Linear Regression

Linear regression is used for finding linear relationship between target and one or more predictors. There are two types of linear regression- Simple and Multiple.

Why use Linear Relationships?

Linear relationships, i.e. lines, are easier to work with and most phenomenon are naturally linearly related. If variables aren't linearly related, then some math can transform that relationship into a linear one, so that it's easier for the researcher (i.e. you) to understand.

Etymology

"Linear" means line. The word Regression came from a 19th-Century Scientist, Sir Francis Galton, who coined the term "regression toward mediocrity" (in modern language, that's regression to the mean. He used the term to describe the phenomenon of how nature tends to dampen excess physical traits from generation to generation

What is Simple Linear Regression?

If you're just beginning to learn about regression analysis, a simple linear is the first type of regression you'll come across.

Linear regression is the most widely used statistical technique; it is a way to model a relationship between two sets of variables. The result is a linear regression equation that can be used to make predictions about data.

Simple linear regression is useful for finding relationship between two continuous variables. One is predictor or independent variable and other is response or dependent variable. It looks for statistical relationship but not deterministic relationship. Relationship between two variables is said to be deterministic if one variable can be accurately expressed by the other. For example, using temperature in degree Celsius it is possible to accurately predict Fahrenheit. Statistical relationship is not accurate in determining relationship between two variables. For example, relationship between height and weight.

- The core idea is to obtain a line that best fits the data. The best fit line is the one for which total prediction error (all data points) are as small as possible. Error is the distance between the point to the regression line.

Simple linear regression formula

$$y = \beta_0 + \beta_1 X + \varepsilon$$

- y is the predicted value of the dependent variable (y) for any given value of the independent variable (x).
- B0 is the intercept, the predicted value of y when the x is 0.
- B1 is the regression coefficient – how much we expect y to change as x increases.
- x is the independent variable (the variable we expect is influencing y).
- e is the error of the estimate, or how much variation there is in our estimate of the regression coefficient.

Multiple Linear Regression?

Multiple linear regression refers to a statistical technique that is used to predict the outcome of a variable based on the value of two or more variables. It is sometimes known simply as multiple regression, and it is an extension of linear regression. The variable that we want to predict is known as the dependent variable, while the variables we use to predict the value of the dependent variable are known as independent or explanatory variables.

Multiple Linear Regression Formula

$$yi = \beta 0 + \beta 1xi1 + \beta 2xi2 + \dots + \beta pxip + \epsilon$$

Where:

- yi is the dependent or predicted variable
- β0 is the y-intercept, i.e., the value of y when both xi and x2 are 0.
- β1 and β2 are the regression coefficients representing the change in y relative to a one-unit change in xi1 and xi2, respectively.
- βp is the slope coefficient for each independent variable
- ϵ is the model's random error (residual) term.

Real-time example

We have a dataset which contains information about relationship between 'number of hours studied' and 'marks obtained'. Many students have been observed and their hours of study and grade are recorded. This will be our training data. Goal is to design a model that can predict marks if given the number of hours studied. Using the training data, a regression line is obtained which will give minimum error. This linear equation is then used for any new data. That is, if we give number of hours studied by a student as an input, our model should predict their mark with minimum error.

- Y(pred) = b0 + b1*x

The values b0 and b1 must be chosen so that they minimize the error. If sum of squared error is taken as a metric to evaluate the model, then goal to obtain a line that best reduces the error.

$$\text{Error} = \sum_{i=1}^{n}(actual\_output - predicted\_output) ** 2$$

If we don't square the error, then positive and negative point will cancel out each other.

For model with one predictor,

$$b_0 = \bar{y} - b_1\bar{x}$$

Co-efficient Formula:

$$b_1 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

Exploring 'b1'

- If b1 > 0, then x(predictor) and y(target) have a positive relationship. That is increase in x will increase y.
- If b1 < 0, then x(predictor) and y(target) have a negative relationship. That is increase in x will decrease y.

Exploring 'b0'

- If the model does not include x=0, then the prediction will become meaningless with only b0. For example, we have a dataset that relates height(x) and weight(y). Taking x=0(that is height as 0), will make equation have only b0 value which is completely meaningless as in real-time height and weight can never be zero. This resulted due to considering the model values beyond its scope.
- If the model includes value 0, then 'b0' will be the average of all predicted values when x=0. But, setting zero for all the predictor variables is often impossible.
- The value of b0 guarantee that residual have mean zero. If there is no 'b0' term, then regression will be forced to pass over the origin. Both the regression co-efficient and prediction will be biased.

Co-efficient from Normal equations

Apart from above equation co-efficient of the model can also be calculated from normal equation.

Co-efficient calculation using Normal Equation:

$$\text{Theta} = (X^T X)^{-1} X^T Y$$

Theta contains co-efficient of all predictors including constant term 'b0'. Normal equation performs computation by taking inverse of input matrix. Complexity of the computation will increase as the number of features increase. It gets very slow when number of features grow large.

Below is the python implementation of the equation.

In [2]:

```
import numpy as np                  #numpy-->Numerical calculation
def theta_calc(x_train,y_train):
    #initializing all varialbles
    n_data=x_train.shape[0]
    bias=np.ones((n_data,1))
    x_train_b=np.append(bias,x_train,axis=1)
    theta_1=np.linalg.inv(np.dot(x_train_b,x_train_b))
    theta_2=np.dot(theta_1,x_train_b.T)
    theta=np.dot(theta_2,y_train)
    return theta
```

Optimizing using gradient descent

Complexity of the normal equation makes it difficult to use, this is where gradient descent method comes into picture. Partial derivative of the cost function with respect to the parameter can give optimal co-efficient value.

Python code for gradient descent

In [3]:

```
#Gradient Descent
def grad_descent(iter_val):
    int_slope=0
    int_intercept=0
    n_pt=float(len(x_train))

    for i in range(len(x_train)):
        int_intercept=-(2/n_pt)*(y_train[i]-((s_slope*x_train[i]) +s_intercept))
        int_slope =-(2/n_pt)*x_train[i]*(y_train[i]-((s_slope*x_train[i])+ s_intercept))
    final_slope=s_slope-(l_rate*int_slope)
    final_intercept=s_intercept-(l_rate*int_intercept)
    s_slope = final_slope
    s_intercept=final_intercept
    return s_slope,s_intercept
```

Residual Analysis

Randomness and unpredictability are the two main components of a regression model.

Prediction = Deterministic + Statistic

Deterministic part is covered by the predictor variable in the model. Stochastic part reveals the fact that the expected and observed value is unpredictable. There will always be some information that are missed to cover. This information can be obtained from the residual information.

Let's explain the concept of residue through an example. Consider, we have a dataset which predicts sales of juice when given a temperature of place. Value predicted from regression equation will always have some difference with the actual value. Sales will not match exactly with the true output value. This difference is called as residue.

Residual plot helps in analyzing the model using the values of residues. It is plotted between predicted values and residue. Their values are standardized. The distance of the point from 0 specifies how bad the prediction was for that value. If the value is positive, then the prediction is low. If the value is negative, then the prediction is high. 0 value indicates prefect prediction. Detecting residual pattern can improve the model.

Non-random pattern of the residual plot indicates that the model is:

- Missing a variable which has significant contribution to the model target
- Missing to capture non-linearity (using polynomial term)
- No interaction between terms in model

Characteristics of a residue

- Residuals do not exhibit any pattern
- Adjacent residuals should not be same as they indicate that there is some information missed by system.

Residual implementation and plot

In [ ]:

```
#Resedual plot
import matplotlib.pyplot as plt                              #matplotlib-->visualization
plt.scatter(prediction,prediction-y_test,c="g",s=40)
plt.hlines(y=0,xmin=0,xmax=100)
plt.title("Residual plot")
plt.ylabel("Residual")
```

Metrics for model evaluation

R-Squared value

This value ranges from 0 to 1. Value '1' indicates predictor perfectly accounts for all the variation in Y. Value '0' indicates that predictor 'x' accounts for no variation in 'y'.

1. Regression sum of squares (SSR)

- This gives information about how far estimated regression line is from the horizontal 'no relationship' line (average of actual output).

$$\text{Error} = \sum_{i=1}^{n}(\text{Predicted\_output} - \text{average\_of\_actual\_output})\hat{}2$$

1. Sum of Squared error (SSE)

- How much the target value varies around the regression line (predicted value).

$$\text{Error} = \sum_{i=1}^{n}(Actual\_output - predicted\_output)\hat{}2$$

1. Total sum of squares (SSTO)

- This tells how much the data point move around the mean.

$$\text{Error} = \sum_{i=1}^{n}(\text{Actual\_output} - \text{average\_of\_actual\_output})\hat{}2$$

R^2 = 1 – (SSE/SSTO)

Python implementation

In [ ]:

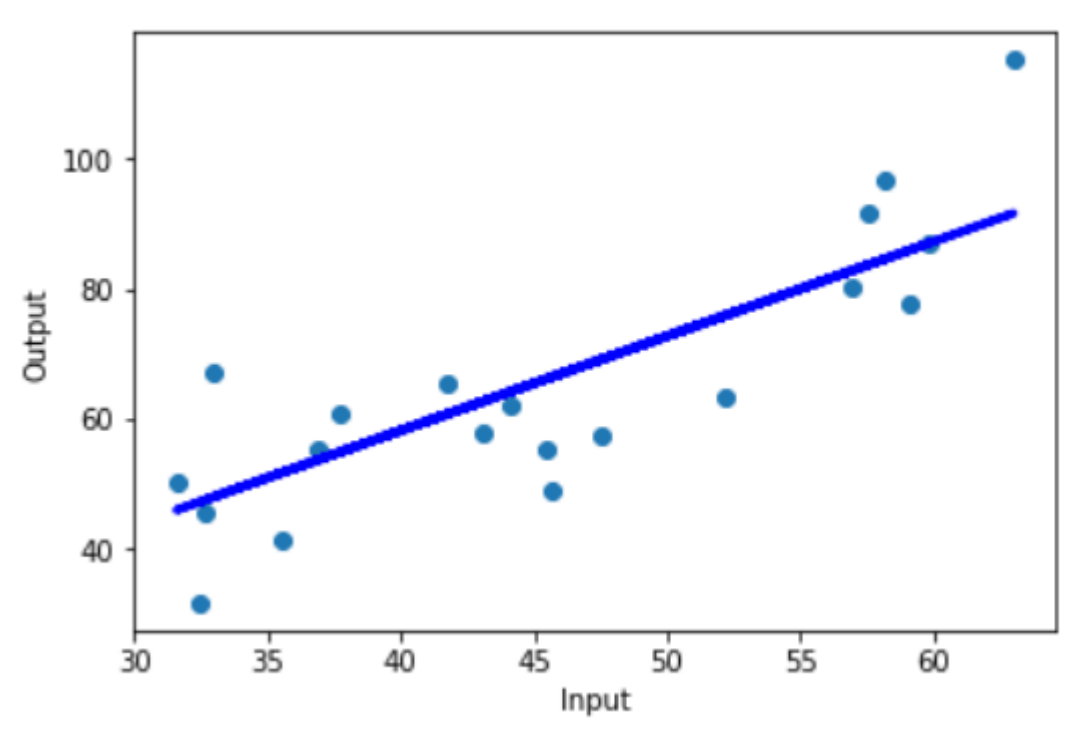
```
def rsq(prediction,y_test):
    total_data=len(prediction)
    y_avg =np.sum(y_test)/total_data                    #Average of total prediction
    tol_err = np.sum((y_test-y_avg)**2)                 #total sum of square error
    res-err=np.sum((y_test-prediction)**2)              #total sum of squared error of residuals
    r2=1-(res_err/tot_err)
    return r2
```

Null-Hypothesis and P-value

Null hypothesis is the initial claim that researcher specify using previous research or knowledge.

- Low P-value: Rejects null hypothesis indicating that the predictor value is related to the response
- High P-value: Changes in predictor are not associated with change in target

Obtained Regression line



Linear Regression Model

In [51]:

```
# Lets build the regression model using linear regression algoritham
#libraries required
import numpy as np                  #For numerical calculation and to produce multidimentional arreys
import pandas as pd                 # for data manupulation and data analysis
import matplotlib.pyplot as plt     # for visualization
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split  #Dividing to training and testing dataset
from sklearn.metrics import accuracy_score
from sklearn.model_selection import cross_val_score
import seaborn as sns
```

In [8]:

```
# import data
df_wf=pd.read_csv("weight-height.csv")
df_wf.head()
```

Out[8]:

	Gender	Height	Weight
0	Male	73.847017	241.893563
1	Male	68.781904	162.310473
2	Male	74.110105	212.740856
3	Male	71.730978	220.042470
4	Male	69.881796	206.349801

In [9]:

```
#checking the shape of the data(rows/columns)
df_wf.shape
```

Out[9]:

```
(10000, 3)
```

In [10]:

```
df_wf.describe()     # used for calculating some stastical data like percentile ,mean and standard deviation of numerical values
```

Out[10]:

	Height	Weight
count	10000.000000	10000.000000
mean	66.367560	161.440357
std	3.847528	32.108439
min	54.263133	64.700127
25%	63.505620	135.818051
50%	66.318070	161.212928
75%	69.174262	187.169525
max	78.998742	269.989698

In [11]:

```
df_wf.info()     #Prints information about the DataFrame which contains columns,columns labels,column data types,memory usage etc
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Gender  10000 non-null  object
 1   Height  10000 non-null  float64
 2   Weight  10000 non-null  float64
dtypes: float64(2), object(1)
memory usage: 234.5+ KB
```

In [12]:

```
df_wf.isnull().sum()   # Check whether is their any null or missing values in the data frame
```

Out[12]:

```
Gender    0
Height    0
Weight    0
dtype: int64
```

In [13]:

```
df_wf.columns
```

Out[13]:

```
Index(['Gender', 'Height', 'Weight'], dtype='object')
```

In [16]:

```
#convert categorical variable to numerical variable using get_dummies
df_wf=pd.get_dummies(df_wf,drop_first=True)
```

In [18]:

```
#devide data to independent and dependent variables
X=df_wf.drop("Weight",axis=1)
y=df_wf["Weight"]
```

In [24]:

```
#Devide into training and testing dataset
#Training dataset is used to traing the model
#Testing dataset is used to check the performance of the model
xtrain,xtest,ytrain,ytest=train_test_split(X,y,test_size=0.20)
print(xtrain.shape)
print(xtest.shape)
print(ytrain.shape)
print(ytest.shape)
```

```
(8000, 2)
(2000, 2)
(8000,)
(2000,)
```

Model building

In [27]:

```
lr=LinearRegression()
lr.fit(xtrain,ytrain)
pred=lr.predict(xtest)
```

In [41]:

```
#check error using cross_val_score
mse=cross_val_score(lr,X,y,scoring='neg_mean_squared_error',cv=5)
mean_mse=np.mean(mse)
print(mean_mse)
```

```
-100.311374498425
```

checking accuracy

In [38]:

```
lr.score(xtest,ytest)
```

Out[38]:

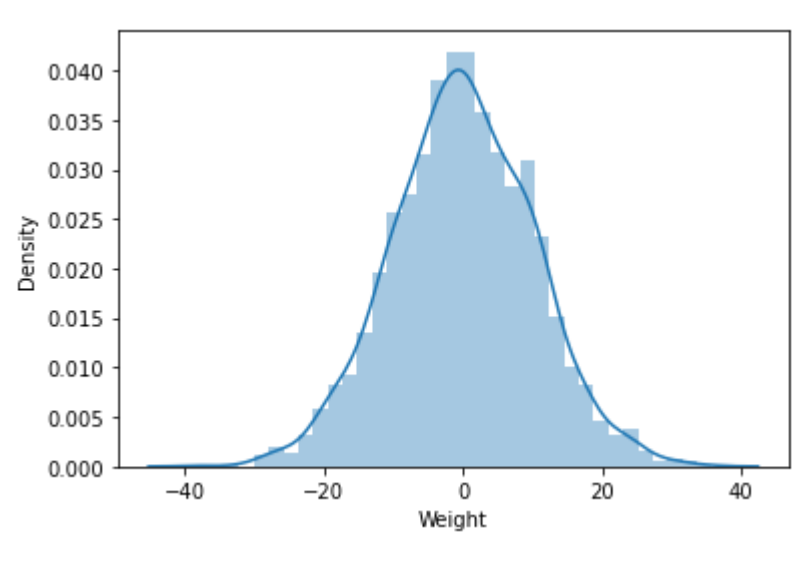
```
0.9009798425495678
```

In [53]:

```
#plotting graph
sns.distplot(ytest-pred)
```

Out[53]:

```
<AxesSubplot:xlabel='Weight', ylabel='Density'>
```



In [ ]: