MACHINE LEARNING LAB - TUTORIAL 2

Juan Fernando Espinosa

303158

1. Pandas: Data Exploration

Import of the dataset: *import-85.names*

```
import pandas as pd
import numpy as np
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns
from google.colab import files
from google.colab import drive
drive.mount('/content/drive')
!ls "/content/drive/My Drive/Colab Notebooks/LAB/tutorial 2/imports-85.data"
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.a

Enter your authorization code:
··········
Mounted at /content/drive
'/content/drive/My Drive/Colab Notebooks/LAB/tutorial 2/imports-85.data'
```

```
column_names = ['symboling', 'normalized-losses', 'make', 'fuel-type', 'aspiration', 'num-of-doors', 'body-style', 'drive-wheels', '
missing_values = ['-','na','Nan','nan','n/a','?']
data = pd.read_csv('/content/drive/My Drive/Colab Notebooks/LAB/tutorial 2/imports-85.data', names=column_names, na_values = missing
data = pd.DataFrame(data)

data.head()
```

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	length	width	height	curb-weight	eng
0	3	NaN	alfa-romero	gas	std	two	convertible	rwd	front	88.6	168.8	64.1	48.8	2548	
1	3	NaN	alfa-romero	gas	std	two	convertible	rwd	front	88.6	168.8	64.1	48.8	2548	
2	1	NaN	alfa-romero	gas	std	two	hatchback	rwd	front	94.5	171.2	65.5	52.4	2823	
3	2	164.0	audi	gas	std	four	sedan	fwd	front	99.8	176.6	66.2	54.3	2337	
4	2	164.0	audi	gas	std	four	sedan	4wd	front	99.4	176.6	66.4	54.3	2824	

Fix missing or incongruent values in the dataset.

```
check = data.empty
print('checking missing values:',check)
print('Sum of errors:',data.isnull().sum())
```

```
checking missing values: False
Sum of errors: symboling             0
normalized-losses    41
make                  0
fuel-type             0
aspiration            0
num-of-doors          2
body-style            0
drive-wheels          0
engine-location       0
wheel-base            0
length                0
width                 0
height                0
curb-weight           0
engine-type           0
num-of-cylinders      0
engine-size           0
fuel-system           0
bore                  4
stroke                4
compression-ratio     0
horsepower            2
peak-rpm              2
city-mpg              0
highway-mpg           0
price                 4
dtype: int64
```

Replace those empty values with the mean for each column.

```
data['normalized-losses'] = data['normalized-losses'].fillna((data['normalized-losses'].mean()))
data['bore'] = data['bore'].fillna((data['bore'].mean()))
data['stroke'] = data['stroke'].fillna((data['stroke'].mean()))
data['horsepower'] = data['horsepower'].fillna((data['horsepower'].mean()))
data['peak-rpm'] = data['peak-rpm'].fillna((data['peak-rpm'].mean()))
data['price'] = data['price'].fillna((data['price'].mean()))
```

Since num-of-doors are integers, it is necessary to fill those empty fields with real information. By finding relevant information about the cars we found that most sedans have 4 doors.

```
print(data[data["num-of-doors"].isnull()])
print(data.iloc[[27,63], [2,3,4,5,6]])
```

```
    symboling  normalized-losses   make  ... city-mpg highway-mpg    price
27          1              148.0  dodge  ...       24          30   8558.0
63          0              122.0  mazda  ...       36          42  10795.0

[2 rows x 26 columns]
     make fuel-type aspiration num-of-doors body-style
27  dodge       gas      turbo          NaN      sedan
63  mazda    diesel        std          NaN      sedan
```

```
data['num-of-doors'] = data['num-of-doors'].fillna(('four'))
```

```
print(data.iloc[[27,63], [2,3,4,5,6]])
```

```
     make fuel-type aspiration num-of-doors body-style
27  dodge       gas      turbo         four      sedan
63  mazda    diesel        std         four      sedan
```

1. 1. Find the mean, median and standard deviation for each NUMERIC Column

```
numeric_data = data.select_dtypes(include=np.number)

numeric_data.head()
```

	symboling	normalized-losses	wheel-base	length	width	height	curb-weight	engine-size	bore	stroke	compression-ratio
count	205.000000	164.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	201.000000	201.000000	205.000000
mean	0.834146	122.000000	98.756585	174.049268	65.907805	53.724878	2555.565854	126.907317	3.329751	3.255423	10.142537
std	1.245307	35.442168	6.021776	12.337289	2.145204	2.443522	520.680204	41.642693	0.273539	0.316717	3.972040
min	-2.000000	65.000000	86.600000	141.100000	60.300000	47.800000	1488.000000	61.000000	2.540000	2.070000	7.000000
25%	0.000000	94.000000	94.500000	166.300000	64.100000	52.000000	2145.000000	97.000000	3.150000	3.110000	8.600000
50%	1.000000	115.000000	97.000000	173.200000	65.500000	54.100000	2414.000000	120.000000	3.310000	3.290000	9.000000
75%	2.000000	150.000000	102.400000	183.100000	66.900000	55.500000	2935.000000	141.000000	3.590000	3.410000	9.400000
max	3.000000	256.000000	120.900000	208.100000	72.300000	59.800000	4066.000000	326.000000	3.940000	4.170000	23.000000

Mean of all the columns

```
numeric_data.mean(axis=0)
```

```
symboling                0.834146
normalized-losses      122.000000
wheel-base              98.756585
length                 174.049268
width                   65.907805
height                  53.724878
curb-weight           2555.565854
engine-size            126.907317
bore                     3.329751
stroke                   3.255423
compression-ratio       10.142537
horsepower             104.256158
peak-rpm              5125.369458
city-mpg                25.219512
highway-mpg             30.751220
price                13207.129353
dtype: float64
```

Median of all the columns

```
numeric_data.median(axis=0)
```

```
symboling                1.00
normalized-losses      122.00
wheel-base              97.00
length                 173.20
width                   65.50
height                  54.10
curb-weight           2414.00
engine-size            120.00
bore                     3.31
stroke                   3.29
compression-ratio        9.00
horsepower              95.00
peak-rpm              5200.00
city-mpg                24.00
highway-mpg             30.00
price                10595.00
dtype: float64
```

Standard deviation of all the columns

```
numeric_data.std(axis=0)
```

```
symboling               1.245307
normalized-losses      31.681008
wheel-base              6.021776
length                 12.337289
width                   2.145204
height                  2.443522
curb-weight           520.680204
engine-size            41.642693
bore                    0.270844
stroke                  0.313597
compression-ratio       3.972040
horsepower             39.519211
peak-rpm              476.979093
city-mpg                6.542142
highway-mpg             6.886443
price                7868.768212
dtype: float64
```

1. 2. Group data by the field 'make'

```
makeField_data = data.groupby(['make'])
makeField_data.first()
```

make	symboling	normalized-losses	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	length	width	height	curb-weight
alfa-romero	3	122.0	gas	std	two	convertible	rwd	front	88.6	168.8	64.1	48.8	2548
audi	2	164.0	gas	std	four	sedan	fwd	front	99.8	176.6	66.2	54.3	2337
bmw	2	192.0	gas	std	two	sedan	rwd	front	101.2	176.8	64.8	54.3	2395
chevrolet	2	121.0	gas	std	two	hatchback	fwd	front	88.4	141.1	60.3	53.2	1488
dodge	1	118.0	gas	std	two	hatchback	fwd	front	93.7	157.3	63.8	50.8	1876
honda	2	137.0	gas	std	two	hatchback	fwd	front	86.6	144.6	63.9	50.8	1713
isuzu	0	122.0	gas	std	four	sedan	rwd	front	94.3	170.7	61.8	53.5	2337
jaguar	0	145.0	gas	std	four	sedan	rwd	front	113.0	199.6	69.6	52.8	4066
mazda	1	104.0	gas	std	two	hatchback	fwd	front	93.1	159.1	64.2	54.1	1890
mercedes-benz	-1	93.0	diesel	turbo	four	sedan	rwd	front	110.0	190.9	70.3	56.5	3515
mercury	1	122.0	gas	turbo	two	hatchback	rwd	front	102.7	178.4	68.0	54.8	2910
mitsubishi	2	161.0	gas	std	two	hatchback	fwd	front	93.7	157.3	64.4	50.8	1918
nissan	1	128.0	gas	std	two	sedan	fwd	front	94.5	165.3	63.8	54.5	1889
peugot	0	161.0	gas	std	four	sedan	rwd	front	107.9	186.7	68.4	56.7	3020
plymouth	1	119.0	gas	std	two	hatchback	fwd	front	93.7	157.3	63.8	50.8	1918
porsche	3	186.0	gas	std	two	hatchback	rwd	front	94.5	168.9	68.3	50.2	2778
renault	0	122.0	gas	std	four	wagon	fwd	front	96.1	181.5	66.5	55.2	2579
saab	3	150.0	gas	std	two	hatchback	fwd	front	99.1	186.6	66.5	56.1	2658
subaru	2	83.0	gas	std	two	hatchback	fwd	front	93.7	156.9	63.4	53.7	2050
toyota	1	87.0	gas	std	two	hatchback	fwd	front	95.7	158.7	63.6	54.5	1985
volkswagen	2	122.0	diesel	std	two	sedan	fwd	front	97.3	171.7	65.5	55.7	2261
volvo	-2	103.0	gas	std	four	sedan	rwd	front	104.3	188.8	67.2	56.2	2912

Find the average price , average highway-mpg and average city-mpg for each make.

```
makeField_data['price', 'highway-mpg', 'city-mpg'].mean()
```

make	price	highway-mpg	city-mpg
alfa-romero	15498.333333	26.666667	20.333333
audi	17194.589908	24.142857	18.857143
bmw	26118.750000	25.375000	19.375000
chevrolet	6007.000000	46.333333	41.000000
dodge	7875.444444	34.111111	28.000000
honda	8184.692308	35.461538	30.384615
isuzu	11061.814677	36.000000	31.000000
jaguar	34600.000000	18.333333	14.333333
mazda	10652.882353	31.941176	25.705882
mercedes-benz	33647.000000	21.000000	18.500000
mercury	16503.000000	24.000000	19.000000
mitsubishi	9239.769231	31.153846	24.923077
nissan	10415.666667	32.944444	27.000000
peugot	15489.090909	26.636364	22.454545
plymouth	7963.428571	34.142857	28.142857
porsche	27761.825871	26.000000	17.400000
renault	9595.000000	31.000000	23.000000
saab	15223.333333	27.333333	20.333333
subaru	8541.250000	30.750000	26.333333
toyota	9885.812500	32.906250	27.500000
volkswagen	10077.500000	34.916667	28.583333
volvo	18063.181818	25.818182	21.181818

Use a seaborn pairplot to visualize all int64 data types. Explain the plot what information can we take out of it

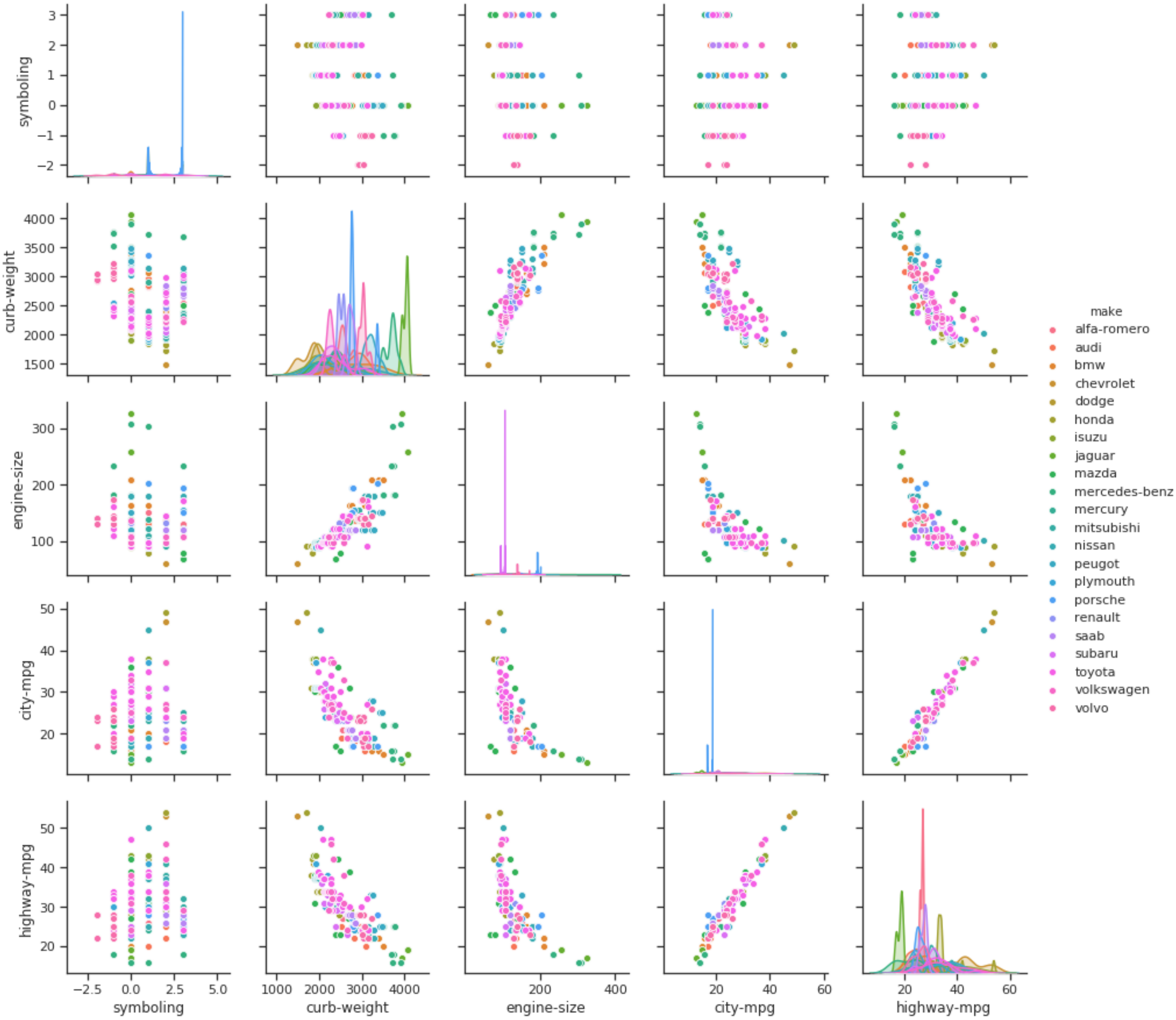
```
int64 = data[['make','symboling', 'curb-weight','engine-size','city-mpg','highway-mpg']]
int64.head()
```

	make	symboling	curb-weight	engine-size	city-mpg	highway-mpg
0	alfa-romero	3	2548	130	21	27
1	alfa-romero	3	2548	130	21	27
2	alfa-romero	1	2823	152	19	26
3	audi	2	2337	109	24	30
4	audi	2	2824	136	18	22

```
import seaborn as sns; sns.set(style="ticks", color_codes=True)
```

```
g = sns.pairplot(int64, hue='make')
```

```
/usr/local/lib/python3.6/dist-packages/statsmodels/nonparametric/kde.py:487: RuntimeWarning: invalid value encountered in true_c
  binned = fast_linbin(X, a, b, gridsize) / (delta * nobs)
/usr/local/lib/python3.6/dist-packages/statsmodels/nonparametric/kdetools.py:34: RuntimeWarning: invalid value encountered in do
  FAC1 = 2*(np.pi*bw/RANGE)**2
/usr/local/lib/python3.6/dist-packages/numpy/core/_methods.py:217: RuntimeWarning: Degrees of freedom <= 0 for slice
  keepdims=keepdims)
/usr/local/lib/python3.6/dist-packages/numpy/core/_methods.py:209: RuntimeWarning: invalid value encountered in double_scalars
  ret = ret.dtype.type(ret / rcount)
```



As we can appreciate in the charts the diagonal gives us the distribution of each variable while the scatter plots give us the relationship between 2 variables.

Observations:

> **Symboling:** As it is possible to appreciate, cars located in the neutral "risky" position tends to differ in the second variables showed, demonstrating no correlation between a risky car and the specifications of it.

> **Curb-Weight**: There is no correlation between curb_weight and symboling. The opposite happens with *engine-size*: there is a linear distribution to the right: the more curb-weight a car has the more engine-size it will have. Finally, *city-mpg* and *highway-mpg* have a negative tendency: the more curb-weight, the less city mpg and highway-mpg a car could go.

> **engine-size:** It keeps a linear distribution with the variables: a directly proportional to its curb-weight. The engine-size affects the total curb-weight. On the contrary, the more engine-size it means a better fuel optimization which decreases the city and highway mpg.

> **City and highway MPG:** both variables has a directly proportionality. If one decreases the other as well because of the same engine-size and vehicle characteristics. Moreover as mentioned before, the higher curb-weight and engine-size the less MPG the car will have.

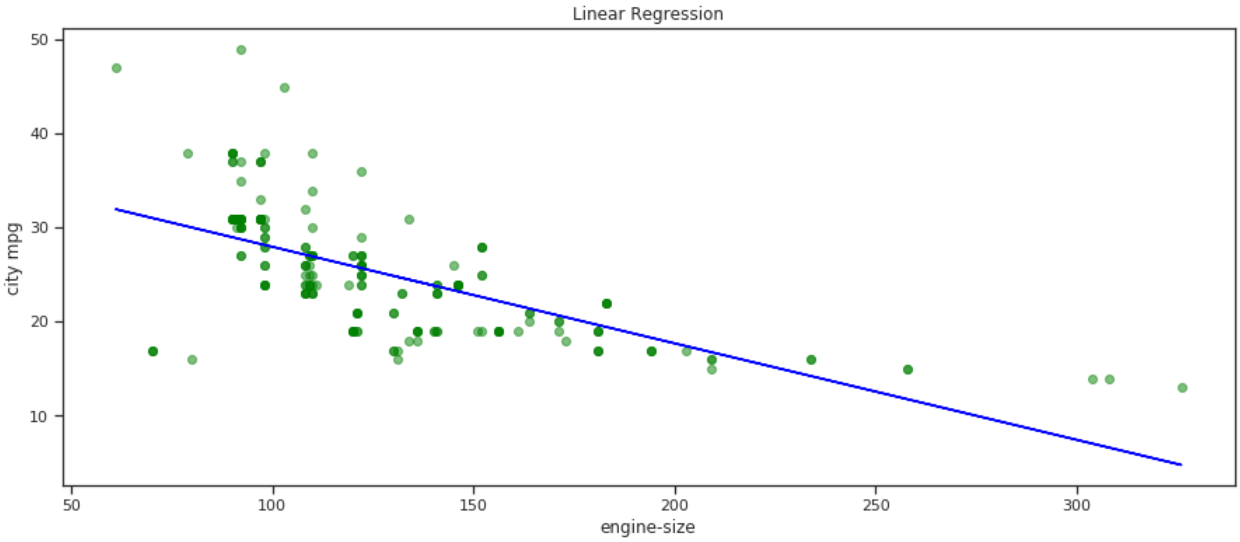
Conclusions:

- A risky or non-risky car not necessarily accomplish the best characteristics.
- The more engine-size/curb-weight the more money a person will save in fuel.

Similar to the first exercise use city-mpg as your dependant variable and engine-size as the independent value. Fit a line, use scatterplot for the data points and plot the line you predicted on top

```
Linear_regression = data[['engine-size', 'city-mpg']]
Linear_regression.head()
mean_horsepower = Linear_regression.mean(axis = 0)
mean1 = mean_horsepower['engine-size']
mean2 = mean_horsepower['city-mpg']

numerator = (Linear_regression['engine-size'] - mean1)*(Linear_regression['city-mpg'] - mean2)
totalNum = 0
for i in numerator:
  totalNum += i

denominator = (Linear_regression['engine-size'] - mean1)**2
totalDen = 0
for i in denominator:
    totalDen += i

# Calculus of the Betas
beta_1 = totalNum / totalDen
beta_0 = mean2 - (beta_1*(mean1))
print('Beta0:', beta_0)
print('Beta1:', beta_1)

# Prediction of y for all datapoints in X.
y_prediction = []
p1 = beta_1*Linear_regression['engine-size'] + beta_0
y_prediction = p1
print("y_prediction:", y_prediction)


#Plotting the graph considering the Betas found and the predictions.
fig_size = plt.rcParams["figure.figsize"]
fig_size[0] = 15
fig_size[1] = 6
plt.rcParams["figure.figsize"] = fig_size

plt.scatter(Linear_regression['engine-size'], Linear_regression['city-mpg'], color='green', alpha=0.5)
plt.title('Linear Regression')
plt.xlabel('engine-size')
plt.ylabel('city mpg')
plt.plot(Linear_regression['engine-size'],y_prediction, c ='blue')
plt.show()
```

```
Beta0: 38.25172970058016
Beta1: -0.10269082828332307
y_prediction: 0      24.901922
1      24.901922
2      22.642724
3      27.058429
4      24.285777
         ...
200    23.772323
201    23.772323
202    20.486216
203    23.361560
204    23.772323
Name: engine-size, Length: 205, dtype: float64
```



note: The graph has been made considering the variables because it does not make sense to create a plot segmented by "make" group.

Observations: It is not a good predicttion because the fit does not capture the esence of the dataset and could infere in **underfitting**. Underfitting means a lack in capturing the underlying structure of the data. Therefore, in this particular example the line overfit the nearest datapoints and does not cover several fa-away positioned datapoints. It is not a good prediction.

2. Linear Regression via Normal Equations

Reuse dataset from Excercise 1. Load it as Xdata

```
x_features = ['symboling', 'normalized-losses', 'make', 'fuel-type', 'aspiration', 'num-of-doors', 'body-style', 'drive-wheels', 'en
Xdata = data[x_features]
Ydata = data.price
```

Choose those columns, which can help you in prediction i.e. contain some useful information. You can drop irrelevant columns. Give reason for choosing or dropping any column.

First a good measure of columns which are going to have an impact on the prediction are the ones with correlation with the dependant variables because its influence on it.

```
pearsonCorr = data.corr(method='pearson')
pearsonCorr['price']
```

```
symboling            -0.082201
normalized-losses     0.133999
wheel-base            0.583168
length                0.682986
width                 0.728699
height                0.134388
curb-weight           0.820825
engine-size           0.861752
bore                  0.532300
stroke                0.082095
compression-ratio     0.070990
horsepower            0.757917
peak-rpm             -0.100854
city-mpg             -0.667449
highway-mpg          -0.690526
price                 1.000000
Name: price, dtype: float64
```

According to the relation between the price column and the independent columns (Xdata) it is optimal to consider all of them which has an influence on the price of the vehicle. Therefore, all columns with correlation near to 0 are going to be dropped.

```
Xdata = data.drop(["symboling", "normalized-losses", "height", "stroke", "compression-ratio", "peak-rpm", "price"], axis=1)
#Xdata = pd.DataFrame(data, columns=['wheel-base',  'length', 'width',  'curb-weight',  'engine-size',  'bore', 'horsepower', 'city-
Xdata = Xdata.select_dtypes(include=np.number)
Xdata.insert(0, 'Column of 1', 1)
```

Step for adding a column of ones to the Xdata dataframe.

```
Xdata = pd.DataFrame(Xdata)

Ydata = pd.DataFrame(Ydata)

Xdata_array = Xdata.rename_axis('datas').values
Ydata_array = Ydata.rename_axis('datas1').values
print(Xdata_array)
Ydata_array.round()
Xdata_array.round()
```

```
[[  1.    88.6 168.8 ... 111.    21.    27. ]
 [  1.    88.6 168.8 ... 111.    21.    27. ]
 [  1.    94.5 171.2 ... 154.    19.    26. ]
 ...
 [  1.   109.1 188.8 ... 134.    18.    23. ]
 [  1.   109.1 188.8 ... 106.    26.    27. ]
 [  1.   109.1 188.8 ... 114.    19.    25. ]]
array([[  1.,  89., 169., ..., 111.,  21.,  27.],
       [  1.,  89., 169., ..., 111.,  21.,  27.],
       [  1.,  94., 171., ..., 154.,  19.,  26.],
       ...,
       [  1., 109., 189., ..., 134.,  18.,  23.],
       [  1., 109., 189., ..., 106.,  26.,  27.],
       [  1., 109., 189., ..., 114.,  19.,  25.]])
```

Split your dataset Xdata, Ydata into Xtrain, Ytrain and Xtest, Ytest i.e. you can randomly assign 80% of the data to a Xtrain, Ytrain set and remaining 20% to a Xtest, ytest set.

```
Xtrain = Xdata.sample(frac=0.8)
Ytrain = Ydata.sample(frac=0.8)
Xtest = Xdata.drop(Xtrain.index)
Ytest = Ydata.drop(Ytrain.index)

Xtest.to_numpy().round()
Ytest.to_numpy().round()

# It will help in the calculus of the y predictions.
Xtest = np.array(Xtest)
Ytest = np.array(Ytest)
```

Implement learn-linreg-NormEq algorithm and learn a parameter vector β using Xtrain set. You have to learn a model to predict sales price of cars i.e. , ytest.

```
X = Xtrain.T
A = np.dot(X, Xtrain)
b = np.dot(X, Ytrain)
```

```
b = np.dot(X, Ytrain)
Betas_LSE = np.linalg.solve(A, b)
print('Betas',Betas_LSE)
```

```
(10, 1)
(10, 10)
Betas [[ 2.73179922e+01]
 [-9.26918472e+01]
 [ 1.49545607e+02]
 [ 5.64692851e+01]
 [-1.39579427e-01]
 [-3.66986823e+01]
 [ 1.09940947e+03]
 [-2.59444346e+00]
 [ 7.07191594e+02]
 [-7.63676613e+02]]
```

Line 6, in learn-linreg-NormEq uses SOLVE-SLE. You have to replace SOLVE-SLE

- Gaussian Elimination

```
def Gaussian_Elimination(A, b):

    n =  len(A)
    # Find the maximum value in the first column and diagonal of the matrix.
    for i in range(len(A)-1):
        # Swaping columns to put the maximum value as the first row
        max_row = abs(A[i:,i]).argmax() + i
        if max_row != i:
            A[[i,max_row]] = A[[max_row, i]]
            b[[i,max_row]] = b[[max_row, i]]
        for j in range(i+1, len(A)):
            ratio = A[j][i]/A[i][i]
            # Select the values other than the selected one for making those zeros.
            A[j][i] = ratio
            for k in range(i + 1, len(A)):
                A[j][k] = A[j][k] - ratio*A[j][k]

            # Updating items for each row.

            b[j] = b[j] - ratio*b[j]

# Return the final values.
    x = np.zeros(len(A))
    j = len(A)-1
    x[j] = b[j]/A[j,j]
    while j >= 0:
        x[j] = (b[j] - np.dot(A[j,j+1:],x[j+1:]))/A[j,j]
        j = j-1
    return x
```

```
Xtrain.shape
X = Xtrain.T
X.shape
A = np.dot(X, Xtrain)
b = np.dot(X, Ytrain)
Betas_Gaussian_Elimination = Gaussian_Elimination(A, b)
print('Betas',Betas_Gaussian_Elimination)
```

```
Betas [ 2.98765389e+02  2.32638855e-01  4.89161835e+00 -2.08056432e+01
 -9.10426212e-02  1.20732513e+01 -2.66393538e+02  1.19894049e+00
  2.14017711e+00  4.34468032e+02]
```

- QR Decomposition

```
def QR(A):
  u =[]
  e = []
  u.append(A[:,0])
  e.append(u[0]/ np.sqrt(np.sum(u[0]**2)))

  for i in range(1,len(A[0])):
    current_a = A[:,i]
    current_u = current_a
    for j in range(0,len(u)):
      current_u -= ((current_a @ e[j])*e[j])
    u.append(current_u)
    e.append(u[i]/ np.sqrt(np.sum(u[i]**2)))
```

```
  return np.array(u), np.array(e)


A1 = A.T
A2 = np.append(A2, b, axis=1)
q = e.T
r = np.dot(q,A2)
print('Q', q)
print('R', r)
```

```
Q [[ 1.00000000e+00 -1.63178187e-07 -9.26225649e-08 -2.37692545e-08
  -1.19739702e-07 -9.75254034e-08 -2.97781533e-08  5.51146129e-10
  -6.13439759e-08 -3.02725863e-09]
 [ 1.63178692e-07  1.00000000e+00  1.24371686e-06  3.18153722e-07
   1.60863649e-06  1.30870917e-06  3.98744284e-07 -7.58162752e-09
   8.22037050e-07  4.06226307e-08]
 [ 9.26224058e-08 -1.24371746e-06  1.00000000e+00 -2.28981754e-07
   5.13664466e-07  1.10279099e-07 -2.60127870e-07 -2.54691339e-08
  -2.47697349e-07 -3.34492974e-09]
 [ 2.37689618e-08 -3.18150491e-07  2.28982040e-07  1.00000000e+00
  -4.05111560e-06  2.55603802e-06 -6.62609658e-06 -2.65890914e-06
   2.71572178e-06  1.00612670e-06]
 [ 1.19741643e-07 -1.60866603e-06 -5.13654242e-07  4.05138477e-06
   9.99999999e-01  6.70484139e-06  4.03895886e-05  1.11391004e-05
   5.12185167e-06 -3.34930749e-06]
 [ 9.75179340e-08 -1.30861195e-06 -1.10314832e-07 -2.55644986e-06
  -6.70116664e-06  9.99999991e-01 -7.45151004e-05 -3.03697121e-07
  -7.20511709e-05  8.49292228e-05]
 [ 2.89671726e-08 -3.87878475e-07  2.56723662e-07  6.64962301e-06
  -4.02702147e-05  7.35968359e-05  9.99902326e-01 -4.48170467e-03
  -1.32279987e-02 -5.14784414e-04]
 [ 3.12760637e-09 -4.17141821e-08  4.10136119e-08  2.53262413e-06
  -1.16239784e-05  4.77699708e-06  5.23433219e-03  9.98337495e-01
   5.73975305e-02  6.08975085e-04]
 [ 6.17284678e-08 -8.27202345e-07  2.48872210e-07 -2.83698138e-06
  -4.77162064e-06  6.72667155e-05  1.29548737e-02 -5.73816089e-02
   9.96197265e-01  6.42691776e-02]
 [ 9.36820411e-10 -1.24986235e-08  1.25613074e-08  8.23387282e-07
  -3.65044510e-06  8.94022604e-05  3.21711263e-04 -3.08397591e-03
   6.41992824e-02 -9.97932277e-01]]
R [[ 1.63942089e+02  1.61711366e+04  2.84445310e+04  1.07909682e+04
   4.18120594e+05  2.06863081e+04  5.46414299e+02  1.70901534e+04
   4.15058768e+03  5.04127741e+03  2.19288637e+06  2.19288637e+06
   2.19288637e+06  2.19288637e+06  2.19288637e+06]
 [ 1.61776425e+04  1.60156248e+06  2.81749495e+06  1.06644095e+06
   4.16541373e+07  2.06323421e+06  5.40510213e+04  1.69846813e+06
   4.06649881e+05  4.93955758e+05  2.17397813e+08  2.17397813e+08
   2.17397813e+08  2.17397813e+08  2.17397813e+08]
 [ 2.84548929e+04  2.81738375e+06  4.96244636e+06  1.87648845e+06
   7.35091915e+07  3.64707895e+06  9.51809526e+04  3.00942738e+06
   7.11448669e+05  8.65271401e+05  3.82779517e+08  3.82779517e+08
   3.82779517e+08  3.82779517e+08  3.82779517e+08]
 [ 1.07931271e+04  1.06622521e+06  1.87618161e+06  7.11134016e+05
   2.76815115e+07  1.37233404e+06  3.60280818e+04  1.13395578e+06
   2.71830086e+05  3.30360527e+05  1.44584764e+08  1.44584764e+08
   1.44584764e+08  1.44584764e+08  1.44584764e+08]
 [ 4.18274358e+05  4.16522034e+07  7.35087399e+07  2.76859819e+07
   1.11100760e+09  5.57105654e+07  1.40907735e+06  4.60847914e+07
   1.01704875e+07  1.24093215e+07  5.65270865e+09  5.65270865e+09
   5.65270865e+09  5.65270865e+09  5.65270865e+09]
 [ 2.06912282e+04  2.06285924e+06  3.64656566e+06  1.37237348e+06
   5.57031392e+07  2.88117390e+06  7.00097124e+04  2.37144450e+06
   4.95409753e+05  6.06233956e+05  2.76869218e+08  2.76869218e+08
   2.76869218e+08  2.76869218e+08  2.76869218e+08]
 [ 3.97167482e+02  3.92800732e+04  6.91481920e+04  2.61718157e+04
   1.02102353e+06  5.04943010e+04  1.33384345e+03  4.15997399e+04
   9.95344585e+03  1.21033704e+04  5.34747548e+06  5.34747548e+06
   5.34747548e+06  5.34747548e+06  5.34747548e+06]
 [ 1.73076007e+04  1.71901466e+06  3.04543089e+06  1.14793608e+06
   4.65942498e+07  2.39635220e+06  5.86481951e+04  2.05922471e+06
   4.05146091e+05  5.00081080e+05  2.30693063e+08  2.30693063e+08
   2.30693063e+08  2.30693063e+08  2.30693063e+08]
 [ 3.48573563e+03  3.40010629e+05  5.92785126e+05  2.27417827e+05
   8.30152837e+06  3.97295795e+05  1.13822498e+04  3.13156222e+05
   9.76731226e+04  1.16530972e+05  4.61135097e+07  4.61135097e+07
   4.61135097e+07  4.61135097e+07  4.61135097e+07]
 [-4.81823419e+03 -4.71994014e+05 -8.26993403e+05 -3.15736830e+05
  -1.18714373e+07 -5.80489536e+05 -1.58984253e+04 -4.72694135e+05
  -1.28439307e+05 -1.55003575e+05 -6.36351767e+07 -6.36351767e+07
  -6.36351767e+07 -6.36351767e+07 -6.36351767e+07]]
```

As it is possible to appreciate in the results of the QR decomposition, the values in the lower triangle of the matrix does not equal 0 which is one the premises to find the values of the betas. Therefore, it is not 100% accurate to use this process.

Perform prediction y on test dataset i.e. Xtest using the set of parameters learned

```
y_prediction = np.matmul(Xtest, Betas_LSE)
print('Betas LSE', y_prediction)
y_prediction.shape

y_prediction_gaussian_elimination = np.matmul(Xtest, Betas_Gaussian_Elimination)
print('Betas Gaussian Elimination', y_prediction_gaussian_elimination)
```

```
Betas LSE [[13529.13348798]
 [ 8734.69080801]
 [ 7717.40855712]
 [16619.74832698]
 [12846.36953893]
 [14492.10079065]
 [15898.17100419]
 [11586.22280844]
 [12011.46346379]
 [14155.58041561]
 [12072.84645395]
 [12101.66801292]
 [12647.68208984]
 [10809.7401909 ]
 [15101.35341624]
 [14213.05262174]
 [13742.23699185]
 [13119.95257706]
 [11976.49586308]
 [13448.95940849]
 [13423.59643659]
 [14543.6931665 ]
 [13767.75199965]
 [12133.50560523]
 [16404.87614408]
 [12122.61468651]
 [13441.05409954]
 [13791.71261099]
 [13818.84636538]
 [13770.2239608 ]
 [14624.76060843]
 [13172.96686059]
 [13035.37036651]
 [11490.77351931]
 [11494.23210639]
 [13024.47587548]
 [12820.27375927]
 [10667.79730064]
 [12406.06846047]
 [13452.2638417 ]
 [14815.61392859]]
Betas Gaussian Elimination [12133.26409402 11347.95968998  9140.89243411 13040.57903769
  9992.4153525  16544.8816706  13146.84412723 14600.17146045
 16558.92301239  9672.35127475 14219.96310193 14217.6870364
 11766.08954115  9344.82952006  9406.32834794 16544.89133172
 14230.98515207 16178.84451048 16196.80462738 15109.16827145
 15111.17120911 10594.02450262 10645.3320718  16583.51003787
 12034.66402913 12459.68739616 16985.10281352 16159.44430092
 20612.91532389 14848.93753074 13597.63920068 14240.44198952
 11423.58246914 20088.99675686 20088.723629   14994.93291009
 14133.82902651 13686.24705166 12629.72705421  9965.28928211
 11328.22211494]
```

As we have different random distribution, the results are different for both predictions.

Final step is to find how close these two models are to the original values.

Plot the residual

```
Residual_LSE = []
for i in range(0, len(y_prediction)):
  a = abs(Ytest[i] - y_prediction[i])
  Residual_LSE.append(a)
print('Residual LSE',Residual_LSE)

Residual_Gaussian_Elimination = []
for i in range(0, len(Ytest)):
  b = abs(Ytest[i] - y_prediction_gaussian_elimination[i])
  Residual_Gaussian_Elimination.append(b)
print('Residual_Gaussian_Elimination',Residual_Gaussian_Elimination)
```

```
Residual LSE [array([4180.86651202]), array([15140.30919199]), array([23042.59144288]), array([24695.25167302]), array([6551.36
Residual_Gaussian_Elimination [array([5576.73590598]), array([12527.04031002]), array([21619.10756589]), array([28274.42096231])
```

- Find the average residual

```
Average_residual_LSE = np.mean(Residual_LSE)
print('Average LSE',Average_residual_LSE)

Average_residual_Gaussian_Elimination = np.mean(Residual_Gaussian_Elimination)
print('Average Gaussian Elimination',Average_residual_Gaussian_Elimination)
```

```
Average LSE 5453.782643818496
Average Gaussian Elimination 6175.845817294606
```

- Find the Root Mean Square Error

```
RMSE = np.sqrt(np.square(np.subtract(Ytest,y_prediction))).mean()
print('RMSE LSE', RMSE)

RMSE_gaussian_elimination = np.sqrt(np.square(np.subtract(Ytest,y_prediction_gaussian_elimination))).mean()
print('RMSE Gaussian Elimination',RMSE_gaussian_elimination)
```

```
RMSE LSE 5453.782643818496
RMSE Gaussian Elimination 5946.823117753272
```

Bibliography

- Gaussian elimination using NumPy. Retrieved from https://gist.github.com/num3ric/1357315.
- Thoma, M. Solving linear equations with Gaussian elimination. Retrieved from: https://gist.github.com/num3ric/1357315