Simple-Linear-Regression

December 4, 2025

1 Simple Linear Regression

Estimated time needed: **15** minutes

1.1 Objectives

After completing this lab, you will be able to:

- Use scikit-learn to implement simple linear regression
- Create, train, and test a linear regression model on real data

1.1.1 Import needed packages

For this lab, you will need to have the following packages: - NumPy - Matplotlib - Pandas - Scikit-learn

To avoid issues importing these libraries, you may execute the following cell to ensure they are available.

[1]:
```
!pip install numpy==2.2.0
!pip install pandas==2.2.3
!pip install scikit-learn==1.6.0
!pip install matplotlib==3.9.3
```

```
Collecting numpy==2.2.0
  Downloading
numpy-2.2.0-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata
(62 kB)
Downloading
numpy-2.2.0-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (16.1 MB)
                              16.1/16.1 MB
97.0 MB/s eta 0:00:00
Installing collected packages: numpy
Successfully installed numpy-2.2.0
Collecting pandas==2.2.3
  Downloading
pandas-2.2.3-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata
(89 kB)
Requirement already satisfied: numpy>=1.26.0 in /opt/conda/lib/python3.12/site-
packages (from pandas==2.2.3) (2.2.0)
```

```
Requirement already satisfied: python-dateutil>=2.8.2 in
/opt/conda/lib/python3.12/site-packages (from pandas==2.2.3) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /opt/conda/lib/python3.12/site-
packages (from pandas==2.2.3) (2024.2)
Collecting tzdata>=2022.7 (from pandas==2.2.3)
  Downloading tzdata-2025.2-py2.py3-none-any.whl.metadata (1.4 kB)
Requirement already satisfied: six>=1.5 in /opt/conda/lib/python3.12/site-
packages (from python-dateutil>=2.8.2->pandas==2.2.3) (1.17.0)
Downloading
pandas-2.2.3-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (12.7
MB)
                          12.7/12.7 MB
139.6 MB/s eta 0:00:00
Downloading tzdata-2025.2-py2.py3-none-any.whl (347 kB)
Installing collected packages: tzdata, pandas
Successfully installed pandas-2.2.3 tzdata-2025.2
Collecting scikit-learn==1.6.0
  Downloading scikit_learn-1.6.0-cp312-cp312-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (18 kB)
Requirement already satisfied: numpy>=1.19.5 in /opt/conda/lib/python3.12/site-
packages (from scikit-learn==1.6.0) (2.2.0)
Collecting scipy>=1.6.0 (from scikit-learn==1.6.0)
  Downloading
scipy-1.16.3-cp312-cp312-manylinux2014_x86_64.manylinux_2_17_x86_64.whl.metadata
(62 kB)
Collecting joblib>=1.2.0 (from scikit-learn==1.6.0)
  Downloading joblib-1.5.2-py3-none-any.whl.metadata (5.6 kB)
Collecting threadpoolctl>=3.1.0 (from scikit-learn==1.6.0)
  Downloading threadpoolctl-3.6.0-py3-none-any.whl.metadata (13 kB)
Downloading
scikit_learn-1.6.0-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl
(13.1 MB)
                          13.1/13.1 MB
125.5 MB/s eta 0:00:00
Downloading joblib-1.5.2-py3-none-any.whl (308 kB)
Downloading
scipy-1.16.3-cp312-cp312-manylinux2014_x86_64.manylinux_2_17_x86_64.whl (35.7
MB)
                          35.7/35.7 MB
50.8 MB/s eta 0:00:00:00:01
Downloading threadpoolctl-3.6.0-py3-none-any.whl (18 kB)
Installing collected packages: threadpoolctl, scipy, joblib, scikit-learn
Successfully installed joblib-1.5.2 scikit-learn-1.6.0 scipy-1.16.3
threadpoolctl-3.6.0
Collecting matplotlib==3.9.3
  Downloading matplotlib-3.9.3-cp312-cp312-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (11 kB)
Collecting contourpy>=1.0.1 (from matplotlib==3.9.3)
```

```
  Downloading contourpy-1.3.3-cp312-cp312-
manylinux_2_27_x86_64.manylinux_2_28_x86_64.whl.metadata (5.5 kB)
Collecting cycler>=0.10 (from matplotlib==3.9.3)
  Downloading cycler-0.12.1-py3-none-any.whl.metadata (3.8 kB)
Collecting fonttools>=4.22.0 (from matplotlib==3.9.3)
  Downloading fonttools-4.61.0-cp312-cp312-
manylinux1_x86_64.manylinux2014_x86_64.manylinux_2_17_x86_64.manylinux_2_5_x86_6
4.whl.metadata (113 kB)
Collecting kiwisolver>=1.3.1 (from matplotlib==3.9.3)
  Downloading kiwisolver-1.4.9-cp312-cp312-
manylinux2014_x86_64.manylinux_2_17_x86_64.whl.metadata (6.3 kB)
Requirement already satisfied: numpy>=1.23 in /opt/conda/lib/python3.12/site-
packages (from matplotlib==3.9.3) (2.2.0)
Requirement already satisfied: packaging>=20.0 in
/opt/conda/lib/python3.12/site-packages (from matplotlib==3.9.3) (24.2)
Collecting pillow>=8 (from matplotlib==3.9.3)
  Downloading pillow-12.0.0-cp312-cp312-
manylinux_2_27_x86_64.manylinux_2_28_x86_64.whl.metadata (8.8 kB)
Collecting pyparsing>=2.3.1 (from matplotlib==3.9.3)
  Downloading pyparsing-3.2.5-py3-none-any.whl.metadata (5.0 kB)
Requirement already satisfied: python-dateutil>=2.7 in
/opt/conda/lib/python3.12/site-packages (from matplotlib==3.9.3) (2.9.0.post0)
Requirement already satisfied: six>=1.5 in /opt/conda/lib/python3.12/site-
packages (from python-dateutil>=2.7->matplotlib==3.9.3) (1.17.0)
Downloading
matplotlib-3.9.3-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (8.3
MB)
                         8.3/8.3 MB
128.8 MB/s eta 0:00:00
Downloading
contourpy-1.3.3-cp312-cp312-manylinux_2_27_x86_64.manylinux_2_28_x86_64.whl (362
kB)
Downloading cycler-0.12.1-py3-none-any.whl (8.3 kB)
Downloading fonttools-4.61.0-cp312-cp312-
manylinux1_x86_64.manylinux2014_x86_64.manylinux_2_17_x86_64.manylinux_2_5_x86_6
4.whl (4.9 MB)
                         4.9/4.9 MB
142.0 MB/s eta 0:00:00
Downloading
kiwisolver-1.4.9-cp312-cp312-manylinux2014_x86_64.manylinux_2_17_x86_64.whl (1.5
MB)
                         1.5/1.5 MB
82.1 MB/s eta 0:00:00
Downloading
pillow-12.0.0-cp312-cp312-manylinux_2_27_x86_64.manylinux_2_28_x86_64.whl (7.0
MB)
                         7.0/7.0 MB
174.6 MB/s eta 0:00:00
```

```
Downloading pyparsing-3.2.5-py3-none-any.whl (113 kB)
Installing collected packages: pyparsing, pillow, kiwisolver, fonttools, cycler,
contourpy, matplotlib
Successfully installed contourpy-1.3.3 cycler-0.12.1 fonttools-4.61.0
kiwisolver-1.4.9 matplotlib-3.9.3 pillow-12.0.0 pyparsing-3.2.5
```

Now, you can import these libraries.

[2]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
%matplotlib inline
```

1.2 Load the data

The dataset you will use resides at the following URL. You can use the URL directly with the Pandas library to load the dataset.

[3]:
```
url= "https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/
↪IBMDeveloperSkillsNetwork-ML0101EN-SkillsNetwork/labs/Module%202/data/
↪FuelConsumptionCo2.csv"
```

[4]:
```
df=pd.read_csv(url)
```

[5]:
```
# verify successful load with some randomly selected records
df.sample(5)
```

[5]:
```
     MODELYEAR       MAKE               MODEL             VEHICLECLASS  \
398       2014       FORD            FLEX AWD         SUV - STANDARD
993       2014     TOYOTA         SEQUOIA 4WD         SUV - STANDARD
210       2014  CHEVROLET  EXPRESS 1500 CARGO            VAN - CARGO
66        2014       AUDI   TTS COUPE QUATTRO             SUBCOMPACT
910       2014        RAM          1500 (MDS)  PICKUP TRUCK - STANDARD

     ENGINESIZE  CYLINDERS TRANSMISSION FUELTYPE  FUELCONSUMPTION_CITY  \
398         3.5          6          AS6        X                  13.7
993         5.7          8          AS6        X                  19.0
210         4.3          6           A4        X                  17.1
66          2.0          4           A6        Z                  11.5
910         5.7          8           A8        X                  15.8

     FUELCONSUMPTION_HWY  FUELCONSUMPTION_COMB  FUELCONSUMPTION_COMB_MPG  \
398                 10.2                  12.1                        23
993                 13.9                  16.7                        17
210                 12.7                  15.1                        19
66                   8.8                  10.3                        27
910                 10.9                  13.6                        21
```

```
     CO2EMISSIONS
398           278
993           384
210           347
66            237
910           313
```

1.3 Understand the data

1.3.1 `FuelConsumption.csv`:

You will use a fuel consumption dataset, **`FuelConsumption.csv`**, which contains model-specific fuel consumption ratings and estimated carbon dioxide emissions for new light-duty vehicles for retail sale in Canada. Dataset source.

- **MODEL YEAR** e.g. 2014
- **MAKE** e.g. VOLVO
- **MODEL** e.g. S60 AWD
- **VEHICLE CLASS** e.g. COMPACT
- **ENGINE SIZE** e.g. 3.0
- **CYLINDERS** e.g 6
- **TRANSMISSION** e.g. AS6
- **FUEL TYPE** e.g. Z
- **FUEL CONSUMPTION in CITY(L/100 km)** e.g. 13.2
- **FUEL CONSUMPTION in HWY (L/100 km)** e.g. 9.5
- **FUEL CONSUMPTION COMBINED (L/100 km)** e.g. 11.5
- **FUEL CONSUMPTION COMBINED MPG (MPG)** e.g. 25
- **CO2 EMISSIONS (g/km)** e.g. 182

Your task will be to create a simple linear regression model from one of these features to predict CO2 emissions of unobserved cars based on that feature.

1.3.2 Explore the data

First, consider a statistical summary of the data.

[6]:
```
df.describe()
```

[6]:
```
       MODELYEAR   ENGINESIZE    CYLINDERS  FUELCONSUMPTION_CITY  \
count     1067.0  1067.000000  1067.000000           1067.000000
mean      2014.0     3.346298     5.794752             13.296532
std          0.0     1.415895     1.797447              4.101253
min       2014.0     1.000000     3.000000              4.600000
25%       2014.0     2.000000     4.000000             10.250000
50%       2014.0     3.400000     6.000000             12.600000
75%       2014.0     4.300000     8.000000             15.550000
max       2014.0     8.400000    12.000000             30.200000

       FUELCONSUMPTION_HWY  FUELCONSUMPTION_COMB  FUELCONSUMPTION_COMB_MPG  \
```

```
count        1067.000000           1067.000000              1067.000000
mean            9.474602             11.580881                26.441425
std             2.794510              3.485595                 7.468702
min             4.900000              4.700000                11.000000
25%             7.500000              9.000000                21.000000
50%             8.800000             10.900000                26.000000
75%            10.850000             13.350000                31.000000
max            20.500000             25.800000                60.000000

       CO2EMISSIONS
count   1067.000000
mean     256.228679
std       63.372304
min      108.000000
25%      207.000000
50%      251.000000
75%      294.000000
max      488.000000
```

From the data, we can see that most cars (about 75%) have a fuel efficiency between 11 and 31 MPG. However, one car shows a value of 60 MPG, which is much higher than the rest. This could either be a valid reading for a highly efficient or hybrid vehicle, or it might be an outlier or a data entry error.

MODELYEAR has 0 standard deviation, and thus has no interesting information content.

Select features Select a few features that might be indicative of CO2 emission to explore more.

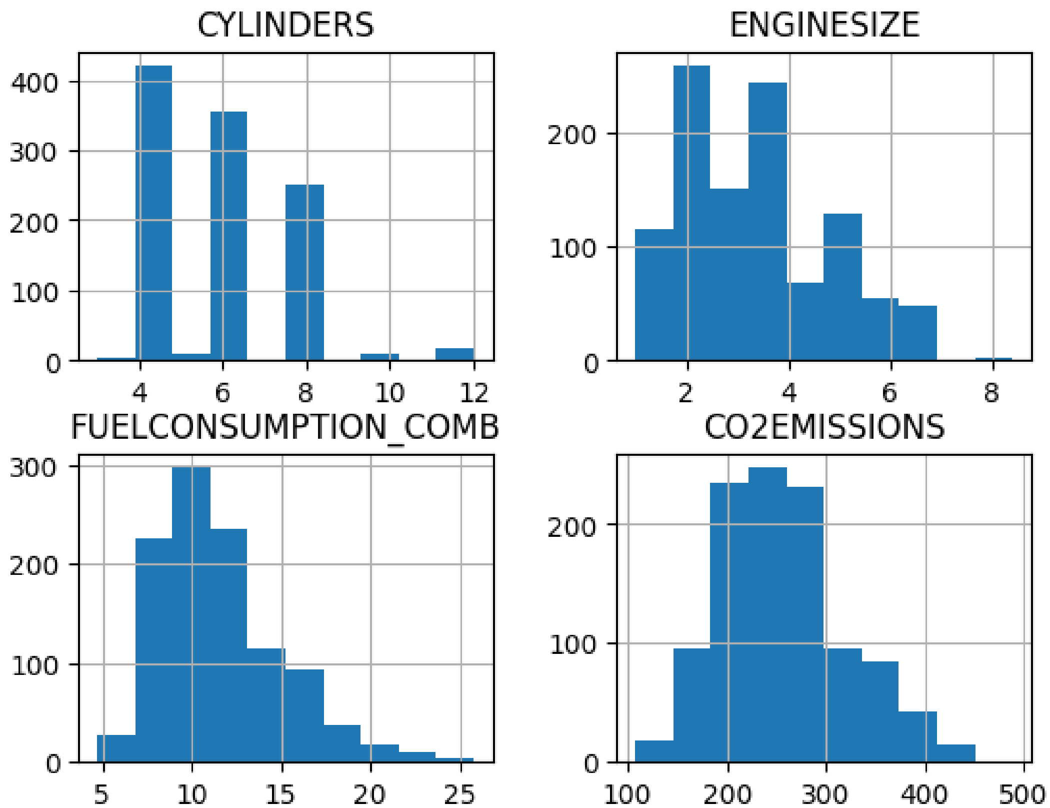
[7]:
```
cdf = df[['ENGINESIZE','CYLINDERS','FUELCONSUMPTION_COMB','CO2EMISSIONS']]
cdf.sample(9)
```

[7]:

	ENGINESIZE	CYLINDERS	FUELCONSUMPTION_COMB	CO2EMISSIONS
503	1.6	4	7.7	177
505	2.0	4	8.5	196
609	1.6	4	9.4	216
557	5.0	8	17.5	280
797	1.6	4	8.5	196
739	2.1	4	7.2	194
133	3.0	6	11.9	274
523	1.6	4	8.0	184
517	2.0	4	9.3	214

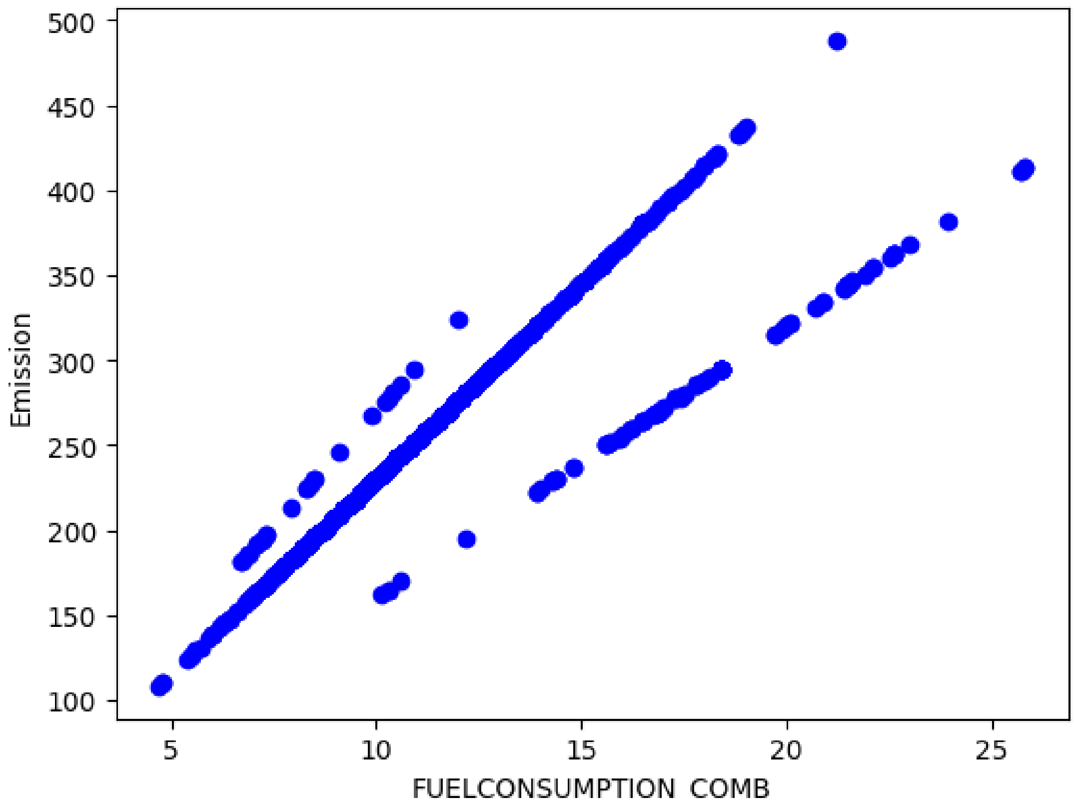
Visualize features Consider the histograms for each of these features.

[8]:
```
viz = cdf[['CYLINDERS','ENGINESIZE','FUELCONSUMPTION_COMB','CO2EMISSIONS']]
viz.hist()
plt.show()
```



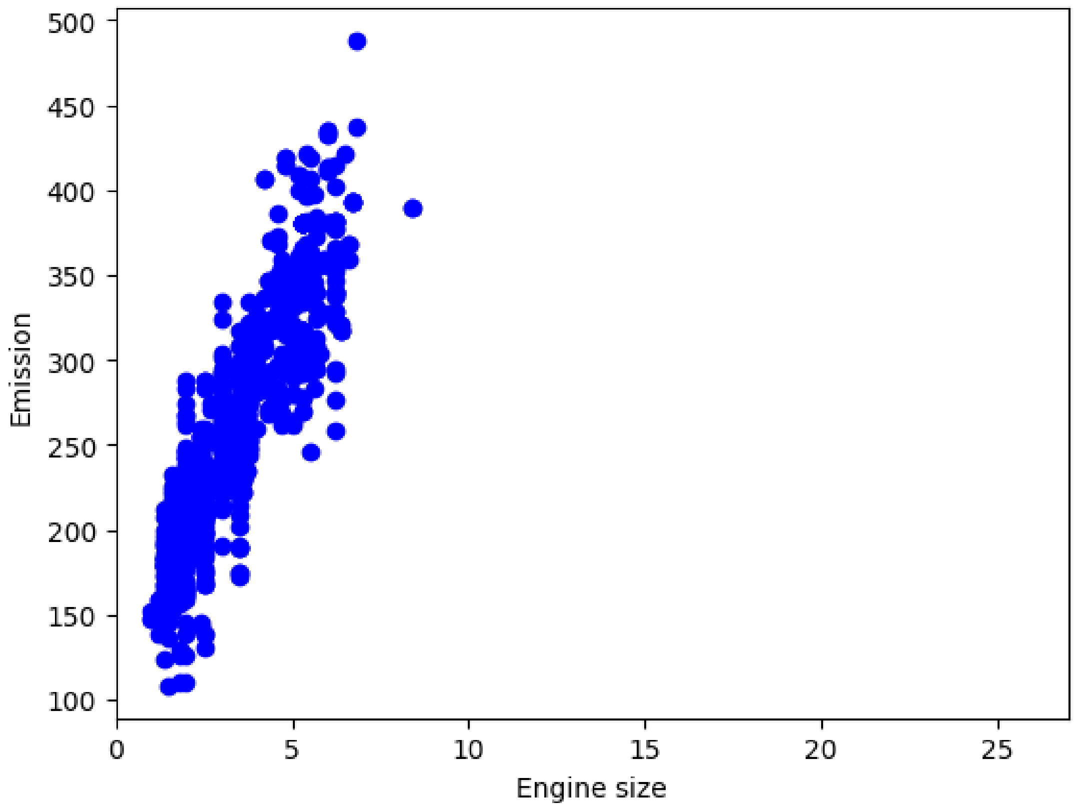
As you can see, most engines have 4, 6, or 8 cylinders, and engine sizes between 2 and 4 liters. As you might expect, combined fuel consumption and CO2 emission have very similar distributions. Go ahead and display some scatter plots of these features against the CO2 emissions, to see how linear their relationships are.

[9]:
```
plt.scatter(cdf.FUELCONSUMPTION_COMB, cdf.CO2EMISSIONS,  color='blue')
plt.xlabel("FUELCONSUMPTION_COMB")
plt.ylabel("Emission")
plt.show()
```



This is an informative result. Three car groups each have a strong linear relationship between their combined fuel consumption and their CO2 emissions. Their intercepts are similar, while they noticeably differ in their slopes.

[10]:
```
plt.scatter(cdf.ENGINESIZE, cdf.CO2EMISSIONS,  color='blue')
plt.xlabel("Engine size")
plt.ylabel("Emission")
plt.xlim(0,27)
plt.show()
```

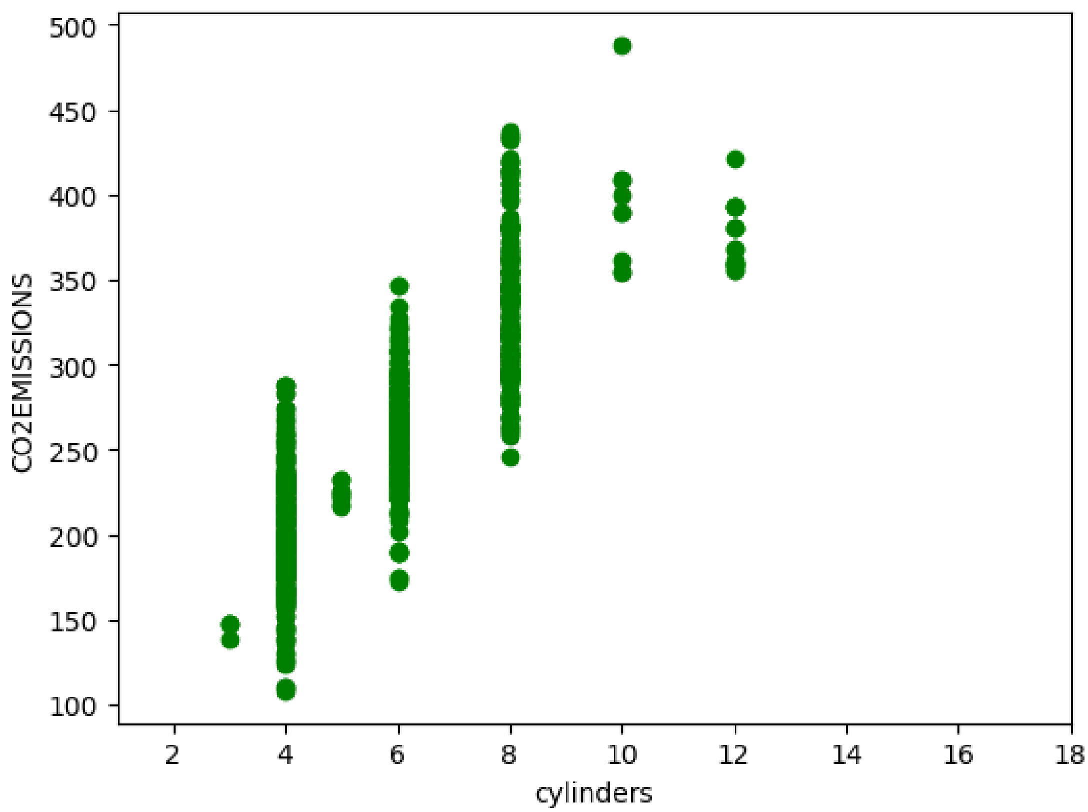


Although the relationship between engine size and CO2 emission is quite linear, you can see that their correlation is weaker than that for each of the three fuel consumption groups. Notice that the x-axis range has been expanded to make the two plots more comparable.

Practice excercise 1 Plot **CYLINDER** against CO2 Emission, to see how linear their relationship is.

[15]:
```
plt.scatter(cdf.CYLINDERS, cdf.CO2EMISSIONS, color='green')
plt.xlabel('cylinders')
plt.ylabel('CO2EMISSIONS')
plt.xlim(1,18)
plt.show
```

[15]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```



Click here for the solution

```
plt.scatter(cdf.CYLINDERS, cdf.CO2EMISSIONS,  color='blue')
plt.xlabel("CYLINDERS")
plt.ylabel("CO2 Emission")
plt.show()
```

1.3.3 Extract the input feature and labels from the dataset

Although perhaps not necessarily the ideal choice of input feature, for illustration purposes, you will use engine size to predict CO2 emission with a linear regression model.
You can begin the process by extracting the input feature and target output variables, X and y, from the dataset.

[16]:
```
X = cdf.ENGINESIZE.to_numpy()
y = cdf.CO2EMISSIONS.to_numpy()
```

Create train and test datasets Next, you will split the dataset into mutually exclusive training and testing sets. You will train a simple linear regression model on the training set and estimate its ability to generalize to unseen data by using it to make predictions on the unseen testing data.

Since the outcome of each data point is part of the testing data, you have a means of evaluating the out-of-sample accuracy of your model.

Now, you want to randomly split your data into train and test sets, using 80% of the dataset for training and reserving the remaining 20% for testing. Which fraction to use here mostly depends on the size of your data, but typical testing sizes range from 20% to 30%. The smaller your data, the larger your training set needs to be because it's easier to find spurious patterns in smaller data. The downside is that your evaluation of generalizability will have less reliability. Bigger is better when it comes to data.

[17]:
```
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)
```

The outputs are one-dimensional NumPy arrays or vectors.

[18]:
```
type(X_train), np.shape(X_train), np.shape(X_train)
```

[18]:
```
(numpy.ndarray, (853,), (853,))
```

1.3.4 Build a simple linear regression model

You'll use scikit-learn to build your model as follows. See Scikit-Learn Linear Regression documentation to learn all about the linear model predictor object.

[19]:
```
from sklearn import linear_model

# create a model object
regressor = linear_model.LinearRegression()

# train the model on the training data
# X_train is a 1-D array but sklearn models expect a 2D array as input for the␣
↪training data, with shape (n_observations, n_features).
# So we need to reshape it. We can let it infer the number of observations␣
↪using '-1'.
regressor.fit(X_train.reshape(-1, 1), y_train)

# Print the coefficients
print ('Coefficients: ', regressor.coef_[0]) # with simple linear regression␣
↪there is only one coefficient, here we extract it from the 1 by 1 array.
print ('Intercept: ',regressor.intercept_)
```

```
Coefficients:  38.992978724434074
Intercept:  126.28970217408721
```

Here, **Coefficient** and **Intercept** are the regression parameters determined by the model. They define the slope and intercept of the 'best-fit' line to the training data.

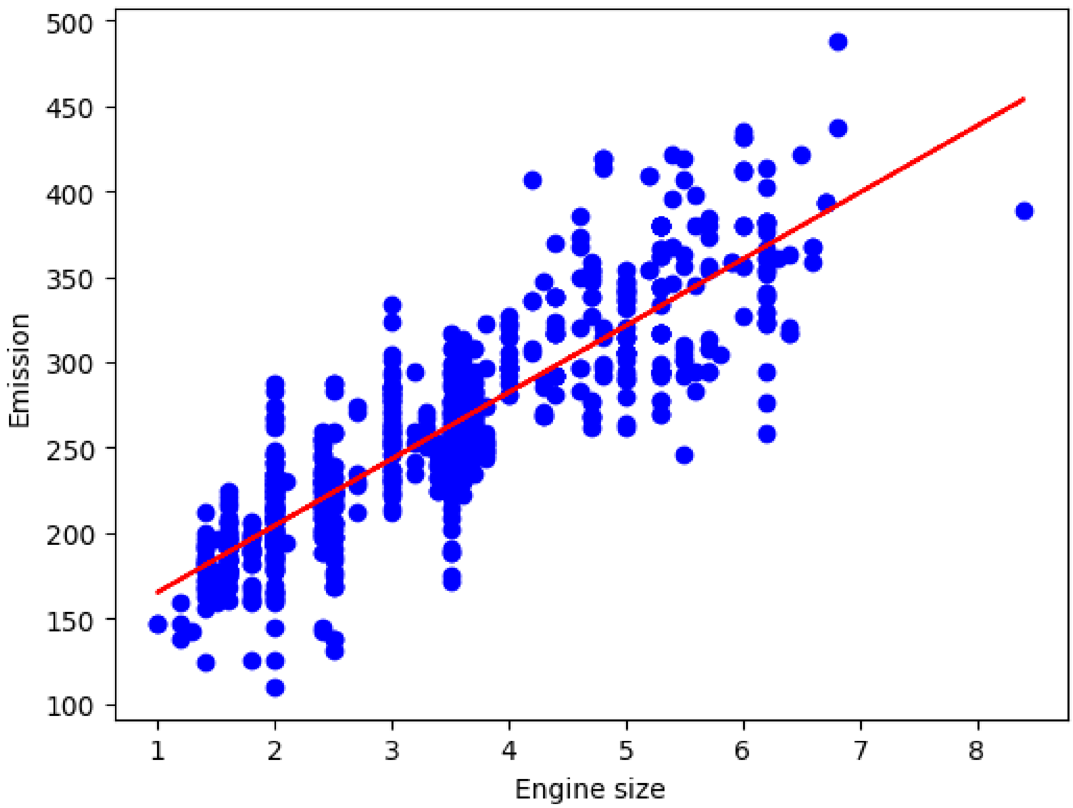
1.3.5 Visualize model outputs

You can visualize the goodness-of-fit of the model to the training data by plotting the fitted line over the data.

The regression model is the line given by y = intercept + coefficient * x.

[20]:
```
plt.scatter(X_train, y_train,  color='blue')
plt.plot(X_train, regressor.coef_ * X_train + regressor.intercept_, '-r')
plt.xlabel("Engine size")
plt.ylabel("Emission")
```

[20]: Text(0, 0.5, 'Emission')



Model evaluation You can compare the actual values and predicted values to calculate the accuracy of a regression model. Evaluation metrics play a key role in the development of a model, as they provide insight into areas that require improvement.

There are different model evaluation metrics, let's use MSE here to calculate the accuracy of our model based on the test set: * Mean Absolute Error: It is the mean of the absolute value of the errors. This is the easiest of the metrics to understand since it's just an average error.

- Mean Squared Error (MSE): MSE is the mean of the squared error. In fact, it's the metric used by the model to find the best fit line, and for that reason, it is also called the residual sum of squares.
- Root Mean Squared Error (RMSE). RMSE simply transforms the MSE into the same units as the variables being compared, which can make it easier to interpret.

- R2-Score is not an error but rather a popular metric used to estimate the performance of your regression model. It represents how close the data points are to the fitted regression line. The higher the R2-Score value, the better the model fits your data. The best possible score is 1.0 and it can be negative (because the model can be arbitrarily worse).

[21]:
```
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

# Use the predict method to make test predictions
y_pred = regressor.predict(X_test.reshape(-1,1))

# Evaluation
print("Mean absolute error: %.2f" % mean_absolute_error(y_test, y_pred))
print("Mean squared error: %.2f" % mean_squared_error(y_test, y_pred))
print("Root mean squared error: %.2f" % np.sqrt(mean_squared_error(y_test,␣
↪y_pred)))
print("R2-score: %.2f" % r2_score(y_test, y_pred))
```

```
Mean absolute error: 24.10
Mean squared error: 985.94
Root mean squared error: 31.40
R2-score: 0.76
```

1.4 Practice exercises

1. Plot the regression model result over the test data instead of the training data. Visually evaluate whether the result is good.

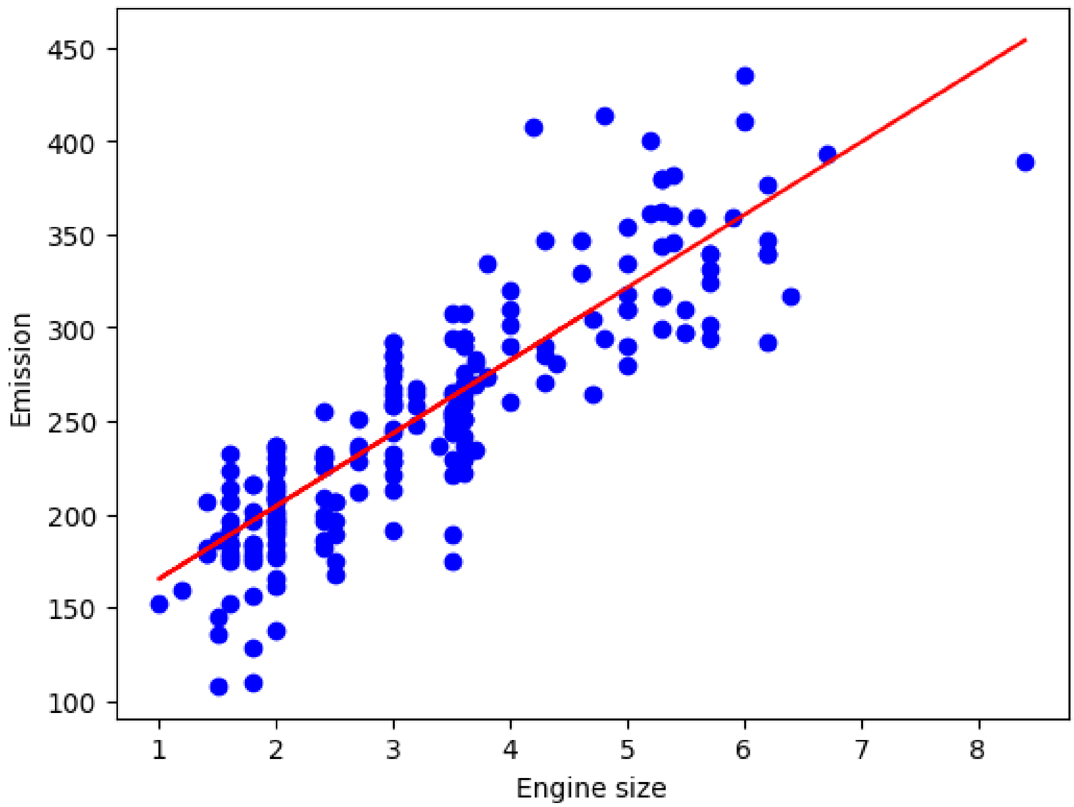
[22]:
```
plt.scatter(...)    #ADD CODE
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[22], line 1
----> 1 plt.scatter(...)    #ADD CODE

TypeError: scatter() missing 1 required positional argument: 'y'
```

[23]:
```
plt.scatter(X_test, y_test,  color='blue')
plt.plot(X_test, regressor.coef_ * X_test + regressor.intercept_, '-r')
plt.xlabel("Engine size")
plt.ylabel("Emission")
```

[23]:
```
Text(0, 0.5, 'Emission')
```



Let's see the evaluation metrics if you train a regression model using the `FUELCONSUMPTION_COMB` feature.

2. Select the fuel consumption feature from the dataframe and split the data 80%/20% into training and testing sets. Use the same random state as previously so you can make an objective comparison to the previous training result.

[ ]:
```
X = # ADD CODE

X_train, X_test, y_train, y_test = #ADD CODE
```

[24]:
```
X = cdf.FUELCONSUMPTION_COMB.to_numpy()
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.
↪2,random_state=42)
```

3. Train a linear regression model using the training data you created. Remember to transform your 1D feature into a 2D array.

[ ]:
```
regr = linear_model.# ADD CODE
```

```
#ADD CODE
```

[25]:
```
regr = linear_model.LinearRegression()
regr.fit(X_train.reshape(-1, 1), y_train)
```

[25]:
```
LinearRegression()
```

4. Use the model to make test predictions on the fuel consumption testing data.

[ ]:
```
y = # ADD CODE
```

[26]:
```
y_pred= regr.predict(X_test.reshape(-1,1))
```

5. Calculate and print the Mean Squared Error of the test predictions.

[27]:
```
print("mean_squared_error: %.2f" %mean_squared_error(y_test, y_pred))
```

```
mean_squared_error: 797.43
```

Click here for the solution

```
print("Mean squared error: %.2f" % mean_squared_error(y_test, y_pred))
```

As you might expect from your exploratory analysis, the MSE is smaller when we train using `FUELCONSUMPTION_COMB` rather than `ENGINESIZE`.

1.4.1 Congratulations! You're ready to move on to your next lesson.

1.5 Author

Jeff Grossman

Other Contributors Abhishek Gagneja

##

<!– ## Changelog | Date | Version | Changed by | Change Description | |————|——|————–|———————————————| | 2024-07-26 | 3.0 | Jeff Grossman | Update content and practice exercises | | 2020-11-03 | 2.1 | Lakshmi Holla | Change URL of the csv | | 2020-08-27 | 2.0 | Lavanya | Move lab to course repo in GitLab |

[ ]:
```
```