**Simple Linear Regression**

Estimated time needed: **15** minutes

**Objectives**

After completing this lab you will be able to:

* Use scikit-learn to implement simple Linear Regression
* Create a model, train it, test it and use the model

**Importing Needed packages**

[ ]:



**import** matplotlib.pyplot **as** plt

**import** pandas **as** pd

**import** pylab **as** pl

**import** numpy **as** np

**%**matplotlib inline

**Downloading Data**

To download the data, we will use !wget to download it from IBM Object Storage.

[ ]:



**!**wget **-**O FuelConsumption.csv https:**//**cf**-**courses**-**data.s3.us.cloud**-**object**-**storage.appdomain.cloud**/**IBMDeveloperSkillsNetwork**-**ML0101EN**-**SkillsNetwork**/**labs**/**Module**%**202**/**data**/**FuelConsumptionCo2.csv

**Did you know?** When it comes to Machine Learning, you will likely be working with large datasets. As a business, where can you host your data? IBM is offering a unique opportunity for businesses, with 10 Tb of IBM Cloud Object Storage: [Sign up now for free](http://cocl.us/ML0101EN-IBM-Offer-CC)

**Understanding the Data**

**FuelConsumption.csv:**

We have downloaded 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](http://open.canada.ca/data/en/dataset/98f1a129-f628-4ce4-b24d-6f16bf24dd64?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utm_id=NA-SkillsNetwork-Channel-SkillsNetworkCoursesIBMDeveloperSkillsNetworkML0101ENSkillsNetwork20718538-2021-01-01)

* **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**

[ ]:



df **=** pd.read\_csv("FuelConsumption.csv")

​

*# take a look at the dataset*

df.head()

​

​

**Data Exploration**

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

[ ]:



*# summarize the data*

df.describe()

Let's select some features to explore more.

[ ]:



cdf **=** df[['ENGINESIZE','CYLINDERS','FUELCONSUMPTION\_COMB','CO2EMISSIONS']]

cdf.head(9)

We can plot each of these features:

[ ]:



viz **=** cdf[['CYLINDERS','ENGINESIZE','CO2EMISSIONS','FUELCONSUMPTION\_COMB']]

viz.hist()

plt.show()

Now, let's plot each of these features against the Emission, to see how linear their relationship is:

[ ]:



plt.scatter(cdf.FUELCONSUMPTION\_COMB, cdf.CO2EMISSIONS, color**=**'blue')

plt.xlabel("FUELCONSUMPTION\_COMB")

plt.ylabel("Emission")

plt.show()

[ ]:



plt.scatter(cdf.ENGINESIZE, cdf.CO2EMISSIONS, color**=**'blue')

plt.xlabel("Engine size")

plt.ylabel("Emission")

plt.show()

**Practice**

Plot **CYLINDER** vs the Emission, to see how linear is their relationship is:

**Did you know? IBM Watson Studio lets you build and deploy an AI solution, using the best of open source and IBM software and giving your team a single environment to work in.**[**Learn more here.**](https://cocl.us/ibm_watson_studio_infobox)

[ ]:



*# write your code here*

​

​

​

Click here for the solution

plt.scatter(cdf.CYLINDERS, cdf.CO2EMISSIONS, color**=**'blue')

plt.xlabel("Cylinders")

plt.ylabel("Emission")

plt.show()

**Creating train and test dataset**

Train/Test Split involves splitting the dataset into training and testing sets that 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.

This means that 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. 80% of the entire dataset will be used for training and 20% for testing. We create a mask to select random rows using **np.random.rand()** function:

[ ]:



msk **=** np.random.rand(len(df)) **<** 0.8

train **=** cdf[msk]

test **=** cdf[**~**msk]

**Simple Regression Model**

Linear Regression fits a linear model with coefficients B = (B1, ..., Bn) to minimize the 'residual sum of squares' between the actual value y in the dataset, and the predicted value yhat using linear approximation.

**Train data distribution**

[ ]:



plt.scatter(train.ENGINESIZE, train.CO2EMISSIONS, color**=**'blue')

plt.xlabel("Engine size")

plt.ylabel("Emission")

plt.show()

**Modeling**

Using sklearn package to model data.

[ ]:



**from** sklearn **import** linear\_model

regr **=** linear\_model.LinearRegression()

train\_x **=** np.asanyarray(train[['ENGINESIZE']])

train\_y **=** np.asanyarray(train[['CO2EMISSIONS']])

regr.fit (train\_x, train\_y)

*# The coefficients*

print ('Coefficients: ', regr.coef\_)

print ('Intercept: ',regr.intercept\_)

As mentioned before, **Coefficient** and **Intercept** in the simple linear regression, are the parameters of the fit line. Given that it is a simple linear regression, with only 2 parameters, and knowing that the parameters are the intercept and slope of the line, sklearn can estimate them directly from our data. Notice that all of the data must be available to traverse and calculate the parameters.

**Plot outputs**

We can plot the fit line over the data:

[ ]:



plt.scatter(train.ENGINESIZE, train.CO2EMISSIONS, color**=**'blue')

plt.plot(train\_x, regr.coef\_[0][0]**\***train\_x **+** regr.intercept\_[0], '-r')

plt.xlabel("Engine size")

plt.ylabel("Emission")

**Evaluation**

We compare the actual values and predicted values to calculate the accuracy of a regression model. Evaluation metrics provide a key role in the development of a model, as it provides insight to areas that require improvement.

There are different model evaluation metrics, lets 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 average error.
* Mean Squared Error (MSE): Mean Squared Error (MSE) is the mean of the squared error. It’s more popular than Mean Absolute Error because the focus is geared more towards large errors. This is due to the squared term exponentially increasing larger errors in comparison to smaller ones.
* Root Mean Squared Error (RMSE).
* R-squared is not an error, but rather a popular metric to measure the performance of your regression model. It represents how close the data points are to the fitted regression line. The higher the R-squared 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).

[ ]:



**from** sklearn.metrics **import** r2\_score

​

test\_x **=** np.asanyarray(test[['ENGINESIZE']])

test\_y **=** np.asanyarray(test[['CO2EMISSIONS']])

test\_y\_ **=** regr.predict(test\_x)

​

print("Mean absolute error: %.2f" **%** np.mean(np.absolute(test\_y\_ **-** test\_y)))

print("Residual sum of squares (MSE): %.2f" **%** np.mean((test\_y\_ **-** test\_y) **\*\*** 2))

print("R2-score: %.2f" **%** r2\_score(test\_y , test\_y\_) )