Mechatronics

Day 3 – 4

Supervised Learning

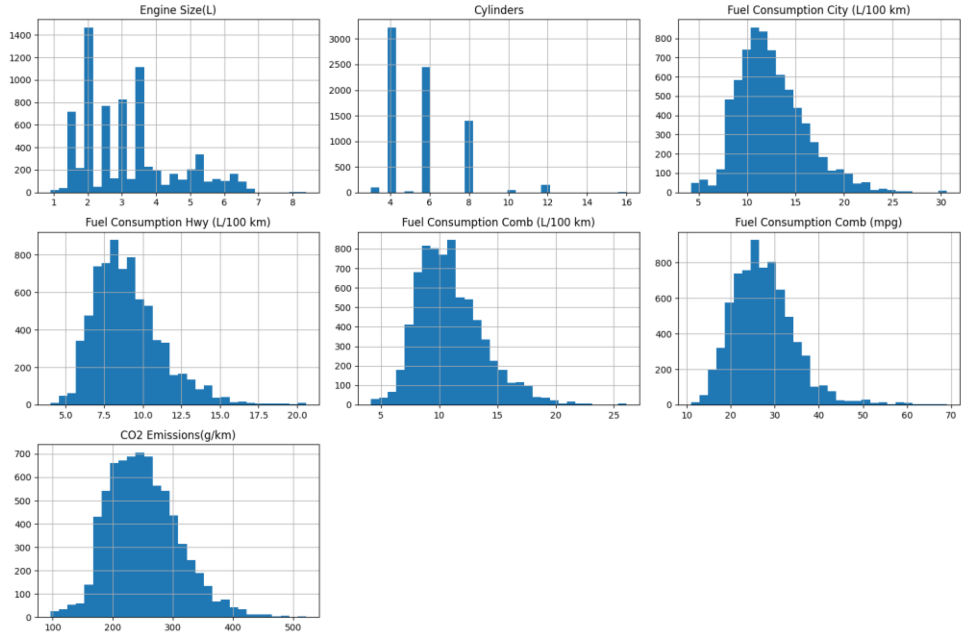
Regression

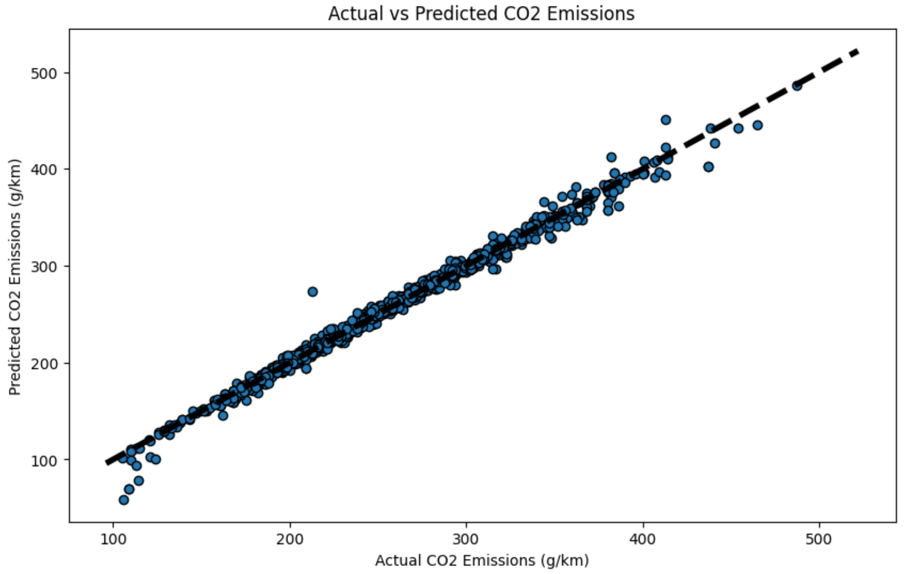
Outline

- Regression problem
- Linear Regression Model
  - Model training
  - Cost function
  - Gradient Descent
  - Learning Rate
- Evaluate regression model performance
- Multiple Linear Regression Model
  - Features selection
- Polynomial Regression Model
- The Problem of Overfitting

Regression Problem

Carbon dioxide Emissions Prediction

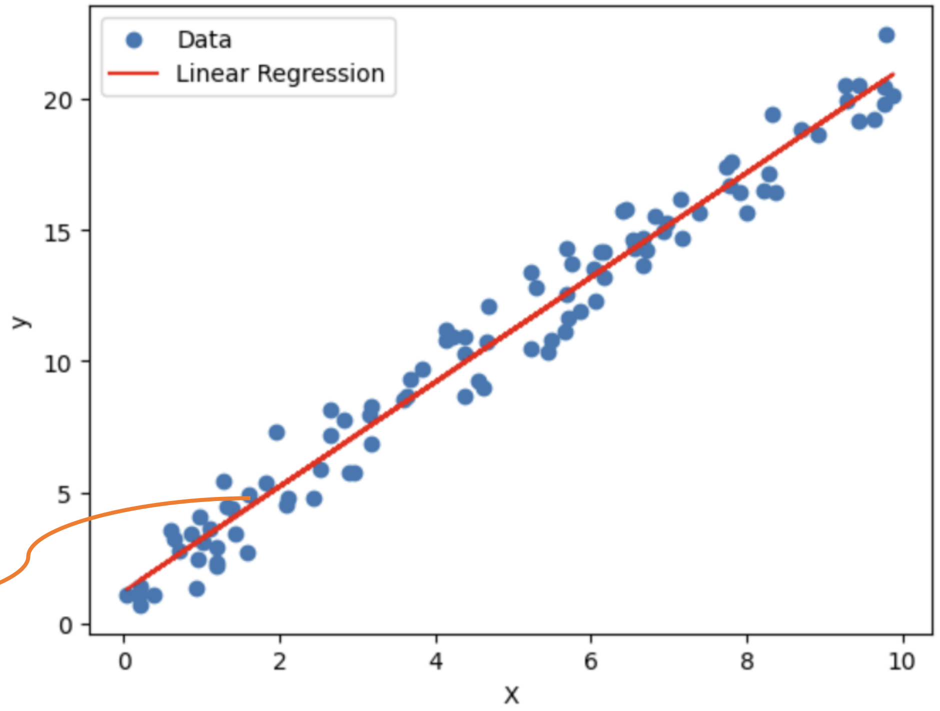




- Selected 7 input features
- Predicted CO2 Emissions as an output target
- Using Multiple Linear Regression Model

Regression Problem

Mean Squared Error: 0.99
R-squared: 0.97



Target

Regression line
(Prediction)

Feature

Linear Regression Model

On training set

index	x (feature)	y (target)
1	20	4
2	30	6
3	40	8
…	…	…
100	1000	50

Notation

$x^{(1)} = 20$

$y^{(1)} = 4$

$x^{(2)} = 30$

$y^{(2)} = 6$

$(x^{(1)}, y^{(1)}) = (20, 4)$

$(x^{(2)}, y^{(2)}) = (30, 6)$

Meaning

x = input variable or feature

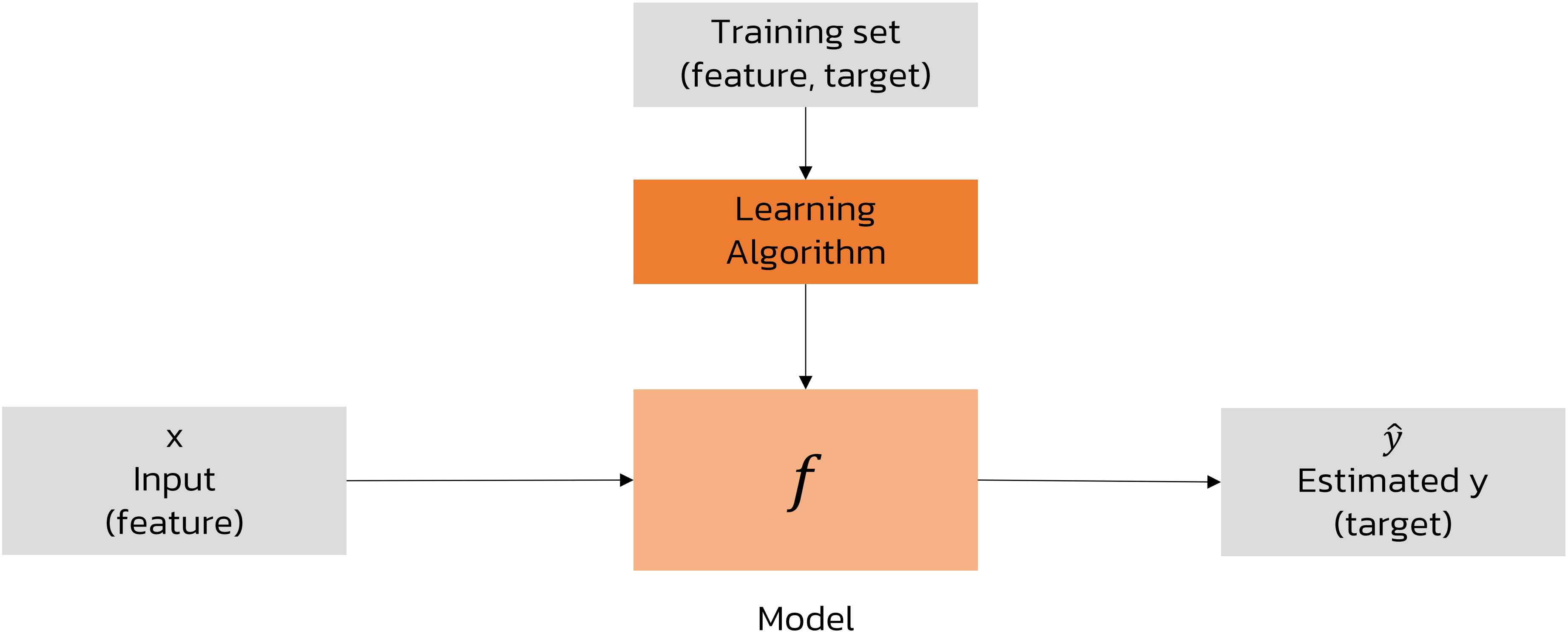
y = output variable or target

m = number of training sample

$(x^{(i)}, y^{(i)}) = i^{th}$ training sample

i = index (1^{st} , 2^{nd} , 3^{rd}, 4^{th}, …)

Model training



Linear regression model : the model

x (feature) e.g. Engine Size (L)	y (target) e.g. CO2 emission (g/km)
20	4
30	6
40	8
…	…
1000	50

Simple / Univariable Linear Regression

- Linear Regression with one variable (single feature of x)

Model

f ➡ $$f(x) = f_{w,b}(x) = wx + b$$

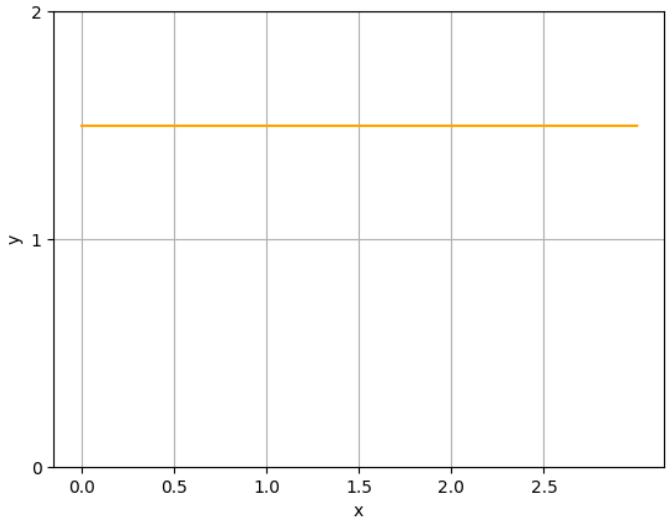
w, b are models' parameters

w = coefficient or “weight”

b = y–intercept or “bias”

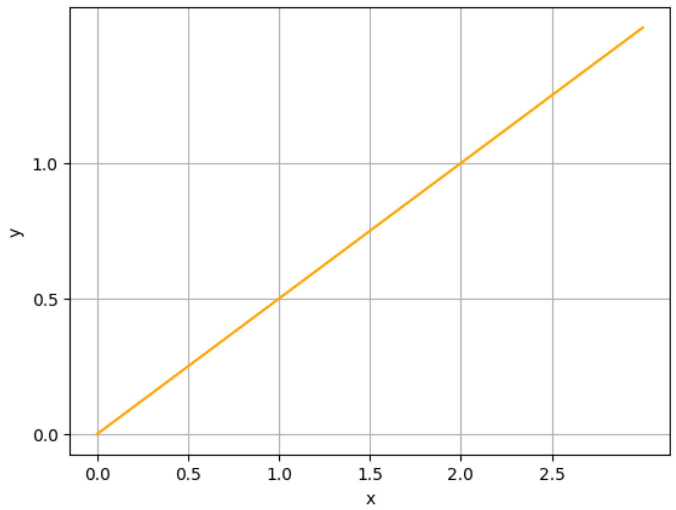
Linear regression model : plot examples

$$f_{w,b}(x) = wx + b$$



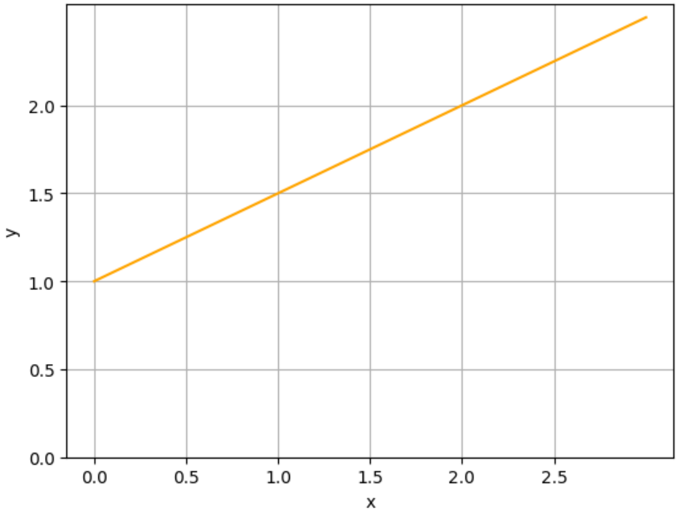
$f(x) = 0x + 1.5$

w =
b =



$f(x) = 0.5x$

w =
b =



$f(x) = 0.5x + 1$

w =
b =

Code 1 : finding coefficient and Intercept

```
from sklearn.linear_model import LinearRegression
import numpy as np
data = [[2, 6], [4, 10], [6, 11], [8, 14], [10, 22], [12,
25]]
data = np.array(data)
x = data[:, 0].reshape(-1, 1)
y = data[:, 1].reshape(-1, 1)
model = LinearRegression()
model.fit(x, y)
y_5 = model.predict([[5]])
print('x = 5, y =', y_5[0, 0])
y_9 = model.predict([[9]])
print('x = 9, y =', y_9[0, 0])
y_11 = model.predict([[11]])
print('x = 11, y =', y_11[0, 0])
print('intercept =', model.intercept_[0])
print('coefficient =', model.coef_[0, 0])
```

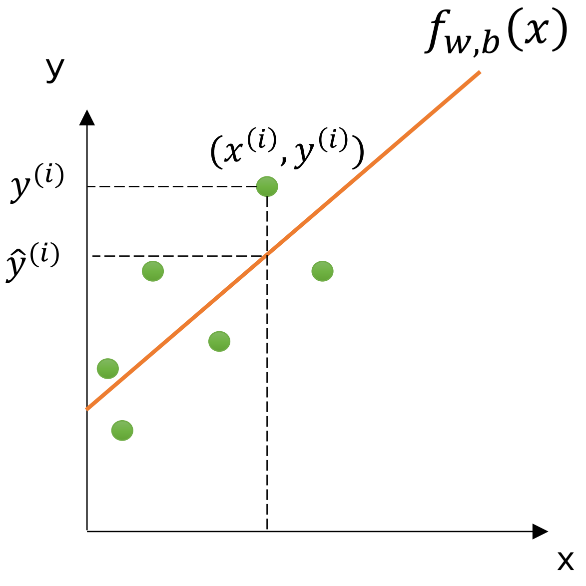
In Scikit-Learn

For Train Model use: `model.fit(__,__)`

For Make prediction use: `model.predict(__)`

```
x = 5, y = 10.838095238095237
x = 9, y = 18.495238095238093
x = 11, y = 22.323809523809523
intercept = 1.2666666666666664
coefficient = 1.9142857142857146
```

Linear regression model : Cost function



$$\hat{y}^{(i)} = f_{w,b}\left(x^{(i)}\right) = wx^{(i)} + b$$

Determine: w, b which make

$\hat{y}^{(i)}$ is close to $y^{(i)}$ for all $(x^{(i)}, y^{(i)})$

Minimize : Squared error cost function

$$J(w,b) = \frac{1}{2m}\sum_{i=1}^{m}(\hat{y}^{(i)} - y^{(i)})^2$$

$$J(w,b) = \frac{1}{2m}\sum_{i=1}^{m}(f_{w,b}\left(x^{(i)}\right) - y^{(i)})^2$$

Use $\frac{1}{2m}$ is simplifies the gradient of cost function during obtimization.

Linear regression model : Cost function intuition

Model : $f_{w,b}(x) = wx + b$

Parameter: w, b

Cost function: $J(w,b) = \frac{1}{2m}\sum_{i=1}^{m}(f_{w,b}(x^{(i)}) - y^{(i)})^2$

Goal: minimize $J(w,b)$

Simplified

Model : $f_w(x) = wx$, b = 0

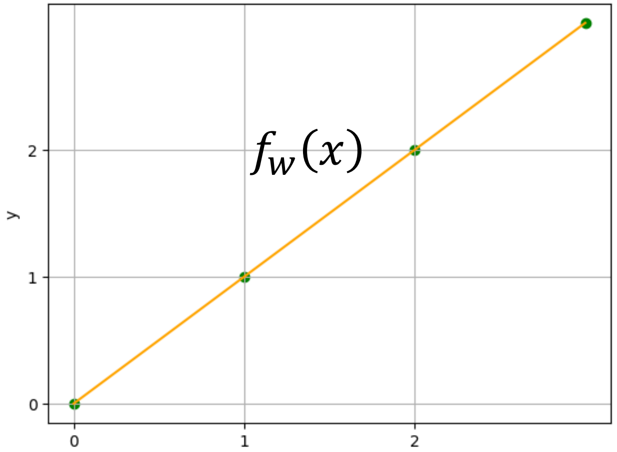
Parameter: w

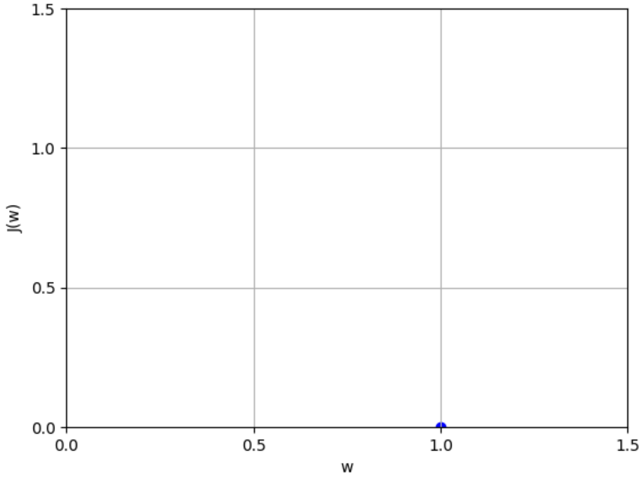
Cost function: $J(w) = \frac{1}{2m}\sum_{i=1}^{m}(f_w(x^{(i)}) - y^{(i)})^2$

Goal: minimize $J(w)$

Linear regression model : Cost function intuition

Example 1





Defined: w = 1

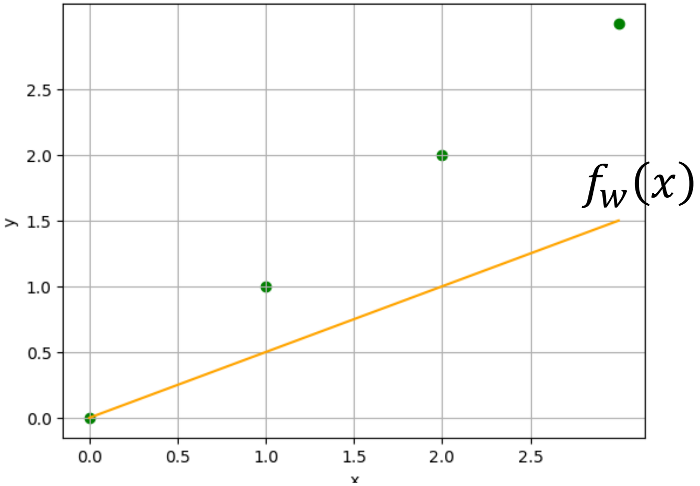
$$f_w(x) = x$$

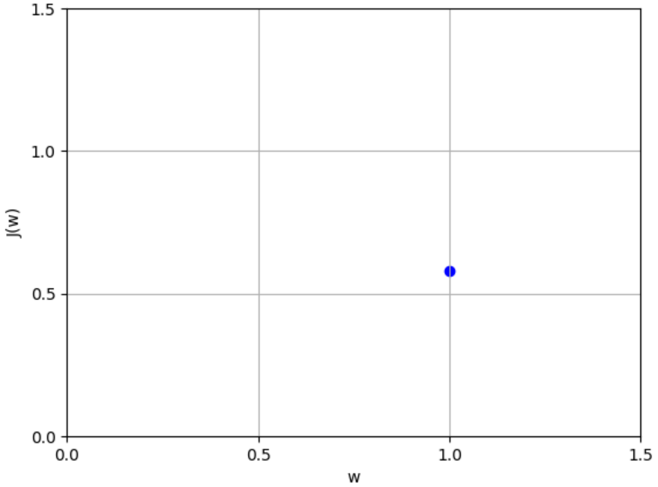
$f_w(1) = 1,\qquad f_w(2) = 2,\qquad f_w(3) = 3$

$$J(w) = \frac{1}{2m}\sum_{i=1}^{m}(w(x^i) - y^{(i)})^2 \quad = \frac{1}{2(3)}(0+0+0) = 0$$

Linear regression model : Cost function intuition

Example 2





Defined: w = 0.5

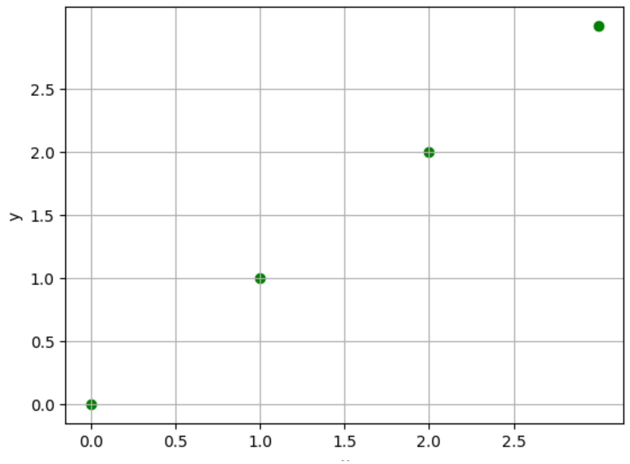
$$f_w(x) = x$$

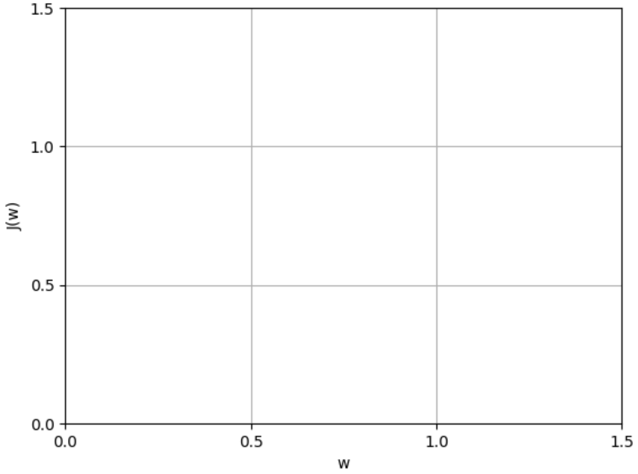
$$f_w(1) = 0.5, \qquad f_w(2) = 1, \qquad f_w(3) = 1.5$$

$$J(w) = \frac{1}{2(3)}\left((0.5-1)^2+(1-2)^2+(1.5-3)^2\right) = 0.58$$

Linear regression model : Cost function intuition

Example 3





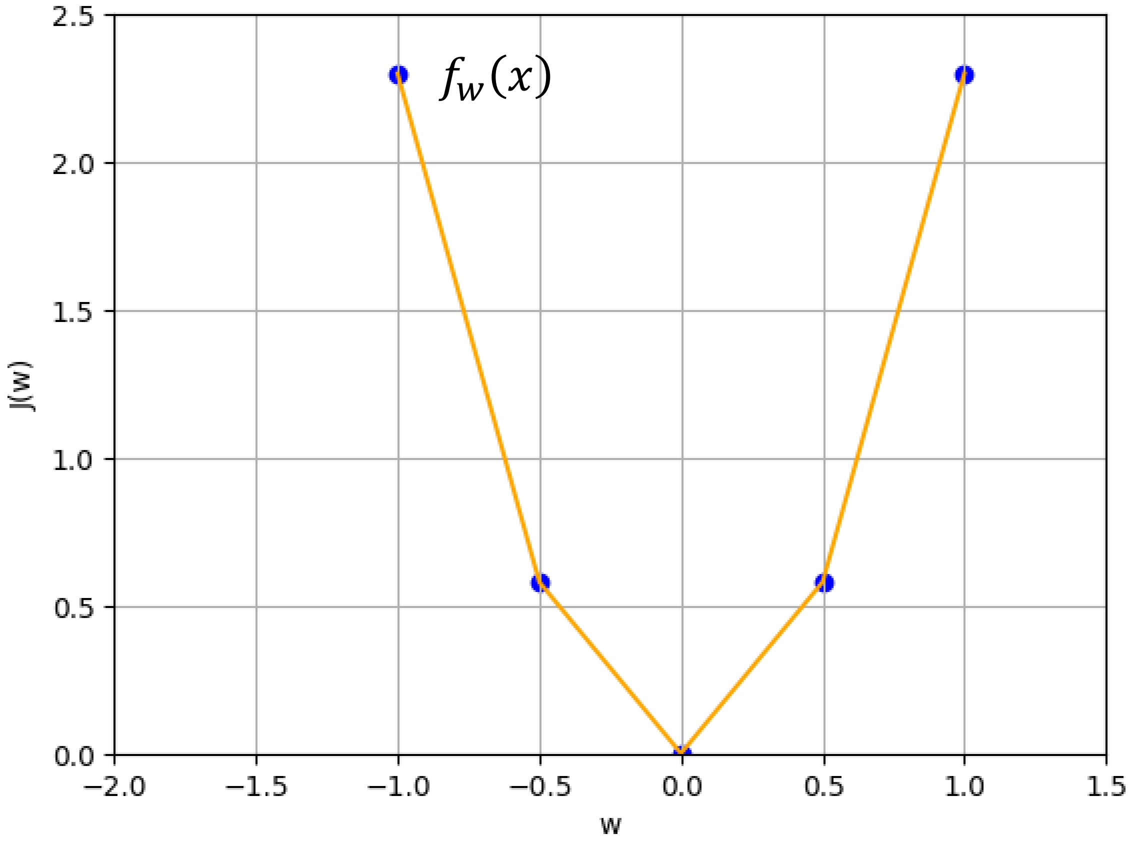
Defined: w = 0

$f_w(x) = x$

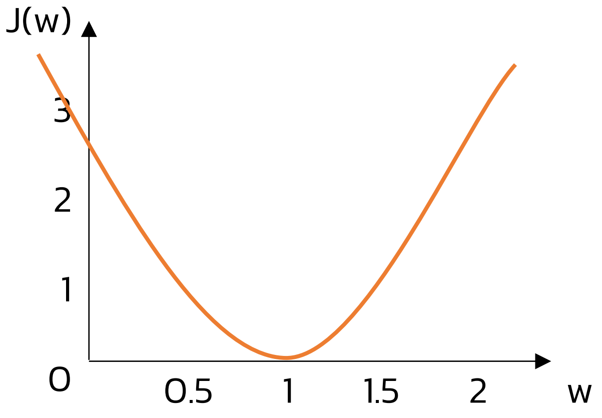
$f_w(1) =,$ $f_w(2) =,$ $f_w(3) =$

$J(w) =$

Linear regression model : Cost function intuition



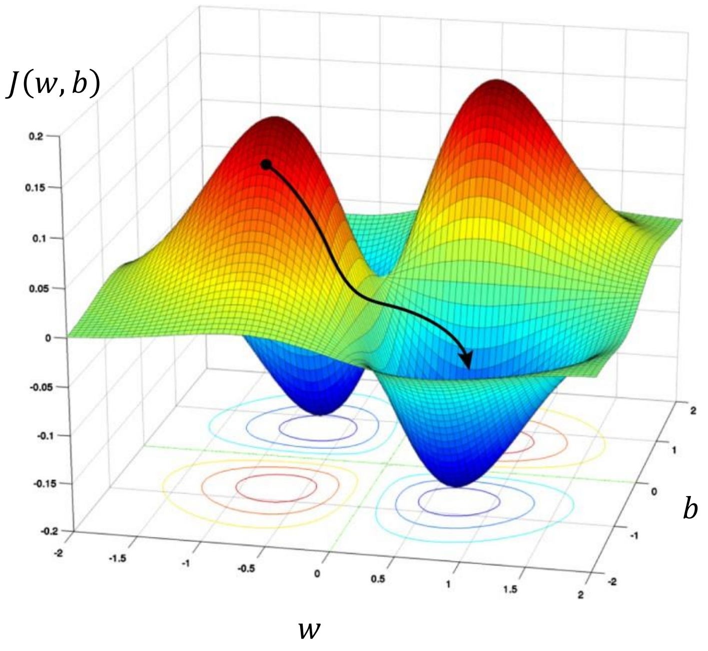
Linear regression model : Cost function intuition



Choose w to minimize $J(w)$

Goal for linear regression (In general cases) : minimize $J(w, b)$

Linear regression model : Gradient Descent



Model cost function : $J(w,b)$ for linear regression or any model

Goal : minimize $J(w,b)$

Outline: 1. Start with some w, b (e.g. w = 0, b = 0)

2. Changing w, b to reduce $J(w,b)$

until it settle at/near a minimum value.

Linear regression model : Gradient Descent Algorithms

$Repeat\ until\ convergence\ \{$

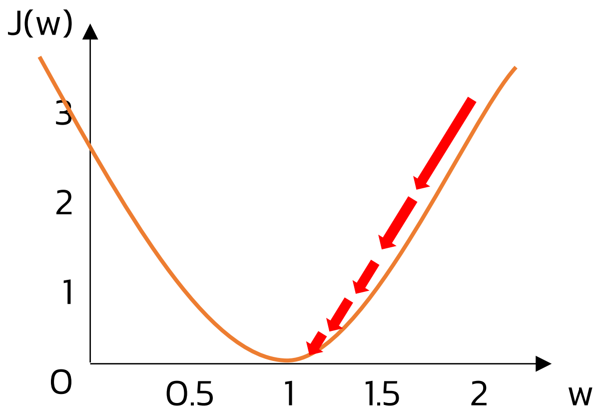
$$w = w - \alpha\frac{\partial J(w,b)}{\partial w}, \qquad b = b - \alpha\frac{\partial J(w,b)}{\partial b} \qquad \}$$

α = Learning rate

Simultaneously update w and b

$$new\_w = w - \alpha\frac{\partial J(w,b)}{\partial w}$$

$$new\_b = b - \alpha\frac{\partial J(w,b)}{\partial b}$$



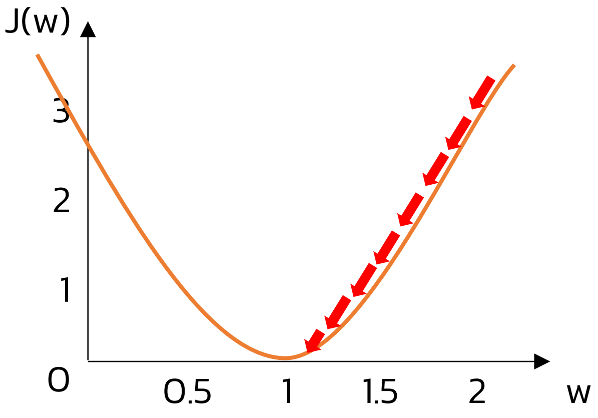
Linear regression model :Learning rate (α)

$$w = w - \alpha \frac{\partial J(w,b)}{\partial w}$$

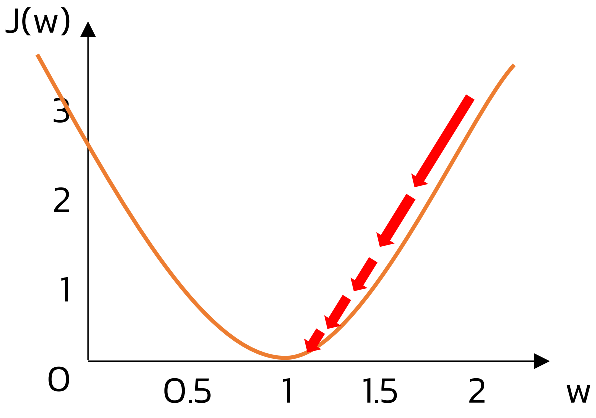
- If α is too small : Gradient descent may be slow from take a lot of iterations.
- If α is too large : Gradient descent may Overshoot or never reach to minimum $J(w,b)$

  or fail to converge (diverge)

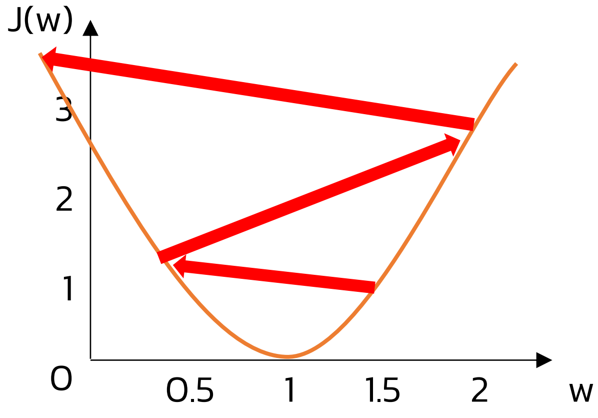
Smaller α (update 8 times)



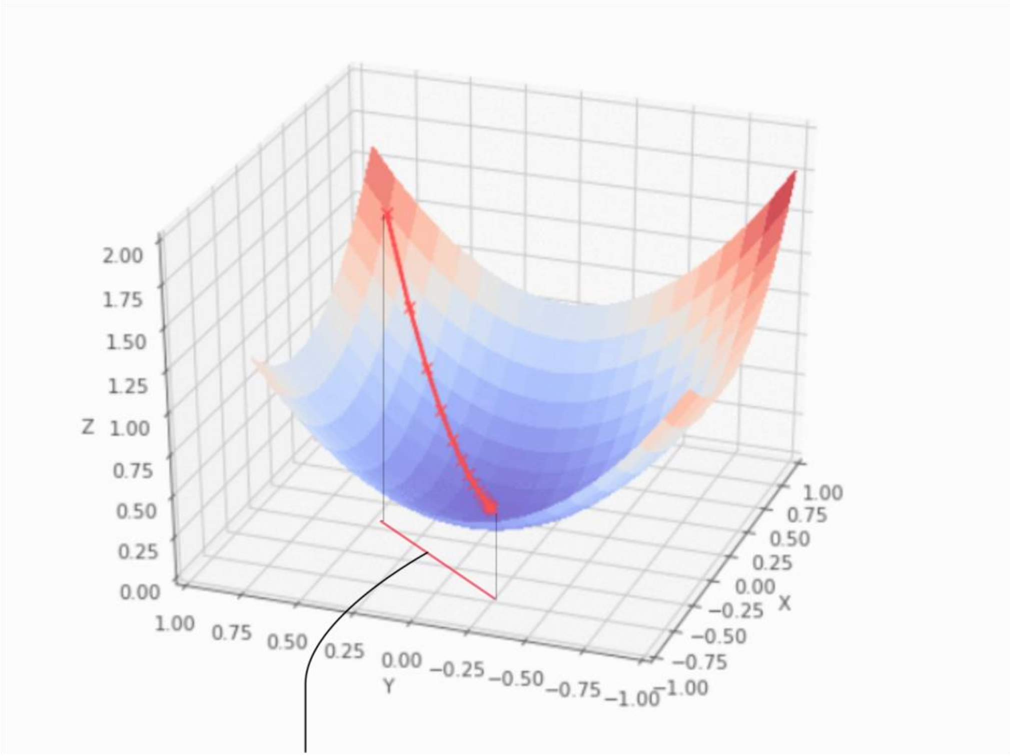
Optimal α (update 5 times)



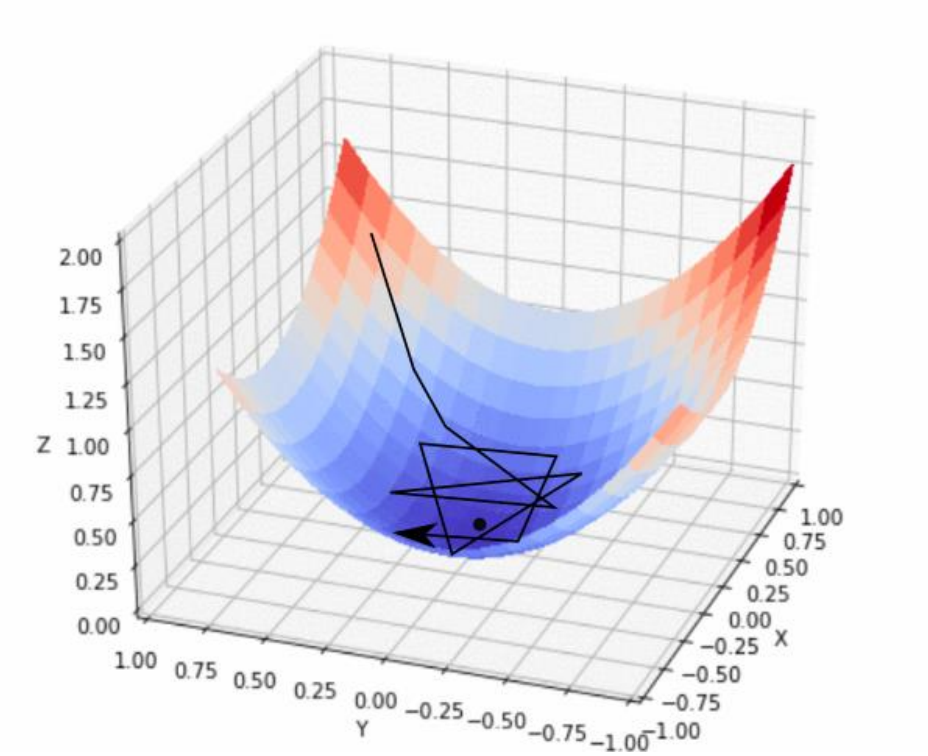
Larger α never reach to minimum



Linear regression model :Learning rate (α)



Real Trajectory of G.D.



Large Learning rate

https://www.digitalocean.com/community/tutorials/intro-to-optimization-in-deep-learning-gradient-descent

Summary for model learning

Linear regression model

$$f_{w,b}(x) = wx + b$$

Cost function

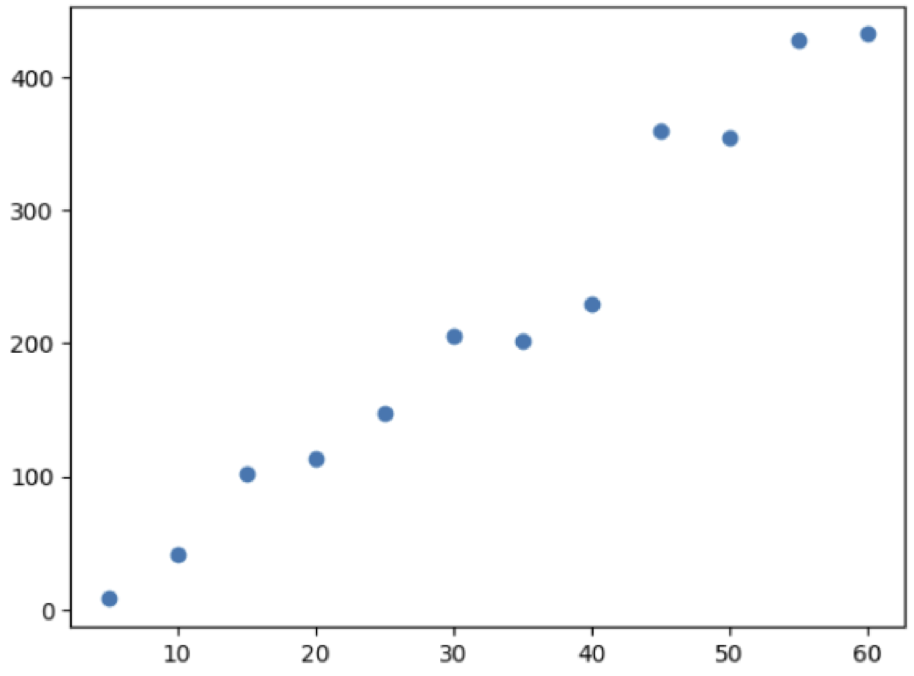
$$J(w,b) = \frac{1}{2m}\sum_{i=1}^{m}(f_{w,b}\left(x^{(i)}\right) - y^{(i)})^2$$

Gradient Descent algorithm

$$Repeat\ until\ convergence\,\{\ w = w\ -\ \alpha\frac{\partial J(w,b)}{\partial w}, b = b -\ \alpha\frac{\partial J(w,b)}{\partial b}\}$$

Code 2 : find coefficient and Intercept

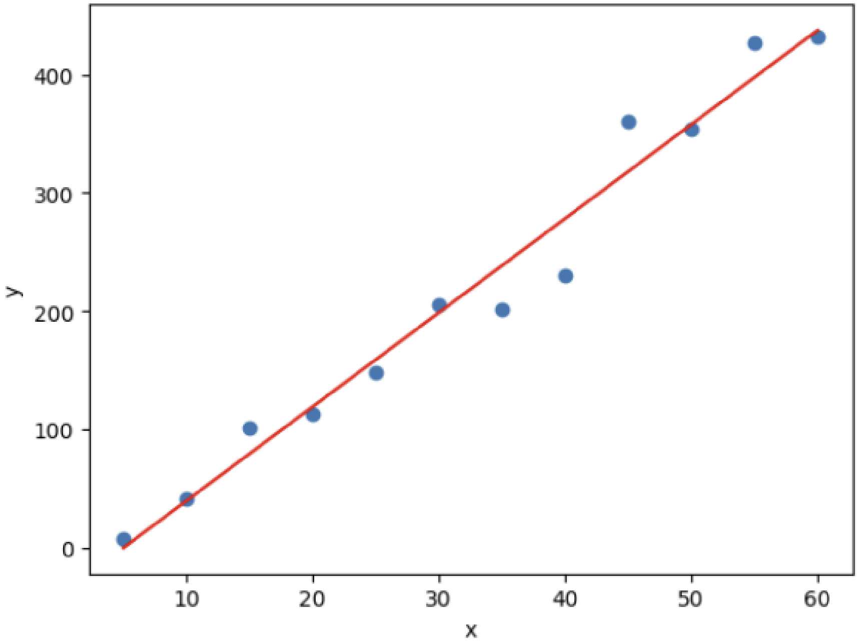
```
from sklearn.linear_model import LinearRegression
import numpy as np
import matplotlib.pyplot as plt

x = list(range(5, 61, 5))
y = [8, 42, 102, 113, 148, 206, 202, 230, 360, 354, 427, 432]
plt.scatter(x, y)
plt.show()

x = np.array(x).reshape(-1, 1)
model = LinearRegression()
model.fit(x, y)

x_predict = [[8], [18], [33], [59]]
y_predict = model.predict(x_predict)

for (i, p) in enumerate(x_predict):
    y_pred = '{:.2f}'.format(y_predict[i])
    print(f’x ={p[0]} predicted y ={y_pred}')
print()
ic = '{:.2f}'.format(model.intercept_)
ce = '{:.2f}'.format(model.coef_[0])
print(f’Linear equation is:  y = {ic} + ({ce})x')
```




```
x = 8 predicted y = 23.87
x = 18 predicted y = 103.38
x = 33 predicted y = 222.64
x = 59 predicted y = 429.37

Linear equation is:  y = -39.74 + (7.95)x
```

Code 2

```
y_predict_all = model.predict(x)
plt.scatter(x,y)
plt.plot(x,y_predict_all,color='red')
plt.xlabel("x")
plt.ylabel("y")
plt.show()
```



Evaluate regression model performance

R-squared or Coefficient of determination – model's accuracy

$$R^2 = 1 - \frac{\sum_i^n \left(y^{(i)} - \hat{y}^{(i)}\right)^2}{\sum_i^n (y^{(i)} - \bar{y}^{(i)})^2}$$

$y^{(i)} = actual\ y$
$\hat{y}^{(i)} = predicted\ y$
$\bar{y}^{(i)} = average\ of\ actual\ y$

$R^2 \in [0,1]$

R^2 = 0.8 (Accuracy =80%)
R^2 = 0.05 (Accuracy =5%)

Evaluate regression model performance

Mean Squared Error (MSE)

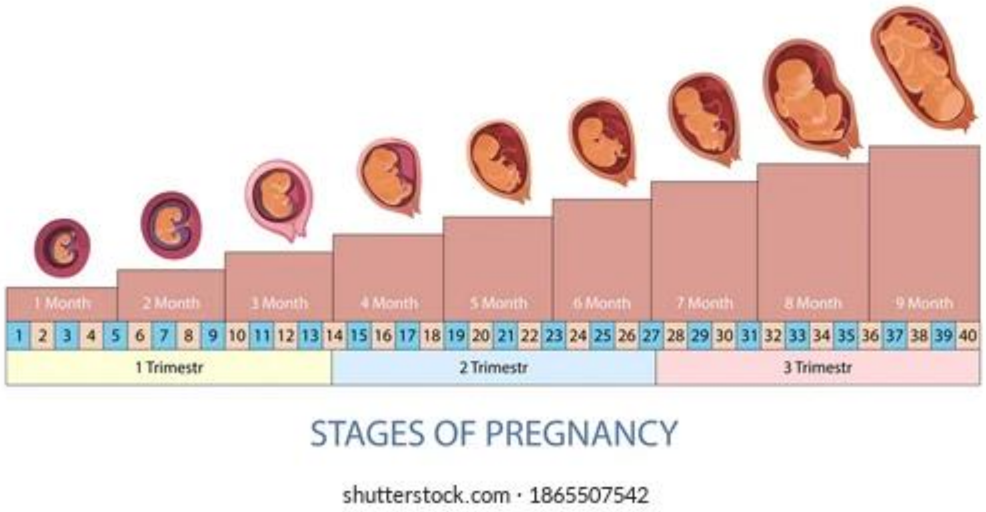
$$MSE = \frac{\sum_{i=1}^{n}(y^{(i)} - \hat{y}^{(i)})^2}{m}$$

Root Mean Squared Error (RMSE) – same unit with $y^{(i)}$ and $\hat{y}^{(i)}$)

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(y^{(i)} - \hat{y}^{(i)})^2}{m}} = \sqrt{MSE}$$

Code 3 : Use gestational age for predict birth of newborn

	A	B
1	Gestational_Age_wks	Birth_Weight_gm
2	34.7	1895
3	36	2030
4	29.3	1440
5	40.1	2835
6	35.7	3090
7	42.4	3827
8	40.3	3260
9	37.3	2690
10	40.9	3285
11	38.3	2920
12	38.5	3430
13	41.4	3657
14	39.7	3685
15	39.7	3345
16	41.1	3260
17	38	2680
18	38.7	2005

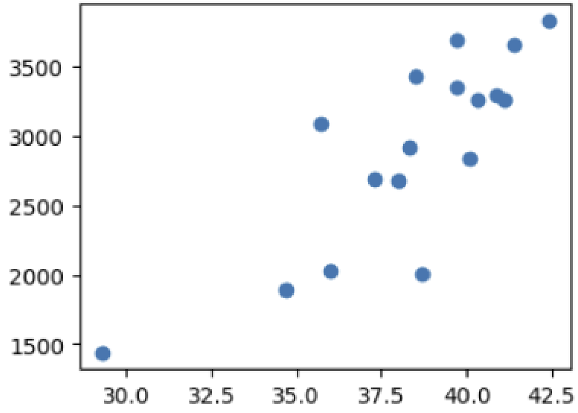


Code 3 : Training Linear Regression and evaluate model

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.metrics import r2_score

df = pd.read_excel('birth_weight.xlsx')
with pd.option_context('display.max_rows', 6): display(df)
x = df['Gestational_Age_wks']
y = df['Birth_Weight_gm’]

plt.figure(figsize=(4, 3))
plt.scatter(x, y)
plt.show()
```

	Gestational_Age_wks	Birth_Weight_gm
0	34.7	1895
1	36.0	2030
2	29.3	1440
...	...	...
14	41.1	3260
15	38.0	2680
16	38.7	2005

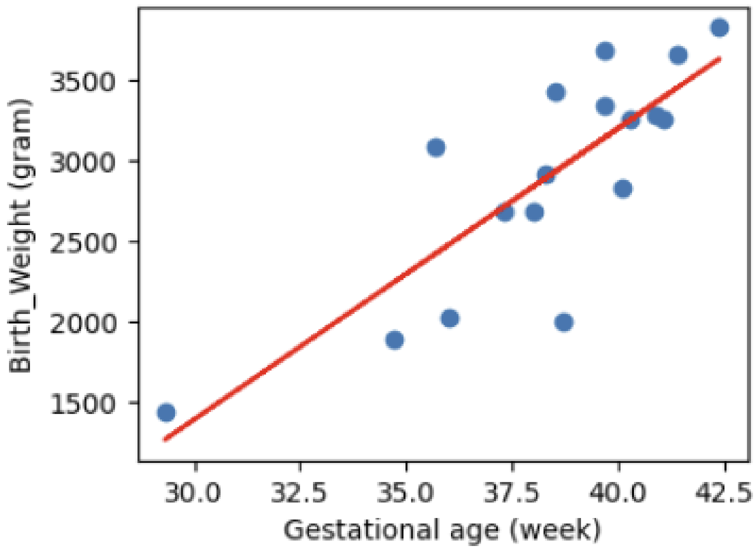
17 rows × 2 columns



Code 3 : Training Linear Regression and evaluate model

```
x = np.array(x).reshape(-1, 1)
model = LinearRegression()
model.fit(x, y)
y_predict = model.predict(x)

mse = mean_squared_error(y, y_predict)
rmse = np.sqrt(mse)
print('MSE:', '{:.2f}'.format(mse))
print('RMSE:', '{:.2f}'.format(rmse))
score = model.score(x, y)
#score = r2_score(y, y_predict)
print('R-Squared:', '{:.2f}'.format(score))
ic = '{:.2f}'.format(model.intercept_)
ce = '{:.2f}'.format(model.coef_[0])
print(f'y = {ic} + ({ce})x')

# plt.figure(figsize=(4, 3))
# plt.scatter(x, y)
# plt.plot(x,y_predict,'red')
# plt.xlabel("Gestational age (week)")
# plt.ylabel("Birth_Weight (gram)")
# plt.show()
```

```
MSE: 151544.06
RMSE: 389.29
R-Squared: 0.67
 y = -4020.05 + (180.46)x
```



Multiple Linear Regression

X1	X2	X3	Y
Engine Size (L)	Cylinders	Fuel Consumption (L/100 km)	CO2 emission (g/km)
20	2	10	4
30	4	12	6
40	6	16	8
…	…	…	…
1000	6		50

- Multiple variables (features)

Model :

$$f_{\vec{w},b}(\vec{x}) = w_1x_1 + w_2x_2 + \cdots + w_nx_n + b = \vec{w} \cdot \vec{x} + b$$

dot product

Features:

$$\vec{x} = [x_1 \;\; x_2 \;\; x_3 \;\; \ldots \; x_n]$$

Parameters:

$$\vec{w} = [w_1 \;\; w_2 \;\; w_3 \;\; \ldots \; w_n]$$

$$b = a \; number$$

Gradient Descent for Multiple Linear Regression

Parameters : $\vec{w} = [w_1 \;\; w_2 \;\; w_3 \;\; \dots \; w_n]$ and b

Model : $f_{\vec{w},b}(\vec{x}) = \vec{w} \cdot \vec{x} + b$

Cost function : $J(\vec{w}, b)$

Gradient descent : $Repeat\ until\ convergence\,\{\, w_j = w_j - \alpha \frac{\partial J(\vec{w},b)}{\partial w_j}, b = b - \alpha \frac{\partial J(\vec{w},b)}{\partial b} \}$

$w_j = \mathrm{w\ of\ feature\ j^{th}}$

Code 4 : Multiple linear regression

```python
import numpy as np
d = [[8, 4, 45],
     [7, 5, 44],
     [8, 6, 50],
     [6, 6, 43],
     [9, 5, 45],
     [8, 3, 44],
     [9, 4, 40],
     [6, 5, 43]]
d = np.array(d)
x = d[:, 0:2]
y = d[:, 2]
model = LinearRegression()
model.fit(x, y)
ic = '{:.2f}'.format(model.intercept_)
ce1 = '{:.2f}'.format(model.coef_[0])
ce2 = '{:.2f}'.format(model.coef_[1])
print(f'y = {ic} + ({ce})x1 + ({ce2})x2')
x_predict = [[6, 6], [7, 6]]
y_predict = model.predict(x_predict)
p_6_6 = '{:.2f}'.format(y_predict[0])
p_7_6 = '{:.2f}'.format(y_predict[1])
print(f'x1 = 6, x2 = 6, y = {p_6_6}')
print(f'x1 = 7, x2 = 6, y = {p_7_6}')
```

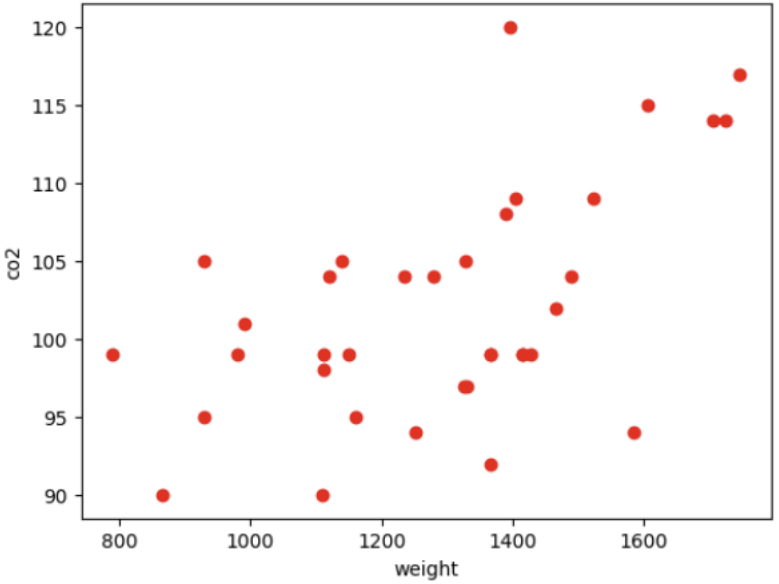
```
y = 31.37 + (180.46)x1 + (1.51)x2
x1 = 6, x2 = 6, y = 44.92
x1 = 7, x2 = 6, y = 45.67
```

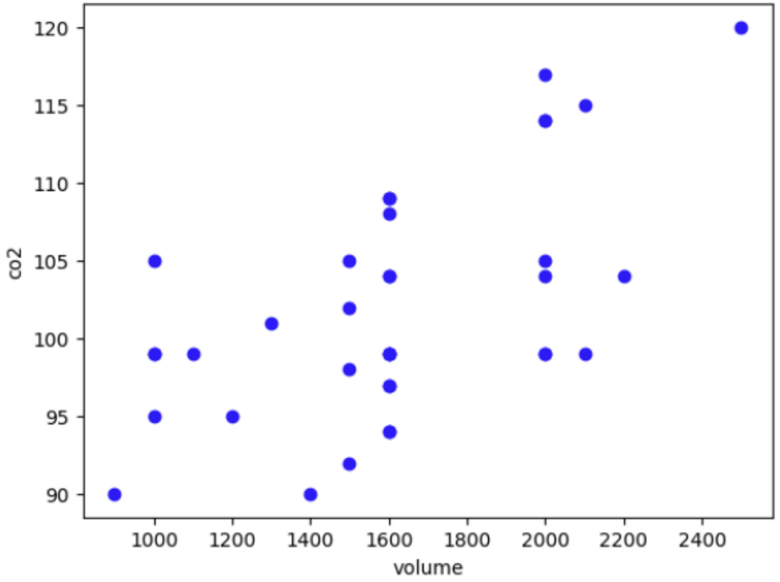
Code 5 :Predict CO2 emission

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split

df = pd.read_csv('co2-emission.csv')
with pd.option_context('display.max_rows', 8): display(df)

x1 = df['volume']
x2 = df['weight']
y = df['co2']
plt.scatter(x1, y, c='b')
plt.xlabel('volume')
plt.ylabel('co2')
plt.show()
plt.scatter(x2, y, c='r')
plt.xlabel('weight')
plt.ylabel('co2')
plt.show()
```

	car	model	volume	weight	co2
0	Toyota	Aygo	1000	790	99
1	Mitsubishi	Space Star	1200	1160	95
2	Skoda	Citigo	1000	929	95
3	Fiat	500	900	865	90
...	...	...	...	...	...
32	Ford	B-Max	1600	1235	104
33	BMW	2	1600	1390	108
34	Opel	Zafira	1600	1405	109
35	Mercedes	SLK	2500	1395	120

36 rows × 5 columns





```python
x = df[['volume','weight']]
x_train, x_test, y_train, y_test = train_test_split(x, y, random_state=0)
model = LinearRegression()
model.fit(x_train, y_train)
# Predict CO2 emission when engine volume: 2300, car weight: 1300
x_predict = [[2300, 1300]]
y_predict = model.predict((x_predict))
for (i, x_p) in enumerate(x_predict):
    co2 = '{:,.0f}'.format(y_predict[i])
    print(f'Engine size: {x_p[0]}, weight: {x_p[1]}, CO2 emission: {co2}')
score = model.score(x_test, y_test)
print('R-Squared:', '{:.2f}'.format(score))
ic = '{:.2f}'.format(model.intercept_)
ce1 = '{:.4f}'.format(model.coef_[0])
ce2 = '{:.4f}'.format(model.coef_[1])
print(f'y = {ic} + ({ce1})x1 + ({ce2})x2')
```

```
Engine size: 2300, weight: 1300, CO2 emission: 107
R-Squared: 0.39
y = 79.52 + (0.0075)x1 + (0.0081)x2
```

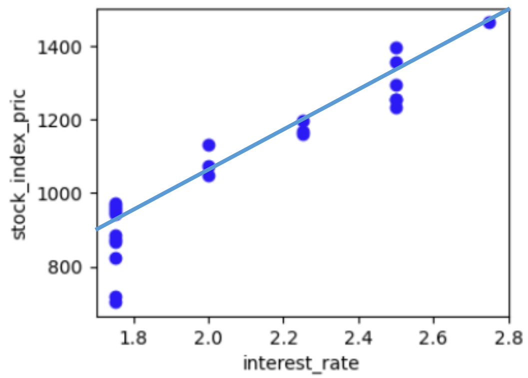
Feature Selection

Correlation Coefficient (r)

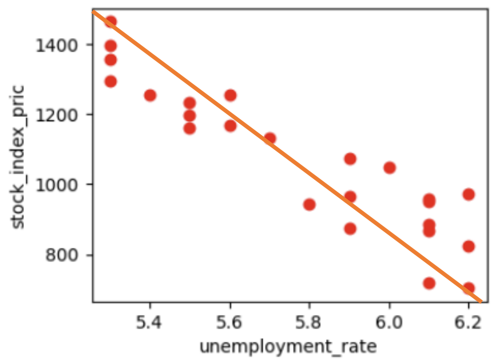
$$r \in [-1,1]$$

- r = 0 : Not have relation between each variables
- r > 0 : Positive relation
- r < 0 : Negative relation
- Large |r| more relation than small |r|

r > 0



r < 0

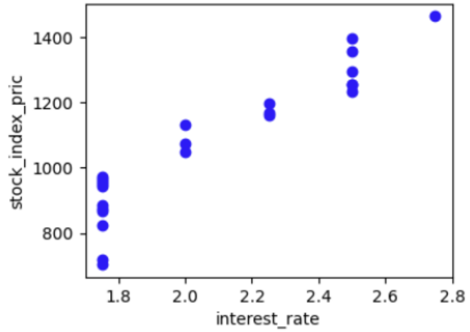


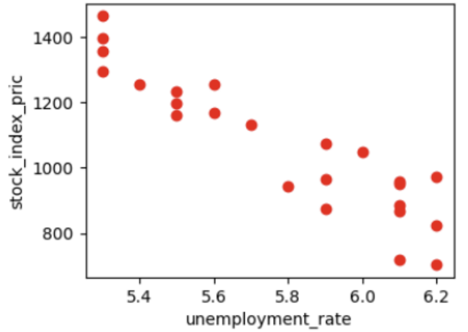
Code 6 : Feature Selection

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression

df = pd.read_csv('stock_index_price.csv')
with pd.option_context('display.max_rows', 6): display(df)

x1 = df['interest_rate']
x2 = df['unemployment_rate']
y = df['stock_index_price’]

plt.figure(figsize=(4, 3))
plt.scatter(x1, y, c='b')
plt.xlabel('interest_rate')
plt.ylabel('stock_index_pric')
plt.show()

plt.figure(figsize=(4, 3))
plt.scatter(x2, y, c='r')
plt.xlabel('unemployment_rate')
plt.ylabel('stock_index_pric')
plt.show()
display(df.corr().round(2))
print()
```

	interest_rate	unemployment_rate	stock_index_price
0	2.75	5.3	1464
1	2.50	5.3	1394
2	2.50	5.3	1357
...	...	...	...
21	1.75	6.2	822
22	1.75	6.2	704
23	1.75	6.1	719

24 rows × 3 columns





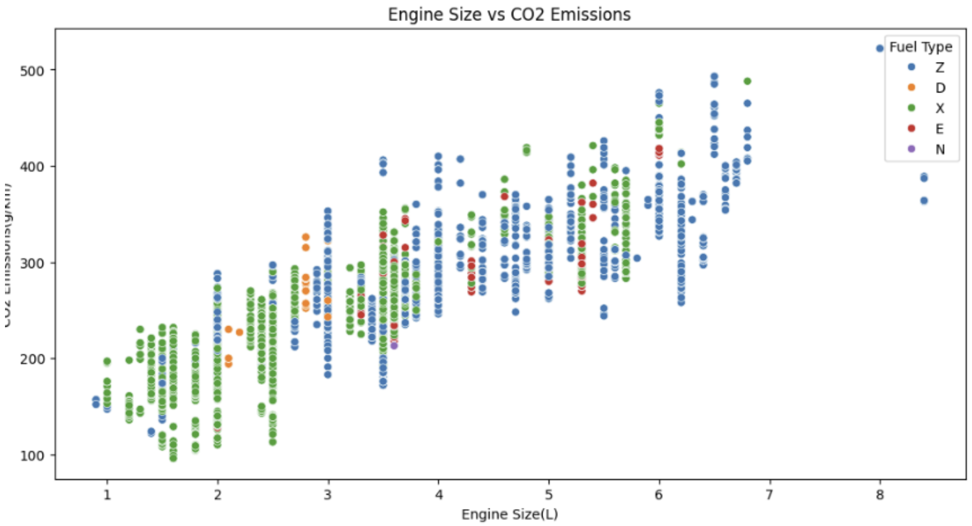
```python
display(df.corr().round(2))
print()
```

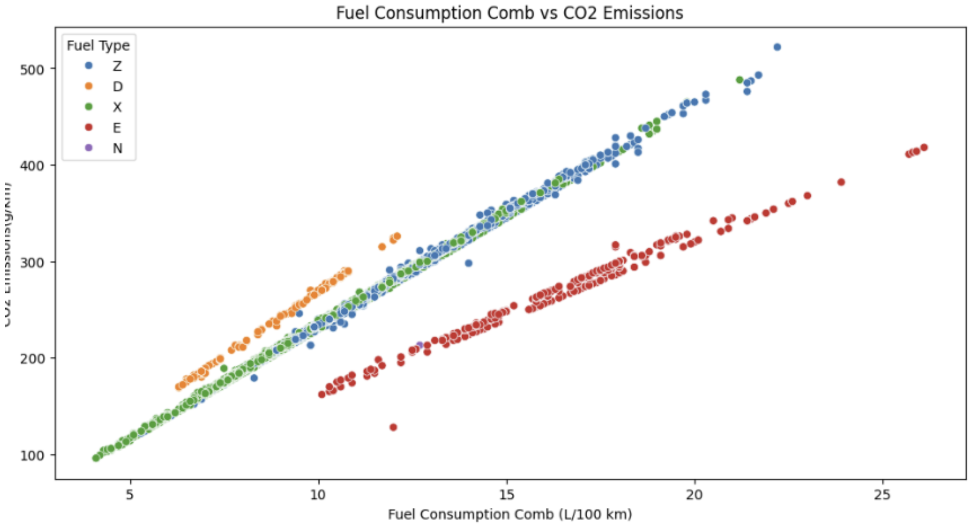
	interest_rate	unemployment_rate	stock_index_price
interest_rate	1.00	-0.93	0.94
unemployment_rate	-0.93	1.00	-0.92
stock_index_price	0.94	-0.92	1.00

```python
# Write code for predict stock price when : Interest rate =  2, unemployment rate = 5 and Interest rate = 2.2, unemployment rate = 5.7
```

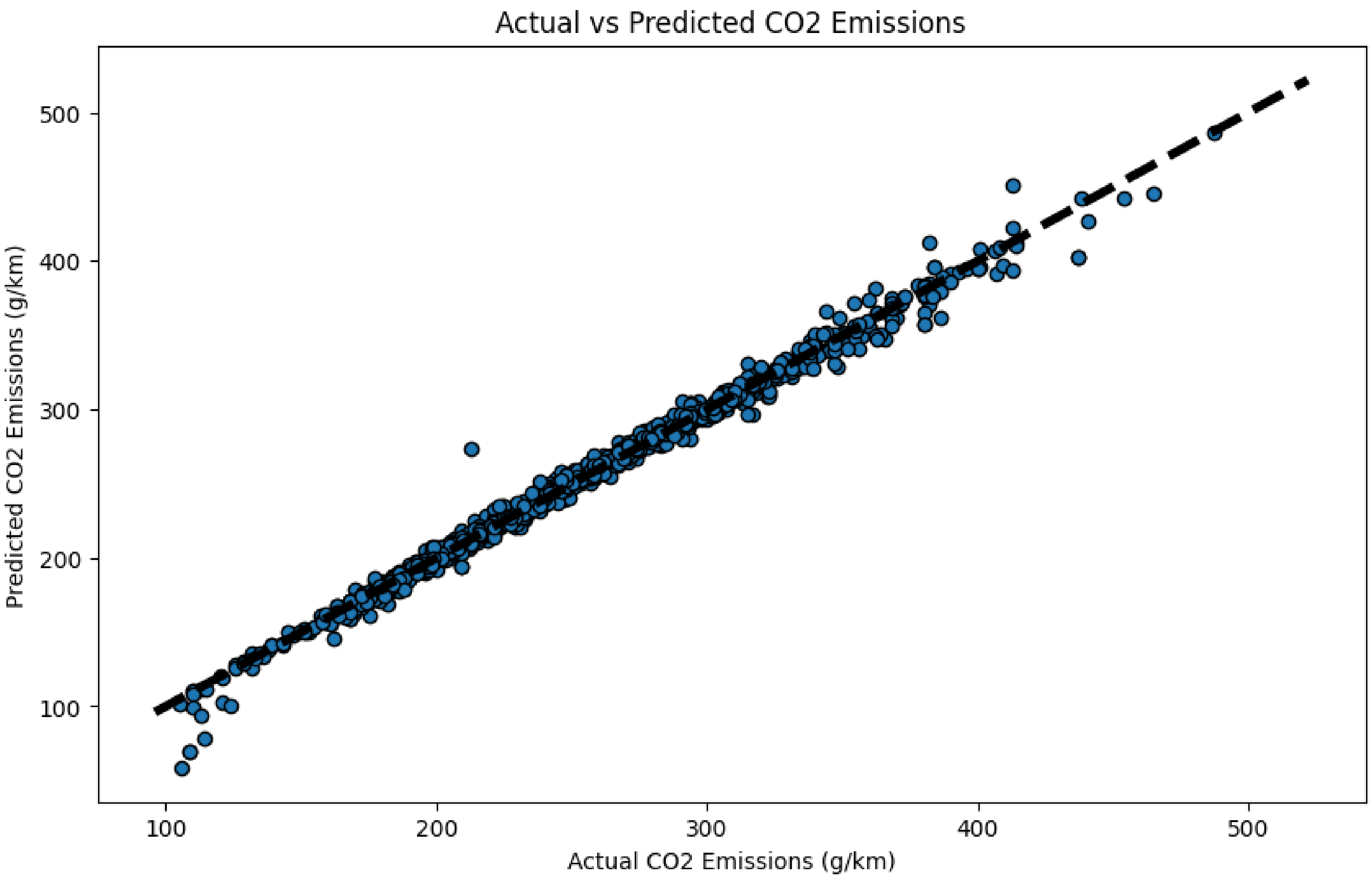
Assignment : Carbon dioxide Emissions Prediction

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	Make	Model	Vehicle Cla	Engine Siz	Cylinders	Transmiss	Fuel Type	Fuel Cons	Fuel Cons	Fuel Cons	Fuel Cons	CO2 Emissions(g/km)	
2	ACURA	ILX	COMPACT	2	4	AS5	Z	9.9	6.7	8.5	33	196	
3	ACURA	ILX	COMPACT	2.4	4	M6	Z	11.2	7.7	9.6	29	221	
4	ACURA	ILX HYBRI	COMPACT	1.5	4	AV7	Z	6	5.8	5.9	48	136	
5	ACURA	MDX 4WD	SUV - SMA	3.5	6	AS6	Z	12.7	9.1	11.1	25	255	
6	ACURA	RDX AWD	SUV - SMA	3.5	6	AS6	Z	12.1	8.7	10.6	27	244	
7	ACURA	RLX	MID-SIZE	3.5	6	AS6	Z	11.9	7.7	10	28	230	
8	ACURA	TL	MID-SIZE	3.5	6	AS6	Z	11.8	8.1	10.1	28	232	
9	ACURA	TL AWD	MID-SIZE	3.7	6	AS6	Z	12.8	9	11.1	25	255	
10	ACURA	TL AWD	MID-SIZE	3.7	6	M6	Z	13.4	9.5	11.6	24	267	
11	ACURA	TSX	COMPACT	2.4	4	AS5	Z	10.6	7.5	9.2	31	212	
12	ACURA	TSX	COMPACT	2.4	4	M6	Z	11.2	8.1	9.8	29	225	
13	ACURA	TSX	COMPACT	3.5	6	AS5	Z	12.1	8.3	10.4	27	239	
14	ALFA ROM	4C	TWO-SEAT	1.8	4	AM6	Z	9.7	6.9	8.4	34	193	
15	ASTON MA	DB9	MINICOM	5.9	12	A6	Z	18	12.6	15.6	18	359	
16	ASTON MA	RAPIDE	SUBCOMP	5.9	12	A6	Z	18	12.6	15.6	18	359	
17	ASTON MA	V8 VANTA	TWO-SEAT	4.7	8	AM7	Z	17.4	11.3	14.7	19	338	
18	ASTON MA	V8 VANTA	TWO-SEAT	4.7	8	M6	Z	18.1	12.2	15.4	18	354	
19	ASTON MA	V8 VANTA	TWO-SEAT	4.7	8	AM7	Z	17.4	11.3	14.7	19	338	
20	ASTON MA	V8 VANTA	TWO-SEAT	4.7	8	M6	Z	18.1	12.2	15.4	18	354	
21	ASTON MA	VANQUISH	MINICOM	5.9	12	A6	Z	18	12.6	15.6	18	359	
22	AUDI	A4	COMPACT	2	4	AV8	Z	9.9	7.4	8.8	32	202	
23	AUDI	A4 QUATT	COMPACT	2	4	AS8	Z	11.5	8.1	10	28	230	
24	AUDI	A4 QUATT	COMPACT	2	4	M6	Z	10.8	7.5	9.3	30	214	
25	AUDI	A5 CABRIC	SUBCOMP	2	4	AS8	Z	11.5	8.1	10	28	230	
26	AUDI	A5 QUATT	SUBCOMP	2	4	AS8	Z	11.5	8.1	10	28	230	
27	AUDI	A5 QUATT	SUBCOMP	2	4	M6	Z	10.8	7.5	9.3	30	214	
28	AUDI	A6 QUATT	MID-SIZE	2	4	AS8	Z	12	8.1	10.2	28	235	
29	AUDI	A6 QUATT	MID-SIZE	3	6	AS8	Z	12.8	8.6	10.9	26	251	
30	AUDI	A6 QUATT	MID-SIZE	3	6	AS8	D	9.8	6.2	8.1	35	217	
31	AUDI	A7 QUATT	MID-SIZE	3	6	AS8	Z	13.3	8.5	11.2	25	262	
32	AUDI	A7 QUATT	MID-SIZE	3	6	AS8	D	9.8	6.2	8.1	35	217	
33	AUDI	A8	MID-SIZE	3	6	AS8	Z	13.1	8.8	11.2	25	258	
34	AUDI	A8	MID-SIZE	4	8	AS8	Z	13.7	8.3	11.3	25	265	
35	AUDI	A8 TDI (m	MID-SIZE	3	6	AS8	D	9.8	6.5	8.4	34	224	
36	AUDI	A8L	FULL-SIZE	3	6	AS8	Z	13.1	8.8	11.2	25	258	
37	AUDI	A8L	FULL-SIZE	4	8	AS8	Z	14.7	9.7	12.5	23	288	
38	AUDI	A8L	FULL-SIZE	6.3	12	AS8	Z	18.7	11.5	15.5	18	363	





Assignment : Carbon dioxide Emissions Prediction



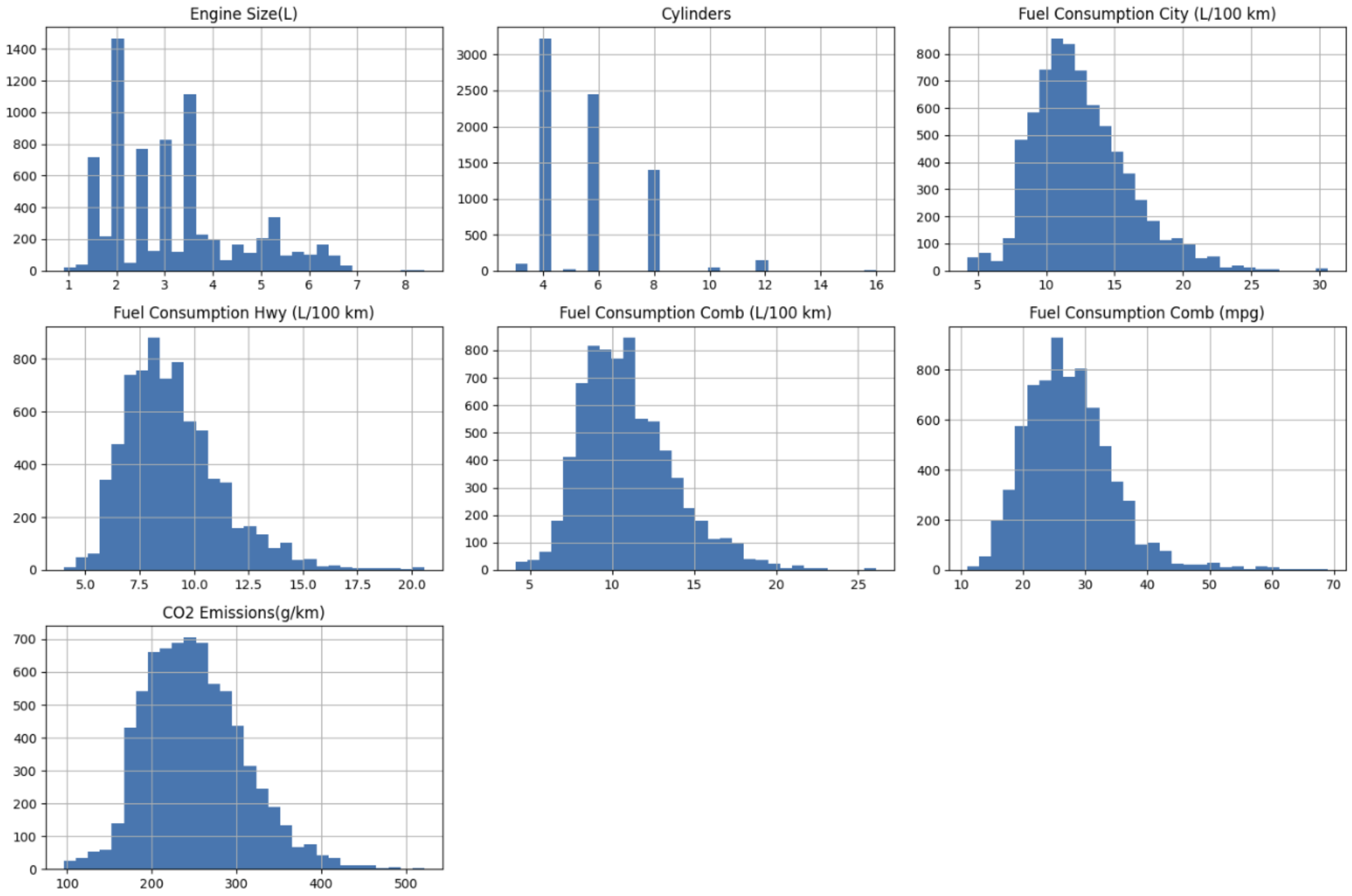
Mean Squared Error: 29.99
R2 Score: 0.99
Root Mean Squared Error: 5.48

Assignment : Carbon dioxide Emissions Prediction

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

df=pd.read_csv("co2.csv")
df.rename(columns={'Make': 'Brand'}, inplace=True)
print(df.info())
print(df.isnull().sum())
print(df.head(15))
```

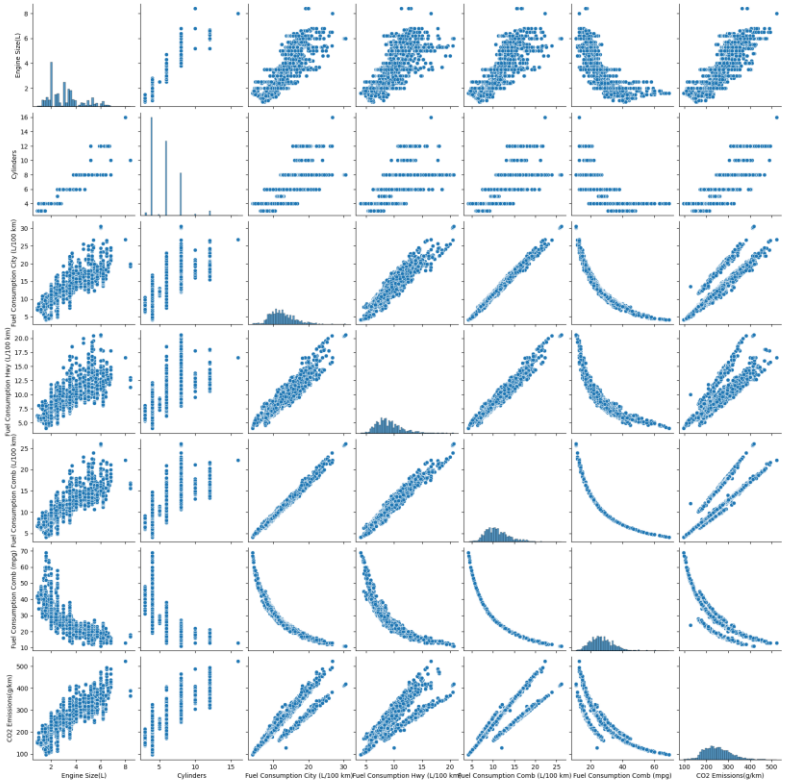
Assignment : Carbon dioxide Emissions Prediction

```
df.hist(bins=30, figsize=(15, 10))
plt.tight_layout()
plt.show()
```



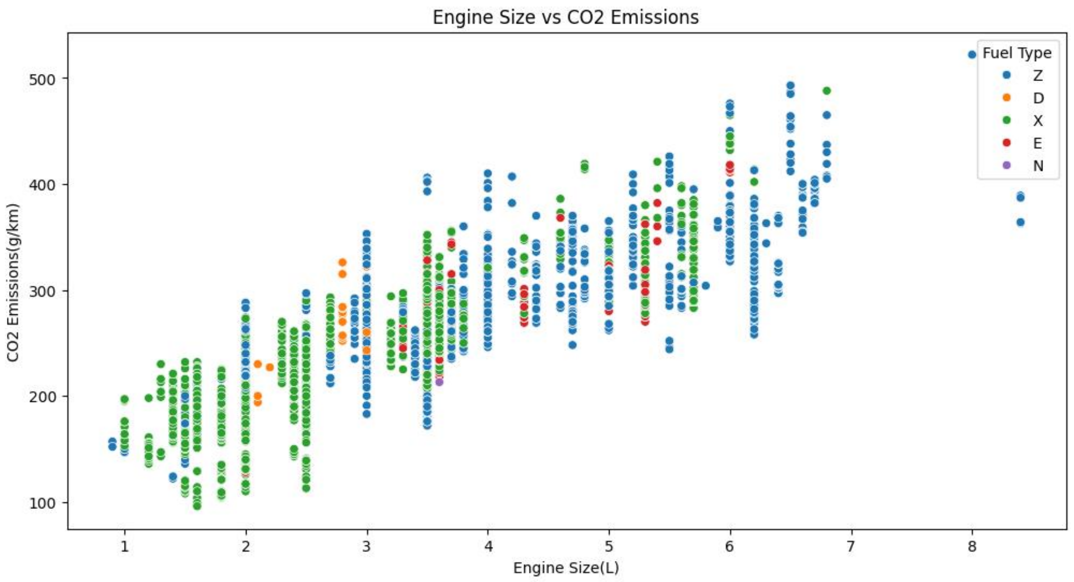
Assignment : Carbon dioxide Emissions Prediction

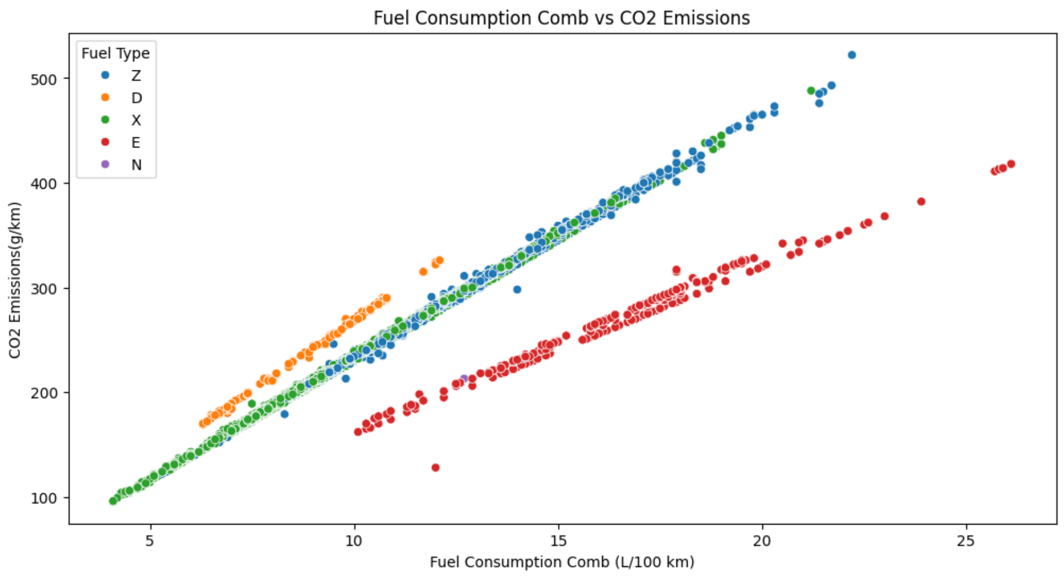
```
sns.pairplot(df[['Engine Size(L)', 'Cylinders', 'Fuel Consumption City (L/100 km)',
                 'Fuel Consumption Hwy (L/100 km)', 'Fuel Consumption Comb (L/100 km)',
                 'Fuel Consumption Comb (mpg)', 'CO2 Emissions(g/km)']])
plt.show()
```



Assignment : Carbon dioxide Emissions Prediction

```
plt.figure(figsize=(12, 6))
sns.scatterplot(x='Engine Size(L)', y='CO2 Emissions(g/km)', hue='Fuel Type', data=df)
plt.title('Engine Size vs CO2 Emissions')
plt.show()

plt.figure(figsize=(12, 6))
sns.scatterplot(x='Fuel Consumption Comb (L/100 km)', y='CO2 Emissions(g/km)', hue='Fuel Type', data=df)
plt.title('Fuel Consumption Comb vs CO2 Emissions')
plt.show()
```





Assignment : Carbon dioxide Emissions Prediction

```
X = df.drop('CO2 Emissions(g/km)', axis=1)
y = df['CO2 Emissions(g/km)']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

numeric_features = ['Engine Size(L)', 'Cylinders', 'Fuel Consumption City (L/100 km)',
                    'Fuel Consumption Hwy (L/100 km)', 'Fuel Consumption Comb (L/100 km)',
                    'Fuel Consumption Comb (mpg)’]
categorical_features = ['Brand', 'Model', 'Vehicle Class', 'Transmission', 'Fuel Type']
```

Assignment : Carbon dioxide Emissions Prediction

```
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
```

Create model with Pipeline

```
numeric_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='median')),
    ('scaler', StandardScaler())
])
```

- imputer: uses `SimpleImputer(strategy='median')` to handle missing values in the numerical columns by replacing them with the median value of the respective column.
- scaler: uses `StandardScaler()` to standardize the numerical features.

```
categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('onehot', OneHotEncoder(handle_unknown='ignore'))
])
```

- imputer: uses `SimpleImputer(strategy='most_frequent')` to handle missing values in the categorical columns by replacing them with the most frequent category in the respective column.
- onehot: uses `OneHotEncoder(handle_unknown='ignore')` to perform one-hot encoding on the categorical features. The handle_unknown='ignore' parameter allows the encoder to handle unseen categories during prediction by simply ignoring them.

Assignment : Carbon dioxide Emissions Prediction

```
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numeric_features),
        ('cat', categorical_transformer, categorical_features)
    ])
```

`preprocessor:` This is a ColumnTransformer object that combines the `numeric_transformer` and `categorical_transformer` to preprocess the entire dataset.

`('num', numeric_transformer, numeric_features)`
Applies the numeric_transformer to the columns listed in numeric_features.

`('cat', categorical_transformer, categorical_features)`
Applies the categorical_transformer to the columns listed in categorical_features.

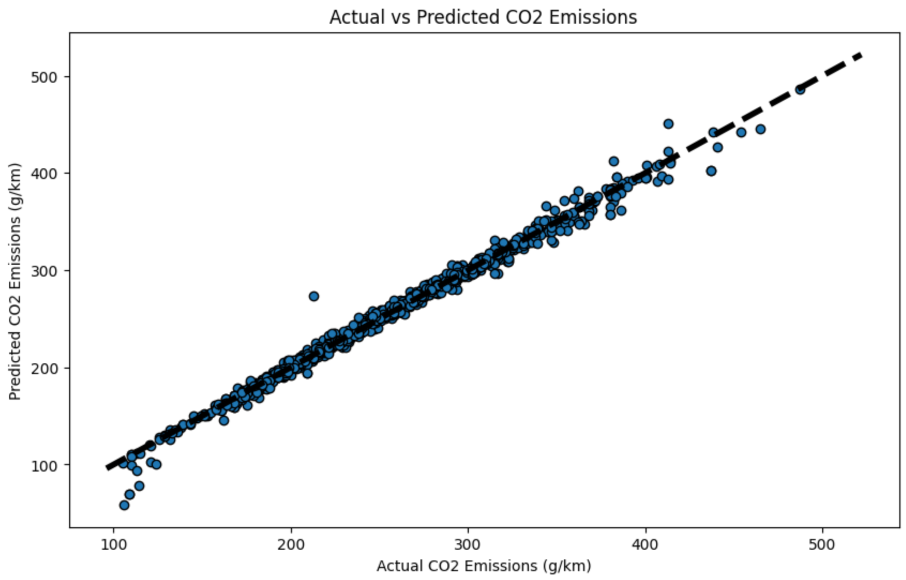
```
model = Pipeline(steps=[('preprocessor', preprocessor),
                        ('regressor', LinearRegression())])
```

`model:` This is a Pipeline object that defines the complete machine learning model.
`preprocessor:` applies the preprocessor to preprocess the input data.
`regressor:` uses `LinearRegression()` to train a linear regression model on the preprocessed data.

Assignment : Carbon dioxide Emissions Prediction

```python
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)

print(f'Mean Squared Error: {mse:.2f}')
print(f'R^2 Score: {r2:.2f}')
print(f'Root Mean Squared Error: {rmse:.2f}’)

plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred, edgecolors=(0, 0, 0))
plt.plot([y.min(), y.max()], [y.min(), y.max()], 'k--', lw=4)
plt.xlabel('Actual CO2 Emissions (g/km)')
plt.ylabel('Predicted CO2 Emissions (g/km)')
plt.title('Actual vs Predicted CO2 Emissions')
plt.show()
```

```
Mean Squared Error: 29.99
R^2 Score: 0.99
Root Mean Squared Error: 5.48
```

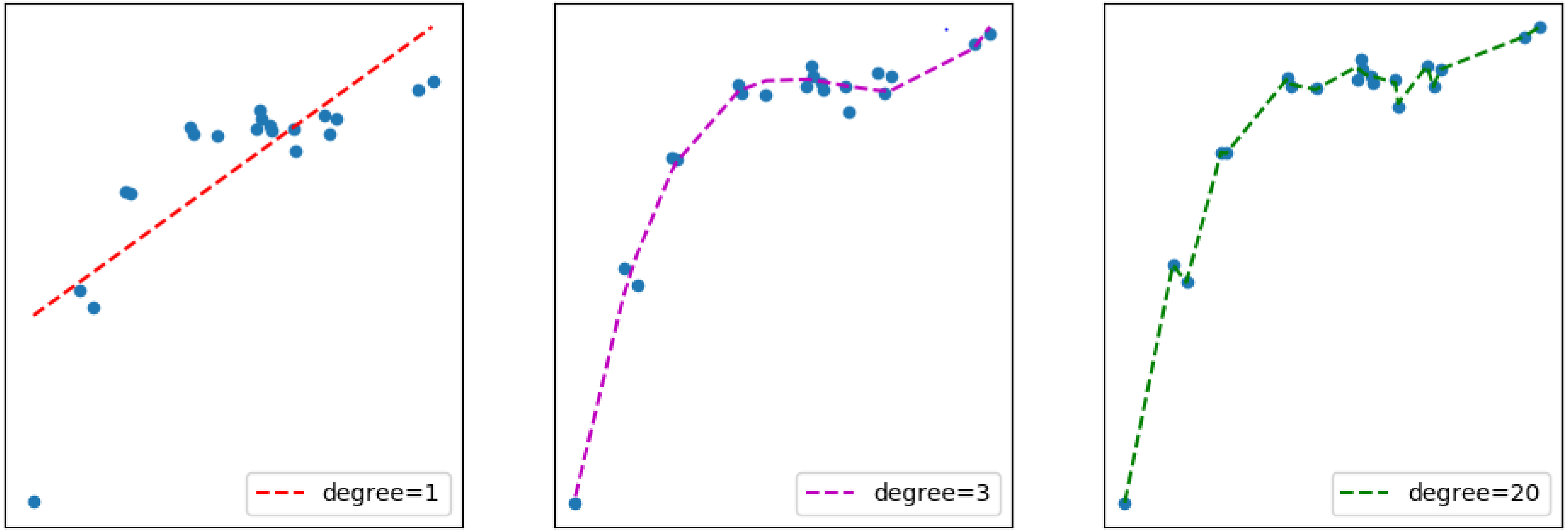


Polynomial Regression

Using when Non-Linear relation:

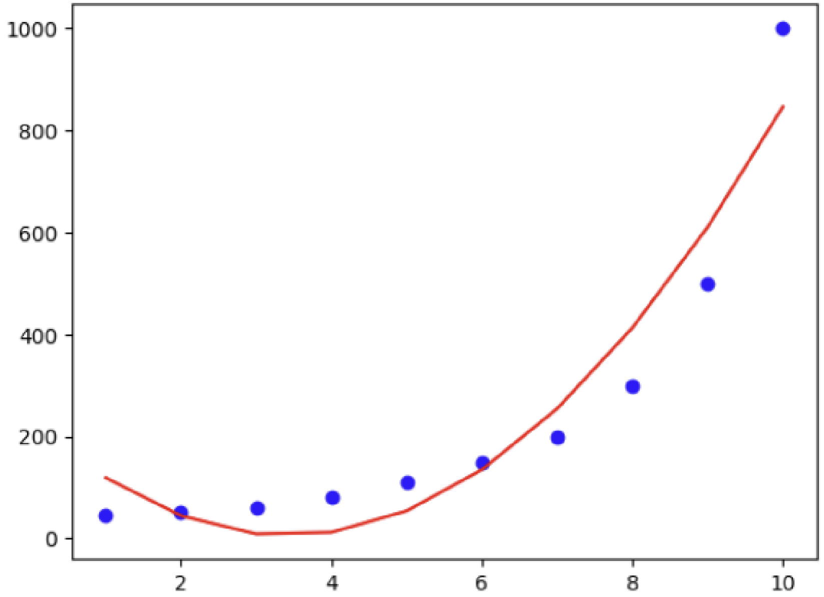
$$f_{\vec{w},b} = w_1 x^1 + w_2 x^2 + \cdots + w_n x^n$$

n = poly nominal degree



Code 7 : Plot polynomial regression

```python
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
import matplotlib.pyplot as plt
import numpy as np

pf = PolynomialFeatures(degree=2)  #1, 2, 3, 4
x = np.array(list(range(1, 11))).reshape(-1, 1)
x_poly= pf.fit_transform(x)

y = [45, 50, 60, 80, 110, 150, 200, 300, 500, 1000]

model = LinearRegression()
model.fit(x_poly, y)

y_predict = model.predict(x_poly)

plt.scatter(x, y, color='b')
plt.plot(x, y_predict, color='r')
plt.show()
```



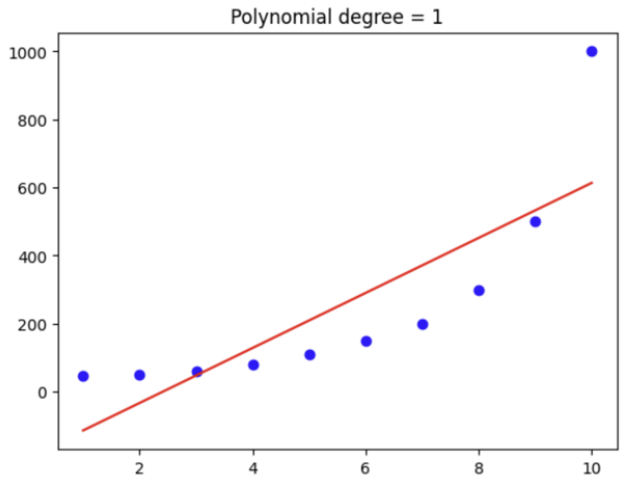
Exercise : from Code 7

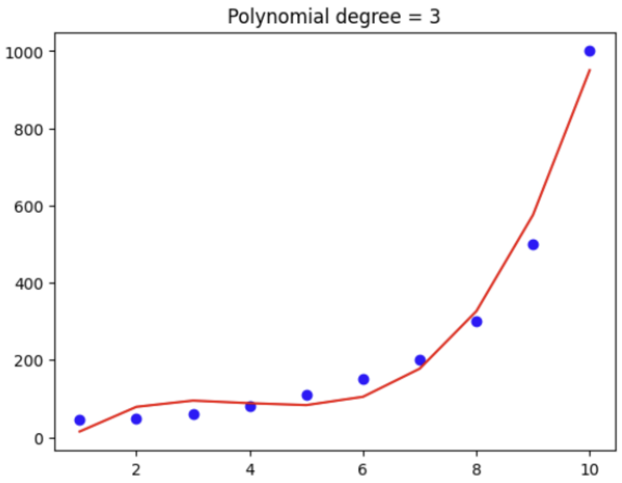
Write code for polynomial degree = 1, 3 and 10

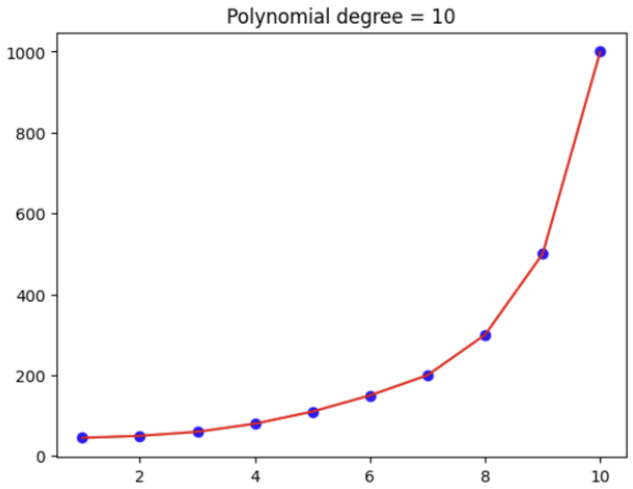
And determine R-squared (R2), MSE and RMSE

From Code 7

Plot of each Polynomial degrees

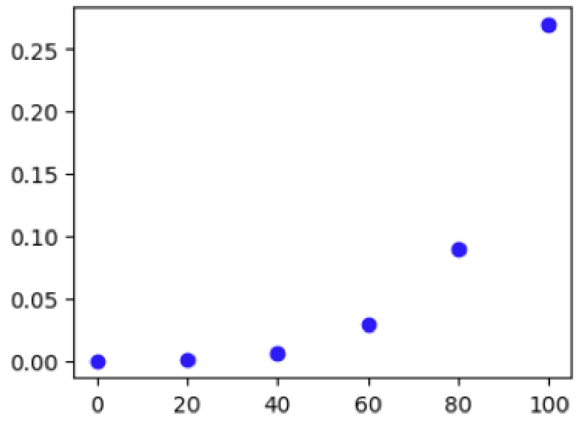






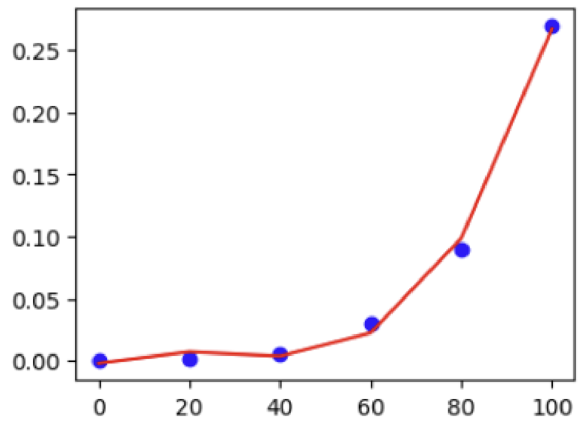
Code 8 : Predict machine pressure from temperature

```python
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt
import numpy as np
from sklearn.metrics import r2_score

x = [0, 20, 40, 60, 80, 100]
y = [0.0002, 0.0012, 0.0060, 0.0300, 0.0900, 0.2700]

plt.figure(figsize=(4, 3))
plt.scatter(x, y, color='b')
plt.show()

pf = PolynomialFeatures(degree=3)

x = np.array(x).reshape(-1, 1)
x_poly = pf.fit_transform(x)

model = LinearRegression()
model.fit(x_poly, y)
```



Code 8

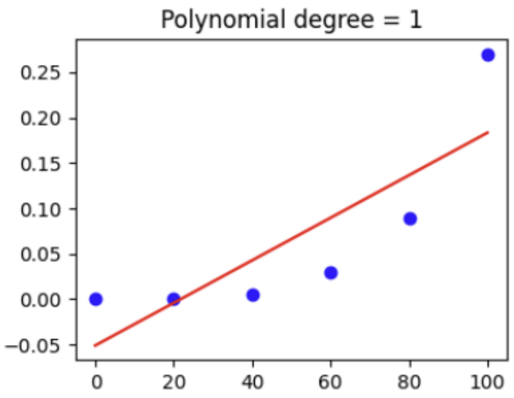
```
y_predict = model.predict(x_poly)

plt.figure(figsize=(4, 3))
plt.scatter(x, y, color='b')
plt.plot(x, y_predict, color='r')
plt.show()

score = r2_score(y, y_predict)
print("Model's Accuracy =",score)


'''
# predict machine psi when temp = 50, 70 and 95
x_predict = [[50], [70], [95]]
y_predict = model.predict(pf.transform(x_predict))

for (i, x_p) in enumerate(x_predict):
    pressure = '{:.4f}'.format(y_predict[i])
    print(f'Temperature = {x_p[0]}, Pressure (psi) = {pressure}')
'''
```




```
Model's Accuracy = 0.9966691251761722
```

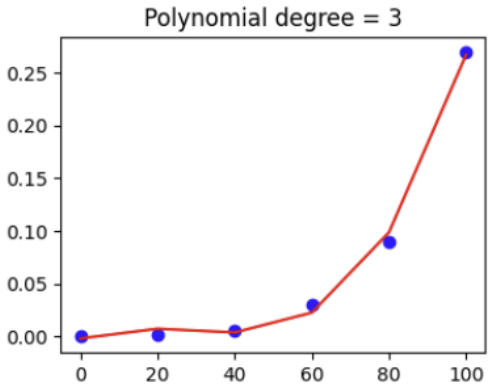
```
Temperature = 50, Pressure (psi) = 0.0081
Temperature = 70, Pressure (psi) = 0.0512
Temperature = 95, Pressure (psi) = 0.2145
```

From code 8



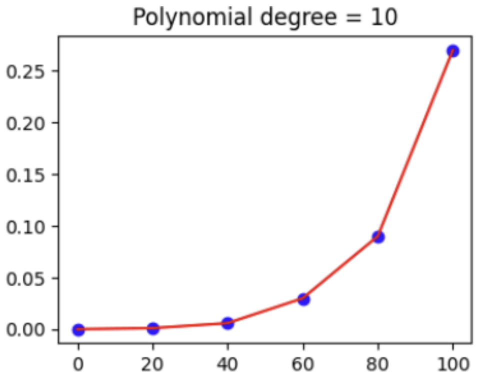
Model's Accuracy = 0.6903499726039811

Underfit
(High Bias)



Model's Accuracy = 0.9966691251761722

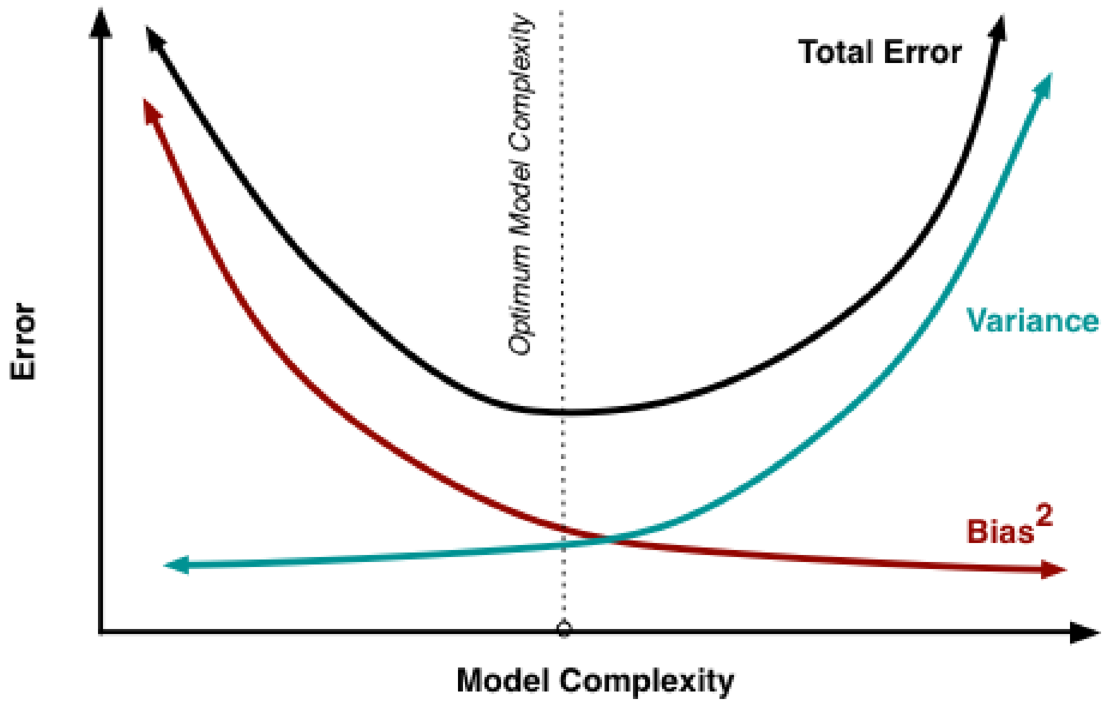
Good fit



Model's Accuracy = 1.0

Overfit
(High Variance)

The problem of Overfitting



Bias refers to the error due to the model's simplistic assumptions in fitting the data. A high bias means that the model is unable to capture the patterns in the data and this results in under-fitting.

Variance refers to the error due to the complex model trying to fit the data. High variance means the model passes through most of the data points and it results in over-fitting the data.

Addressing overfitting

1. Collect more training set samples
2. Select feature to include/exclude
   - All features + insufficient data = Overfitting
3. Regularization technique

Assignment : Car Price Prediction Model

The primary goal of this project is to develop a machine learning model capable of accurately predicting the price of cars based on various attributes such as make, model, year, mileage, and other relevant features.

Data Source: Kaggle

Steps Overview

1. Data Preprocessing: Handling missing data, outliers, and encoding categorical features.
2. Exploratory Data Analysis (EDA): Visualizing feature distributions and relationships to the target variable (car price).
3. Training machine learning models : Linear Regression
4. Evaluation: Assessing models using performance metrics such as R^2 and Mean Squared Error

Assignment (key)

```python
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
```

```python
df = pd.read_csv('CarPrice_Assignment.csv')
```

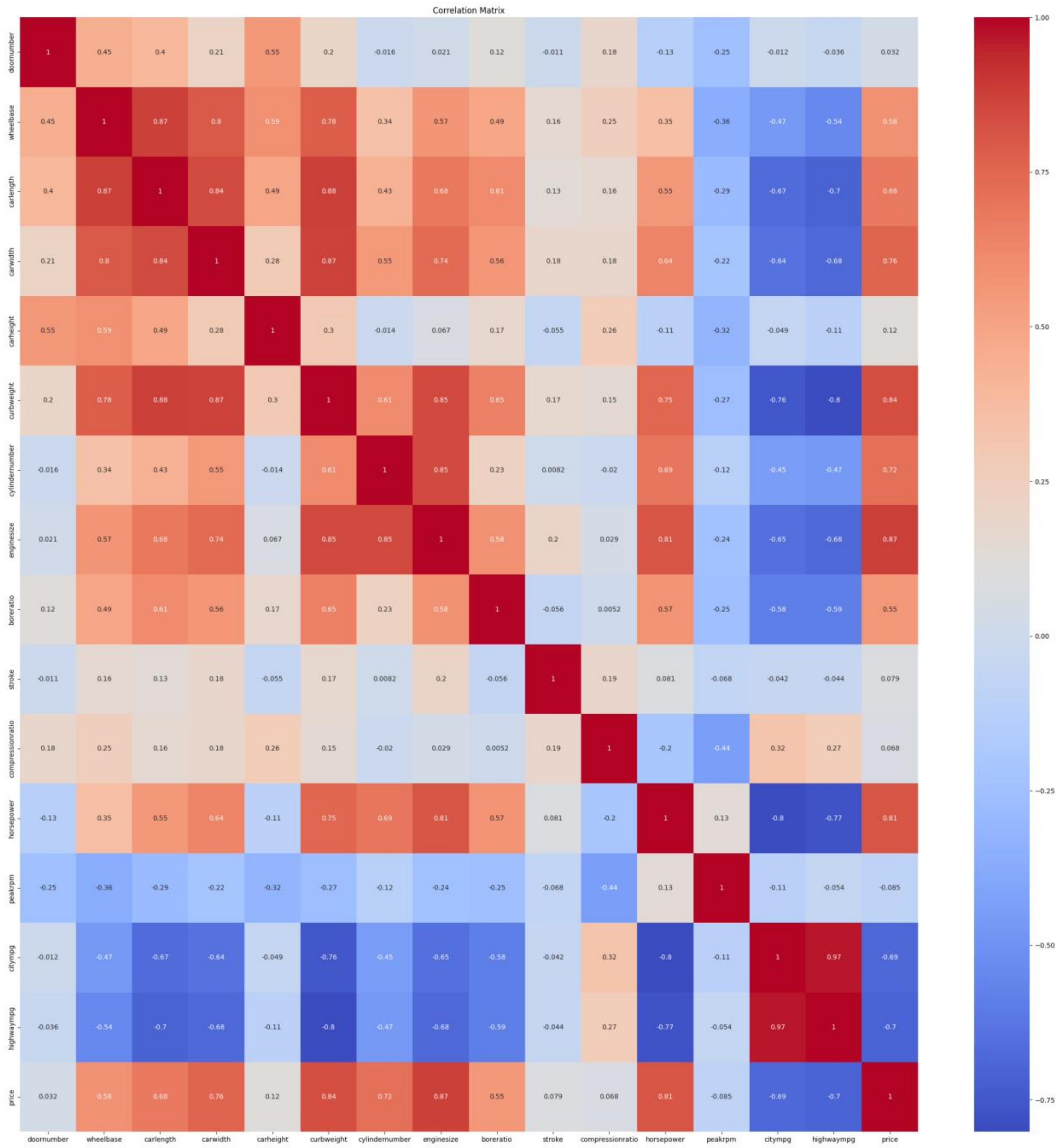
```python
df.head()
```

	car_ID	symboling	CarName	fueltype	aspiration	doornumber	carbody	drivewheel	enginelocation	wheelbase	carlength	carwidth	carheight	curbweight	enginetype	cylindernumber	enginesize	fuelsystem	b
0	1	3	alfa-romero giulia	gas	std	two	convertible	rwd	front	88.6	168.8	64.1	48.8	2548	dohc	four	130	mpfi	
1	2	3	alfa-romero stelvio	gas	std	two	convertible	rwd	front	88.6	168.8	64.1	48.8	2548	dohc	four	130	mpfi	
2	3	1	alfa-romero Quadrifoglio	gas	std	two	hatchback	rwd	front	94.5	171.2	65.5	52.4	2823	ohcv	six	152	mpfi	
3	4	2	audi 100 ls	gas	std	four	sedan	fwd	front	99.8	176.6	66.2	54.3	2337	ohc	four	109	mpfi	
4	5	2	audi 100ls	gas	std	four	sedan	4wd	front	99.4	176.6	66.4	54.3	2824	ohc	five	136	mpfi	

```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 205 entries, 0 to 204
Data columns (total 26 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   car_ID            205 non-null    int64
 1   symboling         205 non-null    int64
 2   CarName           205 non-null    object
 3   fueltype          205 non-null    object
 4   aspiration        205 non-null    object
 5   doornumber        205 non-null    object
 6   carbody           205 non-null    object
 7   drivewheel        205 non-null    object
 8   enginelocation    205 non-null    object
 9   wheelbase         205 non-null    float64
 10  carlength         205 non-null    float64
 11  carwidth          205 non-null    float64
 12  carheight         205 non-null    float64
 13  curbweight        205 non-null    int64
 14  enginetype        205 non-null    object
 15  cylindernumber    205 non-null    object
 16  enginesize        205 non-null    int64
 17  fuelsystem        205 non-null    object
 18  boreratio         205 non-null    float64
 19  stroke            205 non-null    float64
...
 24  highwaympg        205 non-null    int64
 25  price             205 non-null    float64
dtypes: float64(8), int64(8), object(10)
memory usage: 41.8+ KB
```

```
df.drop(columns = ['car_ID', 'symboling', 'CarName',  ], inplace = True)
# df.head()
df['doornumber'] = df['doornumber'].replace({'four': 4, 'two': 2}).astype('int64')
df['cylindernumber'] = df['cylindernumber'].replace({'four': 4, 'six': 6, 'five': 5, 'eight': 8, 'two': 2,
'three': 3, 'twelve': 12}).astype('int64’)

correlation_matrix = df.select_dtypes(include=['int64', 'float64']).corr()
plt.figure(figsize=(30,30))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Matrix')
plt.show()
```



Correlation Matrix using: Seaborn

python install package
pip install seaborn

# import library
import seaborn as sns

```python
corr_matrix = df.select_dtypes(include=['int64', 'float64']).corr()
# df.select_dtypes(include=['int64', 'float64'])
corr_matrix['price'].sort_values(ascending=False)

final_coulmns = ['enginesize', 'curbweight', 'horsepower', 'carwidth',
'cylindernumber', 'carlength', 'wheelbase', 'boreratio', 'citympg',
'highwaympg','fueltype', 'aspiration', 'carbody', 'drivewheel',
'enginelocation', 'enginetype', 'price']
car_df = df[final_coulmns]
# car_df.head()
```

```
price               1.000000
enginesize          0.874145
curbweight          0.835305
horsepower          0.808139
carwidth            0.759325
cylindernumber      0.718305
carlength           0.682920
wheelbase           0.577816
boreratio           0.553173
carheight           0.119336
stroke              0.079443
compressionratio    0.067984
doornumber          0.031835
peakrpm            -0.085267
citympg            -0.685751
highwaympg         -0.697599
Name: price, dtype: float64
```

For some categorical data like:

```
['fueltype', 'aspiration', 'carbody', 'drivewheel', 'enginelocation', 'enginetype']
```

Encode it using OneHot Encoding using: `pd.get_dummies`

```
car_df = pd.get_dummies(car_df, columns=['fueltype', 'aspiration', 'carbody', 'drivewheel', 'enginelocation',
                  'enginetype'], drop_first=True, dtype='int64')
# car_df.head()
```

Before encoding:

	enginesize	curbweight	horsepower	carwidth	cylindernumber	carlength	wheelbase	boreratio	citympg	highwaympg	fueltype	aspiration	carbody	drivewheel	enginelocation	enginetype	price
0	130	2548	111	64.1	4	168.8	88.6	3.47	21	27	gas	std	convertible	rwd	front	dohc	13495.0
1	130	2548	111	64.1	4	168.8	88.6	3.47	21	27	gas	std	convertible	rwd	front	dohc	16500.0
2	152	2823	154	65.5	6	171.2	94.5	2.68	19	26	gas	std	hatchback	rwd	front	ohcv	16500.0
3	109	2337	102	66.2	4	176.6	99.8	3.19	24	30	gas	std	sedan	fwd	front	ohc	13950.0
4	136	2824	115	66.4	5	176.6	99.4	3.19	18	22	gas	std	sedan	4wd	front	ohc	17450.0

After encoding:

	enginesize	curbweight	horsepower	carwidth	cylindernumber	carlength	wheelbase	boreratio	citympg	highwaympg	price	fueltype_gas	aspiration_turbo	carbody_hardtop	carbody_hatchback	carbody_sedan
0	130	2548	111	64.1	4	168.8	88.6	3.47	21	27	13495.0	1	0	0	0	0
1	130	2548	111	64.1	4	168.8	88.6	3.47	21	27	16500.0	1	0	0	0	0
2	152	2823	154	65.5	6	171.2	94.5	2.68	19	26	16500.0	1	0	0	1	0
3	109	2337	102	66.2	4	176.6	99.8	3.19	24	30	13950.0	1	0	0	0	1
4	136	2824	115	66.4	5	176.6	99.4	3.19	18	22	17450.0	1	0	0	0	1

carbody_wagon	drivewheel_fwd	drivewheel_rwd	enginelocation_rear	enginetype_dohcv	enginetype_l	enginetype_ohc	enginetype_ohcf	enginetype_ohcv	enginetype_rotor
0	0	1	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	1	0
0	1	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0

```python
x = car_df.drop(['price'], axis=1)
y = car_df[['price’]]

x_train, x_test, y_train, y_test =
train_test_split(x.values, y.values, random_state=42,
test_size=0.2)

# Print the shape of the splits
print(f"Train features shape: {x_train.shape}")
print(f"Test features shape: {x_test.shape}")
print(f"Train labels shape: {y_train.shape}")
print(f"Test labels shape: {y_test.shape}")

scale = StandardScaler()
x_train_sc = scale.fit_transform(x_train)
x_test_sc = scale.fit_transform(x_test)
```

```
Train features shape: (164, 25)
Test features shape: (41, 25)
Train labels shape: (164, 1)
Test labels shape: (41, 1)
```

```python
def error_metrics(y_train_true, y_train_pred, y_test_true, y_test_pred):
    errors = {}
    # Errors for train data
    errors["Train_MSE"] = mean_squared_error(y_train_true, y_train_pred)
    errors["Train_RMSE"] = np.sqrt(errors["Train_MSE"])
    errors["Train_R2_Score"] = r2_score(y_train_true, y_train_pred)
    # Errors for test data
    errors["Test_MSE"] = mean_squared_error(y_test_true, y_test_pred)
    errors["Test_RMSE"] = np.sqrt(errors["Test_MSE"])
    errors["Test_R2_Score"] = r2_score(y_test_true, y_test_pred)
    return errors
model_evaluation = []

model = LinearRegression()
model.fit(x_train_sc, y_train)
y_train_pred = model.predict(x_train_sc)
y_test_pred = model.predict(x_test_sc)
error_model = error_metrics(y_train, y_train_pred, y_test, y_test_pred)
error_model
```

```
{'Train_MSE': 4795967.326218939,
 'Train_RMSE': np.float64(2189.9697089729207),
 'Train_R2_Score': 0.9195818034920166,
 'Test_MSE': 17626555.2325218,
 'Test_RMSE': np.float64(4198.399127348637),
 'Test_R2_Score': 0.7767208328621653}
```

```python
error_model['Model Name']='Linear Regression'
error_model
model_evaluation = (pd.DataFrame([error_model]))
model_evaluation
```

	Train_MSE	Train_RMSE	Train_R2_Score	Test_MSE	Test_RMSE	Test_R2_Score	Model Name
0	4.795967e+06	2189.969709	0.919582	1.762656e+07	4198.399127	0.776721	Linear Regression

SUT

SURANAREE UNIVERSITY OF TECHNOLOGY

INSTITUTE OF ENGINEERING

Mechatronics

Modern Automotive