Importing required packages

In [8]:
```python
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
import numpy as np
from sklearn.metrics import mean_squared_error,mean_absolute_error,r2_score
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
%matplotlib inline
```

Let's download and import the data on Fuel Consumption using *pandas* `read_csv()` method.

Download Dataset

Understanding the Data

`FuelConsumption.csv`:

We have downloaded a fuel consumption dataset, **`FuelConsumptionData.csv`**, which contains model-specific fuel consumption ratings and estimated carbon dioxide emissions for new light-duty vehicles for retail sale in Canada.

- **MODELYEAR** e.g. 2014
- **MAKE** e.g. Acura
- **MODEL** e.g. ILX
- **VEHICLE CLASS** e.g. SUV
- **ENGINE SIZE** e.g. 4.7
- **CYLINDERS** e.g 6
- **TRANSMISSION** e.g. A6
- **FUEL CONSUMPTION in CITY(L/100 km)** e.g. 9.9
- **FUEL CONSUMPTION in HWY (L/100 km)** e.g. 8.9
- **FUEL CONSUMPTION COMB (L/100 km)** e.g. 9.2
- **CO2 EMISSIONS (g/km)** e.g. 182 --> low --> 0

Reading the data

In [17]:
```python
df = pd.read_csv("FuelConsumptionData.csv")

# take a look at the dataset
df.head()
```

Out[17]:

	MODELYEAR	MAKE	MODEL	VEHICLECLASS	ENGINESIZE	CYLINDERS	TRANSMISSION	FUELTYPE	FUELCONSUMPTION_CITY	FUELC
0	2014	ACURA	ILX	COMPACT	2.0	4	AS5	Z	9.9	
1	2014	ACURA	ILX	COMPACT	2.4	4	M6	Z	11.2	
2	2014	ACURA	ILX HYBRID	COMPACT	1.5	4	AV7	Z	6.0	
3	2014	ACURA	MDX 4WD	SUV - SMALL	3.5	6	AS6	Z	12.7	
4	2014	ACURA	RDX AWD	SUV - SMALL	3.5	6	AS6	Z	12.1	

Data Exploration

Let's first have a descriptive exploration on our data.

In [18]:
```python
# summarize the data
df.describe()
```

Out[18]:

	MODELYEAR	ENGINESIZE	CYLINDERS	FUELCONSUMPTION_CITY	FUELCONSUMPTION_HWY	FUELCONSUMPTION_COMB	FUELCON
count	1067.0	1067.000000	1067.000000	1067.000000	1067.000000	1067.000000	
mean	2014.0	3.346298	5.794752	13.296532	9.474602	11.580881	
std	0.0	1.415895	1.797447	4.101253	2.794510	3.485595	
min	2014.0	1.000000	3.000000	4.600000	4.900000	4.700000	
25%	2014.0	2.000000	4.000000	10.250000	7.500000	9.000000	
50%	2014.0	3.400000	6.000000	12.600000	8.800000	10.900000	
75%	2014.0	4.300000	8.000000	15.550000	10.850000	13.350000	
max	2014.0	8.400000	12.000000	30.200000	20.500000	25.800000	

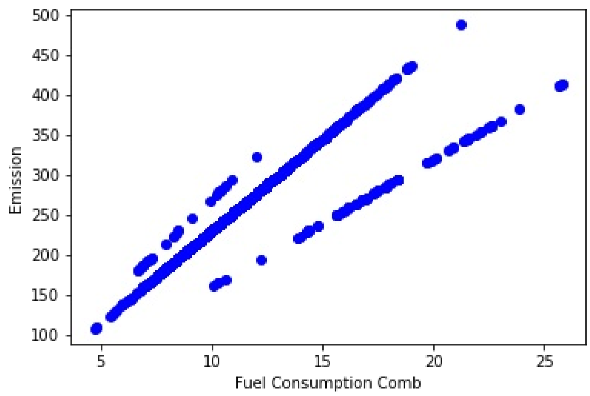
In [4]:
```
new_df = df[['ENGINESIZE','CYLINDERS','FUELCONSUMPTION_CITY','FUELCONSUMPTION_HWY','FUELCONSUMPTION_COMB','CO2E
new_df.head(7)
```

Out[4]:

	ENGINESIZE	CYLINDERS	FUELCONSUMPTION_CITY	FUELCONSUMPTION_HWY	FUELCONSUMPTION_COMB	CO2EMISSIONS
0	2.0	4	9.9	6.7	8.5	196
1	2.4	4	11.2	7.7	9.6	221
2	1.5	4	6.0	5.8	5.9	136
3	3.5	6	12.7	9.1	11.1	255
4	3.5	6	12.1	8.7	10.6	244
5	3.5	6	11.9	7.7	10.0	230
6	3.5	6	11.8	8.1	10.1	232

Let's plot FUELCONSUMPTION_COMB values with respect to Engine size:

In [19]:
```
plt.scatter(new_df.FUELCONSUMPTION_COMB, new_df.CO2EMISSIONS,  color='blue')
plt.xlabel("Fuel Consumption Comb")
plt.ylabel("Emission")
plt.show()
```



Creating train and test dataset

Train/Test Split involves splitting the dataset into training and testing sets respectively, which are mutually exclusive. After which, you train with the training set and test with the testing set. This will provide a more accurate evaluation on out-of-sample accuracy because the testing dataset is not part of the dataset that have been used to train the model. Therefore, it gives us a better understanding of how well our model generalizes on new data.

We know the outcome of each data point in the testing dataset, making it great to test with! Since this data has not been used to train the model, the model has no knowledge of the outcome of these data points. So, in essence, it is truly an out-of-sample testing.

Let's split our dataset into train and test sets. Around 80% of the entire dataset will be used for training and 20% for testing.

In [32]:
```
new_df.columns
```

Out[32]:
```
Index(['ENGINESIZE', 'CYLINDERS', 'FUELCONSUMPTION_CITY',
       'FUELCONSUMPTION_HWY', 'FUELCONSUMPTION_COMB', 'CO2EMISSIONS'],
      dtype='object')
```

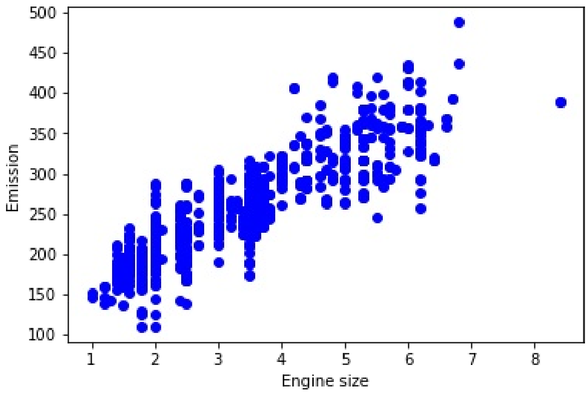
In [33]:
```
X=new_df[['ENGINESIZE','CYLINDERS','FUELCONSUMPTION_CITY','FUELCONSUMPTION_HWY']]
```

In [34]:
```
Y=new_df['CO2EMISSIONS']
```

In [35]:
```
X_train, X_test, y_train, y_test=train_test_split(X,Y,test_size=0.2)
```

Train data distribution

In [36]:
```
plt.scatter(X_train.ENGINESIZE, y_train,  color='blue')
plt.xlabel("Engine size")
plt.ylabel("Emission")
plt.show()
```



Multiple Regression Model

In reality, there are multiple variables that impact the co2emission. When more than one independent variable is present, the process is called multiple linear regression. An example of multiple linear regression is predicting co2emission using the features FUELCONSUMPTION_COMB, EngineSize and Cylinders of cars. The good thing here is that multiple linear regression model is the extension of the simple linear regression model.

Modeling

Using sklearn package to model data.

In [37]:
```
regr = LinearRegression()
regr.fit(X_train,y_train)
```

Out[37]:
```
LinearRegression()
```

In [38]:
```
# The coefficients
print ('Coefficients: ', regr.coef_)
print ('Intercept: ',regr.intercept_)
```

```
Coefficients:  [11.13674052  6.6733168   6.32315111  2.82693435]
Intercept:  69.4289214105693
```

As mentioned before, **Coefficient** and **Intercept** are the parameters of the fitted line. Given that it is a multiple linear regression model with 3 parameters and that the parameters are the intercept and coefficients of the hyperplane, sklearn can estimate them from our data. Scikit-learn uses plain Ordinary Least Squares method to solve this problem.

Ordinary Least Squares (OLS)

OLS is a method for estimating the unknown parameters in a linear regression model. OLS chooses the parameters of a linear function of a set of explanatory variables by minimizing the sum of the squares of the differences between the target dependent variable and those predicted by the linear function. In other words, it tries to minimizes the sum of squared errors (SSE) or mean squared error (MSE) between the target variable (y) and our predicted output (
$\hat{y}$
) over all samples in the dataset.

OLS can find the best parameters using of the following methods:

- Solving the model parameters analytically using closed-form equations
- Using an optimization algorithm (Gradient Descent, Stochastic Gradient Descent, Newton's Method, etc.)

Prediction

In [39]:
```
y_pred=regr.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
score=r2_score(y_test, y_pred)
# display
print("Mean absolute error : " + str(mae))
print("Mean squared error : " + str(mse))
print("r2_score : " + str(score))
```

```
Mean absolute error : 17.920163455993205
Mean squared error : 622.6497510636362
r2_score : 0.8627070777705375
```

Exercise

Try to use a multiple linear regression with the same dataset, but this time use some other columns as in X. Does it result in better accuracy?

Thank you

Author

Moazzam Ali
