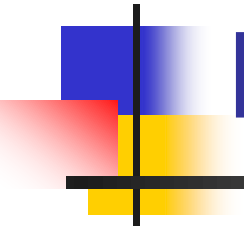
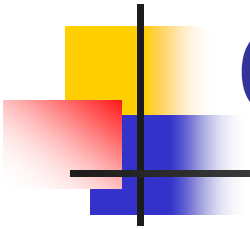
Deep Learning Using TensorFlow



Dr. Ash Pahwa

Lesson 5: Linear Regression in TensorFlow

Lesson 5.2: Linear Regression in TensorFlow Multiple Variables

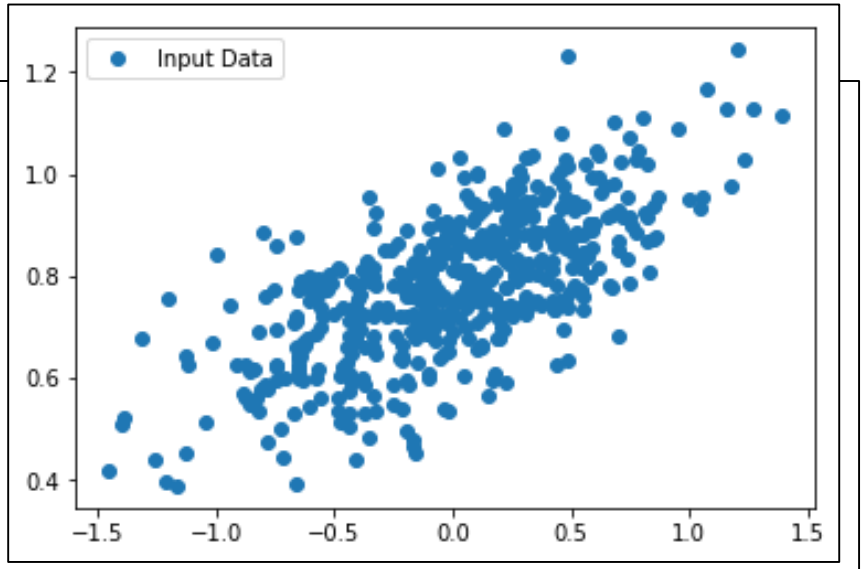


Outline

- 2 Variable Regression
- Multi Variable Regression
  - Data Normalization
  - Dataset
  - Multi Variable Regression in R
  - Python: Un-Normalized Data
    - Linear Regression in Scikit-Learn
    - Linear Regression in TensorFlow (No Solution)
  - Python: Normalized Data
    - Linear Regression in Scikit-Learn
    - Linear Regression in TensorFlow

2 Variable Regression Generate the Dataset

```
##########################################
# Generate data
#
number_of_points = 500
x_point = []
y_point = []
m = 0.22
c = 0.78
for i in range(number_of_points):
    x = np.random.normal (0.0, 0.5)
    y = m*x + c + np.random.normal(0.0,0.1)
    x_point.append([x])
    y_point.append([y])



plt.plot(x_point, y_point, 'o', label='Input Data')
plt.legend()
Out[22]: <matplotlib.legend.Legend at 0x1a8a68333c8>
```



2 Variable Regression
Linear Regression in Scikit-Learn

```
##################################################
# Linear Regression Using SKLearn function
#

linreg = linear_model.LinearRegression()

linreg.fit(x_point, y_point)
Out[27]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)

print (linreg.intercept_)
[ 0.78390055]

print (linreg.coef_)
[[ 0.20174445]]
```

Scikit-Learn: Answer

```
print (linreg.intercept_)
[ 0.78390055]

print (linreg.coef_)
[[ 0.20174445]]
```

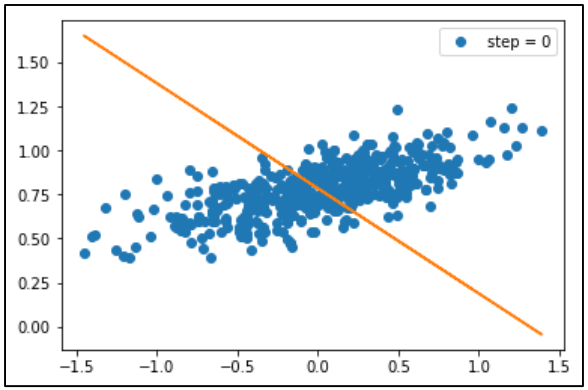
Regression Equation

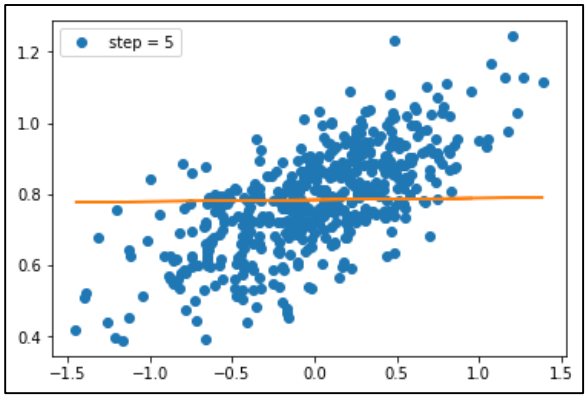
$y = 0.2017x + 0.7839$

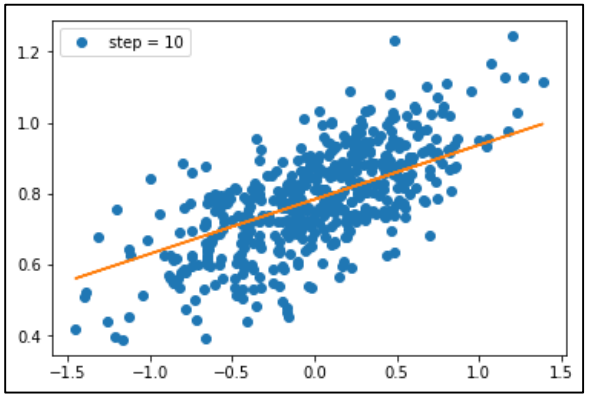
2 Variable Regression Result

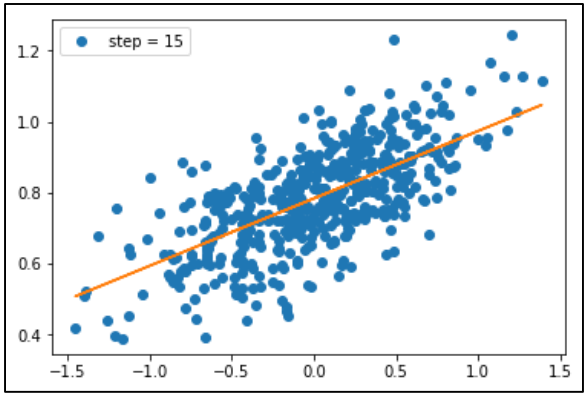
Scikit-Learn: Answer

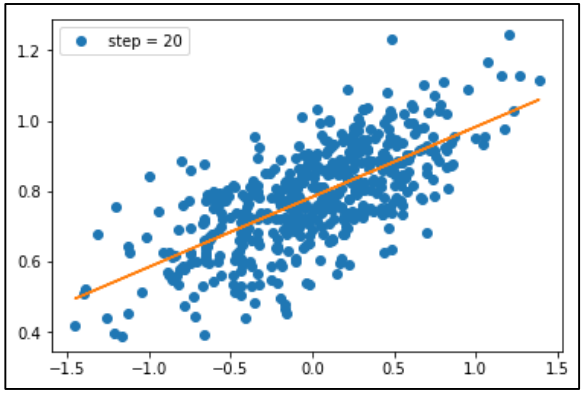
```
print (linreg.intercept_)
[ 0.78390055]

print (linreg.coef_)
[[ 0.20174445]]
```









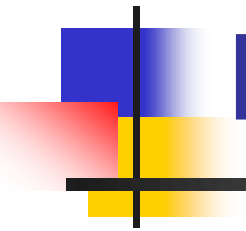


TensorFlow Answer

```
Slope =   [ 0.20152435]
Intercept =   [ 0.78390092]
```

Regression Equation

$$y = 0.2015x + 0.7839$$



Example#1

Multi Variable Regression

AutoMpg Dataset

	A	B	C	D	E	F	G	H	I
1	carID	mpg	cylinders	displacement	horsepower	weight	acceleration	origin	age
2	1	18	8	307	130	3504	12	1	23
3	2	15	8	350	165	3693	11.5	1	23
4	3	18	8	318	150	3436	11	1	23
5	4	16	8	304	150	3433	12	1	23
6	5	17	8	302	140	3449	10.5	1	23
7	6	15	8	429	198	4341	10	1	23
8	7	14	8	454	220	4354	9	1	23
9	8	14	8	440	215	4312	8.5	1	23
10	9	14	8	455	225	4425	10	1	23
11	10	15	8	390	190	3850	8.5	1	23
12	11	15	8	383	170	3563	10	1	23
13	12	14	8	340	160	3609	8	1	23
14	13	15	8	400	150	3761	9.5	1	23
15	14	14	8	455	225	3086	10	1	23
16	15	24	4	113	95	2372	15	3	23

Data Normalization: Standardization & Scaling

Data Standardization and Scaling

- Standardization Data Variation
  - -3 to +3

$$z = \frac{\text{Data Value - Mean}}{\text{Standard Deviation}} = \frac{y - \mu}{\sigma}$$

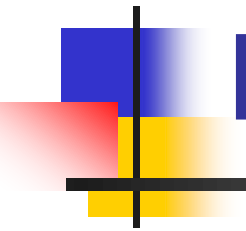
- Scaling Data Variation
  - 0 to 1

$$y_i^j = \frac{x_i^j - min_j}{max_j - min_j}$$

Example

```
> normalize = function(x) {
+   return(  (x-min(x))/(max(x)-min(x)))}
> data = c(124,3,311,341,298,136,23,75,5,51,822,364,663,444,999)
> (standard.data = scale(data))
              [,1]
 [1,] -0.603086904
 [2,] -0.994156118
 [3,]  0.001292791
 [4,]  0.098252100
 [5,] -0.040722910
 [6,] -0.564303180
 [7,] -0.929516578
 [8,] -0.761453776
 [9,] -0.987692164
[10,] -0.839021223
[11,]  1.652833026
[12,]  0.172587571
[13,]  1.138948687
[14,]  0.431145729
[15,]  2.224892951
attr(,"scaled:center")
[1] 310.6
attr(,"scaled:scale")
[1] 309.4081
> (normalized.data = normalize(data))
 [1] 0.121485944 0.000000000 0.309236948 0.339357430 0.296184739
0.133534137 0.020080321 0.072289157 0.002008032 0.048192771 0.822289157
[12] 0.362449799 0.662650602 0.442771084 1.000000000
>
```

	A	B	C	D	E	F
1		#	Data	Standardization		Scaling
2		1	124	-0.60		0.12
3		2	3	-0.99		0.00
4		3	311	0.00		0.31
5		4	341	0.10		0.34
6		5	298	-0.04		0.30
7		6	136	-0.56		0.13
8		7	23	-0.93		0.02
9		8	75	-0.76		0.07
10		9	5	-0.99		0.00
11		10	51	-0.84		0.05
12		11	822	1.65		0.82
13		12	364	0.17		0.36
14		13	663	1.14		0.66
15		14	444	0.43		0.44
16		15	999	2.22		1
17						
18		Mean	310.60		Minimum	3
19		StdDev	309.41		Maximum	999
20						



Multi Variable Regression in R

Multi Variable Regression in R

```
> data <- read.csv("autompg.csv")
> ###############################################
> y = data$mpg
> x1 = data$cylinders
> x2 = data$displacement
> x3 = data$horsepower
> x4 = data$weight
> x5 = data$acceleration
> x6 = data$origin
> x7 = data$age
> result <- lm(y ~ x3 + x4)
> summary(result)
Call:
lm(formula = y ~ x3 + x4)
Residuals:
     Min       1Q   Median       3Q      Max
-11.1154  -2.7785  -0.3008   2.2864  16.6244

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept) 45.6540784  0.7943519  57.473  < 2e-16 ***
x3          -0.0471671  0.0111012  -4.249 2.69e-05 ***
x4          -0.0057880  0.0005031 -11.506  < 2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 4.246 on 389 degrees of freedom
Multiple R-squared:  0.7051,  Adjusted R-squared:  0.7036
F-statistic: 465.1 on 2 and 389 DF,  p-value: < 2.2e-16
```

Regression Equation: R: Un-Normalized Data

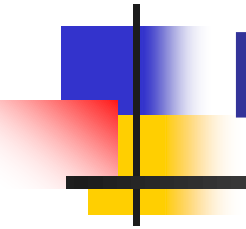
$$mpg = 45.6541 - 0.0472 * horsepower - 0.0058 * weight$$

Multi Variable Reg: Normalized Data

```
> ################################################
> normalize = function(x) {
+   return(  (x-min(x))/(max(x)-min(x)))}
> n.x3 = normalize(x3)
> n.x4 = normalize(x4)
> n.y = normalize(y)
> result <- lm(n.y ~ n.x3 + n.x4)
> summary(result)

Call:
lm(formula = n.y ~ n.x3 + n.x4)

Residuals:
     Min       1Q   Median       3Q      Max
-0.29251 -0.07312 -0.00791  0.06017  0.43749

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  0.66180    0.01079  61.335  < 2e-16 ***
n.x3        -0.22839    0.05375  -4.249 2.69e-05 ***
n.x4        -0.53722    0.04669 -11.506  < 2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.1117 on 389 degrees of freedom
Multiple R-squared:  0.7051,  Adjusted R-squared:  0.7036
F-statistic: 465.1 on 2 and 389 DF,  p-value: < 2.2e-16
```

Regression Equation: R: Normalized Data

$$mpg = 0.6618 - 0.2284 * horsepower - 0.5372 * weight$$



Python

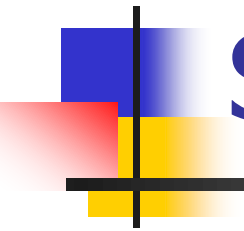
Un-Normalized Data

Dataset

```
##############################################
# 1. Load the Libraries
# 2. Read the Dataset
#
import tensorflow as tf
import numpy as np
import pandas as pd
data = pd.read_csv('autompg.csv')
data.head()
Out[12]:
   carID  mpg  cylinders  displacement  horsepower  weight  acceleration  \
0      1   18          8           307         130    3504          12.0
1      2   15          8           350         165    3693          11.5
2      3   18          8           318         150    3436          11.0
3      4   16          8           304         150    3433          12.0
4      5   17          8           302         140    3449          10.5

   origin  age
0       1   23
1       1   23
2       1   23
3       1   23
4       1   23
```

$$mpg = \beta_0 + \beta_1 * horsepower + \beta_2 * weight$$

```
var_1 = data['horsepower']
var_1 = var_1.tolist()

var_2 = data['weight']
var_2 = var_2.tolist()

pred_vars_skl = data[['horsepower','weight']]

target = data['mpg']
target_skl = data['mpg']
target = target.tolist()
```



Linear Regression in Scikit-Learn

Implementation in Scikit-Learn

```
##############################################
# 3. Compute the regression equation using Scikit-Learn
#

from sklearn import linear_model


linreg = linear_model.LinearRegression()
linreg.fit(pred_vars_skl, target)
Out[26]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)

print (linreg.intercept_)
45.6540784147

print (linreg.coef_)
[-0.04716711 -0.00578798]
```

Regression Equation: R: Un-Normalized Data

$$mpg = 45.6541 - 0.0472 * horsepower - 0.0058 * weight$$

Regression Equation: Scikit-Learn: Un-Normalized Data

$$mpg = 45.6541 - 0.0472 * horsepower - 0.0058 * weight$$

Linear Regression in TensorFlow

Define the Hyper Parameters

```
##############################################
# 4. Define the Hyper Parameters
#

learning_rate = 0.0000001

epochs = 5000000
```

Create Variables Build the Model

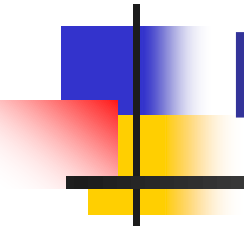
```
##########################################
# 5. Create Variables X,Y,W,B
# Build the model
#

x1 = tf.placeholder(dtype=np.float32)

x2 = tf.placeholder(dtype=np.float32)

y = tf.placeholder(dtype=np.float32)

################################

W1 = tf.Variable(np.random.randn(), dtype=np.float32, name="weight1")

W2 = tf.Variable(np.random.randn(), dtype=np.float32, name="weight2")

B = tf.Variable(np.random.randn(), dtype=np.float32, name="bias")

################################
computed_y = W1*x1 + W2*x2 + B
```

Define the 'cost' and 'optimization' functions

```
##############################################
# 6. Define the 'cost' and 'optimization' Functions
# Initialize the variables
#

cost = tf.reduce_sum(tf.square(computed_y - y))

optimizer = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost)

init = tf.global_variables_initializer()
```

Run the Iteration Loop

```
with tf.Session() as sess:
    sess.run(init)
    for epoch in range(epochs):
        sess.run(optimizer,{x1:var_1, x2:var_2, y:target})

        if not epoch % 10000 :
            c = sess.run(cost,{x1:var_1, x2:var_2, y:target})
            w1 = sess.run(W1)
            w2= sess.run(W2)
            b = sess.run(B)
            #print(f'epoch={epoch: 4d} c={c: 4f} w1={w1: 4f}  w2={w2: 4f} b={b: 4f}')
            print(epoch, c, w1, w2, b)

    weight1 = sess.run(W1)
    weight2 = sess.run(W2)
    bias = sess.run(B)
    print(f'Slope1={weight1: 4f} Slope2={weight2: 4f} Intercept={bias: 4f}')


0 1.06415e+15 -17.772 -531.53 0.928732
10000 nan nan nan nan
20000 nan nan nan nan
30000 nan nan nan nan
40000 nan nan nan nan
```



Python

Normalized Data

Dataset

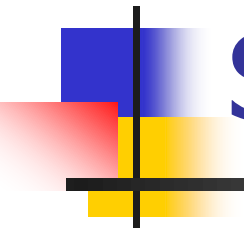
```
##############################################
# Linear Regression auto.csv
#
##############################################
# 1. Load the Libraries
# 2. Read the Dataset
#
import pandas as pd
import tensorflow as tf
import numpy as np
from sklearn import preprocessing

data = pd.read_csv('autompg.csv')

data.head()
Out[13]:
   carID  mpg  cylinders  displacement  horsepower  weight  acceleration  \
0      1   18          8           307         130    3504          12.0
1      2   15          8           350         165    3693          11.5
2      3   18          8           318         150    3436          11.0
3      4   16          8           304         150    3433          12.0
4      5   17          8           302         140    3449          10.5

   origin  age
0       1   23
1       1   23
2       1   23
3       1   23
4       1   23
```

$$mpg = \beta_0 + \beta_1 * horsepower + \beta_2 * weight$$

Normalize the Data

```
var_1 = preprocessing.minmax_scale(data['horsepower'])
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py:475:
DataConversionWarning: Data with input dtype int64 was converted to float64.
  warnings.warn(msg, DataConversionWarning)

var_1 = var_1.tolist()

var_2 = preprocessing.minmax_scale(data['weight'])
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py:475:
DataConversionWarning: Data with input dtype int64 was converted to float64.
  warnings.warn(msg, DataConversionWarning)

var_2 = var_2.tolist()

pred_vars_skl = preprocessing.minmax_scale(data[['horsepower','weight']])

target_skl = preprocessing.minmax_scale(data['mpg'])
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py:475:
DataConversionWarning: Data with input dtype int64 was converted to float64.
  warnings.warn(msg, DataConversionWarning)

target_skl = target_skl.tolist()

target = preprocessing.minmax_scale(data['mpg'])
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py:475:
DataConversionWarning: Data with input dtype int64 was converted to float64.
  warnings.warn(msg, DataConversionWarning)

target = target.tolist()
```

$$mpg = \beta_0 + \beta_1 * horsepower + \beta_2 * weight$$



Linear Regression in Scikit-Learn

Implementation in Scikit-Learn

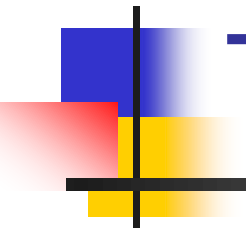
```
##############################################
# 3. Compute the regression equation using Scikit-Learn
#

from sklearn import linear_model

linreg = linear_model.LinearRegression()

linreg.fit(pred_vars_skl, target_skl)
Out[28]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)

print (linreg.intercept_)
0.661799255403

print (linreg.coef_)
[-0.22838813 -0.53721638]
```

Regression Equation: R: Normalized Data

$$mpg = 0.6618 - 0.2284 * horsepower - 0.5372 * weight$$

Regression Equation: Scikit-Learn: Normalized Data

$$mpg = 0.6618 - 0.2284 * horsepower - 0.5372 * weight$$



Linear Regression in TensorFlow

Define the Hyper Parameters

```
##############################################
# 4. Define the Hyper Parameters
#

learning_rate = 0.0000001

epochs = 5000000
```

Create Variables Build the Model

```
##########################################
# 5. Create Variables X,Y,W,B
# Build the model
#

x1 = tf.placeholder(dtype=np.float32)
x2 = tf.placeholder(dtype=np.float32)
y = tf.placeholder(dtype=np.float32)

########################
W1 = tf.Variable([0], dtype=np.float32, name="weight1")
W2 = tf.Variable([0], dtype=np.float32, name="weight2")
B  = tf.Variable([0], dtype=np.float32, name="bias")
########################

computed_y = W1*x1 + W2*x2 + B
```

Define the 'cost' and 'optimization' functions

```
##############################################
# 6. Define the 'cost' and 'optimization' Functions
# Initialize the variables
#

cost = tf.reduce_sum(tf.square(computed_y - y))

optimizer = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost)

init = tf.global_variables_initializer()
```

Run the Iteration Loop

```
with tf.Session() as sess:
    sess.run(init)
    for epoch in range(epochs):
        sess.run(optimizer,{x1:var_1, x2:var_2, y:target})

        if not epoch % 100000 :
            c = sess.run(cost,{x1:var_1, x2:var_2, y:target})
            w1 = sess.run(W1)
            w2= sess.run(W2)
            b = sess.run(B)
            print(epoch, c, w1, w2, b)

    weight1 = sess.run(W1)
    weight2 = sess.run(W2)
    bias = sess.run(B)
    print(weight1, weight2, bias)
```

Regression Equation: TensorFlow: Normalized Data

$$mpg = 0.6394 - 0.3084 * horsepower - 0.4183 * weight$$

```
0 73.4795 [  6.88996124e-06] [  8.35379069e-06] [  2.99000003e-05]
100000 10.8068 [-0.10149054] [-0.12765189] [ 0.44707638]
200000 6.75203 [-0.20843855] [-0.26567879] [ 0.54149044]
300000 5.48872 [-0.26535648] [-0.34467709] [ 0.59411216]
400000 5.09222 [-0.29446131] [-0.39071572] [ 0.62335241]
[-0.30839255] [-0.4183189] [ 0.63943666]
```

Final Result

Regression Equation: R: Normalized Data

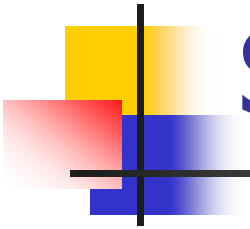
$$mpg = 0.6618 - 0.2284 * horsepower - 0.5372 * weight$$

Regression Equation: Scikit-Learn: Normalized Data

$$mpg = 0.6618 - 0.2284 * horsepower - 0.5372 * weight$$

Regression Equation: TensorFlow: Normalized Data

$$mpg = 0.6394 - 0.3084 * horsepower - 0.4183 * weight$$



Summary

- 2 Variable Regression
- Multi Variable Regression
  - Data Normalization
  - Dataset
  - Multi Variable Regression in R
  - Python: Un-Normalized Data
    - Linear Regression in Scikit-Learn
    - Linear Regression in TensorFlow (No Solution)
  - Python: Normalized Data
    - Linear Regression in Scikit-Learn
    - Linear Regression in TensorFlow