Linear Regression (For Regression Problem)

(Code: Subhajit Das)

What is Linear Regression?

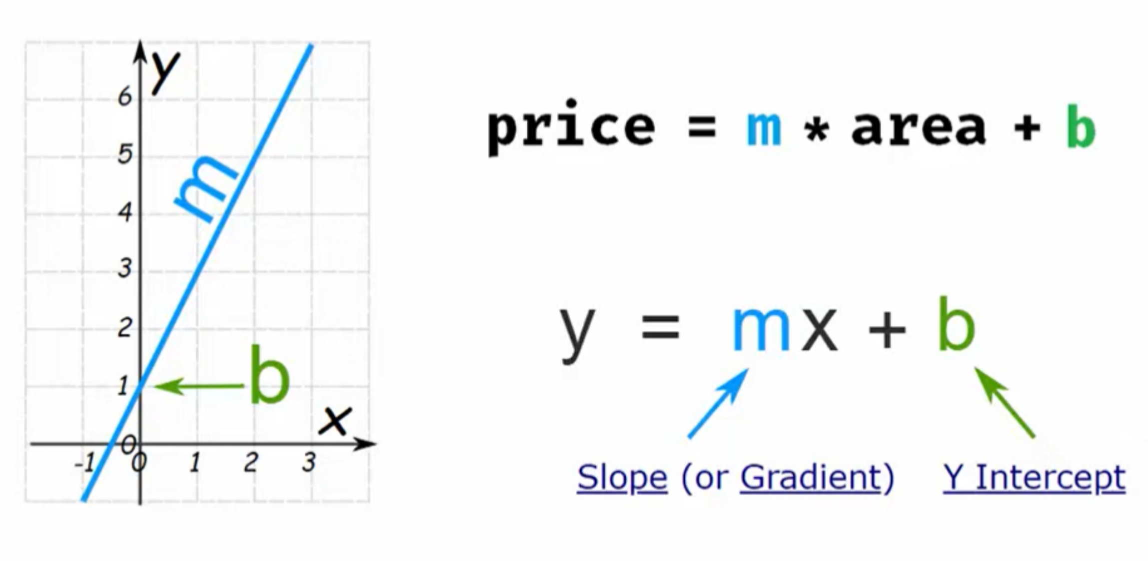
Linear Regression is a type of supervised machine learning algorithm that computes the linear relationship between a dependent variable and one or more independent features.

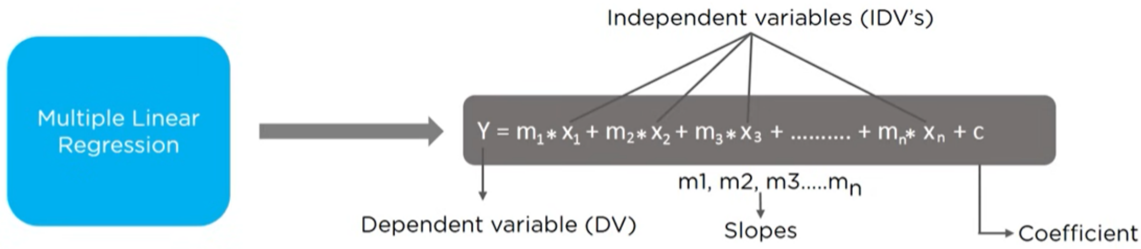
When the number of the independent feature is 1, it is known as **Univariate Linear Regression**, and in the case of more than one feature, it is known as **Multivariate Linear Regression**. The goal of the algorithm is to find the best linear equation that can predict the value of the dependent variable based on the independent variables.

The equation provides a straight line that represents the relationship between the dependent and independent variables. The slope of the line indicates how much the dependent variable changes for a unit change in the independent variable(s).

Linear regression is used in many different fields, including finance, economics, and psychology, to understand and predict the behavior of a particular variable[1]. For example, in finance, linear regression might be used to understand the relationship between a company's stock price and its earnings or to predict the future value of a currency based on its past performance.

In statistics, linear regression is a linear approach for modeling the relationship between a scalar response and one or more explanatory variables (also known as dependent and independent variables). The case of one explanatory variable is called simple linear regression; for more than one, the process is called multiple linear regression.





What we can use Linear Regression?

Linear Regression has a wide range of applications in various fields. Here are some examples:

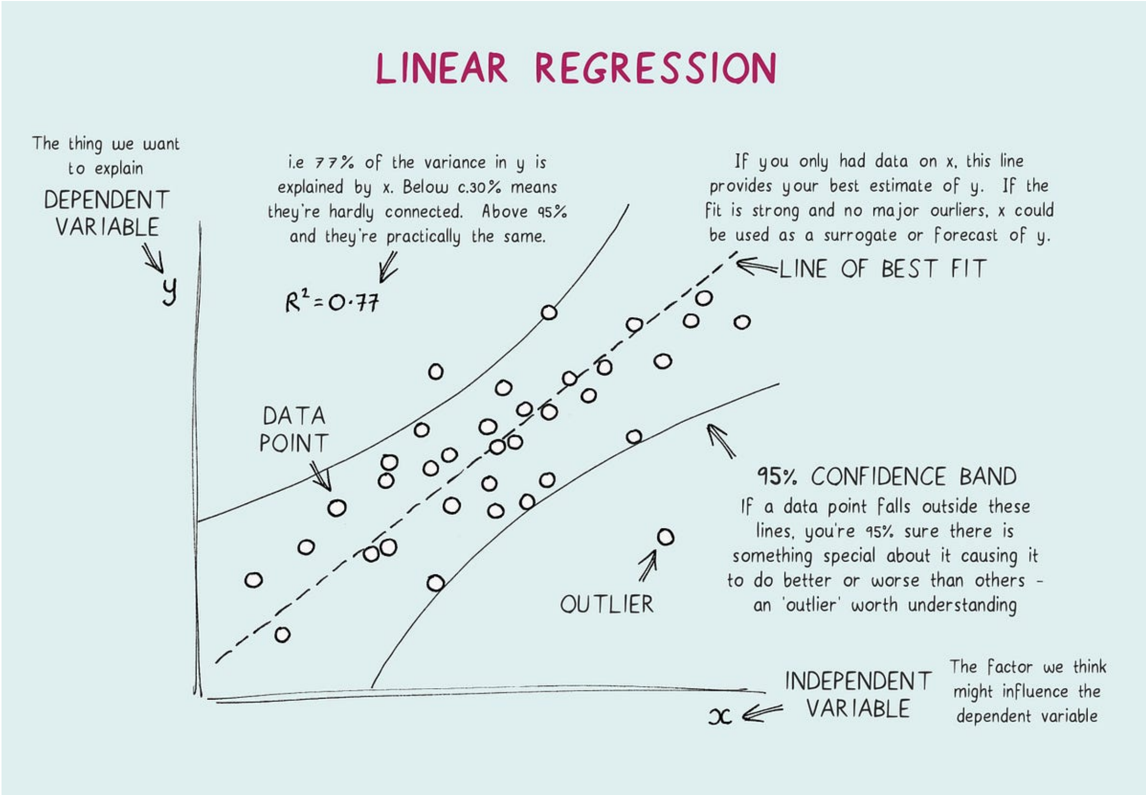
1. **Forecasting Revenue**: Linear regression can be used to forecast a company's revenue based on past performances.
2. **Salary Estimation**: It can be used to estimate the salary range for a job based on the current job market.
3. **Stock Market Analysis**: Linear regression can be used to study how the stock market has performed over time, which can be useful for making investment decisions.
4. **Predicting Future Outcomes**: Linear regression can be used to predict future outcomes based on historical data.
5. **Understanding Relationships**: It can be used to understand the relationship between different variables.
6. **Modelling Trends**: Linear regression can be used to model trends in data.
7. **Machine Learning Models**: Linear regression is often used in machine learning models to predict the behavior of a particular variable.
8. **Forecasting Sectors**: It is used in forecasting sectors to describe the behavior of a set of data.
9. **Quantitative Applications**: Linear regression is used in various quantitative applications.
10. **Finance, Economics, and Psychology**: Linear regression is used in many different fields, including finance, economics, and psychology, to understand and predict the behavior of a particular variable.

Basis	Linear Regression	Logistic Regression
Core Concept	The data is modelled using a straight line	The probability of some obtained event is represented as a linear function of a combination of predictor variables.
Used with	Continuous Variable	Categorical Variable
Output/Prediction	Value of the variable	Probability of occurrence of event
Accuracy and Goodness of fit	measured by loss, R squared, Adjusted R squared etc.	Accuracy, Precision, Recall, F1 score, ROC curve, Confusion Matrix, etc

How Linear Regression works:

1. **Data Collection**: The first step in linear regression is to collect data. This data should include both the independent and dependent variables.
2. **Model Specification**: The next step is to specify the model. This involves deciding which variables will be used as the independent variables and which will be the dependent variable.
3. **Fitting the Model**: Once the model is specified, the next step is to fit the model to the data. This involves finding the line of best fit that minimizes the sum of the squared residuals.
4. **Making Predictions**: After the model has been fitted, it can be used to make predictions. This involves plugging in the values of the independent variables into the equation of the line and solving for the dependent variable.
5. **Interpreting the Results**: The final step is to interpret the results. This involves understanding what the slope and intercept of the line mean in the context of the data.
6. **Evaluation**: The model's performance is evaluated using various metrics like R-squared, Mean Squared Error (MSE), etc.

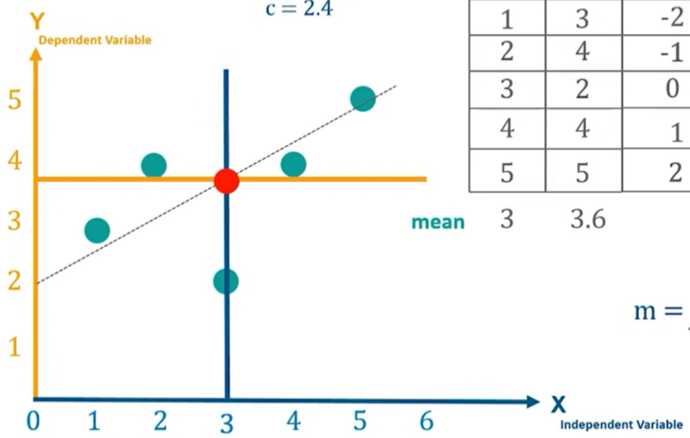
Remember, the goal of linear regression is to find the best linear equation that can predict the value of the dependent variable based on the independent variables.



Understanding Linear Regression Algorithm

y = mx+c

c = 2.4



x	y	$x - \bar{x}$	$y - \bar{y}$	$(x - \bar{x})^2$	$(x - \bar{x})(y - \bar{y})$
1	3	-2	-0.6	4	1.2
2	4	-1	0.4	1	-0.4
3	2	0	-1.6	0	0
4	4	1	0.4	1	0.4
5	5	2	1.4	4	2.8
mean 3	3.6			$\Sigma = 10$	$\Sigma = 4$

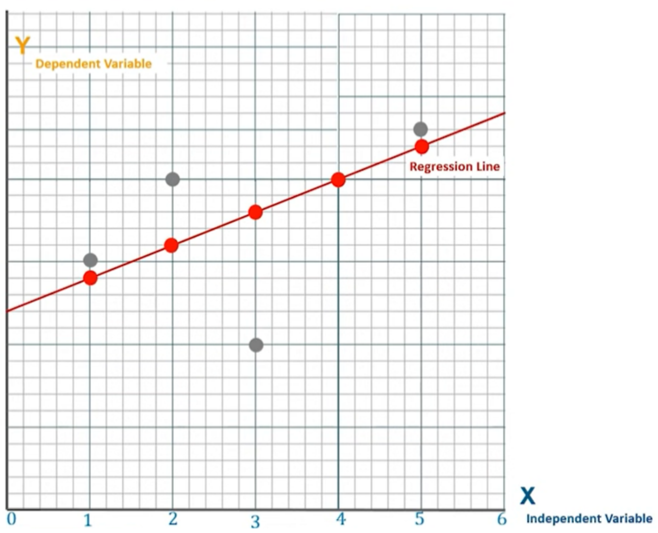
$$m = \sum \frac{(x - \bar{x})(y - \bar{y})}{(x - \bar{x})^2} = \frac{4}{10}$$

m = 0.4

c = 2.4

y = 0.4x + 2.4

Mean Square Error



m = 0.4

c = 2.4

y = 0.4x + 2.4

For given m = 0.4 & c = 2.4, lets predict values for y for x = {1,2,3,4,5}

y = 0.4 x 1+ 2.4 = 2.8

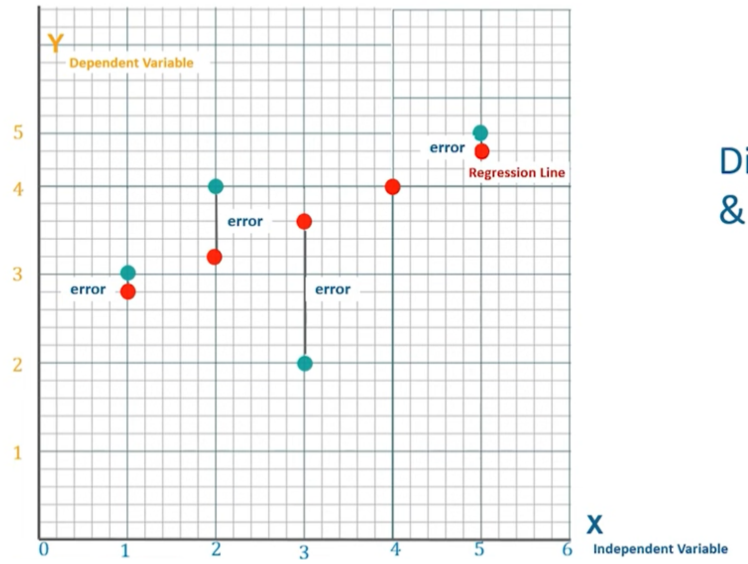
y = 0.4 x 2+ 2.4= 3.2

y = 0.4 x 3+ 2.4= 3.6

y = 0.4 x 4+ 2.4= 4.0

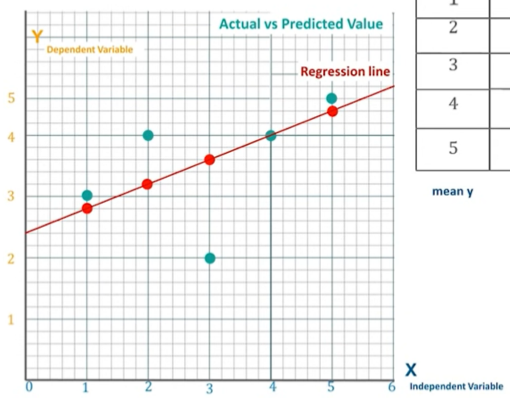
y = 0.4 x 5+ 2.4= 4.4

Mean Square Error



Distance between actual & predicted value

Calculation of R^2



x	y	$y - y$	$(y - y)^2$	y_p	$(y_p - y)$	$(y_p - y)^2$
1	3	− 0.6	0.36	2.8	-0.8	0.64
2	4	0.4	0.16	3.2	-0.4	0.16
3	2	−1.6	2.56	3.6	0	0
4	4	0.4	0.16	4.0	0.4	0.16
5	5	1.4	1.96	4.4	0.8	0.64
mean y	3.6		Σ 5.2			Σ 1.6

$$R^2 = \frac{1.6}{5.2} = \frac{\Sigma\left((y_p - \bar{y})\right)^2}{\Sigma\,(y - \bar{y})^2}$$

Gradient Descent:

Gradient Descent is an iterative optimization algorithm that tries to find the optimum value (Minimum/Maximum) of an objective function. It is one of the most used optimization techniques in machine learning projects for updating the parameters of a model in order to minimize a cost function. The main aim of gradient descent is to find the best parameters of a model which gives the highest accuracy on training as well as testing datasets. In gradient descent, the gradient is a vector that points in the direction of the steepest increase of the function at a specific point. Moving in the opposite direction of the gradient allows the algorithm to gradually descend towards lower values of the function, and eventually reaching to the minimum of the function.

Slope:

In the context of linear regression, the slope (often denoted by ‘b’ or ‘m’) represents the rate of change of the dependent variable (Y) with respect to the independent variable (X). In other words, it quantifies the change in the output Y that results from a unit change in the input X.

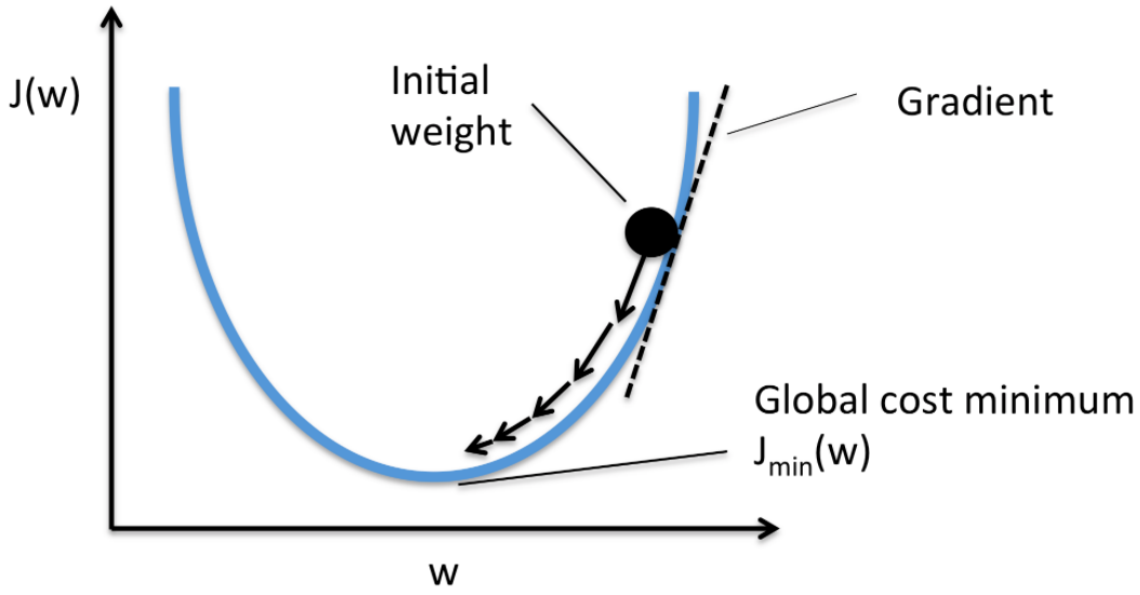
Formula: Y = a + bX, where

‘a’ is the y-intercept,

'b' is the slope,

'X' is the independent variable,

and 'Y' is the dependent variable



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

1000 Companies Profit

In [ ]:
```
profit_df = pd.read_csv("/content/drive/MyDrive/ML and DL DataSets/12_1000_C
profit_df.head()
```

Out[2]:

	R&D Spend	Administration	Marketing Spend	State	Profit
0	165349.20	136897.80	471784.10	New York	192261.83
1	162597.70	151377.59	443898.53	California	191792.06
2	153441.51	101145.55	407934.54	Florida	191050.39
3	144372.41	118671.85	383199.62	New York	182901.99
4	142107.34	91391.77	366168.42	Florida	166187.94

In [ ]:
```
profit_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 5 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   R&D Spend        1000 non-null   float64
 1   Administration   1000 non-null   float64
 2   Marketing Spend  1000 non-null   float64
 3   State            1000 non-null   object 
 4   Profit           1000 non-null   float64
dtypes: float64(4), object(1)
memory usage: 39.2+ KB
```

In [ ]:
```
profit_df.dtypes
```

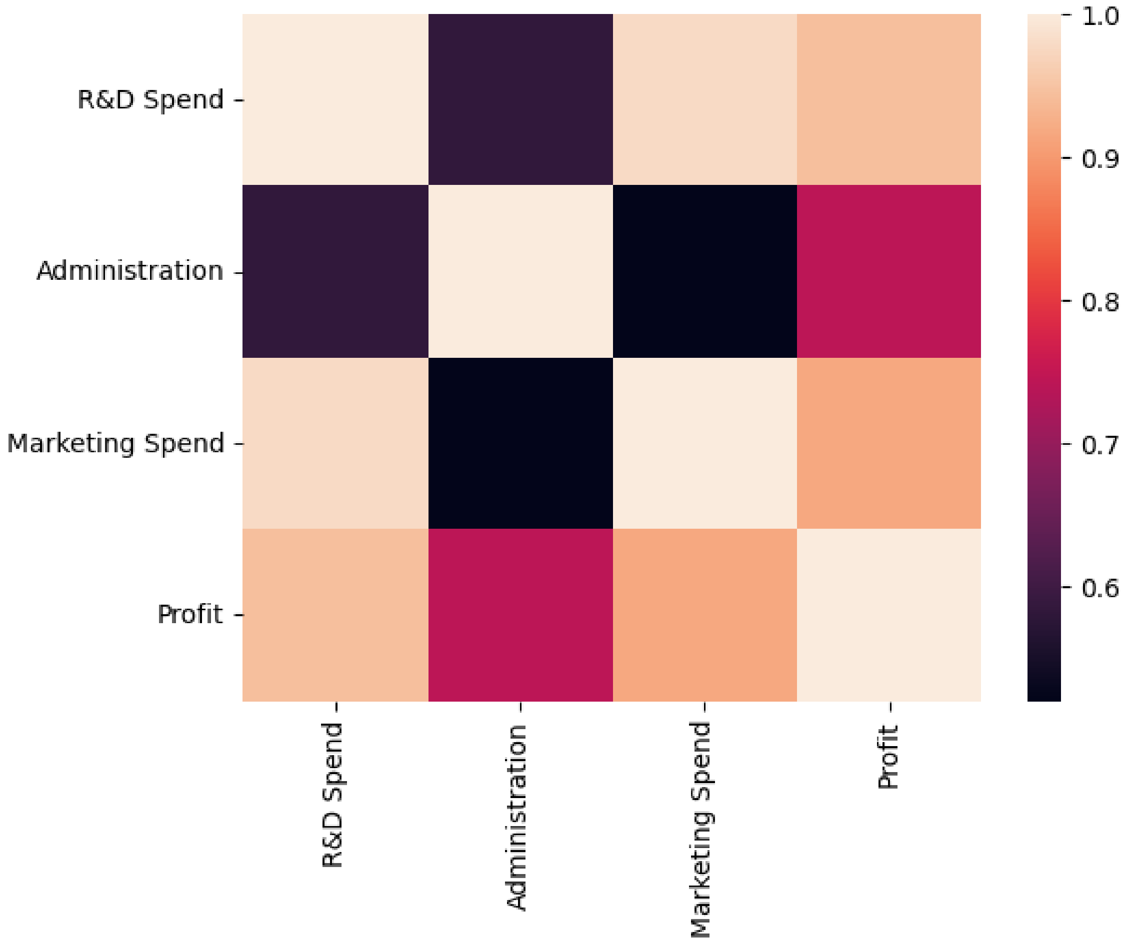
Out[4]:
```
R&D Spend          float64
Administration     float64
Marketing Spend    float64
State               object
Profit             float64
dtype: object
```

Data Visualizations

In [ ]:
```
# Building the Correlation Matrix
# Creating a heatmap of the correlation matrix of the DataFrame. This can be
sns.heatmap(profit_df.corr()) # More closer to 1, it denotes more connection
```

```
<ipython-input-5-5184aefa3c2d>:3: FutureWarning: The default value of nume
ric_only in DataFrame.corr is deprecated. In a future version, it will def
ault to False. Select only valid columns or specify the value of numeric_o
nly to silence this warning.
  sns.heatmap(profit_df.corr()) # More closer to 1, it denotes more connec
tion or correlation
```

Out[5]:
```
<Axes: >
```



Converting State (String to Int) using LabelEncoder

In [ ]:
```
profit_df['State'].value_counts()
```

Out[6]:
```
California    344
New York      334
Florida       322
Name: State, dtype: int64
```

In [ ]:
```
from sklearn.preprocessing import LabelEncoder
```

In [ ]:
```
Le = LabelEncoder()
```

In [ ]:
```
profit_df['State'] = Le.fit_transform(profit_df['State'])
profit_df['State'].head(8)
```

Out[9]:
```
0    2
1    0
2    1
3    2
4    1
5    2
6    0
7    1
Name: State, dtype: int64
```

Separating features and labels

In [ ]:
```
x = profit_df.drop(['Profit'], axis = 1)
x.tail()
```

Out[10]:

	R&D Spend	Administration	Marketing Spend	State
995	54135.00	118451.999	173232.6695	0
996	134970.00	130390.080	329204.0228	0
997	100275.47	241926.310	227142.8200	0
998	128456.23	321652.140	281692.3200	0
999	161181.72	270939.860	295442.1700	2

In [ ]:
```
y = profit_df['Profit']
y.head()
```

Out[11]:
```
0    192261.83
1    191792.06
2    191050.39
3    182901.99
4    166187.94
Name: Profit, dtype: float64
```

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

In [ ]:
```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3)
```

In [ ]:
```
# length or sample of train dataset
len(x_train)
```

Out[14]:
```
700
```

In [ ]:
```
# length or sample of test dataset
len(x_test)
```
Out[15]:
```
300
```

Using Linear Regression

Parameters used in Linear Regression:

The parameters used in Linear Regression include:

1. **Dependent Variable (Y)**: This is the variable that we want to predict or forecast.
2. **Independent Variable (X)**: These are the variables that we use to predict or forecast the dependent variable.
3. **Intercept (b0)**: This is the predicted value of Y when X is 0.
4. **Slope (b1)**: This is the regression coefficient, which represents the change in the dependent variable for a unit change in the independent variable.
5. **Error Term (ϵ)**: This is the difference between the observed and predicted values.
6. **fit_intercept**: This specifies if a constant (a.k.a. bias or intercept) should be added to the decision function.
7. **normalize**: This parameter is used to normalize the input variables (X) before regression.
8. **copy_X**: This parameter is used to copy the input variables (X). If False, the input variables may be overwritten during the normalization process.
9. **n_jobs**: This is the number of CPU cores used during cross-validation when the 'cross_validate' method is used.
10. **positive**: This parameter forces the coefficients to be positive when it is set to True. This is only applicable for certain solvers like 'lsqr', 'sparse_cg', and 'saga'.

In [ ]:
```
from sklearn.linear_model import LinearRegression
```
In [ ]:
```
lin = LinearRegression()
```
In [ ]:
```
# Fit the model
lin.fit(x_train, y_train)
```
Out[18]:
```
LinearRegression()
```
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [ ]:
```
# m is the slope of the line. It measures the steepness of the line or the r
m = lin.coef_
m
```
In [ ]:
```
# c is the y-intercept. It is the value of y when x is 0. In other words, it
c = lin.intercept_
c
```
Out[19]:
```
-69878.10482437348
```

In [ ]:
```
# Printing y = mx + c
# It is the slope-intercept form of a linear equation
Y_pred_train = m * x_train + c
Y_pred_train
```

Out[21]:

	R&D Spend	Administration	Marketing Spend	State
735	-15725.946067	56837.383101	-50444.564261	-69447.840082
627	-28356.955227	53488.137359	-53746.178320	-69878.104824
488	-8336.564930	58796.755788	-48513.057101	-69447.840082
186	-60075.263314	45077.692077	-62037.013245	-69447.840082
555	-30258.263445	52983.985322	-54243.160453	-69878.104824
...	...	...	...	...
283	-62531.214186	44426.470750	-62678.973187	-69878.104824
558	-2658.701113	60302.301477	-47028.922715	-69447.840082
246	12508.141058	64323.950221	-43064.468356	-69878.104824
473	-39037.367188	50656.112988	-56537.926616	-69447.840082
128	-16855.060275	56537.986534	-50739.702935	-69447.840082

700 rows × 4 columns

In [ ]:
```
y_test.head(8)
```

Out[22]:
```
940     94974.98049
172     65814.59883
761    119961.29450
345    100435.61090
915    106235.39500
875    159173.26840
259    158338.62580
1      191792.06000
Name: Profit, dtype: float64
```

In [ ]:
```
y_pred=lin.predict(x_test)
print(y_pred)
```

```
[ 95127.62106008  65176.50783547 120422.80409595  99795.22377353
 106096.96677029 159258.96841799 158844.27294007 210289.4346206
 110739.05938872 129793.90379497 165491.95464159 133951.08136229
  64868.18922501  63688.52945783 165708.60110257 162441.62472831
 154705.52607181 250451.34585631 146748.8902823   87719.73266256
  59334.38653938 111264.02239848 105433.19415916  81094.53504012
 120411.9924245   52693.23813107  50673.37594615 110630.08606903
  95382.75227148  97460.124136    79127.86427744  62207.03840788
  72483.2085765  149819.97671131  98266.59939504 174017.66497855
  72182.6716076   53277.01306511  56299.23838957  86506.77713715
 142886.02907949 151128.49939269 116608.80595612 104498.28945421
 144367.52012944  88813.33871911  85377.71418901  96923.91609589
  73361.03098528 163601.38883302 181847.1655343  136812.17550471
 152993.98456056 130334.86757009 115760.38647013 161379.15339065
 170045.83241601  82521.54241719 114509.81085809 120779.55437045
  63317.94151129  73946.53772004 152929.11866288 156993.92171806
 148935.23341251 103309.11624935  96789.86604774 162093.95432915
 170690.57834757 151548.8173311  168889.9571251  160034.30895046
 171841.26453689  88723.394133   113566.25964435 181466.19937692
 151997.67538675  89718.83637792 120449.18101816 143075.43168938
 177261.04756766 171654.02316977 200120.21355544  58625.68955453
  79645.47704458  53034.42315635 104412.66704847  82704.02412938
 161903.68695376  64348.84476348  68045.22101317  94872.92125529
  87074.12185611 119937.62221982 150427.53612623 180290.00103856
  56827.66076572 164910.34290566  57940.29349801 158248.82362614
 176723.35256261  58610.55354776  97092.1306894  168460.12780303
  93742.12857272 176312.11449955 127542.26336814 143737.04316714
  22376.10337156 150304.29568786 136280.29169897 145158.85938094
 171550.24296563  73035.41532796 134592.53260999  99772.25289243
  84177.30006949 141339.24083464 184788.09474292 158712.81553361
 133008.21611388 172659.41427293 172576.82199112 156540.73942804
  81289.12675927 112396.97808034 134506.91225825 154214.28825924
  91048.5469663  107557.26990558 169305.08792188  79190.13155184
  96801.10705645  64535.65267    140624.01044949  57538.56887185
 178878.12956732 119341.73738361 138094.75060882 127968.63322134
 132036.86525167 124235.06838567 161967.25255599 160173.98352844
 159454.42685492 102979.17635767 137085.46863018 105786.0536599
 183548.33089638 108754.22269799 183327.79366931 121972.18488845
  78210.68802595  89575.70449447 115300.28534409 150453.91304844
  65802.22905507 117417.44234853  86856.17716902 106297.61235652
 142102.90483611 100324.51297762  65259.10217895  50030.79320581
 133817.1940298  161616.12268797 165297.79650012 169644.10778295
 166858.42035533 109002.86853398 117845.54389612 108014.77653168
 104276.88529825 185930.99733074  95952.25617152 125971.25924041
 184915.65942334  99763.22418742 146060.89972661  55439.57583243
 117647.92627811  90257.64108136  79674.45062243 102329.25863779
 102222.8624971  170046.69728323 170749.39027645 179048.93876247
 127048.4330156  141043.46156072 164039.43727889 155926.69473943
  55967.99820857 177161.83231338 184493.61174283  94124.82445685
 185863.97029201 159180.70026116 138917.65597815 102866.31422418
 169081.9541332  185721.26981516  61027.38276097  58661.14644843
 162213.73529866 122391.63795964  50458.46128588 126073.74317073
  91232.32691227 128948.51020784 127608.85497863 162736.10566672
  73333.78920354 144713.89219309  49641.1764186  132657.42786341
  66148.16775984 111818.39430495 177780.63273701 154797.19833068
 137920.9154054   56801.28384812  90212.23532386 109567.61669133
 129957.36078754 176856.10830457 106460.63794249 165188.39373379
 105252.44011973 112070.30777571 144715.62192753 161602.71652426
 120913.17497962 158063.74536002  94903.19100525 106307.55915308
 123294.97457913 105866.48480067  51289.15242473  52238.75956722
 141998.25986647  92216.52814081 112993.72582381 138470.096258
 139828.77826084  72881.03831245  78255.22891624 143234.99780647
  88517.12598458 169536.00129044 154051.69613389  81151.6173363
 177753.82236187 175032.56746538 119944.97246245 183474.81642824
```

```
 153869.0959912  151676.33757568  80782.32565591 118296.26148867
 143455.53493182 169868.10643884 141795.88455348  45145.0187338
  88230.42659367  49742.79548017 162486.16355691 170566.9045503
 110453.65626396  87275.63230188 108310.12233732 119951.45783018
 145518.20631109 158020.50073588 181566.5221662  156695.54990416
 125566.50860676 107447.43171867 161546.93461759  85836.08558828
  63693.72062109  71236.95719882 -16077.57870618  79769.15084952
  82755.48385568 180037.03292049  52813.886031   100474.1323922
 143228.941776   112583.35468196 141209.5130609   84705.29103979
  99693.26540387 125798.72237249  65295.42394543  78694.14016765
 135436.62785394  55814.05455272  67036.37249514  65040.29468632]
```

Metrices used in Linear Regression

There are several metrics used to evaluate the performance of a Linear Regression model:

1. **R-squared (Coefficient of Determination)**: This metric provides an indication of the goodness of fit of a set of predictions to the actual values. In other words, it explains how much the total variance of the dependent variable can be reduced by using the least square regression.
2. **Adjusted R-squared**: This is a modified version of R-squared that has been adjusted for the number of predictors in the model. It increases only if the new term improves the model more than would be expected by chance.
3. **F-Test**: It is used to assess the significance of the overall regression model. Specifically, it tests the null hypothesis that all of the regression coefficients are equal to zero.
4. **Mean Absolute Error (MAE)**: This is the mean of the absolute value of the errors. It measures the average magnitude of the errors in a set of predictions, without considering their direction.
5. **Mean Squared Error (MSE)**: This is the mean of the squared errors. MSE is more popular than MAE because MSE "punishes" larger errors.
6. **Root Mean Squared Error (RMSE)**: This is the square root of the average of squared differences between prediction and actual observation. It's a measure of how spread out the residuals are, in other words, it tells you how concentrated the data is around the line of best fit.

Remember, the choice of metric depends on your specific problem and the business context. It's always a good idea to understand the assumptions and implications of each metric before choosing one.

Viewing the prediction score

In [ ]:
```
lin.score(x_test, y_test)
```

Out[24]:
```
0.9160968103820707
```

In [ ]:

```
from sklearn import metrics
from sklearn.metrics import mean_absolute_error
import statsmodels.api as sm

# R2 Score
r2 = metrics.r2_score(y_test, y_pred) # R2 score of 1 indicates that the moc
print(f'1. R2 Score: {r2}') # A negative R2 score indicates that your model

# Adjusted R2 Score
n = len(y_test)  # Number of observations
k = 1  # Number of predictors
adjusted_r2 = 1 - (1 - r2) * (n - 1) / (n - k - 1)
print(f'2. Adjusted R2 Score: {adjusted_r2}') # A negative R2 score indicate
print()

# Calculate F-test
model = sm.OLS(y_test, sm.add_constant(y_pred))
results = model.fit()
print(f"3. F-statistic: {results.fvalue}")
print(f"Prob (F-statistic): {results.f_pvalue}")
print()

# Calculate Mean Absolute Error (MAE)
mae = mean_absolute_error(y_test, y_pred)
print(f"4. Mean Absolute Error (MAE): {mae}")

# Mean Absolute Error (MAE)
mae = metrics.mean_absolute_error(y_test, y_pred)
print(f'5. MAE: {mae}') # 0.4167, means that on average, your model’s predic

# Root Mean Square Error (RMSE)
rmse = np.sqrt(metrics.mean_squared_error(y_test, y_pred))
print(f'6. RMSE: {rmse}') # 0.6455, means that on average, the square root c
```

```
1. R2 Score: 0.9160968103820707
2. Adjusted R2 Score: 0.9158152560544938

3. F-statistic: 3282.4360119071584
Prob (F-statistic): 6.382309478540762e-163

4. Mean Absolute Error (MAE): 2635.561564387359
5. MAE: 2635.561564387359
6. RMSE: 12360.989041117735
```

Predict Profit

In [ ]:

```
lin_state = LabelEncoder()
categories = ['California', 'New York', 'Florida']
lin_state.fit(categories)
```

Out[26]:

```
LabelEncoder()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [ ]:

```
rd = int(input("Enter the R&D Spend: "))

administration = int(input("Enter the Administration: "))

marketing = int(input("Enter the Marketing Spend: "))

state = input("Enter the State: ")
state_var = lin_state.transform([state])[0] # The [0] at the end is an index

profit = lin.predict([[rd, administration, marketing, state_var]])
print(profit)
```

```
Enter the R&D Spend: 100000
Enter the Administration: 40000
Enter the Marketing Spend: 80000
Enter the State: New York
[34464.17731977]

/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning:
X does not have valid feature names, but LinearRegression was fitted with
feature names
  warnings.warn(
```

Viewing the relation between R&D and Profit (To see Regression Line)

In [ ]:

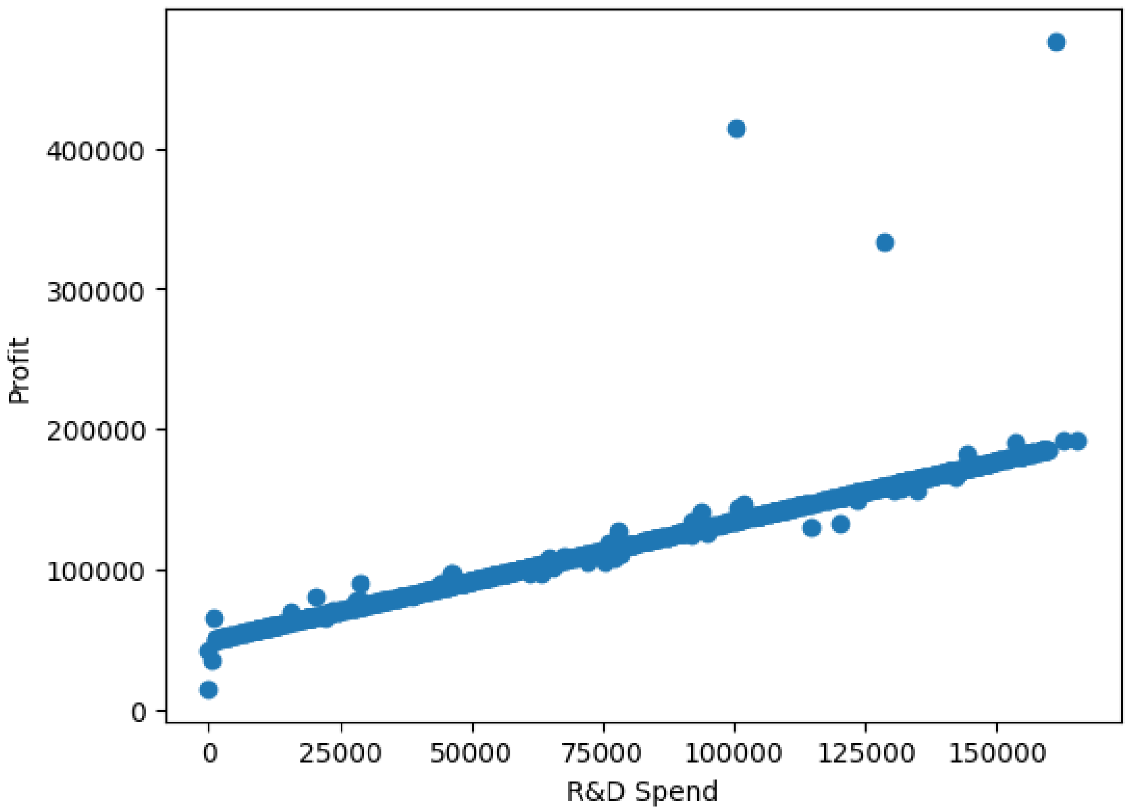
```
x_rd = profit_df['R&D Spend']
y_rd = profit_df['Profit']
```

In [ ]:

```
plt.scatter(x_rd, y_rd)
plt.xlabel('R&D Spend')
plt.ylabel('Profit')
```

Out[29]:

```
Text(0, 0.5, 'Profit')
```



In [ ]:
```
x_train, x_test, y_train, y_test = train_test_split(x_rd, y_rd, test_size=0.
```

In [ ]:
```
# Converting in 2D array
x_train = np.array(x_train).reshape(-1, 1)
y_train = np.array(y_train).reshape(-1, 1)

x_test = np.array(x_test).reshape(-1, 1)
y_test = np.array(y_test).reshape(-1, 1)
```

In [ ]:
```
y_train
```

Out[33]:
```
array([[ 52225.38599],
       [ 90550.60548],
       [ 74208.01155],
       [ 52609.81711],
       [ 94376.97653],
       [123690.2763 ],
       [123671.4819 ],
       [165941.819  ],
       [109877.2392 ],
       [114839.8177 ],
       [184632.0056 ],
       [ 70555.91594],
       [ 60243.7648 ],
       [ 77242.4545 ],
       [ 53395.76517],
       [142575.2414 ],
       [ 71376.03566],
       [144287.2413 ],
       [106070.5168 ],
```

Implementing y = mx + c

In [ ]:
```
lin = LinearRegression()

# Fit the model
lin.fit(x_train, y_train)

c = lin.intercept_
m = lin.coef_

Y_pred_train = m * x_train + c
Y_pred_train
```

Out[34]:
```
array([[ 52247.94597667],
       [ 90558.04484284],
       [ 74221.89863202],
       [ 52632.2254223 ],
       [ 94382.90625832],
       [123684.64096464],
       [123665.85396952],
       [165919.51390215],
       [109877.05350605],
       [114837.67417195],
       [184602.32659427],
       [ 70571.24389855],
       [ 60263.16125816],
       [ 77255.14438951],
       [ 53417.86340003],
       [142562.15524254],
       [ 71391.04004923],
       [144273.47970708],
       [106071.83304001],
```

Predicting x_train with y_train

In [ ]:
```
y_pred_train=lin.predict(x_train)
print(y_pred)
```

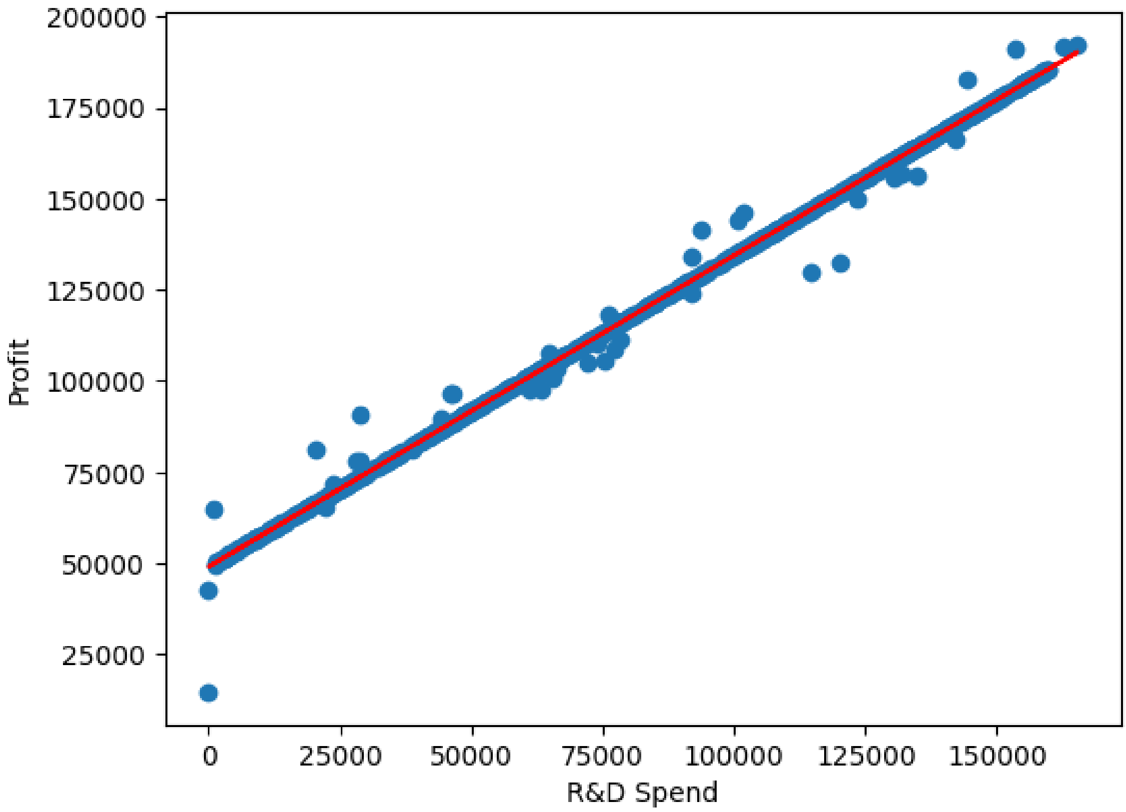
```
[ 95127.62106008  65176.50783547 120422.80409595  99795.22377353
 106096.96677029 159258.96841799 158844.27294007 210289.4346206
 110739.05938872 129793.90379497 165491.95464159 133951.08136229
  64868.18922501  63688.52945783 165708.60110257 162441.62472831
 154705.52607181 250451.34585631 146748.8902823   87719.73266256
  59334.38653938 111264.02239848 105433.19415916  81094.53504012
 120411.9924245   52693.23813107  50673.37594615 110630.08606903
  95382.75227148  97460.124136    79127.86427744  62207.03840788
  72483.2085765  149819.97671131  98266.59939504 174017.66497855
  72182.6716076   53277.01306511  56299.23838957  86506.77713715
 142886.02907949 151128.49939269 116608.80595612 104498.28945421
 144367.52012944  88813.33871911  85377.71418901  96923.91609589
  73361.03098528 163601.38883302 181847.1655343  136812.17550471
 152993.98456056 130334.86757009 115760.38647013 161379.15339065
 170045.83241601  82521.54241719 114509.81085809 120779.55437045
  63317.94151129  73946.53772004 152929.11866288 156993.92171806
 148935.23341251 103309.11624935  96789.86604774 162093.95432915
 170690.57834757 151548.8173311  168889.9571251  160034.30895046
 171841.26453689  88723.394133   113566.25964435 181466.19937692
 151997.67538675  89718.83637792 120449.18101816 143075.43168938
 177261.04756766 171654.02316977 200120.21355544  58625.68955453
  79645.47704458  53034.42315635 104412.66704847  82704.02412938
 161903.68695376  64348.84476348  68045.22101317  94872.92125529
  87074.12185611 119937.62221982 150427.53612623 180290.00103856
  56827.66076572 164910.34290566  57940.29349801 158248.82362614
 176723.35256261  58610.55354776  97092.1306894  168460.12780303
  93742.12857272 176312.11449955 127542.26336814 143737.04316714
  22376.10337156 150304.29568786 136280.29169897 145158.85938094
 171550.24296563  73035.41532796 134592.53260999  99772.25289243
  84177.30006949 141339.24083464 184788.09474292 158712.81553361
 133008.21611388 172659.41427293 172576.82199112 156540.73942804
  81289.12675927 112396.97808034 134506.91225825 154214.28825924
  91048.5469663  107557.26990558 169305.08792188  79190.13155184
  96801.10705645  64535.65267    140624.01044949  57538.56887185
 178878.12956732 119341.73738361 138094.75060882 127968.63322134
 132036.86525167 124235.06838567 161967.25255599 160173.98352844
 159454.42685492 102979.17635767 137085.46863018 105786.0536599
 183548.33089638 108754.22269799 183327.79366931 121972.18488845
  78210.68802595  89575.70449447 115300.28534409 150453.91304844
  65802.22905507 117417.44234853  86856.17716902 106297.61235652
 142102.90483611 100324.51297762  65259.10217895  50030.79320581
 133817.1940298  161616.12268797 165297.79650012 169644.10778295
 166858.42035533 109002.86853398 117845.54389612 108014.77653168
 104276.88529825 185930.99733074  95952.25617152 125971.25924041
 184915.65942334  99763.22418742 146060.89972661  55439.57583243
 117647.92627811  90257.64108136  79674.45062243 102329.25863779
 102222.8624971  170046.69728323 170749.39027645 179048.93876247
 127048.4330156  141043.46156072 164039.43727889 155926.69473943
  55967.99820857 177161.83231338 184493.61174283  94124.82445685
 185863.97029201 159180.70026116 138917.65597815 102866.31422418
 169081.9541332  185721.26981516  61027.38276097  58661.14644843
 162213.73529866 122391.63795964  50458.46128588 126073.74317073
  91232.32691227 128948.51020784 127608.85497863 162736.10566672
  73333.78920354 144713.89219309  49641.1764186  132657.42786341
  66148.16775984 111818.39430495 177780.63273701 154797.19833068
 137920.9154054   56801.28384812  90212.23532386 109567.61669133
 129957.36078754 176856.10830457 106460.63794249 165188.39373379
 105252.44011973 112070.30777571 144715.62192753 161602.71652426
 120913.17497962 158063.74536002  94903.19100525 106307.55915308
 123294.97457913 105866.48480067  51289.15242473  52238.75956722
 141998.25986647  92216.52814081 112993.72582381 138470.096258
 139828.77826084  72881.03831245  78255.22891624 143234.99780647
  88517.12598458 169536.00129044 154051.69613389  81151.6173363
 177753.82236187 175032.56746538 119944.97246245 183474.81642824
```

```
 153869.0959912   151676.33757568  80782.32565591 118296.26148867
 143455.53493182 169868.10643884 141795.88455348  45145.0187338
  88230.42659367  49742.79548017 162486.16355691 170566.9045503
 110453.65626396  87275.63230188 108310.12233732 119951.45783018
 145518.20631109 158020.50073588 181566.5221662  156695.54990416
 125566.50860676 107447.43171867 161546.93461759  85836.08558828
  63693.72062109  71236.95719882 -16077.57870618  79769.15084952
  82755.48385568 180037.03292049  52813.886031   100474.1323922
 143228.941776   112583.35468196 141209.5130609   84705.29103979
  99693.26540387 125798.72237249  65295.42394543  78694.14016765
 135436.62785394  55814.05455272  67036.37249514  65040.29468632]
```

In [ ]:
```
plt.scatter(x_train, y_train)
plt.plot(x_train, y_pred_train, color = 'red')

plt.xlabel('R&D Spend')
plt.ylabel('Profit')
```

Out[36]:
```
Text(0, 0.5, 'Profit')
```



Predicting x_test with y_test

In [ ]:
```
y_pred_test = lin.predict(x_test)
print(y_pred_test)
```

```
[[ 62241.77342598]
 [130418.92476069]
 [146484.36745092]
 [ 49519.60461043]
 [127068.86194914]
 [144736.32295048]
 [154521.78554482]
 [144793.53789016]
 [149833.57630815]
 [111421.85687748]
 [165562.56099488]
 [118940.92469694]
 [ 63680.68646128]
 [101974.56019529]
 [ 75981.898493   ]
 [173836.52443642]
 [154339.03931956]
 [164314.93372807]
 [ 85839.94720485]
```

In [ ]:
```
plt.scatter(x_test, y_test)
plt.plot(x_test, y_pred_test, color = 'red')

plt.xlabel('R&D Spend')
plt.ylabel('Profit')
```

Out[38]: Text(0, 0.5, 'Profit')