

Machine Learning Algorithms applied to CO2 Emissions in Canada

Emanuel Novelo Hernández

June 18, 2024

1 Descripción de los Datos

Este conjunto de datos captura los detalles de cómo las emisiones de CO2 de un vehículo pueden variar con las diferentes características. El conjunto de datos ha sido tomado del sitio web oficial de datos abiertos del Gobierno de Canadá. Esta es una versión compilada que contiene datos de un período de 7 años.

Hay un total de 7385 filas y 12 columnas. Se han utilizado algunas abreviaturas para describir las características. Las estoy enumerando aquí. Las mismas se pueden encontrar en la hoja de Descripción de Datos.

Los datos han sido tomados y compilados del siguiente enlace oficial del Gobierno de Canadá: <https://open.canada.ca/data/en/dataset/98f1a129-f628-4ce4-b24d-6f16bf24dd64wb-auto-6>

Se hace la lectura de los datos

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

```
[19]: # lectura de datos
df = pd.read_csv('C:/Users/emanuel.novelo/Desktop/MCD - 2024-2026/ML - 2do Tetra/
↳MCD----ML----2024/data/CO2 Emissions_Canada.csv')
df.head()
```

```
[19]:      Make      Model Vehicle Class  Engine Size(L)  Cylinders  Transmission  \
0  ACURA      ILX      COMPACT          2.0           4           AS5
1  ACURA      ILX      COMPACT          2.4           4           M6
2  ACURA  ILX HYBRID      COMPACT          1.5           4           AV7
3  ACURA    MDX 4WD    SUV - SMALL          3.5           6           AS6
4  ACURA    RDX AWD    SUV - SMALL          3.5           6           AS6
```

```
      Fuel Type  Fuel Consumption City (L/100 km)  \
0           Z           9.9
1           Z          11.2
2           Z           6.0
3           Z          12.7
4           Z          12.1
```

| | Fuel Consumption Hwy (L/100 km) | Fuel Consumption Comb (L/100 km) | \ |
|---|---------------------------------|----------------------------------|------|
| 0 | 6.7 | | 8.5 |
| 1 | 7.7 | | 9.6 |
| 2 | 5.8 | | 5.9 |
| 3 | 9.1 | | 11.1 |
| 4 | 8.7 | | 10.6 |

| | Fuel Consumption Comb (mpg) | CO2 Emissions(g/km) |
|---|-----------------------------|---------------------|
| 0 | 33 | 196 |
| 1 | 29 | 221 |
| 2 | 48 | 136 |
| 3 | 25 | 255 |
| 4 | 27 | 244 |

Descripción básica de la data

```
[7]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7385 entries, 0 to 7384
Data columns (total 12 columns):
#   Column                                Non-Null Count  Dtype
---  -
0   Make                                  7385 non-null   object
1   Model                                7385 non-null   object
2   Vehicle Class                        7385 non-null   object
3   Engine Size(L)                       7385 non-null   float64
4   Cylinders                            7385 non-null   int64
5   Transmission                         7385 non-null   object
6   Fuel Type                            7385 non-null   object
7   Fuel Consumption City (L/100 km)     7385 non-null   float64
8   Fuel Consumption Hwy (L/100 km)     7385 non-null   float64
9   Fuel Consumption Comb (L/100 km)    7385 non-null   float64
10  Fuel Consumption Comb (mpg)          7385 non-null   int64
11  CO2 Emissions(g/km)                 7385 non-null   int64
dtypes: float64(4), int64(3), object(5)
memory usage: 692.5+ KB
```

2 Unsupervised Learning - DBSCAN & TSNE for Clustering CO2 Emissions

Abstract de la sección Se tomó como inspiración el artículo *Exploring Spatiotemporal Pattern and Agglomeration of Road CO2 Emissions in Guangdong, China* para realizar un modelo de aprendizaje no supervisado (DBSCAN) sobre los datos de emisiones de CO2 en vehículos en Canadá. Se realiza primeramente una reducción de dimensionalidad para tener una visualización 2D de los datos, usando el algoritmo de T-SNE, esto con el fin de ver la densidad de colores (clusters) asignados por la clusterización obtenida del DBSCAN. Para fines de este ejercicio, el DBSCAN se efectúa sobre

las variables numéricas de los datos.

2.1 Metodología

2.1.1 Nearest Neighbors

El algoritmo de vecinos más cercanos (Nearest Neighbors) se utiliza para encontrar puntos en un conjunto de datos que están más cerca unos de otros. La distancia euclidiana es una métrica comúnmente utilizada para medir la proximidad entre dos puntos x_i y x_j :

$$d(x_i, x_j) = \sqrt{\sum_{k=1}^n (x_{ik} - x_{jk})^2} \quad (1)$$

donde x_{ik} y x_{jk} son las coordenadas de los puntos x_i y x_j en el espacio k -dimensional.

2.1.2 t-Distributed Stochastic Neighbor Embedding (t-SNE)

t-SNE es una técnica de reducción de dimensionalidad que se utiliza para visualizar datos de alta dimensión. Primero, t-SNE calcula las probabilidades de similitud entre los puntos en el espacio de alta dimensión utilizando una distribución gaussiana:

$$p_{j|i} = \frac{\exp(-\|x_i - x_j\|^2 / 2\sigma_i^2)}{\sum_{k \neq i} \exp(-\|x_i - x_k\|^2 / 2\sigma_i^2)} \quad (2)$$

Estas probabilidades se simetrizan como:

$$p_{ij} = \frac{p_{j|i} + p_{i|j}}{2n} \quad (3)$$

En el espacio de baja dimensión, t-SNE utiliza una distribución t de Student para calcular las probabilidades q_{ij} :

$$q_{ij} = \frac{(1 + \|y_i - y_j\|^2)^{-1}}{\sum_{k \neq i} (1 + \|y_i - y_k\|^2)^{-1}} \quad (4)$$

El objetivo es minimizar la divergencia de Kullback-Leibler entre las distribuciones P y Q :

$$KL(P||Q) = \sum_{i \neq j} p_{ij} \log \frac{p_{ij}}{q_{ij}} \quad (5)$$

2.1.3 Density-Based Spatial Clustering of Applications with Noise (DBSCAN)

DBSCAN es un algoritmo de clustering basado en densidad que agrupa puntos que están juntos y marca los puntos que están en áreas de baja densidad como ruido. DBSCAN utiliza dos parámetros principales: ϵ (el radio de un punto de consulta) y $minPts$ (el número mínimo de puntos requeridos para formar un cluster).

Un punto p es un punto central si al menos $minPts$ puntos están dentro de una distancia ϵ de p :

$$N_{\epsilon}(p) = \{q \in D \mid d(p, q) \leq \epsilon\} \quad (6)$$

donde $N_{\epsilon}(p)$ es el conjunto de puntos dentro de la distancia ϵ de p .

El algoritmo clasifica los puntos en tres categorías: - Puntos centrales: tienen al menos *minPts* puntos dentro de su radio ϵ . - Puntos borde: tienen menos de *minPts* puntos dentro de su radio ϵ pero están dentro del radio ϵ de un punto central. - Puntos de ruido: no son puntos centrales ni puntos borde.

2.2 Resultados & Conclusiones

Selección de features numéricas para T-SNE y DBSCAN

```
[20]: # Numeric Features for clustering
# Seleccionar las columnas numéricas para clustering
features = df[['Engine Size(L)', 'Cylinders', 'Fuel Consumption City (L/100 km)',
               'Fuel Consumption Hwy (L/100 km)', 'Fuel Consumption Comb (L/
               ↳100 km)',
               'Fuel Consumption Comb (mpg)', 'CO2 Emissions(g/km)']]
```

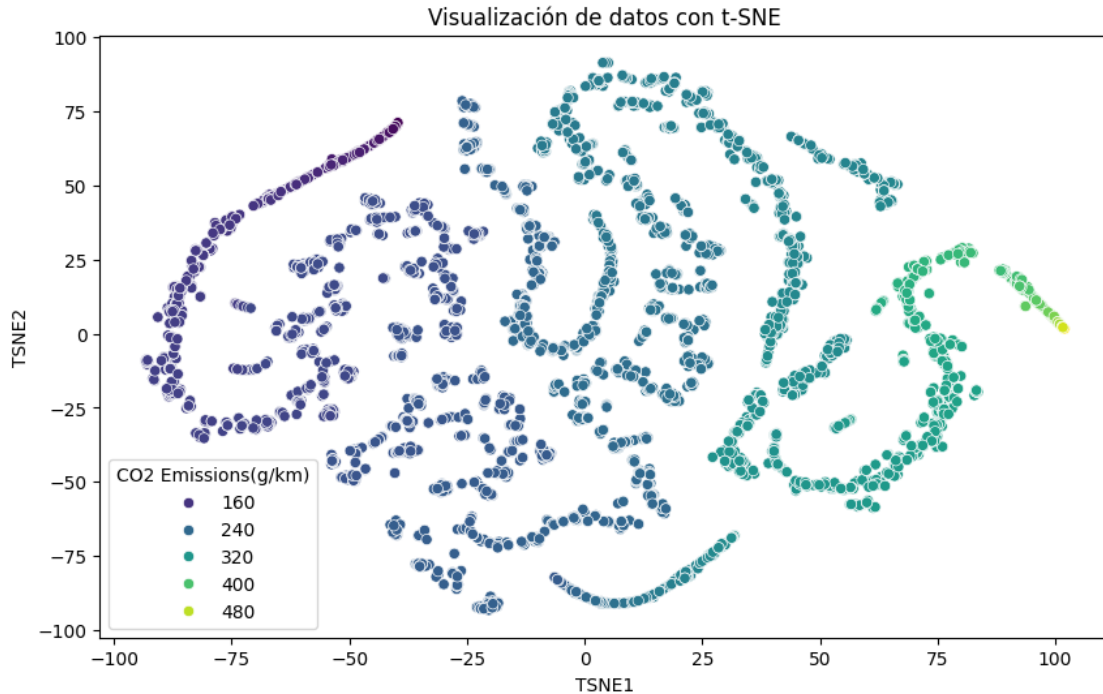
Dimensionality Reduction (T-SNE). Se realiza una reducción de dimensionalidad a 2 componentes para visualizar las features en un espacio 2D. El algoritmo DBSCAN funciona particularmente bien para identificar densidad en los datos, por lo que la visualización 2D resulta apropiada. El T-SNE es una técnica de Aprendizaje No Supervisado de reducción de dimensiones ampliamente utilizada para la exploración de datos y la visualización de datos de altas dimesiones.

```
[21]: from sklearn.manifold import TSNE
import matplotlib.pyplot as plt
import seaborn as sns

# Reducir las dimensiones a 2D
tsne = TSNE(n_components=2, random_state=42)
tsne_results = tsne.fit_transform(features)

# Crear un DataFrame con los resultados de t-SNE
tsne_df = pd.DataFrame(tsne_results, columns=['TSNE1', 'TSNE2'])
tsne_df['CO2 Emissions(g/km)'] = df['CO2 Emissions(g/km)']

# Visualizar los datos
plt.figure(figsize=(10, 6))
sns.scatterplot(x='TSNE1', y='TSNE2', hue='CO2 Emissions(g/km)',
               ↳palette='viridis', data=tsne_df)
plt.title('Visualización de datos con t-SNE')
plt.show()
```



Se emplea el DBSCAN para crear clusters sobre las features, posteriormente se asigna el cluster al df original y al df con dimensionalidades reducidas. Se gráfica la distribución de clusters (colores) respecto al scatterplot de los componentes tsne1 y tsne2. La primera gráfica que se ve es una medida conocida como *Optimal Eps Value* que sirve para determinar el hiperparámetro **eps** del algoritmo DBSCAN (es el parámetro más importante)

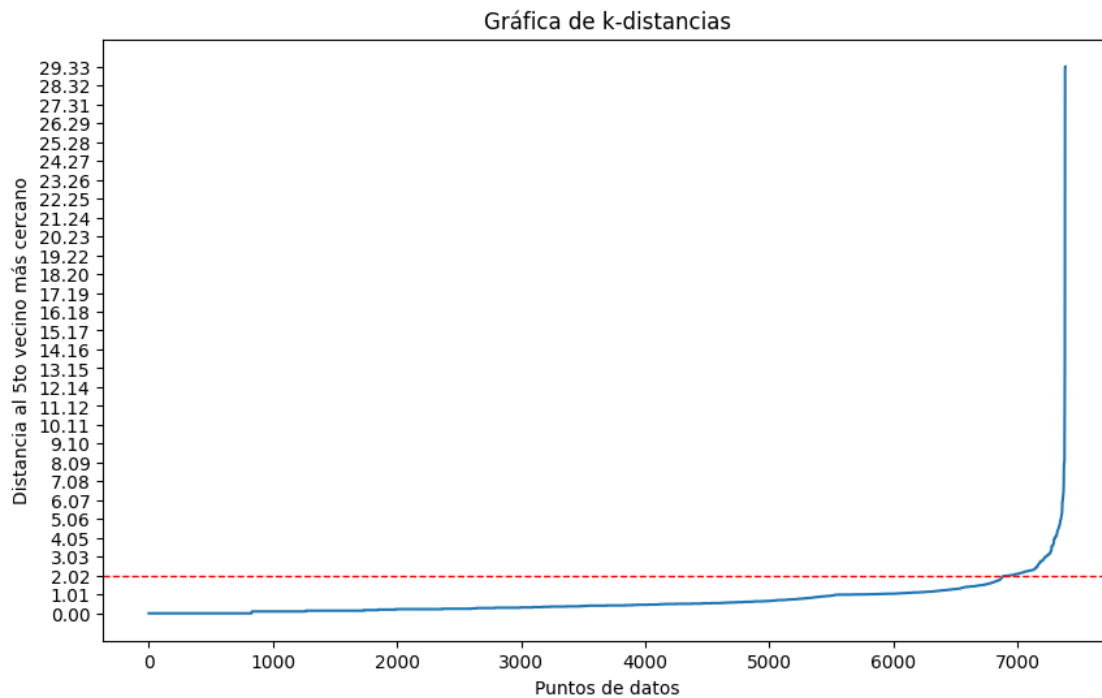
```
[61]: from sklearn.neighbors import NearestNeighbors
import numpy as np
import matplotlib.pyplot as plt

neighbors = NearestNeighbors(n_neighbors=5)
neighbors_fit = neighbors.fit(features)
distances, indices = neighbors_fit.kneighbors(features)

distances = np.sort(distances[:, 4], axis=0)
plt.figure(figsize=(10, 6))
plt.plot(distances)
plt.title('Gráfica de k-distancias')
plt.xlabel('Puntos de datos')
plt.ylabel('Distancia al 5to vecino más cercano')

yticks = np.linspace(0, max(distances), num=30) # 20 marcas en el eje y
plt.yticks(yticks)
plt.axhline(y=2, color='r', linestyle='--', linewidth=1)
```

```
plt.show()
```



Como se puede observar en la gráfica, el punto de corte o cambio de dirección para un número de 5 vecinos cercanos distribidos en la data, es aproximadamente en el valor 2. Por lo que estas observaciones se pasan como parámetros para el modelo DBSCAN.

```
[62]: from sklearn.cluster import DBSCAN

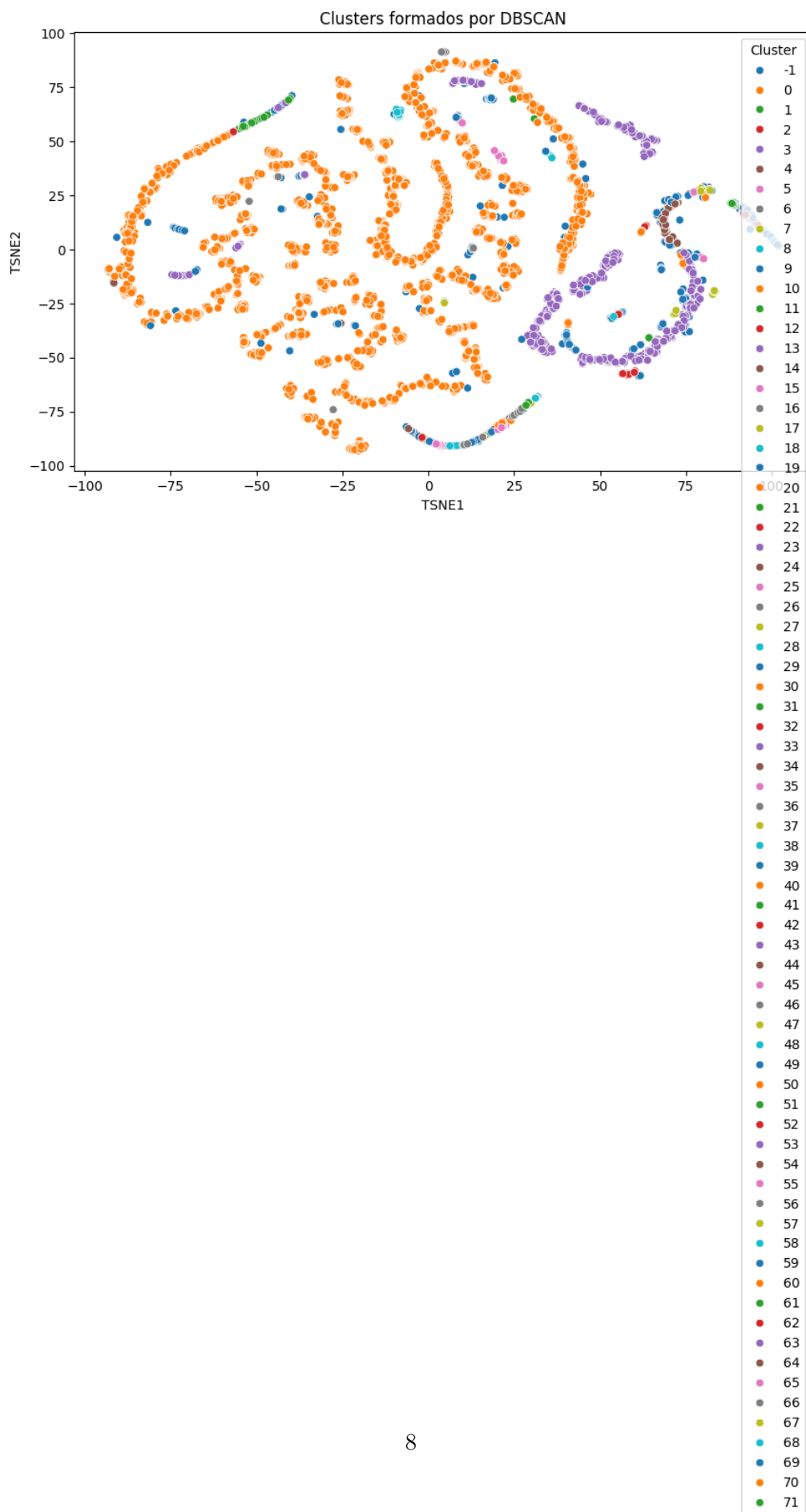
# Ajustar el modelo DBSCAN
dbscan = DBSCAN(eps=2, min_samples=5)
clusters = dbscan.fit_predict(features)

# Añadir los clusters al DataFrame original
df['Cluster'] = clusters

# Añadir los clusters al DataFrame de t-SNE
tsne_df['Cluster'] = clusters

# Visualizar los clusters en t-SNE
plt.figure(figsize=(10, 6))
sns.scatterplot(x='TSNE1', y='TSNE2', hue='Cluster', palette='tab10',
               data=tsne_df)
plt.title('Clusters formados por DBSCAN')
```

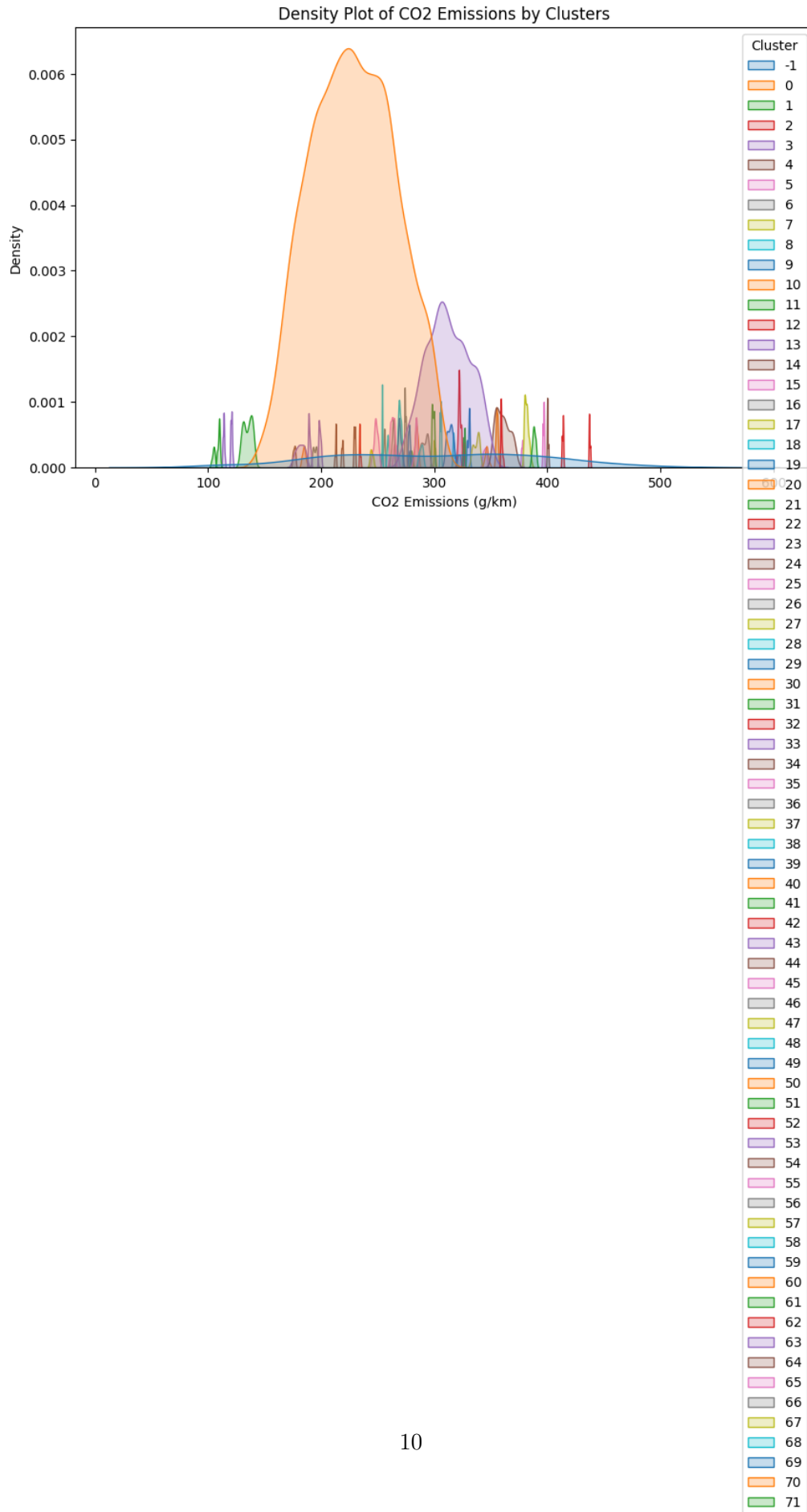
```
plt.show()
```



Density distribution CO2 emissions by cluster

```
[63]: plt.figure(figsize=(10, 6))
sns.kdeplot(data=df, x='CO2 Emissions(g/km)', hue='Cluster', palette='tab10',
            fill=True)
plt.title('Density Plot of CO2 Emissions by Clusters')
plt.xlabel('CO2 Emissions (g/km)')
plt.ylabel('Density')
plt.show()
```

```
C:\Users\emanuel.novelo\AppData\Local\Temp\ipykernel_161472\2493083992.py:2:
UserWarning: Dataset has 0 variance; skipping density estimate. Pass
`warn_singular=False` to disable this warning.
  sns.kdeplot(data=df, x='CO2 Emissions(g/km)', hue='Cluster', palette='tab10',
fill=True)
```



Como forma ilustrativa se observa la densidad (clasificada por nivel de emisiones de bajo a alto) respecto las componentes de la data (obtenidas del T-SNE). Finalmente se muestra una tabla de relación entre las features y las componentes T-SNE 1 & 2, de esta forma podemos complementar la interpretación de la gráfica sabiendo que features influyen más o menos, positiva y negativamente, a cada componente.

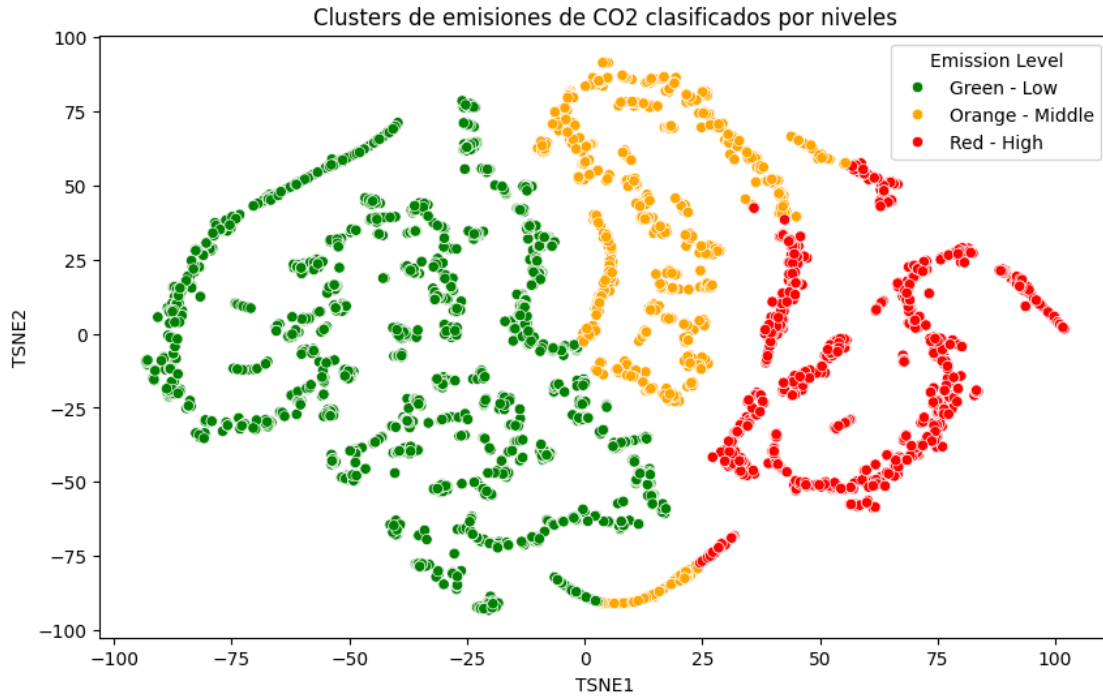
```
[64]: import numpy as np

# Clasificar las emisiones de CO2 en tres grupos usando cuantiles
quantiles = np.percentile(df['CO2 Emissions(g/km)'], [50, 75, 90])

df['Emission Level'] = pd.cut(df['CO2 Emissions(g/km)'],
                              bins=[-np.inf, quantiles[0], quantiles[1], np.
→inf],
                              labels=['Green - Low', 'Orange - Middle', 'Red -
→High'])

tsne_df['Emission Level'] = df['Emission Level']

# Visualizar los clusters en t-SNE
plt.figure(figsize=(10, 6))
sns.scatterplot(x='TSNE1', y='TSNE2', hue='Emission Level', palette=['green',
→'orange', 'red'], data=tsne_df)
plt.title('Clusters de emisiones de CO2 clasificados por niveles')
plt.show()
```



```
[65]: # Calcular la correlación entre las variables originales y las componentes t-SNE
correlations = pd.concat([features, tsne_df[['TSNE1', 'TSNE2']]], axis=1).corr()

# Mostrar las correlaciones de TSNE1 y TSNE2 con las variables originales
tsne_correlations = correlations.loc[['TSNE1', 'TSNE2'], features.columns]
tsne_correlations
```

```
[65]:
```

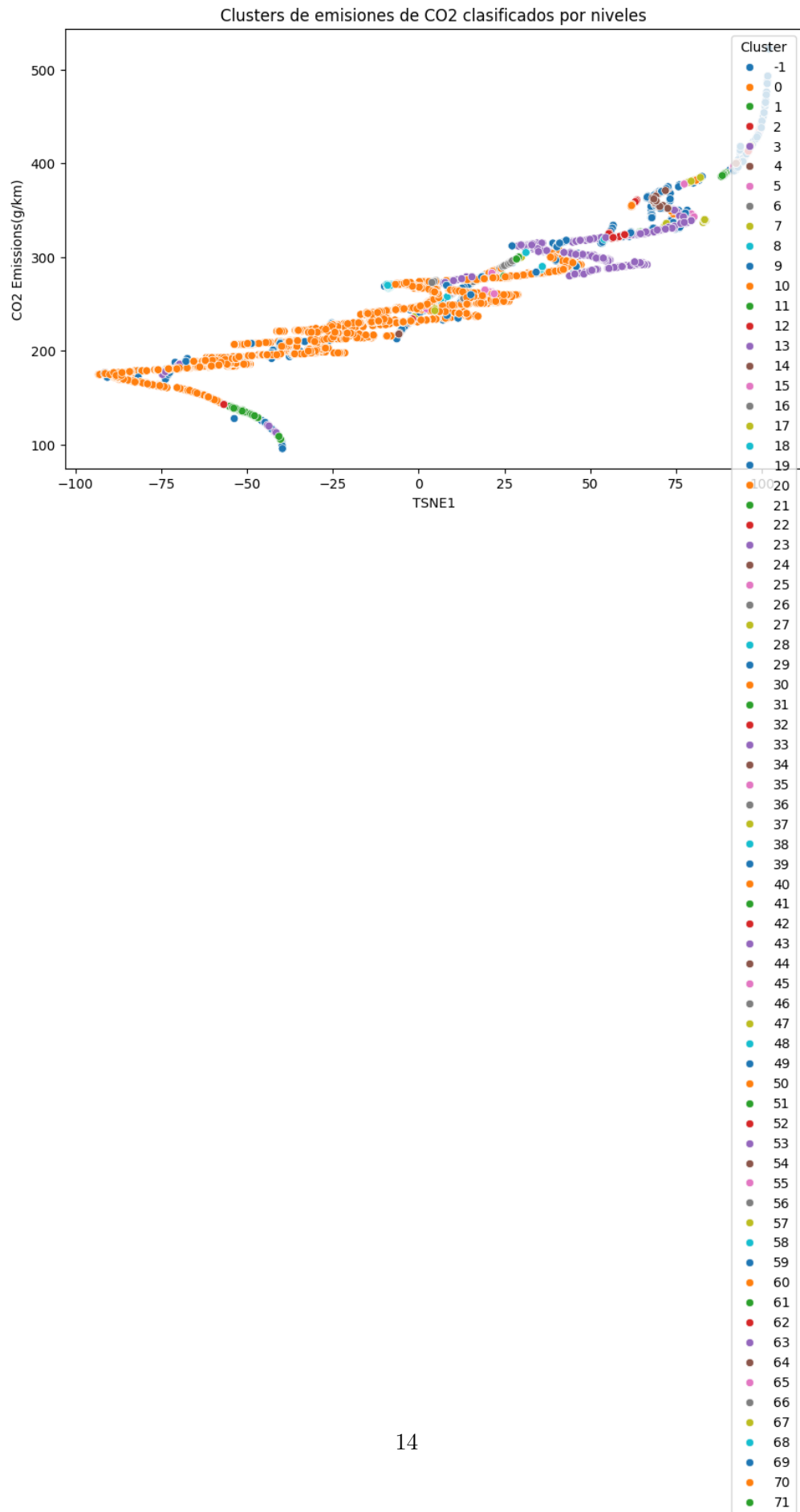
| | Engine Size(L) | Cylinders | Fuel Consumption City (L/100 km) | \ |
|-------|----------------|-----------|----------------------------------|---|
| TSNE1 | 0.834789 | 0.81222 | 0.861233 | |
| TSNE2 | -0.076133 | -0.05065 | -0.155395 | |

| | Fuel Consumption Hwy (L/100 km) | Fuel Consumption Comb (L/100 km) | \ |
|-------|---------------------------------|----------------------------------|---|
| TSNE1 | 0.827188 | 0.859778 | |
| TSNE2 | -0.134140 | -0.150245 | |

| | Fuel Consumption Comb (mpg) | CO2 Emissions(g/km) |
|-------|-----------------------------|---------------------|
| TSNE1 | -0.854690 | 0.939401 |
| TSNE2 | 0.163234 | -0.023285 |

Se puede concluir de las correlaciones (disclaimer: no es una medida súper confiable para determinar “importancia” de las features en el cálculo de componentes de T-SNE) que para el componente 1, las variables están teniendo un mayor impacto. Por lo que se gráfica el T-SNE componente 1 vs las emisiones de CO2 para hallar relaciones.

```
[66]: # Visualizar los clusters en t-SNE
plt.figure(figsize=(10, 6))
sns.scatterplot(x='TSNE1', y='CO2 Emissions(g/km)', hue='Cluster',
               ↪palette='tab10', data=tsne_df)
plt.title('Clusters de emisiones de CO2 clasificados por niveles')
plt.show()
```



Se crea un histograma de la frecuencia de cada cluster del DBSCAN en los datos. El cluster 0 es el más frecuente. Igualmente se identifican cerca de 500 observaciones consideradas como “ruido” por el algoritmo (cluster -1).

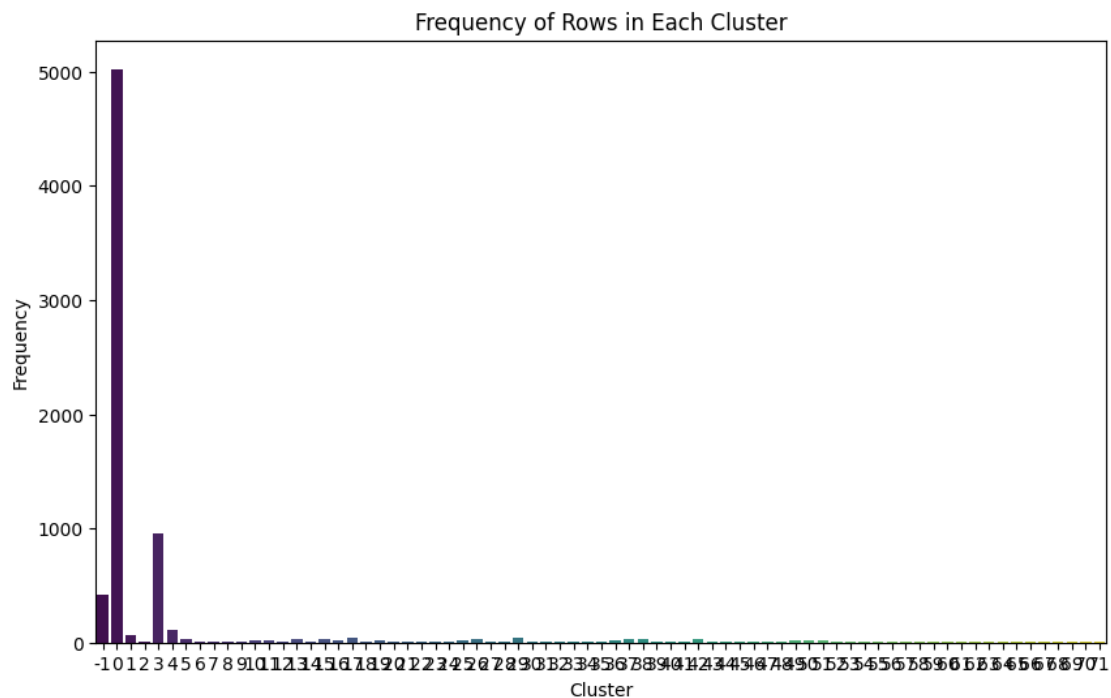
```
[67]: cluster_counts = df['Cluster'].value_counts().sort_index()

# Plot the frequency of rows in each cluster
plt.figure(figsize=(10, 6))
sns.barplot(x=cluster_counts.index, y=cluster_counts.values, palette='viridis')
plt.title('Frequency of Rows in Each Cluster')
plt.xlabel('Cluster')
plt.ylabel('Frequency')
plt.show()
```

C:\Users\emanuel.novelo\AppData\Local\Temp\ipykernel_161472\3883861907.py:5:
FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

```
sns.barplot(x=cluster_counts.index, y=cluster_counts.values,
palette='viridis')
```



Se muestra un resumen de las características e insights de las variables categóricas y numéricas de la data de emisiones de CO2, por cluster del DBSCAN.

```
[68]: summary = df.groupby('Cluster').agg({
    'CO2 Emissions(g/km)': ['mean', 'std'],
    'Make': lambda x: x.value_counts().idxmax(),
    'Model': lambda x: x.value_counts().idxmax(),
    'Vehicle Class': lambda x: x.value_counts().idxmax(),
    'Engine Size(L)': ['mean', 'std'],
    'Cylinders': ['mean', 'std'],
    'Transmission': lambda x: x.value_counts().idxmax(),
    'Fuel Type': lambda x: x.value_counts().idxmax(),
    'Fuel Consumption City (L/100 km)': ['mean', 'std'],
    'Fuel Consumption Hwy (L/100 km)': ['mean', 'std'],
    'Fuel Consumption Comb (L/100 km)': ['mean', 'std'],
    'Fuel Consumption Comb (mpg)': ['mean', 'std']
}).reset_index()

summary.columns = ['Cluster',
                   'CO2 Emissions Mean', 'CO2 Emissions Std',
                   'Most Common Make', 'Most Common Model', 'Most Common Vehicle_
→Class',
                   'Engine Size Mean', 'Engine Size Std',
                   'Cylinders Mean', 'Cylinders Std',
                   'Most Common Transmission', 'Most Common Fuel Type',
                   'Fuel Consumption City Mean', 'Fuel Consumption City Std',
                   'Fuel Consumption Hwy Mean', 'Fuel Consumption Hwy Std',
                   'Fuel Consumption Comb Mean', 'Fuel Consumption Comb Std',
                   'Fuel Consumption Comb MPG Mean', 'Fuel Consumption Comb MPG_
→Std']

summary
```

```
[68]:
```

| | Cluster | CO2 Emissions Mean | CO2 Emissions Std | Most Common Make | \ |
|----|---------|--------------------|-------------------|------------------|---|
| 0 | -1 | 299.502358 | 93.648783 | FORD | |
| 1 | 0 | 228.267370 | 36.345936 | FORD | |
| 2 | 1 | 135.045455 | 3.812658 | TOYOTA | |
| 3 | 2 | 359.363636 | 0.674200 | ASTON MARTIN | |
| 4 | 3 | 311.963312 | 18.889742 | CHEVROLET | |
| .. | ... | ... | ... | ... | |
| 68 | 67 | 419.000000 | 0.000000 | MERCEDES-BENZ | |
| 69 | 68 | 322.000000 | 1.000000 | CHEVROLET | |
| 70 | 69 | 370.000000 | 0.000000 | LAMBORGHINI | |
| 71 | 70 | 382.000000 | 0.000000 | ROLLS-ROYCE | |
| 72 | 71 | 279.800000 | 0.836660 | TOYOTA | |

| | Most Common Model | Most Common Vehicle Class | Engine Size Mean \ |
|----|-------------------|---------------------------|--------------------|
| 0 | FOCUS FFV | TWO-SEATER | 4.162264 |
| 1 | SONIC | SUV - SMALL | 2.522616 |
| 2 | ES 300h | MID-SIZE | 2.004545 |
| 3 | DB9 | MINICOMPACT | 6.072727 |
| 4 | SILVERADO 4WD | SUV - STANDARD | 5.122642 |
| .. | ... | ... | ... |
| 68 | G 550 | SUV - STANDARD | 5.220000 |
| 69 | SUBURBAN 4WD FFV | PICKUP TRUCK - STANDARD | 5.240000 |
| 70 | Huracan Coupe AWD | TWO-SEATER | 5.200000 |
| 71 | Phantom | FULL-SIZE | 6.700000 |
| 72 | TACOMA 4WD | PICKUP TRUCK - SMALL | 2.660000 |

| | Engine Size Std | Cylinders Mean | Cylinders Std | Most Common Transmission \ |
|----|-----------------|----------------|---------------|----------------------------|
| 0 | 1.728769 | 7.198113 | 2.968455 | A6 |
| 1 | 0.749912 | 4.785188 | 1.000718 | AS6 |
| 2 | 0.370211 | 3.924242 | 0.266638 | AV |
| 3 | 0.264919 | 12.000000 | 0.000000 | A6 |
| 4 | 0.698200 | 8.000000 | 0.000000 | AS8 |
| .. | ... | ... | ... | ... |
| 68 | 0.383406 | 8.000000 | 0.000000 | A6 |
| 69 | 0.134164 | 8.000000 | 0.000000 | A6 |
| 70 | 0.000000 | 10.000000 | 0.000000 | AM7 |
| 71 | 0.000000 | 12.000000 | 0.000000 | AS8 |
| 72 | 0.089443 | 4.000000 | 0.000000 | M5 |

| | Most Common Fuel Type | Fuel Consumption City Mean \ |
|----|-----------------------|------------------------------|
| 0 | Z | 16.271226 |
| 1 | X | 11.138244 |
| 2 | X | 5.710606 |
| 3 | Z | 18.372727 |
| 4 | Z | 15.624214 |
| .. | ... | ... |
| 68 | Z | 20.480000 |
| 69 | E | 21.980000 |
| 70 | Z | 17.950000 |
| 71 | Z | 20.000000 |
| 72 | X | 12.800000 |

| | Fuel Consumption City Std | Fuel Consumption Hwy Mean \ |
|----|---------------------------|-----------------------------|
| 0 | 5.724219 | 11.387736 |
| 1 | 1.874373 | 8.155883 |
| 2 | 0.304898 | 5.842424 |
| 3 | 0.374409 | 11.900000 |
| 4 | 0.972837 | 10.622222 |
| .. | ... | ... |
| 68 | 0.749667 | 15.220000 |

| | | |
|----|----------|-----------|
| 69 | 0.408656 | 16.160000 |
| 70 | 0.053452 | 12.950000 |
| 71 | 0.000000 | 11.800000 |
| 72 | 0.578792 | 11.000000 |

| | Fuel Consumption Hwy Std | Fuel Consumption Comb Mean \ |
|----|--------------------------|------------------------------|
| 0 | 3.704874 | 14.071462 |
| 1 | 1.244043 | 9.795899 |
| 2 | 0.382727 | 5.778788 |
| 3 | 0.618061 | 15.472727 |
| 4 | 0.886833 | 13.376834 |
| .. | ... | ... |
| 68 | 0.867179 | 18.140000 |
| 69 | 0.614817 | 19.360000 |
| 70 | 0.053452 | 15.700000 |
| 71 | 0.000000 | 16.300000 |
| 72 | 0.489898 | 12.000000 |

| | Fuel Consumption Comb Std | Fuel Consumption Comb MPG Mean \ |
|----|---------------------------|----------------------------------|
| 0 | 4.756994 | 23.257075 |
| 1 | 1.556013 | 29.579932 |
| 2 | 0.182727 | 48.909091 |
| 3 | 0.190215 | 18.090909 |
| 4 | 0.821228 | 21.146751 |
| .. | ... | ... |
| 68 | 0.134164 | 16.000000 |
| 69 | 0.089443 | 14.800000 |
| 70 | 0.000000 | 18.000000 |
| 71 | 0.000000 | 17.000000 |
| 72 | 0.122474 | 23.800000 |

| | Fuel Consumption Comb MPG Std |
|----|-------------------------------|
| 0 | 10.746045 |
| 1 | 4.910056 |
| 2 | 1.475203 |
| 3 | 0.301511 |
| 4 | 1.380214 |
| .. | ... |
| 68 | 0.000000 |
| 69 | 0.447214 |
| 70 | 0.000000 |
| 71 | 0.000000 |
| 72 | 0.447214 |

[73 rows x 20 columns]

2.3 Bibliografía de la sección

- <https://github.com/d0r1h/CO2-Emission-by-Cars?tab=readme-ov-file>
- <https://www.kaggle.com/datasets/debajyotipodder/co2-emission-by-vehicles>
- <https://www.kdnuggets.com/2019/10/right-clustering-algorithm.html>
- <https://www.sciencedirect.com/science/article/abs/pii/S0048969723007507>
- <https://www.datacamp.com/es/tutorial/introduction-t-sne>
- <https://iopscience.iop.org/article/10.1088/1755-1315/31/1/012012/pdf#:~:text=Sorting%20the%20results%20of%20the%20model%20training,which%20can%20be%20used%20to%20predict%20the%20results%20of%20the%20model%20training>

3 Supervised Learning - Gaussian Process Regression for CO2 predictions

Abstract de la sección: En esta sección de Aprendizaje Supervisado, se explora la aplicación del Gaussian Processes Regression, para la predicción de emisiones de CO2. Se tomó como inspiración el artículo *Can Machine Learning be Applied to Carbon Emissions Analysis: An Application to the CO2 Emissions Analysis Using Gaussian Process Regression* de Ning Ma, Wai Yan Shum y Tingting Han. Los resultados arrojan un R2 de 73% aprox, sin embargo destaca una serie de valores cuya predicción se opta por la media de la distribución (250), lo cual resulta particular. Se comparan distintas métricas como *MAE*, *MSE*, *RMSE* y *MAPE* contra las predicciones de una regresión lineal múltiple. La RLM resulta con mejor performance en general.

```
[ ]: # lectura de datos
import pandas as pd
df = pd.read_csv('C:/Users/emanuel.novelo/Desktop/MCD - 2024-2026/ML - 2do Tetra/
↳MCD----ML----2024/data/CO2 Emissions_Canada.csv')
# filtrando columnas numéricas

[1]: features = df[['Engine Size(L)', 'Cylinders', 'Fuel Consumption City (L/100 km)',
↳'Fuel Consumption Hwy (L/100 km)', 'Fuel Consumption Comb (L/
↳100 km)',
↳'Fuel Consumption Comb (mpg)']]

[2]: X = features
y = df[['CO2 Emissions(g/km)']]

[5]: from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

3.1 Metodología

3.1.1 Regresión Lineal Múltiple

La regresión lineal múltiple es una técnica estadística que modela la relación entre una variable dependiente y y múltiples variables independientes X_1, X_2, \dots, X_p . El modelo de regresión lineal múltiple puede ser representado por la siguiente fórmula:

$$y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \dots + \beta_p X_p + \epsilon$$

Donde:

- y es la variable dependiente.
- X_1, X_2, \dots, X_p son las variables independientes.
- β_0 es el intercepto del modelo.
- $\beta_1, \beta_2, \dots, \beta_p$ son los coeficientes de regresión.
- ϵ es el término de error.

Para ajustar el modelo, se utilizan los datos de entrenamiento para estimar los coeficientes β que minimizan el error cuadrático medio (MSE) entre las predicciones y los valores reales. La métrica MSE se define como:

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

Donde y_i son los valores reales y \hat{y}_i son las predicciones del modelo.

3.1.2 Regresión de Procesos Gaussianos

La regresión de procesos gaussianos (GPR) es un método bayesiano no paramétrico utilizado para la regresión. GPR asume que los datos se distribuyen según un proceso gaussiano, lo que significa que cualquier conjunto finito de datos sigue una distribución normal multivariada. El modelo de GPR está definido por una función de media $m(x)$ y una función de covarianza (o kernel) $k(x, x')$.

El proceso gaussiano se puede expresar como:

$$f(x) \sim \mathcal{GP}(m(x), k(x, x'))$$

Donde:

- $m(x)$ es la función de media, que generalmente se asume como cero.
- $k(x, x')$ es la función de covarianza o kernel, que define la relación entre los puntos de datos.

La predicción de GPR en un nuevo punto x_* se obtiene mediante la siguiente fórmula:

$$\mu_* = K(X_*, X)[K(X, X) + \sigma_n^2 I]^{-1} y$$

$$\Sigma_* = K(X_*, X_*) - K(X_*, X)[K(X, X) + \sigma_n^2 I]^{-1} K(X, X_*)$$

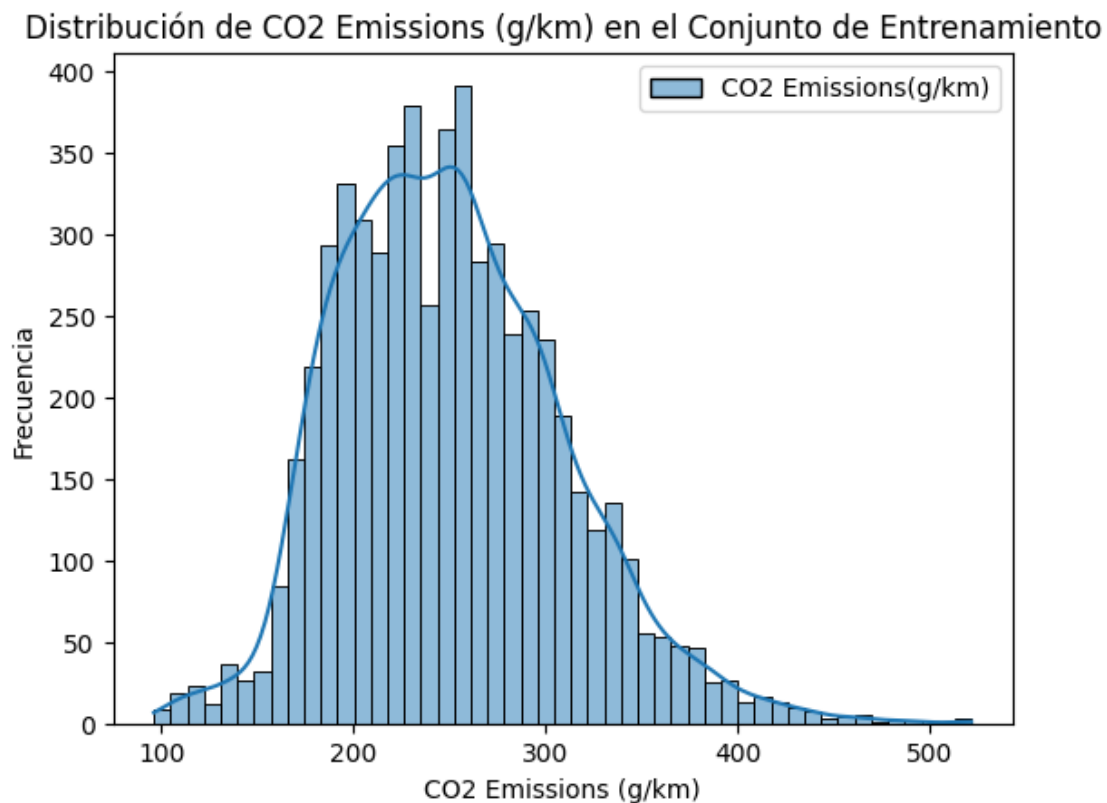
Donde:

- μ_* es la media predictiva en el punto x_* .
- Σ_* es la varianza predictiva en el punto x_* .
- K es la matriz de covarianza calculada mediante el kernel.
- σ_n^2 es la varianza del ruido en los datos.

3.2 Exploración previa

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

sns.histplot(y_train, kde=True)
plt.title('Distribución de CO2 Emissions (g/km) en el Conjunto de Entrenamiento')
plt.xlabel('CO2 Emissions (g/km)')
plt.ylabel('Frecuencia')
plt.show()
```



De la gráfica anterior de emisiones, se observa como la media de los datos se acerca al valor 250, esto tomará peso en los siguientes resultados.

3.3 Resultados & Conclusiones

Se entrena un modelo de un Proceso de Regresión Gaussiano, el cuál ha sido probado previamente en la literatura como un buen predictor de emisiones de CO2.

```
[6]: from sklearn.metrics import r2_score
from sklearn.gaussian_process import GaussianProcessRegressor
from sklearn.gaussian_process.kernels import RBF, ConstantKernel, WhiteKernel
```

```

kernel = ConstantKernel(1.0) + ConstantKernel(1.0) * RBF(10) + WhiteKernel(5)
model = GaussianProcessRegressor(kernel=kernel)
model.fit(X_train, y_train)
y_pred_tr, y_pred_tr_std = model.predict(X_train, return_std=True)
y_pred_te, y_pred_te_std = model.predict(X_test, return_std=True)

```

C:\Users\emanuel.novelo\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.11_qbz5n2kfra8p0\LocalCache\local-packages\Python311\site-packages\sklearn\gaussian_process\kernels.py:419: ConvergenceWarning: The optimal value found for dimension 0 of parameter k1__k2__length_scale is close to the specified lower bound 1e-05. Decreasing the bound and calling fit again may find a better value.

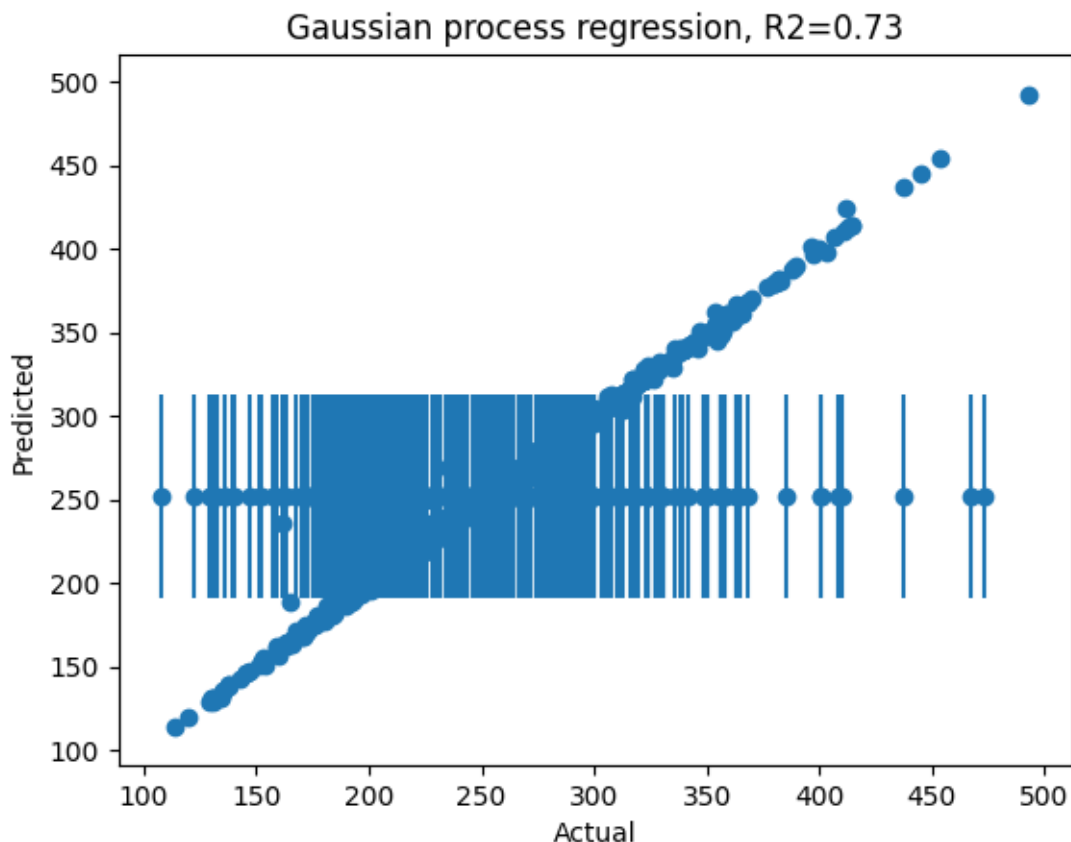
```
warnings.warn(
```

```

[7]: import matplotlib.pyplot as plt
plt.figure()
plt.errorbar(y_test, y_pred_te, yerr=y_pred_te_std, fmt='o')
plt.title('Gaussian process regression, R2=%.2f' % r2_score(y_test, y_pred_te))
plt.xlabel('Actual')
plt.ylabel('Predicted')

```

```
[7]: Text(0, 0.5, 'Predicted')
```



Los resultados del modelo indican un ajuste de 73% sobre los datos predichos vs observados, sin embargo se destaca una serie de predicciones que resultaron en la media de la distribución, de alrededor de 250, comparado contra las observaciones reales se puede percibir una recta horizontal, se discutirá más adelante en este paper las posibles causas.

```
[10]: from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
import numpy as np
y_test_np = y_test.to_numpy().flatten()
y_pred_te_np = y_pred_te.flatten()

# Calcular las métricas
mae = mean_absolute_error(y_test_np, y_pred_te_np)
mse = mean_squared_error(y_test_np, y_pred_te_np)
rmse = np.sqrt(mse)
mape = np.mean(np.abs((y_test_np - y_pred_te_np) / y_test_np)) * 100

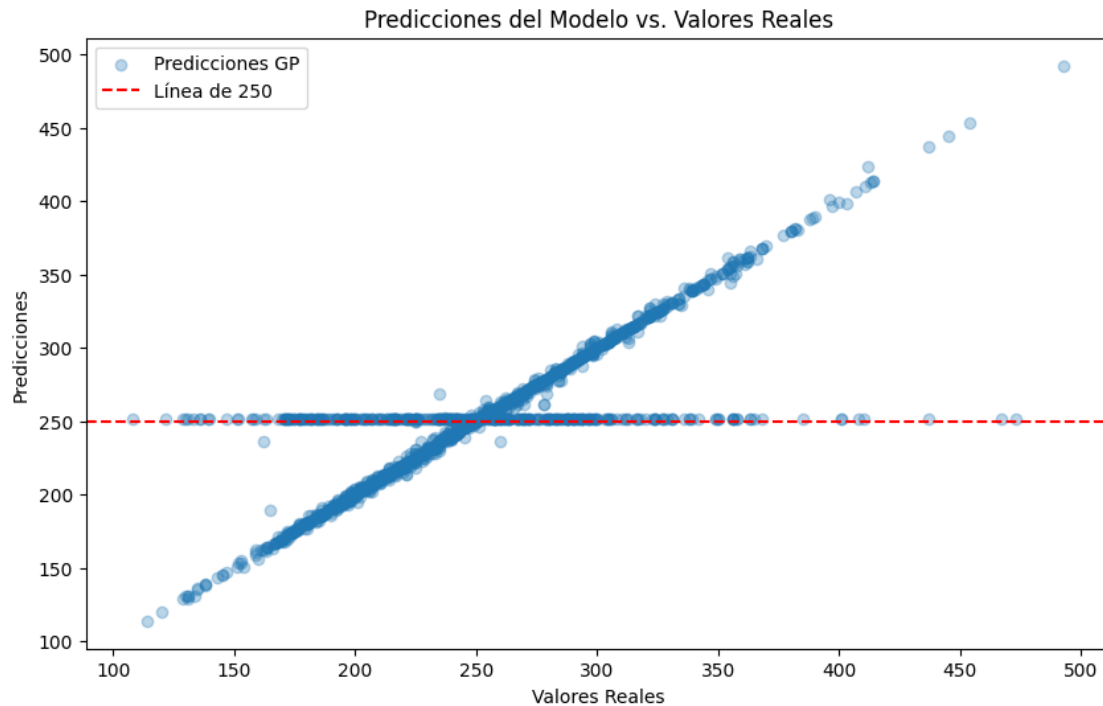
# Crear DataFrame con las métricas
metrics_df = pd.DataFrame({
    'Metric': ['MAE', 'MSE', 'RMSE', 'MAPE'],
    'Value': [mae, mse, rmse, mape]
})
metrics_df
```

```
[10]:
```

| | Metric | Value |
|---|--------|------------|
| 0 | MAE | 12.483388 |
| 1 | MSE | 858.493804 |
| 2 | RMSE | 29.300065 |
| 3 | MAPE | 5.491289 |

Se muestran las métricas de desempeño, los resultados per se podrían decirse no tan negativos, sin embargo es preferible que sean comparados con observaciones similares y resultados de métricas de la literatura.

```
[14]: plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred_te, alpha=0.3, label='Predicciones GP')
plt.axhline(y=250, color='r', linestyle='--', label='Línea de 250')
plt.xlabel('Valores Reales')
plt.ylabel('Predicciones')
plt.title('Predicciones del Modelo vs. Valores Reales')
plt.legend()
plt.show()
```



```
[15]: correlations = features.corrwith(df['CO2 Emissions(g/km)'])
      print(correlations)
```

```
Engine Size(L)          0.851145
Cylinders               0.832644
Fuel Consumption City (L/100 km)  0.919592
Fuel Consumption Hwy (L/100 km)  0.883536
Fuel Consumption Comb (L/100 km)  0.918052
Fuel Consumption Comb (mpg)      -0.907426
dtype: float64
```

Se muestran las correlaciones. las cuales muestran fuerte relación de las variables numéricas hacia el resultado a predecir, que es la emisión de CO2.

```
[11]: from sklearn.linear_model import LinearRegression

linear_model = LinearRegression()
linear_model.fit(X_train, y_train)

# predecir
y_pred_linear = linear_model.predict(X_test)

# calcular métricas para regresión lineal
mae_linear = mean_absolute_error(y_test, y_pred_linear)
mse_linear = mean_squared_error(y_test, y_pred_linear)
```



```

rmse_linear = np.sqrt(mse_linear)
mape_linear = np.mean(np.abs((y_test.to_numpy().flatten() - y_pred_linear.
    ↪flatten()) / y_test.to_numpy().flatten())) * 100
r2_linear = r2_score(y_test, y_pred_linear)

# crear DataFrame con las métricas para regresión lineal
metrics_linear_df = pd.DataFrame({
    'Metric': ['MAE', 'MSE', 'RMSE', 'MAPE', 'R2'],
    'Linear Regression': [mae_linear, mse_linear, rmse_linear, mape_linear,
    ↪r2_linear]
})

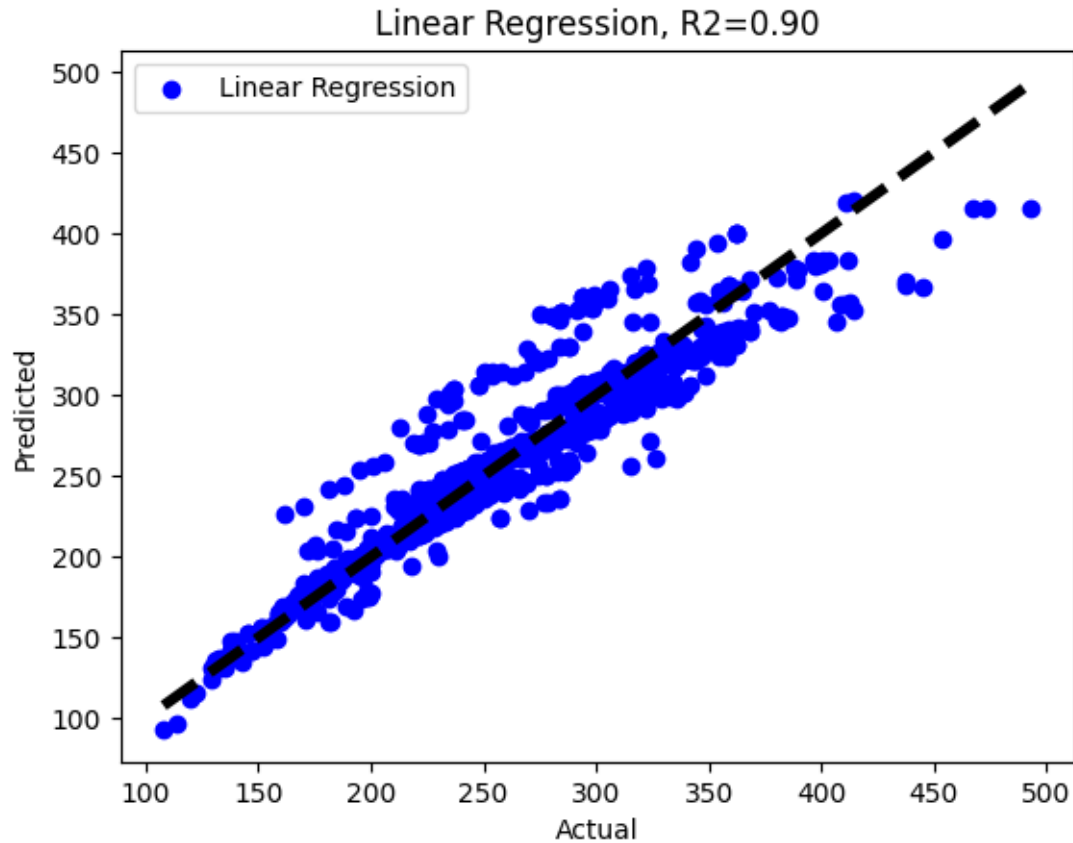
# crear DataFrame con las métricas para regresión gaussiana
metrics_gp_df = pd.DataFrame({
    'Metric': ['MAE', 'MSE', 'RMSE', 'MAPE', 'R2'],
    'Gaussian Process': [mae, mse, rmse, mape, r2_score(y_test_np, y_pred_te_np)]
})

# combinar los DataFrames para comparación
comparison_df = pd.merge(metrics_linear_df, metrics_gp_df, on='Metric')
print(comparison_df)

# visualizar las predicciones de la regresión lineal múltiple
plt.figure()
plt.scatter(y_test, y_pred_linear, color='blue', label='Linear Regression')
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'k--', lw=4)
plt.xlabel('Actual')
plt.ylabel('Predicted')
plt.title('Linear Regression, R2=%.2f' % r2_linear)
plt.legend()
plt.show()

```

| | Metric | Linear Regression | Gaussian Process |
|---|--------|-------------------|------------------|
| 0 | MAE | 10.967599 | 12.483388 |
| 1 | MSE | 312.066899 | 858.493804 |
| 2 | RMSE | 17.665415 | 29.300065 |
| 3 | MAPE | 4.199974 | 5.491289 |
| 4 | R2 | 0.903524 | 0.734595 |



Se comparan los resultados de performance del GPR vs una Regresión Lineal Múltiple, para notar que en este caso particular, un modelo más avanzado no resulta necesariamente mejor que la estadística tradicional, sin embargo, cabe destacar y dejar muy claro que no se realizaron esfuerzos de hyperparameter tuning, feature engineering, dimensionality reduction, ni parameter optimization, que pudieran resultar en un performance de índole superior a favor del modelo GPR.

3.4 Bibliografía de la sección

- <https://www.frontiersin.org/articles/10.3389/fenrg.2021.756311/full>
- <https://towardsdatascience.com/getting-started-with-gaussian-process-regression-modeling-47e7982b534d>