In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
from sklearn.linear_model import LinearRegression, Lasso, Ridge
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error ,r2_score, mean_absolute_error
```

In [2]:
```
# Importing data
```

In [3]:
```
ds = pd.read_csv('CO2 Emissions_India.csv')
```

In [4]:
```
ds.head()
```

Out[4]:

	Make	Model	Vehicle Class	Engine Size(L)	Cylinders	Transmission	Fuel Type	Fuel Consumption City (L/100 km)	Fuel Consumption Hwy (L/100 km)
0	ACURA	ILX	COMPACT	2.0	4	AS5	Z	9.9	6.7
1	ACURA	ILX	COMPACT	2.4	4	M6	Z	11.2	7.7
2	ACURA	ILX HYBRID	COMPACT	1.5	4	AV7	Z	6.0	5.8
3	ACURA	MDX 4WD	SUV - SMALL	3.5	6	AS6	Z	12.7	9.1
4	ACURA	RDX AWD	SUV - SMALL	3.5	6	AS6	Z	12.1	8.7

In [5]:
```
# Renaming columns
```

In [6]:
```
ds.rename(columns={'CO2 Emissions(g/km)':'CO2_emission'}, inplace=True)
```

In [7]:
```
ds.head()
```

Out[7]:

	Make	Model	Vehicle Class	Engine Size(L)	Cylinders	Transmission	Fuel Type	Fuel Consumption City (L/100 km)	Fuel Consumption Hwy (L/100 km)
0	ACURA	ILX	COMPACT	2.0	4	AS5	Z	9.9	6.7
1	ACURA	ILX	COMPACT	2.4	4	M6	Z	11.2	7.7
2	ACURA	ILX HYBRID	COMPACT	1.5	4	AV7	Z	6.0	5.8

	Make	Model	Vehicle Class	Engine Size(L)	Cylinders	Transmission	Fuel Type	Fuel Consumption City (L/100 km)	Fuel Consumption Hwy (L/100 km)
3	ACURA	MDX 4WD	SUV - SMALL	3.5	6	AS6	Z	12.7	9.1
4	ACURA	RDX AWD	SUV - SMALL	3.5	6	AS6	Z	12.1	8.7

Exploratory Data Analysis

In [8]:
```
# Checking for the data types and null values

ds.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_emission                      7385 non-null   int64  
dtypes: float64(4), int64(3), object(5)
memory usage: 692.5+ KB
```

In [9]:
```
# Checking for total null values if any

ds.isnull().sum()
```

Out[9]:
```
Make                                0
Model                               0
Vehicle Class                       0
Engine Size(L)                      0
Cylinders                           0
Transmission                        0
Fuel Type                           0
Fuel Consumption City (L/100 km)    0
Fuel Consumption Hwy (L/100 km)     0
Fuel Consumption Comb (L/100 km)    0
Fuel Consumption Comb (mpg)         0
CO2_emission                        0
dtype: int64
```

In [10]:
```
# Insight of different statistical distribution of features and label

ds.describe().T
```

Out[10]:

	count	mean	std	min	25%	50%	75%	max
Engine Size(L)	7385.0	3.160068	1.354170	0.9	2.0	3.0	3.7	8.4
Cylinders	7385.0	5.615030	1.828307	3.0	4.0	6.0	6.0	16.0
Fuel Consumption City (L/100 km)	7385.0	12.556534	3.500274	4.2	10.1	12.1	14.6	30.6
Fuel Consumption Hwy (L/100 km)	7385.0	9.041706	2.224456	4.0	7.5	8.7	10.2	20.6
Fuel Consumption Comb (L/100 km)	7385.0	10.975071	2.892506	4.1	8.9	10.6	12.6	26.1
Fuel Consumption Comb (mpg)	7385.0	27.481652	7.231879	11.0	22.0	27.0	32.0	69.0
CO2_emission	7385.0	250.584699	58.512679	96.0	208.0	246.0	288.0	522.0

In [11]:

```
# checking for unique variables
print(ds['Make'].unique())
```

```
['ACURA' 'ALFA ROMEO' 'ASTON MARTIN' 'AUDI' 'BENTLEY' 'BMW' 'BUICK'
 'CADILLAC' 'CHEVROLET' 'CHRYSLER' 'DODGE' 'FIAT' 'FORD' 'GMC' 'HONDA'
 'HYUNDAI' 'INFINITI' 'JAGUAR' 'JEEP' 'KIA' 'LAMBORGHINI' 'LAND ROVER'
 'LEXUS' 'LINCOLN' 'MASERATI' 'MAZDA' 'MERCEDES-BENZ' 'MINI' 'MITSUBISHI'
 'NISSAN' 'PORSCHE' 'RAM' 'ROLLS-ROYCE' 'SCION' 'SMART' 'SRT' 'SUBARU'
 'TOYOTA' 'VOLKSWAGEN' 'VOLVO' 'GENESIS' 'BUGATTI']
```

In [12]:

```
# Putting different transmission sub-catagories into their respective catagories
ds['Transmission'] = np.where(ds['Transmission'].isin(['A4','A5','A6','A7','A8','A9'
ds['Transmission'] = np.where(ds['Transmission'].isin(['AS4','AS5','AS6','AS7','AS8'
ds['Transmission'] = np.where(ds['Transmission'].isin(['AM5','AM6','AM7','AM8','AM9'
ds['Transmission'] = np.where(ds['Transmission'].isin(['AV','AV6','AV7','AV8','AV10'
ds['Transmission'] = np.where(ds['Transmission'].isin(['M5','M6','M7']),'Manual',ds[

print(ds['Transmission'].unique())
```

```
['Automatic of Selective type' 'Manual' 'CVT' 'Automated Manual'
 'Automatic']
```

In [13]:

```
# Renaming fuel types for better understanding

print(ds['Fuel Type'].value_counts())

ds['Fuel Type']= np.where(ds['Fuel Type']=='X','Regular gasoline',ds['Fuel Type'])
ds['Fuel Type']= np.where(ds['Fuel Type']=='Z','Premium gasoline',ds['Fuel Type'])
ds['Fuel Type']= np.where(ds['Fuel Type']=='E','Ethanol',ds['Fuel Type'])
ds['Fuel Type']= np.where(ds['Fuel Type']=='D','Diesel',ds['Fuel Type'])
ds['Fuel Type']= np.where(ds['Fuel Type']=='N','Natural gas',ds['Fuel Type'])

print(ds['Fuel Type'].unique())
```

```
X    3637
Z    3202
E     370
D     175
N       1
Name: Fuel Type, dtype: int64
['Premium gasoline' 'Diesel' 'Regular gasoline' 'Ethanol' 'Natural gas']
```

In [14]:

```
print(ds['Vehicle Class'].unique())
```

```
['COMPACT' 'SUV - SMALL' 'MID-SIZE' 'TWO-SEATER' 'MINICOMPACT'
 'SUBCOMPACT' 'FULL-SIZE' 'STATION WAGON - SMALL' 'SUV - STANDARD'
 'VAN - CARGO' 'VAN - PASSENGER' 'PICKUP TRUCK - STANDARD' 'MINIVAN'
 'SPECIAL PURPOSE VEHICLE' 'STATION WAGON - MID-SIZE'
 'PICKUP TRUCK - SMALL']
```

In [15]:

```
ds.shape
```

Out[15]:

```
(7385, 12)
```

In [16]:

```
ds.head()
```

Out[16]:

	Make	Model	Vehicle Class	Engine Size(L)	Cylinders	Transmission	Fuel Type	Fuel Consumption City (L/100 km)	Fu Consumptic Hwy (L/1(kn
0	ACURA	ILX	COMPACT	2.0	4	Automatic of Selective type	Premium gasoline	9.9	6
1	ACURA	ILX	COMPACT	2.4	4	Manual	Premium gasoline	11.2	7
2	ACURA	ILX HYBRID	COMPACT	1.5	4	CVT	Premium gasoline	6.0	5
3	ACURA	MDX 4WD	SUV - SMALL	3.5	6	Automatic of Selective type	Premium gasoline	12.7	9
4	ACURA	RDX AWD	SUV - SMALL	3.5	6	Automatic of Selective type	Premium gasoline	12.1	8

In [17]:

```
ds.corr()['CO2_emission'].sort_values()
```

Out[17]:

```
Fuel Consumption Comb (mpg)        -0.907426
Cylinders                           0.832644
Engine Size(L)                      0.851145
Fuel Consumption Hwy (L/100 km)     0.883536
Fuel Consumption Comb (L/100 km)    0.918052
Fuel Consumption City (L/100 km)    0.919592
CO2_emission                        1.000000
Name: CO2_emission, dtype: float64
```

In [18]:

```
# Correlation between features and label

ds.corr()
```

Out[18]:

	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_e
Engine Size(L)	1.000000	0.927653	0.831379	0.761526	0.817060	-0.757854	(
Cylinders	0.927653	1.000000	0.800702	0.715252	0.780534	-0.719321	(

	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_e
Fuel Consumption City (L/100 km)	0.831379	0.800702	1.000000	0.948180	0.993810	-0.927059	(
Fuel Consumption Hwy (L/100 km)	0.761526	0.715252	0.948180	1.000000	0.977299	-0.890638	(
Fuel Consumption Comb (L/100 km)	0.817060	0.780534	0.993810	0.977299	1.000000	-0.925576	(
Fuel Consumption Comb (mpg)	-0.757854	-0.719321	-0.927059	-0.890638	-0.925576	1.000000	-(
CO2_emission	0.851145	0.832644	0.919592	0.883536	0.918052	-0.907426	'

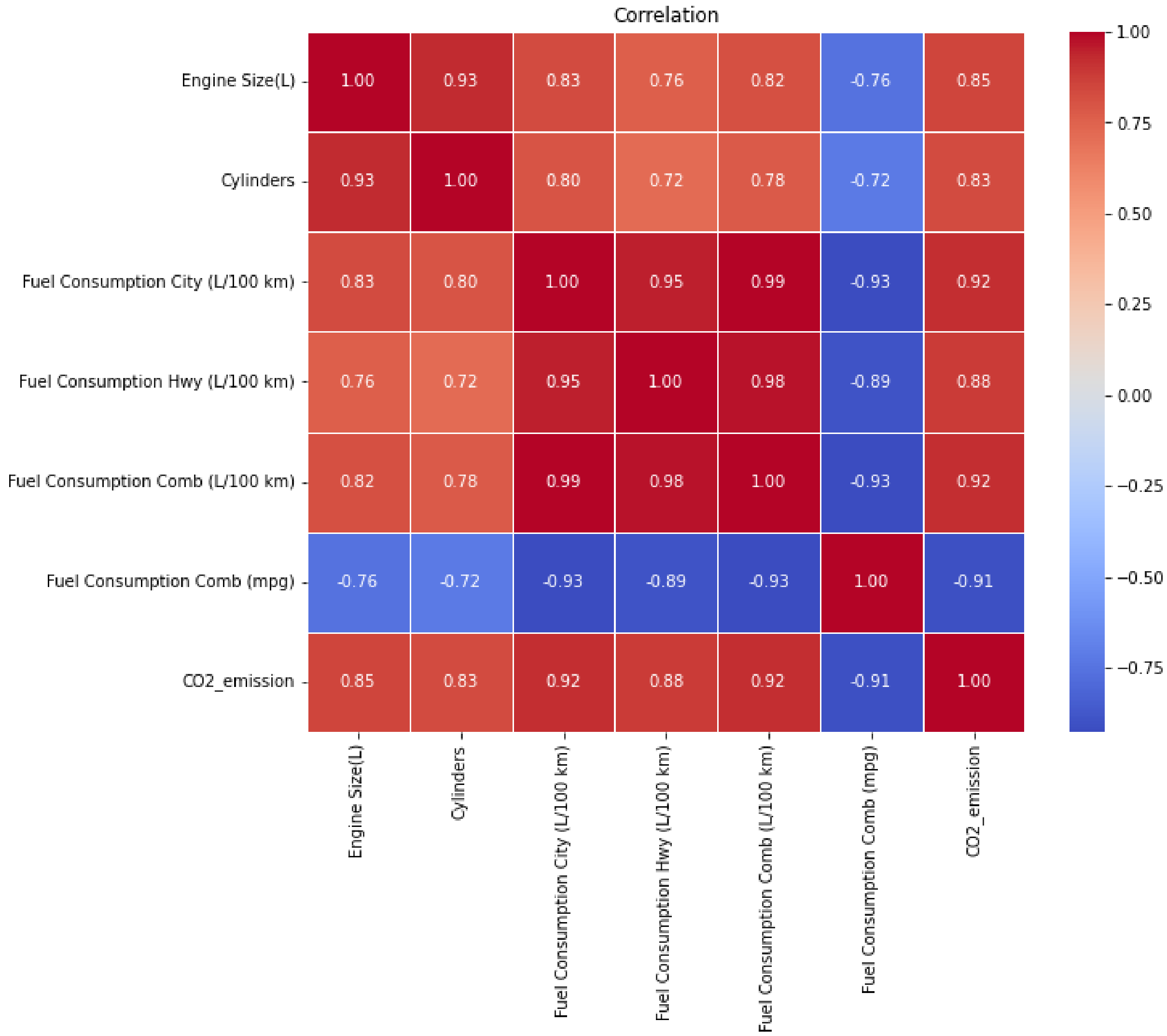
VISUALISATIONS

In [19]:

```
# VISUALISATIONS

corr = ds.corr()

plt.rcParams['figure.figsize']=(10,8)
sns.heatmap(corr, cmap='coolwarm', linewidth=0.5, fmt='0.2f', annot=True)

plt.title('Correlation')
```

Out[19]:

```
Text(0.5, 1.0, 'Correlation')
```

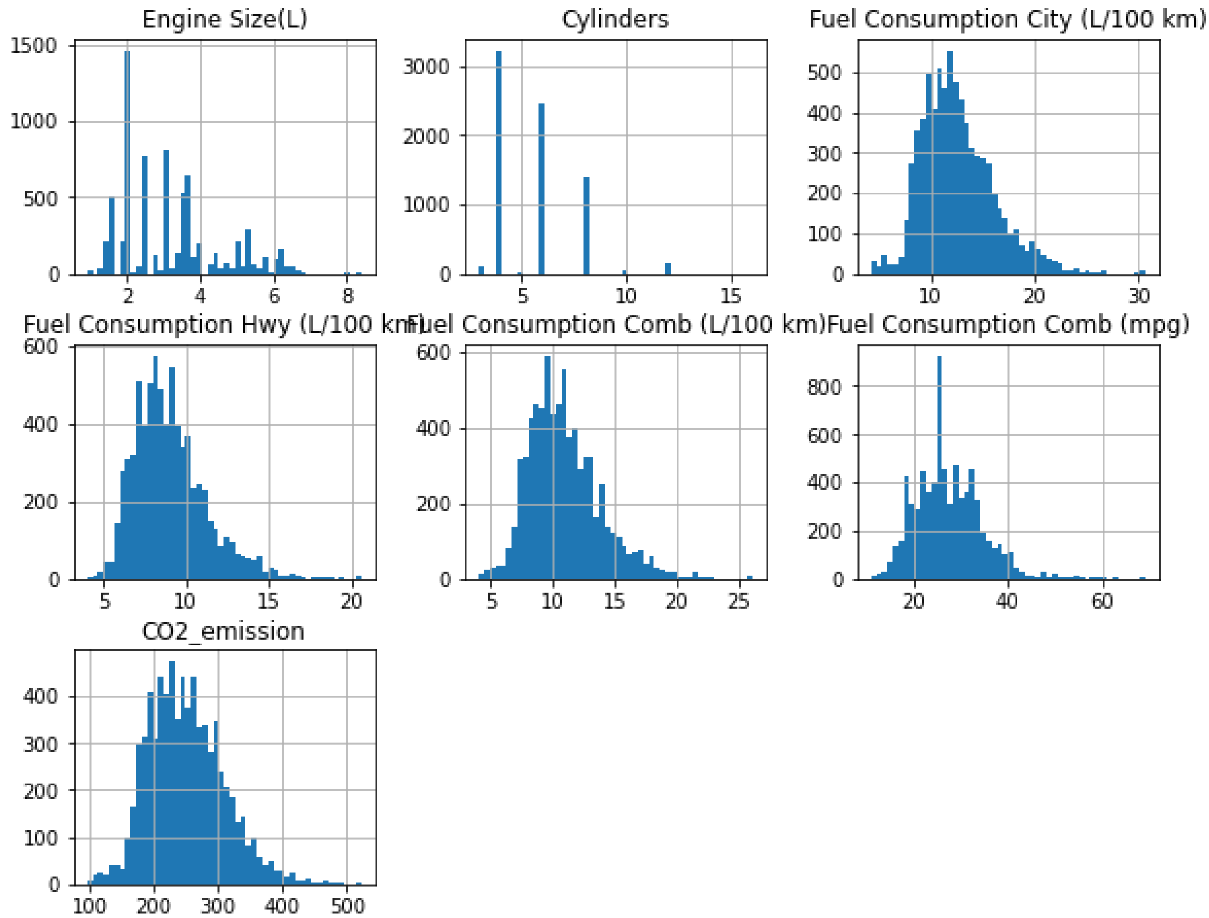


In [20]:

```
# distribution of numerical features
# we can seee that the numerical features are little ight skewed.

ds.hist(figsize=(10,8),bins=50)
```

Out[20]:

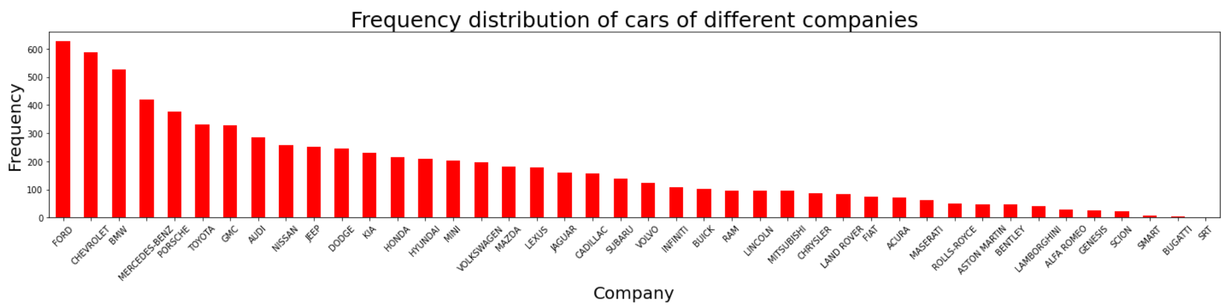
```
array([[<AxesSubplot:title={'center':'Engine Size(L)'}>,
        <AxesSubplot:title={'center':'Cylinders'}>,
        <AxesSubplot:title={'center':'Fuel Consumption City (L/100 km)'}>],
       [<AxesSubplot:title={'center':'Fuel Consumption Hwy (L/100 km)'}>,
        <AxesSubplot:title={'center':'Fuel Consumption Comb (L/100 km)'}>,
        <AxesSubplot:title={'center':'Fuel Consumption Comb (mpg)'}>],
       [<AxesSubplot:title={'center':'CO2_emission'}>, <AxesSubplot:>,
        <AxesSubplot:>]], dtype=object)
```



FREQUENCY DISTRIBUTION OF DIFFFRENT FEATURES

In [21]:

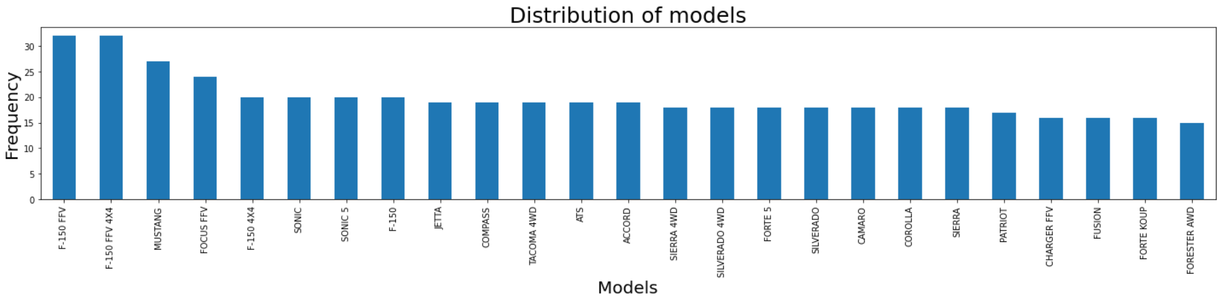
```
#Make
plt.figure(figsize=(20,5))

ds.groupby('Make')['Make'].count().sort_values(ascending=False).plot(kind='bar',colo

plt.title('Frequency distribution of cars of different companies', fontsize=25)
plt.xlabel('Company', fontsize=20)
plt.ylabel('Frequency', fontsize=20)
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



In [22]:

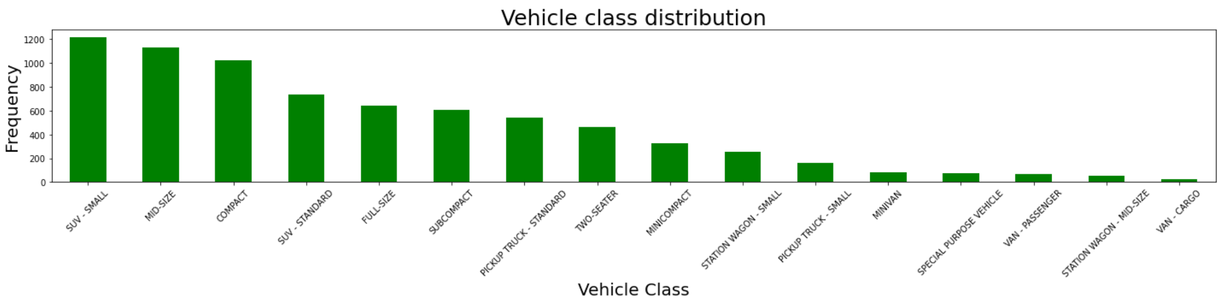
```
# MODEL
plt.figure(figsize=(20,5))

ds.groupby('Model')['Model'].count().sort_values(ascending=False)[:25].plot(kind='ba

plt.title('Distribution of models', fontsize=25)
plt.xlabel('Models', fontsize=20)
plt.ylabel('Frequency', fontsize=20)
```

```
plt.xticks(rotation=90)
plt.tight_layout()
plt.show()
```



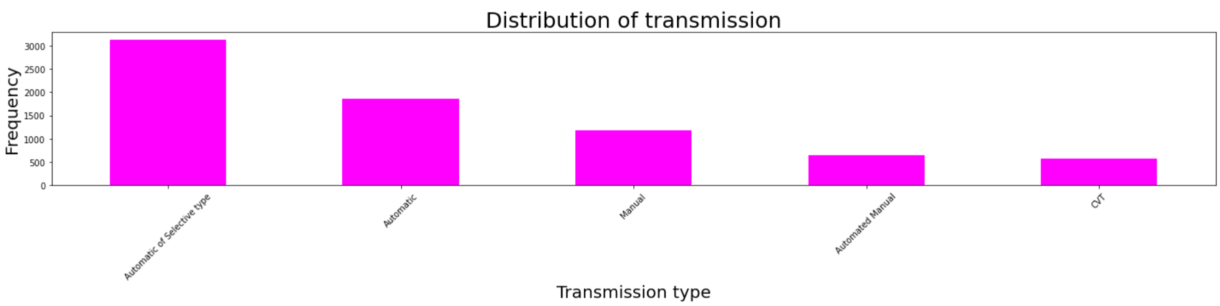
In [23]:

```
# Vehicle Class

plt.figure(figsize=(20,5))

ds.groupby('Vehicle Class')['Vehicle Class'].count().sort_values(ascending=False).pl

plt.title('Vehicle class distribution', fontsize=25)
plt.xlabel('Vehicle Class', fontsize=20)
plt.ylabel('Frequency', fontsize=20)
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



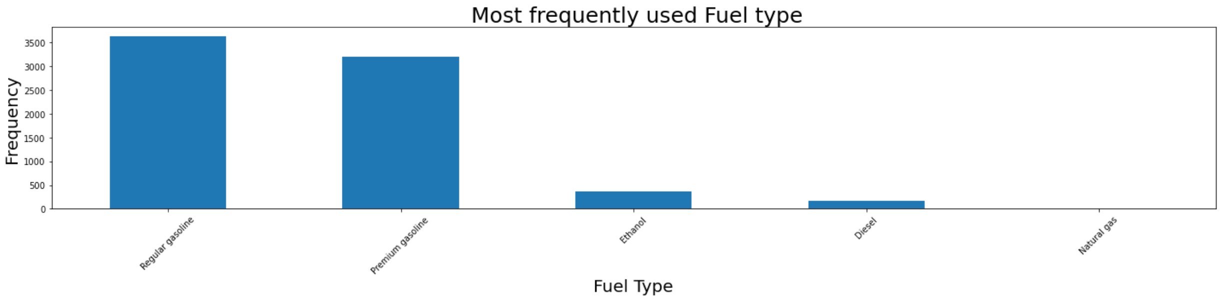
In [24]:

```
# Transmission

plt.figure(figsize=(20,5))

ds.groupby('Transmission')['Transmission'].count().sort_values(ascending=False).plot

plt.title('Distribution of transmission', fontsize=25)
plt.xlabel('Transmission type', fontsize=20)
plt.ylabel('Frequency', fontsize=20)
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



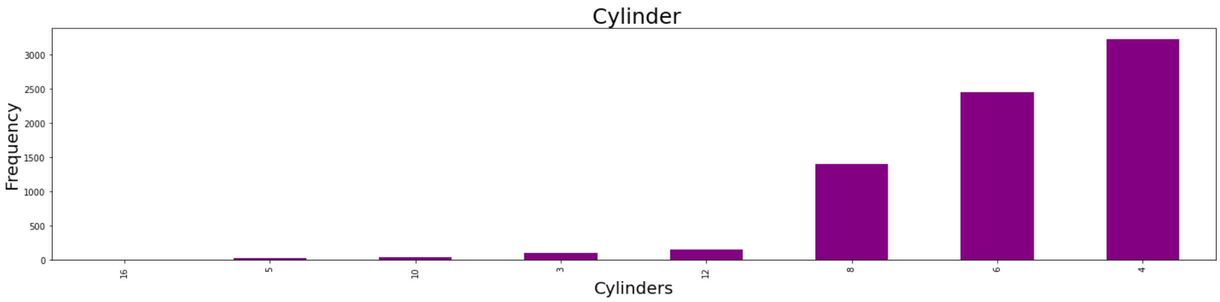
In [25]:

```python
# Fuel Type

plt.figure(figsize=(20,5))

ds.groupby('Fuel Type')['Fuel Type'].count().sort_values(ascending=False).plot(kind=

plt.title(' Most frequently used Fuel type', fontsize=25)
plt.xlabel('Fuel Type', fontsize=20)
plt.ylabel('Frequency', fontsize=20)
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



In [26]:

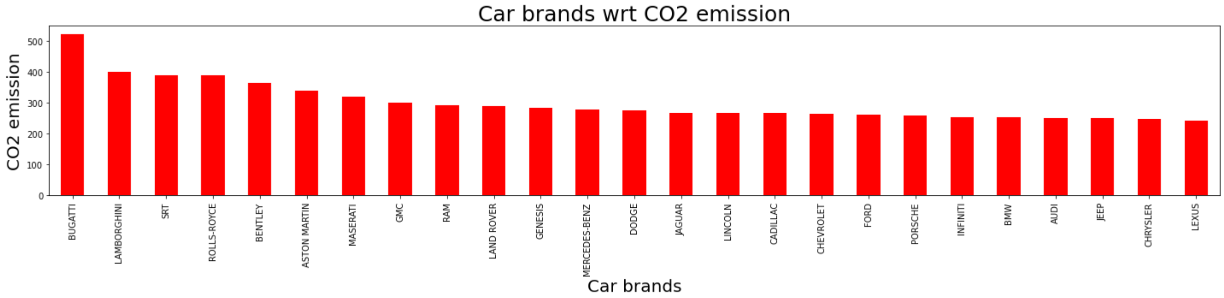
```python
# Cylinders

plt.figure(figsize=(20,5))

ds.groupby('Cylinders')['Cylinders'].count().sort_values(ascending=True).plot(kind='

plt.title(' Cylinder', fontsize=25)
plt.xlabel('Cylinders', fontsize=20)
plt.ylabel('Frequency', fontsize=20)
plt.xticks(rotation=90)
plt.tight_layout()
plt.show()
```



FEATURE DISTRIBUTION WITH RESPECT TO CO2 EMISSION

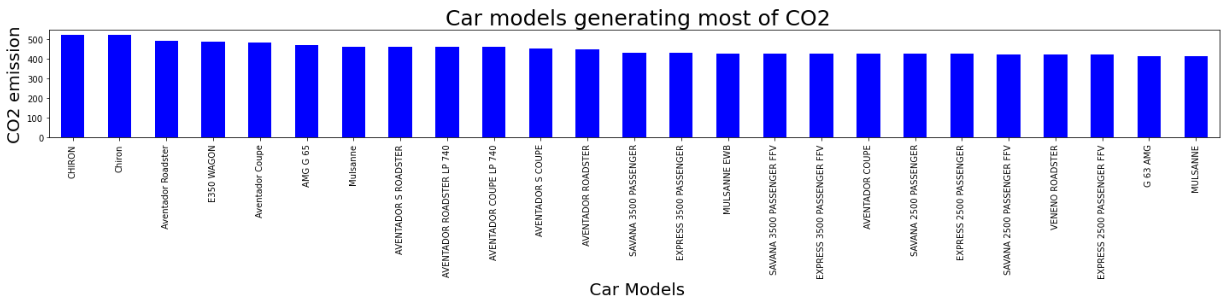
In [27]:

```python
# Visualisation wrt CO2 emision

plt.figure(figsize=(20,5))

ds.groupby('Make')['CO2_emission'].mean().sort_values(ascending=False)[:25].plot(kin

plt.title('Car brands wrt CO2 emission', fontsize=25)
plt.xlabel('Car brands', fontsize=20)
plt.ylabel('CO2 emission', fontsize=20)
plt.xticks(rotation=90)
plt.tight_layout()
plt.show()
```



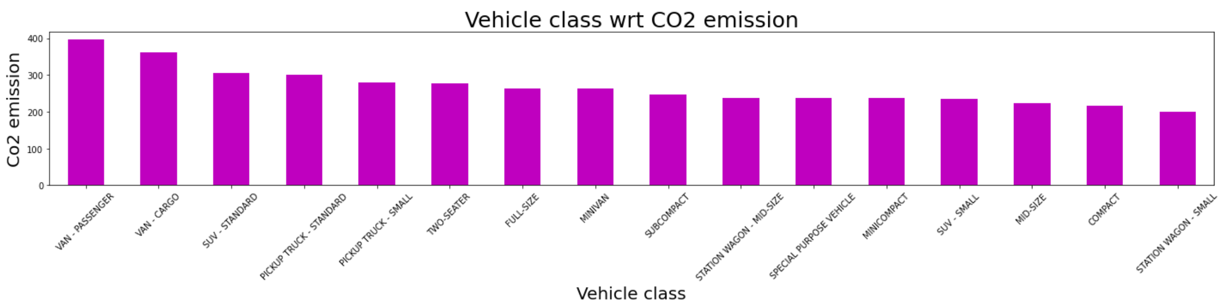
In [28]:

```
plt.figure(figsize=(20,5))

ds.groupby('Model')['CO2_emission'].mean().sort_values(ascending=False)[:25].plot(ki

plt.title(' Car models generating most of CO2', fontsize=25)
plt.xlabel(' Car Models', fontsize=20)
plt.ylabel('CO2 emission', fontsize=20)
plt.xticks(rotation=90)
plt.tight_layout()
plt.show()
```



In [29]:

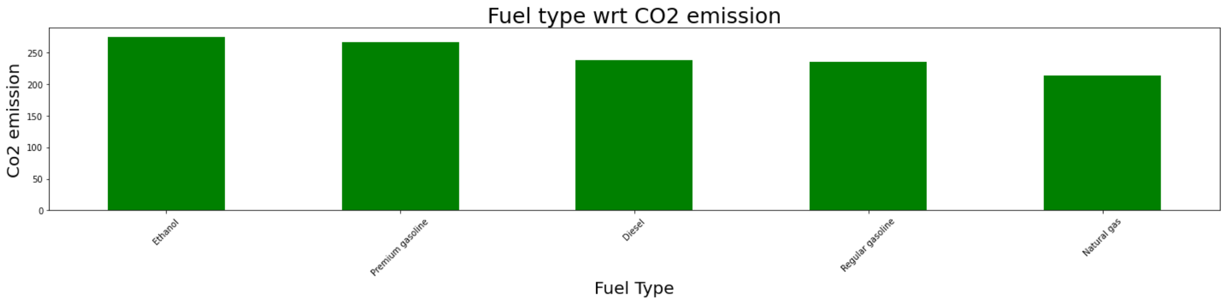
```
plt.figure(figsize=(20,5))

ds.groupby('Vehicle Class')['CO2_emission'].mean().sort_values(ascending=False).plot

plt.title('Vehicle class wrt CO2 emission', fontsize=25)
plt.xlabel('Vehicle class' , fontsize=20)
plt.ylabel('Co2 emission', fontsize=20)
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



In [30]:

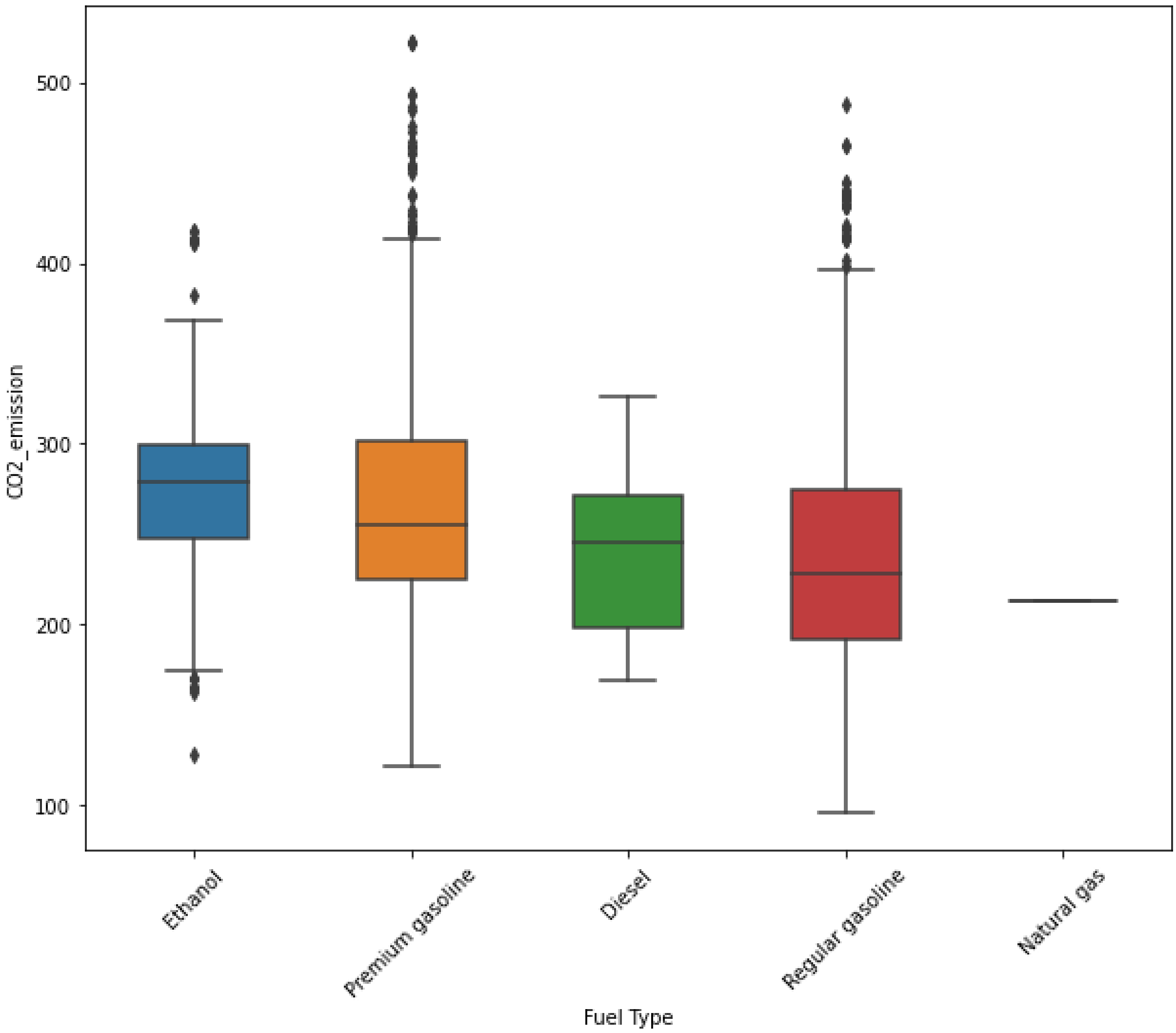
```
plt.figure(figsize=(20,5))

ds.groupby('Fuel Type')['CO2_emission'].mean().sort_values(ascending=False).plot(kin

plt.title('Fuel type wrt CO2 emission', fontsize=25)
plt.xlabel('Fuel Type', fontsize=20)
plt.ylabel('Co2 emission', fontsize=20)
plt.xticks(rotation=45)
```

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



In [31]:

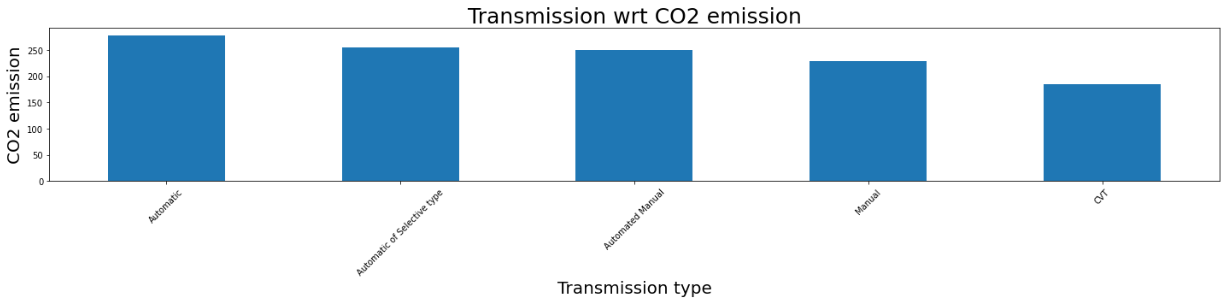
```
fuel_type = ds.groupby('Fuel Type')['CO2_emission'].median().sort_values(ascending=F
plt.figure(figsize=(10,8))
sns.boxplot(x = 'Fuel Type', y='CO2_emission', data =ds, order=fuel_type, width=0.5)
plt.xticks(rotation=45, horizontalalignment='center')

plt.show()
```



In [32]:

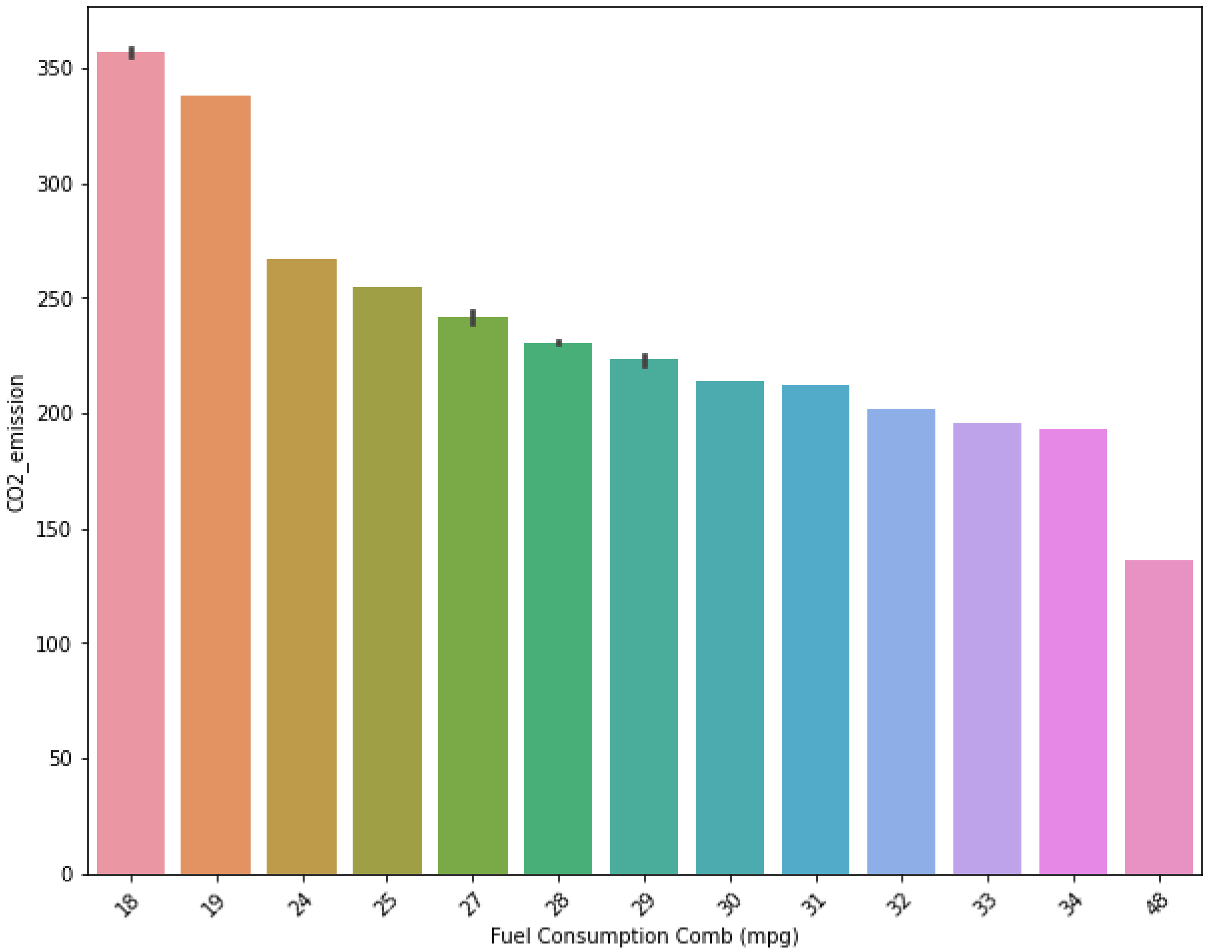
```
plt.figure(figsize=(20,5))

ds.groupby('Transmission')['CO2_emission'].mean().sort_values(ascending=False).plot(

plt.title('Transmission wrt CO2 emission', fontsize=25)
plt.xlabel('Transmission type', fontsize=20)
plt.ylabel('CO2 emission', fontsize=20)
plt.xticks(rotation=45)
```

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



In [33]:

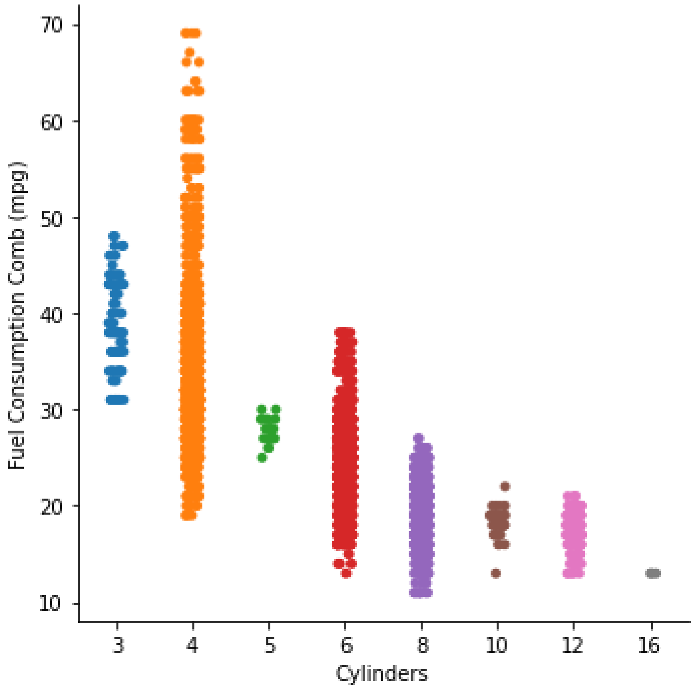
```
plt.figure(figsize=(10,8))
sns.barplot(x = 'Fuel Consumption Comb (mpg)', y='CO2_emission', data =ds[:25])

plt.xticks(rotation=45)
plt.show()
```



In [34]:

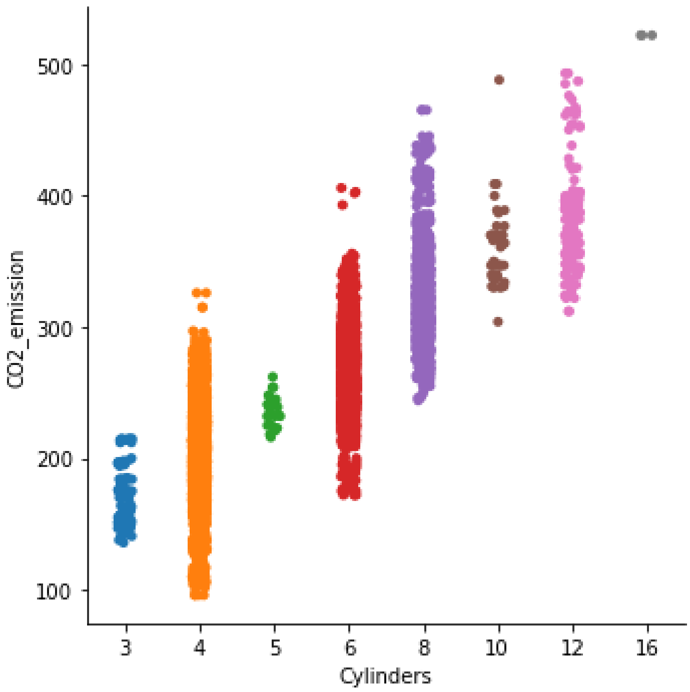
```
plt.figure(figsize=(10,8))

sns.catplot(x='Cylinders', y='Fuel Consumption Comb (mpg)',data = ds)

plt.show()
```

<Figure size 720x576 with 0 Axes>

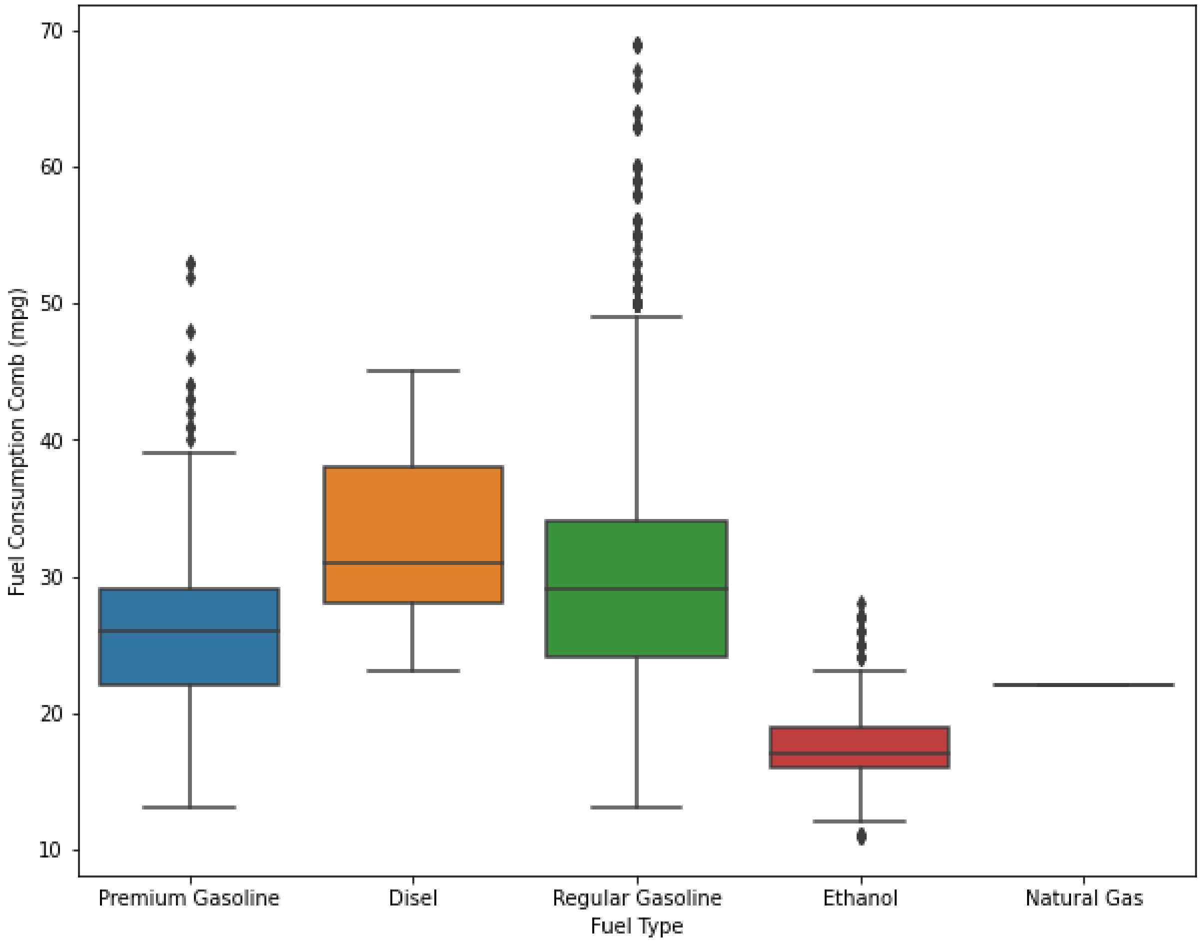


In [35]:
```
plt.figure(figsize=(10,8))

sns.catplot(x='Cylinders', y='CO2_emission',data = ds)

plt.show()                              # co2 emission increases with increasing n
```

```
<Figure size 720x576 with 0 Axes>
```



In [36]:
```
plt.figure(figsize=(10,8))
sns.boxplot(x = 'Fuel Type', y='Fuel Consumption Comb (mpg)', data =ds)

plt.xticks([0,1,2,3,4],['Premium Gasoline','Disel', 'Regular Gasoline','Ethanol', 'N
plt.show()                          # efficiency of fuel
```



DATA PREPROCESSING

In [37]:

```
# DATA PREPROCESSING

ds.head()
```

Out[37]:

	Make	Model	Vehicle Class	Engine Size(L)	Cylinders	Transmission	Fuel Type	Fuel Consumption City (L/100 km)	Fu Consumptic Hwy (L/1(kn
0	ACURA	ILX	COMPACT	2.0	4	Automatic of Selective type	Premium gasoline	9.9	6
1	ACURA	ILX	COMPACT	2.4	4	Manual	Premium gasoline	11.2	7
2	ACURA	ILX HYBRID	COMPACT	1.5	4	CVT	Premium gasoline	6.0	5
3	ACURA	MDX 4WD	SUV - SMALL	3.5	6	Automatic of Selective type	Premium gasoline	12.7	9
4	ACURA	RDX AWD	SUV - SMALL	3.5	6	Automatic of Selective type	Premium gasoline	12.1	8

In [38]:

```
ds['Transmission'].value_counts()
```

```
Automatic of Selective type    3127
```

Out[38]:
```
Automatic          1851
Manual             1185
Automated Manual    646
CVT                 576
Name: Transmission, dtype: int64
```

In [39]:
```
ds['Fuel Type'].value_counts()
```

Out[39]:
```
Regular gasoline    3637
Premium gasoline    3202
Ethanol              370
Diesel               175
Natural gas            1
Name: Fuel Type, dtype: int64
```

In [40]:
```
# Dropping natural gas as there is only one data we have which would not make much d

ds_N = ds[ds['Fuel Type']== 'Natural gas']

ind = ds_N.index

ds_N
```

Out[40]:

	Make	Model	Vehicle Class	Engine Size(L)	Cylinders	Transmission	Fuel Type	Fuel Consumption City (L/100 km)	Consun Hwy (
2439	CHEVROLET	IMPALA DUAL FUEL	MID-SIZE	3.6	6	Automatic of Selective type	Natural gas	15.2	

In [41]:
```
for i in ind:
    ds.drop(i, axis=0, inplace=True)
```

In [42]:
```
ds[ds['Fuel Type']=='Natural gas']
```

Out[42]:

Make	Model	Vehicle Class	Engine Size(L)	Cylinders	Transmission	Fuel Type	Fuel Consumption City (L/100 km)	Fuel Consumption Hwy (L/100 km)	Consu Comb

In [43]:
```
# creating dummy variables of fuel type and transmission (catagorical features)

d_v =pd.get_dummies(ds['Fuel Type'], prefix='Fuel', drop_first=True)
dv = pd.get_dummies(ds["Transmission"], drop_first=True)
d_v.head()
```

Out[43]:

	Fuel_Ethanol	Fuel_Premium gasoline	Fuel_Regular gasoline
0	0	1	0
1	0	1	0

	Fuel_Ethanol	Fuel_Premium gasoline	Fuel_Regular gasoline
2	0	1	0
3	0	1	0
4	0	1	0

In [44]:

```
dv.head()
```

Out[44]:

	Automatic	Automatic of Selective type	CVT	Manual
0	0	1	0	0
1	0	0	0	1
2	0	0	1	0
3	0	1	0	0
4	0	1	0	0

In [45]:

```
df = [ds, d_v,dv]

data = pd.concat(df, axis=1)
data.head()
```

Out[45]:

	Make	Model	Vehicle Class	Engine Size(L)	Cylinders	Transmission	Fuel Type	Fuel Consumption City (L/100 km)	Fu Consumptic Hwy (L/1(kn
0	ACURA	ILX	COMPACT	2.0	4	Automatic of Selective type	Premium gasoline	9.9	6
1	ACURA	ILX	COMPACT	2.4	4	Manual	Premium gasoline	11.2	7
2	ACURA	ILX HYBRID	COMPACT	1.5	4	CVT	Premium gasoline	6.0	5
3	ACURA	MDX 4WD	SUV - SMALL	3.5	6	Automatic of Selective type	Premium gasoline	12.7	9
4	ACURA	RDX AWD	SUV - SMALL	3.5	6	Automatic of Selective type	Premium gasoline	12.1	8

In [46]:

```
data.drop(['Fuel Type'], inplace=True, axis=1)
data.drop(['Transmission'], inplace=True, axis=1)
```

HANDLING OTHER CATAGORICAL FEATURES HAVING MULTIPLE CATAGORIES (MAKE , MODEL, VEHICLE CLASS)

In [47]:

```
df_freq = data['Make'].value_counts().to_dict()
mod_freq = data['Model'].value_counts().to_dict()
veh_freq = data['Vehicle Class'].value_counts().to_dict()
```

In [48]:

```
data['Make'] = data['Make'].map(df_freq)
data['Model'] = data['Model'].map(mod_freq)
data['Vehicle Class'] = data['Vehicle Class'].map(veh_freq)
```

In [49]:

```
data.head()
```

Out[49]:

	Make	Model	Vehicle Class	Engine Size(L)	Cylinders	Fuel Consumption City (L/100 km)	Fuel Consumption Hwy (L/100 km)	Fuel Consumption Comb (L/100 km)	Fu Consumpti Comb (mp
0	72	9	1022	2.0	4	9.9	6.7	8.5	
1	72	9	1022	2.4	4	11.2	7.7	9.6	
2	72	2	1022	1.5	4	6.0	5.8	5.9	
3	72	1	1217	3.5	6	12.7	9.1	11.1	
4	72	7	1217	3.5	6	12.1	8.7	10.6	

DIVIDING DATA SET INTO INDEPENDENT AND DEPENDENT VARIABLE

In [50]:

```
X = data.drop('CO2_emission', axis=1)
y = data['CO2_emission']
```

In [51]:

```
X.head()
```

Out[51]:

	Make	Model	Vehicle Class	Engine Size(L)	Cylinders	Fuel Consumption City (L/100 km)	Fuel Consumption Hwy (L/100 km)	Fuel Consumption Comb (L/100 km)	Fu Consumpti Comb (mp
0	72	9	1022	2.0	4	9.9	6.7	8.5	
1	72	9	1022	2.4	4	11.2	7.7	9.6	
2	72	2	1022	1.5	4	6.0	5.8	5.9	
3	72	1	1217	3.5	6	12.7	9.1	11.1	
4	72	7	1217	3.5	6	12.1	8.7	10.6	

In [52]:

```
y.head()
```

Out[52]:

```
0    196
1    221
2    136
3    255
4    244
Name: CO2_emission, dtype: int64
```

In [53]:

```
data.shape
```

Out[53]: (7384, 17)

FEATURE SELECTION USING CHI-SQUARE TEST

In [54]:
```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
```

In [55]:
```
ranked_feature = SelectKBest(score_func = chi2, k='all')
ordered_feature = ranked_feature.fit(X, y)
```

In [56]:
```
top_feat = pd.DataFrame(ordered_feature.scores_ , columns=['score'])
top_feat['variables'] = X.columns
```

In [57]:
```
top_feat.sort_values(by='score', ascending=False)
```

Out[57]:

	score	variables
2	173723.188600	Vehicle Class
0	95021.040629	Make
8	13236.852695	Fuel Consumption Comb (mpg)
5	6232.307799	Fuel Consumption City (L/100 km)
7	4862.577319	Fuel Consumption Comb (L/100 km)
4	3412.144543	Cylinders
3	3394.028026	Engine Size(L)
6	3293.816329	Fuel Consumption Hwy (L/100 km)
1	2612.972867	Model
14	1938.744672	CVT
12	1022.032185	Automatic
9	825.862516	Fuel_Ethanol
15	774.942190	Manual
10	763.320532	Fuel_Premium gasoline
11	677.576319	Fuel_Regular gasoline
13	497.189839	Automatic of Selective type

CREATING TRAINING SET AND TESTING SET

In [58]:
```
# splitting of traing testing set into X and y

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
```

In [59]:
```
print(X_train.shape)
print(X_test.shape)
```

```
print(y_train.shape)
print(y_test.shape)
```

```
(5907, 16)
(1477, 16)
(5907,)
(1477,)
```

In [60]:
```
y_test.head()
```

Out[60]:
```
5632    368
1550    290
1128    382
6498    211
3270    193
Name: CO2_emission, dtype: int64
```

FEATURE SCALING USING STANDARDIZATION

In [61]:
```
# STANDARDIZATION
from sklearn.preprocessing import StandardScaler
```

In [62]:
```
scaler =StandardScaler()

X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

In [63]:
```
X_train
```

Out[63]:
```
array([[-0.27521116, -0.68670295, -1.80962504, ..., -0.85157221,
        -0.29299418, -0.44024503],
       [-0.54676246, -0.68670295, -0.42577231, ...,  1.17429853,
        -0.29299418, -0.44024503],
       [ 1.2100082 , -0.87774305, -0.52091218, ...,  1.17429853,
        -0.29299418, -0.44024503],
       ...,
       [ 0.11826113, -0.30462276, -0.14900176, ..., -0.85157221,
         3.4130371 , -0.44024503],
       [-0.27521116, -0.30462276,  1.24061702, ..., -0.85157221,
         3.4130371 , -0.44024503],
       [-1.17299301, -1.06878314, -0.71695798, ..., -0.85157221,
        -0.29299418, -0.44024503]])
```

In [64]:
```
data['CO2_emission'].mean()
```

Out[64]:
```
250.58978873239437
```

MODEL IMPLEMENTATION (Approach 1)

LINEAR REGRESSION

In [65]:
```
model = LinearRegression()
model.fit(X_train, y_train)
```

```
LinearRegression()
```

Out[65]:

In [66]:
```
model.intercept_
```

Out[66]:
```
250.98357880480785
```

In [67]:
```
model.coef_
```

Out[67]:
```
array([  0.09846243,  -0.09126632,  -0.03196586,   0.25410999,
         2.17469937,  24.28719842,  13.07132844,  20.53745833,
        -6.35439345, -30.19620392, -15.35456651, -15.09009955,
        -0.35425518,  -0.32284677,  -0.17696755,  -0.3645601 ])
```

In [68]:
```
y_pred = model.predict(X_test)
y_pred
```

Out[68]:
```
array([359.06209907, 292.97346311, 377.59960241, ..., 341.77783102,
       193.05390931, 177.97061893])
```

In [69]:
```
np.sqrt(mean_squared_error(y_test, y_pred))
```

Out[69]:
```
4.918260935039378
```

In [70]:
```
r2_score(y_test, y_pred)
```

Out[70]:
```
0.993041824997087
```

In [71]:
```
frames = [y_pred, y_test.values]
result_pred = pd.DataFrame(data=frames)
result_pred = result_pred.T
```

In [72]:
```
lin_pred = result_pred.rename(columns={0: 'pred_values',1:'real_values'})
lin_pred['pred_values'] = lin_pred['pred_values'].map(lambda x: round(x,2))

lin_pred
```

Out[72]:

	pred_values	real_values
0	359.06	368.0
1	292.97	290.0
2	377.60	382.0
3	210.80	211.0
4	192.94	193.0
...	...	...
1472	233.49	235.0
1473	262.28	263.0
1474	341.78	346.0
1475	193.05	193.0

	pred_values	real_values
1476	177.97	177.0

1477 rows × 2 columns

In [73]:
```
lin_pred['diff'] = abs(lin_pred['pred_values'] - lin_pred['real_values'])

print('mean diff: ', (abs(lin_pred['diff']).mean()))
```

```
mean diff:  2.979058903182132
```

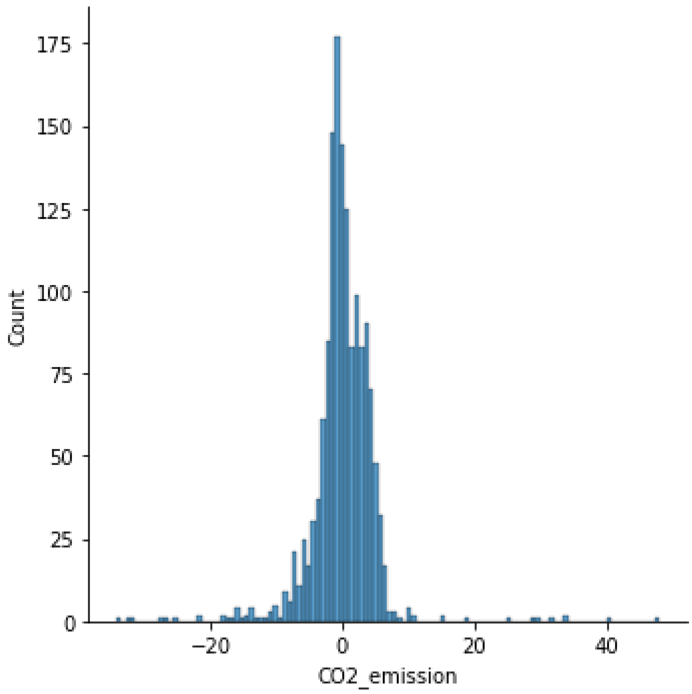
In [74]:
```
lin_pred.head(10)
```

Out[74]:

	pred_values	real_values	diff
0	359.06	368.0	8.94
1	292.97	290.0	2.97
2	377.60	382.0	4.40
3	210.80	211.0	0.20
4	192.94	193.0	0.06
5	249.45	244.0	5.45
6	213.25	210.0	3.25
7	174.25	174.0	0.25
8	266.72	268.0	1.28
9	303.68	305.0	1.32

In [75]:
```
sns.displot(y_pred-y_test)
```

Out[75]:
```
<seaborn.axisgrid.FacetGrid at 0x16f4521e580>
```

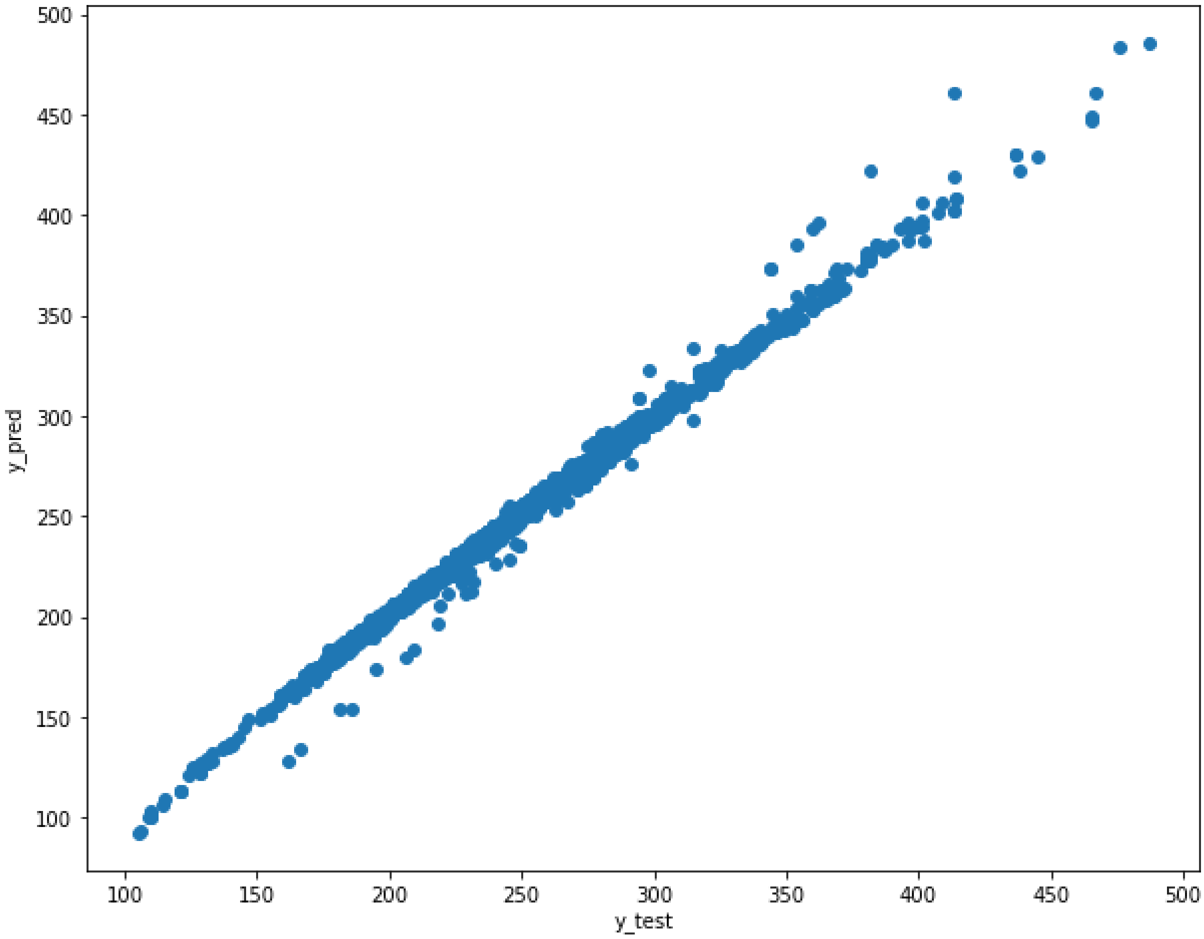


In [76]:

```
plt.scatter( y_test,y_pred)
plt.xlabel('y_test')
plt.ylabel('y_pred')
```

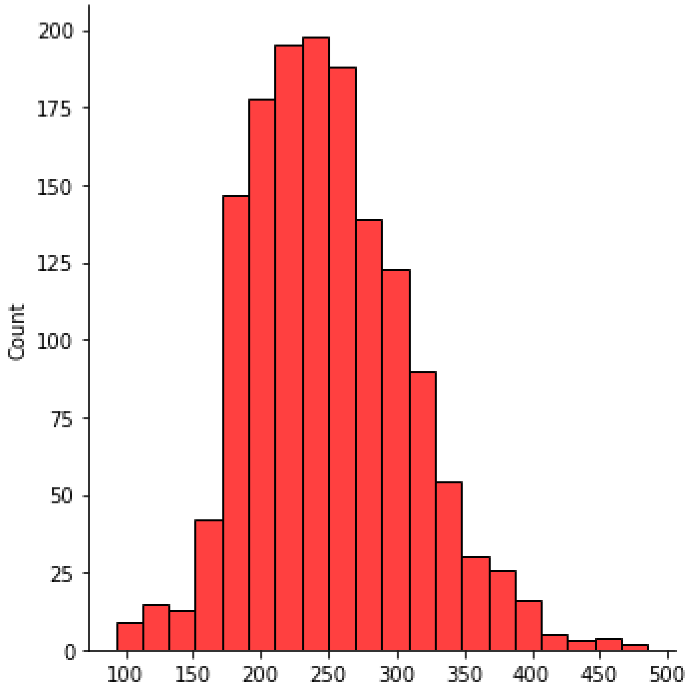
Out[76]:

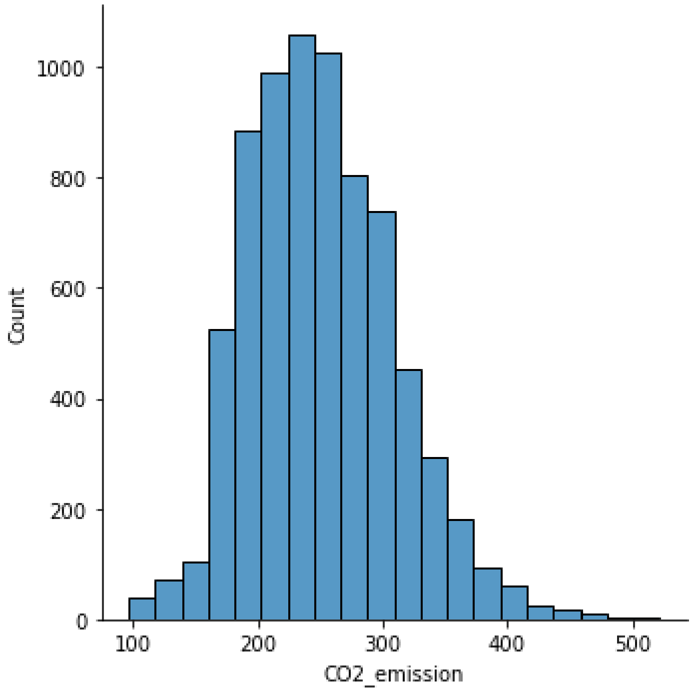
```
Text(0, 0.5, 'y_pred')
```



In [77]:

```
sns.displot(y_pred, bins=20,color='red')
plt.show()
sns.displot(data['CO2_emission'], bins=20)
plt.show()
```





DECISION TREE REGRESSION

In [78]:
```
from sklearn.tree import DecisionTreeRegressor

model = DecisionTreeRegressor(random_state = 42)

model.fit(X_train, y_train)
```

Out[78]:
```
DecisionTreeRegressor(random_state=42)
```

In [79]:
```
dtr_pred = model.predict(X_test)
dtr_pred
```

Out[79]:
```
array([357.        , 290.        , 382.        , ..., 342.66666667,
       193.        , 177.        ])
```

In [80]:
```
np.sqrt(mean_squared_error(y_test,dtr_pred ))
```
Out[80]:
```
3.629475553464912
```
In [81]:
```
r2_score(y_test, dtr_pred)
```
Out[81]:
```
0.9962106914830545
```
In [82]:
```
frames = [dtr_pred, y_test.values]
result_pred = pd.DataFrame(data=frames)
result_pred = result_pred.T
result_pred.head()
```
Out[82]:

	0	1
0	357.0	368.0
1	290.0	290.0
2	382.0	382.0
3	211.0	211.0
4	193.0	193.0

In [83]:
```
dtr_pred = result_pred.rename(columns={0: 'pred_values', 1:'real_values'})
dtr_pred['pred_values'] = (dtr_pred['pred_values'].map(lambda x: round(x,2)))

dtr_pred['diff'] = abs(dtr_pred['real_values'] -dtr_pred['pred_values'])


print('mean diff: ', abs(dtr_pred['diff']).mean())
```
```
mean diff:  1.7914691943127963
```
In [84]:
```
dtr_pred.head(10)
```
Out[84]:

	pred_values	real_values	diff
0	357.0	368.0	11.0
1	290.0	290.0	0.0
2	382.0	382.0	0.0
3	211.0	211.0	0.0
4	193.0	193.0	0.0
5	244.0	244.0	0.0
6	210.0	210.0	0.0
7	174.0	174.0	0.0
8	267.0	268.0	1.0
9	304.6	305.0	0.4

RANDOM FOREST

In [85]:
```
rf_model = RandomForestRegressor()
rf_model.fit(X_train, y_train)
```

Out[85]:
```
RandomForestRegressor()
```

In [86]:
```
y_rf_pred = rf_model.predict(X_test)
y_rf_pred
```

Out[86]:
```
array([359.128     , 290.51619048, 382.96      , ..., 343.2952381 ,
       191.2125    , 177.75      ])
```

In [87]:
```
print('RMSE: {:0.4f}'.format(np.sqrt(mean_squared_error(y_test,y_rf_pred))))
print('MAE: {:0.4f}'.format(mean_absolute_error(y_test,y_rf_pred)))
print('R2_score: {:0.4f}'.format(r2_score(y_test,y_rf_pred)))
```

```
RMSE: 3.1729
MAE: 1.9497
R2_score: 0.9971
```

In [88]:
```
frames = [y_rf_pred, y_test.values]
result_pred = pd.DataFrame(data=frames)
result_pred = result_pred.T
result_pred.head()
```

Out[88]:

	0	1
0	359.128000	368.0
1	290.516190	290.0
2	382.960000	382.0
3	211.193333	211.0
4	192.660000	193.0

In [89]:
```
y_rf_pred = result_pred.rename(columns={0: 'pred_values', 1:'real_values'})
y_rf_pred['pred_values'] = (y_rf_pred['pred_values'].map(lambda x: round(x,2)))

y_rf_pred['diff'] = abs(y_rf_pred['real_values'] -y_rf_pred['pred_values'])


print('mean diff: ', abs(y_rf_pred['diff']).mean())
```

```
mean diff:  1.9495463777928175
```

In [90]:
```
y_rf_pred.head(10)
```

Out[90]:

	pred_values	real_values	diff
0	359.13	368.0	8.87
1	290.52	290.0	0.52
2	382.96	382.0	0.96

	pred_values	real_values	diff
3	211.19	211.0	0.19
4	192.66	193.0	0.34
5	245.38	244.0	1.38
6	211.55	210.0	1.55
7	174.44	174.0	0.44
8	266.94	268.0	1.06
9	304.62	305.0	0.38

SIMPLE VECTOR MACHINE

In [91]:

```
from sklearn.svm import LinearSVR
model = LinearSVR()
model.fit(X_train, y_train)
y_svr_pred = model.predict(X_test)
y_svr_pred
```

Out[91]:

```
array([363.60358933, 294.16469554, 383.39453728, ..., 344.90703477,
       191.74388406, 176.98221956])
```

In [92]:

```
np.sqrt(mean_squared_error(y_svr_pred,y_test))
```

Out[92]:

```
5.445718017534262
```

In [93]:

```
r2_score(y_svr_pred,y_test)
```

Out[93]:

```
0.991821600663927
```

In [94]:

```
frames = [y_svr_pred, y_test.values]
result_pred = pd.DataFrame(data=frames)
result_pred = result_pred.T
result_pred.head()
```

Out[94]:

	0	1
0	363.603589	368.0
1	294.164696	290.0
2	383.394537	382.0
3	210.676727	211.0
4	192.440312	193.0

In [95]:

```
y_svr_pred = result_pred.rename(columns={0: 'pred_values', 1:'real_values'})
y_svr_pred['pred_values'] = (y_svr_pred['pred_values'].map(lambda x: round(x,2)))

y_svr_pred['diff'] = abs(y_svr_pred['real_values'] -y_svr_pred['pred_values'])
```

```
print('mean diff: ', abs(y_svr_pred['diff']).mean())
```

mean diff: 2.806222071767095

In [96]:

```
y_svr_pred.head(10)
```

Out[96]:

	pred_values	real_values	diff
0	363.60	368.0	4.40
1	294.16	290.0	4.16
2	383.39	382.0	1.39
3	210.68	211.0	0.32
4	192.44	193.0	0.56
5	248.70	244.0	4.70
6	210.99	210.0	0.99
7	174.22	174.0	0.22
8	266.64	268.0	1.36
9	304.33	305.0	0.67

APPROACH 2

In [97]:

```
models =['LinReg','DT', 'RF','SVR']

frame = pd.DataFrame(columns={'models':[],'rmse_train':[], 'mae_train':[],'r2_train'




for i in range(len(models)):
    if models[i] == 'LinReg':
        model = LinearRegression()
        model.fit(X_train,y_train)
        pred_train = model.predict(X_train)
        rmse_train = np.sqrt(mean_squared_error(y_train,pred_train))
        mae_train = mean_absolute_error(y_train,pred_train)
        r2_train = r2_score(y_train,pred_train)

        pred_test = model.predict(X_test)
        rmse_test = np.sqrt(mean_squared_error(y_test,pred_test))
        mae_test = mean_absolute_error(y_test,pred_test)
        r2_test = r2_score(y_test,pred_test)
        frame.loc[frame.shape[0]] = ['Linear Regression', rmse_train, mae_train,r2_t


    elif models[i] =='DT':
        model = DecisionTreeRegressor()
        model.fit(X_train,y_train)
        pred_train = model.predict(X_train)
        rmse_train = np.sqrt(mean_squared_error(y_train,pred_train))
        mae_train = mean_absolute_error(y_train,pred_train)
        r2_train = r2_score(y_train,pred_train)
```

```
            pred_test = model.predict(X_test)
            rmse_test = np.sqrt(mean_squared_error(y_test,pred_test))
            mae_test = mean_absolute_error(y_test,pred_test)
            r2_test = r2_score(y_test,pred_test)
            frame.loc[frame.shape[0]] = ['Decision Tree Regression',rmse_train, mae_trai

        elif models[i] =='RF':
            model = RandomForestRegressor()
            model.fit(X_train,y_train)
            pred_train = model.predict(X_train)
            rmse_train = np.sqrt(mean_squared_error(y_train,pred_train))
            mae_train = mean_absolute_error(y_train,pred_train)
            r2_train = r2_score(y_train,pred_train)

            pred_test = model.predict(X_test)
            rmse_test = np.sqrt(mean_squared_error(y_test,pred_test))
            mae_test = mean_absolute_error(y_test,pred_test)
            r2_test = r2_score(y_test,pred_test)
            frame.loc[frame.shape[0]] = ['Random Forest Regression',rmse_train, mae_trai

        else :
            models[i] =='SVM'
            model = LinearSVR()
            model.fit(X_train,y_train)
            pred_train = model.predict(X_train)
            rmse_train = np.sqrt(mean_squared_error(y_train,pred_train))
            mae_train = mean_absolute_error(y_train,pred_train)
            r2_train = r2_score(y_train,pred_train)

            pred_test = model.predict(X_test)
            rmse_test = np.sqrt(mean_squared_error(y_test,pred_test))
            mae_test = mean_absolute_error(y_test,pred_test)
            r2_test = r2_score(y_test,pred_test)
            frame.loc[frame.shape[0]] = ['Simple Vector Regression',rmse_train, mae_trai
```

OVERALL PERFORMANCE OF ALL MODELS IN A DATAFRAME

In [98]:

```
frame
```

Out[98]:

	models	rmse_train	mae_train	r2_train	rmse_test	mae_test	r2_test
0	Linear Regression	4.962163	2.996013	0.992778	4.918261	2.979052	0.993042
1	Decision Tree Regression	0.939813	0.318859	0.999741	3.752893	1.796705	0.995949
2	Random Forest Regression	1.465105	0.866309	0.999370	3.163007	1.941471	0.997122
3	Simple Vector Regression	5.346308	2.713163	0.991617	5.430713	2.799582	0.991516