Multiple linear regression

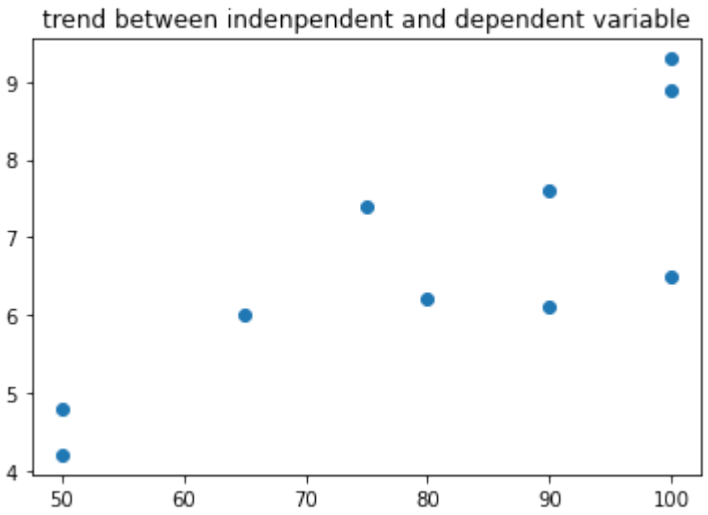
In [1]:
```
# import pandas library
import pandas as pd
```

In [2]:
```
#import data
data = pd.read_excel('C:/Users/kriti/OneDrive/Desktop/\
machine Learning/experiments/TRUCKING.xlsx')
data
```

Out[2]:

	Driving Assignmnet	miles_travelled	n_of_deliveries	travel_time
0	1	100	4	9.3
1	2	50	3	4.8
2	3	100	4	8.9
3	4	100	2	6.5
4	5	50	2	4.2
5	6	80	2	6.2
6	7	75	3	7.4
7	8	65	4	6.0
8	9	90	3	7.6
9	10	90	2	6.1

In [3]:
```
# plot a graph between any input attribute and output label
import matplotlib.pyplot as plt
plt.scatter(data['miles_travelled'], data['travel_time'])
plt.ylabel = ('travel time')
plt.title('trend between indenpendent and dependent variable')
plt.show()
```



In [4]:

```
#Extract the input features in X and the output label in Y
X = data[['miles_travelled', 'n_of_deliveries']]
Y = data['travel_time']
print(X)
print(Y)
```

```
   miles_travelled  n_of_deliveries
0              100                4
1               50                3
2              100                4
3              100                2
4               50                2
5               80                2
6               75                3
7               65                4
8               90                3
9               90                2
0    9.3
1    4.8
2    8.9
3    6.5
4    4.2
5    6.2
6    7.4
7    6.0
8    7.6
9    6.1
Name: travel_time, dtype: float64
```

What is sklearn Scikit-learn (Sklearn) is the most useful and robust library for machine learning in Python. It provides a selection of efficient tools for machine learning and statistical modeling including classification, regression, clustering and dimensionality reduction via a consistence interface in Python.

In [5]:

```
# import LinearRegression class from sklearn.linear_model
from sklearn.linear_model import LinearRegression
# import train_test_split function from sklearn.model_selection
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=45)
#random_state controls the shuffling applied to the data
#before applying the split.
```

In [6]:
```
1 X_train, X_test, Y_train, Y_test
```

Out[6]:
```
(   miles_travelled  n_of_deliveries
 1                50                3
 8                90                3
 9                90                2
 6                75                3
 5                80                2
 7                65                4
 0               100                4
 3               100                2,
    miles_travelled  n_of_deliveries
 2               100                4
 4                50                2,
 1    4.8
 8    7.6
 9    6.1
 6    7.4
 5    6.2
 7    6.0
 0    9.3
 3    6.5
 Name: travel_time, dtype: float64,
 2    8.9
 4    4.2
 Name: travel_time, dtype: float64)
```

In [7]:
```
1 X_train.shape, X_test.shape, Y_train.shape, Y_test.shape
```

Out[7]:
```
((8, 2), (2, 2), (8,), (2,))
```

Create a model and fit it

The next step is to create a linear regression model and fit it using the existing data. With .fit(), you calculate the optimal values of the weights b_0 and b_1, using the existing input and output (x and y) as the arguments. In other words, .fit() fits the model.

Let's create an instance of the class LinearRegression, which will represent the regression model:

In [8]:
```
1 #Create the model by creating the object of class LinearRegression
2 regr = LinearRegression()
3 #use the object to fit the model on the data
4 #First, you need to call .fit() on model:
5 regr.fit(X_train,Y_train)
```

Out[8]:
```
LinearRegression()
```

In [9]:
```
1 # find intercept of the linear equation
2 regr.intercept_
```

Out[9]:
```
-1.3154407102092618
```

In [10]:
```
# find slope for 2 independent variables
regr.coef_
```

Out[10]:
```
array([0.06449588, 0.97831325])
```

In [11]:
```
# Do the prediction for test data set
y_pred= regr.predict(X_test)
y_pred
```

Out[11]:
```
array([9.04740013, 3.86597971])
```

In [12]:
```
# create a dataframe for actual and predicted values
d = pd.DataFrame({"Actual value": Y_test, "predicted values": y_pred})
d
```

Out[12]:

	Actual value	predicted values
2	8.9	9.04740
4	4.2	3.86598

Prediction on a particular data points suppose miles_travelled= 200 n_of_deliveries=5

In [13]:
```
predicted = regr.predict([[200, 5]])
print(predicted)
```

```
[16.4753012]
```

The sklearn. metrics module implements several loss, score, and utility functions to measure classification performance. Some metrics might require probability estimates of the positive class, confidence values, or binary decisions values.

In [14]:
```
# calculate mean squared error
from sklearn.metrics import mean_squared_error
mean_squared_error(Y_test, y_pred)
```

Out[14]:
```
0.06664817632509885
```

R sqaured-coefficient of determination--goodness of fit R-squared (R2) is a statistical measure that represents the proportion of the variance for a dependent variable that's explained by an independent variable

In [15]:
```
# Adjusted  R squared -coefficient of determination--goodness of fit
from sklearn.metrics import r2_score
r2_score(Y_test, y_pred)
```

Out[15]:
```
0.9879315208103036
```

In [ ]:
```

```