

**2020/2021**

**Mini-Project**

Presented in order to obtain

**Final Grade of Mini-Project at Lebanese University – Faculty of Engineering III**

**Specialty: Telecommunication Engineering**

Prepared By:

**Abed El Kader El Shaar (5438)**

**Sara Ghamloush (5548)**

**Car Price Prediction using Machine Learning**

Under the direction of:

**Dr. Mohamad Aoude**

Submitted on **19** July 2021

Contents

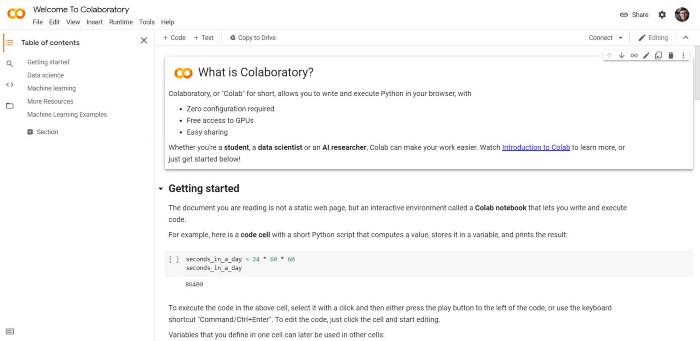
[Google Colab 3](#_Toc77533768)

[Linear Regression: 4](#_Toc77533769)

[Our Project 5](#_Toc77533770)

[Appendix 12](#_Toc77533771)

# Google Colab



Google Colab — Welcome page

Google Colab is a platform that allows you to run code directly on the cloud, this means that you can use very powerful hardware to run your code and the only requirement to do it is to have a Google account.  
In Google Colab you can only use **Python** as a programming language but this is fairly enough for the features it offers. Every line of code you write is automatically saved on your **Google Drive** storage and you can easily access your project notebook whenever you want, wherever you are.

Linear Regression

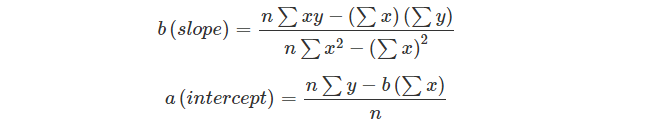
Linear regression is one of the very basic forms of machine learning where we train a model to predict the behavior of your data based on some variables. In the case of linear regression as you can see the name suggests linear that means the two variables which are on the x-axis and y-axis should be linearly correlated.

Linear regression is used to predict a quantitative response Y from the predictor variable X.

Mathematically, we can write a linear regression equation as:

what-is-linear-regression-2


Where a and b given by the formulas:



Here, x and y are two variables on the regression line.

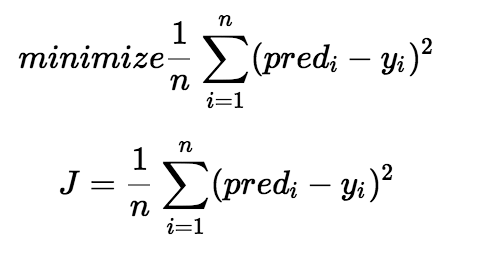
b = Slope of the line a = y-intercept of the line

x = Independent variable from dataset y = Dependent variable from dataset

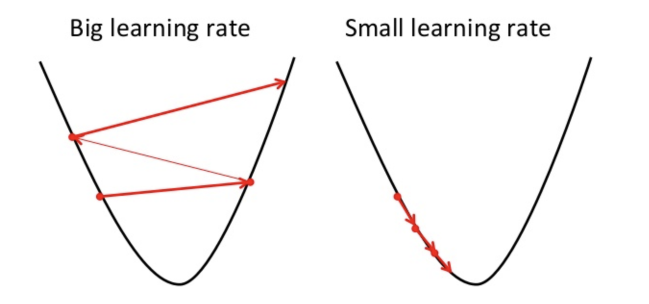
**Cost Function**

The cost function helps us to figure out the best possible values for a and b which would provide the best-fit line for the data points. Since we want the best values for a and b, we convert this search problem into a minimization problem where we would like to minimize the error between the predicted value and the actual value.

**Minimization and Cost Function**

We choose the above function to minimize. The difference between the predicted values and ground truth measures the error difference. We square the error difference and sum over all data points and divide that value by the total number of data points. This provides the average squared error over all the data points. Now, using this MSE function we are going to change the values of a and b such that the MSE value settles at the minima.

**Gradient Descent**

The next important concept needed to understand linear regression is gradient descent. Gradient descent is a method of updating a and b to reduce the cost function(MSE). The idea is that we start with some values for a and b and then we change these values iteratively to reduce the cost.

To draw an analogy, imagine a pit in the shape of U and you are standing at the topmost point in the pit and your objective is to reach the bottom of the pit. There is a catch, you can only take a discrete number of steps to reach the bottom. If you decide to take one step at a time you would eventually reach the bottom of the pit but this would take a longer time. If you choose to take longer steps each time, you would reach sooner but, there is a chance that you could overshoot the bottom of the pit and not exactly at the bottom. In the gradient descent algorithm, the number of steps you take is the learning rate. This decides on how fast the algorithm converges to the minima.

Let’s quickly see the advantage and disadvantage of linear regression algorithm:

1. Linear regression provides a powerful statistical method to find the relationship between variables. It hardly needs further tuning. However, it’s only limited to linear relationships.
2. Linear regression produces the best predictive accuracy for linear relationship whereas its little sensitive to outliers and only looks at the mean of the dependent variable

# Our Project

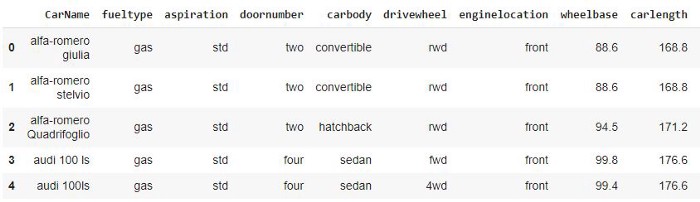
Now, we are going to show you an example of a machine learning algorithm, running on Google Colab. To make it easier we are going to use the **linear regression** algorithm with which you can **predict** linear values. In this example, we will check if there is a linear relation between car features and their price. For this purpose, we used the **CarPricePreditionDataset**, we cleaned it a bit removing some useless columns and leaving others just for data understanding. You can find this dataset and the notebook we made on my GitHub. These are the step to run your first machine learning algorithm on Google Colab:

* First of all, we need to **upload** **our** **dataset** inside the platform, and to do this we suggest using Google Drive. You can easily mount your Goog;e Drive space by clicking on the icon in the “Files” menu on the left. After the confirmation, you will have your drive folder in “content/drive/MyDrive”.
* After, you need to **import the libraries** for our project where we have, “**matplotlib**” and “**seaborn**” for the plots, “**pandas**” for the dataset management, and “**sklearn**” for the Linear Regression algorithm. As you can see, from “sklearn” we import “**train\_test\_split**” for the training set and test set split, “**LinearRegression**” for the algorithm itself, “**MinMaxScaler**” for data normalization, and “**r2\_score**” for the evaluation.

import matplotlib.pyplot as plt  
import seaborn as sns  
import numpy as np  
import pandas as pd  
from sklearn.model\_selection import train\_test\_split  
from sklearn.linear\_model import LinearRegression  
from sklearn.preprocessing import MinMaxScaler  
from sklearn.metrics import r2\_score

* The next step is to**import our dataset**. In this case, we previously removed some unnecessary columns from the table but we left something else like the cars’ name (completely useless but interesting for cars connoisseurs). The elements in this dataset are not ready yet for the linear regression algorithm, in fact, now we only need to check these data and understand them. To do this we have to use Pandas, a library that allows us to work on a dataset and show it.

df = pd.read\_csv("drive/MyDrive/Colab Notebooks/datasets/CarPrice\_Dataset\_cleaned.csv")df.head()



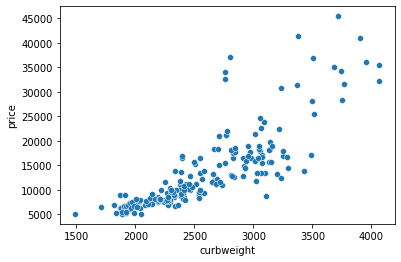
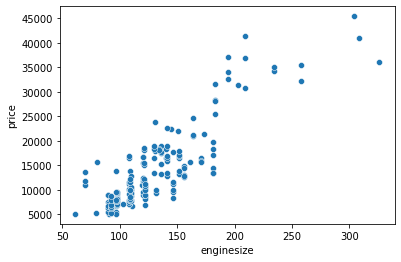
The first 5 rows of our dataset

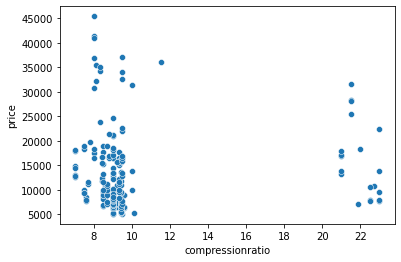
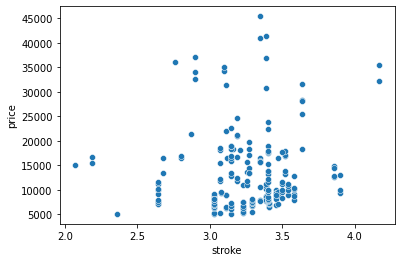
* Now that we have our wonderful dataset we have to focus on our goal: searching for correlations between cars’ features and price. As you might though we have to remove all the columns that are unusable from the algorithm. Linear Regression, like many other ML algorithms, can only use **numerical features**, so to make this dataset usable, we have to **remove all the other features**.

numerical\_feature = [feature for feature in df.columns if df[feature].dtypes!="O"]numerical\_featureOUTPUT:  
['wheelbase', 'carlength', 'carwidth', 'carheight', 'curbweight', 'enginesize', 'boreratio', 'stroke', 'compressionratio', 'horsepower', 'peakrpm', 'citympg', 'highwaympg', 'price']

* At this point, we can **visualize the remaining features** and their **correlations** with the **price** thanks to matplotlib and seaborn. Do not underestimate this step, data visualization is one of the most important steps in **data analysis** and when done right, it can save you a lot of time.  
  As you may notice from the plots there are visible linear correlations between some features and the price like for “curbweight” or “enginesize” and there also are some features with no correlation with the price like “compressionratio” and “stroke”. Thanks to data visualization we can remove the useless values from the dataset that is going to become a training set.

for feature in numerical\_feature:  
 sns.scatterplot(x = df[feature], y = df['price'])  
 plt.show()

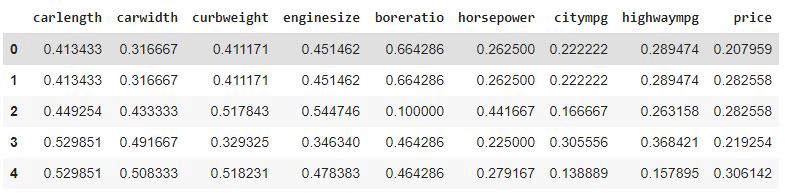




Plots of car features vs price

* At this point, we should **drop the unnecessary columns** for the Linear Regression algorithm and normalize all the numerical features. This step is very important because here we can use a unique measure for each numerical data. We could not use the multiple linear regression algorithm otherwise.

df = df.drop(['CarName', 'fueltype', 'aspiration', 'carbody', 'drivewheel', 'enginelocation', 'enginetype', 'fuelsystem', 'doornumber', 'cylindernumber'], axis=1)df = df.drop(['peakrpm', 'compressionratio', 'peakrpm', 'stroke', 'carheight', 'wheelbase'], axis=1)df['enginesize'] = np.log(df['enginesize'])scaler=MinMaxScaler()  
scaler.fit(df)  
dataset=pd.DataFrame(scaler.transform(df),columns=df.columns)  
dataset.head()



The first 5 rows of our normalized dataset

* Now we have to divide the dataset into a **training set** and **test set**. To do this we have to distinguish the value that the algorithm should guess (y) and the values that should be given to the algorithm (X). After this, we will use the “train\_test\_split” method of the sklearn library to easily create the training set and the test set.

X=df.drop(['price'], axis=1)  
y=df['price']X\_train,X\_test,y\_train,y\_test = train\_test\_split(X,y,test\_size=0.1,random\_state=42)

As you can see inside the “train\_test\_split” function we wrote 0.1, this means that the function will automatically use the **10% of the dataset as a test set**. The smaller this value, the more data we will have for the training set but on the other hand the larger the training set, the less data we will have to verify the effectiveness of the algorithm. Because of this, you always have to adjust this value according to the dimension of your dataset.

* Now we have the step that all of you have been waiting for the **model training**! Let us create the linear regression model with our training set.

lr=LinearRegression()  
lr.fit(X\_train,y\_train)OUTPUT:  
LinearRegression(copy\_X=True, fit\_intercept=True, n\_jobs=None, normalize=False)

* At this point, our model is ready and we can make some predictions using our test set. This process is automatic and after this, we will have a result set that we can compare with the test set. To define the accuracy of our model we can use the most popular metric for linear regression called “**R-squared**”, the closer the value is to 1, the more accurate the model is.

y\_predLR = lr.predict(X\_test)r2\_score(y\_test, y\_predLR)OUTPUT:  
0.8453707785571698

# Appendix

Let’s explain some functions in the code:

1. LinearRegression()

LinearRegression fits a linear model with coefficients w = (w1, …, wp) to minimize the residual sum of squares between the observed targets in the dataset, and the targets predicted by the linear approximation.

1. MinMaxScaler()

Transform features by scaling each feature to a given range.

This estimator scales and translates each feature individually such that it is in the given range on the training set, e.g. between zero and one.

The transformation is given by:

X\_std = (X - X.min(axis=0)) / (X.max(axis=0) - X.min(axis=0))

X\_scaled = X\_std \* (max - min) + min

where min, max = feature\_range.

This transformation is often used as an alternative to zero mean, unit variance scaling.

|  |  |
| --- | --- |
| [fit](https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html#sklearn.preprocessing.MinMaxScaler.fit)(X[, y]) | Compute the minimum and maximum to be used for later scaling. |
| [transform](https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html#sklearn.preprocessing.MinMaxScaler.transform)(X) | Scale features of X according to feature\_range. |

1. We use R2\_score to measure the accuracy of our model.

The r2 score varies between 0 and 100%. It is closely related to the MSE, but not the same. [Wikipedia](https://en.wikipedia.org/wiki/Coefficient_of_determination) defines r2 as

” …the proportion of the variance in the dependent variable that is predictable from the independent variable(s).”

Another definition is “(total variance explained by model) / total variance.” So if it is 100%, the two variables are perfectly correlated, i.e., with no variance at all. A low value would show a low level of correlation, meaning a regression model that is not valid, but not in all cases.