```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

```
df=pd.read_csv('/content/Fuel_Consumption_2000-2022[1].csv')
df.head(20)
```

	YEAR	MAKE	MODEL	VEHICLE CLASS	ENGINE SIZE	CYLINDERS	TRANSMISSION	FUEL	FUEL CONSUMPTION	HWY (L/100 km)
0	2000	ACURA	1.6EL	COMPACT	1.6	4	A4	X	9.2	6.7
1	2000	ACURA	1.6EL	COMPACT	1.6	4	M5	X	8.5	6.5
2	2000	ACURA	3.2TL	MID-SIZE	3.2	6	AS5	Z	12.2	7.4
3	2000	ACURA	3.5RL	MID-SIZE	3.5	6	A4	Z	13.4	9.2
4	2000	ACURA	INTEGRA	SUBCOMPACT	1.8	4	A4	X	10.0	7.0
5	2000	ACURA	INTEGRA	SUBCOMPACT	1.8	4	M5	X	9.3	6.8
6	2000	ACURA	INTEGRA GSR/TYPE R	SUBCOMPACT	1.8	4	M5	Z	9.4	7.0
7	2000	ACURA	NSX	SUBCOMPACT	3.0	6	AS4	Z	13.6	9.2
8	2000	ACURA	NSX	SUBCOMPACT	3.2	6	M6	Z	13.8	9.1
9	2000	AUDI	A4	COMPACT	1.8	4	A5	Z	11.4	7.2
10	2000	AUDI	A4	COMPACT	1.8	4	M5	Z	9.7	6.8
11	2000	AUDI	A4	COMPACT	2.8	6	A5	Z	13.0	8.2
12	2000	AUDI	A4	COMPACT	2.8	6	M5	Z	11.7	7.5
13	2000	AUDI	A4 QUATTRO	COMPACT	1.8	4	A5	Z	12.1	7.7
14	2000	AUDI	A4 QUATTRO	COMPACT	1.8	4	M5	Z	10.7	7.5
15	2000	AUDI	A4 QUATTRO	COMPACT	2.8	6	A5	Z	13.3	8.5
16	2000	AUDI	A4 QUATTRO	COMPACT	2.8	6	M5	Z	12.7	8.7

```
df.tail(15)
```

Data processing

km)

```
df.shape
```

```
(22556, 13)
```

22542 2022 Volkswagen 4MOTION Small 2.0 4 AS8 X 10.6 8.0

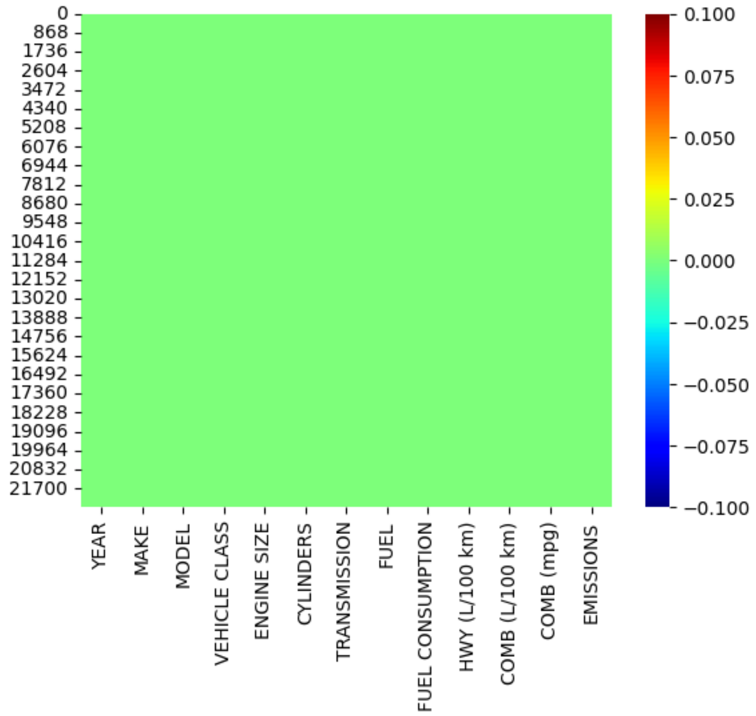
```
#check for null values
df.isna().sum()
```

```
YEAR                0
MAKE                0
MODEL               0
VEHICLE CLASS       0
ENGINE SIZE         0
CYLINDERS           0
TRANSMISSION        0
FUEL                0
FUEL CONSUMPTION    0
HWY (L/100 km)      0
COMB (L/100 km)     0
COMB (mpg)          0
EMISSIONS           0
dtype: int64
```

Small

```
#plot heatmap for null values
sns.heatmap(df.isnull(), cmap='jet')
```

```
<Axes: >
```

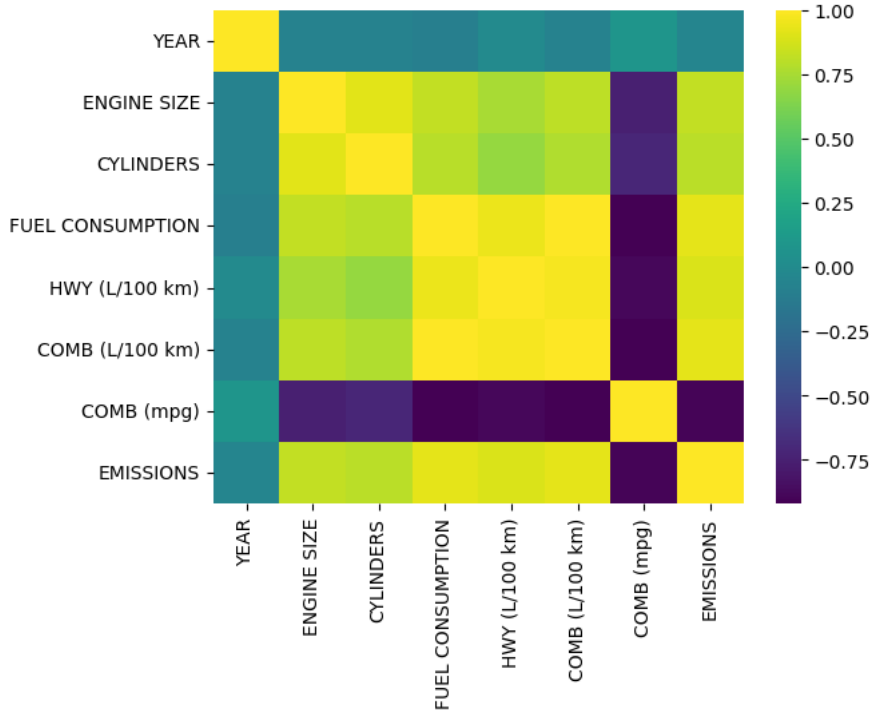



```
#correlation
corr=df.corr()
corr
```

```
<ipython-input-169-78aeb47f0e90>:2: FutureWarning: The default value of numeric_only in DataFrame.corr
```

```
#heatmap of correlation
sns.heatmap(corr, cmap='viridis')
```

```
<Axes: >
```




```
df.dtypes
```

```
YEAR                  int64
MAKE                 object
MODEL                object
VEHICLE CLASS        object
ENGINE SIZE         float64
CYLINDERS             int64
TRANSMISSION         object
FUEL                 object
FUEL CONSUMPTION    float64
HWY (L/100 km)      float64
COMB (L/100 km)     float64
COMB (mpg)            int64
EMISSIONS             int64
dtype: object
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 22556 entries, 0 to 22555
Data columns (total 13 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   YEAR              22556 non-null  int64  
 1   MAKE              22556 non-null  object 
 2   MODEL             22556 non-null  object 
 3   VEHICLE CLASS     22556 non-null  object 
 4   ENGINE SIZE       22556 non-null  float64
 5   CYLINDERS         22556 non-null  int64  
 6   TRANSMISSION      22556 non-null  object 
 7   FUEL              22556 non-null  object 
 8   FUEL CONSUMPTION  22556 non-null  float64
 9   HWY (L/100 km)    22556 non-null  float64
 10  COMB (L/100 km)   22556 non-null  float64
 11  COMB (mpg)        22556 non-null  int64  
 12  EMISSIONS         22556 non-null  int64  
dtypes: float64(4), int64(4), object(5)
memory usage: 2.2+ MB
```

```
#I just renamed the column 'MAKE' to 'Company' for better understanding
df.rename(columns={'MAKE': 'Company'}, inplace=True)
df
```

	YEAR	Company	MODEL	VEHICLE CLASS	ENGINE SIZE	CYLINDERS	TRANSMISSION	FUEL	FUEL CONSUMPTION	HW (L/1(kr
0	2000	ACURA	1.6EL	COMPACT	1.6	4	A4	X	9.2	6
1	2000	ACURA	1.6EL	COMPACT	1.6	4	M5	X	8.5	6
2	2000	ACURA	3.2TL	MID-SIZE	3.2	6	AS5	Z	12.2	7
3	2000	ACURA	3.5RL	MID-SIZE	3.5	6	A4	Z	13.4	9
4	2000	ACURA	INTEGRA	SUBCOMPACT	1.8	4	A4	X	10.0	7
...	...	...	...	...	...	...	...	...	...	
22551	2022	Volvo	XC40 T5 AWD	SUV: Small	2.0	4	AS8	Z	10.7	7
22552	2022	Volvo	XC60 B5 AWD	SUV: Small	2.0	4	AS8	Z	10.5	8

```
#unique values in each column]
cols=df.columns
for cols in (df.columns):
    print ("Unique columns of", cols, "\n", df[cols].unique())
    print ("--------------")
```

```
 28.1 27.3 28.3 26.5]
--------------
Unique columns of HWY (L/100 km)
 [ 6.7  6.5  7.4  9.2  7.   6.8  9.1  7.2  8.2  7.5  7.7  8.5  8.7  8.6
  9.   7.6  7.3  8.9  7.8  7.9  8.4  8.3  8.8 10.1 10.8 10.4  8.  11.1
  8.1 13.4 11.4 11.9  9.9 12.4 12.3 11.  12.  13.6  7.1  6.9 11.8 12.7
 12.2 12.5  4.5  6.2  4.9 12.6  9.7 10.9  9.5  5.9  6.4  6.  12.1 10.2
  9.6 13.2 10.7 10.3 11.7 11.5 13.3 13.  12.9 11.6 13.7 14.  13.5 14.5
 16.6 11.3  6.6 13.1 11.2 10.6  9.3  9.4 10.5  6.1  5.8  3.2 10.   9.8
 14.7  5.4  5.7  5.3  6.3  5.6  5.5  4.8  4.4 12.8 13.8 14.9 15.4  5.2
  4.7 16.4 15.9 15.7 16.9 15.8 16.2 16.3 14.4 14.3 14.8 16.1 17.7 16.
 15.  17.6 15.5 15.1  4.6 13.9  5.1  4.3  5.  14.1 14.6 14.2  3.3  4.2
 17.9 19.   3.8 15.6 16.5 17.3 17.4 17.  15.2 15.3 16.8 17.1  4.  16.7
 17.5 17.2 20.5 20.6 18.6 18.5 18.  18.1 20.9  4.1  3.9 19.6]
--------------
Unique columns of COMB (L/100 km)
 [ 8.1  7.6 10.  11.5  8.6  8.2  8.3 11.6 11.7  9.5  8.4 10.8  9.8 10.1
  9.3 11.1 10.9 11.2 11.3 10.7 10.3  9.7 10.4 12.2 14.4 10.6 14.6 10.5
 11.   9.9 16.6 13.4 13.8 12.8 14.1 12.5 14.9 14.7 12.7 13.1 14.3 16.8
 10.2  8.7  8.9  9.4  8.8  9.1 14.2 15.3 16.4 14.  15.6  5.3  7.1  5.7
 15.  15.1 12.  12.3 13.2 12.6  9.6  8.5  7.4  7.7  9.  15.7 13.6 13.9
 16.3 14.8 17.2 11.8 12.4 13.3 15.2 16.1 16.9 14.5 18.4 18.3 19.9 19.2
 22.7 11.4 15.5 12.9 13.7 13.5 13.  11.9 16.7  7.5  6.8  3.6  7.8  7.3
  9.2 12.1 15.4 17.7  7.9  6.7  7.   6.6  6.3  6.   5.1 15.8 16.  16.5
 17.  16.2 17.8  8.  18.5 17.5 17.4  7.2  6.4  6.1  4.6  6.9 20.7 17.9
 18.2 20.9 18.6 19.  19.1 17.1 17.3 20.6  6.5 19.3 19.7 20.3 19.8 24.8
  4.8  5.9  5.  19.5 18.   4.1  5.5  5.6 17.6 18.7 21.3 23.2 22.3 20.4
 18.1 19.4  4.2 20.  15.9  4.5 18.9 19.6 20.1 21.4 21.7  5.8 18.8 20.5
  5.4 22.1 20.2  4.7 21.   3.8  6.2 23.  23.1 20.8 21.5 23.3  4.3  4.9
  3.7  5.2  4.4 25.9 26.1 22.2 22.9  4.  22.4 21.2]
--------------
Unique columns of COMB (mpg)
 [35 37 28 25 33 34 24 30 26 29 27 23 20 19 17 21 22 32 31 18 53 40 50 38
 16 15 14 12 42 78 36 39 43 45 47 55 44 46 61 41 11 59 48 56 69 51 13 67
 63 49 52 60 74 66 58 76 54 64 71]
--------------
Unique columns of EMISSIONS
 [186 175 230 264 198 189 191 267 269 218 193 248 225 232 214 255 251 258
 260 246 237 223 239 281 331 244 336 242 253 228 382 308 317 294 324 288
 343 338 292 301 329 386 235 200 205 216 202 209 327 352 377 322 359 122
 163 131 345 347 276 283 304 290 221 196 170 177 207 313 320 375 340 396
 271 285 306 350 370 389 334 314 423 421 458 442 522 262 356 297 315 310
 299 274 384 361 172 156  83 179 168 212 278 354 407 182 154 161 152 145
 162 138 363 243 368 380 391 316 319 373 184 426 402 400 166 147 140 106
 159 409 318 412 344 428 393 398 321 330 150 444 467 208 455 570 312 259
 227 110 136 165 135 307 291 346  94 148 151 405 270 341 296 371 513 469
 419 453 414 275 229 265 113 272 197 286 366 104 430 173 305 337 222 233
 339 238 282 302 326 277 298 266 342 416 289 181 446 490 335 178 133 219
 203 293 328 273 124 174 280 254 210 508 323 187 261 245 115 309 108  87
 143 240 325 333 126 195 439 256  99 211 117  85 171 224 139 146 141 120
 194 226 101 234 435 357 142 311 127 379 217 388 213 192 432 437 418 250
 287 206 129 183 180 176 185 249 365 332 362 390 204 201 247 257 263 300
 360 408 231 241 284 215 199 268 252 220 279 167 160 190 236 441 465 295
 188 157 353 364 303 153 130 155 169 158 420 452 132 417 476 369 355 401
 348 403 404 367 134 137 111 351 450 349 438 445 121 387 164 406 114 103
  96 461 372 358 413 128 149 105 378 118 102 454 464 473 410 397 383 381
 487 493 109 537 392 485 535 395 385 608 520 515 539 374 489 498]
--------------
```

```
#value counts of each column
for cols in (df.columns):
```

```
    print ("Value counts of", cols, "\n", df[cols].value_counts())
    print ("--------------")
```

```
35     460
17     460
36     414
37     344
38     335
16     291
39     264
15     251
40     246
42     206
14     170
43     142
41     114
13      64
46      61
45      59
47      52
44      47
50      44
55      29
48      29
12      27
53      23
63      20
49      19
58      18
51      17
56      15
60      14
59      14
61      14
52      12
54      11
69      10
11       8
78       7
74       7
67       6
64       6
66       4
76       2
71       2
Name: COMB (mpg), dtype: int64
--------------
Value counts of EMISSIONS
 221    364
225    352
228    348
232    339
230    338
      ...
372      1
134      1
351      1
450      1
417      1
Name: EMISSIONS, Length: 358, dtype: int64
--------------
```

```
#from below feature selection step, model and transmission columns are having low values/prior
df1=df.drop(['MODEL','TRANSMISSION'], axis=1)
df1
```

```
#I applied get_dummies t0 columns 'Company', and 'Vehicle class for better visibility
dummy=pd.get_dummies(df1[['Company','VEHICLE CLASS']], drop_first=True)
dummy
```

	Company_ALFA ROMEO	Company_ASTON MARTIN	Company_AUDI	Company_Acura	Company_Alfa Romeo	Company_Aston Martin	Company_A
0	0	0	0	0	0	0	
1	0	0	0	0	0	0	
2	0	0	0	0	0	0	
3	0	0	0	0	0	0	
4	0	0	0	0	0	0	
...	...	...	...	...	...	...	
22551	0	0	0	0	0	0	
22552	0	0	0	0	0	0	
22553	0	0	0	0	0	0	
22554	0	0	0	0	0	0	
22555	0	0	0	0	0	0	

22556 rows × 117 columns

```
#combine both dummy and main df
dfe=pd.concat([df1,dummy],axis=1)
dfe
```

	YEAR	Company	VEHICLE CLASS	ENGINE SIZE	CYLINDERS	FUEL	FUEL CONSUMPTION	HWY (L/100 km)	COMB (L/100 km)	COMB (mpg)	...	C
0	2000	ACURA	COMPACT	1.6	4	X	9.2	6.7	8.1	35	...	
1	2000	ACURA	COMPACT	1.6	4	X	8.5	6.5	7.6	37	...	
2	2000	ACURA	MID-SIZE	3.2	6	Z	12.2	7.4	10.0	28	...	
3	2000	ACURA	MID-SIZE	3.5	6	Z	13.4	9.2	11.5	25	...	
4	2000	ACURA	SUBCOMPACT	1.8	4	X	10.0	7.0	8.6	33	...	
...	...	...	...	...	...	...	...	...	...	...	...	
22551	2022	Volvo	SUV: Small	2.0	4	Z	10.7	7.7	9.4	30	...	
22552	2022	Volvo	SUV: Small	2.0	4	Z	10.5	8.1	9.4	30	...	
22553	2022	Volvo	SUV: Small	2.0	4	Z	11.0	8.7	9.9	29	...	
22554	2022	Volvo	SUV: Standard	2.0	4	Z	11.5	8.4	10.1	28	...	
22555	2022	Volvo	SUV: Standard	2.0	4	Z	12.4	8.9	10.8	26	...	

22556 rows × 128 columns

```
#after combining, drop the unwanted columns
dfe=dfe.drop(['Company','VEHICLE CLASS'], axis=1)
dfe
```

	YEAR	ENGINE SIZE	CYLINDERS	FUEL	FUEL CONSUMPTION	HWY (L/100 km)	COMB (L/100 km)	COMB (mpg)	EMISSIONS	Company_ALFA ROMEO	...
0	2000	1.6	4	X	9.2	6.7	8.1	35	186	0	...
1	2000	1.6	4	X	8.5	6.5	7.6	37	175	0	...
2	2000	3.2	6	Z	12.2	7.4	10.0	28	230	0	...
3	2000	3.5	6	Z	13.4	9.2	11.5	25	264	0	...
4	2000	1.8	4	X	10.0	7.0	8.6	33	198	0	...
...	...	...	...	...	...	...	...	...	...	...	...
22551	2022	2.0	4	Z	10.7	7.7	9.4	30	219	0	...
22552	2022	2.0	4	Z	10.5	8.1	9.4	30	219	0	...

```
dfe.columns
```

```
Index(['YEAR', 'ENGINE SIZE', 'CYLINDERS', 'FUEL', 'FUEL CONSUMPTION',
       'HWY (L/100 km)', 'COMB (L/100 km)', 'COMB (mpg)', 'EMISSIONS',
       'Company_ALFA ROMEO',
       ...
       'VEHICLE CLASS_SUV: Standard', 'VEHICLE CLASS_Special purpose vehicle',
       'VEHICLE CLASS_Station wagon: Mid-size',
       'VEHICLE CLASS_Station wagon: Small', 'VEHICLE CLASS_Subcompact',
       'VEHICLE CLASS_TWO-SEATER', 'VEHICLE CLASS_Two-seater',
       'VEHICLE CLASS_VAN - CARGO', 'VEHICLE CLASS_VAN - PASSENGER',
       'VEHICLE CLASS_Van: Passenger'],
      dtype='object', length=126)
```

```
#label encode the data's to convert it to numeric
nw_cols=dfe.columns
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
for nw_cols in dfe.columns:
  dfe[nw_cols]=le.fit_transform(dfe[nw_cols])
```

```
dfe.dtypes
```

```
YEAR                             int64
ENGINE SIZE                      int64
CYLINDERS                        int64
FUEL                             int64
FUEL CONSUMPTION                 int64
                                 ...
VEHICLE CLASS_TWO-SEATER         int64
VEHICLE CLASS_Two-seater         int64
VEHICLE CLASS_VAN - CARGO        int64
VEHICLE CLASS_VAN - PASSENGER    int64
VEHICLE CLASS_Van: Passenger     int64
Length: 126, dtype: object
```

```
dfe
```

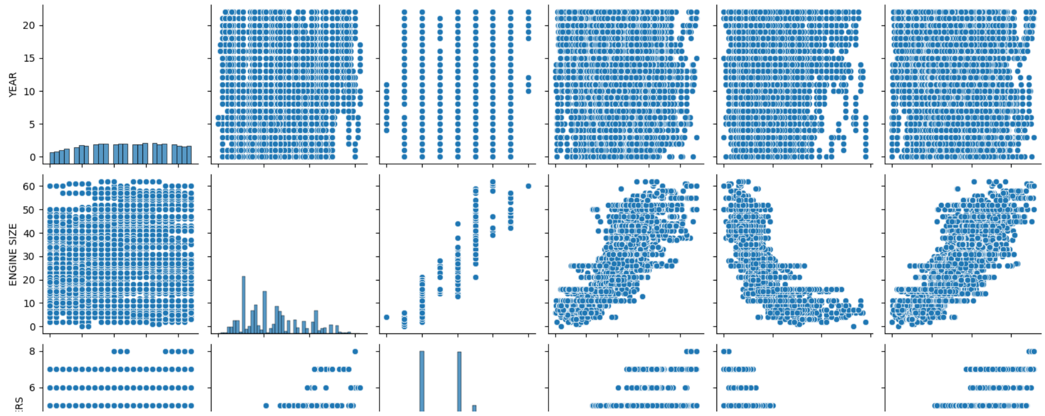
HWY COMB COMB Company_ALFA VEHICLE VEHICLE CLASS_Special VEHICLE CLASS_Station

```
dfe['FUEL CONSUMPTION'].value_counts()
```

```
85     399
82     360
71     346
80     341
81     339
      ...
218      1
212      1
15       1
1        1
214      1
Name: FUEL CONSUMPTION, Length: 228, dtype: int64
```

Visualization

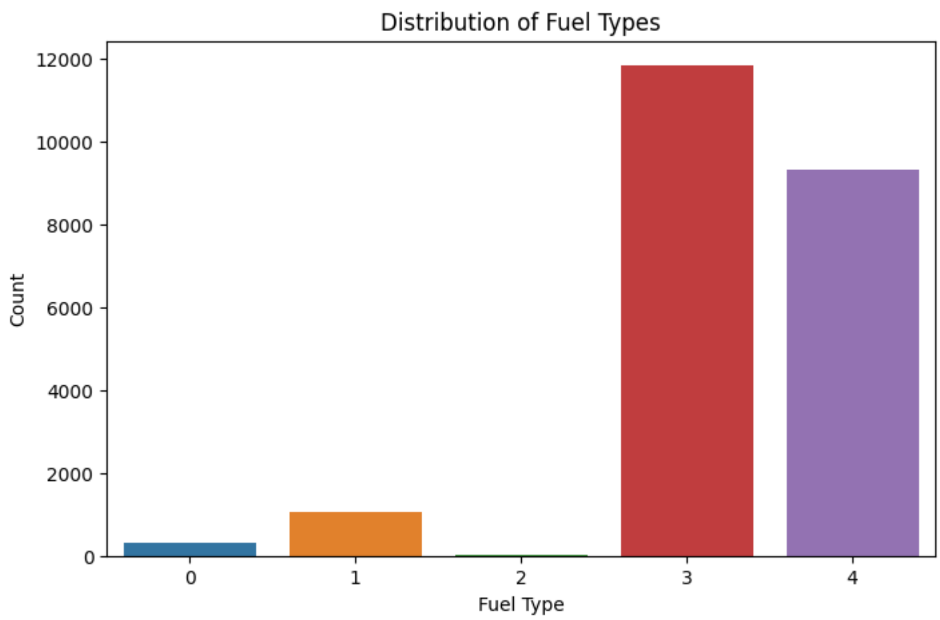
```
#pairplot of columns=('YEAR', 'ENGINE SIZE', 'CYLINDERS', 'FUEL CONSUMPTION', 'COMB (mpg)', 'EMISSIONS')
sns.pairplot(dfe[['YEAR', 'ENGINE SIZE', 'CYLINDERS', 'FUEL CONSUMPTION', 'COMB (mpg)', 'EMISSIONS']])
plt.show()
```



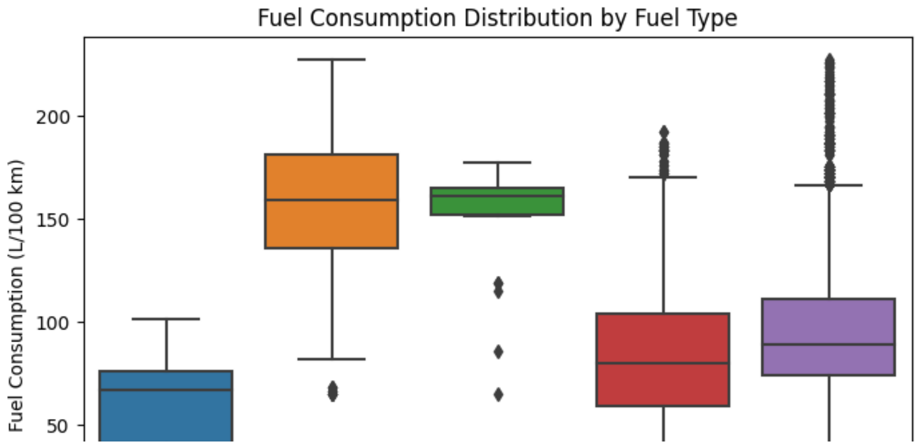

```
dfe['FUEL'].value_counts()
```

```
3    11822
4     9316
1     1071
0      314
2       33
Name: FUEL, dtype: int64
```

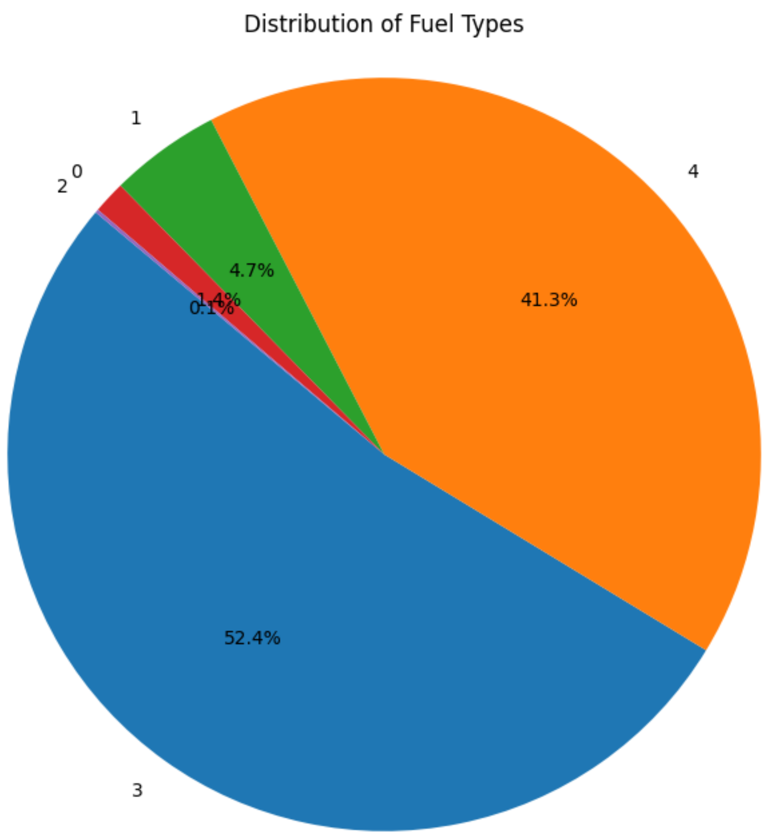
```
#Distribution of fuel type
plt.figure(figsize=(8, 5))
sns.countplot(x='FUEL', data=dfe)
plt.title("Distribution of Fuel Types")
plt.xlabel("Fuel Type")
plt.ylabel("Count")
plt.show()
#where,
       #0-D
       #1-E
       #2-N
       #3-X
       #4-Z
```



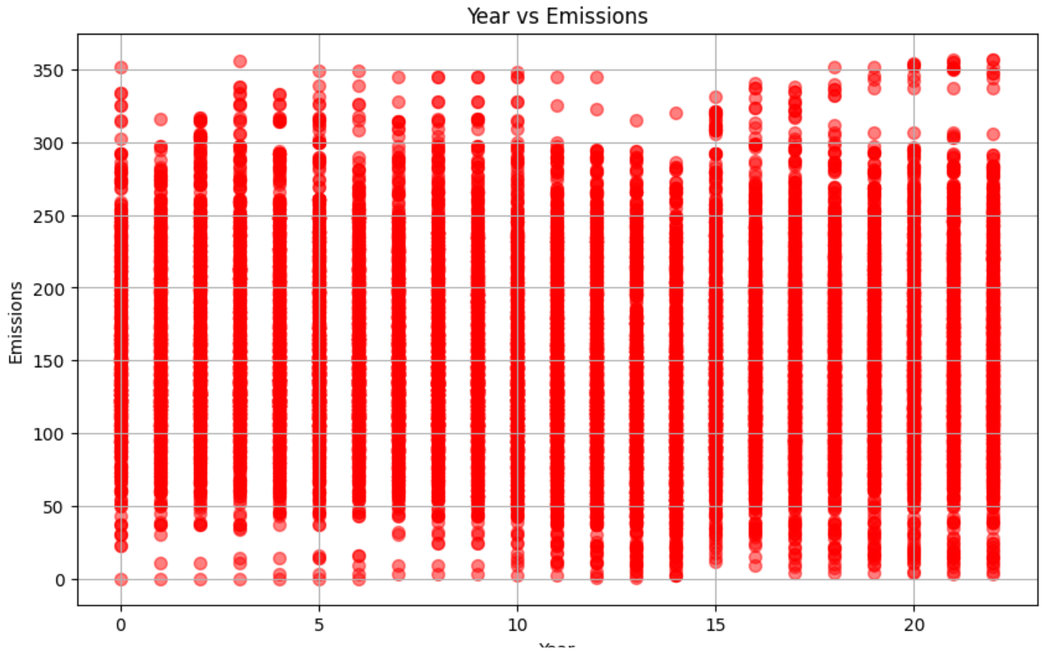

```
#fuel consumption by fuel type
plt.figure(figsize=(8, 5))
sns.boxplot(x='FUEL', y='FUEL CONSUMPTION', data=dfe)
plt.title("Fuel Consumption Distribution by Fuel Type")
plt.xlabel("Fuel Type")
plt.ylabel("Fuel Consumption (L/100 km)")
plt.show()
```



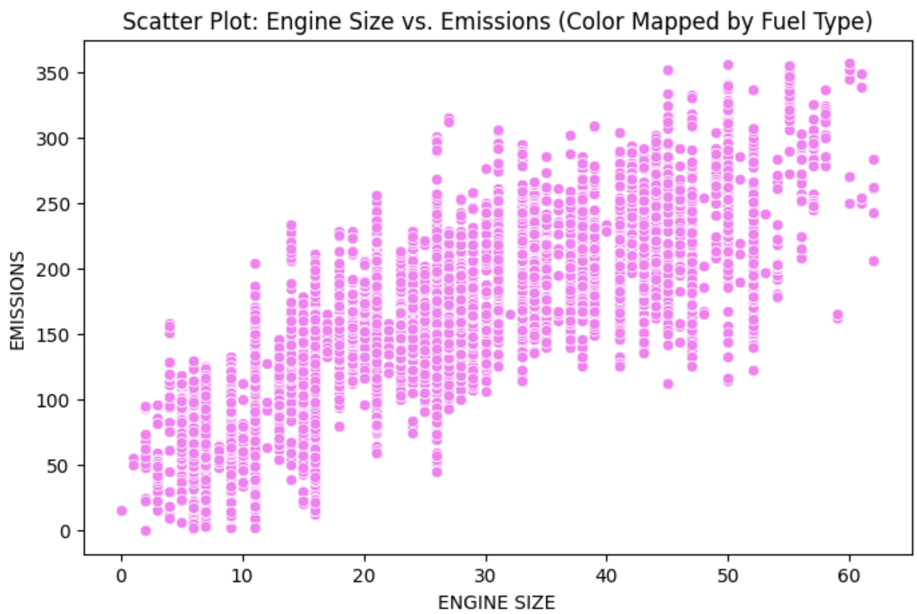

```
#distribution of fuel types
fuel_counts=dfe['FUEL'].value_counts()
plt.figure(figsize=(8, 8))
plt.pie(fuel_counts.values, labels=fuel_counts.index, autopct='%1.1f%%', startangle=140)
plt.title("Distribution of Fuel Types")
plt.axis('equal')
plt.show()
#where,
       #0-D
       #1-E
       #2-N
       #3-X
       #4-Z
```




```
#emission by year
plt.figure(figsize=(10, 6))
plt.scatter(dfe['YEAR'], dfe['EMISSIONS'], s=50, alpha=0.5, color='red')
plt.xlabel('Year')
plt.ylabel('Emissions')
plt.title('Year vs Emissions')
plt.grid(True)
plt.show()
```

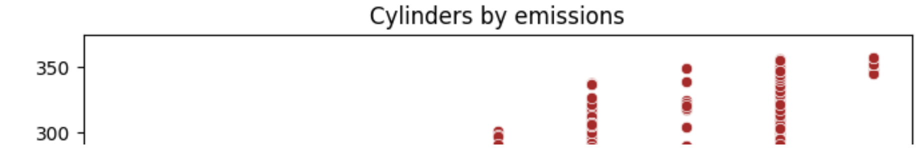



```
plt.figure(figsize=(8, 5))
sns.scatterplot(x='ENGINE SIZE', y='EMISSIONS', data=dfe, color='violet')
plt.title("Scatter Plot: Engine Size vs. Emissions (Color Mapped by Fuel Type)")
plt.show()
```



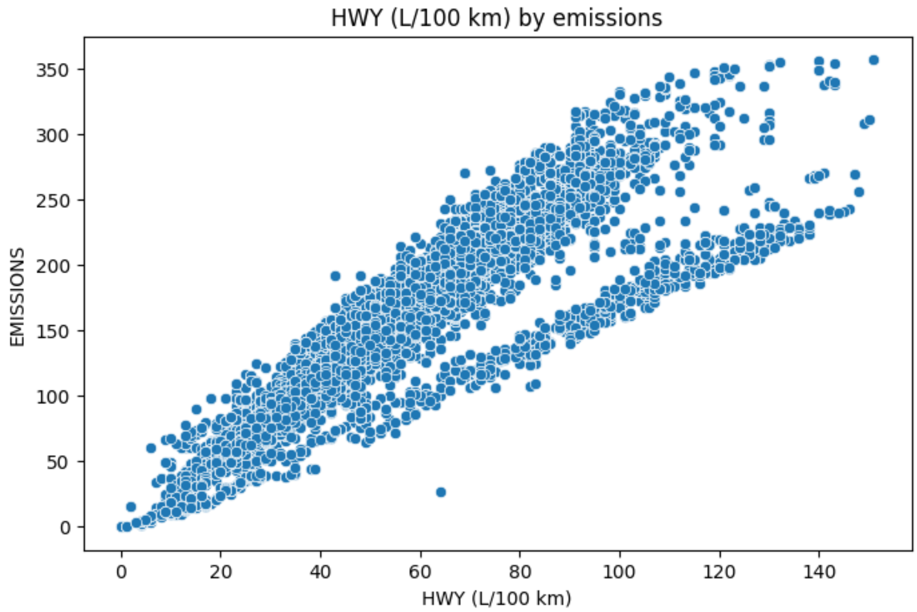

```
#cylinders by emissions
plt.figure(figsize=(8, 5))
sns.scatterplot(x='CYLINDERS', y='EMISSIONS', data=dfe, color='brown')
plt.title('Cylinders by emissions')
```

```
Text(0.5, 1.0, 'Cylinders by emissions')
```



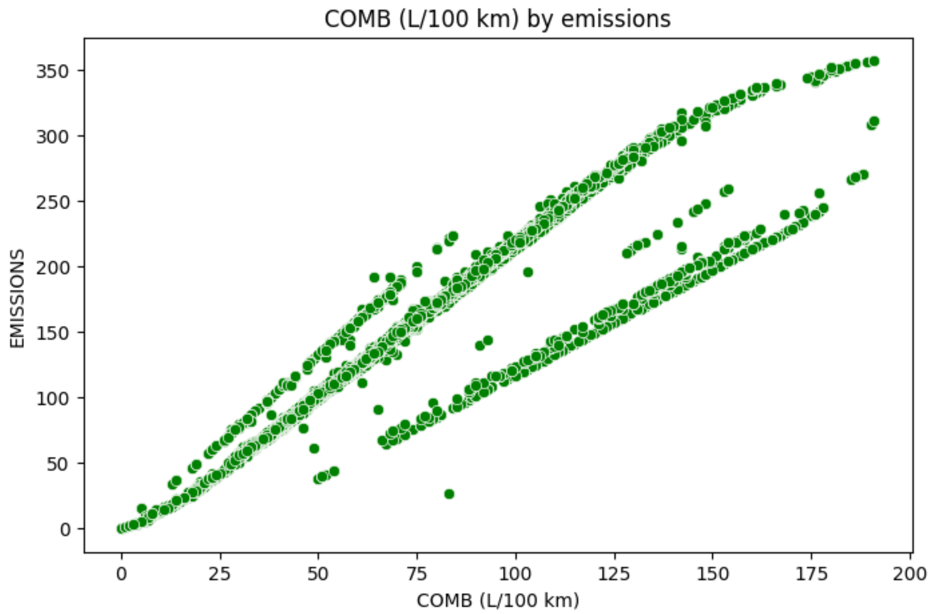

```
#HWY (L/100 km) by emissions
plt.figure(figsize=(8, 5))
sns.scatterplot(x='HWY (L/100 km)', y='EMISSIONS', data=dfe)
plt.title('HWY (L/100 km) by emissions')
```

```
Text(0.5, 1.0, 'HWY (L/100 km) by emissions')
```

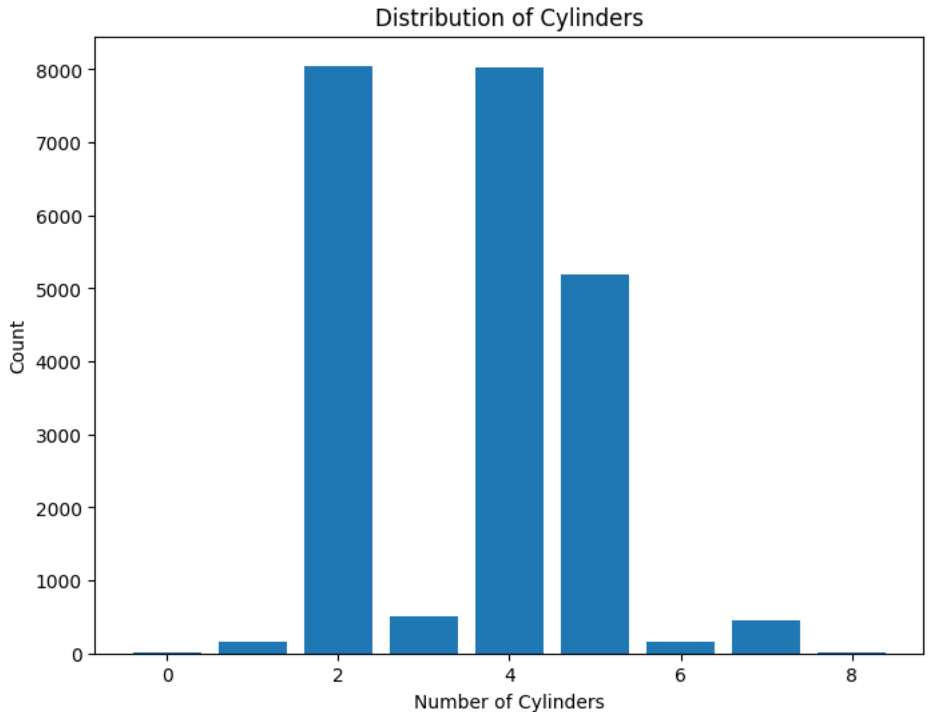



```
#COMB (L/100 km) by emissions
plt.figure(figsize=(8, 5))
sns.scatterplot(x='COMB (L/100 km)', y='EMISSIONS', data=dfe, color='green')
plt.title('COMB (L/100 km) by emissions')
```

```
Text(0.5, 1.0, 'COMB (L/100 km) by emissions')
```

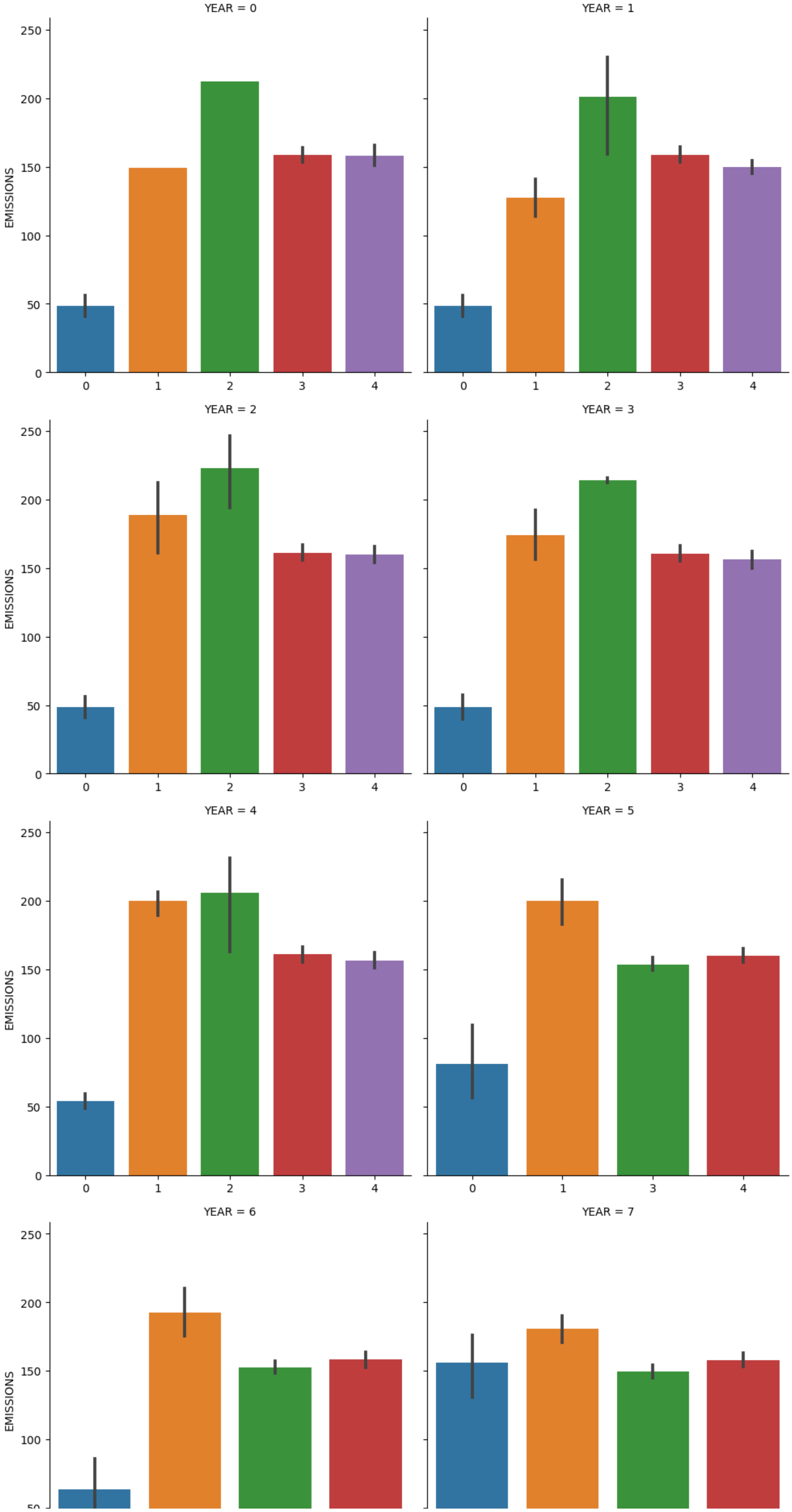


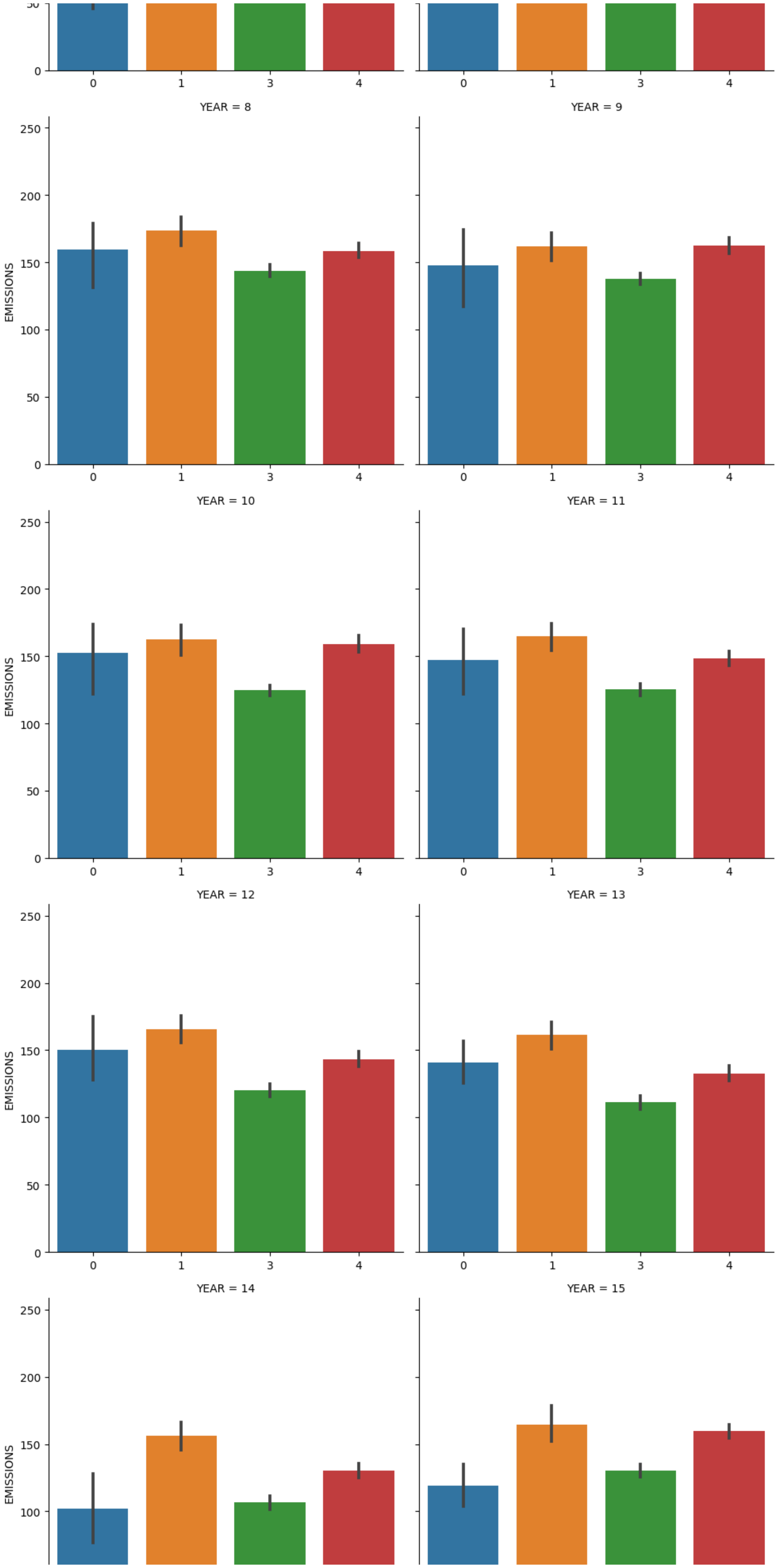

```
#Different cylinders
cylinder_counts=dfe['CYLINDERS'].value_counts()
plt.figure(figsize=(8, 6))
plt.bar(cylinder_counts.index, cylinder_counts.values)
plt.title("Distribution of Cylinders")
plt.xlabel("Number of Cylinders")
plt.ylabel("Count")
plt.show()
```

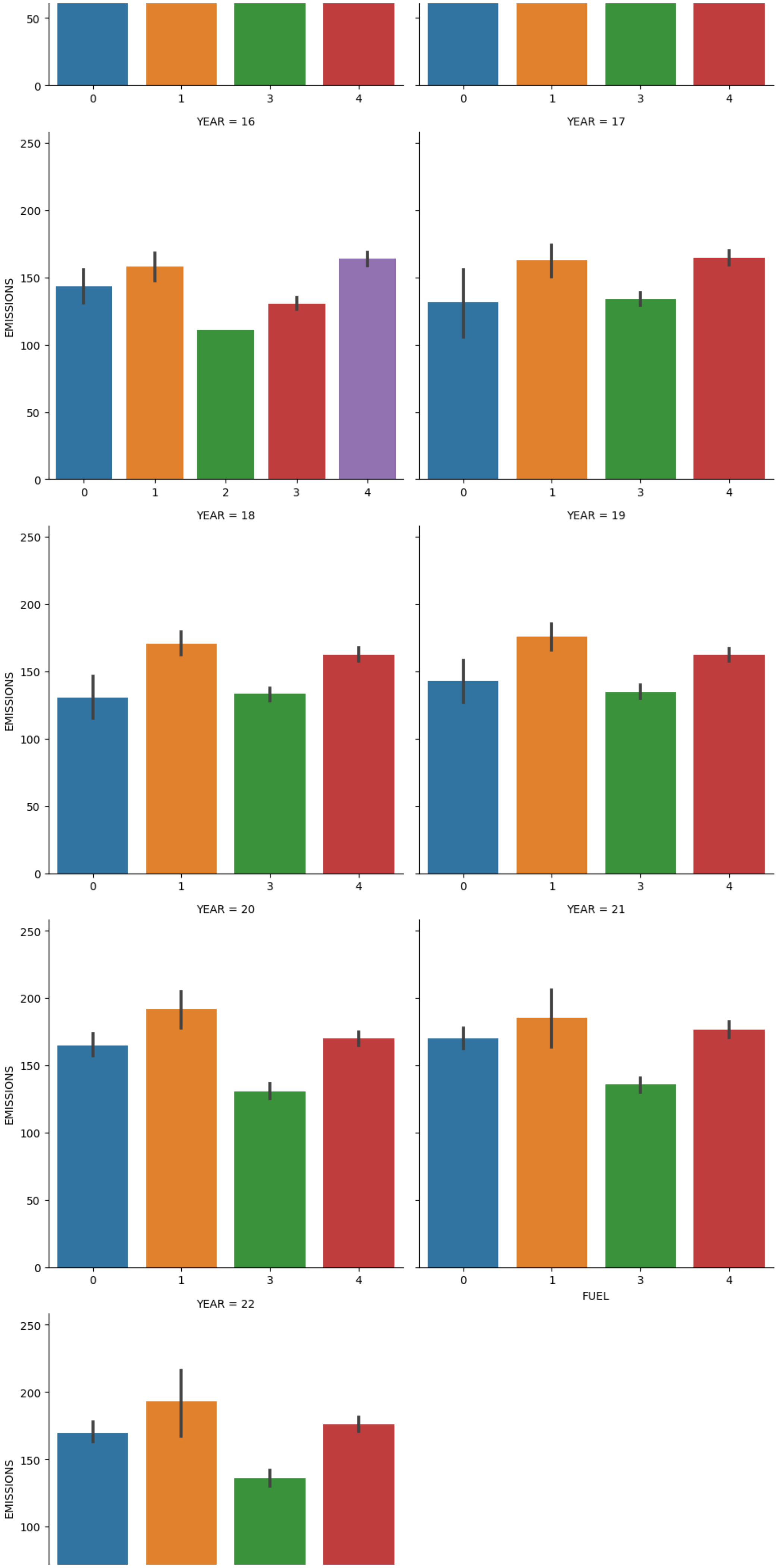


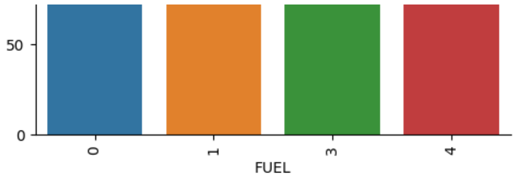

```
#Emission by each year
sns.catplot(data=dfe, x='FUEL', y='EMISSIONS', kind='bar', col='YEAR', col_wrap=2, sharex=False)
plt.xticks(rotation='vertical')
```

```
/usr/local/lib/python3.10/dist-packages/seaborn/categorical.py:3201: UserWarning: Setting `sharex=False`
  warnings.warn(msg.format("sharex", "x"), UserWarning)
(array([0, 1, 2, 3]),
 [Text(0, 0, '0'), Text(1, 0, '1'), Text(2, 0, '3'), Text(3, 0, '4')])
```



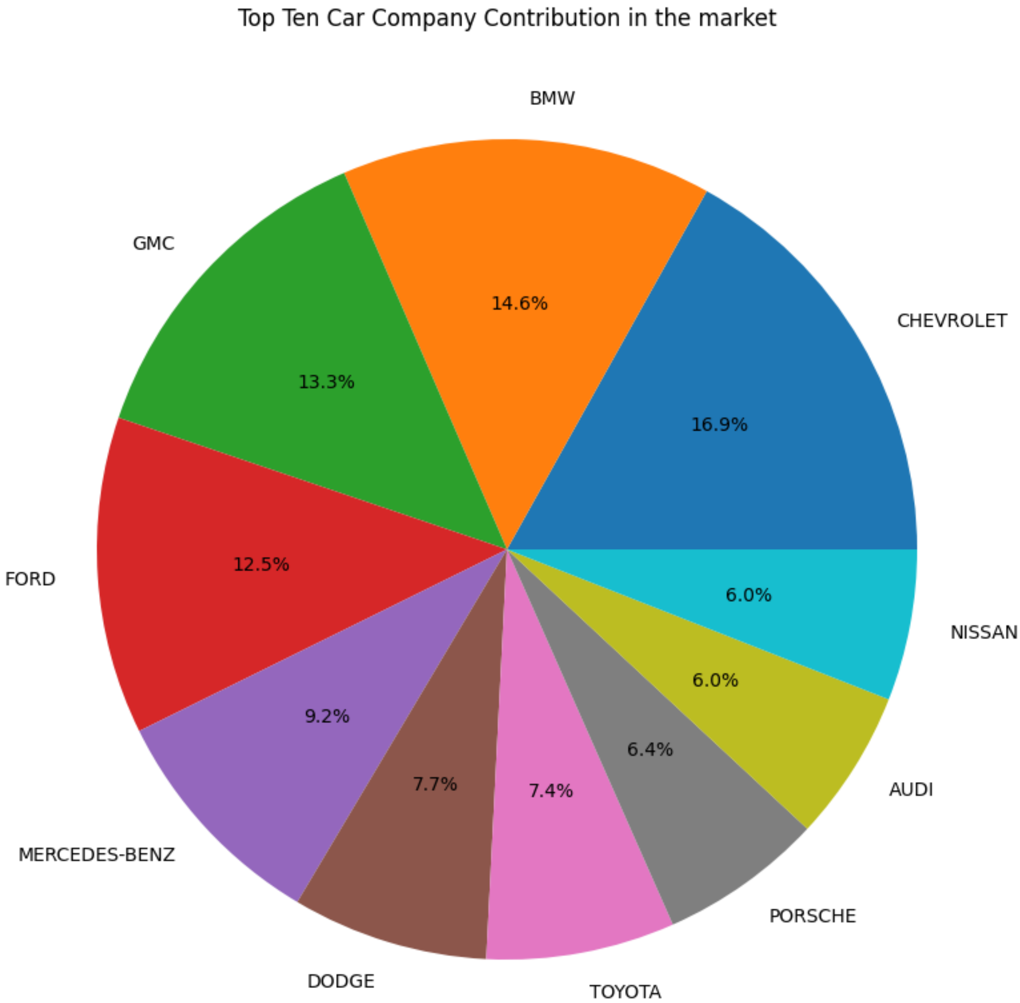

YEAR = 8
YEAR = 9
YEAR = 10
YEAR = 11
YEAR = 12
YEAR = 13
YEAR = 14
YEAR = 15
EMISSIONS
0
50
100
150
200
250
0
1
3
4


YEAR = 16
YEAR = 17
YEAR = 18
YEAR = 19
YEAR = 20
YEAR = 21
YEAR = 22
EMISSIONS
FUEL
0
50
100
150
200
250
0 1 2 3 4
0 1 3 4



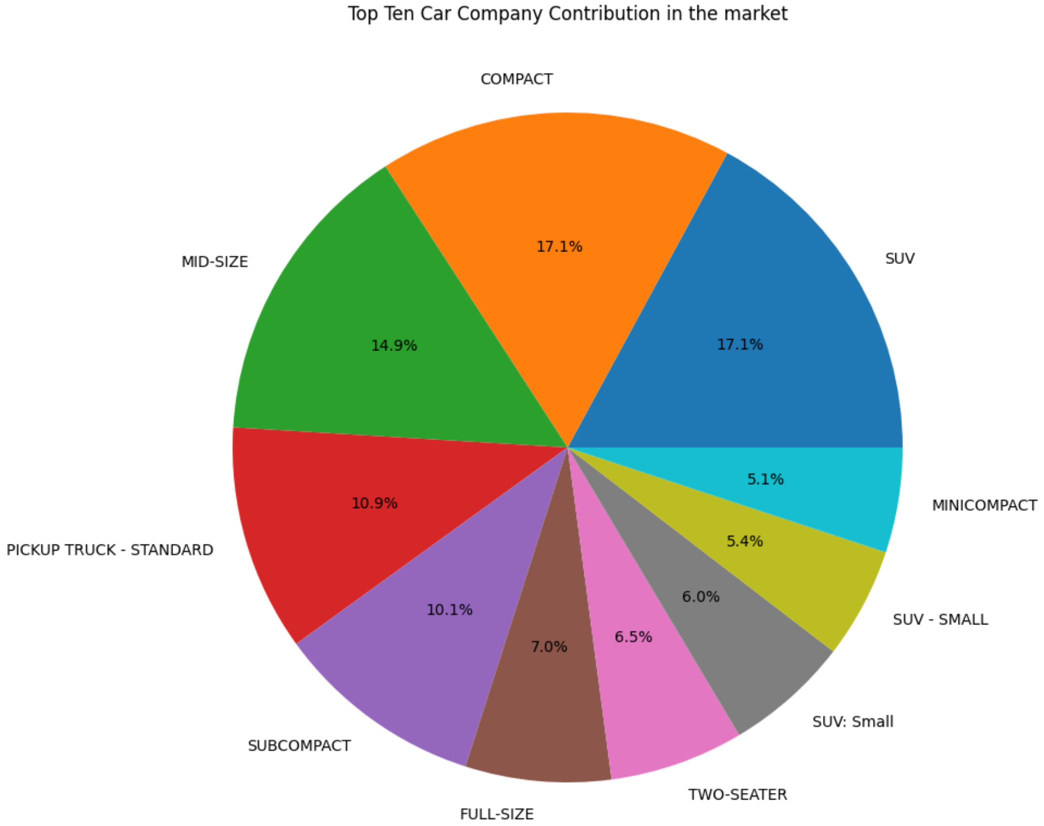

```
#top 10 car company contribution in the market
plt.figure(figsize=(10,10))
Top_10_cars=df1['Company'].value_counts().head(10)
plt.pie(Top_10_cars, labels=Top_10_cars.index, autopct="%0.1f%%")
plt.title('Top Ten Car Company Contribution in the market')
```

```
Text(0.5, 1.0, 'Top Ten Car Company Contribution in the market')
```



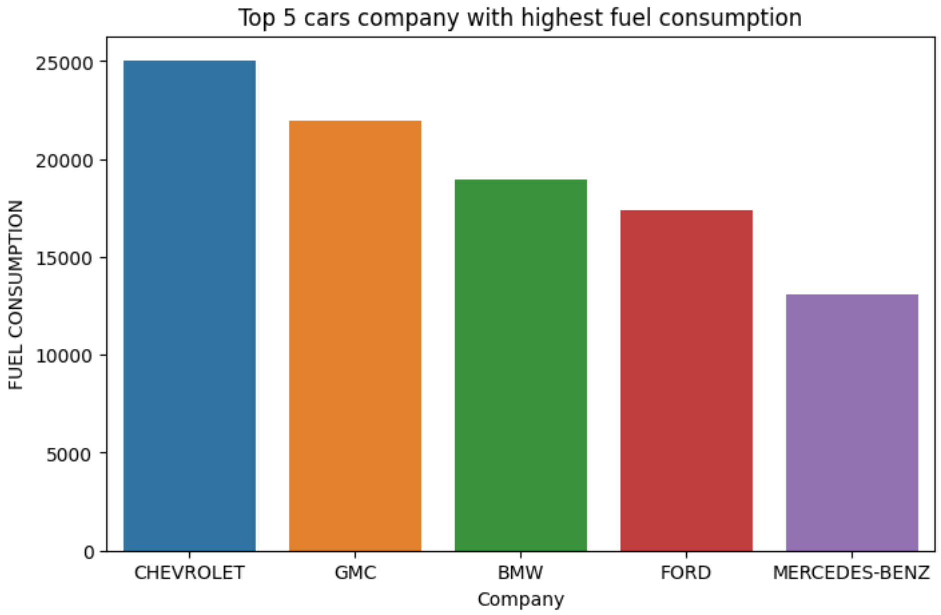

```
#top 10 vehicle class in the market
plt.figure(figsize=(10,10))
Top_10_vehicleclass=df1['VEHICLE CLASS'].value_counts().head(10)
plt.pie(Top_10_vehicleclass, labels=Top_10_vehicleclass.index, autopct="%0.1f%%")
plt.title('Top Ten Car Company Contribution in the market')
```

```
Text(0.5, 1.0, 'Top Ten Car Company Contribution in the market')
```




```
#top 5 cars company with highest fuel consumption
company_consumption=df1.groupby('Company')['FUEL CONSUMPTION'].sum().sort_values(ascending = False).reset_index()
plt.figure(figsize=(8,5))
sns.barplot(x="Company", y="FUEL CONSUMPTION", data=company_consumption[:5])
plt.title('Top 5 cars company with highest fuel consumption')
```

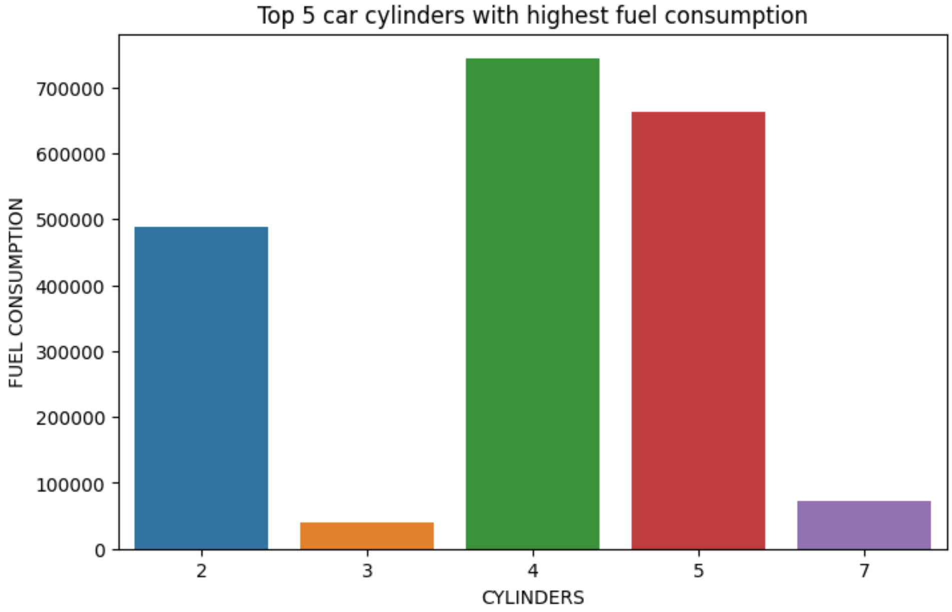
```
Text(0.5, 1.0, 'Top 5 cars company with highest fuel consumption')
```




```
#top 5 car cylinders with highest fuel consumption
carcyl_consumption=dfe.groupby('CYLINDERS')['FUEL CONSUMPTION'].sum().sort_values(ascending = False).reset_index()
plt.figure(figsize=(8,5))
```

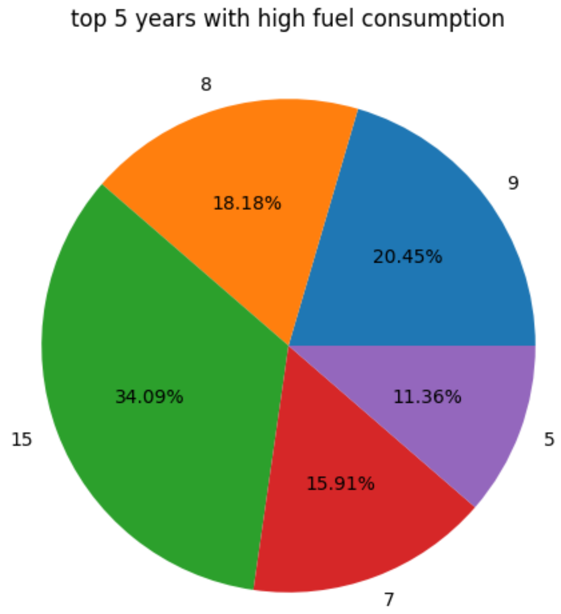
```
sns.barplot(x="CYLINDERS", y="FUEL CONSUMPTION", data=carcyl_consumption[:5])
plt.title('Top 5 car cylinders with highest fuel consumption')
```

```
Text(0.5, 1.0, 'Top 5 car cylinders with highest fuel consumption')
```




```
#top 5 years with high fuel consumption
plt.figure(figsize=(10, 6))
top_fuel=dfe.groupby('YEAR')['FUEL CONSUMPTION'].sum().sort_values(ascending = False).index
plt.pie(top_fuel[:5],labels=top_fuel[:5], autopct="%1.2f%%")
plt.title('top 5 years with high fuel consumption')
```

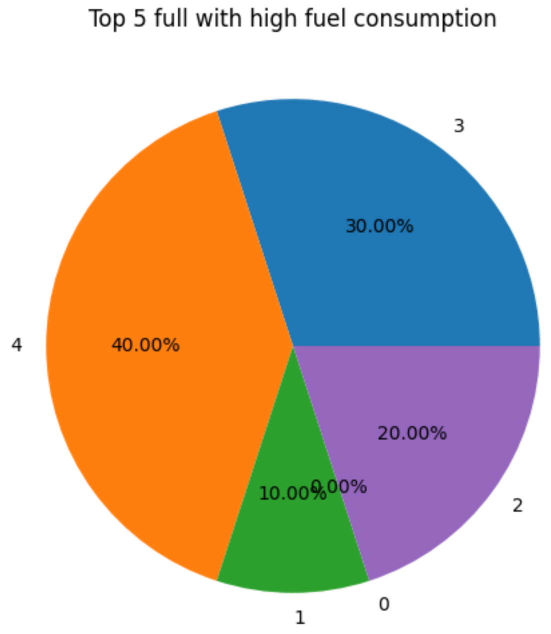
```
Text(0.5, 1.0, 'top 5 years with high fuel consumption')
```



Double-click (or enter) to edit

```
#Top 5 fuel with high fuel consumption
top_5_fuels=dfe.groupby('FUEL')['FUEL CONSUMPTION'].sum().sort_values(ascending=False).index
plt.figure(figsize=(10, 6))
plt.pie(top_5_fuels[:5], labels=top_5_fuels[:5], autopct="%1.2f%%")
plt.title("Top 5 full with high fuel consumption")
#where,
       #0-D
       #1-E
       #2-N
       #3-X
       #4-Z
```

```
Text(0.5, 1.0, 'Top 5 full with high fuel consumption')
```




```
x=dfe.drop(['FUEL CONSUMPTION'], axis=1)
x
```

	YEAR	ENGINE SIZE	CYLINDERS	FUEL	HWY (L/100 km)	COMB (L/100 km)	COMB (mpg)	EMISSIONS	Company_ALFA ROMEO	Company_ASTON MARTIN	...
0	0	7	2	3	31	44	24	84	0	0	..
1	0	7	2	3	29	39	26	73	0	0	..
2	0	23	4	4	38	63	17	128	0	0	..
3	0	26	4	4	56	78	14	162	0	0	..
4	0	9	2	3	34	49	22	96	0	0	..
...	...	...	...	...	...	...	...	...	...	...	..
22551	22	11	2	4	41	57	19	117	0	0	..
22552	22	11	2	4	45	57	19	117	0	0	..
22553	22	11	2	4	51	62	18	130	0	0	..
22554	22	11	2	4	48	64	17	134	0	0	..
22555	22	11	2	4	53	71	15	150	0	0	..

22556 rows × 125 columns

```
y=dfe['FUEL CONSUMPTION']
y
```

```
0        55
1        48
2        85
3        97
4        63
         ..
22551    70
22552    68
22553    73
22554    78
22555    87
Name: FUEL CONSUMPTION, Length: 22556, dtype: int64
```

```
x.dtypes
```

```
YEAR                              int64
ENGINE SIZE                       int64
CYLINDERS                         int64
FUEL                              int64
HWY (L/100 km)                    int64
                                  ...
VEHICLE CLASS_TWO-SEATER          int64
VEHICLE CLASS_Two-seater          int64
VEHICLE CLASS_VAN - CARGO         int64
VEHICLE CLASS_VAN - PASSENGER     int64
VEHICLE CLASS_Van: Passenger      int64
Length: 125, dtype: object
```

```
y.dtypes
```

```
dtype('int64')
```

```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
test=SelectKBest(score_func=chi2)
fi=test.fit(x, y)
fi.scores_
```

```
array([4.62201490e+03, 1.15738655e+05, 7.91521885e+03, 8.41761273e+02,
       1.96701058e+05, 2.55445570e+05, 7.23715384e+04, 4.66631266e+05,
       2.91324966e+02, 1.37664405e+03, 5.44122214e+02, 2.63084816e+02,
       3.82369530e+02, 6.99498270e+02, 4.04658354e+02, 3.02128265e+03,
       8.69811221e+02, 8.35107407e+03, 2.72707918e+02, 2.68676072e+03,
       1.59592159e+04, 2.21538241e+02, 5.12826005e+02, 1.43557778e+03,
       5.25454161e+02, 1.95214048e+02, 5.43075357e+02, 7.53222227e+02,
       2.32527593e+02, 1.57400714e+03, 1.74574252e+03, 6.05277770e+03,
       8.66329831e+02, 5.58495736e+02, 6.11160184e+02, 1.58250981e+02,
       2.33227404e+03, 1.94050226e+02, 1.60774817e+03, 9.60718662e+02,
       6.21125111e+02, 1.45831096e+03, 1.34523296e+03, 3.53164451e+02,
       2.13023297e+02, 7.13352398e+02, 8.84180470e+02, 5.77172506e+02,
       4.60180945e+02, 6.31118154e+02, 5.47247991e+02, 9.90635449e+02,
       3.77104109e+03, 1.28751934e+03, 6.46061149e+02, 4.61876983e+02,
       8.06228215e+03, 2.64193757e+02, 8.54793124e+02, 3.35706534e+02,
       1.72886752e+03, 5.30722821e+02, 7.38870523e+02, 2.05126656e+03,
       3.87748853e+02, 7.06955129e+02, 7.41136349e+02, 3.53253501e+02,
       9.03802478e+02, 5.08956175e+02, 4.37301699e+02, 1.45723097e+02,
       1.54638889e+02, 3.31523575e+02, 6.59131407e+02, 6.45260734e+02,
       4.99450930e+02, 3.20303844e+03, 5.72308579e+02, 4.48421615e+03,
       3.60131415e+02, 2.84043299e+02, 8.32115028e+02, 6.87484991e+03,
       8.55276190e+02, 1.04262765e+03, 8.09694320e+02, 4.35947978e+02,
       1.09856609e+03, 1.62228633e+03, 1.69265342e+03, 6.62306138e+02,
       3.00895520e+02, 2.39286651e+02, 7.01619784e+02, 8.31157593e+02,
       1.29669888e+03, 8.27889883e+02, 6.39277147e+02, 1.14182518e+03,
       1.08325196e+03, 3.89571341e+02, 4.65029894e+02, 6.50734762e+02,
       2.74660421e+03, 4.66278971e+02, 1.00647022e+03, 2.07036823e+02,
       4.77062215e+02, 9.65903539e+02, 6.61323995e+02, 1.17994008e+03,
       5.24390401e+02, 1.23074262e+03, 7.19979808e+02, 1.19218136e+03,
       4.51943264e+02, 7.03835787e+02, 1.10554845e+03, 3.20852436e+02,
       1.33219484e+03, 2.19447994e+03, 3.28365818e+03, 4.34332498e+03,
       8.63804240e+02])
```

```
col=x.columns
score=pd.DataFrame({'features':col,'score_chi2':fi.scores_})
score
```

	features	score_chi2
0	YEAR	4622.014898
1	ENGINE SIZE	115738.655416
2	CYLINDERS	7915.218846
3	FUEL	841.761273
4	HWY (L/100 km)	196701.058097
...	...	...
120	VEHICLE CLASS_TWO-SEATER	1332.194837
121	VEHICLE CLASS_Two-seater	2194.479936
122	VEHICLE CLASS_VAN - CARGO	3283.658181
123	VEHICLE CLASS_VAN - PASSENGER	4343.324980
124	VEHICLE CLASS_Van: Passenger	863.804240

125 rows × 2 columns

```
score.sort_values(by='score_chi2',ascending=False)
```

	features	score_chi2
7	EMISSIONS	466631.266177
5	COMB (L/100 km)	255445.570099
4	HWY (L/100 km)	196701.058097
1	ENGINE SIZE	115738.655416
6	COMB (mpg)	72371.538425
...	...	...
25	Company_Cadillac	195.214048
37	Company_Genesis	194.050226
35	Company_GENESIS	158.250981
72	Company_PLYMOUTH	154.638889
71	Company_OLDSMOBILE	145.723097

125 rows × 2 columns

```
#train test split
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,random_state=42)
```

```
x_train.shape
```

```
(18044, 125)
```

```
x_test.shape
```

```
(4512, 125)
```

```
#normalize
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
x_train=sc.fit_transform(x_train)
x_test=sc.fit_transform(x_test)
```

```
import pandas as pd
from sklearn.linear_model import LinearRegression, Lasso
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import r2_score, mean_squared_error
```

```
#Linear Regression
linear_model=LinearRegression()
linear_model.fit(x_train, y_train)
linear_y_pred=linear_model.predict(x_test)
linear_mse=mean_squared_error(y_test, linear_y_pred)
linear_r2=r2_score(y_test, linear_y_pred)
#Lasso Regression
lasso_model=Lasso(alpha=0.1)
lasso_model.fit(x_train, y_train)
lasso_y_pred=lasso_model.predict(x_test)
lasso_mse=mean_squared_error(y_test, lasso_y_pred)
lasso_r2=r2_score(y_test, lasso_y_pred)
#Random Forest Regressor
rf_model=RandomForestRegressor(n_estimators=100, random_state=42)
rf_model.fit(x_train, y_train)
rf_y_pred=rf_model.predict(x_test)
rf_mse=mean_squared_error(y_test, rf_y_pred)
rf_r2=r2_score(y_test, rf_y_pred)
#Decision Tree Regressor
dt_model=DecisionTreeRegressor(random_state=42)
dt_model.fit(x_train, y_train)
dt_y_pred=dt_model.predict(x_test)
dt_mse=mean_squared_error(y_test, dt_y_pred)
dt_r2=r2_score(y_test, dt_y_pred)
```

```
#compare the MSE values to see which regressor has the lowest MSE
min_mse=min(linear_mse, lasso_mse, rf_mse, dt_mse)
```

```
best_model=None
if min_mse==linear_mse:
   best_model="Logistic Regression"
elif min_mse==lasso_mse:
   best_model="Lasso Regression"
elif min_mse==rf_mse:
   best_model="Random Forest Regression"
else:
   best_model="Random Forest Regressor"
print("\nThe best performing model is:", best_model, "with mse", linear_mse)
```

```
The best performing model is: Logistic Regression with mse 1.3479226877777626
```

```
#print the r2 score results
print("Linear Regression")
print("LR r2 score:", linear_r2)
print("\nLasso Regression:")
print("L r2 score:", lasso_r2)
print("\nRandom Forest Regressor:")
print("RF r2 score:", rf_r2)
print("\nDecision Tree Regressor:")
print("DT r2 score:", dt_r2)
```

```
Linear Regression
LR r2 score: 0.9988681246786404

Lasso Regression:
L r2 score: 0.9980411787536116

Random Forest Regressor:
RF r2 score: 0.9980340943871755

Decision Tree Regressor:
DT r2 score: 0.9973224217428676
```

```
#compare the R2 scores to see which regressor has the highest R2 score.
max_r2=max(linear_r2, lasso_r2, rf_r2, dt_r2)
best_model=None
if max_r2==linear_r2:
   best_model="Linear Regression"
elif max_r2==lasso_r2:
   best_model="Lasso Regression"
else:
   best_model="Random Forest Regressor"
print("\nThe best performing model is:", best_model, "with r2 score", rf_r2)
```

```
The best performing model is: Linear Regression with r2 score 0.9980340943871755
```

```
#finding the difference between y test and linear y pred
results=pd.DataFrame()
results['Actual']=y_test
results['Predicted']=linear_y_pred
results['Difference']=y_test-linear_y_pred
results.sort_index()
```

	Actual	Predicted	Difference
3	97	96.292567	0.707433
17	97	95.616279	1.383721
31	93	93.171219	-0.171219
34	78	77.196463	0.803537
35	86	85.633820	0.366180
...	...	...	...
22526	22	20.704717	1.295283
22532	61	59.899517	1.100483
22541	58	55.957742	2.042258
22551	70	69.927146	0.072854
22554	78	77.054206	0.945794

4512 rows × 3 columns

The best performing model is: Linear Regression