Linear Regression

In statistics, linear regression is a linear approach to modelling the relationship between a scalar response (or dependent variable) and one or more explanatory variables (or independent variables). The case of one explanatory variable is called simple linear regression.

In a simple regression problem (a single x and a single y), the form of the model would be:
$\hat{y} = \beta_0 + \beta_1 . X$

For multiple features, the equation becomes:
$\hat{y} = \beta_0 + \beta_1 . X_1 + \beta_2 . X_2 + \ldots + \beta_n . X_n = \Sigma B_i . X_i$

Imports

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

Read CSV

In [2]:
```
df = pd.read_csv('Advertising.csv')
```

In [3]:
```
df.head()
```

Out[3]:

	TV	radio	newspaper	sales
0	230.1	37.8	69.2	22.1
1	44.5	39.3	45.1	10.4
2	17.2	45.9	69.3	9.3
3	151.5	41.3	58.5	18.5
4	180.8	10.8	58.4	12.9

In [4]:
```
df.describe()
```

Out[4]:

	TV	radio	newspaper	sales
count	200.000000	200.000000	200.000000	200.000000
mean	147.042500	23.264000	30.554000	14.022500
std	85.854236	14.846809	21.778621	5.217457
min	0.700000	0.000000	0.300000	1.600000
25%	74.375000	9.975000	12.750000	10.375000
50%	149.750000	22.900000	25.750000	12.900000
75%	218.825000	36.525000	45.100000	17.400000

	TV	radio	newspaper	sales
max	296.400000	49.600000	114.000000	27.000000

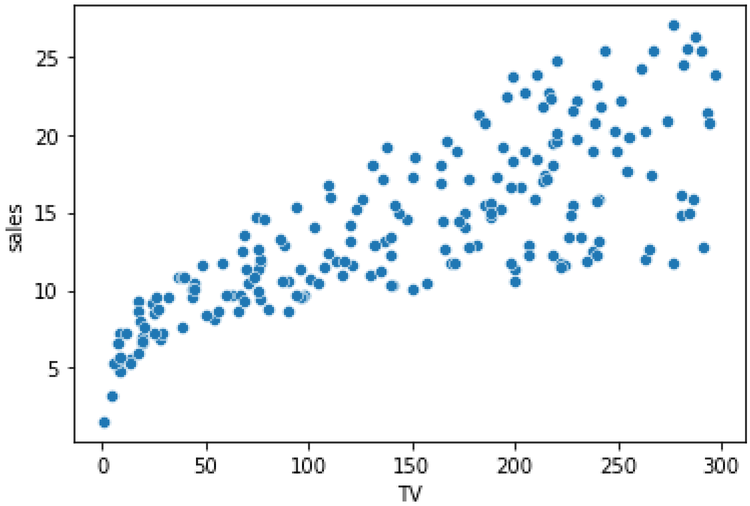
In [5]:
```
df.info()
```

```
<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
```

Exploratory Data Analysis

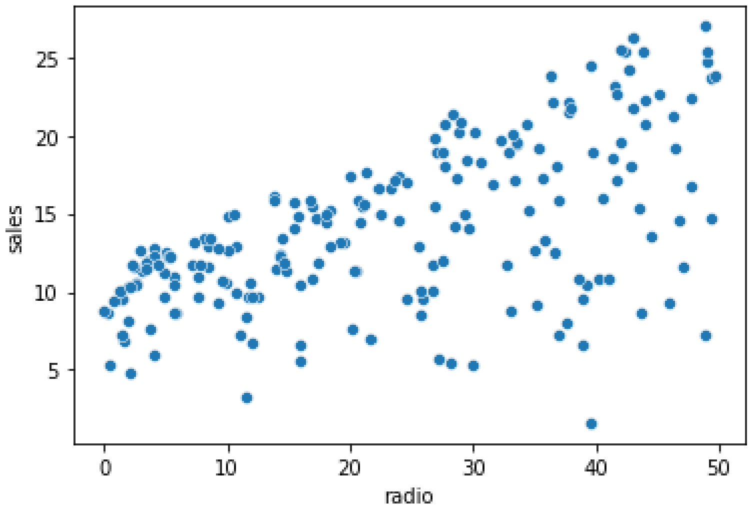
In [6]:
```
sns.scatterplot(x='TV', y='sales', data=df)
```

Out[6]: <AxesSubplot:xlabel='TV', ylabel='sales'>



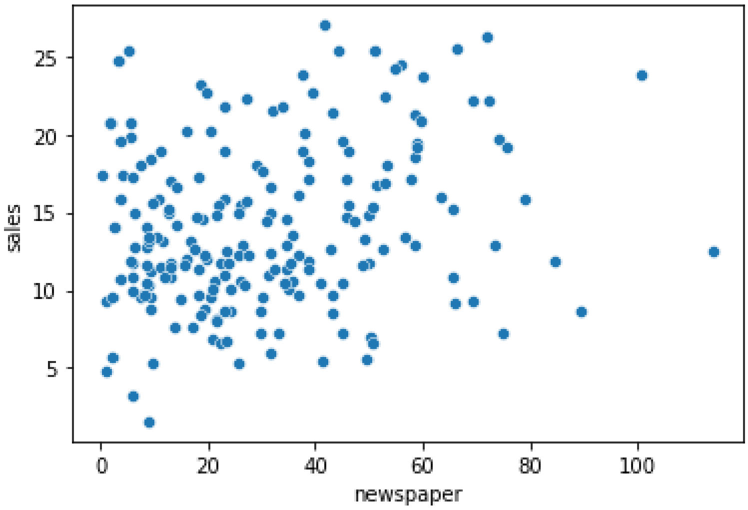
In [7]:
```
sns.scatterplot(x='radio', y='sales', data=df)
```

Out[7]: <AxesSubplot:xlabel='radio', ylabel='sales'>



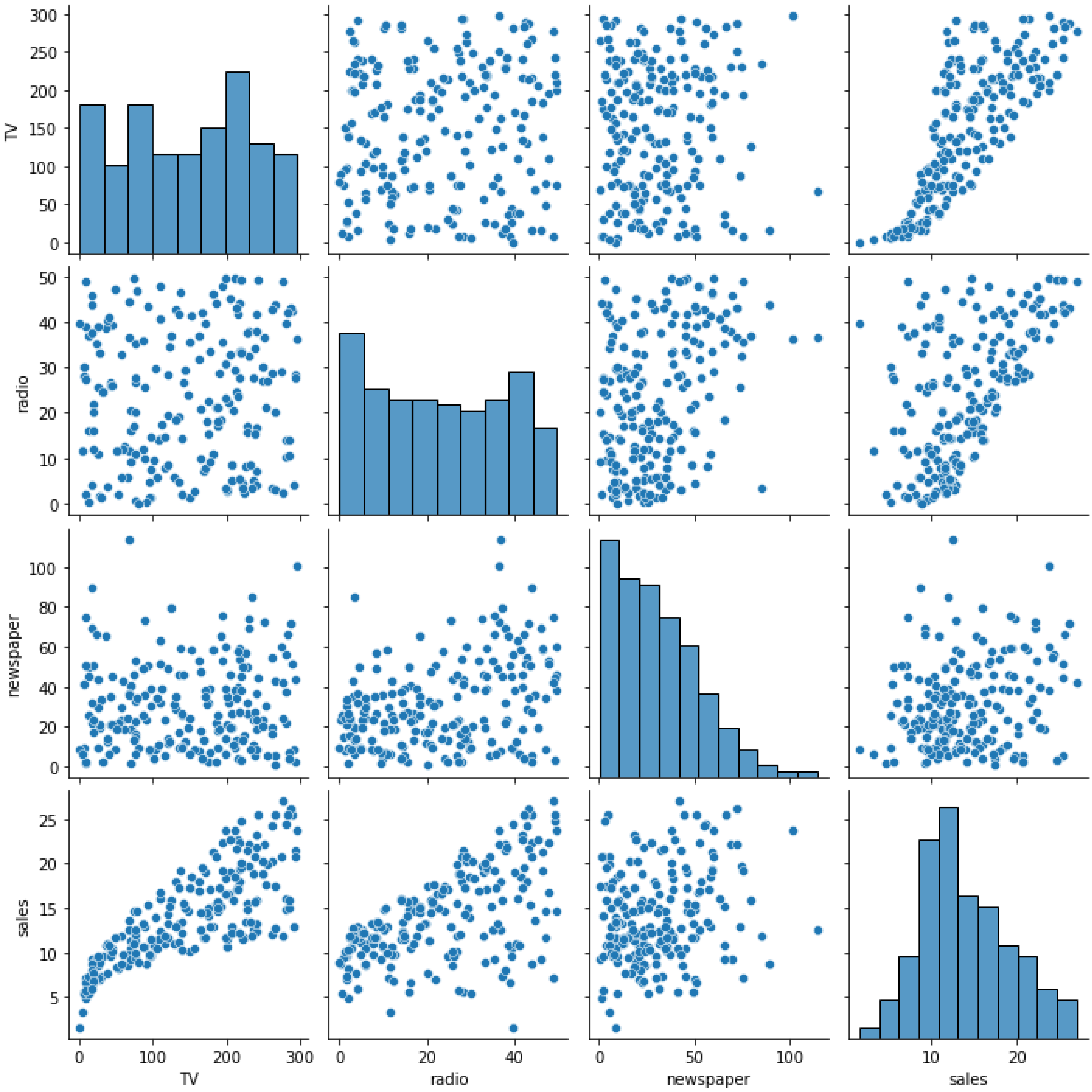
In [8]:
```
sns.scatterplot(x='newspaper', y='sales', data=df)
```

Out[8]: <AxesSubplot:xlabel='newspaper', ylabel='sales'>



In [9]:
```
sns.pairplot(df)
```

Out[9]: <seaborn.axisgrid.PairGrid at 0x18fcc909520>



Feature Selection

In [10]:
```
X = df.drop('sales', axis=1) # Independent Features (Columns used for prediction)
y = df['sales'] # Dependent Feature (Column that is to be predicted)
```

In [11]:
```
X.head()
```

Out[11]:

	TV	radio	newspaper
0	230.1	37.8	69.2
1	44.5	39.3	45.1
2	17.2	45.9	69.3
3	151.5	41.3	58.5
4	180.8	10.8	58.4

In [12]:
```
y.head()
```

Out[12]:
```
0    22.1
1    10.4
2     9.3
3    18.5
4    12.9
Name: sales, dtype: float64
```

Train Test Split

In supervised machine learning, we split our datasets into training and testing sets. The training set is used to train the machine learning algorithm by providing X_train, y_train values to the algorithm. We provide the X_test values to the model and the model gives us predictions in y_pred. Then we compare y_pred and y_test to calculate the performance of our model.

In [13]:
```
from sklearn.model_selection import train_test_split
```

In [20]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=202
```

In [21]:
```
X_train.head()
```

Out[21]:

	TV	radio	newspaper
105	137.9	46.4	59.0
168	215.4	23.6	57.6
173	168.4	7.1	12.8
145	140.3	1.9	9.0
129	59.6	12.0	43.1

In [22]:
```
len(X)
```

Out[22]:
```
200
```

In [23]:
```
len(X_train)
```

Out[23]:
```
140
```

In [24]:
```
len(X_test)
```

Out[24]:
```
60
```

Linear Regression Model

General Equation: $\hat{y} = \beta_0 + \beta_1 . X_1 + \beta_2 . X_2 + \ldots + \beta_n . X_n = \Sigma \beta_i . X_i$

X_i: Input
$\hat{y}$: Predicted Output
β_0: Intercept (bias)
β_i: Coefficients

In our case, $\hat{y} = \beta_0 + \beta_1 . X_1 + \beta_2 . X_2 + \beta_3 . X_3$
$(X_1 : TV, X_2 : radio, X_3 : newspaper, \hat{y} : PredictedSales)$
So the goal of the linear regression model is to accurately calculate the values of $\beta_0, \beta_1, \beta_2 and \beta_3$.

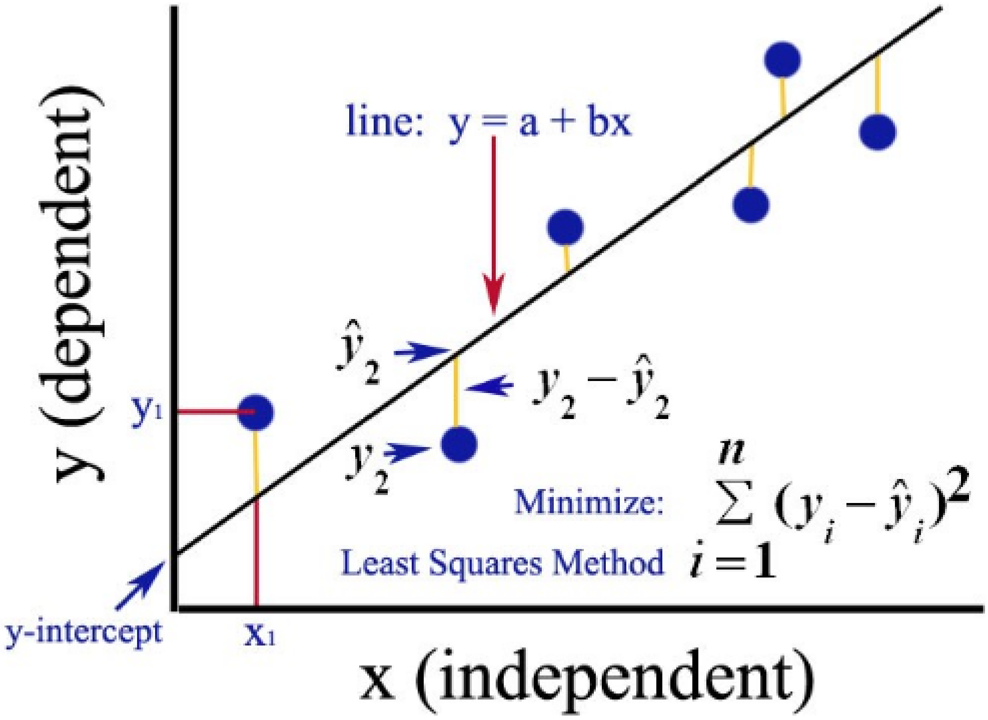
Error = $y - \hat{y}$ (for each and every row)
Average Squard Error = $(\Sigma(y - \hat{y})^2)/n$ (y: Actual Sales, n: no. of samples)
The goal is to minimize the average squared error.

Cost Function = $(\Sigma(y - \Sigma\beta_i . X_i)^2)/n$

To minimize the function, we can set the derivative to zero. For the calculation, we use gradient descent which is an optimization algorithm. Essentially we are optimizing the values of beta.

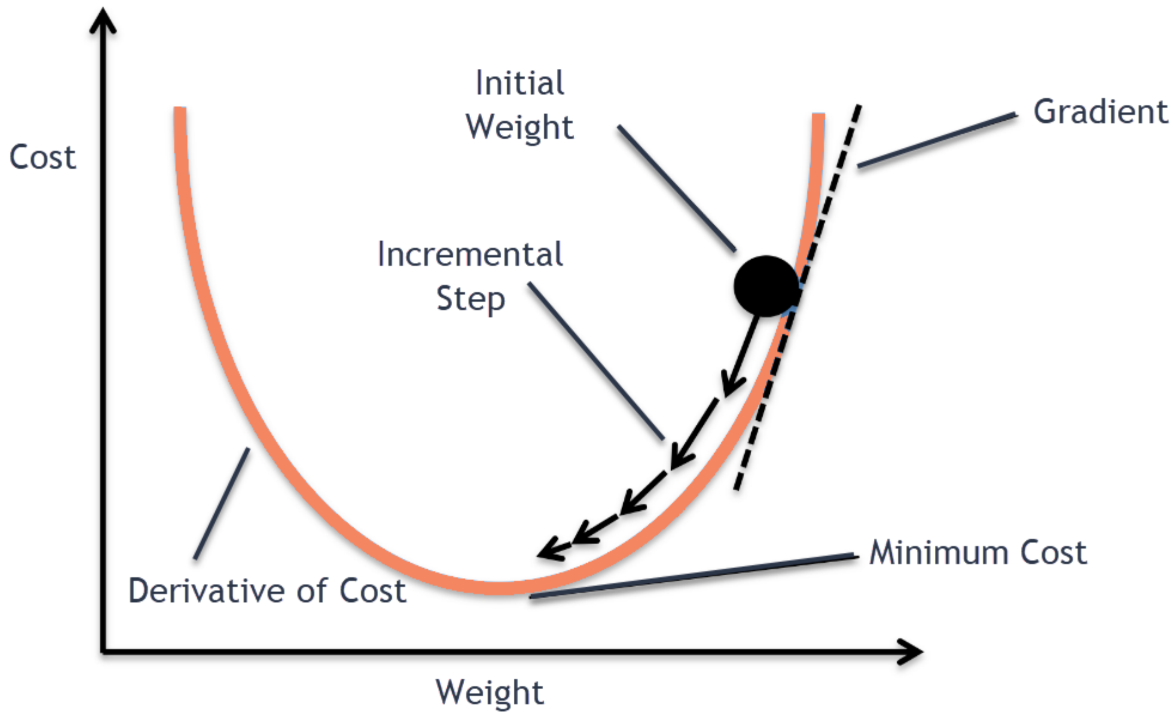
Ordinary Least Squares

The Ordinary Least Squares procedure seeks to minimize the sum of the squared residuals. This means that given a regression line through the data we calculate the distance from each data point to the regression line, square it, and sum all of the squared errors together. This is the quantity that ordinary least squares seeks to minimize.



Gradient Descent

When there are one or more inputs, we use a process of optimizing the values of the coefficients by iteratively minimizing the error of the model on your training data. This operation is called Gradient Descent and works by starting with random values for each coefficient. The sum of the squared errors are calculated for each pair of input and output values. The coefficients are updated in the direction towards minimizing the error. The process is repeated until a minimum sum squared error is achieved.



Create Model

In [25]:
```
from sklearn.linear_model import LinearRegression
```

In [29]:
```
model = LinearRegression()
```

In [30]:
```
model.fit(X_train, y_train)
```

Out[30]:
```
LinearRegression()
```

In [31]:
```
y_pred = model.predict(X_test)
```

In [32]:
```
y_test[:5] # Actual values in dataframe
```

Out[32]:
```
122    11.6
24      9.7
23     15.5
148    10.9
76      6.9
Name: sales, dtype: float64
```

In [33]:
```
y_pred[:5] # Values predicted by our machine learning model
```

Out[33]:
```
array([13.64881754,  8.55025122, 16.51544855, 12.84662834,  4.91417779])
```

Performance Evaluation

In [34]:
```
from sklearn.metrics import mean_absolute_error, mean_squared_error
```

$y : y\_test$

$\hat{y}$: y_pred

$MAE: \frac{\Sigma|y-\hat{y}|}{n}$

$MSE: \frac{\Sigma(y-\hat{y})^2}{n}$ (punishes large errors)

$RMSE: \sqrt{\frac{\Sigma(y-\hat{y})^2}{n}}$ (punishes large errors but also has same unit as error)

In [35]:
```
print(f"MAE: {mean_absolute_error(y_test, y_pred)}")
print(f"MSE: {mean_squared_error(y_test, y_pred)}")
print(f"RMSE: {np.sqrt(mean_squared_error(y_test, y_pred))}")
```

```
MAE: 1.5377057168015353
MSE: 4.759215713246759
RMSE: 2.18156267690084
```

In [36]:
```
model.intercept_ # B0
# B0 is the amount of predicted sales when no advertisement is done (intercept or bias)
```

Out[36]:
```
3.5942796098690906
```

In [37]:
```
model.coef_ # B1, B2, B3
```

Out[37]:
```
array([ 0.043446  ,  0.19119927, -0.00873361])
```

We can interpret the above values as follows:

- For 1 unit increase in TV advertisement, sales will increase by 0.043446 (B1) units
- For 1 unit increase in radio advertisement, sales will increase by 0.19119927 (B2) units
- For 1 unit increase in newspaper advertisement, sales will decrease by 0.00873361 (B1) units
- The newspaper coefficient is so low that it can be ignored. It means that newspaper advertisement has no effect on Sales. Hence the company should stop advertisement in newspaper

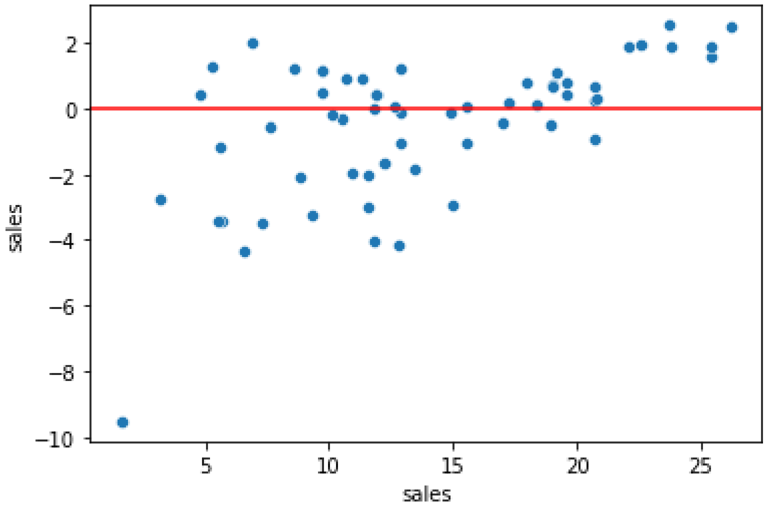
How to check whether Linear Regression is the model that should be used?

A residual plot is a graph that shows the residuals on the vertical axis and the dependent variable on the horizontal axis. If the points in a residual plot are randomly dispersed around the horizontal axis, a linear regression model is appropriate for the data; otherwise, a nonlinear model is more appropriate.

In [38]:
```
residual = y_test - y_pred
```

In [39]:
```
sns.scatterplot(x=y_test, y=residual)
plt.axhline(y=0, color='red')
# There should not be any noticable pattern in this graph (line, curve, etc.)
```

Out[39]:
```
<matplotlib.lines.Line2D at 0x18fcdc0ba90>
```



In [40]:
```
sns.displot(residual, bins=25, kde=True)
# This data should be normally distributed
```

Out[40]: <seaborn.axisgrid.FacetGrid at 0x18fcdd9fbe0>