TRABAJO 1

Juan Gabriel Carvajal Negrete

1003713613

Carmen Daniela Zabaleta Cardeño

1007706542

Introducción a la Minería de Datos

Julieth Veronica Guarin Escudero

Universidad Nacional de Colombia

Sede Medellín

Departamento de Estadística

Medellín, Marzo de 2024

Introducción:

En la etapa de preprocesamiento se tiene como objetivo abordar técnicas que permitan obtener información organizada, evitar información que pueda ser redundante, identificar posibles problemas presentes en las bases de datos y hacer el respectivo tratamiento. Específicamente, en este trabajo se desarrollarán metodologías para el tratamiento de datos atípicos y datos faltantes.

1. A partir del dataset ruidoso.txt realice los siguientes análisis:
   a) Cargue y explore el dataset explicando en qué consiste y las características que posee el mismo.
   b) Realice un breve análisis exploratorio para identificar la distribución de las variables usadas en la base de datos ¿será que existe relación entre las variables?
   c) Verifique si existen problemas de datos atípicos en cada una de las variables usando las metodologías de detección a nivel univariado.
   d) ¿Se detectan valores atípicos a nivel multivariado?
   e) Para el caso univariado, escoja una variable y realice un análisis sobre las implicaciones que tiene realizar diferentes tratamientos a los datos atípicos en la distribución de la respectiva variable.

```
# Librerias a usar
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import numpy as np
import plotly.express as px
import missingno as msno
from sklearn.impute import SimpleImputer
from sklearn.impute import KNNImputer
from pyod.models.mad import MAD
```

Cargando la base de datos

```
# Lectura del archivo y cargando los datos en un DataFrame
ruidoso = pd.read_csv("ruidoso.txt", sep=",", index_col = 0)
```

Dimensiones de los datos

```
# Mirando las dimensiones
ruidoso.shape
```

```
(66, 4)
```

Encabezado y cola de los datos

```
# Encabezado y cola de los datos
print(ruidoso.head())
print()
print(ruidoso.tail())
```

```
   Road_55dB  Road_60dB  Railways_65dB  Industry_65dB
1     166400      79200           1100            0.0
2      20000      11000              0            0.0
3      37800      18000              0            0.0
4      24500      16700              0          100.0
5     103100      33500           1900          200.0

    Road_55dB  Road_60dB  Railways_65dB  Industry_65dB
62      20700      16200              0            0.0
63      15100       9900              0            0.0
64      24500      12400            300          300.0
65      12800       8500              0            0.0
66    3108200    1469100          25800            NaN
```

Mirando las caracteristicas de los datos

```
# Mirando las caracteristicas de los datos
ruidoso.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 66 entries, 1 to 66
Data columns (total 4 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Road_55dB      66 non-null     int64
 1   Road_60dB      66 non-null     int64
 2   Railways_65dB  66 non-null     int64
 3   Industry_65dB  65 non-null     float64
dtypes: float64(1), int64(3)
memory usage: 2.6 KB
```

Vemos que la información fué cargada correctamente y que el conjunto de datos cuenta con 66 registros y 4 variables.

Este conjunto de datos parece representar mediciones de diferentes niveles de ruido en distintos entornos, como carreteras, vías de ferrocarril e industrias. Cada fila representa una observación numerada y contiene información sobre los niveles de ruido en decibeles (dB) en varios lugares.

Las características principales de este conjunto de datos son:

1. **Variables**:

   - Road_55dB: Nivel de ruido en carreteras a 55 decibeles (dB).
   - Road_60dB: Nivel de ruido en carreteras a 60 decibeles (dB).

- Railways_65dB: Nivel de ruido en vías de ferrocarril a 65 decibeles (dB).
- Industry_65dB: Nivel de ruido en industrias a 65 decibeles (dB).

2. **Registros**: Cada fila representa una observación o medición en un determinado contexto o ubicación.
3. **Unidad de Medida**: Las unidades de medida para los valores de las variables son decibeles (dB), que es la medida estándar para el nivel de ruido.
4. **Rango de Datos**: Los valores en el conjunto de datos representan diferentes niveles de ruido en decibelios, que van desde 0 hasta números bastante grandes.

En este conjunto de datos hay un valor nulo en la variable "Industry_65dB", el resto de las variables no poseen aparentemente valores nulos.

b. Realice un breve análisis exploratorio para identificar la distribución de las variables usadas en la base de datos ¿será que existe relación entre las variables?

Gráfico del comportamiento de los datos.

```
# Importar libreria para el gráfico

# Columnas para el gráfico de líneas
columnas_ruido = ['Road_55dB', 'Road_60dB', 'Railways_65dB', 'Industry_65dB']

# Crea el gráfico de líneas
plt.figure(figsize=(8, 5))

# Itera sobre las columnas
for columna in columnas_ruido:
    plt.plot(ruidoso.index, ruidoso[columna], label=columna)

# Etiquetas y título del gráfico
plt.xlabel('Número del registro')
plt.ylabel('Nivel de ruido (dB)')
plt.title('Comportamiento de los niveles de ruido en diferentes ambientes')

# Añade una leyenda
plt.legend()

plt.grid(True)  # Agrega una cuadrícula al gráfico
plt.tight_layout()  # Ajusta el diseño del gráfico
plt.show()
```

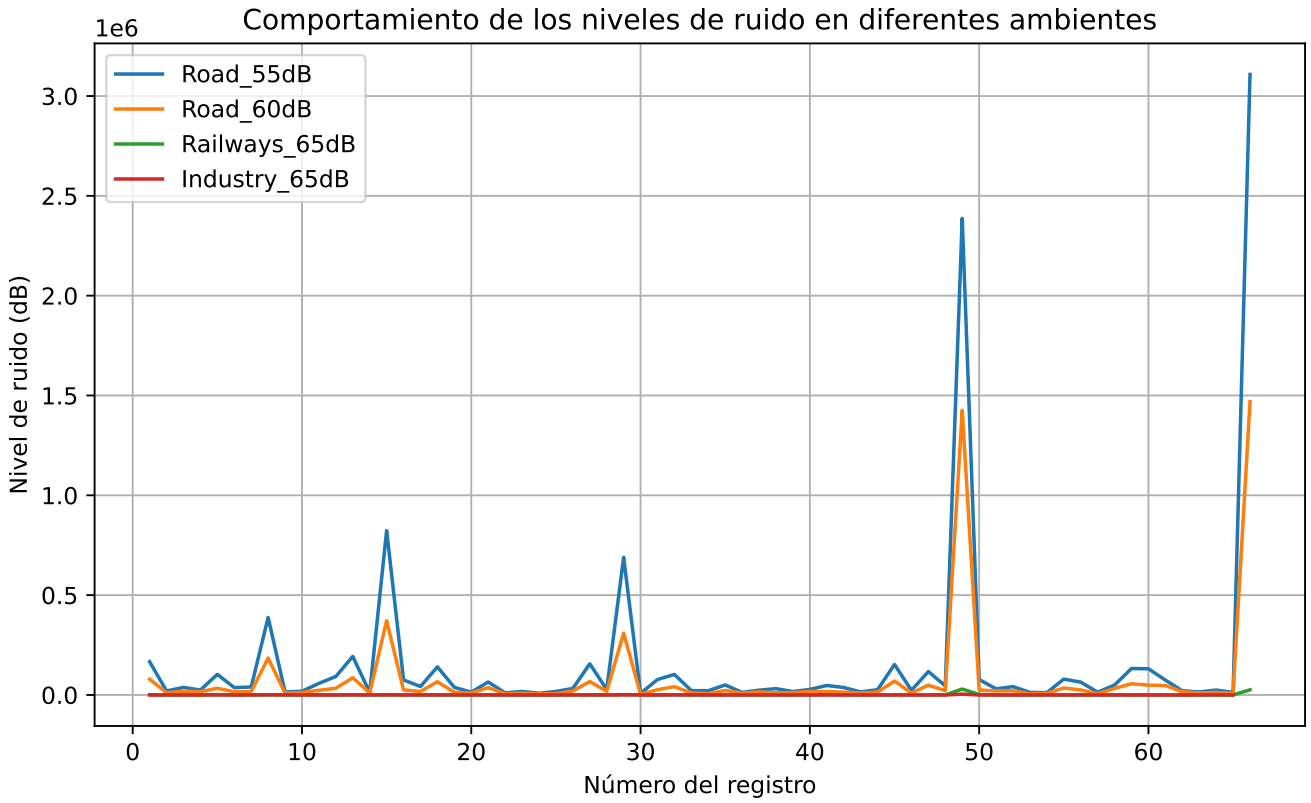
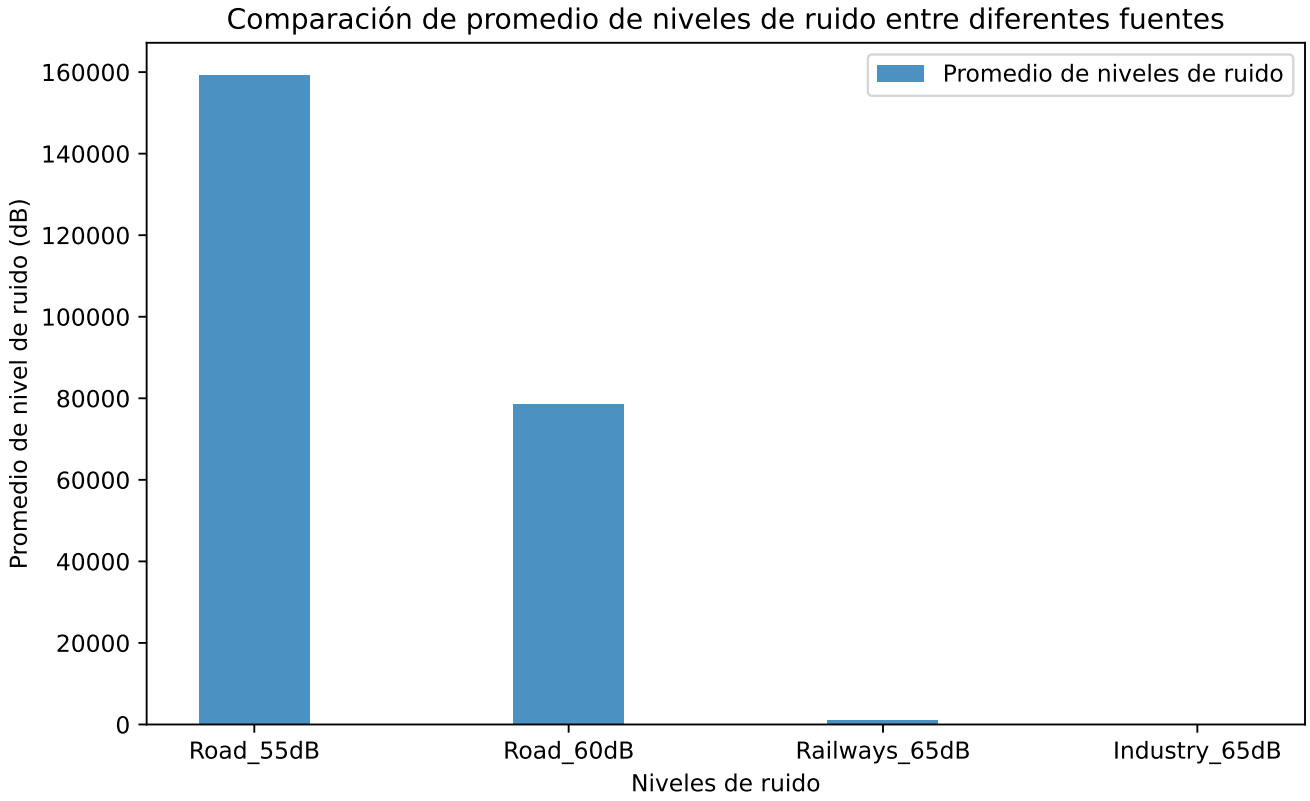


Gráfico de barras agrupadas

```
# Importando libreria necesaria
import numpy as np

# Columnas para el gráfico de barras
columnas_ruido = ['Road_55dB', 'Road_60dB', 'Railways_65dB', 'Industry_65dB']

# Calcula el promedio de los niveles de ruido para cada columna
promedios_ruido = ruidoso[columnas_ruido].mean()

# Crea el gráfico de barras agrupadas
plt.figure(figsize=(8, 5))

# Genera el gráfico de barras agrupadas
index = np.arange(len(columnas_ruido))  # Índices para las barras
bar_width = 0.35  # Ancho de las barras
opacity = 0.8  # Opacidad de las barras

plt.bar(index, promedios_ruido, bar_width, alpha=opacity, label='Promedio de niveles de ruido')

# Añade etiquetas y título al gráfico
plt.xlabel('Niveles de ruido')
plt.ylabel('Promedio de nivel de ruido (dB)')
plt.title('Comparación de promedio de niveles de ruido entre diferentes fuentes')
```

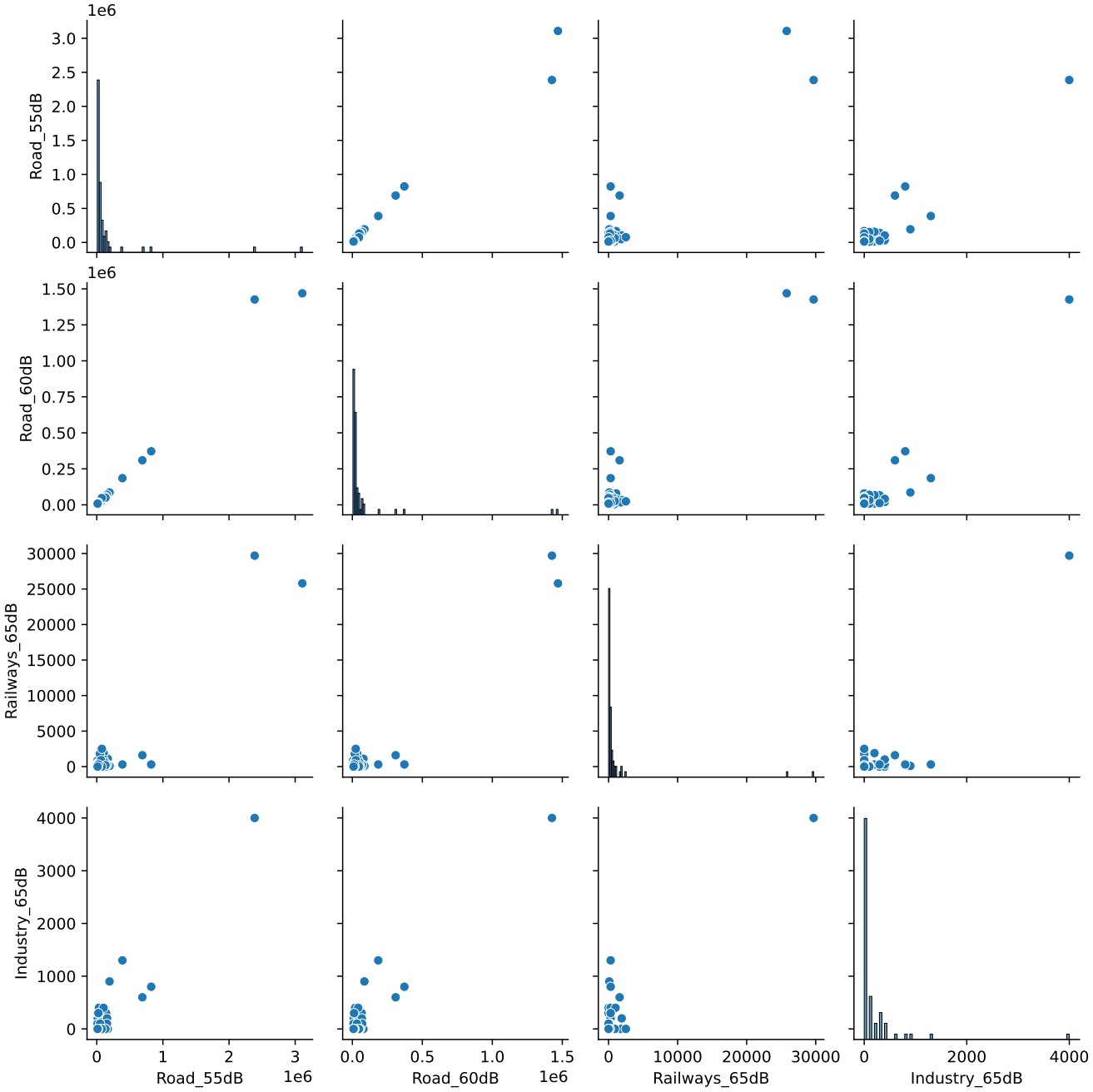
```
plt.xticks(index, columnas_ruido)

# Leyenda para identificar las barras en el gráfico
plt.legend()

plt.tight_layout()
plt.show()
```



De los gráficos anteriores observamos que la variable Road_55dB (Nivel de ruido a 55 decibeles en carreteras) es la que presenta los valores más altos.

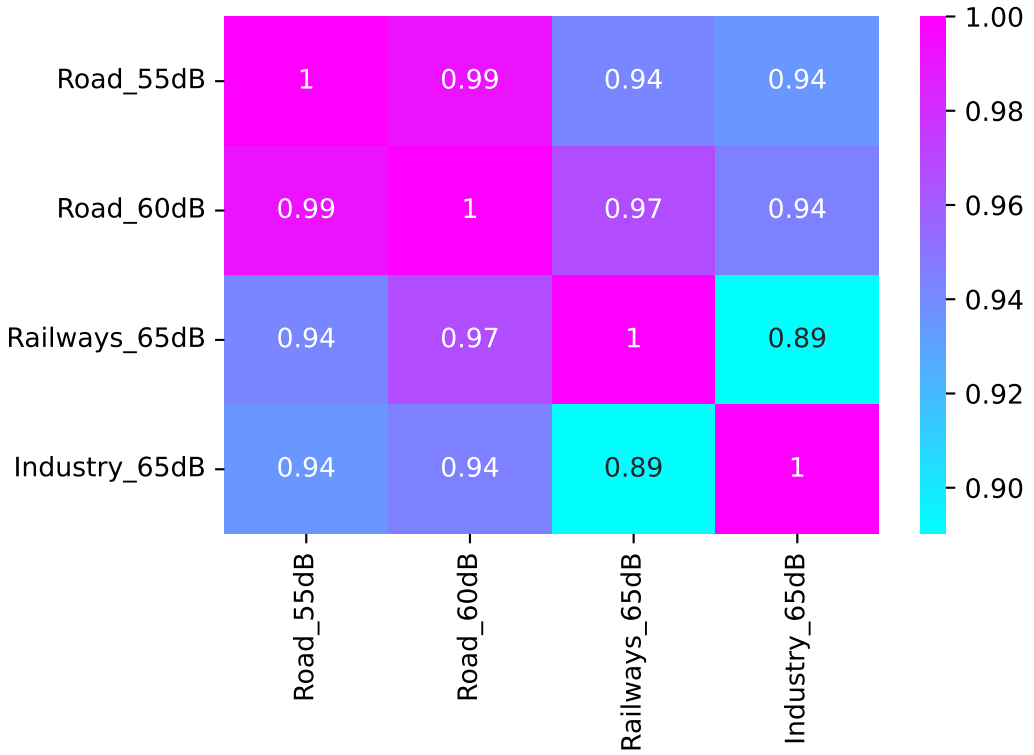
Mirando si hay relación entre las variables.

```
# Importando libreria necesaria
import seaborn as sns
sns.pairplot(ruidoso)
plt.show()
```



Análisis de correlación

```
# Calcular la matriz de correlación
corr_matrix = ruidoso.corr()

# Visualizar la matriz de correlación usando un mapa de calor
sns.heatmap(corr_matrix, annot=True, cmap='cool')
plt.show()
```



Miramos que si hay una posible relación entre las variables mirando el gráfico de pares, además, la matriz de correlación nos confirma esto, puesto que se observa la intensidad de los colores en el mapa de calor, y las correlaciones están cercanas a 1 lo que indica una relación lineal positiva muy fuerte entre las variables. Por ejemplo, las dos fuentes de ruido (Carreteras y ferrocarriles) tienen una correlación cercana a 1, lo que indica que sus niveles de ruido tienden a aumentar.

c. Verifique si existen problemas de datos atípicos en cada una de las variables usando las metodologías de detección a nivel univariado.

Visión general de la distribución de los datos

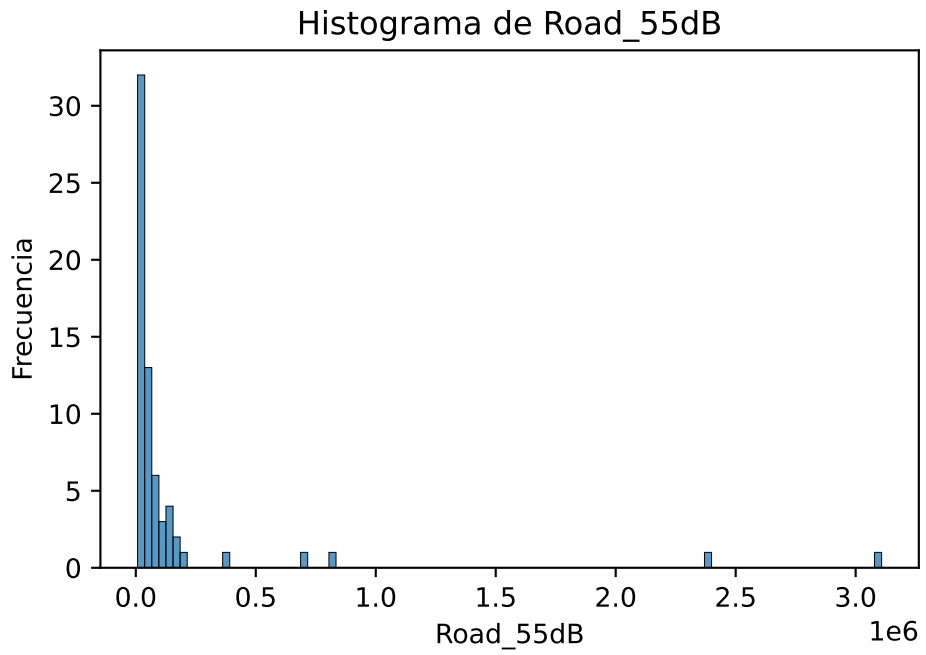
```
# Visión general de la distribución de los datos
ruidoso.describe()
```

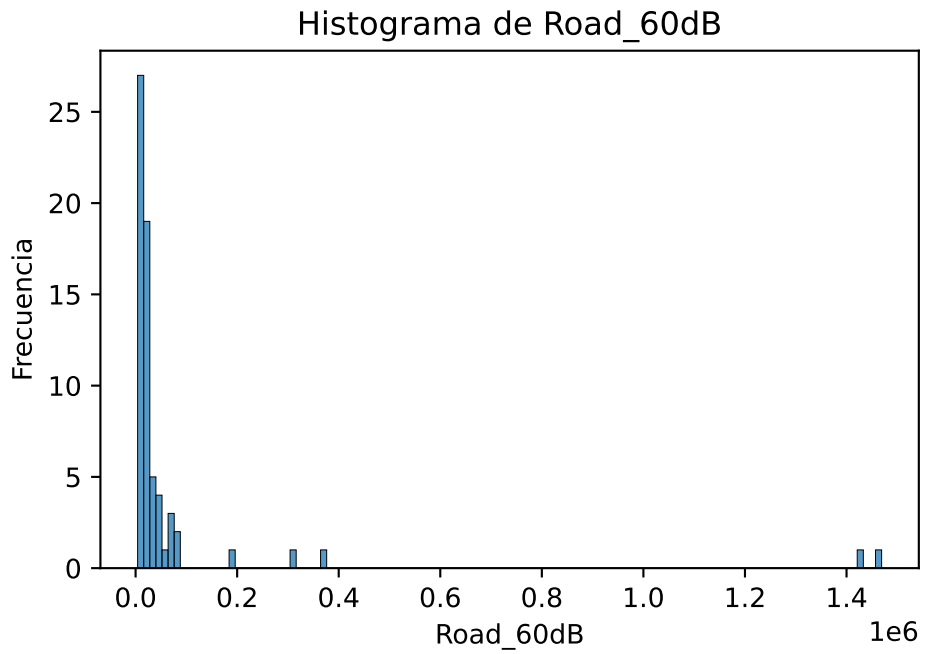
	Road_55dB	Road_60dB	Railways_65dB	Industry_65dB
count	6.600000e+01	6.600000e+01	66.000000	65.000000
mean	1.592288e+05	7.858788e+04	1139.393939	180.000000
std	4.847513e+05	2.514090e+05	4778.827625	537.993959
min	7.600000e+03	4.000000e+03	0.000000	0.000000
25%	1.895000e+04	1.007500e+04	0.000000	0.000000
50%	3.755000e+04	1.740000e+04	100.000000	0.000000
75%	7.890000e+04	3.450000e+04	400.000000	100.000000
max	3.108200e+06	1.469100e+06	29700.000000	4000.000000

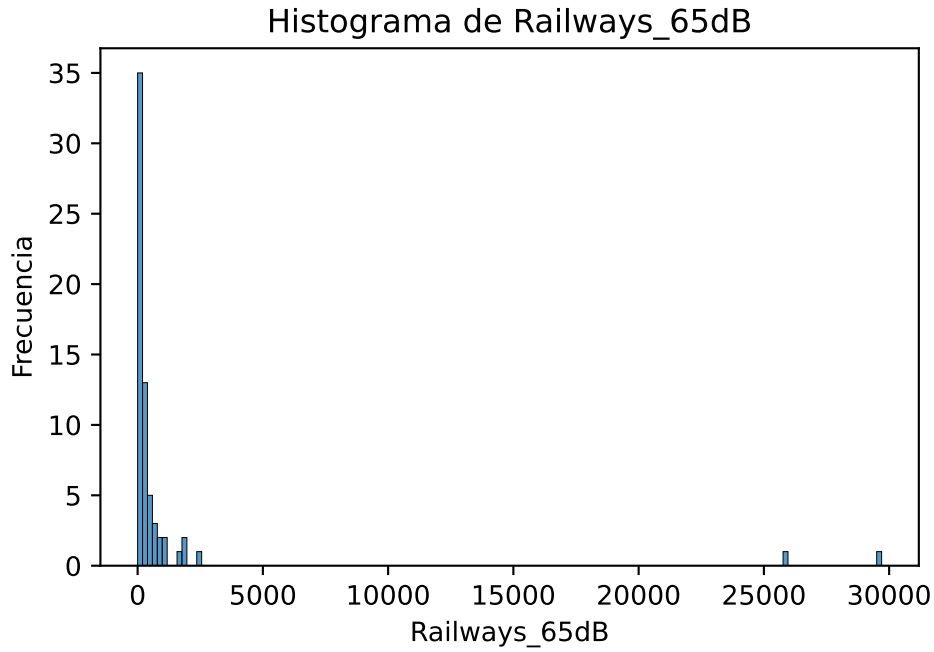
Vemos diferencias notables entre los valores mínimo y máximo de cada variable, así como también notables diferencias entre la media y la mediana, lo que nos da indicios de posibles valores atípicos.

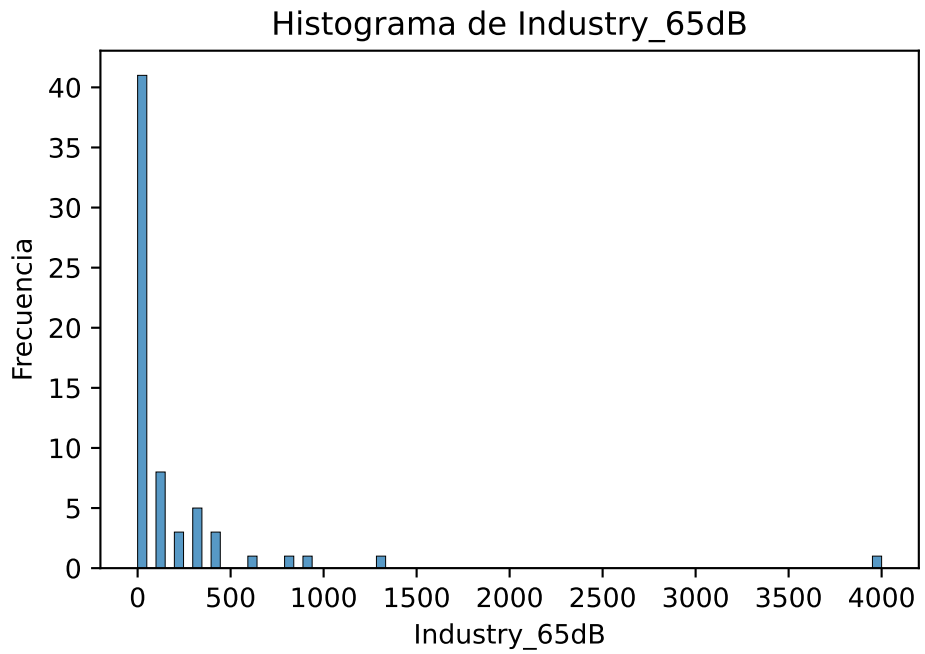
Inspección visual de outliers

```
# Obtener la lista de todas las columnas en el dataset
columnas = ruidoso.columns[0:]
# Generar un histograma para cada variable en el dataset
for columna in columnas:
    plt.figure()
    sns.histplot(data=ruidoso, x=columna)
    plt.title(f'Histograma de {columna}')
    plt.xlabel(columna)
    plt.ylabel('Frecuencia')
    plt.show()
```

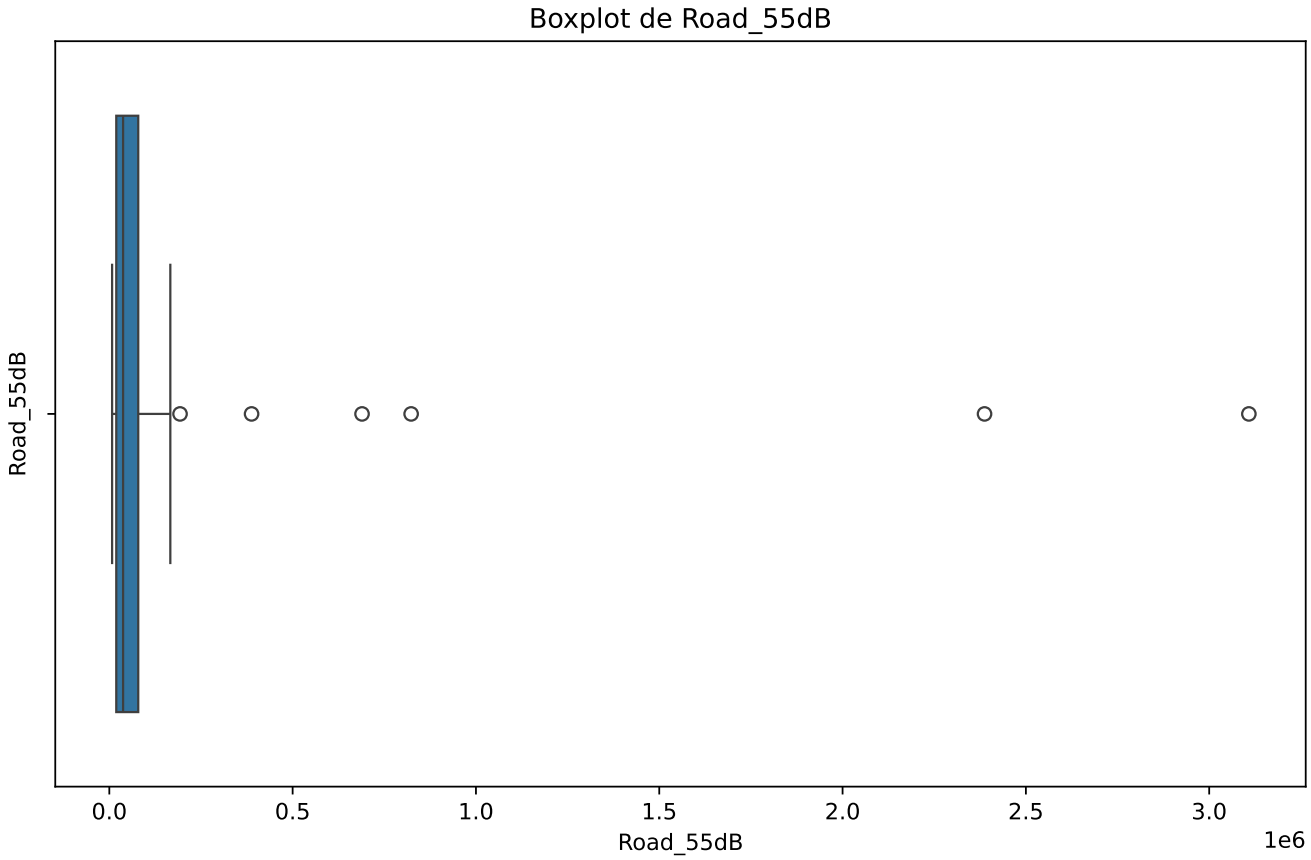


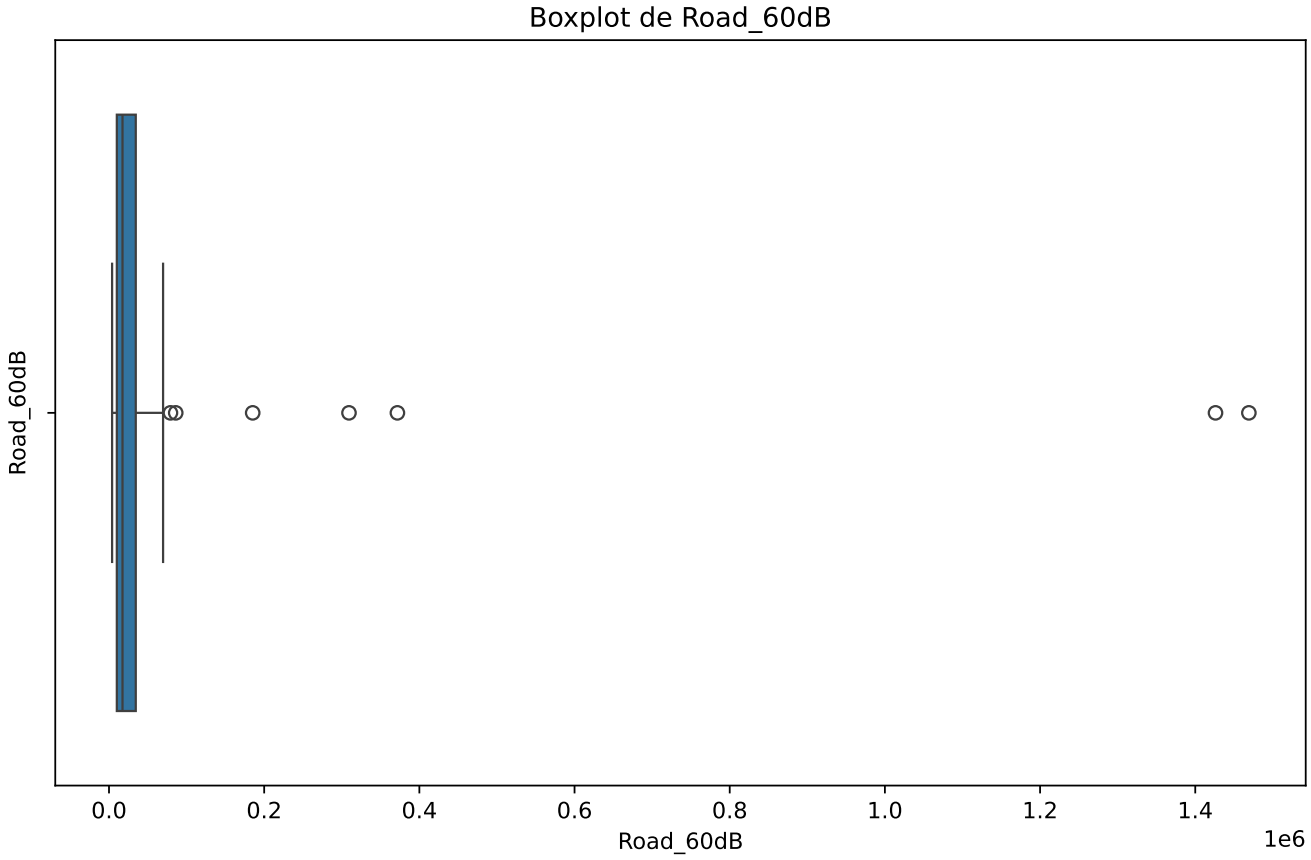

Histograma de Road_60dB
Frecuencia
0
5
10
15
20
25
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
1e6
Road_60dB

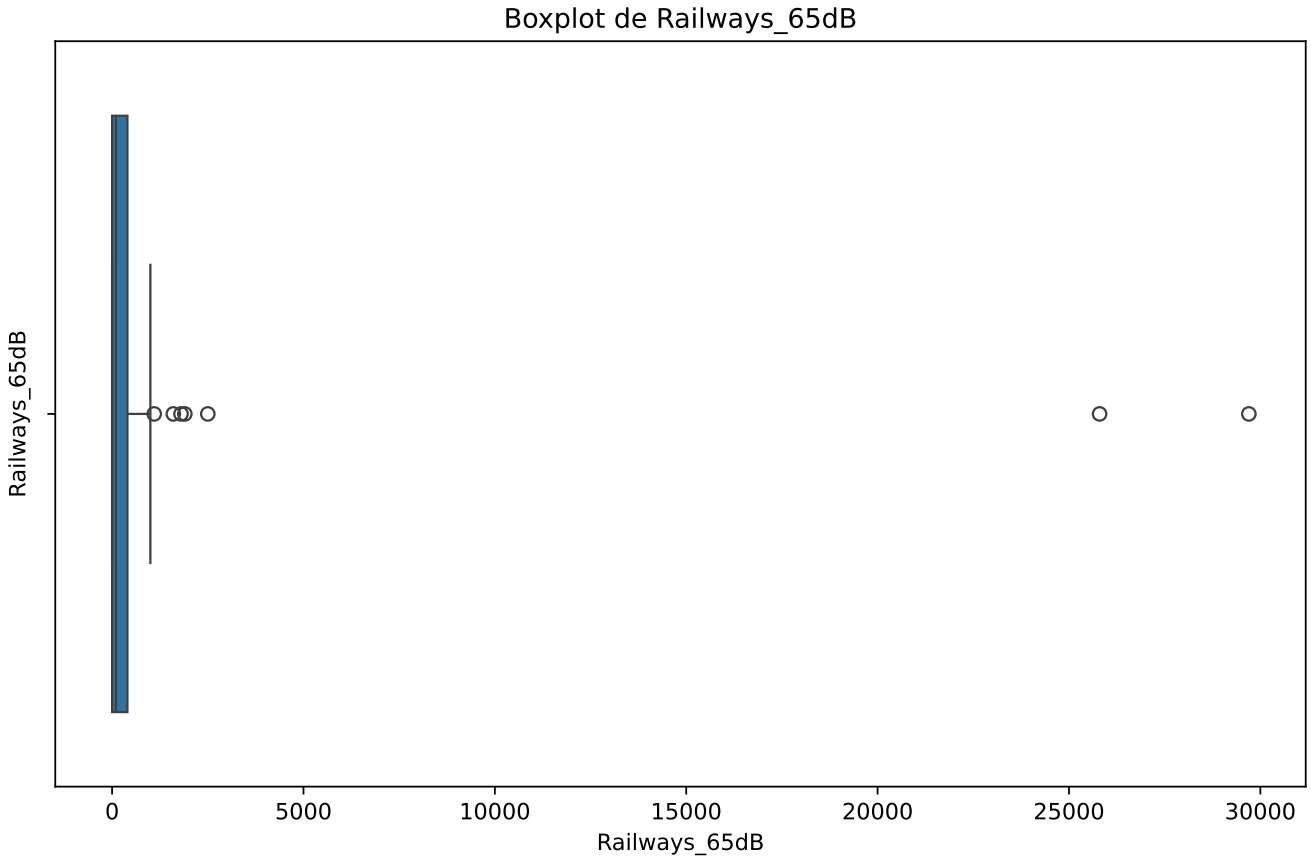

Histograma de Railways_65dB
Frecuencia
0
5
10
15
20
25
30
35
0
5000
10000
15000
20000
25000
30000
Railways_65dB

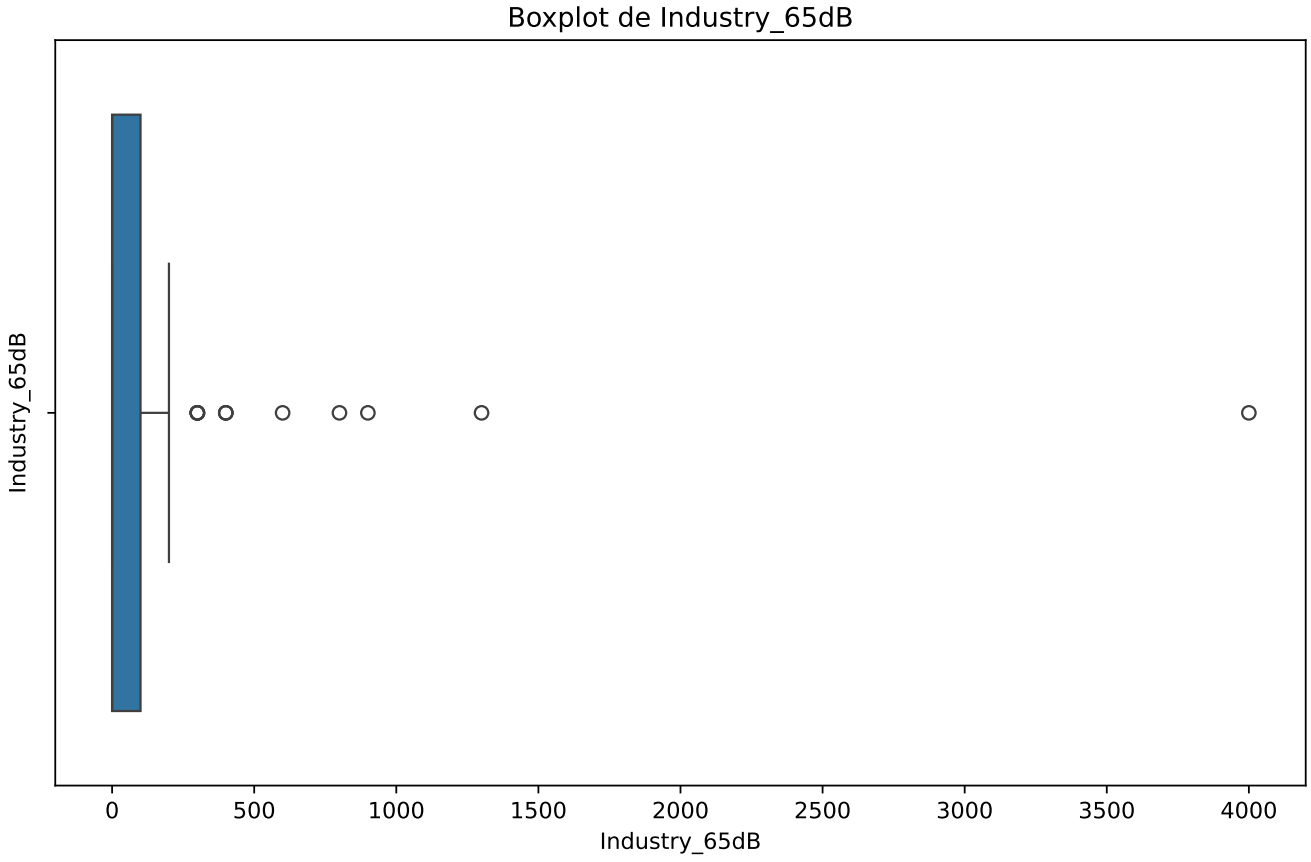



```
# Obtener la lista de todas las columnas en el dataset
columnas = ruidoso.columns[0:]
# Generar un boxplot para cada variable en el dataset
for columna in columnas:
    plt.figure(figsize=(10, 6))
    sns.boxplot(data=ruidoso, x=columna)
    plt.title(f'Boxplot de {columna}')
    plt.ylabel(columna)
    plt.show()
```


Boxplot de Road_55dB
Road_55dB
0.0
0.5
1.0
1.5
2.0
2.5
3.0
Road_55dB
1e6


Boxplot de Road_60dB
Road_60dB
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
Road_60dB
1e6


Boxplot de Railways_65dB
Railways_65dB
0
5000
10000
15000
20000
25000
30000
Railways_65dB


Boxplot de Industry_65dB
Industry_65dB
0
500
1000
1500
2000
2500
3000
3500
4000
Industry_65dB

Comentarios: De los gráficos (histograma y boxplot) podemos observar que las 4 variables presentan outliers, ya que hay valores muy alejados de la distribución en el histograma y hay observaciones por fuera de los bigotes de las cajas, pero acompañaremos estos hallazgos visuales con técnicas adicionales para detectar esos valores atípicos.

Inspección de outliers

- Z-Score

Un Z-Score, es una medida estadística que indica a cuántas desviaciones estándar un punto de datos específico está por encima o por debajo de la media del conjunto de datos.

En este sentido, podemos calcular la puntuación Z de cada dato usando la función zscore de scipy y compararla con un umbral para determinar qué valores son considerados atípicos. Por lo general se establece un umbral de 3, por lo que aquellos puntos de datos cuya puntuación Z absoluta sea superior a 3 son outliers.

```
# Analisis Zscore para cada variable
from scipy.stats import zscore
# Calculo de z-score para cada punto de datos y su valor absoluto
z_scores = zscore(ruidoso["Road_55dB"])
# z_scores = zscore(ruidoso["Road_60dB"])
# z_scores = zscore(ruidoso["Railways_65dB"])
# z_scores = zscore(ruidoso["Industry_65dB"])

abs_z_scores = np.abs(z_scores)
# Seleccionar los valores atípicos usando un umbral de 3
outliers = ruidoso[abs_z_scores > 3.5]
print(outliers)

# Cantidad de valores atipicos
print(f"Numero de valores atípicos: {len(outliers)}")
```

```
    Road_55dB  Road_60dB  Railways_65dB  Industry_65dB
49    2387200    1426100          29700         4000.0
66    3108200    1469100          25800            NaN
Numero de valores atípicos: 2
```

```
z_scores = zscore(ruidoso["Industry_65dB"])

abs_z_scores = np.abs(z_scores)
# Seleccionar los valores atípicos usando un umbral de 3

outliers = ruidoso[abs_z_scores > 3.5]
print(outliers)
# Cantidad de valores atipicos
print(f"Numero de valores atípicos: {len(outliers)}")
```

```
Empty DataFrame
Columns: [Road_55dB, Road_60dB, Railways_65dB, Industry_65dB]
Index: []
Numero de valores atípicos: 0
```

Con este método nos da 2 posibles valores atípicos para todas las variables, excepto para "Industry_65dB" puesto que no detecta ningún valor atípico. Pero sabemos que la regla empírica en la cual se basa el z-score funciona en distribuciones normales, y como vimos en los histogramas que los valores para cada variable tienen una cola pesada a la derecha, este método no es adecuado para este conjunto de datos.

- Z-Score Modificado

Cuando los datos son asimétricos o no se distribuyen de forma normal podemos utilizar el z-score modificado, también conocido como MAD-Z-Score. Este, a diferencia del z-score, utiliza la mediana y la desviación absoluta mediana (MAD en inglés) en lugar de la media y la desviación estándar con el fin de evitar el efecto de los outliers sobre estas dos últimas medidas.

La fórmula está dada por:

$$M_i = \frac{0.6745 * (x_i - Mediana)}{MAD}$$

La biblioteca PyOD, nos ayuda a realizar el cálculo anterior, así:

```
# Z-score modifocado
# Crear una lista de columnas del DataFrame
columnas = ruidoso.columns[0:]
# Crear un diccionario para almacenar los resultados de MAD por cada columna
mad_resultados = {}
# Iterar sobre cada columna y aplicar el estimador MAD
for columna in columnas:
    datos_columna = ruidoso[columna].values.reshape(-1, 1)
# Inicializar y ajustar el estimador MAD con un umbral de 3.5
    mad = MAD(threshold=3.5)
    labels = mad.fit(datos_columna).labels_
# Almacenar los resultados de MAD para la columna actual en el diccionario
    mad_resultados[columna] = labels
# Mostrar los resultados de MAD para cada columna
for columna, labels in mad_resultados.items():
    print(f'Resultados de MAD para {columna}:')
    print(labels)
    print(f"Valores atípicos: {labels.sum()}")
    print()
```

```
Resultados de MAD para Road_55dB:
[1 0 0 0 0 0 0 1 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1]
Valores atípicos: 7

Resultados de MAD para Road_60dB:
[1 0 0 0 0 0 0 1 0 0 0 0 1 0 1 0 0 1 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0
```

```
 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1]
Valores atípicos: 10

Resultados de MAD para Railways_65dB:
[1 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 0 0 0 0 0
 0 0 0 1 0 0 0 0 0 0 0 1 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1]
Valores atípicos: 11

Resultados de MAD para Industry_65dB:
[0 0 0 1 1 1 1 1 1 1 0 1 1 0 1 0 0 1 1 1 1 0 0 0 0 1 1 0 1 0 0 1 0 0 0 0 0
 1 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 1 0 0]
Valores atípicos: 24
```

Nota: con el método 'fit' junto con el atributo 'labels_' podemos acceder a las etiquetas binarias generadas por el modelo, donde el estimador devuelve 0 para los (valores no-atípicos) y 1 para los valores atípicos.

Ahora, podemos filtrar el dataset usando las etiquetas para obtener todos los registros con valores atípicos en las variables según el z-score modificado.

```python
# Mostrar los resultados de MAD para cada columna
for columna, labels in mad_resultados.items():
    print(f"Valores atípicos: {labels.sum()}")
    print()
    outliers = ruidoso[labels==1]
    print(outliers)
```

```
Valores atípicos: 7

    Road_55dB  Road_60dB  Railways_65dB  Industry_65dB
1      166400      79200           1100            0.0
8      388000     185200            300         1300.0
13     192900      86000            100          900.0
15     823200     371700            300          800.0
29     689300     309300           1600          600.0
49    2387200    1426100          29700         4000.0
66    3108200    1469100          25800            NaN
Valores atípicos: 10

    Road_55dB  Road_60dB  Railways_65dB  Industry_65dB
1      166400      79200           1100            0.0
8      388000     185200            300         1300.0
13     192900      86000            100          900.0
15     823200     371700            300          800.0
18     140900      66200            100          300.0
27     156000      67300            300          200.0
29     689300     309300           1600          600.0
45     152500      69700            200          100.0
49    2387200    1426100          29700         4000.0
66    3108200    1469100          25800            NaN
Valores atípicos: 11
```

```
    Road_55dB  Road_60dB  Railways_65dB  Industry_65dB
1      166400      79200           1100            0.0
5      103100      33500           1900          200.0
11      57900      23300            700            0.0
29     689300     309300           1600          600.0
30       7600       4000            800            0.0
32     102800      41600           1000          400.0
41      47500      17400           1800            0.0
49    2387200    1426100          29700         4000.0
50      77200      24300           2500            0.0
56      64300      25100            900            0.0
66    3108200    1469100          25800            NaN
Valores atípicos: 24

    Road_55dB  Road_60dB  Railways_65dB  Industry_65dB
4       24500      16700              0          100.0
5      103100      33500           1900          200.0
6       36800      16100              0          100.0
7       39700      16800              0          300.0
8      388000     185200            300         1300.0
9       15000       9400            300          200.0
10      18600      12000              0          100.0
12      92700      33600              0          100.0
13     192900      86000            100          900.0
15     823200     371700            300          800.0
18     140900      66200            100          300.0
19      38000      13400            100          300.0
20      14100      10000              0          300.0
21      64800      36400              0          400.0
26      32400      19700            300          400.0
27     156000      67300            300          200.0
29     689300     309300           1600          600.0
32     102800      41600           1000          400.0
38      31600      10700            200          100.0
45     152500      69700            200          100.0
49    2387200    1426100          29700         4000.0
54      11600       8300            100          100.0
58      49700      32900              0          100.0
64      24500      12400            300          300.0
```

En este caso hemos identificado más valores atípicos en cada variable que con el método del z-score estándar.

- Rango Intercuartil (IQR)

Otro método que podemos emplear si nuestros datos no siguen la distribución normal es el método del rango intercuartílico (IQR).

El IQR es una medida de dispersión que indica la distancia entre el primer cuartil (percentil 25) y el tercer cuartil (percentil 75) de una distribución. Puede utilizarse para identificar outliers al ayudar a definir los

límites a partir de los cuales los valores se consideran atípicos. Estos límites se calculan de la siguiente manera:

Límite inferior = Q1 — (k * IQR)

Límite superior = Q3 + (k * IQR)

Podemos calcular los límites con la función 'quantile' y usarlos para filtrar el dataset:

```
# Obtener la lista de todas las columnas en el dataset
columnas = ruidoso.columns[0:]

for columna in columnas:
    # Cálculo del rango intercuartílico (IQR)
    Q1 = ruidoso[columna].quantile(0.25)
    Q3 = ruidoso[columna].quantile(0.75)
    IQR = Q3 - Q1
    # Identificar valores atípicos
    outliers = ruidoso[(ruidoso[columna] < (Q1 - 1.5 * IQR)) | (ruidoso[columna] > (Q3 + 1.5 * IQ
    print(f'Posibles valores atípicos en {columna:}')
    print(f'Numero de valores atípicos:{len(outliers)}')
    print(outliers)
```

```
Posibles valores atípicos en Road_55dB
Numero de valores atípicos:6
    Road_55dB  Road_60dB  Railways_65dB  Industry_65dB
8      388000     185200            300         1300.0
13     192900      86000            100          900.0
15     823200     371700            300          800.0
29     689300     309300           1600          600.0
49    2387200    1426100          29700         4000.0
66    3108200    1469100          25800            NaN
Posibles valores atípicos en Road_60dB
Numero de valores atípicos:7
    Road_55dB  Road_60dB  Railways_65dB  Industry_65dB
1      166400      79200           1100            0.0
8      388000     185200            300         1300.0
13     192900      86000            100          900.0
15     823200     371700            300          800.0
29     689300     309300           1600          600.0
49    2387200    1426100          29700         4000.0
66    3108200    1469100          25800            NaN
Posibles valores atípicos en Railways_65dB
Numero de valores atípicos:7
    Road_55dB  Road_60dB  Railways_65dB  Industry_65dB
1      166400      79200           1100            0.0
5      103100      33500           1900          200.0
29     689300     309300           1600          600.0
41      47500      17400           1800            0.0
49    2387200    1426100          29700         4000.0
50      77200      24300           2500            0.0
```

```
66    3108200    1469100          25800            NaN
Posibles valores atípicos en Industry_65dB
Numero de valores atípicos:13
    Road_55dB  Road_60dB  Railways_65dB  Industry_65dB
7       39700      16800              0          300.0
8      388000     185200            300         1300.0
13     192900      86000            100          900.0
15     823200     371700            300          800.0
18     140900      66200            100          300.0
19      38000      13400            100          300.0
20      14100      10000              0          300.0
21      64800      36400              0          400.0
26      32400      19700            300          400.0
29     689300     309300           1600          600.0
32     102800      41600           1000          400.0
49    2387200    1426100          29700         4000.0
64      24500      12400            300          300.0
```

Con este método encontramos menos datos atípicos que con el Z-score modificado, pero menos que con el Z-score estándar. Por lo tanto, debemos decidir como tratar los outliers identificados en cada variable.

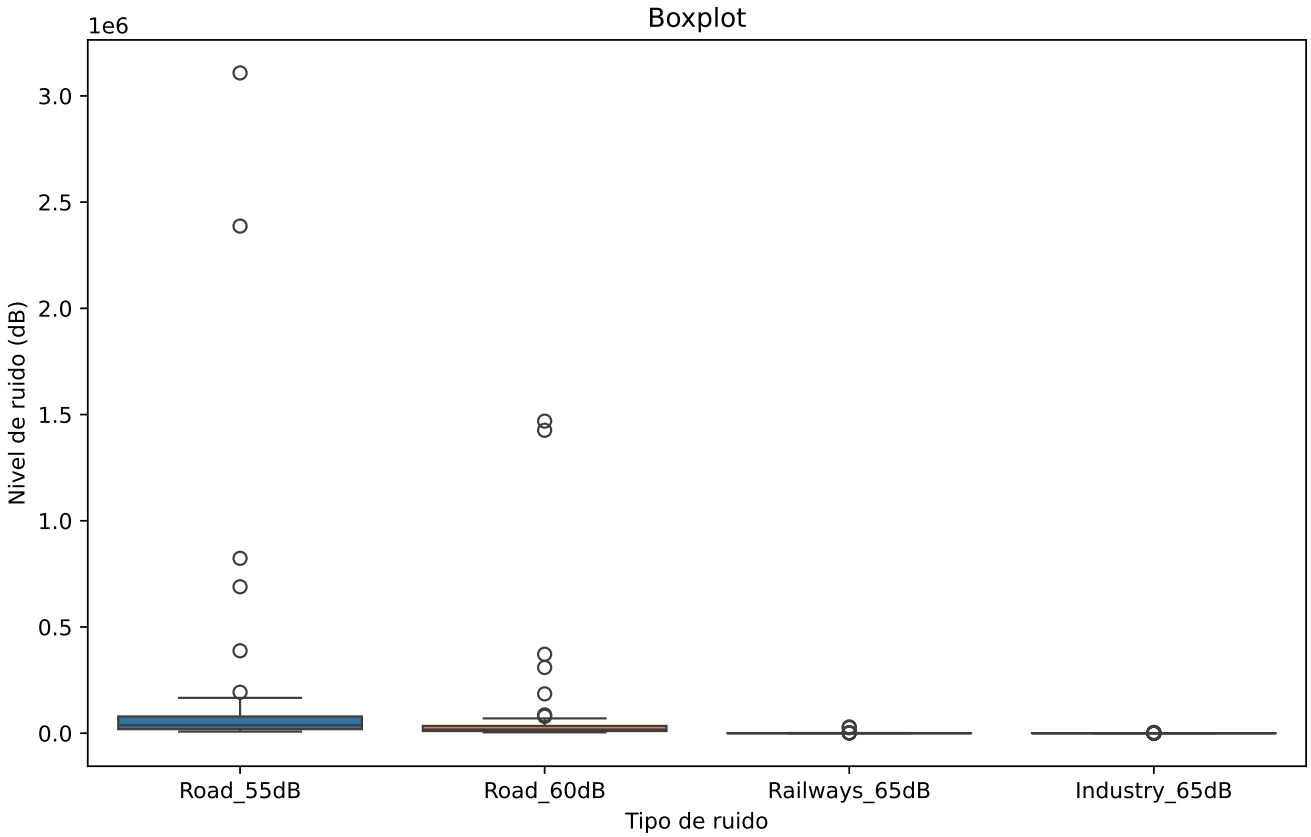
d. ¿Se detectan valores atípicos a nivel multivariado?

- Distancias de Mahalanobis

Es una medida de distancia entre el punto $\vec{x} = (x_1, x_2, x_3)$ y el vector de medias $\vec{\mu} = (\mu_1, \mu_2, \mu_3)$, teniendo en cuenta la matriz de varianzas y covarianzas S. La distancia de Mahalanobis se define como:

$$D_M(\vec{x}) = \sqrt{(\vec{x} - \vec{\mu})^T S^{-1} (\vec{x} - \vec{\mu})}$$

```python
# Seleccionamos solo las columnas de interés
df_selected = ruidoso[['Road_55dB', 'Road_60dB', 'Railways_65dB', 'Industry_65dB']]

# Creamos el gráfico de caja
plt.figure(figsize=(10, 6))
sns.boxplot(data=df_selected)
plt.title('Boxplot')
plt.xlabel('Tipo de ruido')
plt.ylabel('Nivel de ruido (dB)')
plt.show()
```




```
from scipy.spatial import distance
copia = ruidoso.copy()
# Calculamos la matriz de covarianza de las variables
cov_matrix = np.cov(df_selected, rowvar=False)

# Calculamos la inversa de la matriz de covarianza
cov_inv = np.linalg.inv(cov_matrix)

# Calculamos la distancia de Mahalanobis para cada fila en el DataFrame
dist_mahalanobis = []
for row in df_selected.values:
    dist = distance.mahalanobis(row, df_selected.mean(), cov_inv)
    dist_mahalanobis.append(dist)

# Agregamos la distancia de Mahalanobis como una nueva columna en el DataFrame
copia['mahalanobis_distance'] = dist_mahalanobis

# Mostramos el DataFrame con la distancia de Mahalanobis
print(copia)
```

```
    Road_55dB  Road_60dB  Railways_65dB  Industry_65dB  mahalanobis_distance
1      166400      79200           1100            0.0                   NaN
2       20000      11000              0            0.0                   NaN
3       37800      18000              0            0.0                   NaN
4       24500      16700              0          100.0                   NaN
```

```
5      103100      33500           1900          200.0                   NaN
..        ...        ...            ...            ...                   ...
62      20700      16200              0            0.0                   NaN
63      15100       9900              0            0.0                   NaN
64      24500      12400            300          300.0                   NaN
65      12800       8500              0            0.0                   NaN
66    3108200    1469100          25800            NaN                   NaN

[66 rows x 5 columns]
```

Hay un posible error en la información que impide calcular las distancias, por lo que analizaremos que puede estar produciendo esto:

```
contar_faltantes = df_selected.isna().sum()

total = np.product(df_selected.shape)
total_faltantes = contar_faltantes.sum()

# Porcentale de datos faltantes
(total_faltantes/total) * 100
```

```
0.3787878787878788
```

Como solo hay aproximadamente un 0.37% de datos faltantes, concentrados en una sola variable, y además, del estudio de datos atípicos univariados analizados anteriormente, en la mayoria nos dió que la observación 66 de las variables eran datos atípicos, decidimos entonces eliminar el registro 66 del conjunto de datos.

```
ruidoso_copia = ruidoso.copy()
# Eliminar la fila del DataFrame copia
ruidoso_copia = ruidoso_copia.drop(index=66)

# Mostrar el DataFrame actualizado
print(ruidoso_copia)
```

```
    Road_55dB  Road_60dB  Railways_65dB  Industry_65dB
1      166400      79200           1100            0.0
2       20000      11000              0            0.0
3       37800      18000              0            0.0
4       24500      16700              0          100.0
5      103100      33500           1900          200.0
..        ...        ...            ...            ...
61      74400      47000              0            0.0
62      20700      16200              0            0.0
63      15100       9900              0            0.0
64      24500      12400            300          300.0
65      12800       8500              0            0.0

[65 rows x 4 columns]
```

Ahora, volvamos a calcular las distancias de Mahalanobis:

```
from scipy.spatial import distance
# Calculamos la matriz de covarianza de las variables
cov_matrix = np.cov(ruidoso_copia, rowvar=False)

# Calculamos la inversa de la matriz de covarianza
cov_inv = np.linalg.inv(cov_matrix)

# Calculamos la distancia de Mahalanobis para cada fila en el DataFrame
dist_mahalanobis = []
for row in ruidoso_copia.values:
    dist = distance.mahalanobis(row, ruidoso_copia.mean(), cov_inv)
    dist_mahalanobis.append(dist)

# Agregamos la distancia de Mahalanobis como una nueva columna en el DataFrame
ruidoso_copia['mahalanobis_distance'] = dist_mahalanobis
```

```
# Ordenamos el DataFrame por la columna de distancia de Mahalanobis en orden descendente
df_sorted = ruidoso_copia.sort_values(by='mahalanobis_distance', ascending=False)

# Seleccionamos los registros con las distancias más grandes
dist_mas_grandes = df_sorted.head(3)
dist_mas_grandes
```

	Road_55dB	Road_60dB	Railways_65dB	Industry_65dB	mahalanobis_distance
49	2387200	1426100	29700	4000.0	7.921944
15	823200	371700	300	800.0	5.798900
8	388000	185200	300	1300.0	4.968816

- Local Outlier Factor:

Local outlier factor es un método basado en **densidad** que utiliza la búsqueda de vecinos más cercanos. Ese método calcula los **scores** para cada uno de los puntos a partir de la tasa promedio de densidad de los puntos vecinos con respecto a si mismo.

```
from sklearn.neighbors import LocalOutlierFactor
# Calculamos el LOF para las 4 variables
lof = LocalOutlierFactor(novelty=False)
lof.fit_predict(ruidoso_copia)

# Obtenemos los scores LOF para cada punto
scores_lof = -lof.negative_outlier_factor_

# Agregamos los scores LOF como una nueva columna en el DataFrame
ruidoso_copia['lof_score'] = scores_lof

# Mostramos el DataFrame con los scores LOF
print(ruidoso_copia)
```

```
    Road_55dB  Road_60dB  Railways_65dB  Industry_65dB  mahalanobis_distance  \
1      166400      79200           1100            0.0              1.429036
2       20000      11000              0            0.0              0.750562
3       37800      18000              0            0.0              0.689198
4       24500      16700              0          100.0              0.983754
5      103100      33500           1900          200.0              3.648459
..        ...        ...            ...            ...                   ...
61      74400      47000              0            0.0              2.121306
62      20700      16200              0            0.0              1.253857
63      15100       9900              0            0.0              0.856649
64      24500      12400            300          300.0              1.441410
65      12800       8500              0            0.0              0.820636

    lof_score
1    2.209951
2    0.959957
3    1.170028
4    1.041296
5    1.626144
..        ...
61   1.594346
62   0.962463
63   0.987861
64   0.984511
65   1.003104

[65 rows x 6 columns]
```

```
# Ordenamos el DataFrame por la columna 'lof_score' en orden descendente
df_sorted = ruidoso_copia.sort_values(by='lof_score', ascending=False)

# Seleccionamos los primeros tres registros con los scores LOF más grandes
primeros_tres_lof = df_sorted.head(3)
primeros_tres_lof
```

	Road_55dB	Road_60dB	Railways_65dB	Industry_65dB	mahalanobis_distance	lof_score
49	2387200	1426100	29700	4000.0	7.921944	37.888661
15	823200	371700	300	800.0	5.798900	12.333829
29	689300	309300	1600	600.0	4.533022	10.208054

- Isolation Forest:

Las instancias de datos anómalos se pueden aislar de los datos normales mediante la partición recursiva del conjunto de datos.

```
from sklearn.ensemble import IsolationForest

# Definimos la cantidad de árboles
```

```
n_arboles = 250

# Inicializamos y ajustamos el modelo Isolation Forest con la cantidad de árboles especificada
isolation_forest = IsolationForest(n_estimators=n_arboles, random_state=42)
puntajes_anomalia = isolation_forest.fit(ruidoso_copia)

# Obtenemos los puntajes de anomalía para cada muestra
puntajes_anomalia = isolation_forest.score_samples(ruidoso_copia)

# Convertimos los puntajes de anomalía a valores positivos
puntajes_anomalia = -puntajes_anomalia

# Agregamos las etiquetas de anomalía como una nueva columna en el DataFrame
ruidoso_copia['puntajes_anomalia'] = puntajes_anomalia
```

```
# Ordenamos el DataFrame por los puntajes de anomalía en orden descendente
df_sorted = ruidoso_copia.sort_values(by='puntajes_anomalia', ascending=False)

# Seleccionamos los primeros tres registros con los puntajes de anomalía más altos
primeros_tres_anomalies = df_sorted.head(3)
primeros_tres_anomalies
```

	Road_55dB	Road_60dB	Railways_65dB	Industry_65dB	mahalanobis_distance	lof_score	puntajes_
49	2387200	1426100	29700	4000.0	7.921944	37.888661	0.878678
15	823200	371700	300	800.0	5.798900	12.333829	0.705660
29	689300	309300	1600	600.0	4.533022	10.208054	0.691057

e. Para el caso univariado, escoja una variable y realice un análisis sobre las implicaciones que tiene realizar diferentes tratamientos a los datos atípicos en la distribución de la respectiva variable.

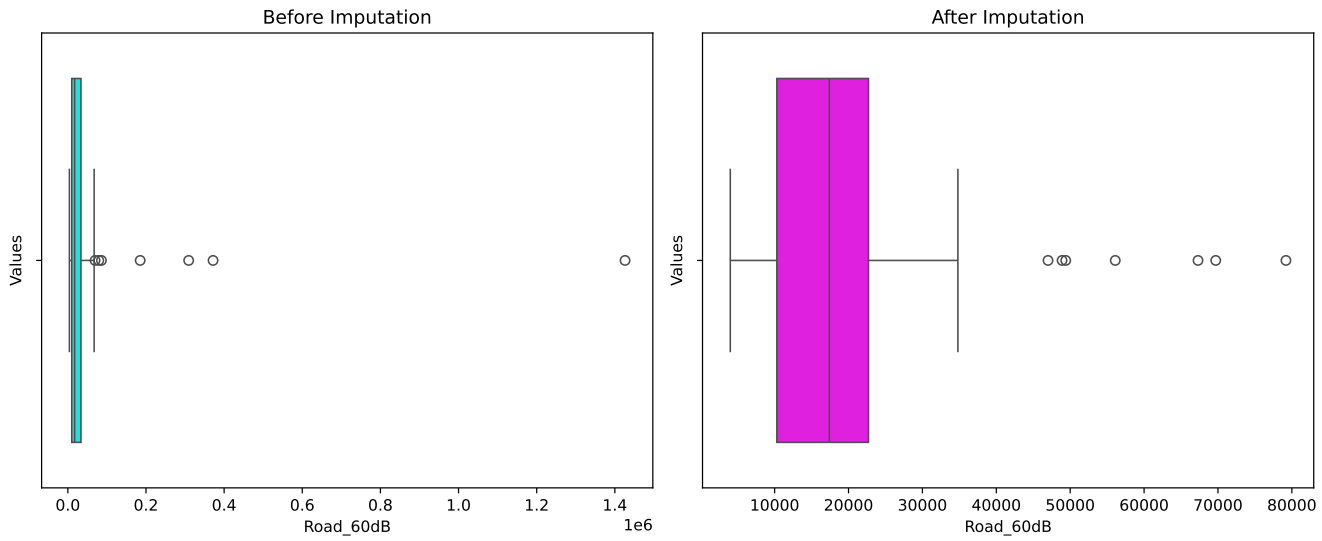
Escogimos la variable "Road_60dB"

- Imputar — queremos conservar la mayor cantidad de datos, pero eliminando el efecto de los outliers. Esto implica reemplazar los valores atípicos con otros valores como la mediana o la media.

```
# Calcular la media
median_value = ruidoso_copia['Road_60dB'].median()

# Imputar outliers con la media
ruidoso_imputed = ruidoso_copia.copy()
ruidoso_imputed.loc[outliers.index, 'Road_60dB'] = median_value
```

```
# Create a box plot
# Crear una figura y ejes para los dos gráficos
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 5))

# Crear el boxplot para 'Road_60dB' antes de imputar con la mediana
```

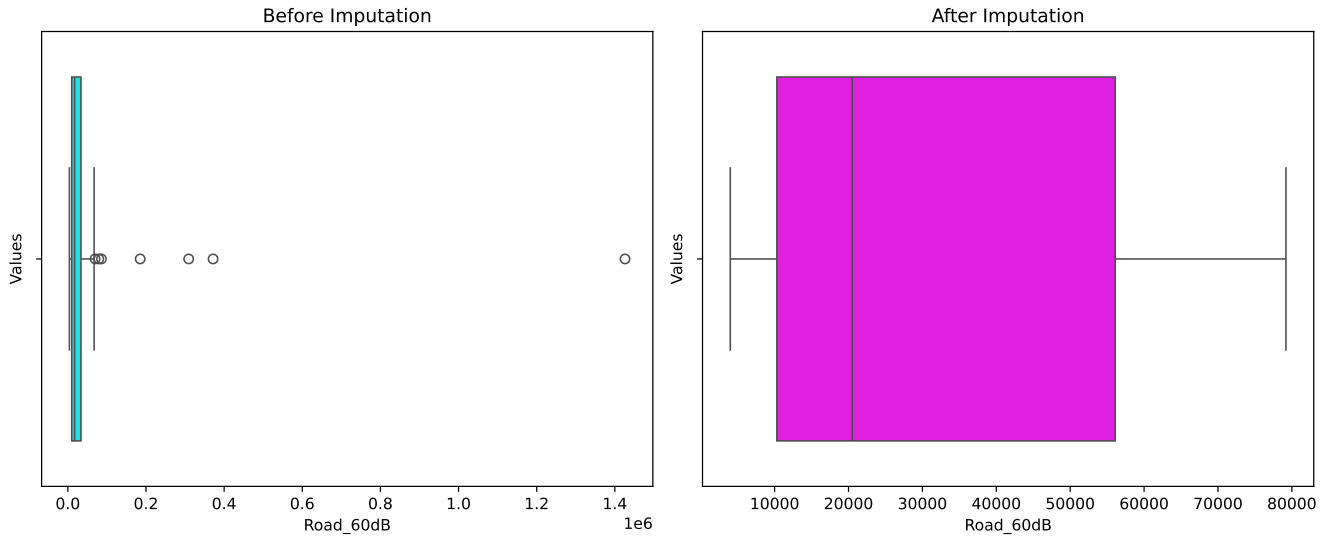
```
sns.boxplot(data=ruidoso_copia, x='Road_60dB', ax=ax1, color='cyan')

# Crear el boxplot para 'Road_60dB' después de imputar con la mediana
sns.boxplot(data=ruidoso_imputed, x='Road_60dB', ax=ax2, color='magenta')

# Establecer títulos y etiquetas de ejes
ax1.set_title('Before Imputation')
ax1.set_xlabel('Road_60dB')
ax1.set_ylabel('Values')

ax2.set_title('After Imputation')
ax2.set_xlabel('Road_60dB')
ax2.set_ylabel('Values')

# Ajustar el espacio entre los dos gráficos
plt.tight_layout()

# Mostrar los gráficos uno al lado del otro
plt.show()
```




```
# Calcular la media
mean_value = ruidoso_copia['Road_60dB'].mean()

# Imputar outliers con la media
ruidoso_imputed2 = ruidoso_copia.copy()
ruidoso_imputed2.loc[outliers.index, 'Road_60dB'] = mean_value
```

```
# Create a box plot
# Crear una figura y ejes para los dos gráficos
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 5))

# Crear el boxplot para 'Road_60dB' antes de imputar con la mediana
sns.boxplot(data=ruidoso_copia, x='Road_60dB', ax=ax1, color='cyan')
```

```
# Crear el boxplot para 'Road_60dB' después de imputar con la mediana
sns.boxplot(data=ruidoso_imputed2, x='Road_60dB', ax=ax2, color='magenta')

# Establecer títulos y etiquetas de ejes
ax1.set_title('Before Imputation')
ax1.set_xlabel('Road_60dB')
ax1.set_ylabel('Values')

ax2.set_title('After Imputation')
ax2.set_xlabel('Road_60dB')
ax2.set_ylabel('Values')

# Ajustar el espacio entre los dos gráficos
plt.tight_layout()

# Mostrar los gráficos uno al lado del otro
plt.show()
```

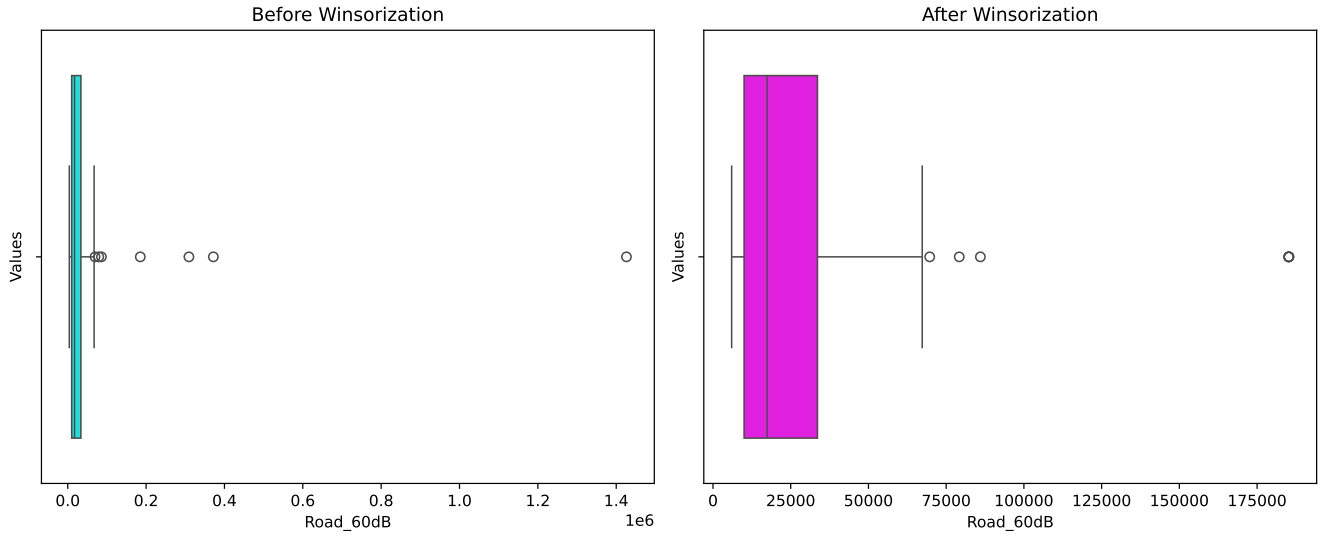


- **Winzorizar** — La winsorización es una técnica que reemplaza los valores atípicos por el valor más cercano que no se considera un outlier según ciertos criterios.

```
from scipy.stats.mstats import winsorize

ruidoso_winsorized = ruidoso_copia.copy()
ruidoso_winsorized['Road_60dB'] = winsorize(ruidoso_winsorized['Road_60dB'],\
  limits = [0.05, 0.05], inplace = True)
```

limits = [0.05, 0.05]: Establece los límites de winsorización para el 5% de los valores en ambos extremos de la distribución. Es decir, el 5% de los valores más bajos y el 5% de los valores más altos serán reemplazados por los valores en el percentil 5 y el percentil 95, respectivamente.

```
# Create a box plot
# Crear una figura y ejes para los dos gráficos
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 5))

# Crear el boxplot para 'Road_60dB' antes de winsorization
sns.boxplot(data=ruidoso_copia, x='Road_60dB', ax=ax1, color='cyan')

# Crear el boxplot para 'Road_60dB' después de winsorization
sns.boxplot(data=ruidoso_winsorized, x='Road_60dB', ax=ax2, color='magenta')

# Establecer títulos y etiquetas de ejes
ax1.set_title('Before Winsorization')
ax1.set_xlabel('Road_60dB')
ax1.set_ylabel('Values')

ax2.set_title('After Winsorization')
ax2.set_xlabel('Road_60dB')
ax2.set_ylabel('Values')

# Ajustar el espacio entre los dos gráficos
plt.tight_layout()

# Mostrar los gráficos uno al lado del otro
plt.show()
```



El análisis de los diferentes tratamientos de datos atípicos en la distribución de la variable 'Road_60dB' puede ofrecer una visión importante sobre cómo estas acciones afectan la interpretación de los datos y los resultados obtenidos en un estudio o análisis. Aquí hay algunas implicaciones clave a considerar:

1. **Eliminación de datos atípicos**:
    - Si se eliminan los datos atípicos de la variable 'Road_60dB', la distribución de los datos restantes podría cambiar significativamente. Esto puede afectar la media, la mediana y otros estadísticos descriptivos de la variable.

- La eliminación de datos atípicos puede tener un impacto en el análisis de correlación y en la identificación de relaciones entre 'Road_60dB' y otras variables. Si los datos atípicos contienen información importante o representan escenarios reales, eliminarlos podría distorsionar la verdadera naturaleza de la variable.

2. **Winsorization**:

   - Al aplicar la técnica de winsorization a los datos atípicos en 'Road_60dB', se reemplazan los valores atípicos por los valores del percentil correspondiente. Esto puede reducir el impacto de los valores extremos en las estadísticas de resumen, como la media y la desviación estándar.
   - La winsorization puede ayudar a mejorar la robustez de los análisis estadísticos y modelización, ya que reduce la influencia de los valores extremos en la distribución de la variable.

3. **Análisis de sensibilidad**:

   - Es crucial realizar un análisis de sensibilidad para evaluar cómo diferentes tratamientos de datos atípicos afectan los resultados del análisis. Esto implica comparar los resultados obtenidos antes y después de aplicar los tratamientos y evaluar las diferencias significativas.
   - También se recomienda realizar análisis robustos que sean menos sensibles a los datos atípicos, como utilizar estadísticas de resumen robustas (por ejemplo, la mediana en lugar de la media) o modelos robustos.

En resumen, el tratamiento de datos atípicos en la distribución de las variables tiene implicaciones importantes en la interpretación de los datos y los resultados del análisis. Es fundamental considerar cuidadosamente los diferentes enfoques y realizar análisis de sensibilidad para tomar decisiones informadas sobre cómo abordar los datos atípicos de manera adecuada.

2. A partir del dataset auto-mpg.data-original.txt
https://archive.ics.uci.edu/dataset/9/auto+mpg realice los siguientes análisis:

   a) Cargue y explore el dataset explicando en qué consiste y las características que posee el mismo.

   b) Realice un breve análisis exploratorio para identificar la distribución de las variables usadas en la base de datos ¿será que existe relación entre las variables?

   c) Verifique si existen datos faltantes en cada uno de las variables. ¿Cuál es la proporción de datos faltantes en la distribución de las variables?

   d) ¿Cuál cree que es el mecanismo inherente a esos datos faltantes?

   e) Aplique las técnicas de tratamiento de datos faltantes vistas en clase.

   f) Analice gráfica y analíticamente la variación en la distribución de los datos al aplicar las técnicas de imputación de datos. ¿Qué técnica afecta menos la distribución original?

Solución

Punto 1:

a) Lectura y exploración de la base de datos.

```
nombres = ['mpg', 'cylinders', 'displacement', 'horsepower', 'weight', 'acceleration',
'model_year', 'origin', 'car_name']
consumo = pd.read_csv("auto-mpg.data-original.txt", sep='\s+', header=None,
names = nombres, na_values="NA")
consumo.head().to_latex;
```

mpg	cylinders	displacement	horsepower	weight	acceleration	model_year	origin	car_name
18.0	8.0	307.0	130.0	3504.0	12.0	70.0	1.0	chevrolet chevelle malibu
15.0	8.0	350.0	165.0	3693.0	11.5	70.0	1.0	buick skylark 320
18.0	8.0	318.0	150.0	3436.0	11.0	70.0	1.0	plymouth satellite
16.0	8.0	304.0	150.0	3433.0	12.0	70.0	1.0	amc rebel sst
17.0	8.0	302.0	140.0	3449.0	10.5	70.0	1.0	ford torino

La base de datos **auto-mpg.data-original** se refieren al consumo de combustible del ciclo urbano en millas por galón, esta base cuenta con 406 registros y 9 columnas.Cada variable está descrita a continuación.

mpg: Consumo en millas por galón.
cylinders: Cilindros.
displacement: Desplazamiento.
horsepower: Caballos de fuerza.
weight: Peso
acceleration: Aceleración.
model_year: Año del modelo.
origin: Origen.
car_name: Nombre del coche.

```
consumo.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 406 entries, 0 to 405
Data columns (total 9 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   mpg           398 non-null    float64
 1   cylinders     406 non-null    float64
 2   displacement  406 non-null    float64
 3   horsepower    400 non-null    float64
 4   weight        406 non-null    float64
 5   acceleration  406 non-null    float64
 6   model_year    406 non-null    float64
 7   origin        406 non-null    float64
 8   car_name      406 non-null    object
dtypes: float64(8), object(1)
memory usage: 28.7+ KB
```

Como primera observación, tenemos que las variables **mpg** y **horsepower** presentan valores faltantes.

b) Análisis descriptivos de la base de datos.

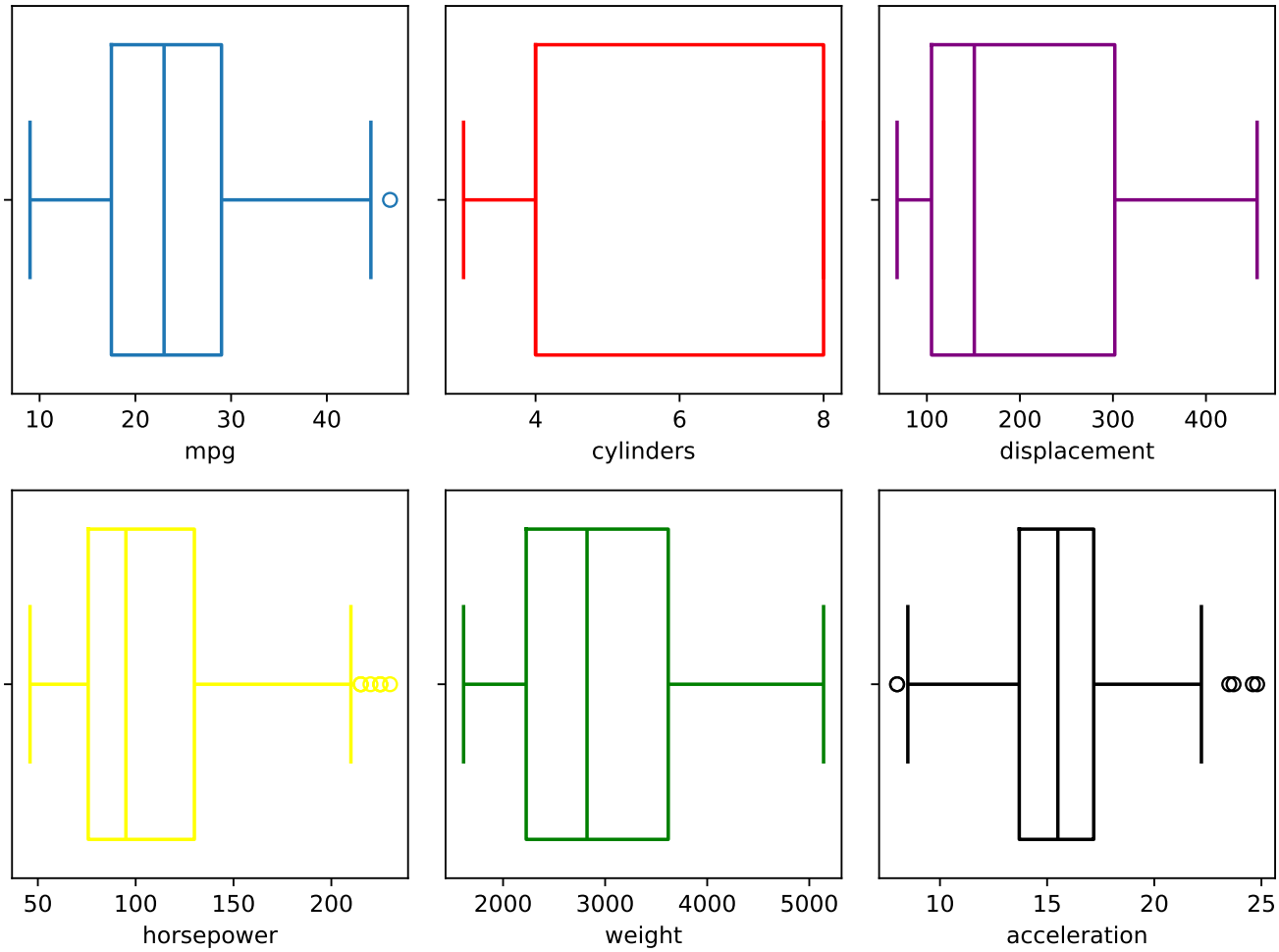
Resumenes estadisticos de todas las variables de la base de datos.

```
consumo.describe(include='all').to_latex();
```

	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year	origin	car_name
count	398.00	406.00	406.00	400.00	406.00	406.00	406.00	406.00	406
unique	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	312
top	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	ford pinto
freq	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	6
mean	23.51	5.48	194.78	105.08	2979.41	15.52	75.92	1.57	NaN
std	7.82	1.71	104.92	38.77	847.00	2.80	3.75	0.80	NaN
min	9.00	3.00	68.00	46.00	1613.00	8.00	70.00	1.00	NaN
25%	17.50	4.00	105.00	75.75	2226.50	13.70	73.00	1.00	NaN
50%	23.00	4.00	151.00	95.00	2822.50	15.50	76.00	1.00	NaN
75%	29.00	8.00	302.00	130.00	3618.25	17.18	79.00	2.00	NaN
max	46.60	8.00	455.00	230.00	5140.00	24.80	82.00	3.00	NaN

Para investigar de manera más visual, analicemos un diagrama de cajas para cada variable numérica de nuestro conjunto de datos. Esto nos permitirá tener una comprensión más clara de la distribución y la presencia de posibles valores atípicos en cada variable.

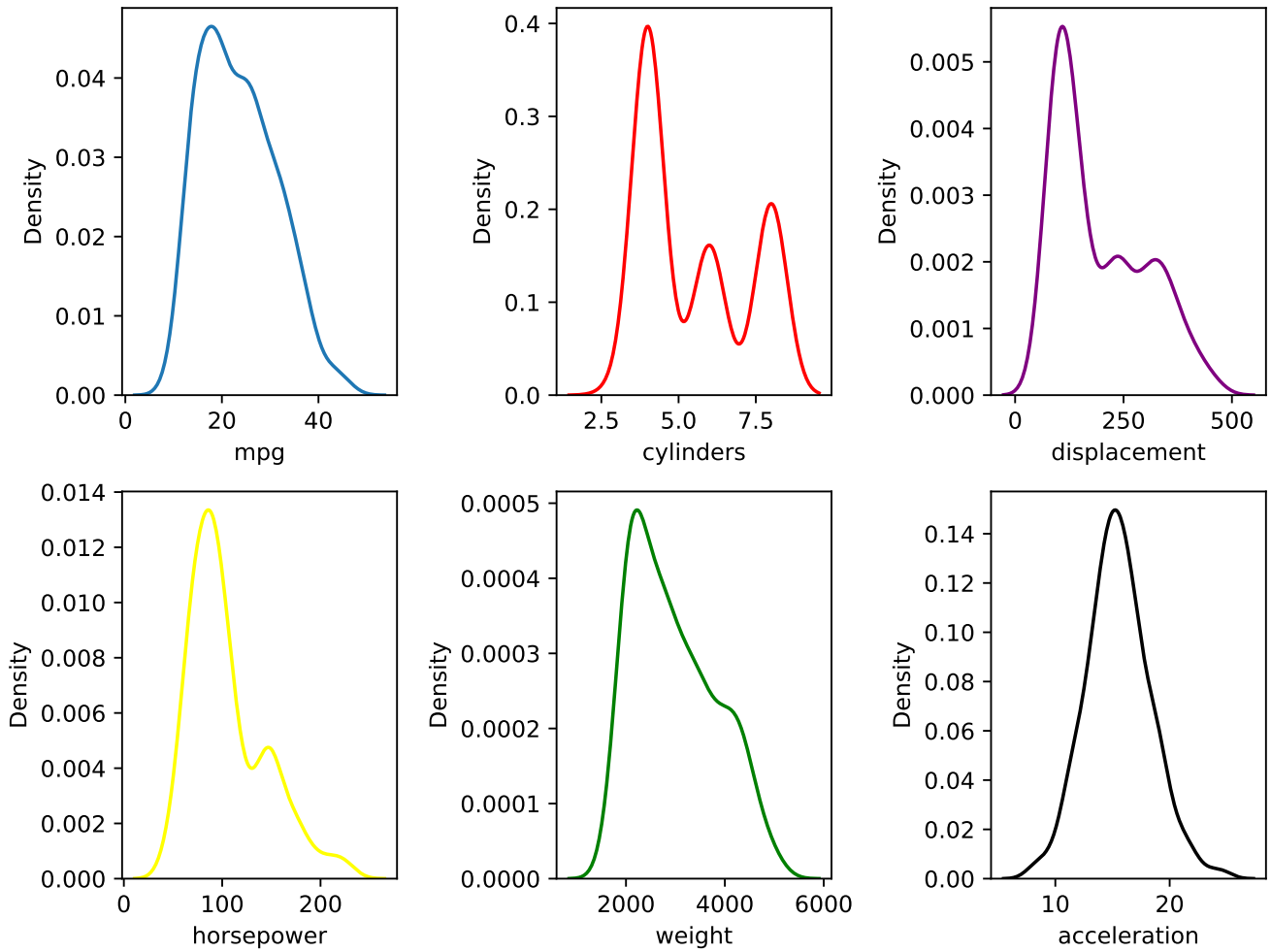
```
fig, axes = plt.subplots(2,3, figsize=(8, 6), sharex=False)
sns.boxplot(ax=axes[0,0],x=consumo['mpg'],fill=False)
sns.boxplot(ax=axes[0,1],x=consumo['cylinders'],fill=False,color="red")
sns.boxplot(ax=axes[0,2],x=consumo['displacement'],fill=False,color="purple")
sns.boxplot(ax=axes[1,0],x=consumo['horsepower'],fill=False,
color="yellow")
sns.boxplot(ax=axes[1,1],x=consumo['weight'],fill=False,
color="green")
sns.boxplot(ax=axes[1,2],x=consumo['acceleration'],fill=False,
color="black")
plt.tight_layout()  # Ajustar el espaciado entre subplots
plt.show()
```



Al analizar las gráficas de cajas anteriores, podemos notar que la gran mayoría de las variables exhiben una distribución sin datos atípicos ni valores iguales a cero. Sin embargo, al examinar con detenimiento las variables **mpg**,**acceleration** y **horsepower**, encontramos que algunos datos se sitúan un poco más allá del rango típico de la variable, lo que sugiere la presencia de valores que podrían considerarse atípicos o inusuales dentro de la muestra.

Estudiemos la distribución de esas variables.

```
fig, axes = plt.subplots(2,3, figsize=(8, 6), sharex=False)
sns.kdeplot(ax=axes[0,0],x=consumo['mpg'],fill=False)
sns.kdeplot(ax=axes[0,1],x=consumo['cylinders'],fill=False,color="red")
sns.kdeplot(ax=axes[0,2],x=consumo['displacement'],fill=False,color="purple")
sns.kdeplot(ax=axes[1,0],x=consumo['horsepower'],fill=False,
color="yellow")
sns.kdeplot(ax=axes[1,1],x=consumo['weight'],fill=False,
color="green")
sns.kdeplot(ax=axes[1,2],x=consumo['acceleration'],fill=False,
color="black")
plt.tight_layout()  # Ajustar el espaciado entre subplots
plt.show()
```



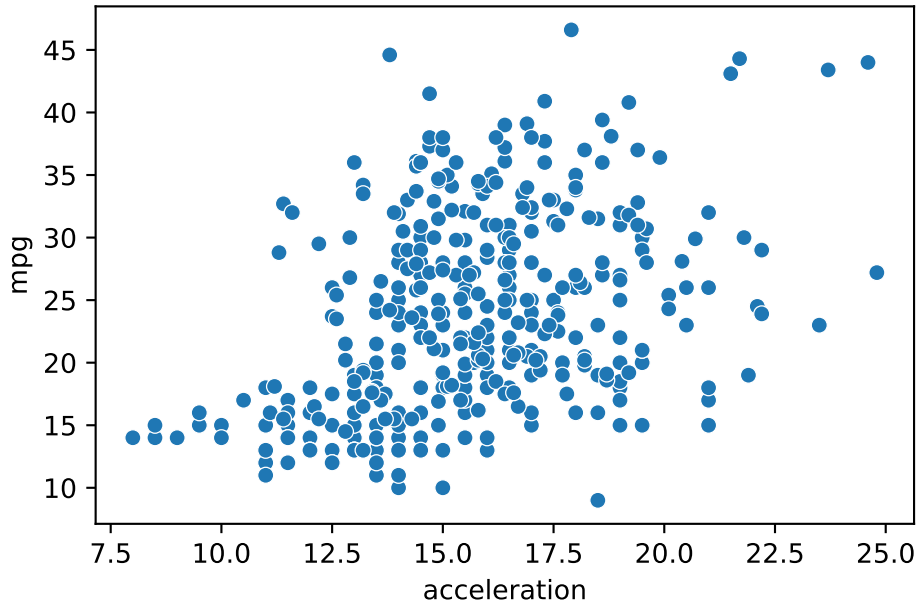
La distribución del consumo de galones por milla es ligeramente asimétrica hacia la derecha. La mayoría de los vehículos se encuentran en el centro de la distribución, consumiendo entre 20 y 30 galones por milla. Sin embargo, existe una pequeña cantidad de vehículos que son más eficientes (consumen menos de 20 galones por milla) o menos eficientes (consumen más de 30 galones por milla) que la mayoría.

Por otro lado, la distribución para la variable de cilindros muestra una forma asimétrica hacia la derecha. La opción más común entre los autos es tener 4 cilindros. Aunque también hay una cantidad considerable de autos con 6 y 8 cilindros, que son opciones bastante comunes.

En cuanto al comportamiento de la variable de aceleración, la mayoría de los autos en el grupo tienen una aceleración similar a la media. Sin embargo, hay una pequeña cantidad de autos con aceleraciones muy bajas o muy altas. Además, la variabilidad en la aceleración es relativamente pequeña.

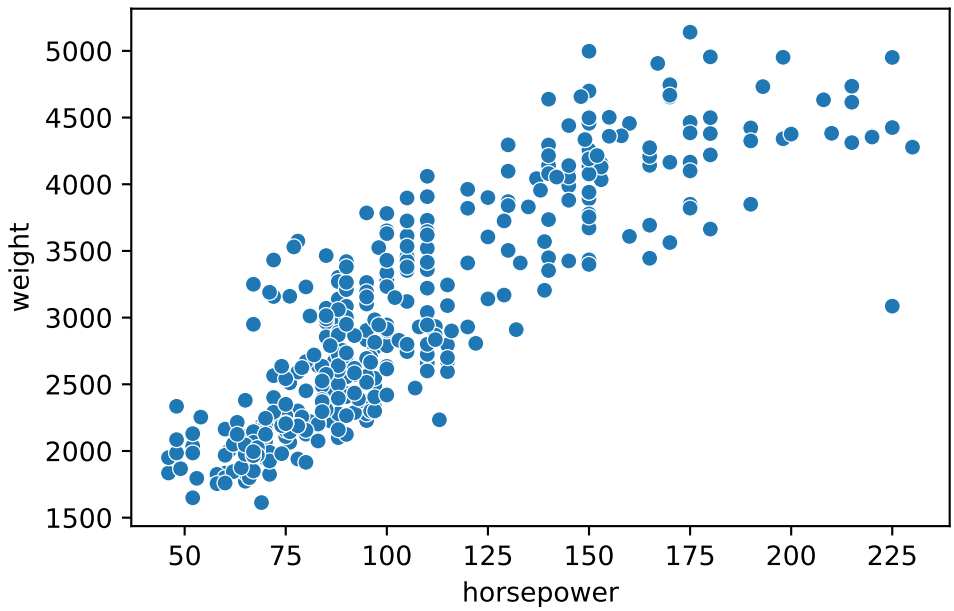
Busquemos si existe relaciones entre variables del conjunto de datos, se podria pensar que a mayor aceleración del auto mayor será su consumo, esto lo veremos en el suiguiente grafico de disperción.

```
sns.scatterplot(data=consumo, y='mpg', x='acceleration')
```



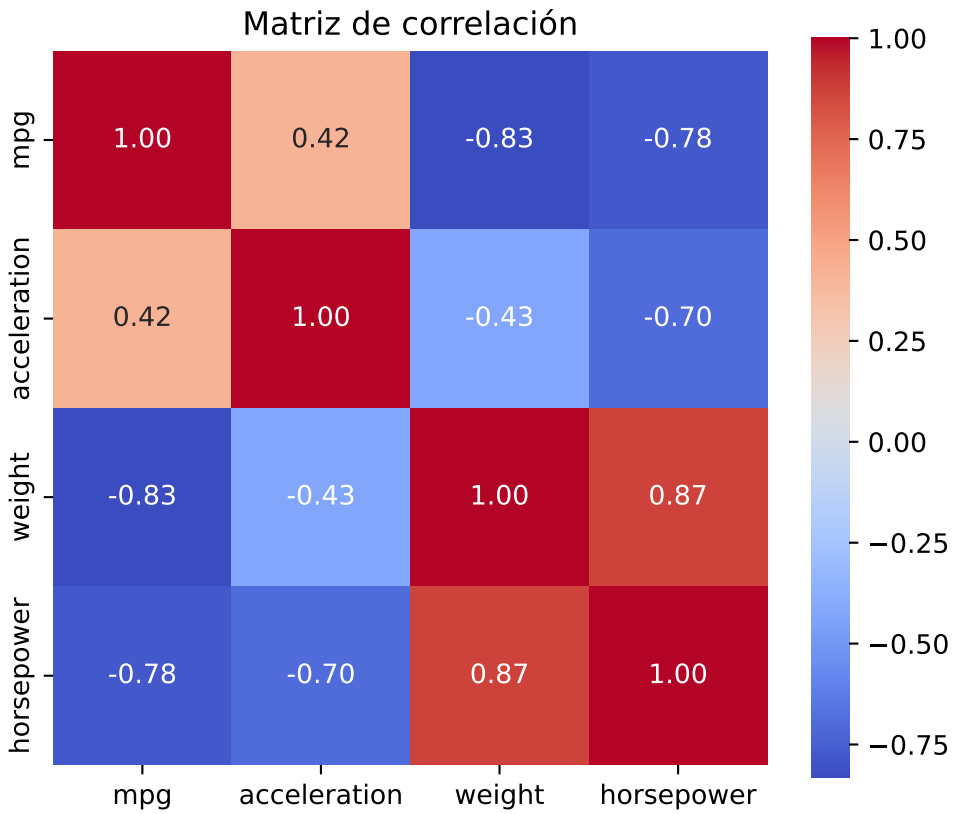
Otra posible relación es el peso del auto con los caballo de fuerza que contenga.

```
sns.scatterplot(data=consumo, y='weight', x='horsepower')
```



Así de esta manera podemos ir determinando posibles relaciones entre variables, acontinuacion se muestra una matriz de correlación entre las variables tratadas anteriormete.

```
plt.figure(figsize=(6, 5))
sns.heatmap(consumo.loc[:, ['mpg','acceleration','weight','horsepower']].corr(), annot=True,
cmap='coolwarm',fmt=".2f", square=True)
plt.title('Matriz de correlación')
plt.show()
```



c) Determinación de datos faltante.

Datos faltantes en cada variable.

```
NaN = consumo.isna().sum()
NaN = pd.DataFrame(NaN)
NaN['porcentaje'] = pd.DataFrame(consumo.isna().sum()/406*(100))
NaN.columns = ['nan','porcentaje']
NaN
```

	nan	porcentaje
mpg	8	1.970443
cylinders	0	0.000000
displacement	0	0.000000
horsepower	6	1.477833
weight	0	0.000000
acceleration	0	0.000000
model_year	0	0.000000
origin	0	0.000000
car_name	0	0.000000

El porcentaje que representan estos datos faltantes a toda la base de datos se puede determinar.

```
por = (consumo.isna().sum().sum()/np.product(consumo.shape))*100
print(f"Así el porcentaje total de datos faltantes  es de {por:.2f}% del total de los datos.")
```

```
Así el porcentaje total de datos faltantes  es de 0.38% del total de los datos.
```

d) Mecanismo inherente de datos faltantes.

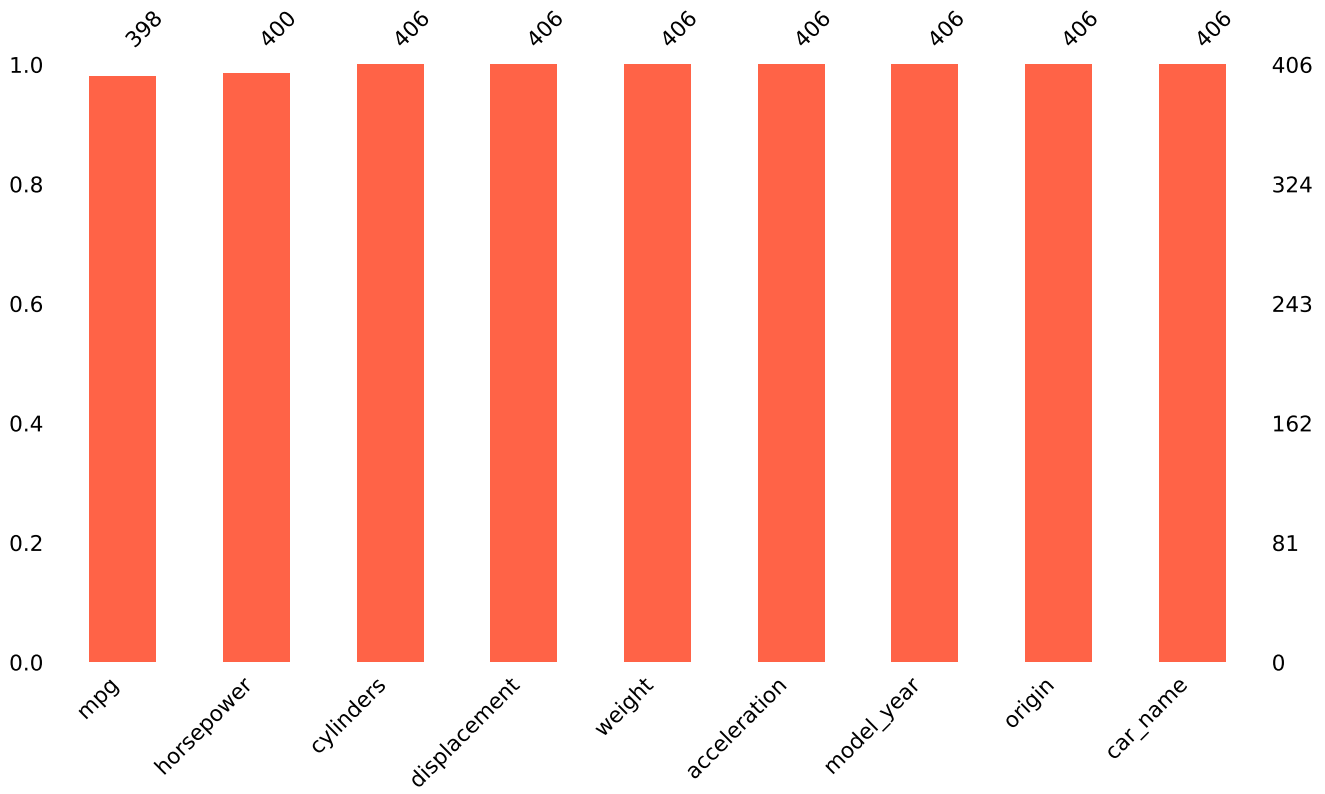
```
consumo.columns
consumo.loc[:, ['mpg','cylinders','displacement','acceleration','weight','horsepower']].corr()
```

	mpg	cylinders	displacement	acceleration	weight	horsepower
mpg	1.000000	-0.775396	-0.804203	0.420289	-0.831741	-0.778427
cylinders	-0.775396	1.000000	0.951787	-0.522452	0.895220	0.844158
displacement	-0.804203	0.951787	1.000000	-0.557984	0.932475	0.898326
acceleration	0.420289	-0.522452	-0.557984	1.000000	-0.430086	-0.697124
weight	-0.831741	0.895220	0.932475	-0.430086	1.000000	0.866586
horsepower	-0.778427	0.844158	0.898326	-0.697124	0.866586	1.000000

Como lo determinamos anteriormente los datos presentan altos niveles de correlación entre las variables, lo que nos puede llevar a pensar que el mecanismo inherente de datos atipicos puede ser Missing at Random (MAR) ya que las variables que presentan datos faltantes pueden ser explicadas por otras variables dela misma base de datos.

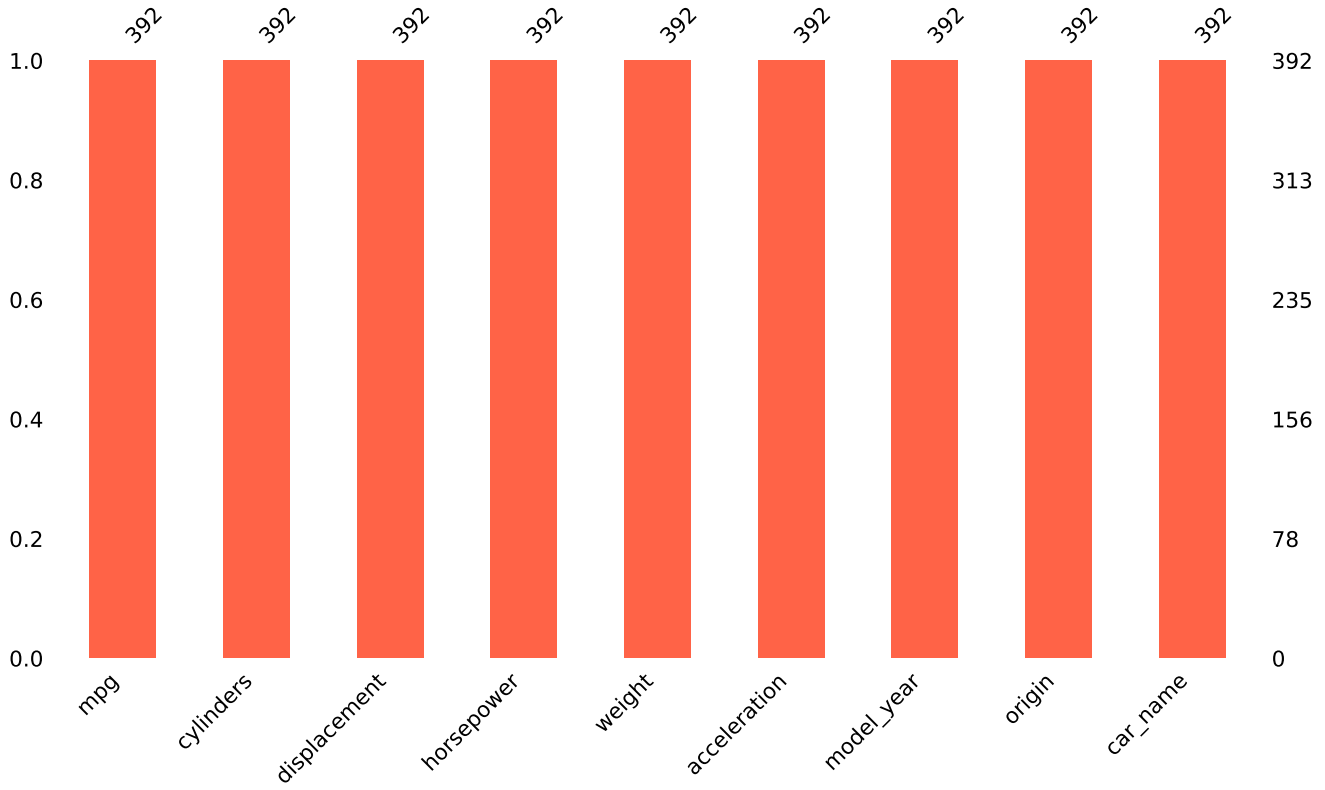
e) Tecnicas de tratamiento de datos faltantes.

```
msno.bar(consumo,figsize=(12, 6), sort="ascending",fontsize=12, color='tomato')
```



Eliminar: Por el método de eliminación optaremos por eliminar todas las filas que contengan datos.

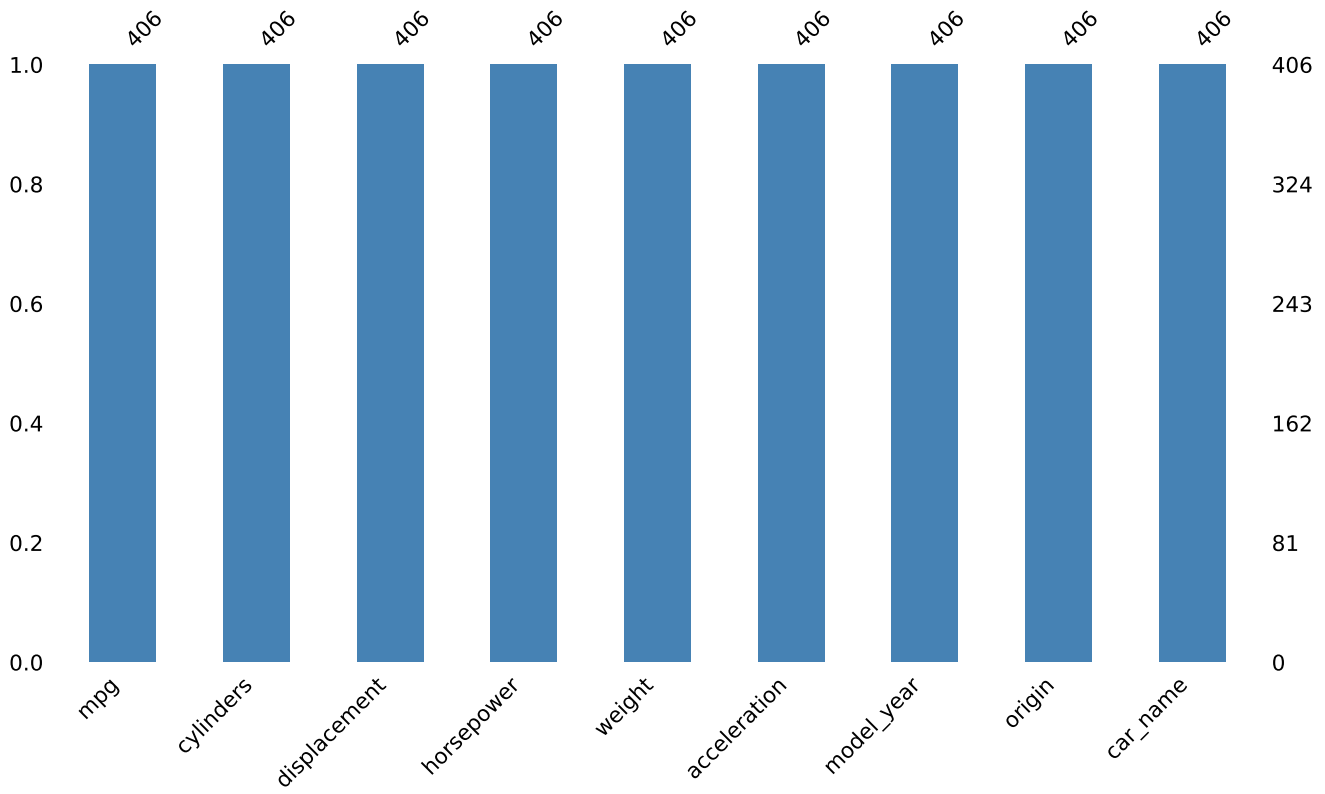
```
consumo_delete = consumo.copy()
consumo_delete.dropna(inplace=True)
msno.bar(consumo_delete,figsize=(12, 6), sort="ascending",fontsize=12, color='tomato')
```



Así obtenemos una base de datos con 392 registros. De esta manera, verificamos que todas las variables tienen el mismo número de registros.

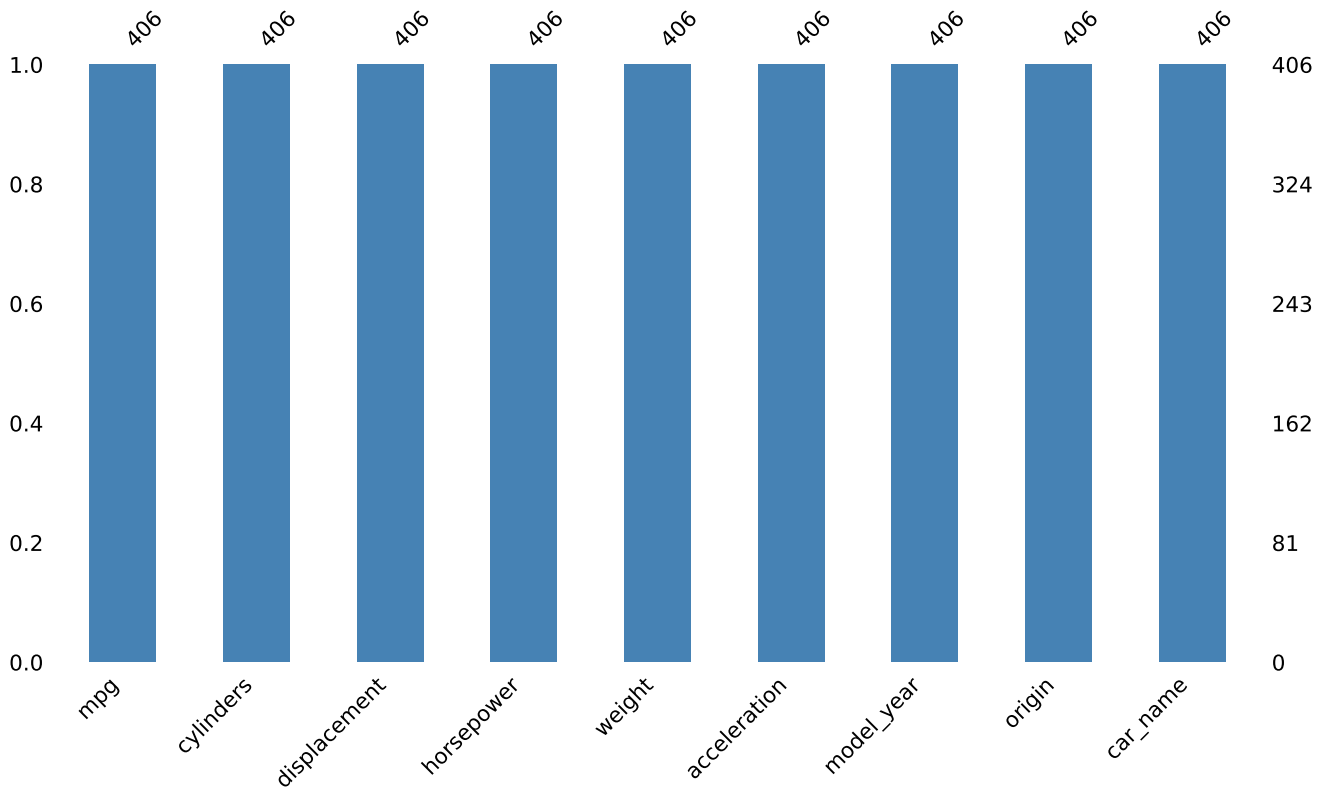
Imputar: De forma general vamos a imputar con la media.

```
consumo_imp_media = consumo.copy()
mean_imputer = SimpleImputer(strategy='mean')
consumo_imp_media.iloc[:,:8] = mean_imputer.fit_transform(consumo_imp_media.iloc[:,:8])
msno.bar(consumo_imp_media,figsize=(12, 6), fontsize=12, color='steelblue')
```



Como ultimo método utilizaremos KNNImputer de Sklearn.

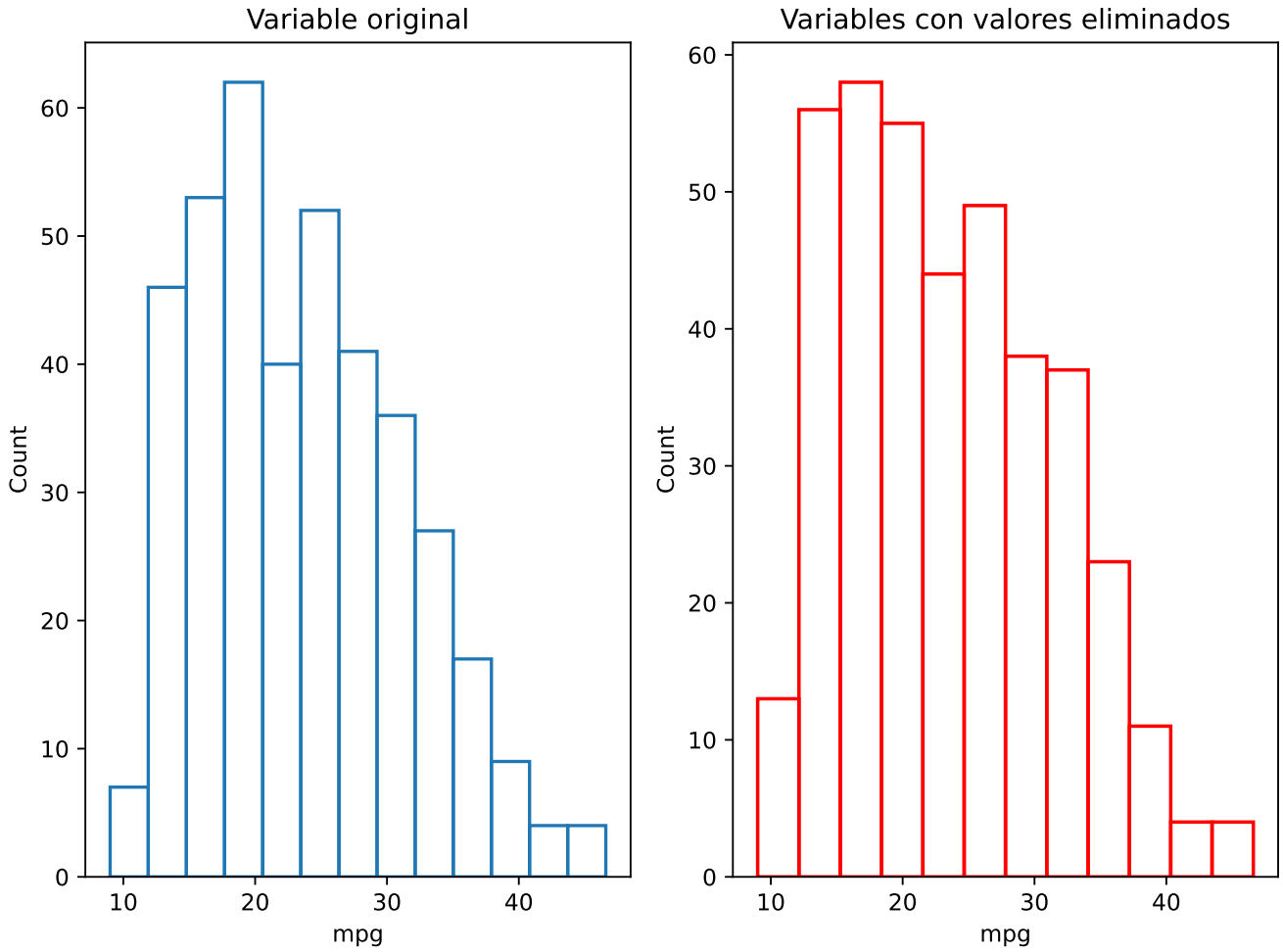
```
consumo_KNN = consumo.copy(deep=True)

knn_imputer = KNNImputer(n_neighbors=2, weights="uniform")
consumo_KNN[['mpg','horsepower']] = knn_imputer.fit_transform(consumo_KNN[['mpg','horsepower']])

msno.bar(consumo_KNN,figsize=(12, 6), fontsize=12, color='steelblue')
```



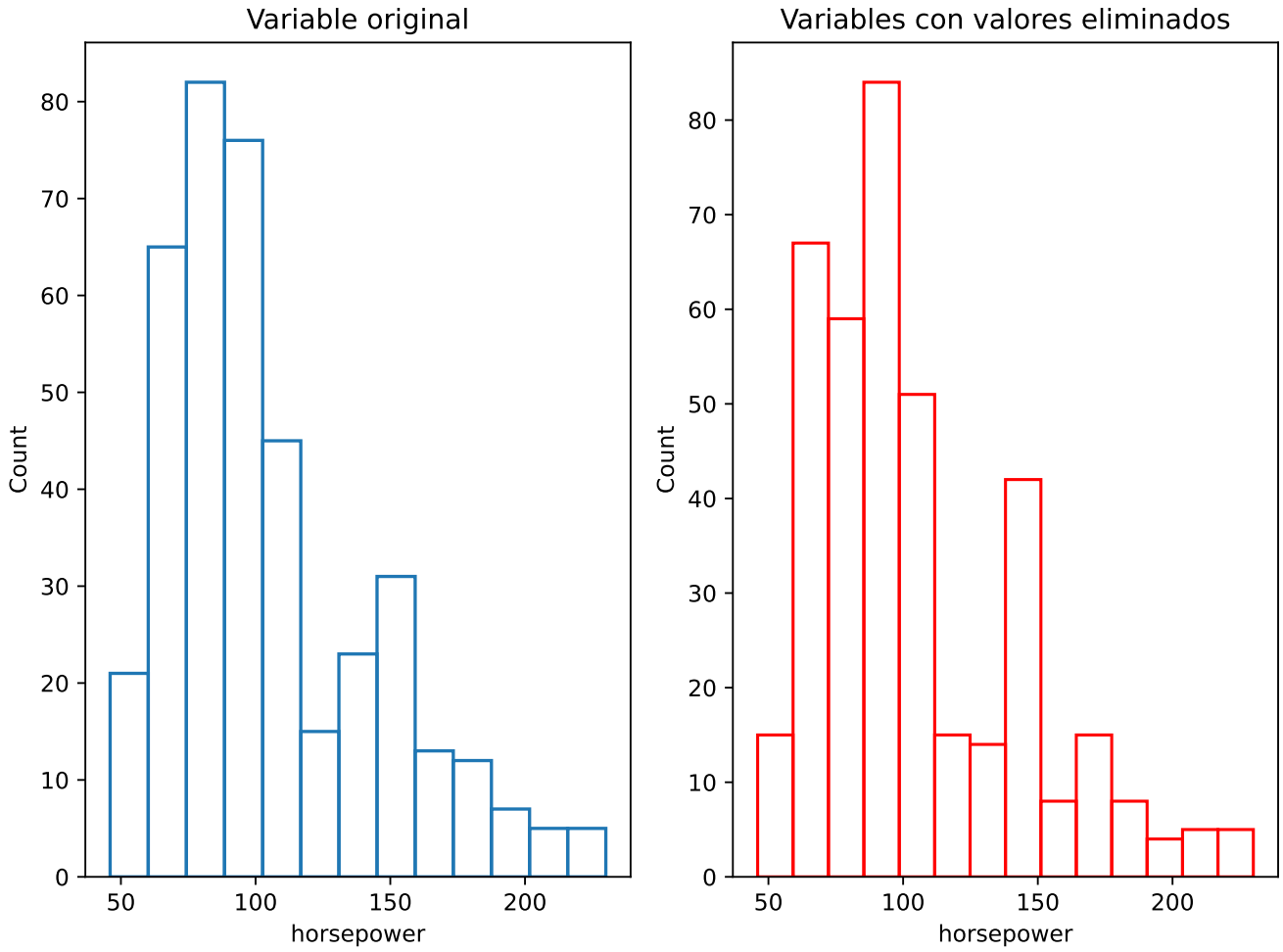
f) Comparación de la variables originales con las variables imputadas .

Por el método de elimación de resgistros

```
# Para la Variable mpg por elimacion
fig, axes = plt.subplots(1,2, figsize=(8, 6), sharex=False)
sns.histplot(ax=axes[0],x=consumo['mpg'],fill=False)
axes[0].set_title('Variable original')
sns.histplot(ax=axes[1],x=consumo_delete['mpg'],fill=False,color="red")
axes[1].set_title('Variables con valores eliminados')
plt.tight_layout()  # Ajustar el espaciado entre subplots
plt.show()
```

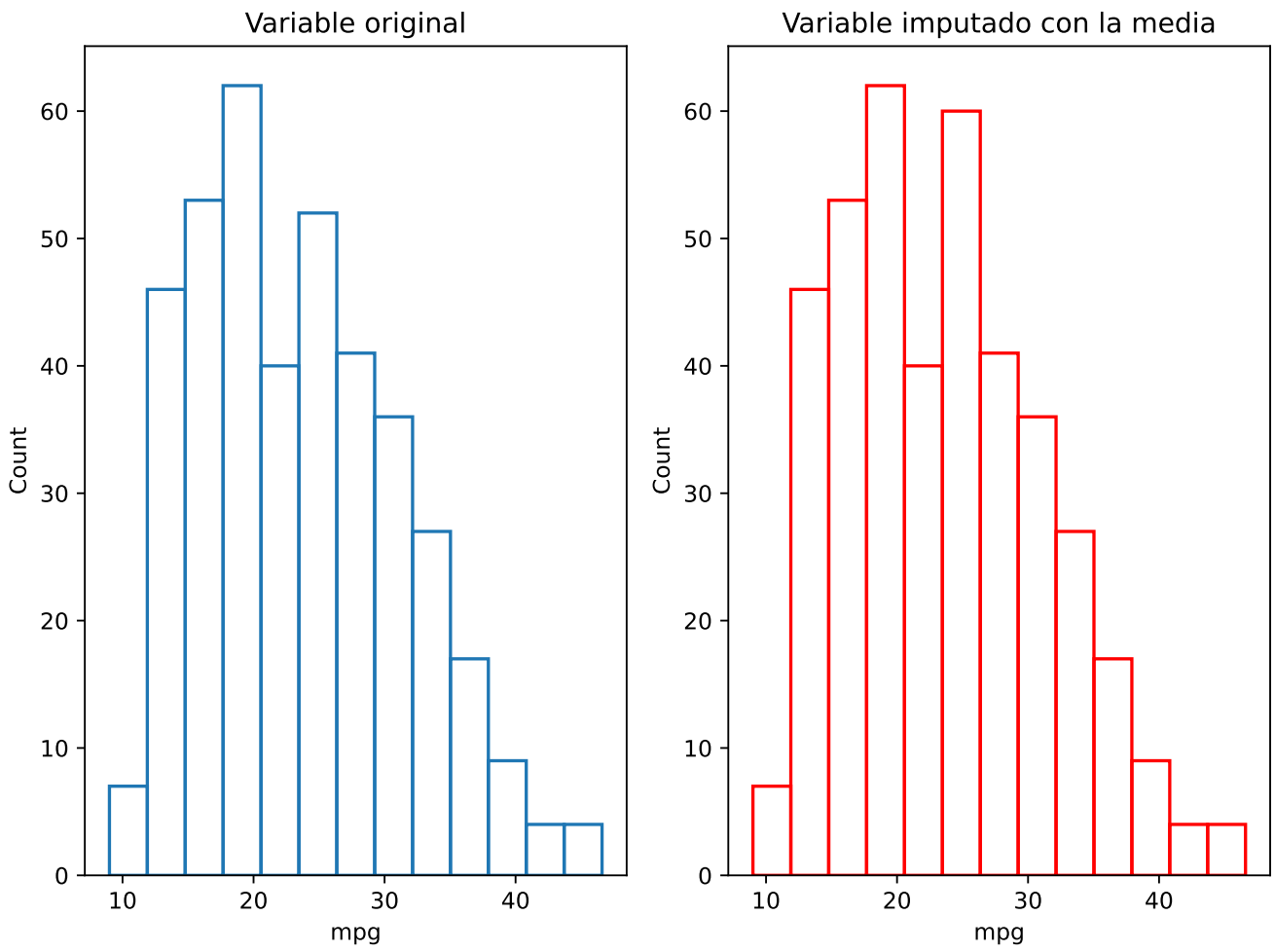



```
# Para la Variable horsepower por elimacion
fig, axes = plt.subplots(1,2, figsize=(8, 6), sharex=False)
sns.histplot(ax=axes[0],x=consumo['horsepower'],fill=False)
axes[0].set_title('Variable original')
sns.histplot(ax=axes[1],x=consumo_delete['horsepower'],fill=False,color="red")
axes[1].set_title('Variables con valores eliminados')
plt.tight_layout()  # Ajustar el espaciado entre subplots
plt.show()
```

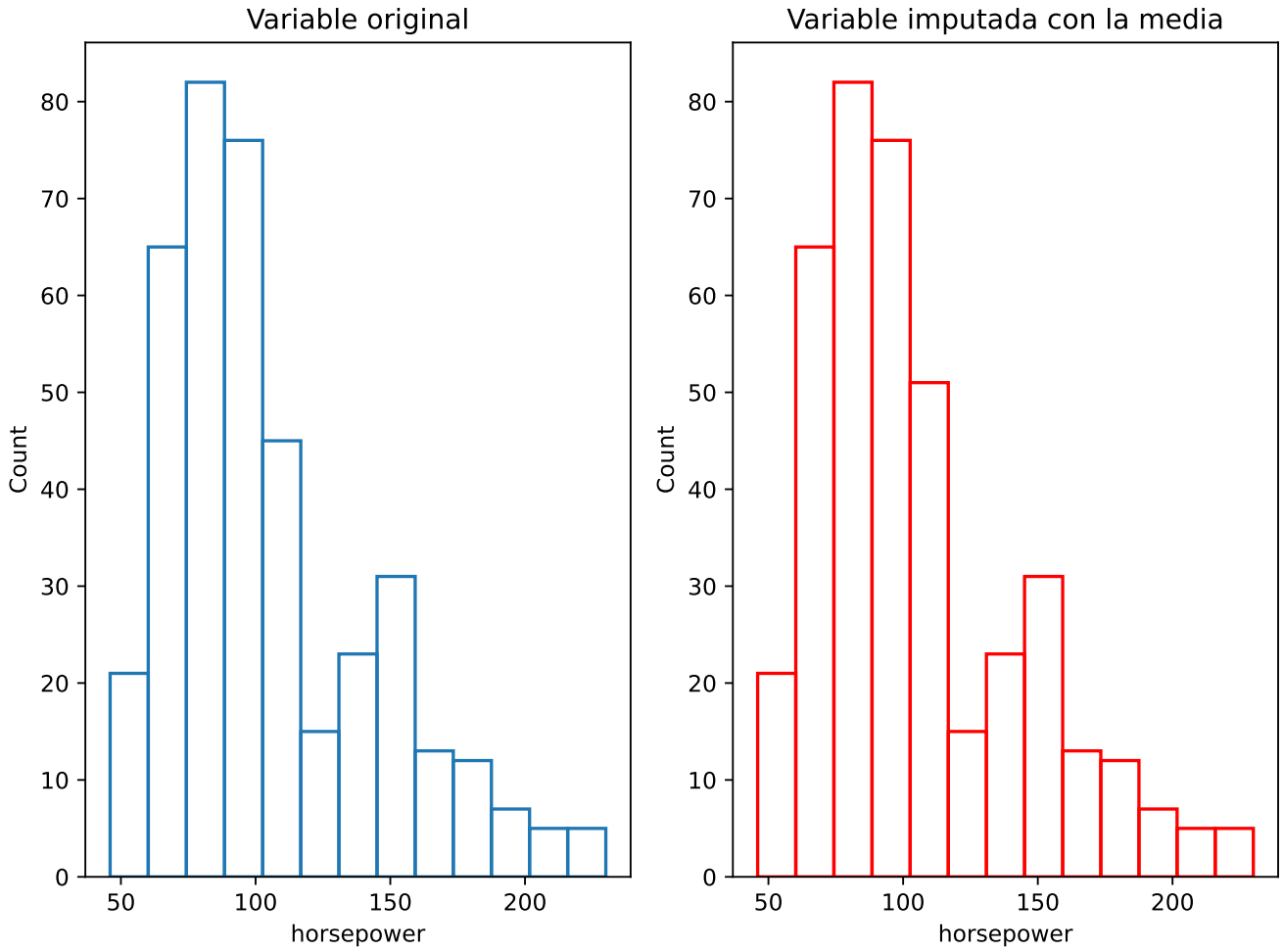


Por el método de Imputación por la media

```
# Para la Variable mpg imputacion por la media
fig, axes = plt.subplots(1,2, figsize=(8, 6), sharex=False)
sns.histplot(ax=axes[0],x=consumo['mpg'],fill=False)
axes[0].set_title('Variable original')
sns.histplot(ax=axes[1],x=consumo_imp_media['mpg'],fill=False,color="red")
axes[1].set_title('Variable imputado con la media')
plt.tight_layout()  # Ajustar el espaciado entre subplots
plt.show()
```

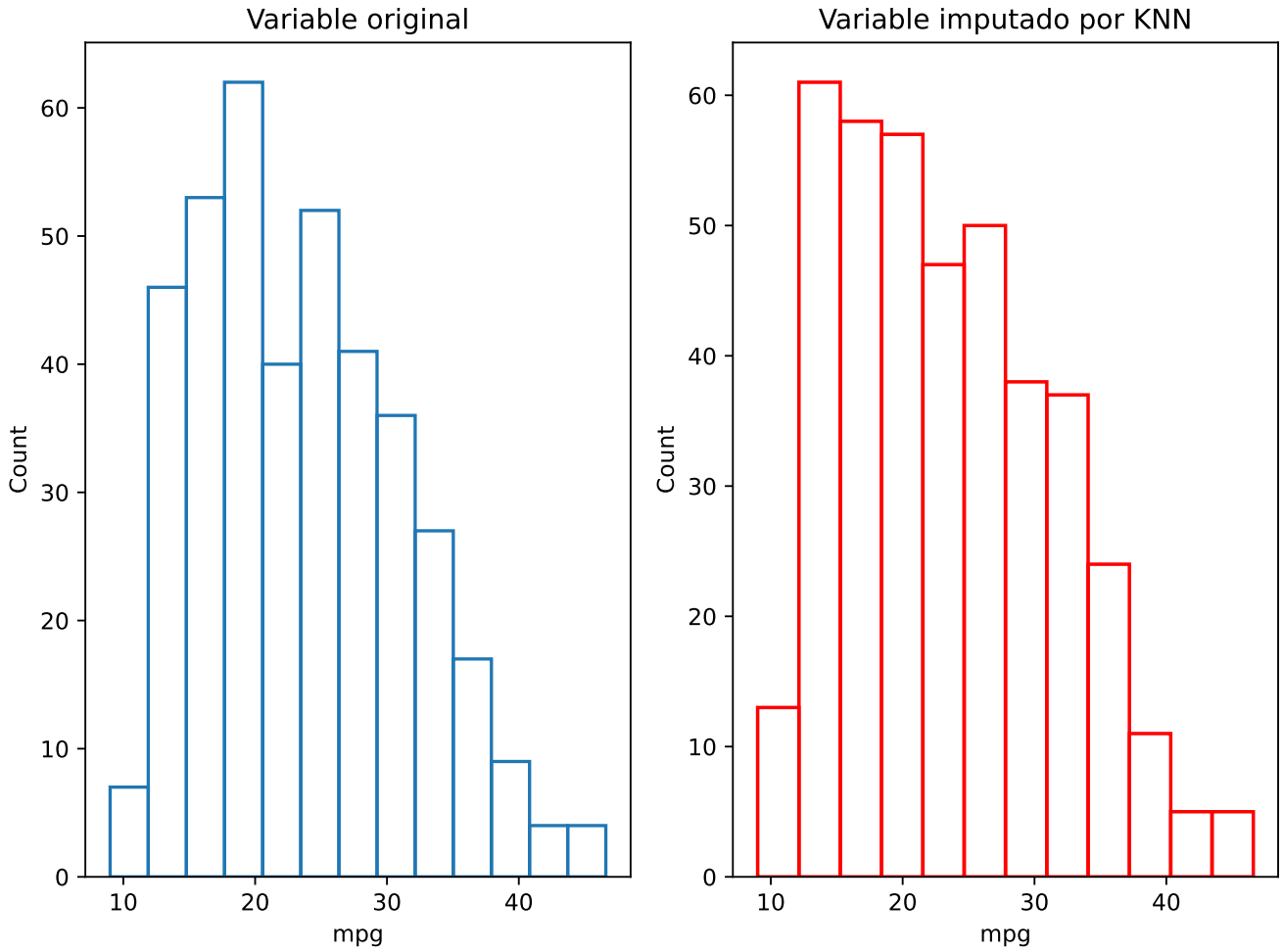



```
# Para la Variable horsepower imputacion con la media
fig, axes = plt.subplots(1,2, figsize=(8, 6), sharex=False)
sns.histplot(ax=axes[0],x=consumo['horsepower'],fill=False)
axes[0].set_title('Variable original')
sns.histplot(ax=axes[1],x=consumo_imp_media['horsepower'],fill=False,color="red")
axes[1].set_title('Variable imputada con la media')
plt.tight_layout()  # Ajustar el espaciado entre subplots
plt.show()
```

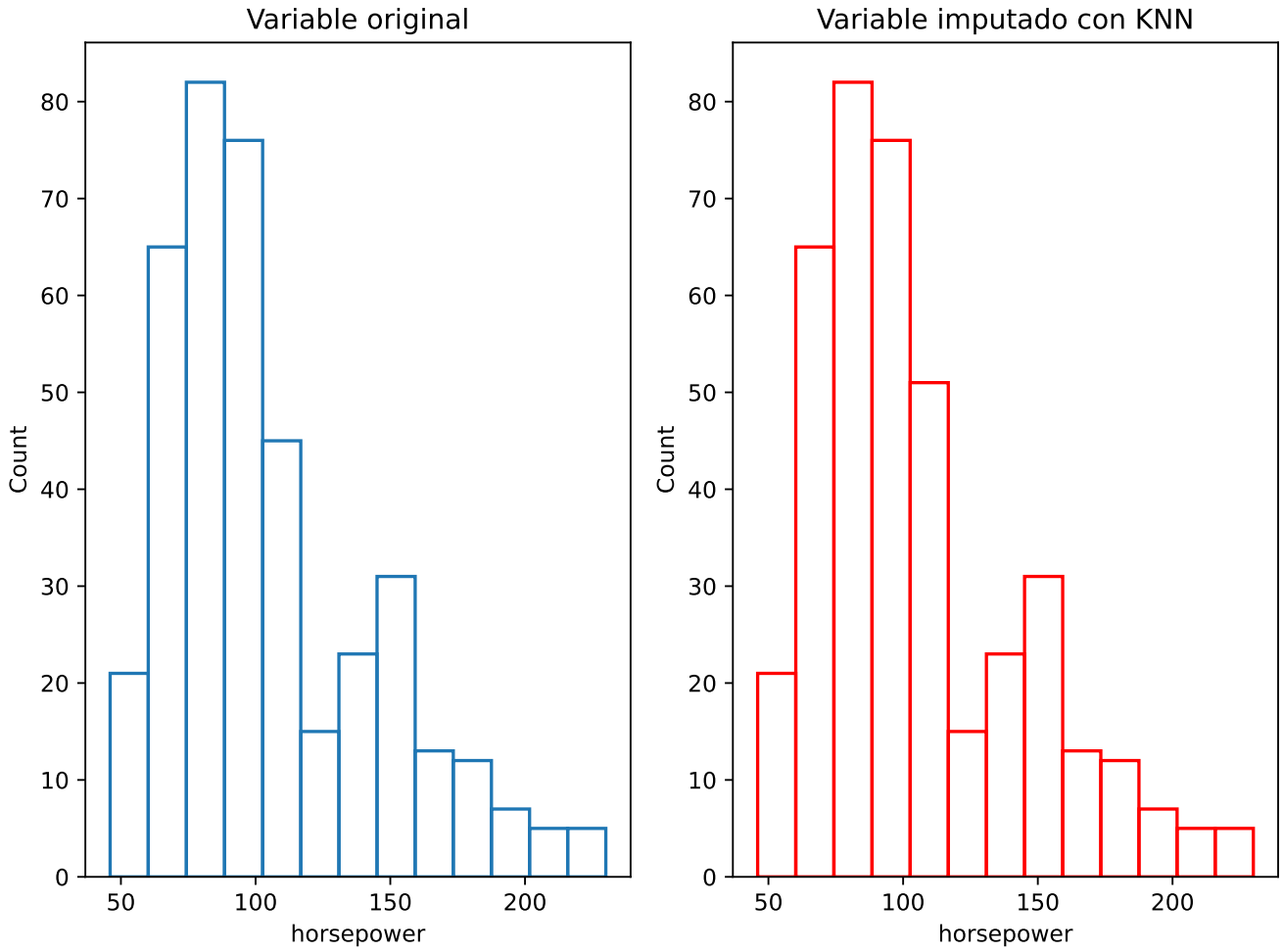


Por el método de KNNImputer de Sklearn

```
# Para la Variable mpg imputacion por KNNImputer de Sklearn
fig, axes = plt.subplots(1,2, figsize=(8, 6), sharex=False)
sns.histplot(ax=axes[0],x=consumo['mpg'],fill=False)
axes[0].set_title('Variable original')
sns.histplot(ax=axes[1],x=consumo_KNN['mpg'],fill=False,color="red")
axes[1].set_title('Variable imputado por KNN')
plt.tight_layout()  # Ajustar el espaciado entre subplots
plt.show()
```




```
# Para la Variable horsepower imputacion por KNN
fig, axes = plt.subplots(1,2, figsize=(8, 6), sharex=False)
sns.histplot(ax=axes[0],x=consumo['horsepower'],fill=False)
axes[0].set_title('Variable original')
sns.histplot(ax=axes[1],x=consumo_imp_media['horsepower'],fill=False,color="red")
axes[1].set_title('Variable imputado con KNN')
plt.tight_layout()  # Ajustar el espaciado entre subplots
plt.show()
```



El método de imputación utilizando la media de la variable se destaca como la opción preferida en comparación con la eliminación de registros y el método KNN (vecinos más cercanos) debido a su capacidad para preservar la integridad del conjunto de datos al conservar todas las muestras originales. Además, este enfoque es simple, eficiente y fácilmente generalizable a una variedad de situaciones, lo que lo hace ampliamente aplicable en el preprocesamiento de datos. Al reemplazar los valores faltantes con un valor central representativo de la variable, la imputación con la media también mantiene la distribución general de los datos, facilitando la interpretación y el análisis, mientras que proporciona robustez frente a valores atípicos que pueden afectar métodos más complejos como el KNN.