In [7]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
import os
os.chdir('/Users/Lenovo/Desktop/EBAC')
```

In [27]:
```
fuel = pd.read_excel('FuelConsumptionCo2.xlsx')
fuel.head()
```

Out[27]:

| | MODELYEAR | MAKE | MODEL | VEHICLECLASS | ENGINESIZE | CYLINDERS | TRANSMISSION | FUELTYPE | FUELCONSUMPTION_CIT |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 2022 | Acura | ILX | Compact | 2.4 | 4 | AM8 | Z | 9 |
| **1** | 2022 | Acura | MDX SH-AWD | SUV: Small | 3.5 | 6 | AS10 | Z | 12 |
| **2** | 2022 | Acura | RDX SH-AWD | SUV: Small | 2.0 | 4 | AS10 | Z | 11 |
| **3** | 2022 | Acura | RDX SH-AWD A-SPEC | SUV: Small | 2.0 | 4 | AS10 | Z | 11 |
| **4** | 2022 | Acura | TLX SH-AWD | Compact | 2.0 | 4 | AS10 | Z | 11 |

In [29]:
```
# Remover caracteristicas categoricas
fuel.drop(['MODELYEAR','MAKE','MODEL','VEHICLECLASS', 'ENGINESIZE', 'CYLINDERS', 'TRANSMISSION', 'FUELTYPE'], a
#Remover renglones con valores faltantes
fuel.dropna(inplace=True)
fuel
```

Out[29]:

| | FUELCONSUMPTION_CITY | FUELCONSUMPTION_HWY | FUELCONSUMPTION_COMB | FUELCONSUMPTION_COMB_MPG | CO2EMI |
|---|---|---|---|---|---|
| **0** | 9.9 | 7.0 | 8.6 | 33 | |
| **1** | 12.6 | 9.4 | 11.2 | 25 | |
| **2** | 11.0 | 8.6 | 9.9 | 29 | |
| **3** | 11.3 | 9.1 | 10.3 | 27 | |
| **4** | 11.2 | 8.0 | 9.8 | 29 | |
| **...** | ... | ... | ... | ... | |
| **940** | 10.7 | 7.7 | 9.4 | 30 | |
| **941** | 10.5 | 8.1 | 9.4 | 30 | |
| **942** | 11.0 | 8.7 | 9.9 | 29 | |
| **943** | 11.5 | 8.4 | 10.1 | 28 | |
| **944** | 12.4 | 8.9 | 10.8 | 26 | |

945 rows × 5 columns

In [31]:
```
plt.figure(figsize=(20,6))
sns.heatmap(fuel.isnull(), yticklabels= False, cbar= False, cmap= 'viridis')
```

Out[31]:
```
<Axes: >
```


In [33]:
```
X = fuel.drop('CO2EMISSIONS', axis = 1)
y = fuel['CO2EMISSIONS']
```

In [35]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 1)
```

In [37]:
```
from sklearn.linear_model import LinearRegression
linreg = LinearRegression()
linreg.fit(X_train, y_train)
```

Out[37]:

▾ LinearRegression

```
LinearRegression()
```

In [39]:
```
# Impresion de coeficientes de regresion
print('Intercepto: ', linreg.intercept_)
print('Coeficientes: ', linreg.coef_)
```

```
Intercepto:  95.93361319267356
Coeficientes:  [10.85607131  5.91025898  0.71589393 -1.33087138]
```

In [41]:
```
# Predicciones
y_pred =  linreg.predict(X_test)
y_pred
```

Out[41]:
```
array([323.9684702 , 318.99111862, 351.63995823, 174.32801949,
       196.22179696, 246.30366202, 249.80010686, 327.69501755,
       323.43951275, 172.67624174, 272.69662047, 221.15870395,
       322.0301954 , 263.34932072, 183.81461788, 267.43679146,
       162.2817638 , 224.20058458, 334.07766686, 320.6056832 ,
       318.99111862, 340.5415991 , 254.69901248, 224.20058458,
       338.96141074, 205.27024177, 368.03937916, 321.36758011,
       297.85515221, 130.73051692, 180.24094285, 183.81461788,
       247.36441388, 296.30265608, 318.85746077, 280.22130457,
       326.34493169, 309.94267381, 323.43951275, 223.34507996,
       296.33986926, 284.3860055 , 235.0688412 , 185.37275481,
       170.09172917, 255.95265367, 194.22831029, 341.07055655,
       143.31137086, 184.67012251, 295.61518552, 246.56814074,
       170.18817383, 258.29190199, 109.70865471, 313.74361439,
       341.56513778, 333.81318814, 304.22547674, 371.84035287,
       198.65748995, 154.71706269, 189.63109657, 311.02828094,
       187.55632698, 349.22628356, 190.29371187, 165.36085761,
       252.95079006, 320.34120447, 305.80566511, 273.09475704,
       226.74508014, 179.64991695, 243.83075586, 247.72533727,
       318.99111862, 222.85049873, 256.08631151, 267.70127019,
       283.26318519, 341.20421439, 217.4321566 , 234.07967874,
       277.18222778, 234.67070464, 116.56947462, 289.00527065,
       242.63354232, 214.77605464, 231.21147297, 188.50827627,
       140.9569608 , 213.30747268, 285.14506546, 172.27810517,
       336.01877862, 182.33087419, 267.37472301, 247.95543977,
       285.90412542, 227.04957588, 210.097558  , 178.85364382,
       234.80436248, 247.13431137, 191.98550664, 182.33087419,
       354.37450616, 225.26133644, 223.34507996, 246.37525141,
       341.07055655, 209.5065321 , 199.41654991, 301.35440089,
       218.87868712, 347.60888202, 218.0231825 , 298.12246789,
       255.65096177, 320.6056832 , 210.6885839 , 174.42446416,
       263.71024411, 249.60721753, 177.27345545, 202.48048197,
       230.90978107, 235.0688412 ,  80.68515357, 293.37238186,
       194.81933619, 264.10838068, 354.37450616, 354.86908739,
       183.81461788, 316.34734143, 299.14884354, 255.95265367,
       354.86908739, 220.26598615, 341.73597568, 185.27631015,
       270.64670615, 325.12850368, 259.28106445, 195.32907916,
       238.14793501, 248.08909762, 216.08207075, 298.3525704 ,
       108.81593691, 333.18494906, 284.3860055 , 192.48008787,
       276.98933845, 208.97473769, 215.68393418, 233.01892688,
       270.64670615, 265.95304777, 257.39918419, 265.39639808,
       285.80768075, 192.87822444, 347.70532669, 248.08909762,
       351.10816382, 234.07967874, 254.69901248, 183.41648132,
       298.67911758, 255.82183279, 305.78361367, 374.84221648,
       271.61101334, 360.68272912, 283.6833732 , 137.38328576,
       223.41666935, 260.38183333, 262.72108165, 190.88473776,
       214.58316531, 376.36033639, 182.92190009, 199.45376308,
       263.11921821, 261.27455113, 225.81798612, 369.29302035,
       240.78887523, 308.29089605, 199.05562652, 257.13470547,
       222.65760939, 248.58084189, 296.30265608, 289.26974938,
       252.64909816, 254.39732058, 214.39307981, 376.36033639,
       278.83400554, 276.82130439, 210.19400266, 333.18494906,
       320.6056832 , 272.86465453, 224.59872115, 315.42024742,
       292.3832194 , 220.56767806, 280.22130457, 238.01427717,
       232.06697759, 161.02812261, 272.10559457, 354.37450616,
       246.17000418, 281.02038154, 218.51776373, 219.48487476,
       231.83687509, 192.24998536, 320.47486231, 197.80198533,
       233.32061878, 271.84111585, 212.43680632, 298.08809168,
       263.34932072, 205.93285706, 254.69901248, 273.15398853,
       289.26974938, 190.39015653, 191.98550664, 273.81944078,
       218.87868712, 218.42131907, 286.09701475, 314.62117045,
       218.78224246, 217.4321566 , 235.26173054, 318.99111862,
       332.86123885, 173.26726763, 247.13431137, 188.73837878,
       244.49337115, 245.21805489, 252.13935518, 253.80629468,
       213.09942161, 320.11110196, 178.98730166, 293.37238186,
       212.11025914, 330.64045351, 325.12566672, 249.60721753,
       154.28171295, 236.66419131, 215.28579761, 230.08865266,
       223.41666935, 182.33087419, 277.90691152, 157.75894332,
       268.37921981, 247.13431137, 267.37472301, 226.44338824,
       322.84848685, 265.59212438, 250.26983282, 183.18637881,
       263.71024411, 236.85708064, 246.56814074, 236.76063598])
```

In [43]:
```python
# Calculo de indicadores de bondad de ajuste
from sklearn.metrics import r2_score
from sklearn import metrics
# Impresion de indicadores de bondad de ajuste
print("Valor de R cuadrada", r2_score(y_test, y_pred))
print('Error absoluto medio: ', metrics.mean_absolute_error(y_test, y_pred))
print('Error cuadratico medio: ', metrics.mean_squared_error(y_test, y_pred))
print('Raiz del Error cuadratico medio: ', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

```
Valor de R cuadrada 0.9809388418270502
Error absoluto medio:  5.694519664627076
Error cuadratico medio:  72.94435631403724
Raiz del Error cuadratico medio:  8.540746824138814
```

# Modelo de Ridge

In [48]:
```
from sklearn.linear_model import RidgeCV
# Definicion un rango de prueba para Alpha
alpha_range = 10.**np.arange(-2,3)
ridgeregcv = RidgeCV(alphas = alpha_range, scoring = 'neg_mean_squared_error')
ridgeregcv.fit(X_train, y_train)
ridgeregcv.alpha_
```

Out[48]:
```
10.0
```

In [50]:
```
# Prediccion mediante el valor de Alpha
y_pred = ridgeregcv.predict(X_test)

# Impresion de indicadores de bondad de ajuste
print("Valor de R cuadrada", r2_score(y_test, y_pred))
print('Error absoluto medio: ', metrics.mean_absolute_error(y_test, y_pred))
print('Error cuadratico medio: ', metrics.mean_squared_error(y_test, y_pred))
print('Raiz del Error cuadratico medio: ', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

```
Valor de R cuadrada 0.9808997033817982
Error absoluto medio:  5.709158252323924
Error cuadratico medio:  73.09413360826727
Raiz del Error cuadratico medio:  8.549510723326058
```

In [52]:
```
# Examinar coeficientes de la regresion Ridge
print('Intercepto: ', ridgeregcv.intercept_)
print('Coeficientes: ', ridgeregcv.coef_)
```

```
Intercepto:  97.12499222320125
Coeficientes:  [ 8.07997596  3.67823551  5.67297035 -1.35146758]
```

# Modelo de Lasso

In [55]:
```
from sklearn.linear_model import LassoCV
lassoregcv = LassoCV(n_alphas = 100, random_state = 1)
lassoregcv.fit(X_train, y_train)
print('Alpha Optimo: ', lassoregcv.alpha_)
```

```
Alpha Optimo:  0.45442206714715033
```

In [57]:
```
# Examinar coeficientes de la regresion Lasso
print('Intercepto: ', lassoregcv.intercept_)
print('Coeficientes: ', lassoregcv.coef_)
```

```
Intercepto:  99.64116220383292
Coeficientes:  [11.10723214  5.8512792   0.28317683 -1.38536394]
```

In [59]:
```
# Prediccion mediante regresion de Lasso con un Alpha Optimo
y_pred = lassoregcv.predict(X_test)

# Impresion de indicadores de bondad de ajuste
print("Valor de R cuadrada", r2_score(y_test, y_pred))
print('Error absoluto medio: ', metrics.mean_absolute_error(y_test, y_pred))
print('Error cuadratico medio: ', metrics.mean_squared_error(y_test, y_pred))
print('Raiz del Error cuadratico medio: ', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

```
Valor de R cuadrada 0.980417172261453
Error absoluto medio:  5.842640097466095
Error cuadratico medio:  74.94071195653494
Raiz del Error cuadratico medio:  8.656830364315507
```

Luego de desarrollar los 3 modelos, llegue a la conclusion que en este caso el modelo que tiene la R2 mas cerca del 1, fue la regresion lineal multiple. Aunque estamos hablando de milesimas de diferencia, lo que en este caso, tendria la confianza de usar cualquiera de los 3.