE SUPERAR LAS 3 HORAS              52238647
…                                                  …                    …
121256  3 - TRIAGE III - NO DEBE SUPERAR LAS 3 HORAS            1031160974
121257  3 - TRIAGE III - NO DEBE SUPERAR LAS 3 HORAS               5966047
121258  3 - TRIAGE III - NO DEBE SUPERAR LAS 3 HORAS            1018458868
121259      2 - TRIAGE II -  ANTES DE LOS 30 MINUTOS            1033768047
121260                                           NaN                   NaN

                 11               12         13               14  \
0       PACIENTE_EDAD     EDAD_RANGO       SEXO  RÉGIMEN PACIENTE
1           77 AÑO(S)          VEJEZ  MASCULINO        SUBSIDIADO
2           21 AÑO(S)        JUVENTUD   FEMENINO        SUBSIDIADO
3           65 AÑO(S)          VEJEZ  MASCULINO        SUBSIDIADO
4           40 AÑO(S)         ADULTEZ   FEMENINO         VINCULADO
…                 …                …         …                …
121256       3 AÑO(S)  PRIMERA INFANCIA   FEMENINO      CONTRIBUTIVO
121257      70 AÑO(S)          VEJEZ  MASCULINO        SUBSIDIADO
121258      24 AÑO(S)        JUVENTUD   FEMENINO        SUBSIDIADO
121259       4 AÑO(S)  PRIMERA INFANCIA   FEMENINO        SUBSIDIADO
121260            NaN               NaN        NaN               NaN

                                15                              16  \
0                  NOMBRE_ENTIDAD                NOM_TIPO_HISTORIA
1       CAPITAL SALUD EPS-S S.A.S  CONSULTA  INICIAL DE URGENCIAS
2       CAPITAL SALUD EPS-S S.A.S  CONSULTA  INICIAL DE URGENCIAS
3       CAPITAL SALUD EPS-S S.A.S  CONSULTA  INICIAL DE URGENCIAS
4       CAPITAL SALUD EPS-S S.A.S       CONSULTA MEDICINA GENERAL
…                             …                               …
121256        SALUD TOTAL SA EPS   CONSULTA  INICIAL DE URGENCIAS
121257  CAPITAL SALUD EPS-S S.A.S  CONSULTA  INICIAL DE URGENCIAS
121258  CAPITAL SALUD EPS-S S.A.S  CONSULTA  INICIAL DE URGENCIAS
121259  CAPITAL SALUD EPS-S S.A.S  CONSULTA  INICIAL DE URGENCIAS
121260                        NaN                             NaN

                17                                                  18
0       DIAGNOSTICO                                        NOMBRE DX
1              H813  OTROS VERTIGOS PERIFERICOS                     …
2              A060  DISENTERIA AMEBIANA AGUDA                      …
3              A09X  DIARREA Y GASTROENTERITIS DE PRESUNTO ORIGEN I…
4              J039  AMIGDALITIS AGUDA, NO ESPECIFICADA             …
…               …                                                  …
```

```
121256         N390  INFECCION DE VIAS URINARIAS, SITIO NO ESPECIFI…
121257         K297  GASTRITIS, NO ESPECIFICADA                    …
121258         Z359  SUPERVISION DE EMBARAZO DE ALTO RIESGO, SIN OT…
121259         S903  CONTUSION DE OTRAS PARTES Y DE LAS NO ESPECIFI…
121260          NaN                                                NaN

[121261 rows x 19 columns]
```

[ ]:
```
headers =␣
 ↪["TIEMPO_LLEGADA_A_TRIAGE","TIEMPO_TRIAGE_A_INGRESO","TIEMPO_INGRESO_A_FOLIO","TIEMPO_TOTAL
 ↪PACIENTE","NOMBRE_ENTIDAD","NOM_TIPO_HISTORIA","DIAGNOSTICO","NOMBRE DX"]
print("headers\n", headers)
df.columns = headers
df = df.drop(0)
```

```
headers
 ['TIEMPO_LLEGADA_A_TRIAGE', 'TIEMPO_TRIAGE_A_INGRESO',
'TIEMPO_INGRESO_A_FOLIO', 'TIEMPO_TOTAL', 'FECHA_LLEGADA', 'FECHA_TRIAGE',
'FECHA_INGRESO', 'FECHA_CONSULTA', 'CENTRO_ATENCION', 'CLASIFICACION_TRIAGE',
'PACIENTE_#_DOCUMENTO', 'PACIENTE_EDAD', 'EDAD_RANGO', 'SEXO', 'RÉGIMEN
PACIENTE', 'NOMBRE_ENTIDAD', 'NOM_TIPO_HISTORIA', 'DIAGNOSTICO', 'NOMBRE DX']
```

[ ]:
```
df = df.drop(0)
```

[ ]:
```
df
```

[ ]:
```
       TIEMPO_LLEGADA_A_TRIAGE TIEMPO_TRIAGE_A_INGRESO TIEMPO_INGRESO_A_FOLIO  \
1                      0:00:00                0:16:13                0:08:47
2                      0:00:00                0:08:06                0:01:27
3                      0:00:00                0:33:41                0:04:12
4                      0:00:00                0:17:52                0:02:22
5                      0:00:00                0:13:57                0:21:41
…                        …                      …                      …
121256                 0:00:00                0:41:05                0:18:40
121257                 0:00:00                0:31:57                0:11:33
121258                 0:00:00                0:10:19                0:06:26
121259                 0:00:00                0:26:17                0:05:49
121260                 0:00:00                0:06:08                0:06:01

       TIEMPO_TOTAL     FECHA_LLEGADA      FECHA_TRIAGE     FECHA_INGRESO  \
1           0:25:00    2/08/2016 7:19    2/08/2016 7:19    2/08/2016 7:36
2           0:09:33    2/08/2016 7:58    2/08/2016 7:58    2/08/2016 8:07
3           0:37:53    3/08/2016 9:10    3/08/2016 9:10    3/08/2016 9:44
4           0:20:14   3/08/2016 13:05   3/08/2016 13:05   3/08/2016 13:23
5           0:35:38    1/08/2016 8:35    1/08/2016 8:35    1/08/2016 8:49
…             …              …                 …                 …
121256      0:59:45  19/05/2017 18:05  19/05/2017 18:05  19/05/2017 18:47
```

```
121257      0:43:30  18/05/2017 13:18  18/05/2017 13:18  18/05/2017 13:50
121258      0:16:45  20/05/2017 11:31  20/05/2017 11:31  20/05/2017 11:42
121259      0:32:06  25/05/2017 14:20  25/05/2017 14:20  25/05/2017 14:47
121260      0:12:09  25/05/2017 17:39  25/05/2017 17:39  25/05/2017 17:46

          FECHA_CONSULTA                                CENTRO_ATENCION  \
1         2/08/2016 7:45  VA - CENTRO DE SALUD CANDELARIA  I C. BOLÍVAR
2         2/08/2016 8:09  VA - CENTRO DE SALUD CANDELARIA  I C. BOLÍVAR
3         3/08/2016 9:48  VA - CENTRO DE SALUD CANDELARIA  I C. BOLÍVAR
4        3/08/2016 13:26                VC - CENTRO DE SALUD JERUSALÉN
5         1/08/2016 9:11                         ME - HOSPITAL MEISSEN
…                    …                                            …
121256  19/05/2017 19:05                   VB - HOSPITAL VISTA HERMOSA
121257  18/05/2017 14:02                         ME - HOSPITAL MEISSEN
121258  20/05/2017 11:48                JC - CENTRO DE SALUD EL CARMEN
121259  25/05/2017 14:52                JC - CENTRO DE SALUD EL CARMEN
121260  25/05/2017 17:52                    VJ - CENTRO DE SALUD MANUE

                                CLASIFICACION_TRIAGE PACIENTE_#_DOCUMENTO  \
1       3 - TRIAGE III - NO DEBE SUPERAR LAS 3 HORAS              4108268
2       3 - TRIAGE III - NO DEBE SUPERAR LAS 3 HORAS           1026581403
3       3 - TRIAGE III - NO DEBE SUPERAR LAS 3 HORAS             19334228
4       3 - TRIAGE III - NO DEBE SUPERAR LAS 3 HORAS             52238647
5       3 - TRIAGE III - NO DEBE SUPERAR LAS 3 HORAS           1033743521
…                                                 …                   …
121256  3 - TRIAGE III - NO DEBE SUPERAR LAS 3 HORAS           1031160974
121257  3 - TRIAGE III - NO DEBE SUPERAR LAS 3 HORAS              5966047
121258  3 - TRIAGE III - NO DEBE SUPERAR LAS 3 HORAS           1018458868
121259      2 - TRIAGE II -  ANTES DE LOS 30 MINUTOS           1033768047
121260                                           NaN                  NaN

       PACIENTE_EDAD        EDAD_RANGO       SEXO RÉGIMEN PACIENTE  \
1          77 AÑO(S)             VEJEZ  MASCULINO       SUBSIDIADO
2          21 AÑO(S)           JUVENTUD   FEMENINO       SUBSIDIADO
3          65 AÑO(S)             VEJEZ  MASCULINO       SUBSIDIADO
4          40 AÑO(S)            ADULTEZ   FEMENINO        VINCULADO
5           6 AÑO(S)           INFANCIA  MASCULINO       SUBSIDIADO
…                 …                  …         …                …
121256      3 AÑO(S)  PRIMERA INFANCIA   FEMENINO     CONTRIBUTIVO
121257     70 AÑO(S)             VEJEZ  MASCULINO       SUBSIDIADO
121258     24 AÑO(S)           JUVENTUD   FEMENINO       SUBSIDIADO
121259      4 AÑO(S)  PRIMERA INFANCIA   FEMENINO       SUBSIDIADO
121260           NaN                NaN        NaN              NaN

                  NOMBRE_ENTIDAD              NOM_TIPO_HISTORIA DIAGNOSTICO  \
1       CAPITAL SALUD EPS-S S.A.S  CONSULTA  INICIAL DE URGENCIAS        H813
2       CAPITAL SALUD EPS-S S.A.S  CONSULTA  INICIAL DE URGENCIAS        A060
```

```
3       CAPITAL SALUD EPS-S S.A.S  CONSULTA  INICIAL DE URGENCIAS        A09X
4       CAPITAL SALUD EPS-S S.A.S       CONSULTA MEDICINA GENERAL        J039
5       CAPITAL SALUD EPS-S S.A.S  CONSULTA  INICIAL DE URGENCIAS        M798
…                             …                                 …         …
121256        SALUD TOTAL SA EPS   CONSULTA  INICIAL DE URGENCIAS        N390
121257  CAPITAL SALUD EPS-S S.A.S  CONSULTA  INICIAL DE URGENCIAS        K297
121258  CAPITAL SALUD EPS-S S.A.S  CONSULTA  INICIAL DE URGENCIAS        Z359
121259  CAPITAL SALUD EPS-S S.A.S  CONSULTA  INICIAL DE URGENCIAS        S903
121260                       NaN                              NaN         NaN

                                                 NOMBRE DX
1       OTROS VERTIGOS PERIFERICOS                       …
2       DISENTERIA AMEBIANA AGUDA                        …
3       DIARREA Y GASTROENTERITIS DE PRESUNTO ORIGEN I…
4       AMIGDALITIS AGUDA, NO ESPECIFICADA               …
5       OTROS TRASTORNOS ESPECIFICADOS DE LOS TEJIDOS …
…                                                        …
121256  INFECCION DE VIAS URINARIAS, SITIO NO ESPECIFI…
121257  GASTRITIS, NO ESPECIFICADA                       …
121258  SUPERVISION DE EMBARAZO DE ALTO RIESGO, SIN OT…
121259  CONTUSION DE OTRAS PARTES Y DE LAS NO ESPECIFI…
121260                                                 NaN

[121260 rows x 19 columns]
```

[ ]:
```
# crear la lista headers
headers =␣
  ↪["FECHA_LLEGADA","FECHA_TRIAGE","FECHA_INGRESO","FECHA_ATENCION","TIEMPO_DGTURNO_A_TRIAGE",
          ␣
  ↪"CENTRO_ATENCION","CLASIFICACION_TRIAGE","PACIENTE_#_DOCUMENTO","EDAD","EDAD_RANGO","SEXO",
  ↪PACIENTE","NOMBRE_ENTIDAD","MEDICO","AÑO","MES","DIA_SEMANA","HOUR","Turnos","TIME","DIA"]
print("headers\n", headers)
df.columns = headers
df = df.drop(0)
```

```
headers
 ['FECHA_LLEGADA', 'FECHA_TRIAGE', 'FECHA_INGRESO', 'FECHA_ATENCION',
'TIEMPO_DGTURNO_A_TRIAGE', 'TIEMPO_TRIAGE_A_INGRESO',
'TIEMPO_INGRESO_A_CONSULTA', 'TIEMPO_TOTAL', 'Tiempo_Minutos_Total',
'CENTRO_ATENCION', 'CLASIFICACION_TRIAGE', 'PACIENTE_#_DOCUMENTO', 'EDAD',
'EDAD_RANGO', 'SEXO', 'RÉGIMEN PACIENTE', 'NOMBRE_ENTIDAD', 'MEDICO', 'AÑO',
'MES', 'DIA_SEMANA', 'HOUR', 'Turnos', 'TIME', 'DIA']
```

[ ]:
```
df.dtypes
```

[ ]:
```
TIEMPO_LLEGADA_A_TRIAGE    object
TIEMPO_TRIAGE_A_INGRESO    object
```

```
TIEMPO_INGRESO_A_FOLIO     object
TIEMPO_TOTAL               object
FECHA_LLEGADA              object
FECHA_TRIAGE               object
FECHA_INGRESO              object
FECHA_CONSULTA             object
CENTRO_ATENCION            object
CLASIFICACION_TRIAGE       object
PACIENTE_#_DOCUMENTO       object
PACIENTE_EDAD              object
EDAD_RANGO                 object
SEXO                       object
RÉGIMEN PACIENTE           object
NOMBRE_ENTIDAD             object
NOM_TIPO_HISTORIA          object
DIAGNOSTICO                object
NOMBRE DX                  object
dtype: object
```

[ ]:
```python
# Arreglar Datos

df['Turnos'] = df['Turnos'].astype('category')
df['DIA_SEMANA'] = df['DIA_SEMANA'].astype('category')
df['CENTRO_ATENCION'] = df['CENTRO_ATENCION'].astype('category')

# convertir datos
df['Tiempo_Total'] = df['Tiempo_Minutos_Total'].str.replace(',', '.',␣
 ↪regex=True)
df['Tiempo_Minutos_Total'] = df['Tiempo_Minutos_Total'].str.replace(',', '.',␣
 ↪regex=True)
df['Tiempo_Minutos_Total'] = pd.to_numeric(df['Tiempo_Minutos_Total'],␣
 ↪errors='coerce')
df['Tiempo_Total'] = pd.to_numeric(df['Tiempo_Total'], errors='coerce')

df['FECHA_LLEGADA'] = pd.to_datetime(df['FECHA_LLEGADA'])

# Luego, usa la función strftime para obtener la fecha en el formato deseado
df['Month'] = df['FECHA_LLEGADA'].dt.strftime('%Y-%m-01')
df['Month'] = pd.to_datetime(df['Month'])

df['MES2'] = df['FECHA_LLEGADA'].dt.strftime('%Y-%m')
df['MES2'] = pd.to_datetime(df['MES2'])

# Cadena Más Común (Moda)  -  para reemplazar los datos vacios con el valor más␣
 ↪frecuente o la moda
promedio = df['Tiempo_Minutos_Total'].median()
df.loc[df['Tiempo_Minutos_Total'] > 420, 'Tiempo_Minutos_Total'] = promedio
```

```
df.loc[df['Tiempo_Minutos_Total'] < 0, 'Tiempo_Minutos_Total'] = promedio
```

[ ]:
```
# cargar una copia
dataset = df
```

[ ]:
```
dataset
```

[ ]:
```
                FECHA_LLEGADA             FECHA_TRIAGE  \
1     2023-01-01 01:20:23.853  2023-01-01 01:28:01.847
2     2023-01-01 01:29:46.050  2023-01-01 01:48:03.070
3     2023-01-01 03:15:35.623  2023-01-01 03:23:01.990
4     2023-01-01 05:54:53.563  2023-01-01 06:00:07.943
5     2023-01-01 06:37:27.237  2023-01-01 07:52:31.687
…                         …                         …
82412 2023-09-18 04:44:41.970  2023-09-18 04:53:22.553
82413 2023-09-18 06:17:00.573  2023-09-18 06:28:43.040
82414 2023-09-18 06:21:37.273  2023-09-18 07:00:57.420
82415 2023-09-18 06:25:33.483  2023-09-18 06:42:02.883
82416 2023-09-18 07:14:58.180  2023-09-18 07:30:50.643

                 FECHA_INGRESO           FECHA_ATENCION  \
1      2023-01-01 01:29:41.210  2023-01-01 02:00:07.590
2      2023-01-01 01:49:40.973  2023-01-01 02:02:53.663
3      2023-01-01 03:23:39.793  2023-01-01 03:30:21.233
4      2023-01-01 06:02:07.320  2023-01-01 06:26:17.050
5      2023-01-01 07:52:37.717  2023-01-01 09:31:15.597
…                          …                         …
82412  2023-09-18 05:05:51.423  2023-09-18 06:09:35.867
82413  2023-09-18 06:35:38.213  2023-09-18 07:40:45.957
82414  2023-09-18 07:16:45.907  2023-09-18 08:27:27.337
82415  2023-09-18 06:51:35.970  2023-09-18 07:28:28.290
82416  2023-09-18 07:34:08.370  2023-09-18 07:49:18.440

      TIEMPO_DGTURNO_A_TRIAGE TIEMPO_TRIAGE_A_INGRESO  \
1                     0:07:38                0:01:40
2                     0:18:17                0:01:37
3                     0:07:26                0:00:38
4                     0:05:14                0:02:00
5                     1:15:04                0:00:06
…                       …                         …
82412                 0:08:41                0:12:29
82413                 0:11:43                0:06:55
82414                 0:39:20                0:15:48
82415                 0:16:29                0:09:33
82416                 0:15:52                0:03:18

      TIEMPO_INGRESO_A_CONSULTA TIEMPO_TOTAL  Tiempo_Minutos_Total  \
```

```
1                      0:30:26      0:39:44                39.73
2                      0:13:13      0:33:07                33.12
3                      0:06:42      0:14:46                14.77
4                      0:24:10      0:31:24                31.40
5                      1:38:38      2:53:48               173.80
…                        …          …                   …
82412                  1:03:44      1:24:54                84.90
82413                  1:05:07      1:23:45                83.75
82414                  1:10:42      2:05:50               125.83
82415                  0:36:53      1:02:55                62.92
82416                  0:15:10      0:34:20                34.33

      CENTRO_ATENCION  …   AÑO MES DIA_SEMANA HOUR  Turnos  \
1                  TN   …  2023   1    DOMINGO    1   NOCHE
2                  ME   …  2023   1    DOMINGO    1   NOCHE
3                  UC   …  2023   1    DOMINGO    3   NOCHE
4                  UC   …  2023   1    DOMINGO    5   NOCHE
5                  TN   …  2023   1    DOMINGO    6   NOCHE
…                 …     …   …    ..        …  …     …
82412              ME   …  2023   9      LUNES    4   NOCHE
82413              ME   …  2023   9      LUNES    6   NOCHE
82414              ME   …  2023   9      LUNES    6   NOCHE
82415              UB   …  2023   9      LUNES    6   NOCHE
82416              TN   …  2023   9      LUNES    7  MAÑANA

                          TIME DIA Tiempo_Total      Month       MES2
1      2023-01-01 01:20:23.853   1        39.73 2023-01-01 2023-01-01
2      2023-01-01 01:29:46.050   1        33.12 2023-01-01 2023-01-01
3      2023-01-01 03:15:35.623   1        14.77 2023-01-01 2023-01-01
4      2023-01-01 05:54:53.563   1        31.40 2023-01-01 2023-01-01
5      2023-01-01 06:37:27.237   1       173.80 2023-01-01 2023-01-01
…                         …  ..          …         …          …
82412  2023-09-18 04:44:41.970  18        84.90 2023-09-01 2023-09-01
82413  2023-09-18 06:17:00.573  18        83.75 2023-09-01 2023-09-01
82414  2023-09-18 06:21:37.273  18       125.83 2023-09-01 2023-09-01
82415  2023-09-18 06:25:33.483  18        62.92 2023-09-01 2023-09-01
82416  2023-09-18 07:14:58.180  18        34.33 2023-09-01 2023-09-01

[82416 rows x 28 columns]
```

[ ]:
```
print(df.isnull().sum())
```

[ ]:
```
# La prueba de Dickey-Fuller aumentada
def Augmented_Dickey_Fuller_Test_func(series , column_name):
    print (f'Resultados de la prueba de Dickey-Fuller para columna:␣
 ↪{column_name}')
    dftest = adfuller(series, autolag='AIC')
```

```
    dfoutput = pd.Series(dftest[0:4], index=['Test Statistic','p-value','No␣
↪Lags Used','Número de observaciones utilizadas'])
    for key,value in dftest[4].items():
       dfoutput['Critical Value (%s)'%key] = value
    print (dfoutput)
    if dftest[1] <= 0.05:
        print("Conclusion:====>")
        print("Rechazar la hipótesis nula")
        print("Los datos son estacionarios")
    else:
        print("Conclusion:====>")
        print("No se puede rechazar la hipótesis nula")
        print("Los datos no son estacionarios")
```

[ ]:
```
# ejecutar la prueba
Augmented_Dickey_Fuller_Test_func(df["Tiempo_Minutos_Total"],"Tiempo_Minutos_Total")
```

[ ]:
```
plt.rcParams["figure.figsize"] = (12, 8)
a = seasonal_decompose(df["Tiempo_Minutos_Total"], model = "add")
a.plot();
```

[ ]:
```
# División de para entrenamiento y prueba
train_data = df[:len(df)-12]
test_data = df[len(df)-12:]
test=test_data.copy()
```

[ ]:
```
def evaluacion_metrica(y_true, y_pred):

    def mean_absolute_percentage_error(y_true, y_pred):
        y_true, y_pred = np.array(y_true), np.array(y_pred)
        return np.mean(np.abs((y_true - y_pred) / y_true)) * 100
    print('Evaluation metric results:-')
    print(f'MSE is : {metrics.mean_squared_error(y_true, y_pred)}')
    print(f'MAE is : {metrics.mean_absolute_error(y_true, y_pred)}')
    print(f'RMSE is : {np.sqrt(metrics.mean_squared_error(y_true, y_pred))}')
    print(f'MAPE is : {mean_absolute_percentage_error(y_true, y_pred)}')
    print(f'R2 is : {metrics.r2_score(y_true, y_pred)}',end='\n\n')
```

[ ]:
```
df = dataset
```

[ ]:
```
df = df.set_index("Month")
#df.index.freq = 'MS'
```

[ ]:
```
df
```

[ ]:
```
df.index.freq = 'MS'
```

```
---------------------------------------------------------------------------
OutOfBoundsDatetime                       Traceback (most recent call last)
/usr/local/lib/python3.10/dist-packages/pandas/core/arrays/datetimelike.py in␣
 ↪_validate_frequency(cls, index, freq, **kwargs)
   1003         try:
-> 1004             on_freq = cls._generate_range(
   1005                 start=index[0], end=None, periods=len(index), freq=freq␣
 ↪**kwargs

/usr/local/lib/python3.10/dist-packages/pandas/core/arrays/datetimes.py in␣
 ↪_generate_range(cls, start, end, periods, freq, tz, normalize, ambiguous,␣
 ↪nonexistent, inclusive)
    396                 xdr = generate_range(start=start, end=end,␣
 ↪periods=periods, offset=freq)
--> 397                 i8values = np.array([x.value for x in xdr], dtype=np.
 ↪int64)
    398

/usr/local/lib/python3.10/dist-packages/pandas/core/arrays/datetimes.py in␣
 ↪<listcomp>(.0)
    396                 xdr = generate_range(start=start, end=end,␣
 ↪periods=periods, offset=freq)
--> 397                 i8values = np.array([x.value for x in xdr], dtype=np.
 ↪int64)
    398

/usr/local/lib/python3.10/dist-packages/pandas/core/arrays/datetimes.py in␣
 ↪generate_range(start, end, periods, offset)
   2552     if end is None:
-> 2553         end = start + (periods - 1) * offset
   2554

/usr/local/lib/python3.10/dist-packages/pandas/_libs/tslibs/offsets.pyx in␣
 ↪pandas._libs.tslibs.offsets.BaseOffset.__add__()

/usr/local/lib/python3.10/dist-packages/pandas/_libs/tslibs/offsets.pyx in␣
 ↪pandas._libs.tslibs.offsets.BaseOffset.__add__()

/usr/local/lib/python3.10/dist-packages/pandas/_libs/tslibs/offsets.pyx in␣
 ↪pandas._libs.tslibs.offsets.apply_wraps.wrapper()

/usr/local/lib/python3.10/dist-packages/pandas/_libs/tslibs/offsets.pyx in␣
 ↪pandas._libs.tslibs.offsets.MonthOffset._apply()

/usr/local/lib/python3.10/dist-packages/pandas/_libs/tslibs/offsets.pyx in␣
 ↪pandas._libs.tslibs.offsets.shift_month()
```

```
/usr/local/lib/python3.10/dist-packages/pandas/_libs/tslibs/timestamps.pyx in␣
 ↪pandas._libs.tslibs.timestamps.Timestamp.replace()

/usr/local/lib/python3.10/dist-packages/pandas/_libs/tslibs/conversion.pyx in␣
 ↪pandas._libs.tslibs.conversion.convert_datetime_to_tsobject()

/usr/local/lib/python3.10/dist-packages/pandas/_libs/tslibs/np_datetime.pyx in␣
 ↪pandas._libs.tslibs.np_datetime.check_dts_bounds()

OutOfBoundsDatetime: Out of bounds nanosecond timestamp: 8890-12-01 00:00:00

The above exception was the direct cause of the following exception:

ValueError                                Traceback (most recent call last)
<ipython-input-38-0dc1e7b74d6b> in <cell line: 1>()
----> 1 df.index.freq = 'MS'

/usr/local/lib/python3.10/dist-packages/pandas/core/indexes/extension.py in␣
 ↪fset(self, value)
     79
     80             def fset(self, value):
---> 81                 setattr(self._data, name, value)
     82
     83             fget.__name__ = name

/usr/local/lib/python3.10/dist-packages/pandas/core/arrays/datetimelike.py in␣
 ↪freq(self, value)
    935         if value is not None:
    936             value = to_offset(value)
--> 937             self._validate_frequency(self, value)
    938
    939             if self.ndim > 1:

/usr/local/lib/python3.10/dist-packages/pandas/core/arrays/datetimelike.py in␣
 ↪_validate_frequency(cls, index, freq, **kwargs)
   1017             #  raise a ValueError, which we re-raise with a more target↪d
   1018             #  message.
-> 1019             raise ValueError(
   1020                 f"Inferred frequency {inferred} from passed values "
   1021                 f"does not conform to passed frequency {freq.freqstr}"

ValueError: Inferred frequency None from passed values does not conform to␣
 ↪passed frequency MS
```

[ ]:
```
# agrupar x mes
```

```
DfSalidas = train_data_pr.set_index("MES")
DfSalidas['Original'] = DfSalidas.groupby('MES')['y'].mean()
DfSalida = train_data_pr.groupby('MES')['y'].mean()
DfSalida
```

[ ]:
```
# Modelo 1. Prophet

from prophet import Prophet

df1 = df.copy()
df1=df1.reset_index()

# Asegúrate de que la columna 'ds' sea de tipo datetime

df_fb=df1.rename(columns={"Month":"ds", "Tiempo_Minutos_Total":"y"} )

train_data_pr = df_fb.iloc[:len(df1)-12]
test_data_pr = df_fb.iloc[len(df1)-12:]

m = Prophet()
#m.fit(train_data_pr[mask])
m.fit(train_data_pr)

future = m.make_future_dataframe(periods=12,freq='MS')
#future = m.make_future_dataframe(periods=12)

forecast = m.predict(future)
forecast.tail()
```

```
INFO:prophet:Disabling yearly seasonality. Run prophet with
yearly_seasonality=True to override this.
INFO:prophet:Disabling weekly seasonality. Run prophet with
weekly_seasonality=True to override this.
INFO:prophet:Disabling daily seasonality. Run prophet with
daily_seasonality=True to override this.
DEBUG:cmdstanpy:input tempfile: /tmp/tmplapidlw2/golukt5_.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmplapidlw2/qv29t5sp.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.10/dist-
packages/prophet/stan_model/prophet_model.bin', 'random', 'seed=72001', 'data',
'file=/tmp/tmplapidlw2/golukt5_.json', 'init=/tmp/tmplapidlw2/qv29t5sp.json',
'output',
'file=/tmp/tmplapidlw2/prophet_model5hto0ztf/prophet_model-20231108134739.csv',
'method=optimize', 'algorithm=lbfgs', 'iter=10000']
13:47:39 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
```

```
13:47:43 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
```

[ ]:
```
           ds       trend  yhat_lower  yhat_upper  trend_lower  trend_upper  \
16 2024-05-01  100.188032   12.523192  185.699177    44.276265   151.734825
17 2024-06-01  101.359661   17.585352  190.565635    34.391777   164.849146
18 2024-07-01  102.493496    7.623568  191.643113    25.680326   177.399690
19 2024-08-01  103.665125   -4.099542  204.909838    14.731213   189.954833
20 2024-09-01  104.836754  -11.027283  222.708577     3.026620   204.322978

    additive_terms  additive_terms_lower  additive_terms_upper  \
16             0.0                   0.0                   0.0
17             0.0                   0.0                   0.0
18             0.0                   0.0                   0.0
19             0.0                   0.0                   0.0
20             0.0                   0.0                   0.0

    multiplicative_terms  multiplicative_terms_lower  \
16                   0.0                         0.0
17                   0.0                         0.0
18                   0.0                         0.0
19                   0.0                         0.0
20                   0.0                         0.0

    multiplicative_terms_upper        yhat
16                         0.0  100.188032
17                         0.0  101.359661
18                         0.0  102.493496
19                         0.0  103.665125
20                         0.0  104.836754
```

[ ]:
```
# asignar para grafica
prophet_pred = pd.DataFrame({"Date" : forecast[-12:]['ds'], "Pred" :␣
 ↪forecast[-12:]["yhat"]})
prophet_pred = prophet_pred.set_index("Date")
prophet_pred.index.freq = "MS"
prophet_pred
```

[ ]:
```
                Pred
Date
2023-10-01   92.137806
2023-11-01   93.309436
2023-12-01   94.443270
2024-01-01   95.614899
2024-02-01   96.786528
2024-03-01   97.882568
2024-04-01   99.054197
```

```
2024-05-01  100.188032
2024-06-01  101.359661
2024-07-01  102.493496
2024-08-01  103.665125
2024-09-01  104.836754
```

[ ]:
```
train_data_pr
```

[ ]:
```
       index           FECHA_LLEGADA             FECHA_TRIAGE  \
0          1 2023-01-01 01:20:23.853  2023-01-01 01:28:01.847
1          2 2023-01-01 01:29:46.050  2023-01-01 01:48:03.070
2          3 2023-01-01 03:15:35.623  2023-01-01 03:23:01.990
3          4 2023-01-01 05:54:53.563  2023-01-01 06:00:07.943
4          5 2023-01-01 06:37:27.237  2023-01-01 07:52:31.687
…        …                       …                         …
82399  82400 2023-09-17 21:12:50.687  2023-09-17 21:19:39.883
82400  82401 2023-09-17 22:39:05.587  2023-09-17 22:57:11.653
82401  82402 2023-09-17 22:59:10.483  2023-09-17 23:16:29.810
82402  82403 2023-09-17 23:31:10.827  2023-09-17 23:47:05.270
82403  82404 2023-09-17 23:56:53.490  2023-09-18 00:05:14.693

                  FECHA_INGRESO           FECHA_ATENCION  \
0      2023-01-01 01:29:41.210  2023-01-01 02:00:07.590
1      2023-01-01 01:49:40.973  2023-01-01 02:02:53.663
2      2023-01-01 03:23:39.793  2023-01-01 03:30:21.233
3      2023-01-01 06:02:07.320  2023-01-01 06:26:17.050
4      2023-01-01 07:52:37.717  2023-01-01 09:31:15.597
…                          …                       …
82399  2023-09-17 21:39:45.623  2023-09-17 22:32:02.003
82400  2023-09-17 22:59:29.137  2023-09-17 23:22:06.090
82401  2023-09-17 23:21:23.987  2023-09-18 00:35:01.290
82402  2023-09-17 23:48:34.393  2023-09-18 00:07:43.070
82403  2023-09-18 00:06:16.227  2023-09-18 01:59:02.380

      TIEMPO_DGTURNO_A_TRIAGE TIEMPO_TRIAGE_A_INGRESO  \
0                     0:07:38                 0:01:40
1                     0:18:17                 0:01:37
2                     0:07:26                 0:00:38
3                     0:05:14                 0:02:00
4                     1:15:04                 0:00:06
…                       …                       …
82399                 0:06:49                 0:20:06
82400                 0:18:06                 0:02:18
82401                 0:17:19                 0:04:54
82402                 0:15:55                 0:01:29
82403                 0:08:21                 0:01:02
```

```
      TIEMPO_INGRESO_A_CONSULTA TIEMPO_TOTAL       y  … MEDICO   AÑO MES  \
0                       0:30:26      0:39:44   39.73  …  11065  2023   1
1                       0:13:13      0:33:07   33.12  …   8861  2023   1
2                       0:06:42      0:14:46   14.77  …   5855  2023   1
3                       0:24:10      0:31:24   31.40  …  11072  2023   1
4                       1:38:38      2:53:48  173.80  …   1239  2023   1
…                         …          …     …  …    …   …  ..
82399                   0:52:17      1:19:12   79.20  …   7805  2023   9
82400                   0:22:37      0:43:01   43.02  …  11006  2023   9
82401                   1:13:38      1:35:51   95.85  …    962  2023   9
82402                   0:19:09      0:36:33   36.55  …   8491  2023   9
82403                   1:52:46      2:02:09  122.15  …   3540  2023   9

      DIA_SEMANA HOUR Turnos                      TIME DIA Tiempo_Total  \
0        DOMINGO    1  NOCHE  2023-01-01 01:20:23.853   1        39.73
1        DOMINGO    1  NOCHE  2023-01-01 01:29:46.050   1        33.12
2        DOMINGO    3  NOCHE  2023-01-01 03:15:35.623   1        14.77
3        DOMINGO    5  NOCHE  2023-01-01 05:54:53.563   1        31.40
4        DOMINGO    6  NOCHE  2023-01-01 06:37:27.237   1       173.80
…          …  …    …                          …  ..          …
82399    DOMINGO   21  NOCHE  2023-09-17 21:12:50.687  17        79.20
82400    DOMINGO   22  NOCHE  2023-09-17 22:39:05.587  17        43.02
82401    DOMINGO   22  NOCHE  2023-09-17 22:59:10.483  17        95.85
82402    DOMINGO   23  NOCHE  2023-09-17 23:31:10.827  17        36.55
82403    DOMINGO   23  NOCHE  2023-09-17 23:56:53.490  17       122.15

              ds
0     2023-01-01
1     2023-01-01
2     2023-01-01
3     2023-01-01
4     2023-01-01
…          …
82399 2023-09-01
82400 2023-09-01
82401 2023-09-01
82402 2023-09-01
82403 2023-09-01

[82404 rows x 28 columns]
```

[ ]:
```python
# crear un datagrame
DfSalidas = train_data_pr.set_index("MES")
DfSalidas['Original'] = DfSalidas.groupby('MES')['y'].mean()
DfSalida = train_data_pr.groupby('MES')['y'].mean()
DfSalida
```

[ ]:
```
MES
3    107.974032
4     95.203582
5    104.839185
6     97.895240
7     88.040411
8     90.971641
9     90.144204
1     84.455127
2     92.712531
3    101.072168
4     94.493696
5    105.348111
6     96.142285
7     74.907123
Name: y, dtype: float64
```

[ ]:
```
DfSalidas
```

[ ]:
```
      index           FECHA_LLEGADA             FECHA_TRIAGE  \
MES
1         1 2023-01-01 01:20:23.853  2023-01-01 01:28:01.847
1         2 2023-01-01 01:29:46.050  2023-01-01 01:48:03.070
1         3 2023-01-01 03:15:35.623  2023-01-01 03:23:01.990
1         4 2023-01-01 05:54:53.563  2023-01-01 06:00:07.943
1         5 2023-01-01 06:37:27.237  2023-01-01 07:52:31.687
..      ...                     ...                      ...
9     82400 2023-09-17 21:12:50.687  2023-09-17 21:19:39.883
9     82401 2023-09-17 22:39:05.587  2023-09-17 22:57:11.653
9     82402 2023-09-17 22:59:10.483  2023-09-17 23:16:29.810
9     82403 2023-09-17 23:31:10.827  2023-09-17 23:47:05.270
9     82404 2023-09-17 23:56:53.490  2023-09-18 00:05:14.693

                FECHA_INGRESO           FECHA_ATENCION TIEMPO_DGTURNO_A_TRIAGE  \
MES
1     2023-01-01 01:29:41.210  2023-01-01 02:00:07.590                0:07:38
1     2023-01-01 01:49:40.973  2023-01-01 02:02:53.663                0:18:17
1     2023-01-01 03:23:39.793  2023-01-01 03:30:21.233                0:07:26
1     2023-01-01 06:02:07.320  2023-01-01 06:26:17.050                0:05:14
1     2023-01-01 07:52:37.717  2023-01-01 09:31:15.597                1:15:04
..                        ...                      ...                    ...
9     2023-09-17 21:39:45.623  2023-09-17 22:32:02.003                0:06:49
9     2023-09-17 22:59:29.137  2023-09-17 23:22:06.090                0:18:06
9     2023-09-17 23:21:23.987  2023-09-18 00:35:01.290                0:17:19
9     2023-09-17 23:48:34.393  2023-09-18 00:07:43.070                0:15:55
9     2023-09-18 00:06:16.227  2023-09-18 01:59:02.380                0:08:21
```

```
    TIEMPO_TRIAGE_A_INGRESO TIEMPO_INGRESO_A_CONSULTA TIEMPO_TOTAL       y  \
MES
1                  0:01:40                   0:30:26      0:39:44   39.73
1                  0:01:37                   0:13:13      0:33:07   33.12
1                  0:00:38                   0:06:42      0:14:46   14.77
1                  0:02:00                   0:24:10      0:31:24   31.40
1                  0:00:06                   1:38:38      2:53:48  173.80
..                     …                         …          …     …
9                  0:20:06                   0:52:17      1:19:12   79.20
9                  0:02:18                   0:22:37      0:43:01   43.02
9                  0:04:54                   1:13:38      1:35:51   95.85
9                  0:01:29                   0:19:09      0:36:33   36.55
9                  0:01:02                   1:52:46      2:02:09  122.15

     … MEDICO   AÑO DIA_SEMANA HOUR Turnos                     TIME DIA  \
MES  …
1    …  11065  2023    DOMINGO    1  NOCHE  2023-01-01 01:20:23.853   1
1    …   8861  2023    DOMINGO    1  NOCHE  2023-01-01 01:29:46.050   1
1    …   5855  2023    DOMINGO    3  NOCHE  2023-01-01 03:15:35.623   1
1    …  11072  2023    DOMINGO    5  NOCHE  2023-01-01 05:54:53.563   1
1    …   1239  2023    DOMINGO    6  NOCHE  2023-01-01 06:37:27.237   1
..   …    …   …         …  …    …                       …  ..
9    …   7805  2023    DOMINGO   21  NOCHE  2023-09-17 21:12:50.687  17
9    …  11006  2023    DOMINGO   22  NOCHE  2023-09-17 22:39:05.587  17
9    …    962  2023    DOMINGO   22  NOCHE  2023-09-17 22:59:10.483  17
9    …   8491  2023    DOMINGO   23  NOCHE  2023-09-17 23:31:10.827  17
9    …   3540  2023    DOMINGO   23  NOCHE  2023-09-17 23:56:53.490  17

    Tiempo_Total         ds   Original
MES
1          39.73 2023-01-01  84.455127
1          33.12 2023-01-01  84.455127
1          14.77 2023-01-01  84.455127
1          31.40 2023-01-01  84.455127
1         173.80 2023-01-01  84.455127
..           …        …        …
9          79.20 2023-09-01  90.144204
9          43.02 2023-09-01  90.144204
9          95.85 2023-09-01  90.144204
9          36.55 2023-09-01  90.144204
9         122.15 2023-09-01  90.144204

[82404 rows x 28 columns]
```

[ ]:
```
DfSalida
```

[ ]:
```
MES
3    107.974032
4     95.203582
5    104.839185
6     97.895240
7     88.040411
8     90.971641
9     90.144204
1     84.455127
2     92.712531
3    101.072168
4     94.493696
5    105.348111
6     96.142285
7     74.907123
Name: y, dtype: float64
```

[ ]:
```
# crear la lista headers
headers = ["MES","T_PROM_ORI"]
print("headers\n", headers)
DfSalida.columns = headers
#df = df.drop(0)
```

```
headers
 ['MES', 'T_PROM_ORI']
```

[ ]:
```
#DfSalida
DfSalida.dtypes
```

[ ]:
```
dtype('float64')
```

[ ]:
```
# Modelo 1. Prophet

from prophet import Prophet
mask_Prediccion = 90

df1=df.reset_index()
df1['Fecha'] = df1['FECHA_LLEGADA'].dt.date
dfp = df1[['Fecha', 'Tiempo_Minutos_Total']].copy()
dfp.rename(columns={'Fecha': 'ds', 'Tiempo_Minutos_Total': 'y'}, inplace=True)
# dfp["y"] = pd.to_numeric(dfp["y"],errors='coerce')

train_data_pr = dfp.iloc[:len(dfp)-12]
test_data_pr = dfp.iloc[len(dfp)-12:]
m = Prophet()
#m.fit(dfp[mask])
m.fit(train_data_pr)
```

```
future = m.make_future_dataframe(periods=12,freq='MS')
forecast = m.predict(future)

# forecast.tail()
```

[ ]:
```
dfp
```

[ ]:
```
                ds       y
0       2023-01-01   39.73
1       2023-01-01   33.12
2       2023-01-01   14.77
3       2023-01-01   31.40
4       2023-01-01  173.80
…              …      …
82411   2023-09-18   84.90
82412   2023-09-18   83.75
82413   2023-09-18  125.83
82414   2023-09-18   62.92
82415   2023-09-18   34.33

[82416 rows x 2 columns]
```

[ ]:
```
dfp
dfp['MesO'] = dfp['ds'].dt.month
```

[ ]:
```
dfp
dfp['MesO'] = dfp['ds'].dt.month
dfp2 = dfp.groupby('MesO')['y'].mean()
dfp2
```

[ ]:
```
forecast['MesP'] = forecast['ds'].dt.month
prophet_pred = forecast.groupby('MesP')['yhat'].mean()
prophet_pred
```

[ ]:
```
MesP
1       84.842009
2       90.059295
3       99.192942
4       94.603973
5      103.509761
6       94.966810
7       86.573396
8       90.462421
9       87.756800
10      79.649965
11      88.539396
12      85.312914
```

```
Name: yhat, dtype: float64
```

[ ]:
```
dfp2["Prophet_Predictions"] = prophet_pred['Pred'].values
```

[ ]:
```
dfp2
```

[ ]:
```
ds
1                                                    84.455127
2                                                    92.712531
3                                                   101.368506
4                                                    94.984109
5                                                   104.962839
6                                                    97.645105
7                                                    87.853414
8                                                    90.971641
9                                                     90.11574
Prophet_Predictions    [84.49256491, 92.79432996437498, 101.096437922…
Name: y, dtype: object
```

[ ]:
```
forecast
```

[ ]:
```
                           ds         trend    yhat_lower    yhat_upper  \
0  1970-01-01 00:00:00.000000001  8.449256e+01  2.270472e+01  1.512173e+02
1  1970-01-01 00:00:00.000000002  9.279433e+01  2.718982e+01  1.548589e+02
2  1970-01-01 00:00:00.000000003  1.010964e+02  3.479014e+01  1.647193e+02
3  1970-01-01 00:00:00.000000004  9.536102e+01  3.106702e+01  1.532594e+02
4  1970-01-01 00:00:00.000000005  1.047706e+02  4.215136e+01  1.663403e+02
..                           …           …           …           …
94 1970-03-28 00:00:00.000000009  8.743220e+15 -1.671278e+17  1.934413e+17
95 1970-03-29 00:00:00.000000009  8.844885e+15 -1.696684e+17  1.969466e+17
96 1970-03-30 00:00:00.000000009  8.946550e+15 -1.734813e+17  2.011411e+17
97 1970-03-31 00:00:00.000000009  9.048216e+15 -1.766101e+17  2.043906e+17
98 1970-04-01 00:00:00.000000009  9.149881e+15 -1.794139e+17  2.077006e+17

     trend_lower   trend_upper  additive_terms  additive_terms_lower  \
0   8.449256e+01  8.449256e+01             0.0                   0.0
1   9.279433e+01  9.279433e+01             0.0                   0.0
2   1.010964e+02  1.010964e+02             0.0                   0.0
3   9.536102e+01  9.536102e+01             0.0                   0.0
4   1.047706e+02  1.047706e+02             0.0                   0.0
..           …             …               …                     …
94 -1.671278e+17  1.934413e+17             0.0                   0.0
95 -1.696684e+17  1.969466e+17             0.0                   0.0
96 -1.734813e+17  2.011411e+17             0.0                   0.0
97 -1.766101e+17  2.043906e+17             0.0                   0.0
98 -1.794139e+17  2.077006e+17             0.0                   0.0
```

```
    additive_terms_upper  multiplicative_terms  multiplicative_terms_lower  \
0                    0.0                   0.0                         0.0
1                    0.0                   0.0                         0.0
2                    0.0                   0.0                         0.0
3                    0.0                   0.0                         0.0
4                    0.0                   0.0                         0.0
..                   …                     …                           …
94                   0.0                   0.0                         0.0
95                   0.0                   0.0                         0.0
96                   0.0                   0.0                         0.0
97                   0.0                   0.0                         0.0
98                   0.0                   0.0                         0.0

    multiplicative_terms_upper          yhat
0                          0.0  8.449256e+01
1                          0.0  9.279433e+01
2                          0.0  1.010964e+02
3                          0.0  9.536102e+01
4                          0.0  1.047706e+02
..                         …             …
94                         0.0  8.743220e+15
95                         0.0  8.844885e+15
96                         0.0  8.946550e+15
97                         0.0  9.048216e+15
98                         0.0  9.149881e+15

[99 rows x 13 columns]
```

[ ]:
```
prophet_pred
```

[ ]:
```
                 Pred
Date
2023-10-01   92.137806
2023-11-01   93.309436
2023-12-01   94.443270
2024-01-01   95.614899
2024-02-01   96.786528
2024-03-01   97.882568
2024-04-01   99.054197
2024-05-01  100.188032
2024-06-01  101.359661
2024-07-01  102.493496
2024-08-01  103.665125
2024-09-01  104.836754
```

[ ]:
```
prophet_pred = pd.DataFrame({"Date" : forecast['ds'], "Pred" :␣
↪forecast["yhat"]})
```

[ ]:
```
prophet_pred = pd.DataFrame({"Date" : forecast['ds'], "Pred" : forecast["y"]})
#prophet_pred = prophet_pred.set_index("Date")
#prophet_pred.index.freq = "MS"
```

[ ]:
```
test_data["Prophet_Predictions"] = prophet_pred['Pred'].values
```

[ ]:
```
prophet_pred
```

[ ]:
```
                 Pred
Date
2023-10-01   92.137806
2023-11-01   93.309436
2023-12-01   94.443270
2024-01-01   95.614899
2024-02-01   96.786528
2024-03-01   97.882568
2024-04-01   99.054197
2024-05-01  100.188032
2024-06-01  101.359661
2024-07-01  102.493496
2024-08-01  103.665125
2024-09-01  104.836754
```

[ ]:
```
test_data["Prophet_Predictions"] = prophet_pred['Pred'].values
```

[ ]:
```
test_data
```

[ ]:
```
                FECHA_LLEGADA             FECHA_TRIAGE  \
82405 2023-09-18 07:00:26.697  2023-09-18 07:08:30.043
82406 2023-09-17 17:28:11.360  2023-09-17 17:33:14.930
82407 2023-09-17 18:12:40.660  2023-09-17 18:43:16.430
82408 2023-09-17 18:52:28.640  2023-09-17 19:48:17.070
82409 2023-09-17 19:57:56.033  2023-09-17 20:15:56.793
82410 2023-09-17 20:22:04.970  2023-09-17 20:36:16.703
82411 2023-09-17 21:22:59.137  2023-09-17 21:42:20.273
82412 2023-09-18 04:44:41.970  2023-09-18 04:53:22.553
82413 2023-09-18 06:17:00.573  2023-09-18 06:28:43.040
82414 2023-09-18 06:21:37.273  2023-09-18 07:00:57.420
82415 2023-09-18 06:25:33.483  2023-09-18 06:42:02.883
82416 2023-09-18 07:14:58.180  2023-09-18 07:30:50.643

                FECHA_INGRESO           FECHA_ATENCION  \
82405  2023-09-18 07:14:16.823  2023-09-18 07:32:53.187
82406  2023-09-17 17:34:42.173  2023-09-17 18:36:55.167
82407  2023-09-17 18:48:36.403  2023-09-17 19:40:11.800
82408  2023-09-17 19:49:47.657  2023-09-17 20:32:54.070
82409  2023-09-17 20:28:57.590  2023-09-17 20:46:29.250
```

```
82410  2023-09-17 20:41:12.783  2023-09-17 22:00:50.287
82411  2023-09-17 21:51:04.433  2023-09-17 22:43:13.710
82412  2023-09-18 05:05:51.423  2023-09-18 06:09:35.867
82413  2023-09-18 06:35:38.213  2023-09-18 07:40:45.957
82414  2023-09-18 07:16:45.907  2023-09-18 08:27:27.337
82415  2023-09-18 06:51:35.970  2023-09-18 07:28:28.290
82416  2023-09-18 07:34:08.370  2023-09-18 07:49:18.440

      TIEMPO_DGTURNO_A_TRIAGE TIEMPO_TRIAGE_A_INGRESO  \
82405                 0:08:04                 0:05:46
82406                 0:05:03                 0:01:28
82407                 0:30:36                 0:05:20
82408                 0:55:49                 0:01:30
82409                 0:18:00                 0:13:01
82410                 0:14:12                 0:04:56
82411                 0:19:21                 0:08:44
82412                 0:08:41                 0:12:29
82413                 0:11:43                 0:06:55
82414                 0:39:20                 0:15:48
82415                 0:16:29                 0:09:33
82416                 0:15:52                 0:03:18

      TIEMPO_INGRESO_A_CONSULTA TIEMPO_TOTAL  Tiempo_Minutos_Total  \
82405                   0:18:37      0:32:27                 32.45
82406                   1:02:13      1:08:44                 68.73
82407                   0:51:35      1:27:31                 87.52
82408                   0:43:07      1:40:26                100.43
82409                   0:17:32      0:48:33                 48.55
82410                   1:19:38      1:38:46                 98.77
82411                   0:52:09      1:20:14                 80.23
82412                   1:03:44      1:24:54                 84.90
82413                   1:05:07      1:23:45                 83.75
82414                   1:10:42      2:05:50                125.83
82415                   0:36:53      1:02:55                 62.92
82416                   0:15:10      0:34:20                 34.33

      CENTRO_ATENCION  …   AÑO MES DIA_SEMANA HOUR  Turnos  \
82405              ME  …  2023   9      LUNES    7  MAÑANA
82406              ME  …  2023   9    DOMINGO   17   TARDE
82407              ME  …  2023   9    DOMINGO   18   TARDE
82408              ME  …  2023   9    DOMINGO   18   TARDE
82409              UC  …  2023   9    DOMINGO   19   TARDE
82410              ME  …  2023   9    DOMINGO   20   NOCHE
82411              ME  …  2023   9    DOMINGO   21   NOCHE
82412              ME  …  2023   9      LUNES    4   NOCHE
82413              ME  …  2023   9      LUNES    6   NOCHE
82414              ME  …  2023   9      LUNES    6   NOCHE
```

```
82415              UB  …  2023   9      LUNES    6   NOCHE
82416              TN  …  2023   9      LUNES    7  MAÑANA

                         TIME DIA Tiempo_Total      Month Prophet_Predictions
82405  2023-09-18 07:00:26.697  18        32.45 2023-09-01           92.137806
82406  2023-09-17 17:28:11.360  17        68.73 2023-09-01           93.309436
82407  2023-09-17 18:12:40.660  17        87.52 2023-09-01           94.443270
82408  2023-09-17 18:52:28.640  17       100.43 2023-09-01           95.614899
82409  2023-09-17 19:57:56.033  17        48.55 2023-09-01           96.786528
82410  2023-09-17 20:22:04.970  17        98.77 2023-09-01           97.882568
82411  2023-09-17 21:22:59.137  17        80.23 2023-09-01           99.054197
82412  2023-09-18 04:44:41.970  18        84.90 2023-09-01          100.188032
82413  2023-09-18 06:17:00.573  18        83.75 2023-09-01          101.359661
82414  2023-09-18 06:21:37.273  18       125.83 2023-09-01          102.493496
82415  2023-09-18 06:25:33.483  18        62.92 2023-09-01          103.665125
82416  2023-09-18 07:14:58.180  18        34.33 2023-09-01          104.836754

[12 rows x 28 columns]
```

[ ]:
```
test_data
```

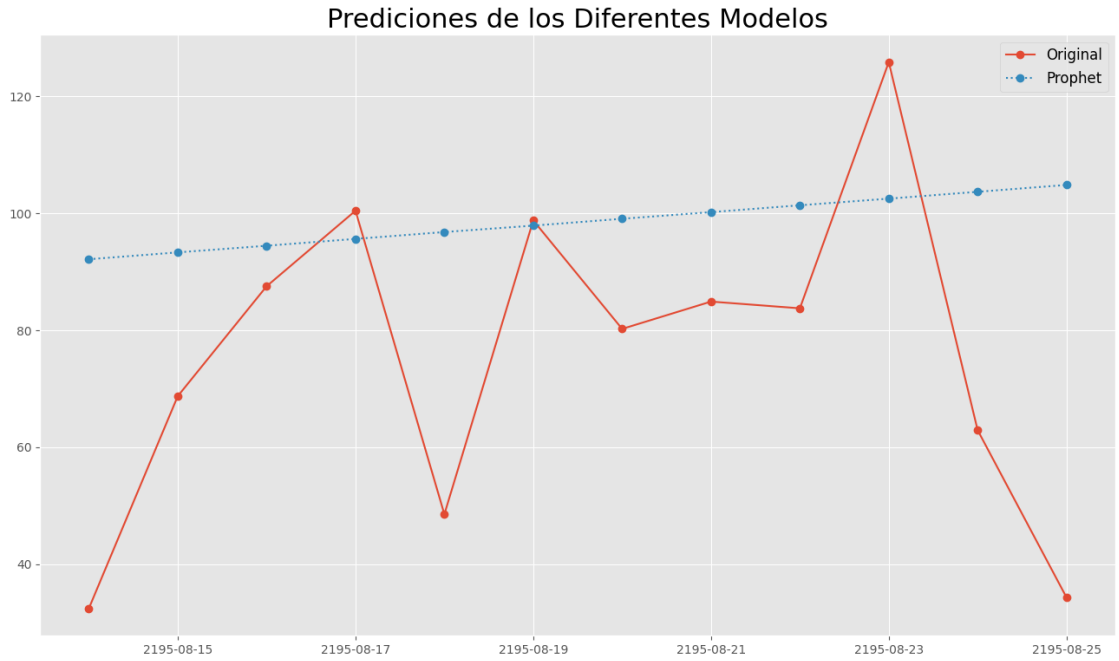
[ ]:
```
a=test_data[["Tiempo_Total","Prophet_Predictions"]]
fig = px.line(a, x=test_data.index, y=a.columns,template = "plotly_dark",
              title="Predicción con Modelo Prophet")
fig.show()
```

[ ]:
```
evaluacion_metrica(test_data["Tiempo_Total"],test_data["Prophet_Predictions"])
```

```
Evaluation metric results:-
MSE is : 1219.9653629266488
MAE is : 27.619987145454747
RMSE is : 34.92800256136398
MAPE is : 56.98639526952371
R2 is : -0.7227982433450808
```

[ ]:
```
# Grafica
plt.figure(figsize=(16,9))
plt.plot_date(test_data.index, test_data["Tiempo_Total"],label="Original",␣
 ↪linestyle="-")
#plt.plot_date(test_data.index, test_data["ARIMA_Predictions"],␣
 ↪label="Arima",linestyle="-.")
#plt.plot_date(test_data.index, test_data["LSTM_Predictions"],label="LSTM",␣
 ↪linestyle="--")
plt.plot_date(test_data.index, test_data["Prophet_Predictions"],␣
 ↪label="Prophet",linestyle=":")
plt.legend(fontsize=12)
```

```
plt.title("Prediciones de los Diferentes Modelos", fontsize=22)
plt.show();
```



[ ]:
```
#!pip install --tarjet=$nb_path skforecast
!pip install skforecast
```

```
Collecting skforecast
  Downloading skforecast-0.10.1-py3-none-any.whl (397 kB)
                             397.3/397.3
kB 6.1 MB/s eta 0:00:00
Requirement already satisfied: numpy<1.26,>=1.20 in
/usr/local/lib/python3.10/dist-packages (from skforecast) (1.23.5)
Requirement already satisfied: pandas<2.1,>=1.2 in
/usr/local/lib/python3.10/dist-packages (from skforecast) (1.5.3)
Collecting tqdm<4.66,>=4.57.0 (from skforecast)
  Downloading tqdm-4.65.2-py3-none-any.whl (77 kB)
                             77.1/77.1 kB
11.9 MB/s eta 0:00:00
Requirement already satisfied: scikit-learn<1.4,>=1.0 in
/usr/local/lib/python3.10/dist-packages (from skforecast) (1.2.2)
Collecting optuna<3.3,>=2.10.0 (from skforecast)
  Downloading optuna-3.2.0-py3-none-any.whl (390 kB)
                             390.6/390.6
kB 36.2 MB/s eta 0:00:00
Requirement already satisfied: joblib<1.4,>=1.1.0 in
```

```
/usr/local/lib/python3.10/dist-packages (from skforecast) (1.3.2)
Collecting alembic>=1.5.0 (from optuna<3.3,>=2.10.0->skforecast)
  Downloading alembic-1.12.1-py3-none-any.whl (226 kB)
                             226.8/226.8
kB 30.4 MB/s eta 0:00:00
Collecting cmaes>=0.9.1 (from optuna<3.3,>=2.10.0->skforecast)
  Downloading cmaes-0.10.0-py3-none-any.whl (29 kB)
Collecting colorlog (from optuna<3.3,>=2.10.0->skforecast)
  Downloading colorlog-6.7.0-py2.py3-none-any.whl (11 kB)
Requirement already satisfied: packaging>=20.0 in
/usr/local/lib/python3.10/dist-packages (from optuna<3.3,>=2.10.0->skforecast)
(23.2)
Requirement already satisfied: sqlalchemy>=1.3.0 in
/usr/local/lib/python3.10/dist-packages (from optuna<3.3,>=2.10.0->skforecast)
(2.0.23)
Requirement already satisfied: PyYAML in /usr/local/lib/python3.10/dist-packages
(from optuna<3.3,>=2.10.0->skforecast) (6.0.1)
Requirement already satisfied: python-dateutil>=2.8.1 in
/usr/local/lib/python3.10/dist-packages (from pandas<2.1,>=1.2->skforecast)
(2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-
packages (from pandas<2.1,>=1.2->skforecast) (2023.3.post1)
Requirement already satisfied: scipy>=1.3.2 in /usr/local/lib/python3.10/dist-
packages (from scikit-learn<1.4,>=1.0->skforecast) (1.11.3)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/usr/local/lib/python3.10/dist-packages (from scikit-
learn<1.4,>=1.0->skforecast) (3.2.0)
Collecting Mako (from alembic>=1.5.0->optuna<3.3,>=2.10.0->skforecast)
  Downloading Mako-1.2.4-py3-none-any.whl (78 kB)
                             78.7/78.7 kB
12.1 MB/s eta 0:00:00
Requirement already satisfied: typing-extensions>=4 in
/usr/local/lib/python3.10/dist-packages (from
alembic>=1.5.0->optuna<3.3,>=2.10.0->skforecast) (4.5.0)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-
packages (from python-dateutil>=2.8.1->pandas<2.1,>=1.2->skforecast) (1.16.0)
Requirement already satisfied: greenlet!=0.4.17 in
/usr/local/lib/python3.10/dist-packages (from
sqlalchemy>=1.3.0->optuna<3.3,>=2.10.0->skforecast) (3.0.1)
Requirement already satisfied: MarkupSafe>=0.9.2 in
/usr/local/lib/python3.10/dist-packages (from
Mako->alembic>=1.5.0->optuna<3.3,>=2.10.0->skforecast) (2.1.3)
Installing collected packages: tqdm, Mako, colorlog, cmaes, alembic, optuna,
skforecast
  Attempting uninstall: tqdm
    Found existing installation: tqdm 4.66.1
    Uninstalling tqdm-4.66.1:
```

```
      Successfully uninstalled tqdm-4.66.1
Successfully installed Mako-1.2.4 alembic-1.12.1 cmaes-0.10.0 colorlog-6.7.0
optuna-3.2.0 skforecast-0.10.1 tqdm-4.65.2
```

[ ]:
```python
# Modelo 2

import numpy as np
import pandas as pd

# Gráficos
# ==============================================================================
import matplotlib.pyplot as plt
plt.style.use('fivethirtyeight')
plt.rcParams['lines.linewidth'] = 1.5
plt.rcParams['font.size'] = 10

# Modelado y Forecasting
# ==============================================================================
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Lasso
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from sklearn.preprocessing import StandardScaler

from skforecast.ForecasterAutoreg import ForecasterAutoreg
from skforecast.ForecasterAutoregCustom import ForecasterAutoregCustom
from skforecast.ForecasterAutoregDirect import ForecasterAutoregDirect
from skforecast.model_selection import grid_search_forecaster
from skforecast.model_selection import backtesting_forecaster
from skforecast.utils import save_forecaster
from skforecast.utils import load_forecaster

# Configuración warnings
# ==============================================================================
import warnings
# warnings.filterwarnings('ignore')
```

[ ]:
```python
datos = df
```

[ ]:
```python
url = 'https://raw.githubusercontent.com/JoaquinAmatRodrigo/skforecast/master/
 ↪data/h2o_exog.csv'
datos1 = pd.read_csv(url, sep=',')
```

[ ]:
```python
datos
```

[ ]:

	FECHA_LLEGADA	FECHA_TRIAGE	\
1	2023-01-01 01:20:23.853	2023-01-01 01:28:01.847	
2	2023-01-01 01:29:46.050	2023-01-01 01:48:03.070	
3	2023-01-01 03:15:35.623	2023-01-01 03:23:01.990	
4	2023-01-01 05:54:53.563	2023-01-01 06:00:07.943	
5	2023-01-01 06:37:27.237	2023-01-01 07:52:31.687	
…	…	…	
82412	2023-09-18 04:44:41.970	2023-09-18 04:53:22.553	
82413	2023-09-18 06:17:00.573	2023-09-18 06:28:43.040	
82414	2023-09-18 06:21:37.273	2023-09-18 07:00:57.420	
82415	2023-09-18 06:25:33.483	2023-09-18 06:42:02.883	
82416	2023-09-18 07:14:58.180	2023-09-18 07:30:50.643	

	FECHA_INGRESO	FECHA_ATENCION	\
1	2023-01-01 01:29:41.210	2023-01-01 02:00:07.590	
2	2023-01-01 01:49:40.973	2023-01-01 02:02:53.663	
3	2023-01-01 03:23:39.793	2023-01-01 03:30:21.233	
4	2023-01-01 06:02:07.320	2023-01-01 06:26:17.050	
5	2023-01-01 07:52:37.717	2023-01-01 09:31:15.597	
…	…	…	
82412	2023-09-18 05:05:51.423	2023-09-18 06:09:35.867	
82413	2023-09-18 06:35:38.213	2023-09-18 07:40:45.957	
82414	2023-09-18 07:16:45.907	2023-09-18 08:27:27.337	
82415	2023-09-18 06:51:35.970	2023-09-18 07:28:28.290	
82416	2023-09-18 07:34:08.370	2023-09-18 07:49:18.440	

	TIEMPO_DGTURNO_A_TRIAGE	TIEMPO_TRIAGE_A_INGRESO	\
1	0:07:38	0:01:40	
2	0:18:17	0:01:37	
3	0:07:26	0:00:38	
4	0:05:14	0:02:00	
5	1:15:04	0:00:06	
…	…	…	
82412	0:08:41	0:12:29	
82413	0:11:43	0:06:55	
82414	0:39:20	0:15:48	
82415	0:16:29	0:09:33	
82416	0:15:52	0:03:18	

	TIEMPO_INGRESO_A_CONSULTA	TIEMPO_TOTAL	Tiempo_Minutos_Total	\
1	0:30:26	0:39:44	39.73	
2	0:13:13	0:33:07	33.12	
3	0:06:42	0:14:46	14.77	
4	0:24:10	0:31:24	31.40	
5	1:38:38	2:53:48	173.80	
…	…	…	…	
82412	1:03:44	1:24:54	84.90	

```
82413                  1:05:07      1:23:45                83.75
82414                  1:10:42      2:05:50               125.83
82415                  0:36:53      1:02:55                62.92
82416                  0:15:10      0:34:20                34.33

      CENTRO_ATENCION  …   AÑO MES DIA_SEMANA HOUR  Turnos  \
1                  TN  …  2023   1    DOMINGO    1   NOCHE
2                  ME  …  2023   1    DOMINGO    1   NOCHE
3                  UC  …  2023   1    DOMINGO    3   NOCHE
4                  UC  …  2023   1    DOMINGO    5   NOCHE
5                  TN  …  2023   1    DOMINGO    6   NOCHE
…                 …  …   …   …  ..        …  …     …
82412              ME  …  2023   9      LUNES    4   NOCHE
82413              ME  …  2023   9      LUNES    6   NOCHE
82414              ME  …  2023   9      LUNES    6   NOCHE
82415              UB  …  2023   9      LUNES    6   NOCHE
82416              TN  …  2023   9      LUNES    7  MAÑANA

                          TIME DIA Tiempo_Total      Month       fecha
1      2023-01-01 01:20:23.853   1        39.73 2023-01-01  2023-01-01
2      2023-01-01 01:29:46.050   1        33.12 2023-01-01  2023-01-01
3      2023-01-01 03:15:35.623   1        14.77 2023-01-01  2023-01-01
4      2023-01-01 05:54:53.563   1        31.40 2023-01-01  2023-01-01
5      2023-01-01 06:37:27.237   1       173.80 2023-01-01  2023-01-01
…                         …  ..          …         …          …
82412  2023-09-18 04:44:41.970  18        84.90 2023-09-01  2023-09-01
82413  2023-09-18 06:17:00.573  18        83.75 2023-09-01  2023-09-01
82414  2023-09-18 06:21:37.273  18       125.83 2023-09-01  2023-09-01
82415  2023-09-18 06:25:33.483  18        62.92 2023-09-01  2023-09-01
82416  2023-09-18 07:14:58.180  18        34.33 2023-09-01  2023-09-01

[82416 rows x 28 columns]
```

[ ]:
```
datos1
```

[ ]:
```
# agrupar datos
datos2 = datos.groupby('Month')['Tiempo_Total'].mean()
datos2
```

[ ]:
```
Month
2023-01-01    107.490400
2023-02-01    104.363828
2023-03-01    112.497437
2023-04-01    107.439636
2023-05-01    116.109004
2023-06-01    108.183079
2023-07-01    100.240377
```

```
2023-08-01    106.878000
2023-09-01    111.367566
Name: Tiempo_Total, dtype: float64
```

[ ]:
```
# Preparación del dato
# ==============================================================================
datos['fecha'] = df['Month'].dt.strftime('%Y-%m-01')
datos = datos.set_index('fecha')
datos = datos.rename(columns={'x': 'Tiempo_Total'})
datos = datos.asfreq('MS')
datos = datos.sort_index()
datos.head()
```

[ ]:
```
# Preparación del dato
# ==============================================================================
datos1['fecha'] = pd.to_datetime(datos1['fecha'], format='%Y-%m-%d')
datos1 = datos1.set_index('fecha')
datos1 = datos1.rename(columns={'x': 'y'})
datos1 = datos1.asfreq('MS')
datos1 = datos1.sort_index()
datos1.head()
```

[ ]:
```
datos2
```

[ ]:
```
Month
2023-01-01    107.490400
2023-02-01    104.363828
2023-03-01    112.497437
2023-04-01    107.439636
2023-05-01    116.109004
2023-06-01    108.183079
2023-07-01    100.240377
2023-08-01    106.878000
2023-09-01    111.367566
Name: Tiempo_Total, dtype: float64
```

[ ]:
```
print(f'Número de filas con missing values: {datos.isnull().any(axis=1).
  ↪mean()}')
```

```
Número de filas con missing values: 0.0
```

[ ]:
```
# Verificar que un índice temporal está completo
# ==============================================================================
(datos.index == pd.date_range(
                    start = datos.index.min(),
                    end   = datos.index.max(),
                    freq  = datos.index.freq)
```

```
).all()
```

[ ]:
```
datos_train
```

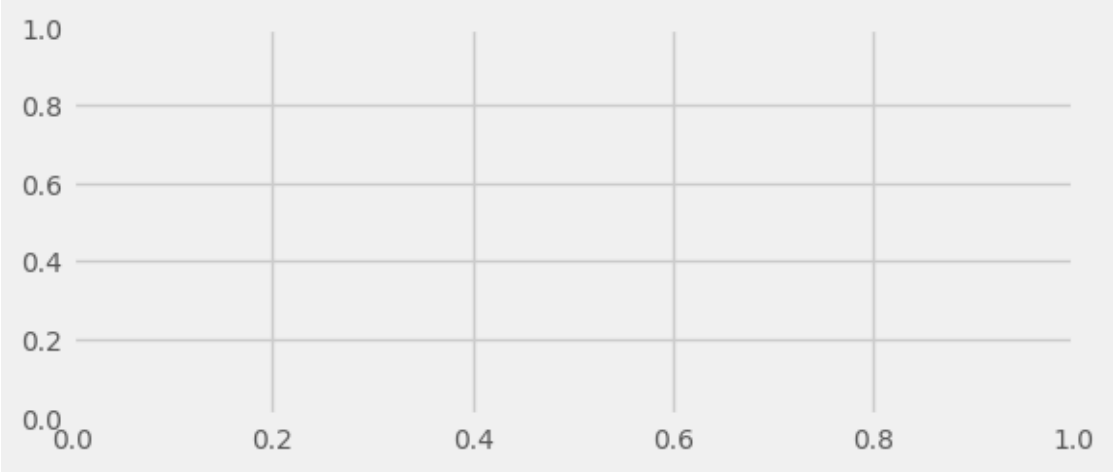
[ ]:
```
datos_test
```

[ ]:
```
steps = 3
datos_train = datos[:-steps]
datos_test  = datos[-steps:]

print(f"Fechas train : {datos_train.index.min()} --- {datos_train.index.max()} ␣
 ↪(n={len(datos_train)})")
print(f"Fechas test  : {datos_test.index.min()} --- {datos_test.index.max()} ␣
 ↪(n={len(datos_test)})")

fig, ax = plt.subplots(figsize=(6, 2.5))
datos_train['y'].plot(ax=ax, label='train')
datos_test['y'].plot(ax=ax, label='test')
ax.legend();
```

```
Fechas train : 2023-01-01 00:00:00 --- 2023-09-01 00:00:00  (n=82413)
Fechas test  : 2023-09-01 00:00:00 --- 2023-09-01 00:00:00  (n=3)
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
/usr/local/lib/python3.10/dist-packages/pandas/core/indexes/base.py in␣
 ↪get_loc(self, key, method, tolerance)
   3801             try:
-> 3802                 return self._engine.get_loc(casted_key)
   3803             except KeyError as err:

/usr/local/lib/python3.10/dist-packages/pandas/_libs/index.pyx in pandas._libs.
 ↪index.IndexEngine.get_loc()

/usr/local/lib/python3.10/dist-packages/pandas/_libs/index.pyx in pandas._libs.
 ↪index.IndexEngine.get_loc()

pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.
 ↪PyObjectHashTable.get_item()

pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.
 ↪PyObjectHashTable.get_item()

KeyError: 'y'

The above exception was the direct cause of the following exception:
```

```
KeyError                                  Traceback (most recent call last)
<ipython-input-78-6fe142492622> in <cell line: 9>()
      7
      8 fig, ax = plt.subplots(figsize=(6, 2.5))
----> 9 datos_train['y'].plot(ax=ax, label='train')
     10 datos_test['y'].plot(ax=ax, label='test')
     11 ax.legend();

/usr/local/lib/python3.10/dist-packages/pandas/core/frame.py in␣
 ↪__getitem__(self, key)
   3805             if self.columns.nlevels > 1:
   3806                 return self._getitem_multilevel(key)
-> 3807             indexer = self.columns.get_loc(key)
   3808             if is_integer(indexer):
   3809                 indexer = [indexer]

/usr/local/lib/python3.10/dist-packages/pandas/core/indexes/base.py in␣
 ↪get_loc(self, key, method, tolerance)
   3802                 return self._engine.get_loc(casted_key)
   3803             except KeyError as err:
-> 3804                 raise KeyError(key) from err
   3805             except TypeError:
   3806                 # If we have a listlike key, _check_indexing_error will␣
 ↪raise

KeyError: 'y'
```



arios-modelos-14nov-version9-final

November 23, 2023

[85]:
```
#!pip install pmdarima
#!pip install skforecast
#!pip install --upgrade statsmodels pmdarima
#!pip install train_feature
!pip install ForecasterAutoreg
```

```
ERROR: Could not find a version that satisfies the requirement
ForecasterAutoreg (from versions: none)
ERROR: No matching distribution found for ForecasterAutoreg
```

[3]:
```
# Manipulación y tratamiento de Datos
import numpy as np
import pandas as pd
import os

# Visualización de datos
import plotly.express as px
import matplotlib.pyplot as plt
%matplotlib inline
plt.style.use('ggplot')

# Modelación Arima
from statsmodels.tsa.statespace.sarimax import SARIMAX
from statsmodels.graphics.tsaplots import plot_acf,plot_pacf
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.tsa.stattools import adfuller

# Modelo Auto-Arima
from pmdarima import auto_arima
import pmdarima as pm

# Métrica de Evaluación
from sklearn.metrics import mean_squared_error
from statsmodels.tools.eval_measures import rmse
from sklearn import metrics
```

```
# No presentar advertencias
import warnings
warnings.filterwarnings("ignore")
```

[40]:
```
# ruta de archivos

#files = os.listdir("c:\\archivos\proyecto")
#os.chdir(r'C:\archivos\proyecto')

csv_path = 'urgencias final_nov.txt'

# Intentar leer el archivo con diferentes codificaciones y manejo de errores
encodings_to_try = ['utf-8', 'latin-1', 'ISO-8859-1']

for encoding in encodings_to_try:
    try:
        df_real = pd.read_csv(csv_path, sep=";", header=None, encoding=encoding)
        # Si se llega a este punto, la lectura fue exitosa, así que puedes␣
 ↪salir del bucle
        break
    except UnicodeDecodeError:
        print(f"Error al leer con la codificación {encoding}. Intentando con␣
 ↪otra.")
```

```
Error al leer con la codificación utf-8. Intentando con otra.
```

[41]:
```
headers =␣
 ↪["FECHA_LLEGADA","TIEMPO_TOTAL_FINAL","CENTRO_ATENCION","CLASIFICACION_TRIAGE","PACIENTE_ED
print("headers\n", headers)
df_real.columns = headers
df_real = df_real.drop(0)
```

```
headers
 ['FECHA_LLEGADA', 'TIEMPO_TOTAL_FINAL', 'CENTRO_ATENCION',
'CLASIFICACION_TRIAGE', 'PACIENTE_EDAD', 'PACIENTE_#_DOCUMENTO', 'EDAD_RANGO',
'NOMBRE_ENTIDAD', 'SEXO', 'Month']
```

[42]:
```
# convertir y arreglar datos
df_real['Month'] = pd.to_datetime(df_real['Month'])

# crear indice de frecuencia
df_real['Month'] = df_real['Month'].dt.to_period('M')
# nuevo_df.index.freq = 'MS'

# indexar
df_real = df_real.set_index("Month")
```

```
# numerico
df_real["TIEMPO_TOTAL_FINAL"] = pd.
 ↪to_numeric(df_real["TIEMPO_TOTAL_FINAL"],errors='coerce')

# Calcular la mediana y arreglar datos

#df["TIEMPO_TOTAL_FINAL"] = pd.
 ↪to_numeric(df["TIEMPO_TOTAL_FINAL"],errors='coerce')
median = df_real['TIEMPO_TOTAL_FINAL'].median()

# Corregir valores atípicos
df_real.loc[df_real['TIEMPO_TOTAL_FINAL'] > 420, 'TIEMPO_TOTAL_FINAL'] = median
df_real.loc[df_real['TIEMPO_TOTAL_FINAL'] < 0, 'TIEMPO_TOTAL_FINAL'] = median
```

[43]:
```
df_real['FECHA_LLEGADA'] = pd.to_datetime(df_real['FECHA_LLEGADA'])
df_real['DIA_SEMANA'] = df_real['FECHA_LLEGADA'].dt.dayofweek
df_real['ANUAL'] = df_real['FECHA_LLEGADA'].dt.year
```

[44]:
```
# Filtrar a partir del año 2021

anho_inicio = 2022
data1 = df_real[df_real['ANUAL'] >= anho_inicio]
#df_hearth.to_csv('ind_urgencias_final_2023.txt', sep=';', index=False)
```

[45]:
```
data1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
PeriodIndex: 224412 entries, 2023-07 to 2023-04
Freq: M
Data columns (total 11 columns):
 #   Column                Non-Null Count   Dtype
---  ------                --------------   -----
 0   FECHA_LLEGADA         224412 non-null  datetime64[ns]
 1   TIEMPO_TOTAL_FINAL    224410 non-null  float64
 2   CENTRO_ATENCION       224412 non-null  object
 3   CLASIFICACION_TRIAGE  224412 non-null  object
 4   PACIENTE_EDAD         224412 non-null  object
 5   PACIENTE_#_DOCUMENTO  224412 non-null  object
 6   EDAD_RANGO            224412 non-null  object
 7   NOMBRE_ENTIDAD        224412 non-null  object
 8   SEXO                  224412 non-null  object
 9   DIA_SEMANA            224412 non-null  int64
 10  ANUAL                 224412 non-null  int64
dtypes: datetime64[ns](1), float64(1), int64(2), object(7)
memory usage: 20.5+ MB
```

[46]:
```
df = data1
```

[47]:
```
data = data1
```

[48]:
```
data
```

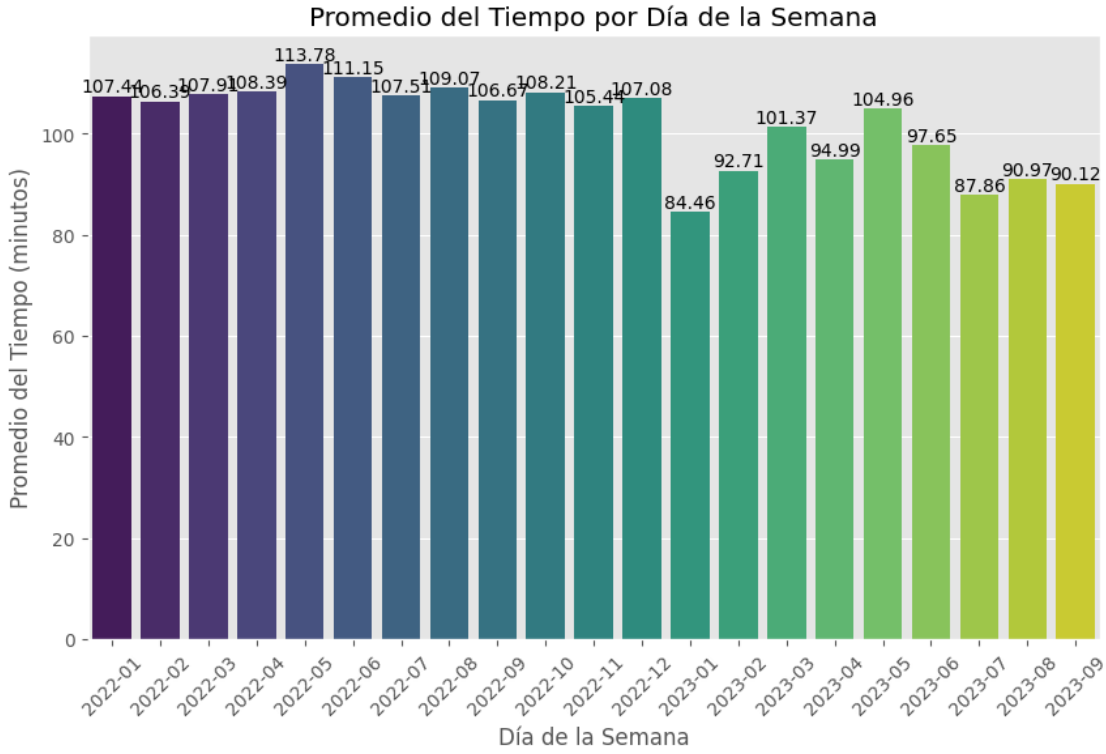
[48]:

```
              FECHA_LLEGADA  TIEMPO_TOTAL_FINAL  \
Month
2023-07 2023-07-11 21:19:10               15.00
2023-02 2023-02-24 05:31:45               16.00
2023-08 2023-08-07 23:16:10               16.00
2023-09 2023-09-13 01:56:53               17.00
2023-06 2023-06-29 02:42:38               18.00
…                       …                   …
2022-07 2022-07-21 10:27:00               84.33
2022-12 2022-12-15 14:50:00               84.33
2023-04 2023-04-10 18:02:25               84.33
2023-07 2023-07-01 16:37:54               84.33
2023-04 2023-04-13 17:48:28               84.33

                              CENTRO_ATENCION  \
Month
2023-07                                   VC
2023-02                                   VB
2023-08                                   VC
2023-09                                   UC
2023-06                                   UC
…                                          …
2022-07                 ME - HOSPITAL MEISSEN
2022-12  UC - CENTRO DE SALUD SANTA LIBRADA I
2023-04                                   ME
2023-07                                   TN
2023-04                                   ME

                                 CLASIFICACION_TRIAGE PACIENTE_EDAD  \
Month
2023-07                                             3            24
2023-02                                             3            69
2023-08                                             3            50
2023-09                                             3            79
2023-06                                             3            72
…                                                   …             …
2022-07  3 - TRIAGE III - NO DEBE SUPERAR LAS 3 HORAS     58 ANO(S)
2022-12  3 - TRIAGE III - NO DEBE SUPERAR LAS 3 HORAS     54 ANO(S)
2023-04                                             3             4
2023-07                                             3             3
2023-04                                             3            17
```

```
        PACIENTE_#_DOCUMENTO        EDAD_RANGO             NOMBRE_ENTIDAD  \
Month
2023-07            1024595942          JUVENTUD                      EPSC34
2023-02               4450531      ADULTO MAYOR                      EPSC34
2023-08               5267156            ADULTO                      EPSC34
2023-09               5667630      ADULTO MAYOR                      EPSC34
2023-06               3014617      ADULTO MAYOR                      EPSS05
…                         …                 …                           …
2022-07              39532594           ADULTEZ  CAPITAL SALUD EPS-S S.A.S
2022-12              39723106           ADULTEZ  CAPITAL SALUD EPS-S S.A.S
2023-04            1033822858  PRIMERA INFANCIA                      EPSC34
2023-07            1243858533  PRIMERA INFANCIA                      EPSC34
2023-04            1010176536       ADOLECENCIA                      EPSS41

              SEXO  DIA_SEMANA  ANUAL
Month
2023-07  MASCULINO           1   2023
2023-02  MASCULINO           4   2023
2023-08   FEMENINO           0   2023
2023-09  MASCULINO           2   2023
2023-06  MASCULINO           3   2023
…              …           …     …
2022-07   FEMENINO           3   2022
2022-12   FEMENINO           3   2022
2023-04  MASCULINO           0   2023
2023-07   FEMENINO           5   2023
2023-04   FEMENINO           3   2023

[224412 rows x 11 columns]
```

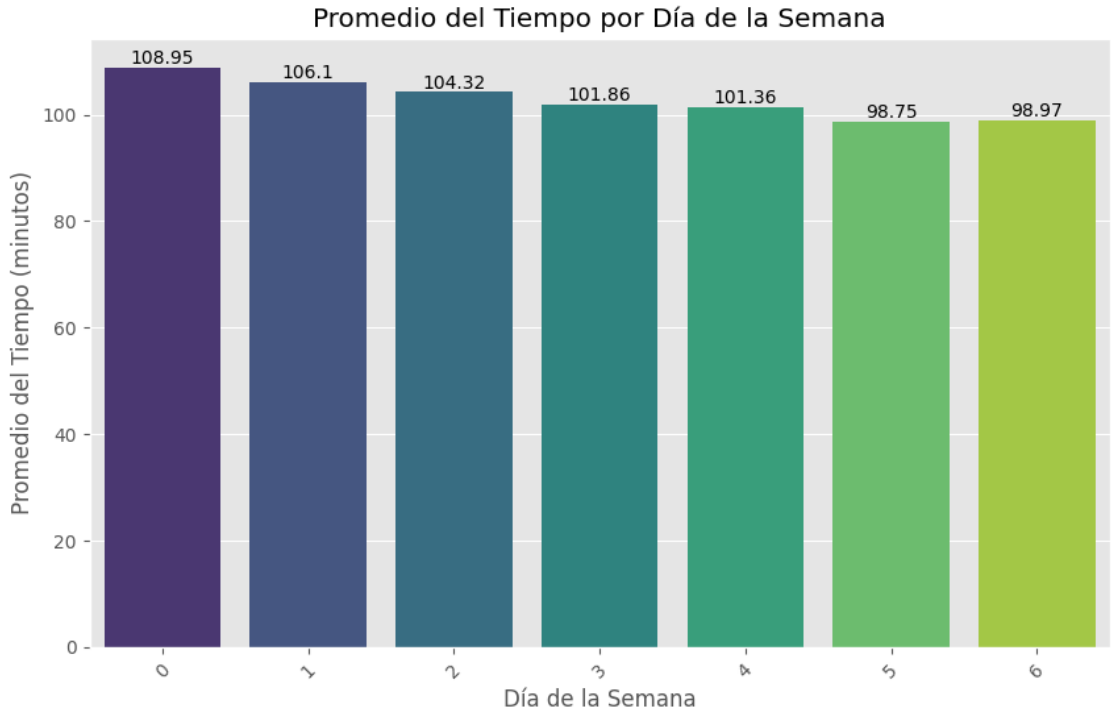
[51]:
```
import seaborn as sns

# Agrupar por día de la semana y calcular el promedio del Tiempo_total_minutos␣
 ↪en cada grupo

promedio_tiempo_por_grupo =data1.groupby(['Month'])['TIEMPO_TOTAL_FINAL'].
 ↪mean().reset_index()

# Renombrar la columna del promedio
promedio_tiempo_por_grupo = promedio_tiempo_por_grupo.
 ↪rename(columns={'TIEMPO_TOTAL_FINAL': 'Promedio_Tiempo'})

plt.figure(figsize=(10, 6))
ax = sns.barplot(data=promedio_tiempo_por_grupo, x='Month',␣
 ↪y='Promedio_Tiempo', palette='viridis')
plt.xlabel('Día de la Semana')
```

```
plt.ylabel('Promedio del Tiempo (minutos)')
plt.title('Promedio del Tiempo por Día de la Semana')
plt.xticks(rotation=45)

# Agregar etiquetas en las barras
for index, row in promedio_tiempo_por_grupo.iterrows():
    ax.annotate(str(round(row['Promedio_Tiempo'], 2)), (index,␣
 ↪row['Promedio_Tiempo']), ha='center', va='bottom')
plt.show()
```



[52]:
```
import seaborn as sns

# Agrupar por día de la semana y calcular el promedio del Tiempo_total_minutos␣
 ↪en cada grupo

promedio_tiempo_por_grupo =data1.groupby(['DIA_SEMANA'])['TIEMPO_TOTAL_FINAL'].
 ↪mean().reset_index()

# Renombrar la columna del promedio
promedio_tiempo_por_grupo = promedio_tiempo_por_grupo.
 ↪rename(columns={'TIEMPO_TOTAL_FINAL': 'Promedio_Tiempo'})
```

```
plt.figure(figsize=(10, 6))
ax = sns.barplot(data=promedio_tiempo_por_grupo, x='DIA_SEMANA',␣
 ↪y='Promedio_Tiempo', palette='viridis')
plt.xlabel('Día de la Semana')
plt.ylabel('Promedio del Tiempo (minutos)')
plt.title('Promedio del Tiempo por Día de la Semana')
plt.xticks(rotation=45)

# Agregar etiquetas en las barras
for index, row in promedio_tiempo_por_grupo.iterrows():
    ax.annotate(str(round(row['Promedio_Tiempo'], 2)), (index,␣
 ↪row['Promedio_Tiempo']), ha='center', va='bottom')
plt.show()
```



[53]:
```
# Funcion para evaluar

def evaluacion_metrica(y_true, y_pred):

    def mean_absolute_percentage_error(y_true, y_pred):
        y_true, y_pred = np.array(y_true), np.array(y_pred)
        return np.mean(np.abs((y_true - y_pred) / y_true)) * 100
    print('Evaluation metric results:-')
    print(f'MSE is : {metrics.mean_squared_error(y_true, y_pred)}')
```

```
    print(f'MAE is : {metrics.mean_absolute_error(y_true, y_pred)}')
    print(f'RMSE is : {np.sqrt(metrics.mean_squared_error(y_true, y_pred))}')
    print(f'MAPE is : {mean_absolute_percentage_error(y_true, y_pred)}')
    print(f'R2 is : {metrics.r2_score(y_true, y_pred)}',end='\n\n')
```

[60]:
```
# Agrupar de acuerdo a la grafica

# agrupamiento de mes
nuevo_df = df.groupby('Month')['TIEMPO_TOTAL_FINAL'].mean().reset_index()

df6 = nuevo_df.copy()

# Convierte el índice Period a cadena
#df6.index = df6.index.astype(str)
df6['Month1'] = df6['Month'].astype(str)

# Crea el gráfico de líneas
fig = px.line(df6, x='Month1', y='TIEMPO_TOTAL_FINAL', template='plotly_dark',␣
 ↪title='Total minutos')

# Muestra el gráfico
fig.show()
```

[62]:
```
# División de para entrenamiento y prueba
train_data = df6[:len(df6)-5]
test_data = df6[len(df6)-5:]
test=test_data.copy()

df6=df6.reset_index()

df_fb=df6.rename(columns={"Month":"ds", "TIEMPO_TOTAL_FINAL":"y"} )

train_data_pr = df_fb.iloc[:len(df6)-5]
test_data_pr = df_fb.iloc[len(df6)-5:]
```

[63]:
```
# 1. Modelo : Prophet Forecast

from prophet import Prophet

train_data_pr['ds'] = train_data_pr['ds'].dt.to_timestamp()
m = Prophet()
m.fit(train_data_pr)

future = m.make_future_dataframe(periods=5,freq='MS')
prophet_pred = m.predict(future)
# prophet_pred.tail()    ---  ver detalle
```

```
# asignar a prophet_pred
prophet_pred = pd.DataFrame({"Date" : prophet_pred[-5:]['ds'], "Pred" :␣
 ↪prophet_pred[-5:]["yhat"]})
prophet_pred = prophet_pred.set_index("Date")
prophet_pred.index.freq = "MS"
prophet_pred
test_data["Prophet_Predictions"] = prophet_pred['Pred'].values
test_data.head()

# Grafica test_data
a=test_data[["TIEMPO_TOTAL_FINAL","Prophet_Predictions"]]
fig = px.line(a, x=test_data.index, y=a.columns,template = "plotly_dark",
              title="Predicción con Modelo Prophet")
fig.show()

# evaluacion de Prophet_Predictions
evaluacion_metrica(test_data["TIEMPO_TOTAL_FINAL"],test_data["Prophet_Predictions"])
```

```
INFO:prophet:Disabling yearly seasonality. Run prophet with
yearly_seasonality=True to override this.
INFO:prophet:Disabling weekly seasonality. Run prophet with
weekly_seasonality=True to override this.
INFO:prophet:Disabling daily seasonality. Run prophet with
daily_seasonality=True to override this.
INFO:prophet:n_changepoints greater than number of observations. Using 11.
DEBUG:cmdstanpy:input tempfile: /tmp/tmp6jfyo2yv/so_ou8rj.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmp6jfyo2yv/h3sk1zyi.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.10/dist-
packages/prophet/stan_model/prophet_model.bin', 'random', 'seed=62709', 'data',
'file=/tmp/tmp6jfyo2yv/so_ou8rj.json', 'init=/tmp/tmp6jfyo2yv/h3sk1zyi.json',
'output',
'file=/tmp/tmp6jfyo2yv/prophet_modelkyr1dxed/prophet_model-20231114195133.csv',
'method=optimize', 'algorithm=newton', 'iter=10000']
19:51:33 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
19:51:33 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
```

```
Evaluation metric results:-
MSE is : 26.468700671305225
MAE is : 4.183549665873139
RMSE is : 5.144774112758035
MAPE is : 4.32849893927382
R2 is : 0.3213828643966856
```

[19]:
```
# 2.  Modelo : ARIMA

#train_data['Month'] = train_data['Month'].dt.to_timestamp()
train_data['Month_numeric'] = train_data['Month'].dt.year * 12 +␣
 ↪train_data['Month'].dt.month

# Luego, utiliza 'Month_numeric' como la serie temporal en auto_arima
modelo_auto = auto_arima(train_data['Month_numeric'], start_p=0, d=1, start_q=0,
                         max_p=4, max_d=2, max_q=4, start_P=0,
                         D=1, start_Q=0, max_P=2, max_D=1,
                         max_Q=2, m=9, seasonal=True,
                         suppress_warnings=True, stepwise=True,
                         random_state=20, n_fits=50)
print(modelo_auto)

arima_model = SARIMAX(train_data["TIEMPO_TOTAL_FINAL"], order = (0,1,0),␣
 ↪seasonal_order = (0,1,0,9))
arima_result = arima_model.fit()
arima_result.summary()
```

```
ARIMA(0,1,0)(0,1,0)[9] intercept
```

[19]:

Dep. Variable:	TIEMPO_TOTAL_FINAL	**No. Observations:**	12
Model:	SARIMAX(0, 1, 0)x(0, 1, 0, 12)	**Log Likelihood**	0.000
Date:	Tue, 14 Nov 2023	**AIC**	2.000
Time:	19:24:55	**BIC**	nan
Sample:	0	**HQIC**	nan
	- 12		
Covariance Type:	opg		

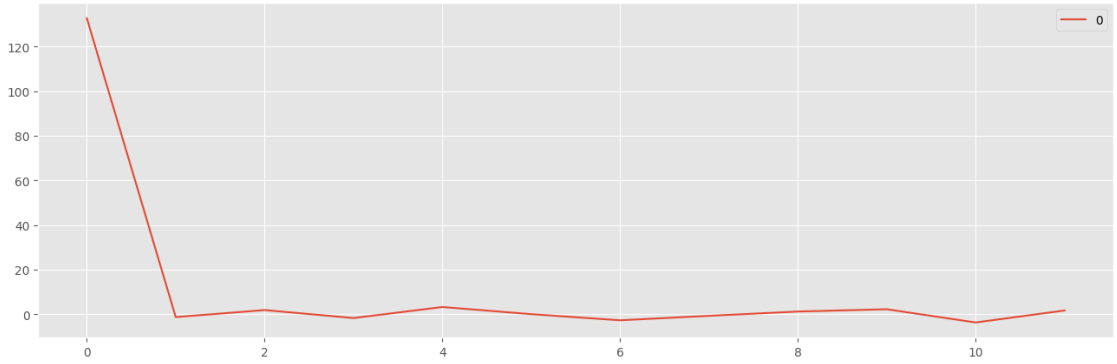
	coef	**std err**	**z**	**P> |z|**	**[0.025**	**0.975]**
sigma2	1e-10	0	inf	0.000	1e-10	1e-10

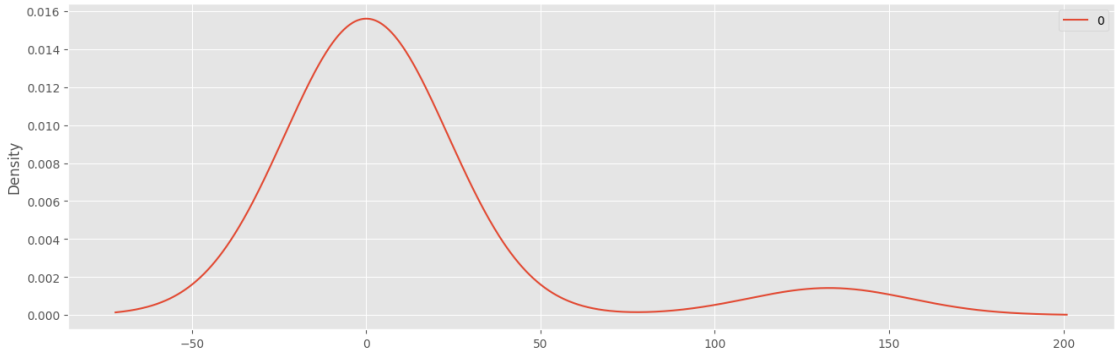
Ljung-Box (L1) (Q):	nan	**Jarque-Bera (JB):**	nan
Prob(Q):	nan	**Prob(JB):**	nan
Heteroskedasticity (H):	nan	**Skew:**	nan
Prob(H) (two-sided):	nan	**Kurtosis:**	nan

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
[2] Covariance matrix is singular or near-singular, with condition number inf. Standard errors may be unstable.

[20]:
```
# line plot of residual errors
residuals = pd.DataFrame(arima_result.resid)
residuals.plot(figsize = (16,5));
plt.show();

# kernel density plot of residual errors
```

```
residuals.plot(kind='kde', figsize = (16,5))
plt.show()
print(residuals.describe())
```





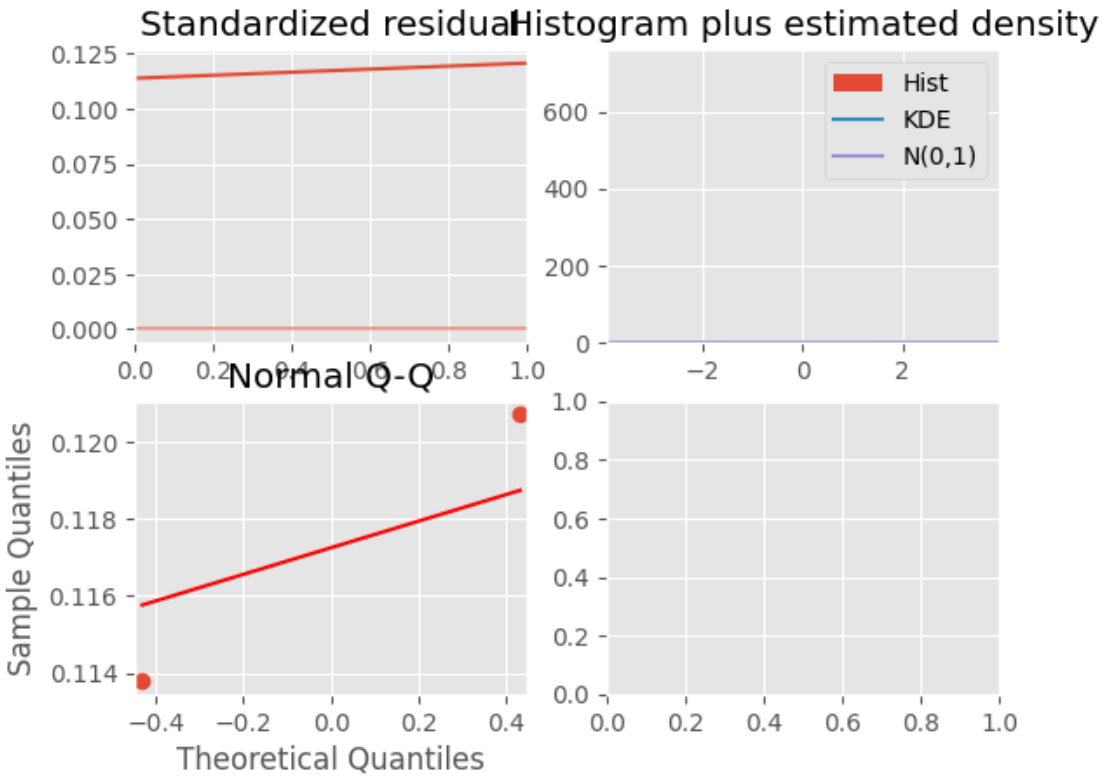

```
                0
count   12.000000
mean    11.101323
std     38.363697
min     -3.668906
25%     -1.356471
50%      0.651822
75%      1.987594
max    132.736505
```

[33]:
```
# Obtener los diagnósticos del modelo ARIMA ajustado por auto_arima
arima_diagnostics = modelo_auto.plot_diagnostics(lags=9)  # Ajusta el número de␣
↪lags según tus necesidades
plt.show()
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
```

```
<ipython-input-33-b8a04f291845> in <cell line: 2>()
      1 # Obtener los diagnósticos del modelo ARIMA ajustado por auto_arima
----> 2 arima_diagnostics = modelo_auto.plot_diagnostics(lags=9)  # Ajusta el␣
 ↪número de lags según tus necesidades
      3 plt.show()

/usr/local/lib/python3.10/dist-packages/pmdarima/utils/metaestimators.py in␣
 ↪<lambda>(*args, **kwargs)
     51
     52         # lambda, but not partial, allows help() to work with␣
 ↪update_wrapper
---> 53         out = (lambda *args, **kwargs: self.fn(obj, *args, **kwargs))
     54         # update the docstring of the returned function
     55         update_wrapper(out, self.fn)

/usr/local/lib/python3.10/dist-packages/pmdarima/arima/arima.py in␣
 ↪plot_diagnostics(self, variable, lags, fig, figsize)
   1395         ax = fig.add_subplot(224)
   1396         from statsmodels.graphics import tsaplots
-> 1397         tsaplots.plot_acf(resid, ax=ax, lags=lags)
   1398         ax.set_title('Correlogram')
   1399

/usr/local/lib/python3.10/dist-packages/pandas/util/_decorators.py in␣
 ↪wrapper(*args, **kwargs)
    209                     else:
    210                         kwargs[new_arg_name] = new_arg_value
--> 211             return func(*args, **kwargs)
    212
    213         return cast(F, wrapper)

/usr/local/lib/python3.10/dist-packages/statsmodels/graphics/tsaplots.py in␣
 ↪plot_acf(x, ax, lags, alpha, use_vlines, adjusted, fft, missing, title, zero,␣
 ↪auto_ylims, bartlett_confint, vlines_kwargs, **kwargs)
    225         acf_x, confint = acf_x[:2]
    226
--> 227     _plot_corr(
    228         ax,
    229         title,

/usr/local/lib/python3.10/dist-packages/statsmodels/graphics/tsaplots.py in␣
 ↪_plot_corr(ax, title, acf_x, confint, lags, irregular, use_vlines,␣
 ↪vlines_kwargs, auto_ylims, **kwargs)
     47
     48     if use_vlines:
---> 49         ax.vlines(lags, [0], acf_x, **vlines_kwargs)
     50         ax.axhline(**kwargs)
     51
```

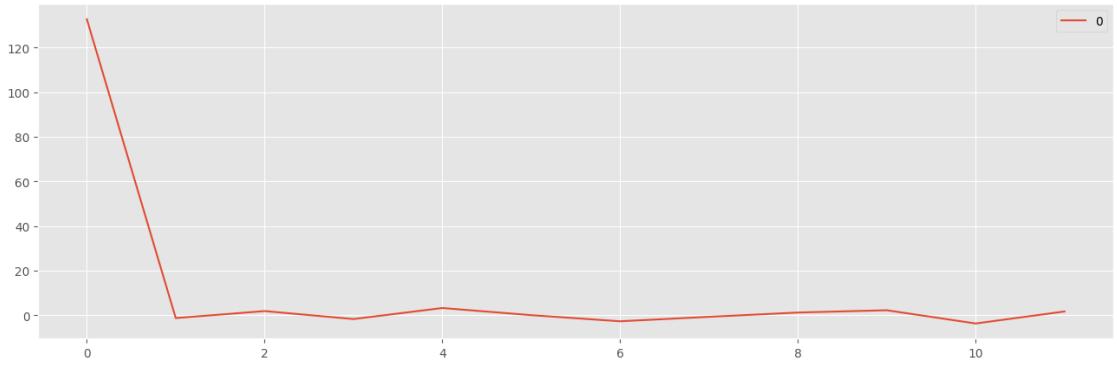
```
/usr/local/lib/python3.10/dist-packages/matplotlib/__init__.py in inner(ax,␣
 ↪data, *args, **kwargs)
   1440     def inner(ax, *args, data=None, **kwargs):
   1441         if data is None:
-> 1442             return func(ax, *map(sanitize_sequence, args), **kwargs)
   1443
   1444         bound = new_sig.bind(ax, *args, **kwargs)

/usr/local/lib/python3.10/dist-packages/matplotlib/axes/_axes.py in vlines(self ␣
 ↪x, ymin, ymax, colors, linestyles, label, **kwargs)
   1170         masked_verts[:, 0, 1] = ymin
   1171         masked_verts[:, 1, 0] = x
-> 1172         masked_verts[:, 1, 1] = ymax
   1173
   1174         lines = mcoll.LineCollection(masked_verts, colors=colors,

/usr/local/lib/python3.10/dist-packages/numpy/ma/core.py in __setitem__(self,␣
 ↪indx, value)
   3375         if _mask is nomask:
   3376             # Set the data, then the mask
-> 3377             _data[indx] = dval
   3378             if mval is not nomask:
   3379                 _mask = self._mask = make_mask_none(self.shape, _dtype)

ValueError: could not broadcast input array from shape (2,) into shape (10,)
```

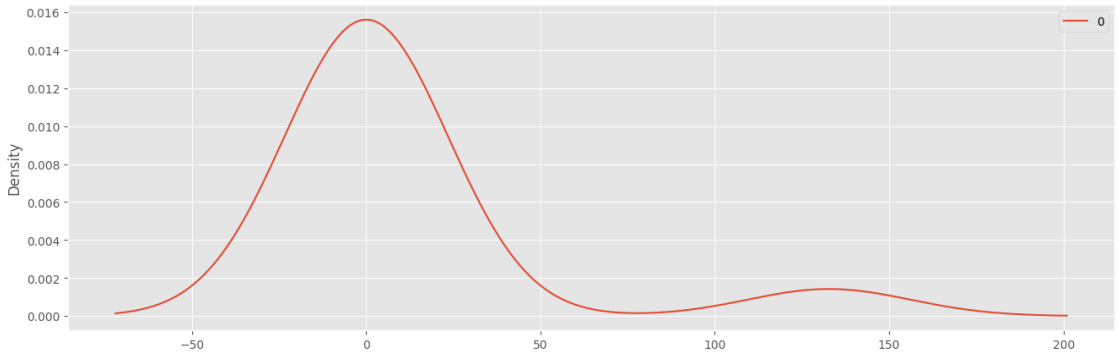


[34]:
```
# 2.  Modelo : ARIMA

#train_data['Month'] = train_data['Month'].dt.to_timestamp()
train_data['Month_numeric'] = train_data['Month'].dt.year * 12 +␣
 ↪train_data['Month'].dt.month

# Luego, utiliza 'Month_numeric' como la serie temporal en auto_arima
modelo_auto = auto_arima(train_data['Month_numeric'], start_p=0, d=1, start_q=0,
                         max_p=4, max_d=2, max_q=4, start_P=0,
                         D=1, start_Q=0, max_P=2, max_D=1,
                         max_Q=2, m=9, seasonal=True,
                         suppress_warnings=True, stepwise=True,
                         random_state=20, n_fits=50)
print(modelo_auto)

arima_model = SARIMAX(train_data["TIEMPO_TOTAL_FINAL"], order = (0,1,0),␣
 ↪seasonal_order = (0,1,0,12))
arima_result = arima_model.fit()
arima_result.summary()

# line plot of residual errors
residuals = pd.DataFrame(arima_result.resid)
```

```
residuals.plot(figsize = (16,5));
plt.show();

# kernel density plot of residual errors
residuals.plot(kind='kde', figsize = (16,5))
plt.show()
print(residuals.describe())

modelo_auto.plot_diagnostics(figsize=(20,8))
plt.show()

print(modelo_auto.summary())

arima_pred = arima_result.predict(start = len(train_data), end = len(df6)-1,␣
 ↪typ="levels").rename("ARIMA Predictions")
test_data['ARIMA_Predictions'] = arima_pred

# Grafica test_data
a=test_data[["TIEMPO_TOTAL_FINAL","ARIMA_Predictions"]]
fig = px.line(a, x=test_data.index, y=a.columns,template = "plotly_dark",
              title="Predicción con Modelo ARIMA")
fig.show()

# evalaucion metricas de Modelo : Arima
evaluacion_metrica(test_data["TIEMPO_TOTAL_FINAL"],test_data["ARIMA_Predictions"])
```

```
ARIMA(0,1,0)(0,1,0)[9] intercept
```



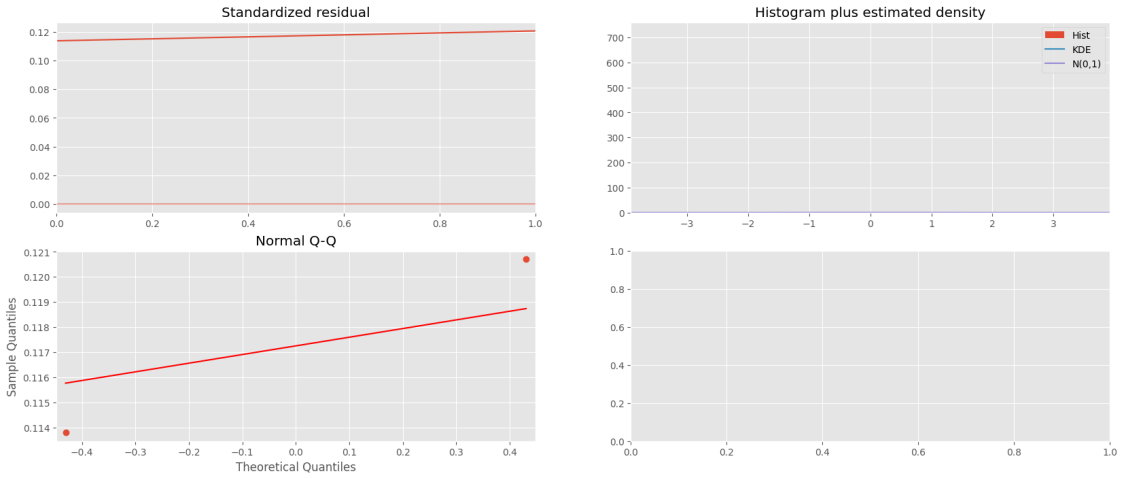



```
                0
count   12.000000
mean    11.101323
std     38.363697
min     -3.668906
25%     -1.356471
50%      0.651822
75%      1.987594
max    132.736505
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-34-571ec03d9461> in <cell line: 29>()
     27 print(residuals.describe())
     28
---> 29 modelo_auto.plot_diagnostics(figsize=(20,8))
     30 plt.show()
     31

/usr/local/lib/python3.10/dist-packages/pmdarima/utils/metaestimators.py in␣
 ↪<lambda>(*args, **kwargs)
     51
     52         # lambda, but not partial, allows help() to work with␣
 ↪update_wrapper
---> 53         out = (lambda *args, **kwargs: self.fn(obj, *args, **kwargs))
     54         # update the docstring of the returned function
     55         update_wrapper(out, self.fn)

/usr/local/lib/python3.10/dist-packages/pmdarima/arima/arima.py in␣
 ↪plot_diagnostics(self, variable, lags, fig, figsize)
   1395         ax = fig.add_subplot(224)
   1396         from statsmodels.graphics import tsaplots
-> 1397         tsaplots.plot_acf(resid, ax=ax, lags=lags)
   1398         ax.set_title('Correlogram')
```

```
   1399

/usr/local/lib/python3.10/dist-packages/pandas/util/_decorators.py in␣
 ↪wrapper(*args, **kwargs)
    209                 else:
    210                     kwargs[new_arg_name] = new_arg_value
--> 211             return func(*args, **kwargs)
    212
    213         return cast(F, wrapper)

/usr/local/lib/python3.10/dist-packages/statsmodels/graphics/tsaplots.py in␣
 ↪plot_acf(x, ax, lags, alpha, use_vlines, adjusted, fft, missing, title, zero,␣
 ↪auto_ylims, bartlett_confint, vlines_kwargs, **kwargs)
    225         acf_x, confint = acf_x[:2]
    226
--> 227     _plot_corr(
    228         ax,
    229         title,

/usr/local/lib/python3.10/dist-packages/statsmodels/graphics/tsaplots.py in␣
 ↪_plot_corr(ax, title, acf_x, confint, lags, irregular, use_vlines,␣
 ↪vlines_kwargs, auto_ylims, **kwargs)
     47
     48     if use_vlines:
---> 49         ax.vlines(lags, [0], acf_x, **vlines_kwargs)
     50         ax.axhline(**kwargs)
     51

/usr/local/lib/python3.10/dist-packages/matplotlib/__init__.py in inner(ax,␣
 ↪data, *args, **kwargs)
   1440     def inner(ax, *args, data=None, **kwargs):
   1441         if data is None:
-> 1442             return func(ax, *map(sanitize_sequence, args), **kwargs)
   1443
   1444         bound = new_sig.bind(ax, *args, **kwargs)

/usr/local/lib/python3.10/dist-packages/matplotlib/axes/_axes.py in vlines(self␣
 ↪x, ymin, ymax, colors, linestyles, label, **kwargs)
   1170         masked_verts[:, 0, 1] = ymin
   1171         masked_verts[:, 1, 0] = x
-> 1172         masked_verts[:, 1, 1] = ymax
   1173
   1174         lines = mcoll.LineCollection(masked_verts, colors=colors,

/usr/local/lib/python3.10/dist-packages/numpy/ma/core.py in __setitem__(self,␣
 ↪indx, value)
   3375         if _mask is nomask:
   3376             # Set the data, then the mask
```

```
-> 3377                 _data[indx] = dval
   3378             if mval is not nomask:
   3379                 _mask = self._mask = make_mask_none(self.shape, _dtype)

ValueError: could not broadcast input array from shape (2,) into shape (11,)
```



[ ]:
```
# 2.  Modelo : ARIMA

import pandas as pd
import matplotlib.pyplot as plt
from pmdarima import auto_arima
from statsmodels.tsa.statespace.sarimax import SARIMAX
import plotly.express as px

#train_data['Month'] = train_data['Month'].dt.to_timestamp()
train_data['Month_numeric'] = train_data['Month'].dt.year * 12 +␣
 ↪train_data['Month'].dt.month

# Luego, utiliza 'Month_numeric' como la serie temporal en auto_arima
modelo_auto = auto_arima(train_data['Month_numeric'], start_p=0, d=1, start_q=0,
                         max_p=4, max_d=2, max_q=4, start_P=0,
                         D=1, start_Q=0, max_P=2, max_D=1,
                         max_Q=2, m=9, seasonal=True,
                         suppress_warnings=True, stepwise=True,
                         random_state=20, n_fits=50)
print(modelo_auto)

arima_model = SARIMAX(train_data["TIEMPO_TOTAL_FINAL"], order = (0,1,0),␣
 ↪seasonal_order = (0,1,0,12))
arima_result = arima_model.fit()
```

```
arima_result.summary()

# line plot of residual errors
residuals = pd.DataFrame(arima_result.resid)
residuals.plot(figsize = (16,5));
plt.show();

# kernel density plot of residual errors
residuals.plot(kind='kde', figsize = (16,5))
plt.show()
print(residuals.describe())

#modelo_auto.plot_diagnostics(figsize=(20,8))
#plt.show()

# Ajustar auto_arima y obtener el mejor modelo
modelo_auto.fit(train_data['Month_numeric'])
best_order = modelo_auto.order
best_seasonal_order = modelo_auto.seasonal_order

# Crear y ajustar manualmente el mejor modelo
best_model = SARIMAX(train_data["TIEMPO_TOTAL_FINAL"], order=best_order,␣
 ↪seasonal_order=best_seasonal_order)
best_result = best_model.fit()

# Mostrar los diagnósticos
best_result.plot_diagnostics(figsize=(20, 8))
plt.show()

print(modelo_auto.summary())

arima_pred = arima_result.predict(start = len(train_data), end = len(df6)-1,␣
 ↪typ="levels").rename("ARIMA Predictions")
test_data['ARIMA_Predictions'] = arima_pred

# Grafica test_data
a=test_data[["TIEMPO_TOTAL_FINAL","ARIMA_Predictions"]]
fig = px.line(a, x=test_data.index, y=a.columns,template = "plotly_dark",
              title="Predicción con Modelo ARIMA")
fig.show()

# evalaucion metricas de Modelo : Arima
evaluacion_metrica(test_data["TIEMPO_TOTAL_FINAL"],test_data["ARIMA_Predictions"])
```

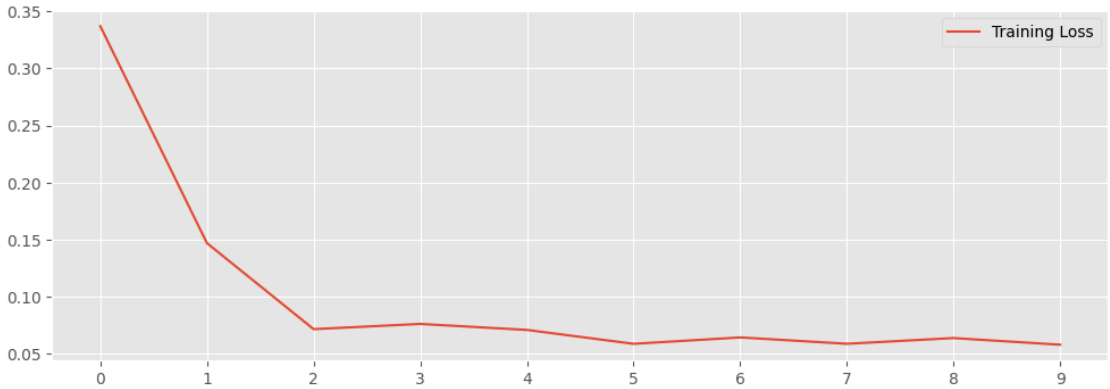
[64]:
```
# 3. Modelo : LSTM_Predictions

from sklearn.preprocessing import MinMaxScaler
```

```
from keras.preprocessing.sequence import TimeseriesGenerator
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM

# train_data = train_data_ori

# Seleccionar solo columnas numéricas
numeric_columns = train_data.select_dtypes(include=['float64']).columns

# Aplicar MinMaxScaler solo a las columnas numéricas
scaler = MinMaxScaler()
scaler.fit(train_data[numeric_columns])

# Transformar el conjunto de entrenamiento y prueba solo en las columnas␣
 ↪numéricas
scaled_train_data = scaler.transform(train_data[numeric_columns])
scaled_test_data = scaler.transform(test_data[numeric_columns])

# Definir parámetros
n_input = 5
n_features = 1

# Antes de crear el modelo LSTM, debemos crear un objeto Generador de series␣
 ↪temporales.

# Crear el generador
generator = TimeseriesGenerator(scaled_train_data, scaled_train_data,␣
 ↪length=n_input, batch_size=1)

# Crear el modelo LSTM
lstm_model = Sequential()
lstm_model.add(LSTM(200, activation='relu', input_shape=(n_input, n_features)))
lstm_model.add(Dense(1))
lstm_model.compile(optimizer='adam', loss='mse')

# Resumen del modelo
lstm_model.summary()

# Entrenar el modelo utilizando el generador
lstm_model.fit(generator, epochs=10)  # Ajusta el número de épocas según sea␣
 ↪necesario

# Obtener el historial de entrenamiento
history = lstm_model.history

# Verificar si hay algún error durante el entrenamiento
```

```
if history is None:
    print("Error: El objeto History no se ha devuelto. Revisa tu código.")
else:
    # Acceder a las métricas de entrenamiento
    losses_lstm = history.history['loss']
    plt.figure(figsize=(12, 4))
    plt.xticks(np.arange(0, 21, 1))
    plt.plot(range(len(losses_lstm)), losses_lstm, label='Training Loss')
    plt.legend()
    plt.show()


lstm_predictions_scaled = list()

batch = scaled_train_data[-n_input:]
current_batch = batch.reshape((1, n_input, n_features))

for i in range(len(test_data)):
    lstm_pred = lstm_model.predict(current_batch)[0]
    lstm_predictions_scaled.append(lstm_pred)
    current_batch = np.append(current_batch[:,1:,:],[[lstm_pred]],axis=1)

lstm_predictions_scaled
lstm_predictions = scaler.inverse_transform(lstm_predictions_scaled)
lstm_predictions
test_data['LSTM_Predictions'] = lstm_predictions

test_data = test_data.drop(columns=['Month1'])

ai=test_data[["TIEMPO_TOTAL_FINAL","LSTM_Predictions"]]
fig = px.line(ai, x=test_data.index, y=ai.columns,title="Predicción con Modelo␣
 ↪LSTM")
fig.show()

# evalaucion metricas de Modelo : LSTM
evaluacion_metrica(test_data["TIEMPO_TOTAL_FINAL"],test_data["LSTM_Predictions"])
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_2 (LSTM)               (None, 200)               161600

 dense_2 (Dense)             (None, 1)                 201

=================================================================
Total params: 161801 (632.04 KB)
```

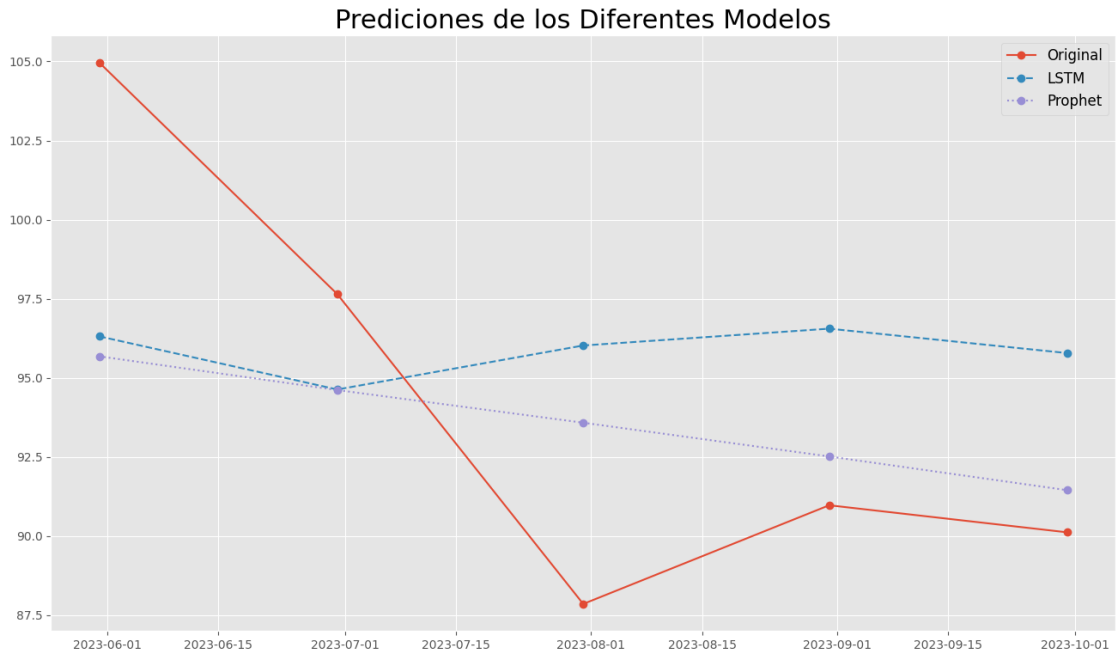
```
Trainable params: 161801 (632.04 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Epoch 1/10
11/11 [==============================] - 3s 16ms/step - loss: 0.3369
Epoch 2/10
11/11 [==============================] - 0s 18ms/step - loss: 0.1469
Epoch 3/10
11/11 [==============================] - 0s 25ms/step - loss: 0.0716
Epoch 4/10
11/11 [==============================] - 0s 22ms/step - loss: 0.0761
Epoch 5/10
11/11 [==============================] - 0s 29ms/step - loss: 0.0709
Epoch 6/10
11/11 [==============================] - 0s 20ms/step - loss: 0.0587
Epoch 7/10
11/11 [==============================] - 0s 19ms/step - loss: 0.0643
Epoch 8/10
11/11 [==============================] - 0s 18ms/step - loss: 0.0587
Epoch 9/10
11/11 [==============================] - 0s 19ms/step - loss: 0.0637
Epoch 10/10
11/11 [==============================] - 0s 18ms/step - loss: 0.0579
```




```
1/1 [==============================] - 0s 494ms/step
1/1 [==============================] - 0s 34ms/step
1/1 [==============================] - 0s 34ms/step
1/1 [==============================] - 0s 34ms/step
1/1 [==============================] - 0s 33ms/step

Evaluation metric results:-
MSE is : 42.779276156479
MAE is : 6.215969210720715
RMSE is : 6.540586835787673
```

```
MAPE is : 6.609435367074856
R2 is : -0.09679542675720532
```

[65]:
```
# mostrar todas los modelos

plt.figure(figsize=(16,9))
plt.plot_date(test_data["Month"],␣
 ↪test_data["TIEMPO_TOTAL_FINAL"],label="Original", linestyle="-")
#plt.plot_date(test_data["Month"], test_data["ARIMA_Predictions"],␣
 ↪label="Arima",linestyle="-.")
plt.plot_date(test_data["Month"], test_data["LSTM_Predictions"],label="LSTM",␣
 ↪linestyle="--")
plt.plot_date(test_data["Month"], test_data["Prophet_Predictions"],␣
 ↪label="Prophet",linestyle=":")
plt.legend(fontsize=12)
plt.title("Prediciones de los Diferentes Modelos", fontsize=22)
plt.show();
```



[66]:
```
test_data
```

[66]:

	index	Month	TIEMPO_TOTAL_FINAL	Prophet_Predictions	LSTM_Predictions
16	16	2023-05	104.964626	95.680576	96.312116
17	17	2023-06	97.646652	94.614332	94.632453
18	18	2023-07	87.855113	93.582482	96.022866
19	19	2023-08	90.973791	92.516238	96.554446

```
20     20  2023-09          90.118430           91.449993         95.783160
```

[39]:
```
df
```

[39]:
```
              FECHA_LLEGADA  TIEMPO_TOTAL_FINAL  \
Month
2023-02 2023-02-24 05:31:45               16.00
2023-08 2023-08-07 23:16:10               16.00
2023-09 2023-09-13 01:56:53               17.00
2023-06 2023-06-29 02:42:38               18.00
2023-01 2023-01-20 13:25:45               20.00
…                       …                   …
2022-07 2022-07-21 10:27:00              994.45
2022-12 2022-12-15 14:50:00              994.80
2023-04 2023-04-10 18:02:25              994.92
2023-07 2023-07-01 16:37:54              995.08
2023-04 2023-04-13 17:48:28              998.92

                              CENTRO_ATENCION  \
Month
2023-02                                   VB
2023-08                                   VC
2023-09                                   UC
2023-06                                   UC
2023-01                                   UC
…                                          …
2022-07                 ME - HOSPITAL MEISSEN
2022-12  UC - CENTRO DE SALUD SANTA LIBRADA I
2023-04                                   ME
2023-07                                   TN
2023-04                                   ME

                                 CLASIFICACION_TRIAGE PACIENTE_EDAD  \
Month
2023-02                                             3           69
2023-08                                             3           50
2023-09                                             3           79
2023-06                                             3           72
2023-01                                             3           48
…                                                   …           …
2022-07  3 - TRIAGE III - NO DEBE SUPERAR LAS 3 HORAS     58 ANO(S)
2022-12  3 - TRIAGE III - NO DEBE SUPERAR LAS 3 HORAS     54 ANO(S)
2023-04                                             3            4
2023-07                                             3            3
2023-04                                             3           17

        PACIENTE_#_DOCUMENTO        EDAD_RANGO             NOMBRE_ENTIDAD  \
```

```
Month
2023-02              4450531      ADULTO MAYOR                     EPSC34
2023-08              5267156            ADULTO                     EPSC34
2023-09              5667630      ADULTO MAYOR                     EPSC34
2023-06              3014617      ADULTO MAYOR                     EPSS05
2023-01             52277907            ADULTO                     EPSC34
…                        …                 …                          …
2022-07             39532594           ADULTEZ  CAPITAL SALUD EPS-S S.A.S
2022-12             39723106           ADULTEZ  CAPITAL SALUD EPS-S S.A.S
2023-04           1033822858  PRIMERA INFANCIA                     EPSC34
2023-07           1243858533  PRIMERA INFANCIA                     EPSC34
2023-04           1010176536       ADOLECENCIA                     EPSS41

              SEXO  DIA_SEMANA  ANUAL  TIEMPO_TOTAL_FINAL_change
Month
2023-02  MASCULINO           4   2023                   0.066667
2023-08   FEMENINO           0   2023                   0.000000
2023-09  MASCULINO           2   2023                   0.062500
2023-06  MASCULINO           3   2023                   0.058824
2023-01   FEMENINO           4   2023                   0.111111
…              …          …     …                          …
2022-07   FEMENINO           3   2022                   0.000251
2022-12   FEMENINO           3   2022                   0.000352
2023-04  MASCULINO           0   2023                   0.000121
2023-07   FEMENINO           5   2023                   0.000161
2023-04   FEMENINO           3   2023                   0.003859

[224409 rows x 12 columns]
```

[68]:
```
test_data
```

[68]:
```
    index    Month  TIEMPO_TOTAL_FINAL  Prophet_Predictions  LSTM_Predictions  \
17     17  2023-06           97.646652            94.614332         94.632453
18     18  2023-07           87.855113            93.582482         96.022866
19     19  2023-08           90.973791            92.516238         96.554446
20     20  2023-09           90.118430            91.449993         95.783160

    TIEMPO_TOTAL_FINAL_change
17                  -0.069718
18                  -0.100275
19                   0.035498
20                  -0.009402
```

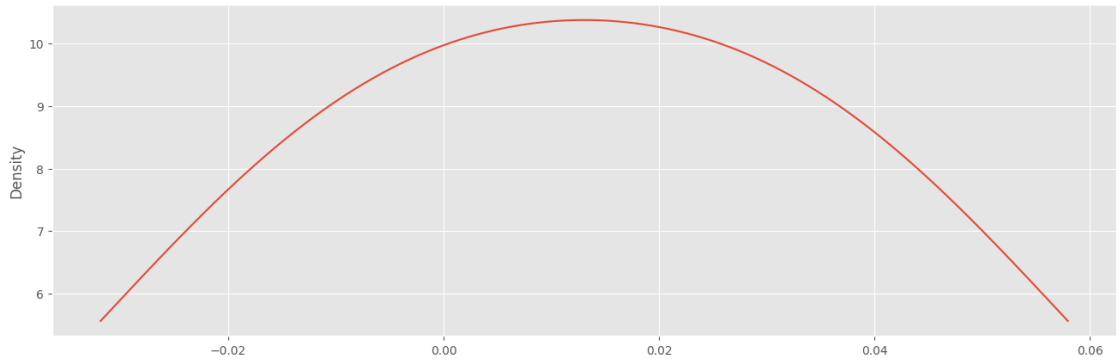
[70]:
```
# 4. Modelo :  Random Forest
df5 = test_data
#df5 = test_data
```

```
df5['TIEMPO_TOTAL_FINAL_change']= df5['TIEMPO_TOTAL_FINAL'].pct_change()
df5.dropna(inplace=True)
df5.head()
df5['TIEMPO_TOTAL_FINAL_change'].describe()
```

[70]:
```
count    2.000000
mean     0.013048
std      0.031749
min     -0.009402
25%      0.001823
50%      0.013048
75%      0.024273
max      0.035498
Name: TIEMPO_TOTAL_FINAL_change, dtype: float64
```

[71]:
```
# Convertir la columna 'Month' a un formato serializable (cadena)
df5['Month'] = df5['Month'].dt.strftime('%Y-%m')  # Ajusta el formato según tus␣
 ↪necesidades

# Crear el gráfico
fig = px.line(df5, x="Month", y="TIEMPO_TOTAL_FINAL_change",␣
 ↪template="plotly_dark",
              title="Porcentaje de Cambio")

# Mostrar el gráfico
fig.show()
```

[74]:
```
df5['TIEMPO_TOTAL_FINAL_change'].plot(kind='kde',figsize = (16,5));

# Seasonality variables

df5['Month'] = pd.to_datetime(df5['Month'], format='%Y-%m')

# Ahora puedes continuar con las otras conversiones
df5['Year'] = df5['Month'].apply(lambda x: x.year)
df5['Mes'] = df5['Month'].apply(lambda x: x.month)
```



[ ]:
```
# Adding a year of lagged data
df5['L1'] = df5["TIEMPO_TOTAL_FINAL_change"].shift(1)
df5['L2'] = df5["TIEMPO_TOTAL_FINAL_change"].shift(2)
df5['L3'] = df5["TIEMPO_TOTAL_FINAL_change"].shift(3)
df5['L4'] = df5["TIEMPO_TOTAL_FINAL_change"].shift(4)
df5['L5'] = df5["TIEMPO_TOTAL_FINAL_change"].shift(5)
df5['L6'] = df5["TIEMPO_TOTAL_FINAL_change"].shift(6)

df5.head(13)

# df5 = df5.dropna()
df5= df5.set_index("Month")
df5.head()
```

[78]:
```
df5
```

[78]:
```
            index  TIEMPO_TOTAL_FINAL  Prophet_Predictions  LSTM_Predictions  \
Month
2023-08-01     19           90.973791            92.516238         96.554446
2023-09-01     20           90.118430            91.449993         95.783160

            TIEMPO_TOTAL_FINAL_change  Year  Mes        L1  L2  L3  L4  L5  \
Month
2023-08-01                   0.035498  2023    8       NaN NaN NaN NaN NaN
2023-09-01                  -0.009402  2023    9  0.035498 NaN NaN NaN NaN

            L6  L7
Month
2023-08-01 NaN NaN
2023-09-01 NaN NaN
```

[79]:
```
# Modelación
furn = df5
# split into lagged variables (features) and original time series data (target)
X2= df5.iloc[:,2:-1]  # slice all rows and start with column 0 and go up to but␣
 ↪not including the last column
y2 = furn.iloc[:,1]  # slice all rows and last column, essentially separating␣
 ↪out 't' column
X2

# Target Train-Test split
from pandas import read_csv

Y2 = y2
```

```
traintarget_size = int(len(Y2) * 0.80)   # Set split
train_target, test_target = Y2[0:traintarget_size], Y2[traintarget_size:len(Y2)]

print('Observations for Target: %d' % (len(Y2)))
print('Training Observations for Target: %d' % (len(train_target)))
print('Testing Observations for Target: %d' % (len(test_target)))
```

```
Observations for Target: 2
Training Observations for Target: 1
Testing Observations for Target: 1
```

[80]:
```
# Random Forest
from sklearn.ensemble import RandomForestRegressor

# Creamos el modelo con 500 árboles
rfr = RandomForestRegressor(n_estimators=500)

# Entrenamos el modelo
rfr.fit(train_feature, train_target)

# Hacemos las predicciones
fcst = rfr.predict(test_feature)

b=pd.DataFrame({"Actual":test_target, "Random Forest":fcst})
b

fig = px.line(b, x=b.index, y=b.columns,template = "plotly_dark",
              title="Predicción con Modelo Random Forest")
fig.show()

# Evaluacion metricas del modelo : Random Forest
evaluacion_metrica(test_target,fcst)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-80-ad6d3717ffaf> in <cell line: 8>()
      6
      7 # Entrenamos el modelo
----> 8 rfr.fit(train_feature, train_target)
      9
     10 # Hacemos las predicciones

NameError: name 'train_feature' is not defined
```

[67]:
```
# 4. Modelo :  Random Forest
df5 = test_data
```

```
#df5 = test_data

df5['TIEMPO_TOTAL_FINAL_change']= df5['TIEMPO_TOTAL_FINAL'].pct_change()
df5.dropna(inplace=True)
df5.head()
df5['TIEMPO_TOTAL_FINAL_change'].describe()

fig = px.line(df5, x="Month", y="TIEMPO_TOTAL_FINAL_change",template =␣
 ↪"plotly_dark",
              title="Porcentaje de Cambio")
fig.show()

df5['TIEMPO_TOTAL_FINAL_change'].plot(kind='kde',figsize = (16,5));

# Seasonality variables
df5['Month'] = df5['Month'].dt.to_timestamp()

# Ahora puedes continuar con las otras conversiones
df5['Year'] = df5['Month'].apply(lambda x: x.year)
df5['Mes'] = df5['Month'].apply(lambda x: x.month)

# Adding a year of lagged data
df5['L1'] = df5["TIEMPO_TOTAL_FINAL_change"].shift(1)
df5['L2'] = df5["TIEMPO_TOTAL_FINAL_change"].shift(2)
df5['L3'] = df5["TIEMPO_TOTAL_FINAL_change"].shift(3)
df5['L4'] = df5["TIEMPO_TOTAL_FINAL_change"].shift(4)
df5['L5'] = df5["TIEMPO_TOTAL_FINAL_change"].shift(5)
df5['L6'] = df5["TIEMPO_TOTAL_FINAL_change"].shift(6)
df5['L7'] = df5["TIEMPO_TOTAL_FINAL_change"].shift(7)
df5['L8'] = df5["TIEMPO_TOTAL_FINAL_change"].shift(8)
df5['L9'] = df5["TIEMPO_TOTAL_FINAL_change"].shift(9)
df5['L10'] = df5["TIEMPO_TOTAL_FINAL_change"].shift(10)
df5['L11'] = df5["TIEMPO_TOTAL_FINAL_change"].shift(11)
df5['L12'] = df5["TIEMPO_TOTAL_FINAL_change"].shift(12)

df5.head(13)

# df5 = df5.dropna()
df5= df5.set_index("Month")
df5.head()


# Modelación
furn = df5
# split into lagged variables (features) and original time series data (target)
X2= df5.iloc[:,2:-1]  # slice all rows and start with column 0 and go up to but␣
 ↪not including the last column
```

```
y2 = furn.iloc[:,1]  # slice all rows and last column, essentially separating␣
 ↪out 't' column
X2

# Target Train-Test split
from pandas import read_csv

Y2 = y2
traintarget_size = int(len(Y2) * 0.80)   # Set split
train_target, test_target = Y2[0:traintarget_size], Y2[traintarget_size:len(Y2)]

print('Observations for Target: %d' % (len(Y2)))
print('Training Observations for Target: %d' % (len(train_target)))
print('Testing Observations for Target: %d' % (len(test_target)))


# Features Train-Test split

trainfeature_size = int(len(X2) * 0.80)
train_feature, test_feature = X2[0:trainfeature_size], X2[trainfeature_size:
 ↪len(X2)]
print('Observations for feature: %d' % (len(X2)))
print('Training Observations for feature: %d' % (len(train_feature)))
print('Testing Observations for feature: %d' % (len(test_feature)))

# Random Forest
from sklearn.ensemble import RandomForestRegressor

# Creamos el modelo con 500 árboles
rfr = RandomForestRegressor(n_estimators=500)

# Entrenamos el modelo
rfr.fit(train_feature, train_target)

# Hacemos las predicciones
fcst = rfr.predict(test_feature)

b=pd.DataFrame({"Actual":test_target, "Random Forest":fcst})
b

fig = px.line(b, x=b.index, y=b.columns,template = "plotly_dark",
              title="Predicción con Modelo Random Forest")
fig.show()

# Evaluacion metricas del modelo : Random Forest
evaluacion_metrica(test_target,fcst)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-67-935392045a18> in <cell line: 12>()
     10 fig = px.line(df5, x="Month", y="TIEMPO_TOTAL_FINAL_change",template =␣
 ↪"plotly_dark",
     11               title="Porcentaje de Cambio")
---> 12 fig.show()
     13
     14 df5['TIEMPO_TOTAL_FINAL_change'].plot(kind='kde',figsize = (16,5));

/usr/local/lib/python3.10/dist-packages/plotly/basedatatypes.py in show(self,␣
 ↪*args, **kwargs)
   3407         import plotly.io as pio
   3408
-> 3409         return pio.show(self, *args, **kwargs)
   3410
   3411     def to_json(self, *args, **kwargs):

/usr/local/lib/python3.10/dist-packages/plotly/io/_renderers.py in show(fig,␣
 ↪renderer, validate, **kwargs)
    386
    387     # Mimetype renderers
--> 388     bundle = renderers._build_mime_bundle(fig_dict,␣
 ↪renderers_string=renderer, **kwargs)
    389     if bundle:
    390         if not ipython_display:

/usr/local/lib/python3.10/dist-packages/plotly/io/_renderers.py in␣
 ↪_build_mime_bundle(self, fig_dict, renderers_string, **kwargs)
    294                         setattr(renderer, k, v)
    295
--> 296                 bundle.update(renderer.to_mimebundle(fig_dict))
    297
    298         return bundle

/usr/local/lib/python3.10/dist-packages/plotly/io/_base_renderers.py in␣
 ↪to_mimebundle(self, fig_dict)
    377                 post_script.extend(self.post_script)
    378
--> 379         html = to_html(
    380             fig_dict,
    381             config=self.config,

/usr/local/lib/python3.10/dist-packages/plotly/io/_html.py in to_html(fig,␣
 ↪config, auto_play, include_plotlyjs, include_mathjax, post_script, full_html,␣
 ↪animation_opts, default_width, default_height, validate, div_id)
```

```
    142
    143     # ## Serialize figure ##
--> 144     jdata = to_json_plotly(fig_dict.get("data", []))
    145     jlayout = to_json_plotly(fig_dict.get("layout", {}))
    146

/usr/local/lib/python3.10/dist-packages/plotly/io/_json.py in␣
 ↪to_json_plotly(plotly_object, pretty, engine)
    141
    142         return _safe(
--> 143             json.dumps(plotly_object, cls=PlotlyJSONEncoder, **opts),␣
 ↪_swap_json
    144         )
    145     elif engine == "orjson":

/usr/lib/python3.10/json/__init__.py in dumps(obj, skipkeys, ensure_ascii,␣
 ↪check_circular, allow_nan, cls, indent, separators, default, sort_keys, **kw)
    236         check_circular=check_circular, allow_nan=allow_nan,␣
 ↪indent=indent,
    237         separators=separators, default=default, sort_keys=sort_keys,
--> 238         **kw).encode(obj)
    239
    240

/usr/local/lib/python3.10/dist-packages/_plotly_utils/utils.py in encode(self, o)
     57         """
     58         # this will raise errors in a normal-expected way
---> 59         encoded_o = super(PlotlyJSONEncoder, self).encode(o)
     60         # Brute force guessing whether NaN or Infinity values are in the␣
 ↪string
     61         # We catch false positive cases (e.g. strings such as titles,␣
 ↪labels etc.)

/usr/lib/python3.10/json/encoder.py in encode(self, o)
    197         # exceptions aren't as detailed.  The list call should be roughly
    198         # equivalent to the PySequence_Fast that ''.join() would do.
--> 199         chunks = self.iterencode(o, _one_shot=True)
    200         if not isinstance(chunks, (list, tuple)):
    201             chunks = list(chunks)

/usr/lib/python3.10/json/encoder.py in iterencode(self, o, _one_shot)
    255                 self.key_separator, self.item_separator, self.sort_keys,
    256                 self.skipkeys, _one_shot)
--> 257         return _iterencode(o, 0)
    258
    259 def _make_iterencode(markers, _default, _encoder, _indent, _floatstr,
```

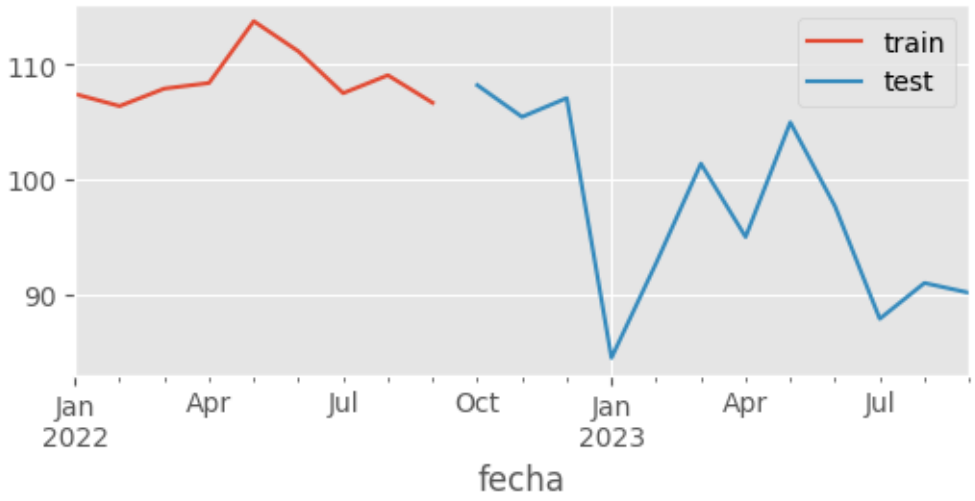
```
/usr/local/lib/python3.10/dist-packages/_plotly_utils/utils.py in default(self,
 ↪obj)
    134             except NotEncodable:
    135                 pass
--> 136         return _json.JSONEncoder.default(self, obj)
    137
    138     @staticmethod

/usr/lib/python3.10/json/encoder.py in default(self, o)
    177
    178         """
--> 179         raise TypeError(f'Object of type {o.__class__.__name__} '
    180                         f'is not JSON serializable')
    181

TypeError: Object of type Period is not JSON serializable
```

[ ]:
```
#  5. Modelo :  forecasting series temporales con Python y Scikit-learn

# Gráficos
# ==============================================================================
import matplotlib.pyplot as plt
plt.style.use('fivethirtyeight')
plt.rcParams['lines.linewidth'] = 1.5
plt.rcParams['font.size'] = 10

# Modelado y Forecasting
# ==============================================================================
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Lasso
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from sklearn.preprocessing import StandardScaler

from skforecast.ForecasterAutoreg import ForecasterAutoreg
from skforecast.ForecasterAutoregCustom import ForecasterAutoregCustom
from skforecast.ForecasterAutoregDirect import ForecasterAutoregDirect
from skforecast.model_selection import grid_search_forecaster
from skforecast.model_selection import backtesting_forecaster
from skforecast.utils import save_forecaster
from skforecast.utils import load_forecaster
```

[83]:
```
# 5. Modelo : Multiple output forecasting

# Resetear el índice antes de la conversión
```

```
df6 = df6.reset_index(drop=True)

# Convierte 'Month' a cadena y luego a formato Timestamp
df6['Month'] = pd.to_datetime(df6['Month'].astype(str), format='%Y-%m')

# Ahora puedes continuar con las otras conversiones
datos = df6
datos['fecha'] = pd.to_datetime(datos['Month'], format='%Y-%m-%d')
datos = datos.set_index('fecha')
datos = datos.rename(columns={'x': 'y'})
datos = datos.asfreq('MS')
datos = datos.sort_index()
datos = datos.rename(columns={'TIEMPO_TOTAL_FINAL': 'y'})

# Verificar que un índice temporal está completo
# ==============================================================================
(datos.index == pd.date_range(
                    start = datos.index.min(),
                    end   = datos.index.max(),
                    freq  = datos.index.freq)
).all()

# Separación datos train-test
# ==============================================================================
steps = 12
datos_train = datos[:-steps]
datos_test  = datos[-steps:]

print(f"Fechas train : {datos_train.index.min()} --- {datos_train.index.max()} ␣
 ↪(n={len(datos_train)})")
print(f"Fechas test  : {datos_test.index.min()} --- {datos_test.index.max()} ␣
 ↪(n={len(datos_test)})")

fig, ax = plt.subplots(figsize=(6, 2.5))
datos_train['y'].plot(ax=ax, label='train')
datos_test['y'].plot(ax=ax, label='test')
ax.legend();
```

```
Fechas train : 2022-01-01 00:00:00 --- 2022-09-01 00:00:00  (n=9)
Fechas test  : 2022-10-01 00:00:00 --- 2023-09-01 00:00:00  (n=12)
```

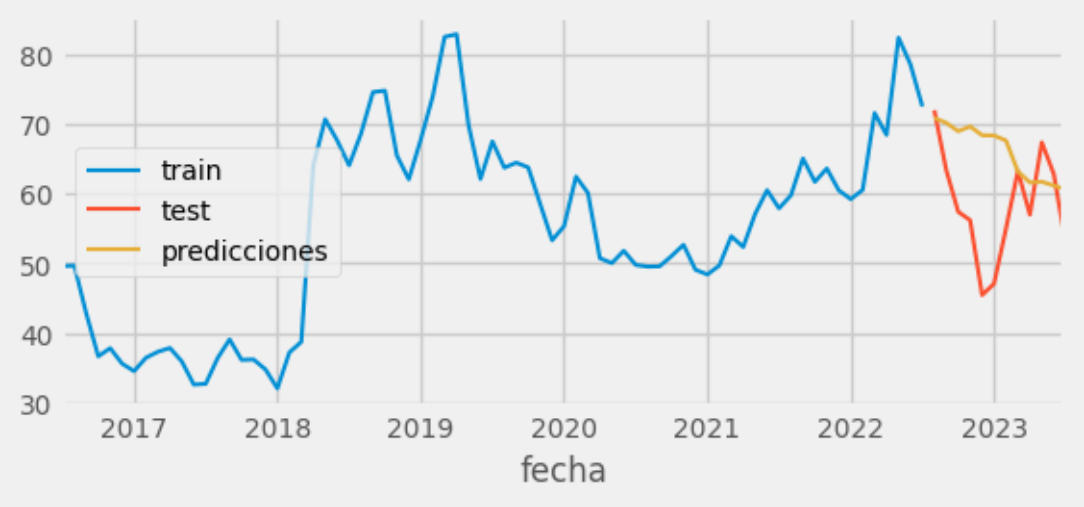


[84]:
```
# 6. Modelo : Forecasting autorregresivo recursivo

# Crear y entrenar forecaster
# ==============================================================================
forecaster = ForecasterAutoreg(
                regressor = RandomForestRegressor(random_state=123),
                lags = 6
             )

forecaster.fit(y=datos_train['y'])
forecaster

# Predicciones
# ==============================================================================
steps = 12
predicciones = forecaster.predict(steps=steps)
predicciones.head(5)

# Gráfico
# ==============================================================================
fig, ax = plt.subplots(figsize=(6, 2.5))
datos_train['y'].plot(ax=ax, label='train')
datos_test['y'].plot(ax=ax, label='test')
predicciones.plot(ax=ax, label='predicciones')
ax.legend();

# Error test
# ==============================================================================
error_mse = mean_squared_error(
```

```
                y_true = datos_test['y'],
                y_pred = predicciones
            )

print(f"Error de test (mse): {error_mse}")
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-84-1e5d17cddc5b> in <cell line: 5>()
      3 # Crear y entrenar forecaster
      4 #␣
 ↪==========================================================================
----> 5 forecaster = ForecasterAutoreg(
      6                 regressor = RandomForestRegressor(random_state=123),
      7                 lags = 6

NameError: name 'ForecasterAutoreg' is not defined
```

[ ]:
```
# Crear y entrenar forecaster con mejores hiperparámetros
# ==============================================================================
regressor = RandomForestRegressor(max_depth=3, n_estimators=100,␣
 ↪random_state=123)
forecaster = ForecasterAutoreg(
                regressor = regressor,
                lags      = 20
             )

forecaster.fit(y=datos_train['y'])

# Predicciones
# ==============================================================================
predicciones = forecaster.predict(steps=steps)

# Gráfico
# ==============================================================================
fig, ax = plt.subplots(figsize=(6, 2.5))
datos_train['y'].plot(ax=ax, label='train')
datos_test['y'].plot(ax=ax, label='test')
predicciones.plot(ax=ax, label='predicciones')
ax.legend();
```



[ ]:
```
# Grid search de hiperparámetros
# ==============================================================================
steps = 12
forecaster = ForecasterAutoreg(
                regressor = RandomForestRegressor(random_state=123),
                lags      = 12 # Este valor será remplazado en el grid search
             )

# Lags utilizados como predictores
lags_grid = [10, 20]

# Hiperparámetros del regresor
param_grid = {'n_estimators': [100, 500],
              'max_depth': [3, 5, 10]}

resultados_grid = grid_search_forecaster(
                        forecaster         = forecaster,
                        y                  = datos_train['y'],
                        param_grid         = param_grid,
                        lags_grid          = lags_grid,
                        steps              = steps,
                        refit              = False,
                        metric             = 'mean_squared_error',
                        initial_train_size = int(len(datos_train)*0.5),
                        fixed_train_size   = False,
                        return_best        = True,
                        n_jobs             = 'auto',
                        verbose            = False
                    )
```

```
# Resultados Grid Search
# ==============================================================================
resultados_grid
```

```
Number of models compared: 12.
lags grid:   0%|          | 0/2 [00:00<?, ?it/s]
params grid:   0%|          | 0/6 [00:00<?, ?it/s]
`Forecaster` refitted using the best-found lags and parameters, and the whole
data set:
  Lags: [ 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20]
  Parameters: {'max_depth': 3, 'n_estimators': 100}
  Backtesting metric: 71.00953593067813
```

[ ]:

	lags	params	mean_squared_error	max_depth	n_estimators
6	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14…	{'max_depth': 3, 'n_estimators': 100}	71.009536	3	100
8	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14…	{'max_depth': 5, 'n_estimators': 100}	71.434458	5	100
10	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14…	{'max_depth': 10, 'n_estimators': 100}	71.552524	10	100
11	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14…	{'max_depth': 10, 'n_estimators': 500}	71.584441	10	500
9	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14…	{'max_depth': 5, 'n_estimators': 500}	71.617320	5	500
7	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14…	{'max_depth': 3, 'n_estimators': 500}	71.752363	3	500
2	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]	{'max_depth': 5, 'n_estimators': 100}	198.464857	5	100
3	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]	{'max_depth': 5, 'n_estimators': 500}	201.324686	5	500
5	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]	{'max_depth': 10, 'n_estimators': 500}	203.142210	10	500
4	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]	{'max_depth': 10, 'n_estimators': 100}	204.810631	10	100
1	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]	{'max_depth': 3, 'n_estimators': 500}	206.166672	3	500
0	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]	{'max_depth': 3, 'n_estimators': 100}	211.709367	3	100

11	500
9	500
7	500
2	100
3	500
5	500
4	100
1	500
0	100