Unit-5 Linear Regression

Linear regression is a statistical method used to model the relationship between a dependent variable and one or more independent variables.

The goal of linear regression is to find the best-fitting line (or hyperplane in higher dimensions) that describes the relationship between the variables.

Key Concepts

Dependent Variable (Y): The outcome or the variable we are trying to predict or explain.

Independent Variables (X): The predictors or the variables we use to predict the dependent variable.

Linear Relationship: The relationship between the dependent and independent variables is assumed to be linear.

Simple Linear Regression

In simple linear regression, we model the relationship between two variables by fitting a linear equation to the observed data:

$$Y = \beta_0 + \beta_1 X + \epsilon$$

- Y: Dependent variable
- X: Independent variable
- β_0: Intercept
- β_1: Slope of the line
- ϵ: Error term (residual)

Multiple Linear Regression

In multiple linear regression, we extend the concept to include multiple independent variables:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_n X_n + \epsilon$$

- $X_1, X_2, \ldots, X_n$: Independent variables
- β_0: Intercept
- $\beta_1, \beta_2, \ldots, \beta_n$: Coefficients for each independent variable
- ϵ: Error term

Polynomial Regression

Polynomial regression is an extension of linear regression that models the relationship between the independent variable x and the dependent variable y as an n-degree polynomial. Instead of fitting a straight line to the data, polynomial regression fits a curve, which can capture more complex patterns.

Mathematical Formulation

The polynomial regression model of degree n can be expressed as:

$$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \cdots + \beta_n x^n + \epsilon$$

Where:

- y is the dependent variable.
- x is the independent variable.
- $\beta_0, \beta_1, \beta_2, \ldots, \beta_n$ are the coefficients of the polynomial.
- ϵ is the error term.

Steps to Perform Polynomial Regression

1. **Data Preparation:**
   - Collect the data and preprocess it (handle missing values, normalization, etc.).
2. **Feature Engineering:**
   - Generate polynomial features from the original data. For example, if you have a single feature x and you want to fit a polynomial of degree 3, you create new features: x, x^2, x^3.
3. **Model Fitting:**
   - Use a linear regression model to fit the polynomial features to the target variable y.
4. **Evaluation:**
   - Evaluate the model using appropriate metrics like R-squared, Mean Squared Error (MSE), etc.

Simple Linear Regression

Model-1 - Book1.csv

In [3]:
```python
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

dataset=pd.read_csv('Book1.csv')
dataset
```

Out[3]:

	cgpa	package
0	6.89	3.26
1	5.12	1.98
2	7.82	3.25
3	7.42	3.67
4	6.94	3.57
...	...	...
195	6.93	2.46
196	5.89	2.57
197	7.21	3.24
198	7.63	3.96
199	6.22	2.33

200 rows × 2 columns

In [4]:
```python
dataset.isna().sum()
```

Out[4]:
```
cgpa       0
package    0
dtype: int64
```

In [5]:
```python
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 2 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   cgpa     200 non-null    float64
 1   package  200 non-null    float64
dtypes: float64(2)
memory usage: 3.2 KB
```

In [6]:
```python
x=dataset.iloc[:,0:1]
y=dataset.iloc[:,-1]
```

In [7]:
```python
print(x.shape)
print(y.shape)
```

```
(200, 1)
(200,)
```

In [8]:
```python
print(type(x))
print(type(y))
```

```
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.series.Series'>
```

In [21]:
```
#Split the data in training & testing
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=1)

print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(160, 1)
(40, 1)
(160,)
(40,)
```

In [22]:
```
#Creating LinearRegression Model

from sklearn.linear_model import LinearRegression
lr=LinearRegression()

lr.fit(x_train,y_train)

y_pred=lr.predict(x_test)
print(y_pred)
```

```
[2.9383335  4.36894346 3.18258398 1.89736121 3.49662031 3.35123312
 2.76968435 2.94996447 3.07208971 3.94441286 3.57222165 2.94996447
 2.75805338 2.64755911 3.67108494 3.2174769  3.97930579 2.90925606
 2.19395108 3.31052471 4.29915761 2.8918096  1.87409926 2.30444534
 3.62456104 2.12998071 3.9269664  2.36841571 1.5716939  2.06601035
 2.31026083 3.6885314  3.5024358  3.03719679 2.57195777 2.39167766
 3.170953   3.82228762 3.15932203 2.94414898]
```

In [23]:
```
diff=pd.DataFrame({"Actual":y_test,"Predicted":y_pred})
print(diff)
```

```
     Actual  Predicted
58     3.09   2.938333
40     4.02   4.368943
34     3.42   3.182584
102    1.37   1.897361
184    3.14   3.496620
198    3.96   3.351233
95     2.79   2.769684
4      3.57   2.949964
29     3.49   3.072090
168    3.52   3.944413
171    3.76   3.572222
18     2.98   2.949964
11     2.60   2.758053
89     2.72   2.647559
110    3.76   3.671085
118    2.88   3.217477
159    4.08   3.979306
35     2.87   2.909256
136    2.10   2.193951
59     3.31   3.310525
51     3.79   4.299158
16     2.35   2.891810
44     1.86   1.874099
94     2.42   2.304445
31     3.89   3.624561
```

```
162    2.55   2.129981
38     4.36   3.926966
28     2.24   2.368416
193    1.94   1.571694
27     2.16   2.066010
47     3.26   2.310261
165    4.08   3.688531
194    3.67   3.502436
177    3.64   3.037197
176    3.23   2.571958
97     2.84   2.391678
174    2.99   3.170953
73     4.03   3.822288
69     2.94   3.159322
172    2.51   2.944149
```

In [24]:

```
print("Coefficient:",lr.coef_)
print("Intercept:",lr.intercept_)
```

```
Coefficient: [0.58154877]
Intercept: -1.0859839580358042
```

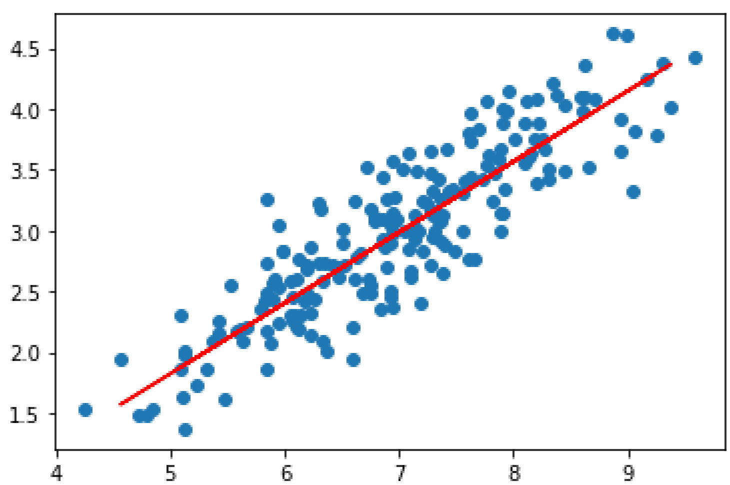
In [25]:

```
from sklearn import metrics
print("MAE: ",metrics.mean_absolute_error(y_test,y_pred))
print("MSE: ",metrics.mean_squared_error(y_test,y_pred))
print("R2 Score: ",metrics.mean_squared_error(y_test,y_pred))
```

```
MAE:  0.2993118859331679
MSE:  0.1370062519255721
R2 Score:  0.1370062519255721
```

In [26]:

```
plt.scatter(dataset["cgpa"],dataset["package"])
plt.plot(x_test,y_pred,color="red")
plt.show()
```



Model-2 Olympic100m.csv

In [173…

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

df=pd.read_csv("olympic100m.csv")
df.head(10)
```

Out[173…

	year	time
0	1896	12.0
1	1900	11.0
2	1904	11.0
3	1906	11.2
4	1908	10.8
5	1912	10.8
6	1920	10.8
7	1924	10.6
8	1928	10.8
9	1932	10.3

In [174…

```
df.isna().sum()
```

Out[174…

```
year    0
time    0
dtype: int64
```

In [175…

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 29 entries, 0 to 28
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   year    29 non-null     int64
 1   time    29 non-null     float64
dtypes: float64(1), int64(1)
memory usage: 592.0 bytes
```

In [176…

```
# Format data into correct shape
x = df['year']
x_train = np.array(x).reshape((-1, 1)) #To make it 2 dimension array (Data Frame)
x_train.shape
```

Out[176…

```
(29, 1)
```

In [179…

```
# Format data into correct shape
y = df['time']
y_train = np.array(y) #To make it 1 dimension array (Data Frame)
y_train.shape
```

Out[179…

```
(29,)
```

In [180…

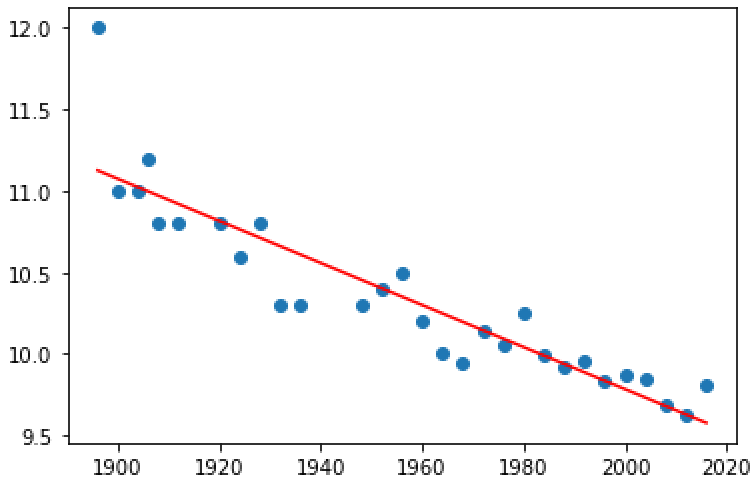
```
from sklearn.linear_model import LinearRegression

# Let's create the model object using LinearRegression
model = LinearRegression()

# Fit our model to our input data x and y
model.fit(x_train, y_train)
```

```
y_pred = model.predict(x_train) #Same training data used for prediction
print(y_pred)
```

```
[11.12455601 11.07301534 11.02147467 10.99570434 10.969934   10.91839333
 10.81531199 10.76377132 10.71223065 10.66068998 10.60914931 10.45452731
 10.40298664 10.35144597 10.2999053  10.24836463 10.19682396 10.14528329
 10.09374262 10.04220195  9.99066128  9.93912061  9.88757994  9.83603927
  9.7844986   9.73295793  9.68141726  9.62987659  9.57833592]
```

In [181…

```
plt.scatter(x_train, y_train)
plt.plot(x, y_pred, color='r')
plt.show()
```



In [182…

```
# Predict for 2020 Olympics

prediction = model.predict([[2020]])
print("The Time prediction for olympic 2020 :",prediction)
```

```
The Time prediction for olympic 2020 : [9.52679525]
```

Multiple Linear Regression

Model-3 - car data.csv

In [63]:

```
#Importing necessary libraries and understanding the data
import pandas as pd
import numpy as np
df=pd.read_csv('car data.csv')
df
```

Out[63]:

	Car_Name	Year	Selling_Price	Present_Price	Kms_Driven	Fuel_Type	Seller_Type	Transmission	Ov
0	ritz	2014	3.35	5.59	27000	Petrol	Dealer	Manual	
1	sx4	2013	4.75	9.54	43000	Diesel	Dealer	Manual	
2	ciaz	2017	7.25	9.85	6900	Petrol	Dealer	Manual	
3	wagon r	2011	2.85	4.15	5200	Petrol	Dealer	Manual	
4	swift	2014	4.60	6.87	42450	Diesel	Dealer	Manual	
...	...	...	...	...	...	...	...	...	

	Car_Name	Year	Selling_Price	Present_Price	Kms_Driven	Fuel_Type	Seller_Type	Transmission	Ov
296	city	2016	9.50	11.60	33988	Diesel	Dealer	Manual	
297	brio	2015	4.00	5.90	60000	Petrol	Dealer	Manual	
298	city	2009	3.35	11.00	87934	Petrol	Dealer	Manual	
299	city	2017	11.50	12.50	9000	Diesel	Dealer	Manual	
300	brio	2016	5.30	5.90	5464	Petrol	Dealer	Manual	

301 rows × 9 columns

Reading and understanding the dataset

In [64]:

```
df.head(20)
```

Out[64]:

	Car_Name	Year	Selling_Price	Present_Price	Kms_Driven	Fuel_Type	Seller_Type	Transmission	Ow
0	ritz	2014	3.35	5.59	27000	Petrol	Dealer	Manual	
1	sx4	2013	4.75	9.54	43000	Diesel	Dealer	Manual	
2	ciaz	2017	7.25	9.85	6900	Petrol	Dealer	Manual	
3	wagon r	2011	2.85	4.15	5200	Petrol	Dealer	Manual	
4	swift	2014	4.60	6.87	42450	Diesel	Dealer	Manual	
5	vitara brezza	2018	9.25	9.83	2071	Diesel	Dealer	Manual	
6	ciaz	2015	6.75	8.12	18796	Petrol	Dealer	Manual	
7	s cross	2015	6.50	8.61	33429	Diesel	Dealer	Manual	
8	ciaz	2016	8.75	8.89	20273	Diesel	Dealer	Manual	
9	ciaz	2015	7.45	8.92	42367	Diesel	Dealer	Manual	
10	alto 800	2017	2.85	3.60	2135	Petrol	Dealer	Manual	
11	ciaz	2015	6.85	10.38	51000	Diesel	Dealer	Manual	
12	ciaz	2015	7.50	9.94	15000	Petrol	Dealer	Automatic	
13	ertiga	2015	6.10	7.71	26000	Petrol	Dealer	Manual	
14	dzire	2009	2.25	7.21	77427	Petrol	Dealer	Manual	
15	ertiga	2016	7.75	10.79	43000	Diesel	Dealer	Manual	
16	ertiga	2015	7.25	10.79	41678	Diesel	Dealer	Manual	
17	ertiga	2016	7.75	10.79	43000	Diesel	Dealer	Manual	
18	wagon r	2015	3.25	5.09	35500	CNG	Dealer	Manual	
19	sx4	2010	2.65	7.98	41442	Petrol	Dealer	Manual	

In [65]:

```
df.shape
```

Out[65]: (301, 9)

In [66]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 301 entries, 0 to 300
Data columns (total 9 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Car_Name       301 non-null    object 
 1   Year           301 non-null    int64  
 2   Selling_Price  301 non-null    float64
 3   Present_Price  301 non-null    float64
 4   Kms_Driven     301 non-null    int64  
 5   Fuel_Type      301 non-null    object 
 6   Seller_Type    301 non-null    object 
 7   Transmission   301 non-null    object 
 8   Owner          301 non-null    int64  
dtypes: float64(2), int64(3), object(4)
memory usage: 21.3+ KB
```

In [67]:
```
df.describe()
```

Out[67]:

	Year	Selling_Price	Present_Price	Kms_Driven	Owner
count	301.000000	301.000000	301.000000	301.000000	301.000000
mean	2013.627907	4.661296	7.628472	36947.205980	0.043189
std	2.891554	5.082812	8.644115	38886.883882	0.247915
min	2003.000000	0.100000	0.320000	500.000000	0.000000
25%	2012.000000	0.900000	1.200000	15000.000000	0.000000
50%	2014.000000	3.600000	6.400000	32000.000000	0.000000
75%	2016.000000	6.000000	9.900000	48767.000000	0.000000
max	2018.000000	35.000000	92.600000	500000.000000	3.000000

In [68]:
```
df.isna().sum()
```

Out[68]:
```
Car_Name         0
Year             0
Selling_Price    0
Present_Price    0
Kms_Driven       0
Fuel_Type        0
Seller_Type      0
Transmission     0
Owner            0
dtype: int64
```

Data Preprocessing

In [69]:
```
#Adding New Feature for Age - Feature Engineering
df['Age']=2024-df['Year']

#Drop Year Column
df.drop('Year', axis = 1, inplace=True)
```

In [70]:
```
df
```

Out[70]:

	Car_Name	Selling_Price	Present_Price	Kms_Driven	Fuel_Type	Seller_Type	Transmission	Owner
0	ritz	3.35	5.59	27000	Petrol	Dealer	Manual	0
1	sx4	4.75	9.54	43000	Diesel	Dealer	Manual	0
2	ciaz	7.25	9.85	6900	Petrol	Dealer	Manual	0
3	wagon r	2.85	4.15	5200	Petrol	Dealer	Manual	0
4	swift	4.60	6.87	42450	Diesel	Dealer	Manual	0
...	...	...	...	...	...	...	...	...
296	city	9.50	11.60	33988	Diesel	Dealer	Manual	0
297	brio	4.00	5.90	60000	Petrol	Dealer	Manual	0
298	city	3.35	11.00	87934	Petrol	Dealer	Manual	0
299	city	11.50	12.50	9000	Diesel	Dealer	Manual	0
300	brio	5.30	5.90	5464	Petrol	Dealer	Manual	0

301 rows × 9 columns

Data Preparation

Creating dummie variables for categorical features

In [71]:
```
df.drop(labels='Car_Name', axis=1, inplace=True)
df.head()
```

Out[71]:

	Selling_Price	Present_Price	Kms_Driven	Fuel_Type	Seller_Type	Transmission	Owner	Age
0	3.35	5.59	27000	Petrol	Dealer	Manual	0	10
1	4.75	9.54	43000	Diesel	Dealer	Manual	0	11
2	7.25	9.85	6900	Petrol	Dealer	Manual	0	7
3	2.85	4.15	5200	Petrol	Dealer	Manual	0	13
4	4.60	6.87	42450	Diesel	Dealer	Manual	0	10

In [ ]:
```
#Alternate Methods for drop Car_Name
df.drop('Car_Name', axis=1, inplace=True)
```

In [ ]:
```
#Alternate Methods for drop Car_Name
df.drop(columns='Car_Name', inplace=True)
```

In [72]:
```
#One Hot Encoding - get_dummies function of pandas can be used to dummy code categorica
df=pd.get_dummies(data=df, drop_first=True)
df.head()
```

Out[72]:

	Selling_Price	Present_Price	Kms_Driven	Owner	Age	Fuel_Type_Diesel	Fuel_Type_Petrol	Seller_Type_
0	3.35	5.59	27000	0	10	0	1	
1	4.75	9.54	43000	0	11	1	0	
2	7.25	9.85	6900	0	7	0	1	
3	2.85	4.15	5200	0	13	0	1	
4	4.60	6.87	42450	0	10	1	0	

In [73]:
```
df.columns
```

Out[73]:
```
Index(['Selling_Price', 'Present_Price', 'Kms_Driven', 'Owner', 'Age',
       'Fuel_Type_Diesel', 'Fuel_Type_Petrol', 'Seller_Type_Individual',
       'Transmission_Manual'],
      dtype='object')
```

Splitting dataset into train and test subsets

In [74]:
```
#y-prediction so dataseries , x- dataframe because multiple columns
y=df['Selling_Price']
x=df.drop('Selling_Price', axis=1)
```

In [75]:
```
x
```

Out[75]:

	Present_Price	Kms_Driven	Owner	Age	Fuel_Type_Diesel	Fuel_Type_Petrol	Seller_Type_Individual
0	5.59	27000	0	10	0	1	0
1	9.54	43000	0	11	1	0	0
2	9.85	6900	0	7	0	1	0
3	4.15	5200	0	13	0	1	0
4	6.87	42450	0	10	1	0	0
...	...	...	...	...	...	...	...
296	11.60	33988	0	8	1	0	0
297	5.90	60000	0	9	0	1	0
298	11.00	87934	0	15	0	1	0
299	12.50	9000	0	7	1	0	0
300	5.90	5464	0	8	0	1	0

301 rows × 8 columns

In [76]:
```
y
```

Out[76]:
```
0      3.35
1      4.75
2      7.25
3      2.85
4      4.60
```

```
       ...
296      9.50
297      4.00
298      3.35
299     11.50
300      5.30
Name: Selling_Price, Length: 301, dtype: float64
```

In [77]:
```
x.shape
```

Out[77]:
```
(301, 8)
```

In [78]:
```
y.shape
```

Out[78]:
```
(301,)
```

In [79]:
```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x,y, test_size=0.2, random_state=1)
```

In [80]:
```
print("x train: ", x_train.shape)
print("x test: ", x_test.shape)
print("y train: ", y_train.shape)
print("y test: ",y_test.shape)
```

```
x train:  (240, 8)
x test:  (61, 8)
y train:  (240,)
y test:  (61,)
```

Creating models (Evaluation)

In [81]:
```
from sklearn.linear_model import LinearRegression
lm = LinearRegression()

model = lm.fit(x_train, y_train)

y_pred = model.predict(x_test)
print(y_pred)
```

```
[ 7.86273200e+00  2.96828691e+00 -5.90305107e-01  4.21335952e+00
  4.83175534e-01  5.82053813e+00  1.95679784e+00  2.55809661e+00
  7.70870833e+00  9.78692192e-01  8.13484343e+00  3.51207180e+00
  4.90669281e+00  4.63905587e+00 -2.15886643e+00  3.13874624e+00
  7.98256903e+00  6.75937638e+00  6.90426580e+00  8.01440587e+00
  4.31168610e+00  4.00336757e+00  1.13040883e+01  8.07939189e+00
  9.54399823e+00  3.52133877e+00  3.80609808e+00  1.06074722e+00
 -6.01732475e-01 -6.19712043e-01  1.32818516e-03 -1.28500691e+00
  4.28533553e+00  2.06769487e+01  1.87563232e+01  4.27292100e+00
  3.48602852e+00  1.66739677e+00 -4.38707073e-02  5.78536030e+00
  8.03940428e+00  9.88367483e+00  4.09684249e-01  6.07997517e+00
  5.88038915e+00  4.32745252e+00  7.37534505e+00  5.86171335e+00
  8.21129880e+00  1.65455816e+00  3.83033706e+00  1.75047060e+00
  2.51406796e+00  4.20404709e+00  1.48991546e+00 -3.44185843e+00
  2.04806215e+01  6.62503544e-01  5.40027412e+00  5.65856241e+00
  6.47876234e-01]
```

In [82]:
```
diff=pd.DataFrame({"Actual":y_test,"Predicted":y_pred})
print(diff)
```

```
     Actual  Predicted
285    7.40   7.862732
248    4.00   2.968287
```

```
150    0.50  -0.590305
217    3.15   4.213360
107    1.25   0.483176
..      ...        ...
62    18.75  20.480622
154    0.50   0.662504
218    6.45   5.400274
286    5.65   5.658562
186    0.25   0.647876

[61 rows x 2 columns]
```

In [83]:
```
x.columns
```

Out[83]:
```
Index(['Present_Price', 'Kms_Driven', 'Owner', 'Age', 'Fuel_Type_Diesel',
       'Fuel_Type_Petrol', 'Seller_Type_Individual', 'Transmission_Manual'],
      dtype='object')
```

In [84]:
```
#To get the Coefficient & Intercept
print('Coefficients:  ', model.coef_)
print('Intercept:   ', model.intercept_)
```

```
Coefficients:   [ 4.37233976e-01 -5.30613944e-06  3.45912849e-01 -4.13270098e-01
  2.23050770e+00  4.58549217e-01 -1.20927814e+00 -1.87014327e+00]
Intercept:   7.073759933489662
```

In [85]:
```
from sklearn.metrics import mean_squared_error,mean_absolute_error,r2_score

mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test,y_pred)
r2 = r2_score(y_test, y_pred)

print('Mean squared error:', mse)
print('Mean absolute error:', mae)
print('Coefficient of determination (R^2):', r2)
```

```
Mean squared error: 2.98238486185975
Mean absolute error: 1.0998575552990955
Coefficient of determination (R^2): 0.8625260513315252
```

Prediction for specific data provided

In [86]:
```
x.columns
```

Out[86]:
```
Index(['Present_Price', 'Kms_Driven', 'Owner', 'Age', 'Fuel_Type_Diesel',
       'Fuel_Type_Petrol', 'Seller_Type_Individual', 'Transmission_Manual'],
      dtype='object')
```

In [87]:
```
prediction = model.predict([[5.5,30000,0,10,0,1,1,1]])
print("Predicted Value for Car for given Data: ",prediction)
```

```
Predicted Value for Car for given Data:  [2.56578944]
```

Model-4 - Advertising.csv

In [106…
```
import pandas as pd
dataset=pd.read_csv("Advertising.csv")
dataset.head()
```

Out[106…

	TV	Radio	Newspaper	Sales
0	230.1	37.8	69.2	22.1

	TV	Radio	Newspaper	Sales
1	44.5	39.3	45.1	10.4
2	17.2	45.9	69.3	12.0
3	151.5	41.3	58.5	16.5
4	180.8	10.8	58.4	17.9

In [107…

```
dataset.isna().sum() # to check if na is there in data or not
```

Out[107…

```
TV           0
Radio        0
Newspaper    0
Sales        0
dtype: int64
```

In [108…

```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 4 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   TV         200 non-null    float64
 1   Radio      200 non-null    float64
 2   Newspaper  200 non-null    float64
 3   Sales      200 non-null    float64
dtypes: float64(4)
memory usage: 6.4 KB
```

In [109…

```
x=dataset[["TV","Radio","Newspaper"]]
y=dataset["Sales"]
print(x.shape)
print(y.shape)
```

```
(200, 3)
(200,)
```

In [110…

```
#To check the type of x & y
print(type(x))
print(type(y))
```

```
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.series.Series'>
```

In [111…

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=1)
```

In [112…

```
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(160, 3)
(40, 3)
(160,)
(40,)
```

In [113…

```
from sklearn.linear_model import LinearRegression
lm=LinearRegression()
```

```
lm.fit(x_train,y_train)

y_pred=lm.predict(x_test)
```

In [114...

```
diff=pd.DataFrame({"Actual":y_test,"Predicted":y_pred})
diff
```

Out[114...

	Actual	Predicted
58	23.8	21.327278
40	16.6	18.061384
34	11.9	10.046303
102	19.8	21.092542
184	17.6	20.785275
198	25.5	24.527870
95	16.9	16.841803
4	17.9	15.656542
29	10.5	10.138780
168	17.1	18.882480
171	17.5	15.809838
18	11.3	10.545831
11	17.4	18.933461
89	16.7	15.566434
110	18.4	17.868771
118	15.9	15.293500
159	12.9	13.757078
35	17.8	21.063979
136	9.5	10.059597
59	18.4	19.275341
51	10.7	11.153899
16	12.5	12.042160
44	8.5	8.630380
94	11.5	11.986448
31	11.9	12.614910
162	19.9	16.857222
38	10.1	9.732270
28	18.9	21.114177
193	19.6	18.151096

	Actual	Predicted
27	20.9	19.562902
47	23.2	22.112375
165	16.9	17.827641
194	17.3	16.547340
177	16.7	14.784358
176	20.2	21.414054
97	20.5	16.966640
174	16.5	17.225802
73	11.0	12.324184
69	22.3	21.079624
172	7.6	7.773868

In [115…

```
#To get the Coefficient & Intercept
print('Coefficients:  ', lm.coef_)
print('Intercept:  ', lm.intercept_)
```

```
Coefficients:    [ 0.05507865  0.10308563 -0.00090115]
Intercept:    4.637624442397916
```

In [116…

```
from sklearn.metrics import mean_squared_error,mean_absolute_error,r2_score

mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test,y_pred)
r2 = r2_score(y_test, y_pred)

print('Mean squared error:', mse)
print('Mean absolute error:', mae)
print('Coefficient of determination (R2 Score):', r2)
```

```
Mean squared error: 2.4093336128923672
Mean absolute error: 1.2754390912939682
Coefficient of determination (R2 Score): 0.8747226291661847
```

Model-5 insurance.csv

In [118…

```
import pandas as pd
df=pd.read_csv("insurance.csv")
df.head()
```

Out[118…

	age	sex	bmi	children	smoker	region	expenses
0	19	female	27.9	0	yes	southwest	16884.92
1	18	male	33.8	1	no	southeast	1725.55
2	28	male	33.0	3	no	southeast	4449.46
3	33	male	22.7	0	no	northwest	21984.47
4	32	male	28.9	0	no	northwest	3866.86

In [119…

```
df.isna().sum() # to check if na is there in data or not
```

Out[119…

```
age         0
sex         0
bmi         0
children    0
smoker      0
region      0
expenses    0
dtype: int64
```

In [120…

```
#Creating dummie variables for categorical features**
df=pd.get_dummies(data=df,drop_first=True)
df.head()
```

Out[120…

	age	bmi	children	expenses	sex_male	smoker_yes	region_northwest	region_southeast	region_sout
0	19	27.9	0	16884.92	0	1	0	0	
1	18	33.8	1	1725.55	1	0	0	1	
2	28	33.0	3	4449.46	1	0	0	1	
3	33	22.7	0	21984.47	1	0	1	0	
4	32	28.9	0	3866.86	1	0	1	0	

In [121…

```
x=df[["age","bmi","smoker_yes"]]
y=df["expenses"]
print(x.shape)
print(y.shape)
```

```
(1338, 3)
(1338,)
```

In [127…

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=1)

print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(1070, 3)
(268, 3)
(1070,)
(268,)
```

In [128…

```
from sklearn.linear_model import LinearRegression
lm=LinearRegression()
lm.fit(x_train,y_train)
y_pred=lm.predict(x_test)
```

In [129…

```
diff=pd.DataFrame({"Actual":y_test,"Predicted":y_pred})
diff
```

Out[129…

	Actual	Predicted
559	1646.43	4637.513777

	Actual	Predicted
1087	11353.23	13263.376708
1020	8798.59	13378.823508
460	10381.48	12739.553621
802	2103.08	1149.576521
...	...	...
682	40103.89	33478.481582
629	42983.46	35896.028305
893	44202.65	36642.503356
807	2136.88	4971.332002
1165	5227.99	5927.936841

268 rows × 2 columns

In [130...

```
print(lm.coef_)
print(lm.intercept_)
```

```
[  258.94072065   303.47111341 23722.84761499]
-11055.584441475954
```

In [131...

```
from sklearn import metrics
print("MAE: ",metrics.mean_absolute_error(y_test,y_pred))
print("MSE: ",metrics.mean_squared_error(y_test,y_pred))
print("Coefficient of Determination (R2 Score): ",r2_score(y_test,y_pred))
```

```
MAE:  4107.492142539742
MSE:  36305111.1936047
Coefficient of Determination (R2 Score):  0.7567996129313422
```

Model-6 - FuelConsumptionCo2.csv

In [139...

```
import pandas as pd
data=pd.read_csv('FuelConsumptionCo2.csv')
data
```

Out[139...

	MODELYEAR	MAKE	MODEL	VEHICLECLASS	ENGINESIZE	CYLINDERS	TRANSMISSION	FUELTYP
0	2014	ACURA	ILX	COMPACT	2.0	4	AS5	
1	2014	ACURA	ILX	COMPACT	2.4	4	M6	
2	2014	ACURA	ILX HYBRID	COMPACT	1.5	4	AV7	
3	2014	ACURA	MDX 4WD	SUV - SMALL	3.5	6	AS6	
4	2014	ACURA	RDX AWD	SUV - SMALL	3.5	6	AS6	
...	...	...	...	...	...	...	...	
1062	2014	VOLVO	XC60	SUV - SMALL	3.0	6	AS6	

	MODELYEAR	MAKE	MODEL	VEHICLECLASS	ENGINESIZE	CYLINDERS	TRANSMISSION	FUELTYP
			AWD					
1063	2014	VOLVO	XC60 AWD	SUV - SMALL	3.2	6	AS6	
1064	2014	VOLVO	XC70 AWD	SUV - SMALL	3.0	6	AS6	
1065	2014	VOLVO	XC70 AWD	SUV - SMALL	3.2	6	AS6	
1066	2014	VOLVO	XC90 AWD	SUV - STANDARD	3.2	6	AS6	

1067 rows × 13 columns

In [140…

```
#Removing columns which is of no use for model making
```

In [141…

```
data.drop(columns=['MAKE','MODEL','VEHICLECLASS','TRANSMISSION'],axis=1,inplace=True)
```

In [142…

```
data
```

Out[142…

	MODELYEAR	ENGINESIZE	CYLINDERS	FUELTYPE	FUELCONSUMPTION_CITY	FUELCONSUMPTION_I
0	2014	2.0	4	Z	9.9	
1	2014	2.4	4	Z	11.2	
2	2014	1.5	4	Z	6.0	
3	2014	3.5	6	Z	12.7	
4	2014	3.5	6	Z	12.1	
...	...	...	...	...	...	
1062	2014	3.0	6	X	13.4	
1063	2014	3.2	6	X	13.2	
1064	2014	3.0	6	X	13.4	
1065	2014	3.2	6	X	12.9	
1066	2014	3.2	6	X	14.9	

1067 rows × 9 columns

In [143…

```
#Converting Model year to age column
data['Age']=2023-data['MODELYEAR']

data.drop(labels='MODELYEAR',axis=1,inplace=True)
data
```

Out[143…

	ENGINESIZE	CYLINDERS	FUELTYPE	FUELCONSUMPTION_CITY	FUELCONSUMPTION_HWY	FUELCO
0	2.0	4	Z	9.9	6.7	

	ENGINESIZE	CYLINDERS	FUELTYPE	FUELCONSUMPTION_CITY	FUELCONSUMPTION_HWY	FUELCO
1	2.4	4	Z	11.2	7.7	
2	1.5	4	Z	6.0	5.8	
3	3.5	6	Z	12.7	9.1	
4	3.5	6	Z	12.1	8.7	
...	...	...	...	...	...	
1062	3.0	6	X	13.4	9.8	
1063	3.2	6	X	13.2	9.5	
1064	3.0	6	X	13.4	9.8	
1065	3.2	6	X	12.9	9.3	
1066	3.2	6	X	14.9	10.2	

1067 rows × 9 columns

In [144...

```
pd.get_dummies(data,drop_first=True)
```

Out[144...

	ENGINESIZE	CYLINDERS	FUELCONSUMPTION_CITY	FUELCONSUMPTION_HWY	FUELCONSUMPTIOI
0	2.0	4	9.9	6.7	
1	2.4	4	11.2	7.7	
2	1.5	4	6.0	5.8	
3	3.5	6	12.7	9.1	
4	3.5	6	12.1	8.7	
...	...	...	...	...	
1062	3.0	6	13.4	9.8	
1063	3.2	6	13.2	9.5	
1064	3.0	6	13.4	9.8	
1065	3.2	6	12.9	9.3	
1066	3.2	6	14.9	10.2	

1067 rows × 11 columns

In [145...

```
#Checking for corelation between parameters then we choose columns for x & y
data.corr()
```

Out[145...

	ENGINESIZE	CYLINDERS	FUELCONSUMPTION_CITY	FUELCONSUMPTIOI
ENGINESIZE	1.000000	0.934011	0.832225	0.
CYLINDERS	0.934011	1.000000	0.796473	0.

	ENGINESIZE	CYLINDERS	FUELCONSUMPTION_CITY	FUELCONSUMPTIO
FUELCONSUMPTION_CITY	0.832225	0.796473	1.000000	0.
FUELCONSUMPTION_HWY	0.778746	0.724594	0.965718	1.
FUELCONSUMPTION_COMB	0.819482	0.776788	0.995542	0.
FUELCONSUMPTION_COMB_MPG	-0.808554	-0.770430	-0.935613	-0.
CO2EMISSIONS	0.874154	0.849685	0.898039	0.
Age	NaN	NaN	NaN	

We are Choosing Engine Size,Cylinders & Fuelconsumsion_Comb for X and Predicting the CO2Emissions

In [146...

```
x=data[['ENGINESIZE','CYLINDERS','FUELCONSUMPTION_COMB']]
y=data['CO2EMISSIONS']
```

In [147...

```
print(x.shape)
print(y.shape)
```

```
(1067, 3)
(1067,)
```

In [149...

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=2)

print("x train: ", x_train.shape)
print("x test: ", x_test.shape)
print("y train: ", y_train.shape)
print("y test: ",y_test.shape)
```

```
x train:  (853, 3)
x test:  (214, 3)
y train:  (853,)
y test:  (214,)
```

In [151...

```
from sklearn.linear_model import LinearRegression
lr=LinearRegression()
lr.fit(x_train,y_train)
y_pred=lr.predict(x_test)
y_pred
```

Out[151...

```
array([304.41959015, 241.93535939, 310.44963749, 244.19052986,
       183.08327806, 180.99412035, 311.32782898, 328.68185636,
       323.15049785, 256.72547612, 217.60158149, 193.26780359,
       307.86179095, 256.96290191, 321.29876594, 259.74049978,
       320.84765292, 196.75767885, 205.58916128, 398.28791452,
       205.56532405, 182.8696895 , 304.41959015, 194.7161956 ,
       212.9722517 , 393.39816957, 254.63631841, 259.50307399,
       251.17028038, 206.46735278, 285.4035822 , 258.7907966 ,
       255.56218436, 306.48491063, 401.92061586, 236.52233917,
       188.40104801, 196.28282725, 204.63945809, 389.50410654,
       208.60418495, 313.22723536, 192.57936343, 223.18061446,
       260.66636574, 291.14852927, 320.58648855, 307.19718802,
       389.50410654, 308.33664255, 306.95966357, 248.81985964,
       274.79178089, 194.43109534, 250.24441442, 267.14742744,
       215.74984957, 185.62345014, 187.02416769, 257.15265325,
       269.68759951, 268.07329339, 284.66746757, 182.44251237,
```

```
       187.95003364, 163.40266718, 314.60411568, 324.81247843,
       198.1583964 , 215.74984957, 303.01877395, 261.1173801 ,
       284.52529205, 369.96632174, 217.60158149, 227.78620567,
       187.04800492, 350.99798824, 265.53312132, 217.62541872,
       288.18118007, 193.79032965, 210.21849107, 186.09830173,
       380.34059859, 175.93761344, 186.12213896, 253.71045245,
       212.07022298, 347.67402708, 182.15741211, 315.55381887,
       374.45282544, 349.525759  , 205.82658708, 318.52116807,
       212.99608894, 197.20869321, 250.24441442, 207.41705596,
       249.08112267, 194.45493257, 363.96011164, 215.77368681,
       310.18837446, 347.24684995, 168.98170015, 240.08362747,
       204.66329533, 198.58557353, 212.9722517 , 256.48805032,
       306.74617366, 308.59790557, 314.60411568, 368.51792972,
       375.80586853, 233.34130275, 266.67257584, 213.89811766,
       272.70262318, 266.91000164, 246.99196496, 339.6024965 ,
       194.90594693, 409.27986904, 258.57720803, 322.43822046,
       178.71521131, 258.0785192 , 298.12827979, 364.50562706,
       199.53527672, 204.63945809, 253.94787825, 281.72385694,
       203.71359214, 193.98008098, 253.94787825, 209.26878788,
       298.38944416, 215.74984957, 199.08426236, 209.53005091,
       209.53005091, 226.86024105, 191.17864588, 217.62541872,
       266.91000164, 321.5123545 , 218.55128468, 221.17446313,
       244.21436709, 328.51594226, 192.57936343, 267.14742744,
       202.78772618, 223.18061446, 339.6024965 , 239.56120007,
       189.80176556, 200.01012831, 187.95003364, 303.63580107,
       204.63945809, 187.95003364, 334.33240103, 242.60006097,
       256.03703596, 219.47715063, 265.53312132, 197.49379347,
       270.37603967, 366.30978317, 204.90072112, 237.97073119,
       163.3196608 , 209.53005091, 181.94382354, 199.06042512,
       359.33078185, 196.73384162, 260.85611707, 327.85133933,
       334.99700395, 187.04800492, 194.45493257, 194.43109534,
       245.11639581, 217.62541872, 180.0682544 , 304.20600159,
       353.03947149, 187.95003364, 301.38072926, 196.30666449,
       191.20248311, 253.94787825, 317.40555078, 299.52899734,
       247.89399368, 314.60411568, 209.29262511, 325.97567153,
       240.48706603, 314.60411568, 184.00914402, 214.15938069,
       241.41293199, 247.46681655, 252.78458649, 253.23560086,
       264.13240377, 401.92061586])
```

In [152…

```python
diff=pd.DataFrame({'Actual':y_test,'Predicted':y_pred})
diff
```

Out[152…

	Actual	Predicted
455	292	304.419590
954	288	241.935359
738	301	310.449637
913	286	244.190530
702	170	183.083278
...	...	...
311	235	247.466817
848	251	252.784586
508	258	253.235601
330	276	264.132404
476	354	401.920616

214 rows × 2 columns

In [153…

```
print(lr.coef_)
print(lr.intercept_)
```

```
[11.63291754  7.01508244  9.25865957]
69.05949122332137
```

In [156…

```
from sklearn import metrics
print('MAE: ',metrics.mean_absolute_error(y_test,y_pred))
print('MSE: ',metrics.mean_squared_error(y_test,y_pred))
print('R2 Score: ', metrics.r2_score(y_test,y_pred))
```

```
MAE:  16.132545872637
MSE:  503.8619843994965
R2 Score:  0.89119029063663
```

Prediction based on Generating Synthetic Data (Without use of dataset)

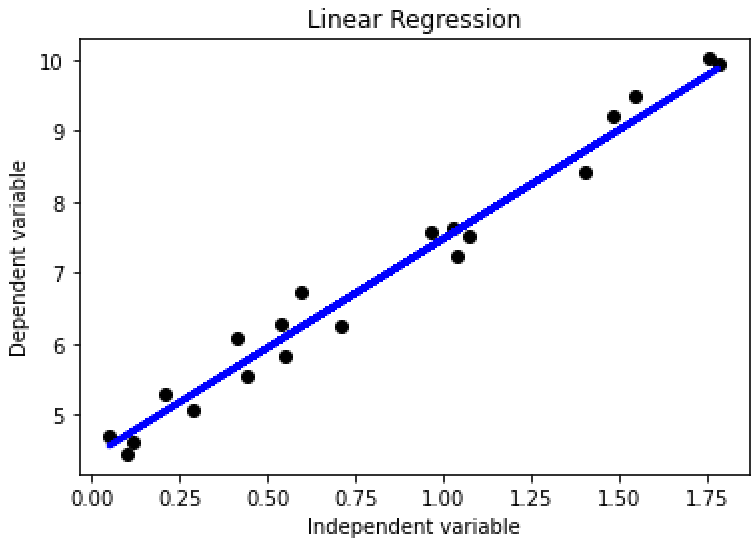
In [193…

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

# Generate synthetic data
x = 2 * np.random.rand(100,1)
y = 4 + 3 * x + np.random.rand(100,1)

# Split the data into training/testing sets
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=0

# Create linear regression object
reg = LinearRegression()

# Train the model using the training sets
reg.fit(x_train, y_train)

# Make predictions using the testing set
y_pred = reg.predict(x_test)

# The coefficients
print('Coefficients: ', reg.coef_)
print('Intercept: ', reg.intercept_)


# The mean absolute error
print('Mean squared error: %.2f' % mean_absolute_error(y_test, y_pred))

# The mean squared error
print('Mean squared error: %.2f' % mean_squared_error(y_test, y_pred))

# The coefficient of determination: 1 is perfect prediction
print('Coefficient of determination: %.2f' % r2_score(y_test, y_pred))
```

```
Coefficients:  [[3.07689823]]
Intercept:  [4.40632915]
Mean squared error: 0.25
Mean squared error: 0.07
Coefficient of determination: 0.98
```

In [194…

```
# Plot outputs
plt.scatter(x_test, y_test, color='black')
plt.plot(x_test, y_pred, color='blue', linewidth=3)
plt.xlabel('Independent variable')
plt.ylabel('Dependent variable')
plt.title('Linear Regression')
plt.show()
```



This code generates synthetic data, splits it into training and testing sets, trains a linear regression model, makes predictions, and evaluates the model's performance.

Linear regression is a fundamental tool in statistics and machine learning, providing a simple yet powerful way to model and understand relationships between variables.

Polynomial Regression

Model-7 polylinearregression.csv

In [200…

```
# Importing the libraries
# y = a + b_1x + b_2x^2 +....+ b_nx^n
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Importing the dataset
df = pd.read_csv('polylinearregression.csv')
df
```

Out[200…

	sno	Temperature	Pressure
0	1	0	0.0002
1	2	20	0.0012
2	3	40	0.0060
3	4	60	0.0300
4	5	80	0.0900
5	6	100	0.2700

In [201…

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6 entries, 0 to 5
Data columns (total 3 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   sno          6 non-null      int64  
 1   Temperature  6 non-null      int64  
 2   Pressure     6 non-null      float64
dtypes: float64(1), int64(2)
memory usage: 272.0 bytes
```

In [202…

```
df.isna().sum()
```

Out[202…

```
sno            0
Temperature    0
Pressure       0
dtype: int64
```

In [203…

```
# Extract our x values, the column Temperature
x = df.iloc[:, 1:2]

# Extract our y or target variable Pressure
y = df.iloc[:, 2]
```

In [204…

```
print(x.shape)
print(y.shape)
```

```
(6, 1)
(6,)
```

In [221…

```
# Fitting Polynomial Regression to the dataset
from sklearn.preprocessing import PolynomialFeatures
poly = PolynomialFeatures(degree = 3)
x_Poly = poly.fit_transform(x)


# Fitting the Polynomial Regression model on two components X and y.
from sklearn.linear_model import LinearRegression
lr = LinearRegression()
lr.fit(x_Poly, y)
```

Out[221…

```
LinearRegression()
```

In [222…

```
y_pred=lr.predict(x_Poly)
y_pred
```

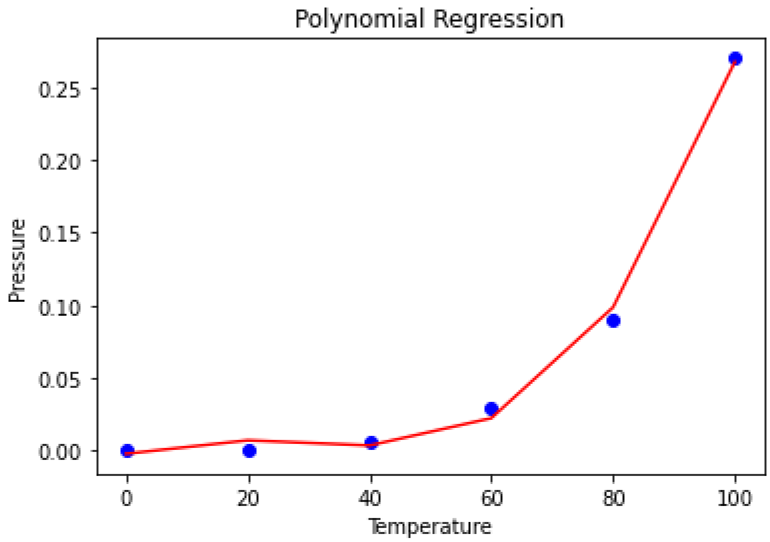
Out[222…

```
array([-0.00198889,  0.00724444,  0.00371111,  0.02248889,  0.09865556,
        0.26728889])
```

In [223…

```
# Visualising the Polynomial Regression results
plt.scatter(x, y, color = 'blue')

plt.plot(x, y_pred, color = 'red')
plt.title('Polynomial Regression')
plt.xlabel('Temperature')
plt.ylabel('Pressure')
```

```
plt.show()
```



In [225...

```
# train_rmse = np.sqrt(mean_squared_error(y_train, y_train_pred))
test_rmse = np.sqrt(mean_squared_error(y, y_pred))

# train_r2 = r2_score(y_train, y_train_pred)
test_r2 = r2_score(y, y_pred)

# print(f'Train RMSE: {train_rmse}')
print(f'Test RMSE: {test_rmse}')

# print(f'Train R-squared: {train_r2}')
print(f'Test R-squared: {test_r2}')
```

```
Test RMSE: 0.005556544356449009
Test R-squared: 0.9966691251761722
```

Model-8 - car_data.csv

In [226...

```
import pandas as pd
data=pd.read_csv('car_data.csv')
data
```

Out[226...

	Horsepower	Weight	MPG
0	130	3504	18
1	165	3693	15
2	150	3436	18
3	140	3433	16
4	198	4341	14
5	220	4354	12
6	95	2372	25
7	88	2130	27
8	98	2228	24

In [227…

```
data.isna().sum()
```

Out[227…

```
Horsepower    0
Weight        0
MPG           0
dtype: int64
```

In [228…

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9 entries, 0 to 8
Data columns (total 3 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Horsepower  9 non-null      int64
 1   Weight      9 non-null      int64
 2   MPG         9 non-null      int64
dtypes: int64(3)
memory usage: 344.0 bytes
```

In [229…

```
# Extract the independent and dependent variables
x = data[['Horsepower', 'Weight']]
y = data['MPG']
```

In [230…

```
print(x.shape)
print(y.shape)
```

```
(9, 2)
(9,)
```

In [231…

```
from sklearn.preprocessing import PolynomialFeatures

# Transform the features to polynomial features
polynomial_features = PolynomialFeatures(degree=2)
x_poly=polynomial_features.fit_transform(x)
```

In [232…

```
from sklearn.model_selection import train_test_split

# Split the data into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(x_poly, y, test_size=0.2, random_st
```

In [233…

```
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_train.shape)
```

```
(7, 6)
(2, 6)
(7,)
(7,)
```

In [234…

```
from sklearn.linear_model import LinearRegression
# Fit the linear regression model
model = LinearRegression()
model.fit(x_train, y_train)
```

Out[234…

```
LinearRegression()
```

In [235…

```
# Predict on the testing set
y_pred = model.predict(x_test)
```

In [236…

```
from sklearn.metrics import mean_squared_error, r2_score
# Evaluate the model

# train_r2 = r2_score(y_train, y_train_pred)
test_r2 = r2_score(y_test, y_pred)

# print(f'Train R-squared: {train_r2}')
print(f'Test R-squared: {test_r2}')
```

```
Test R-squared: 0.5889966459486313
```

Model-9 - Iris.csv

In [237…

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split

# Load the iris dataset
iris = pd.read_csv('Iris.csv')
iris
```

Out[237…

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa
...	...	...	...	...	...	...
145	146	6.7	3.0	5.2	2.3	Iris-virginica
146	147	6.3	2.5	5.0	1.9	Iris-virginica
147	148	6.5	3.0	5.2	2.0	Iris-virginica
148	149	6.2	3.4	5.4	2.3	Iris-virginica
149	150	5.9	3.0	5.1	1.8	Iris-virginica

150 rows × 6 columns

In [238…

```
# Extract the sepal length feature
x = iris[['SepalLengthCm']]

# Extract the petal length feature
y = iris['PetalLengthCm']
```

In [239…
```
print(type(x))
print(type(y))
```

```
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.series.Series'>
```

In [240…
```
# Create polynomial features with feature names
poly = PolynomialFeatures(degree=2)
x_poly = poly.fit_transform(x)
```

In [241…
```
# Split the data into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(x_poly, y, test_size=0.2, random_st

print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_train.shape)
```

```
(120, 3)
(30, 3)
(120,)
(120,)
```

In [242…
```
# Create a linear regression model
model = LinearRegression()

# Fit the model on the polynomial features
model.fit(x_train, y_train)

# Generate test data, reshape to a column vector
y_pred = model.predict(x_test)
print(y_pred)
```

```
[3.95957079 2.41859282 5.32997728 3.11808307 6.66801986 4.86491374
 5.73643413 2.41859282 1.39477321 5.73643413 3.55185063 3.11808307
 5.9748461  5.02644677 4.34124353 0.55853405 3.95957079 3.55185063
 3.11808307 2.1724057  3.33822278 3.11808307 5.47197475 2.1724057
 6.08428431 3.75896664 3.33822278 2.41859282 4.52231212 4.86491374]
```

In [243…
```
# Get the coefficients and intercept
coefficients = model.coef_
intercept = model.intercept_

print("Coefficients (for polynomial features):")
print(coefficients)
print("\nIntercept:")
print(intercept)
```

```
Coefficients (for polynomial features):
[ 0.          5.75035707 -0.32559266]

Intercept:
-18.439563215944794
```

In [244…
```
# train_rmse = np.sqrt(mean_squared_error(y_train, y_train_pred))
test_rmse = np.sqrt(mean_squared_error(y_test, y_pred))

# train_r2 = r2_score(y_train, y_train_pred)
test_r2 = r2_score(y_test, y_pred)

# print(f'Train RMSE: {train_rmse}')
```

```
print(f'Test RMSE: {test_rmse}')

# print(f'Train R-squared: {train_r2}')
print(f'Test R-squared: {test_r2}')
```

```
Test RMSE: 0.9914273722647334
Test R-squared: 0.6687317077540569
```

Conclusion

Linear regression, with its variants and extensions, is a powerful and widely-used tool in data analysis and predictive modeling. By understanding the fundamental concepts, assumptions, and techniques for regularization and diagnostics, you can effectively apply linear regression to a wide range of practical problems.

In [ ]: