Department of Data Science

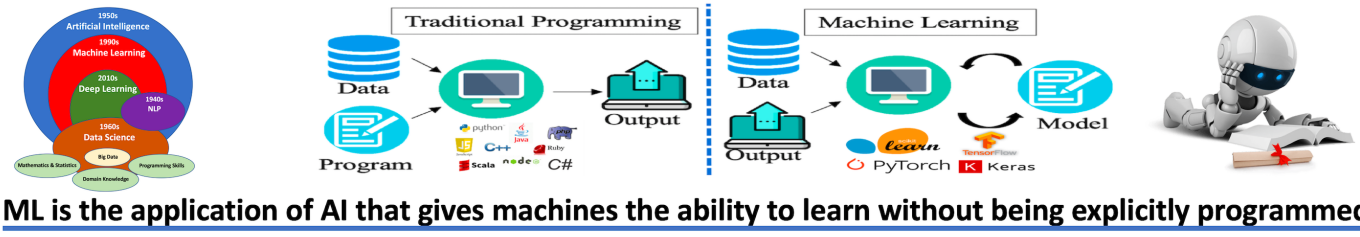
Course: Tools and Techniques for Data Science

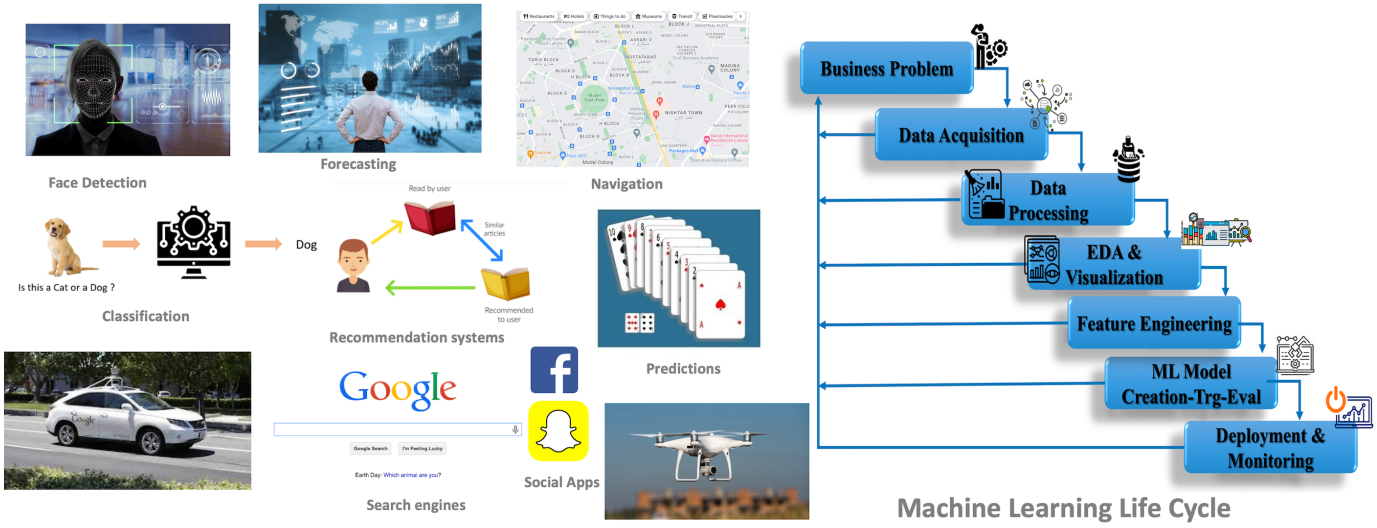
Instructor: Muhammad Arif Butt, Ph.D.

Lecture 6.3 (Multiple Linear Regression: Math Behind The Curtain)

Open in Colab

(https://colab.research.google.com/github/arifpucit/data-science/blob/master/Section-4-Mathematics-for-Data-Science/Lec-4.1(Descriptive-Statistics).ipynb)





Learning agenda of this notebook

1. Simple Linear Regression (A Recap)
2. Solving System of Linear Equations using Matrix Algebra (A Recap)
3. What is Multiple Linear Regression and Why to do it?
4. How to Do Multiple Linear Regression using Matrices?

5. Example (Multiple Linear Regression using OLS)
6. Task To Do

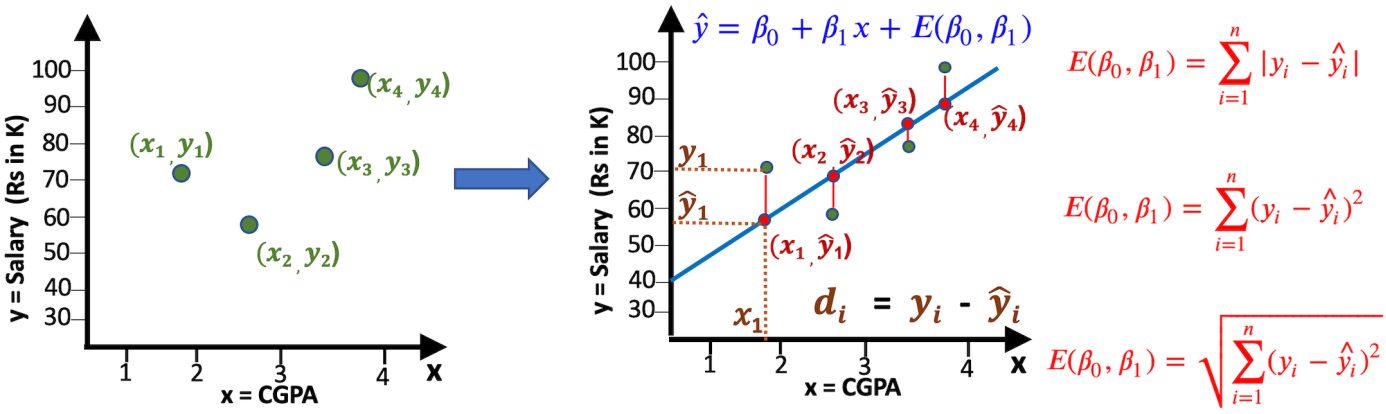
1. What we have done so far?

- Data Acquisition
- Data Preprocessing
- Choosing the right estimator
- Fit or train the model on training data
- Do predictions on the test data
- **Improve the Baseline Model**
  - From a data perspective
    - Could we collect more data?
    - Could we improve our data
      - Add more relevent features (if possible_
      - Drop irrelevant features (

1. Simple Linear Regression (A Recap)

- Linear regression is a type of model where the relationship between a dependent variable and one or more independent variables is assumed to be linear.
- There are two kinds of Linear Regression Model:
  - Simple Linear Regression: A linear regression model with one independent and one dependent variable.
  - Multiple Linear Regression: A linear regression model with more than one independent variable and one dependent variable.
- The process of fitting the best-fit line is called simple linear regression.
- Ordinary Least Squares is a technique that works by minimizing the sum of squares of the differences between the observed dependent variable (cgpa) and the independent variable (salary).

Dependent Variable (y)	**Independent Variable (x)**
GPA of a student	Number of hours studied daily
Forgetness level	Drug dosage in ml
Salary of a person	Number of Education years
House price	Covered area of the house
Electricity Bill	Amount of electricity consumed
Distance travelled	Time
Sales	Advertising Expenditures
Expense	Cost of an apple



2. Solving System of Linear Equations using Matrix Algebra (A Recap)

Solving System of Linear Equations: https://www.youtube.com/watch?v=CLGbFiZStJU&list=PL7B2bn3G_wfAs3C49i12i_rblzvuU1dFN&index=53&t=4885s

$$A_{n\times m} \quad x_{m\times 1} = b_{n\times 1}$$

$$x = A^{-1} \quad b$$

$$X_{n\times m} \quad \beta_{m\times 1} = Y_{n\times 1}$$

$$\beta = X^{-1} \quad Y$$

- **Overdetermined System:** $n > m$
- **Underdetermined System:** $n < m$

- **Possible Solutions for Non-Square Matrix X:**
  - Gaussian Elimination
  - Cramer's Rule
  - Simple Matrix Inversion
- **Possible Solutions for Non-Square Matrix X:**
  - Gauss Jordan Elimination
  - Moore Penrose Pseudo-inverse
  - SVD, QR Decomposition, and Cholesky Decomposition

3. What is Multiple Linear Regression and Why to do it?

a. Overview of Multiple Regression

Dependent Variable (y)	Independent Variable (x_1)	Independent Variable (x_2)	Independent Variable (x_3)
GPA of a student	Number of hours studied daily	Teaching Aids	Instructor Qualification
Forgetness level	Drug dosage in ml	Social bindings	Family Environment
Salary of a person	Number of Education years	Managerial skills	Communication skills
House price	Covered area of the house	Number of bed rooms	distance from office
Electricity Bill	Amount of electricity consumed	Loadshedding	Billing slabs
Distance travelled	Time	type of car	traffic conditions
Sales of AC	Advertising Expenditures	Marketing	Season

$$\hat{y} = \beta_0 + \beta_1 x_1$$
$$\hat{y} = \beta_0 + \beta_1 x_1 + \beta_2 x_2$$
$$\hat{y} = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3$$
$$\vdots$$
$$\hat{y} = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \cdots + \beta_n$$
$$\hat{y} = \beta_0 + \sum_{i=1}^{m} \beta_i x_i$$

- Where,
  - $\hat{y}$ is the predicted dependent or outcome or response variable.
  - $x_1, x_2, x_3, \cdots, x_m$ are the independent or feature or predictor variables.
  - β_0 is the minimum value of y when all the feature variables are zero (constant term).
  - $\beta_1, \beta_2, \beta_3, \cdots, \beta_m$ are the values that quantify the effect of respective independent variables on the output or dependent variable.

b. Points to Ponder for Multiple Regression

In [1]:

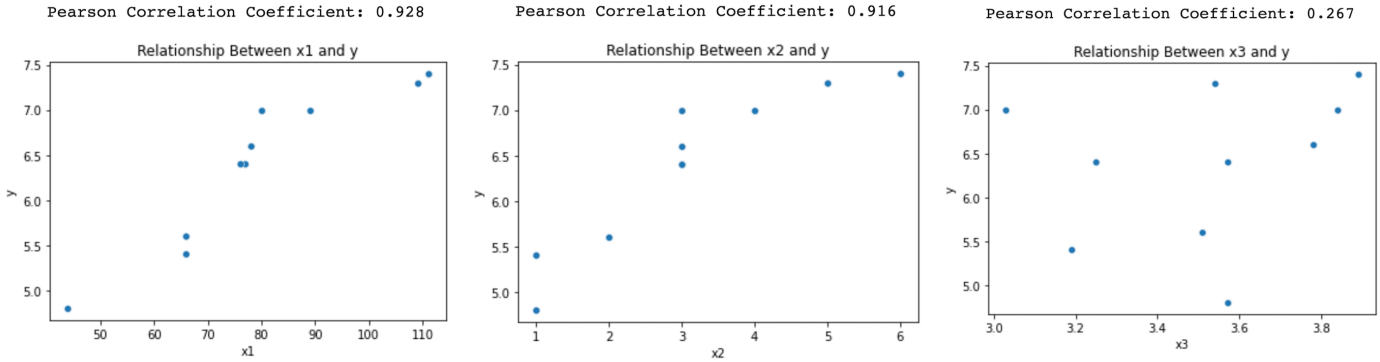
```
import numpy as np
import pandas as pd
import matplotlib
from matplotlib import pyplot as plt
import seaborn as sns
df = pd.read_csv("datasets/sampledata4D.csv")
df
```

Out[1]:

	x1	x2	x3	y
0	89	4	3.84	7.0
1	66	1	3.19	5.4
2	78	3	3.78	6.6
3	111	6	3.89	7.4
4	44	1	3.57	4.8
5	77	3	3.57	6.4
6	80	3	3.03	7.0
7	66	2	3.51	5.6
8	109	5	3.54	7.3
9	76	3	3.25	6.4

(i) Overfitting

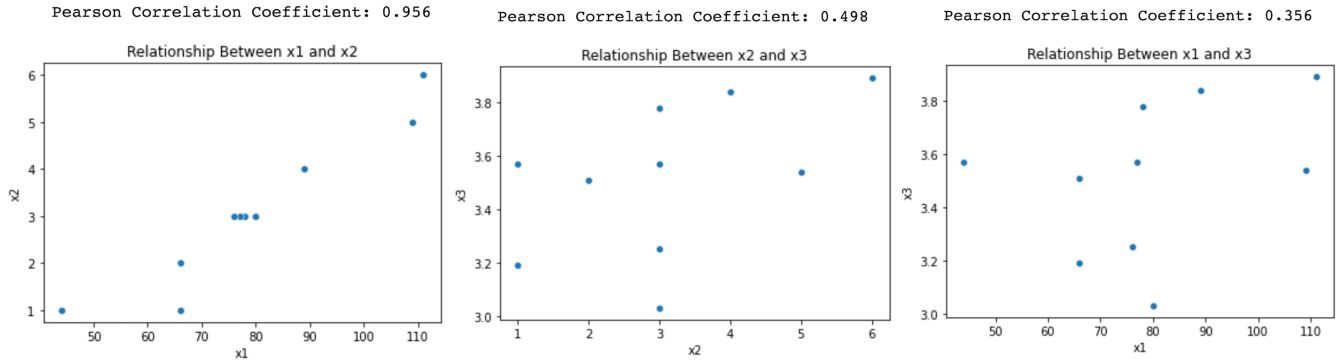
- Overfitting is caused by adding too many feature variables, which account for more variance but add nothing to the model.
- Multicollinearity occurs when some or all of the independent variables are correlated with eachother.
- We can check the relationships between each independent variable and the dependent variable using scatter plot and correlations.
- Since there are three input variables, so we have three relationships to analyze (between three feature variables and the only output variable).
- Before using multiple linear regression, one must ensure that there should exist a linear relationship between the dependent variable (y), and all the independent variables (x_1, x_2, x_3)



- The feature x_3 is adding high variance in the model. A model with high variance pays a lot of attention to traning data and does not generalize on the data which it hasn't seen before. As a result such models perform very well on training data, but has high error rates on test data. (More on this later)

(ii) Multicollinearity

- Multicollinearity occurs when some or all of the independent variables are correlated with eachother.
- Since there are three input or feature variables, so we have three relationships among them to analyze.



All the feature variables should be somehow correlated with the output variable,
but not with eachother

Multicollinearity may not affect the accuracy of the model as much but we might lose reliability in determining the effects of individual independent features on the output variable in your model and that can be a problem when we want to interpret our model.

4. How to Do Multiple Linear Regression using Matrices?

a. Formulation of Multiple Linear Regression using Matrix Representation

$$\hat{y} = \beta_0 + \sum_{i=1}^{m} \beta_i x_i$$

$$\begin{aligned}
\hat{y_1} &= \beta_0 + \beta_1 X_{1,1} + \beta_2 X_{1,2} + \beta_3 X_{1,3} + \cdots + \beta_m X_{1,m} \\
\hat{y_2} &= \beta_0 + \beta_1 X_{2,1} + \beta_2 X_{2,2} + \beta_3 X_{2,3} + \cdots + \beta_m X_{2,m} \\
\hat{y_3} &= \beta_0 + \beta_1 X_{3,1} + \beta_2 X_{3,2} + \beta_3 X_{3,3} + \cdots + \beta_m X_{3,m} \\
&\vdots
\end{aligned}$$

$$\hat{y}_n = \beta_0 + \beta_1 X_{n,1} + \beta_2 X_{n,2} + \beta_3 X_{n,3} + \cdots + \beta_m X_{n,m}$$

- The above overdetermined system of linear equations with n equations and m+1 unknowns can be written in matrix form as:

$$\begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \hat{y}_3 \\ \vdots \\ \hat{y}_n \end{bmatrix} = \begin{bmatrix} \beta_0 + \beta_1 X_{1,1} + \beta_2 X_{1,2} + \beta_3 X_{1,3} + \cdots + \beta_m X_{1,m} \\ \beta_0 + \beta_1 X_{2,1} + \beta_2 X_{2,2} + \beta_3 X_{2,3} + \cdots + \beta_m X_{2,m} \\ \beta_0 + \beta_1 X_{3,1} + \beta_2 X_{3,2} + \beta_3 X_{3,3} + \cdots + \beta_m X_{3,m} \\ \vdots \\ \beta_0 + \beta_1 X_{n,1} + \beta_2 X_{n,2} + \beta_3 X_{n,3} + \cdots + \beta_m X_{n,m} \end{bmatrix}$$

$$\begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \hat{y}_3 \\ \vdots \\ \hat{y}_n \end{bmatrix} = \begin{bmatrix} 1 & X_{1,1} & X_{1,2} & X_{1,3} & \cdots & X_{1,m} \\ 1 & X_{2,1} & X_{2,2} & X_{2,3} & \cdots & X_{2,m} \\ 1 & X_{3,1} & X_{3,2} & X_{3,3} & \cdots & X_{3,m} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & X_{n,1} & X_{n,2} & X_{n,3} & \cdots & X_{n,m} \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \vdots \\ \beta_m \end{bmatrix}$$

$$\hat{Y}_{n\times 1} = X_{n\times(m+1)} \quad \beta_{(m+1)\times 1}$$

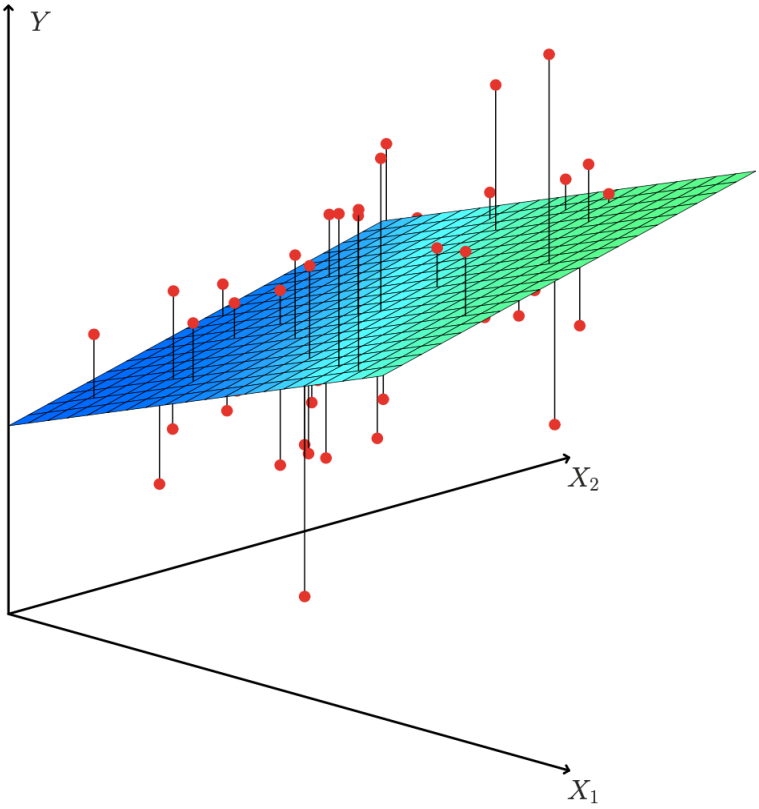
For a single input feature:

$$\begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \hat{y}_3 \\ \vdots \\ \hat{y}_n \end{bmatrix} = \begin{bmatrix} 1 & X_{1,1} \\ 1 & X_{2,1} \\ 1 & X_{3,1} \\ \vdots & \vdots \\ 1 & X_{n,1} \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix}$$

For two input features:

$$\begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \hat{y}_3 \\ \vdots \\ \hat{y}_n \end{bmatrix} = \begin{bmatrix} 1 & X_{1,1} & X_{1,2} \\ 1 & X_{2,1} & X_{2,2} \\ 1 & X_{3,1} & X_{3,2} \\ \vdots & \vdots & \vdots \\ 1 & X_{n,1} & X_{n,2} \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \end{bmatrix}$$

b. Intuitive Understanding of the Cost Function for Multiple Linear Regression



$$y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + E(\beta_0, \beta_1, \beta_2)$$

$$y = \hat{y} + E(\beta_0, \beta_1, \beta_2)$$

$$E(\beta_0, \beta_1, \beta_2) = y - \hat{y}$$

c. Derivation of $\beta_0, \beta_1, \beta_2, \cdots, \beta_m$ for MLR using OLS Method

- System of Linear equations for the residual errors can be written as:

$$\begin{aligned} e_1 &= y_1 - \hat{y_1} \\ e_2 &= y_2 - \hat{y_2} \\ e_3 &= y_3 - \hat{y_3} \\ &\vdots \\ e_n &= y_n - \hat{y_n} \end{aligned}$$

- The above equations can be written in matrix form:

$$E = \begin{bmatrix} e_1 \\ e_2 \\ e_3 \\ \vdots \\ e_n \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_n \end{bmatrix} - \begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \hat{y}_3 \\ \vdots \\ \hat{y}_n \end{bmatrix} = \begin{bmatrix} y_1 - \hat{y}_1 \\ y_2 - \hat{y}_2 \\ y_3 - \hat{y}_3 \\ \vdots \\ y_n - \hat{y}_n \end{bmatrix}$$

$$E = y_i - \hat{y}_i$$

- We need to minimize the sum of squares of errors:

$$SSE = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

$$SSE = \sum_{i=1}^{n} E_i^2$$

$$SSE = E^T E$$

- Substitute $E = y - \hat{y}$:

$$SSE = (y - \hat{y})^T (y - \hat{y})$$

$$SSE = (y^T - \hat{y}^T)(y - \hat{y}) \qquad \ldots\ldots\ldots (i)$$

- Substitute $\hat{y} = X\beta$

$$SSE = (y^T - (X\beta)^T)(y - X\beta)$$

$$SSE = y^T y - y^T X\beta - (X\beta)^T y + (X\beta)^T)(X\beta) \qquad \ldots\ldots\ldots (ii$$

- We know that $(A^T B)^T = B^T A$, therefore, in above expression $y^T X\beta = (X\beta)^T)y$

$$SSE = y^T y - y^T X\beta - y^T X\beta + \beta^T X^T X\beta$$

$$SSE = y^T y - 2y^T X\beta + \beta^T X^T X\beta \qquad \ldots\ldots\ldots (iii)$$

- Now we need to perform matrix differentiation of eq(iii), w.r.t. vector β and equate to zero:

$$\frac{\partial}{\partial\beta}(y^T y - 2y^T X\beta + \beta^T X^T X\beta) = 0$$

$$\frac{\partial}{\partial\beta}(y^T y) - \frac{\partial}{\partial\beta}(2y^T X\beta) + \frac{\partial}{\partial\beta}(\beta^T X^T X\beta) = 0 \qquad .....(iv)$$

Recap of some formulas of Matrix Differentiation:

- If y is $n \times 1$ vector, β is a $m \times 1$ vector and X is an $n \times m$ matrix. Moreover, if matrix X is independent of vector β, then we can use the following formulas for matrix differentiation:

$y = X$, then $\quad \frac{\partial y}{\partial\beta} = 0$

$y = X\beta$, then $\quad \frac{\partial y}{\partial\beta} = X$

$y = \beta X$, then $\quad \frac{\partial y}{\partial\beta} = X^T$

$y = \beta^T X\beta$, then $\quad \frac{\partial y}{\partial\beta} = 2\beta^T X$

$$\frac{\partial}{\partial\beta}(y^T y) - \frac{\partial}{\partial\beta}(2y^T X\beta) + \frac{\partial}{\partial\beta}(\beta^T X^T X\beta) = 0 \qquad .....(iv)$$

- Differentiating equation (iv), using above formulas, we get:

$$0 - 2y^T X + 2\beta^T X^T X = 0 \qquad .....(v)$$

$$2\beta^T X^T X = 2y^T X$$

$$\beta^T X^T X = y^T X$$

$$\beta^T = y^T X(X^T X)^{-1}$$

- Taking transpose of both sides:

$$(\beta^T)^T = (y^T X(X^T X)^{-1})^T$$

$$\beta = ((X^T X)^{-1})^T (y^T X)^T$$

$$\beta = ((X^T X)^{-1})^T (X^T y)$$

- Now $X^T X$ is a symmetric matrix, therefore, its transpose doesn't change its value:

$$\beta = (X^T X)^{-1} X^T y$$

> - Taking the derivative of cost function and then solving it for zero to get the set of β coefficients becomes computationally expensive when the number of input features are in hundreds or thousands.
> - The Gauss Jordan Elimination technique of computing the inverse of a nxn matrix is $O(n^3)$. So for 999 input features the size of X matrix will be 1000×1000, and the complexity of computing the inverse of this matrix will be 10^9
> - Solution to this problem is `Gradient Descent` in which instead of using a closed form solution, we go slowly towards the answer using approximation technique. (Next Session)

5. Example (Simple Linear Regression using Matrices Algebra)

a. Load Dataset

In [23]:

```
import pandas as pd
import numpy as np
df = pd.read_csv("datasets/study-hours.csv")
df
```

Out[23]:

	study_hours	gpa
0	1.0	1.4
1	2.0	1.6
2	3.0	2.5
3	4.0	2.6
4	5.0	3.5
5	6.0	3.7
6	7.0	4.0

b. Create X matrix and y Vector

In [24]:

```
sh = np.array(df['study_hours'])
y = np.array(df['gpa'])
sh, y
```

Out[24]:

```
(array([1., 2., 3., 4., 5., 6., 7.]),
 array([1.4, 1.6, 2.5, 2.6, 3.5, 3.7, 4. ]))
```

In [25]:

```
sh = sh.reshape(sh.size, 1)
sh
```

Out[25]:

```
array([[1.],
       [2.],
       [3.],
       [4.],
       [5.],
       [6.],
       [7.]])
```

In [26]:

```
X = np.hstack((np.ones((sh.size,1)), sh)) # stack arrays horizontally or column-
X
```

Out[26]:

```
array([[1., 1.],
       [1., 2.],
       [1., 3.],
       [1., 4.],
       [1., 5.],
       [1., 6.],
       [1., 7.]])
```

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

c. Calculate the Regression Coefficients

Using Close Form Solution: $\beta = (X^T X)^{-1} X^T Y$

In [27]:

```
beta = np.linalg.inv(X.T@X) @ X.T @y
#beta = np.linalg.pinv(X)@y
beta
```

Out[27]:

```
array([0.9       , 0.46428571])
```

Using `LinearRegression` Model:

In [28]:

```
from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(sh, y) #Note that we have just passed the i/p feature as a column vect
print("Y-intercept (β0): ", model.intercept_)
print("Slope (β1): ", model.coef_)
```

```
Y-intercept (β0):  0.9000000000000006
Slope (β1):  [0.46428571]
```

d. Carry out the Prediction

- What will be the gpa if a student has studied for five hours daily

$$y = \beta_0 + \beta_1 x$$
$$gpa = 0.9 + 0.464 * sh$$

In [29]:

```
studyhours = 5.0
predicted_gpa = beta[0] + beta[1] * studyhours
print("Predicted GPA value for new study hours: {:.4f}".format(predicted_gpa)) #
```

```
Predicted GPA value for new study hours: 3.2214
```

Using `LinearRegression` Model:

In [30]:

```
predicted_gpa = model.predict([[studyhours]])
print("Predicted GPA value for new study hours: {:.4f}".format(predicted_gpa[0])
```

```
Predicted GPA value for new study hours: 3.2214
```

6. Example (Multiple Linear Regression using OLS)

In [2]:

```
import numpy as np
import pandas as pd
import matplotlib
from matplotlib import pyplot as plt
import seaborn as sns
df = pd.read_csv("datasets/advertising4D.csv")
df
```

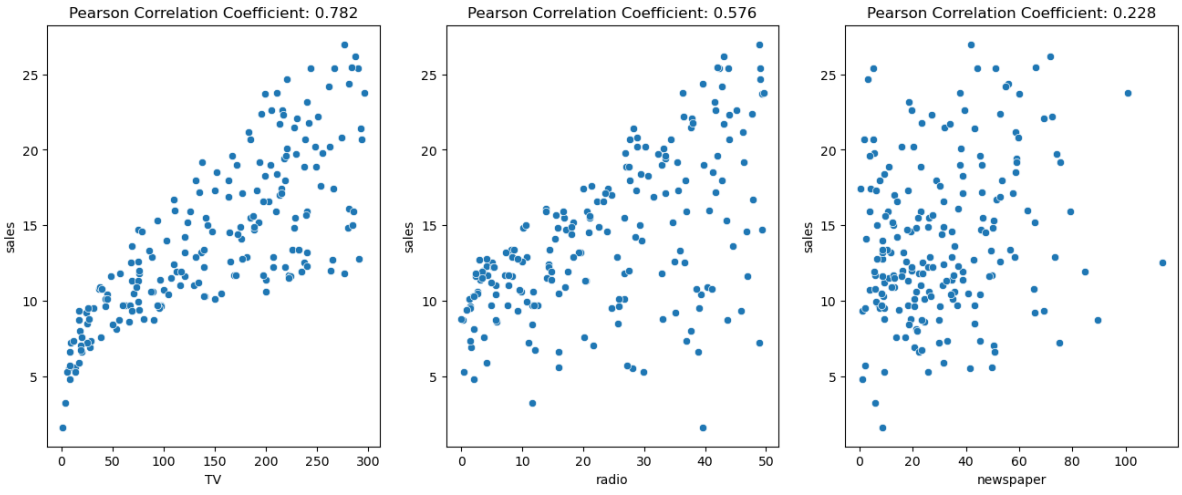
Out[2]:

	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
...	...	...	...	...
195	38.2	3.7	13.8	7.6
196	94.2	4.9	8.1	9.7
197	177.0	9.3	6.4	12.8
198	283.6	42.0	66.2	25.5

a. Relationship between Output (Sales) and Feature Variables

In [3]:

```
tv = np.array(df['TV'])
radio = np.array(df['radio'])
newspaper = np.array(df['newspaper'])

x=tv
y = np.array(df['sales'])
numerator = sum(np.multiply(x - np.mean(x), y - np.mean(y)))
denominator = np.multiply((np.sqrt(np.sum((x-np.mean(x))**2))), np.sqrt(np.sum((
corr_coef_tv = numerator / denominator

x=radio
y = np.array(df['sales'])
numerator = sum(np.multiply(x - np.mean(x), y - np.mean(y)))
denominator = np.multiply((np.sqrt(np.sum((x-np.mean(x))**2))), np.sqrt(np.sum((
corr_coef_radio = numerator / denominator

x=newspaper
y = np.array(df['sales'])
numerator = sum(np.multiply(x - np.mean(x), y - np.mean(y)))
denominator = np.multiply((np.sqrt(np.sum((x-np.mean(x))**2))), np.sqrt(np.sum((
corr_coef_newspaper = numerator / denominator


fig, (ax1,ax2, ax3) = plt.subplots(1,3, figsize=(16,6))
ax1.set_title("Pearson Correlation Coefficient: {:.3f}".format(corr_coef_tv))
sns.scatterplot(x='TV', y='sales', data=df, ax=ax1)
ax2.set_title("Pearson Correlation Coefficient: {:.3f}".format(corr_coef_radio))
sns.scatterplot(x='radio', y='sales', data=df, ax=ax2)
ax3.set_title("Pearson Correlation Coefficient: {:.3f}".format(corr_coef_newspap
sns.scatterplot(x='newspaper', y='sales', data=df, ax=ax3);
```

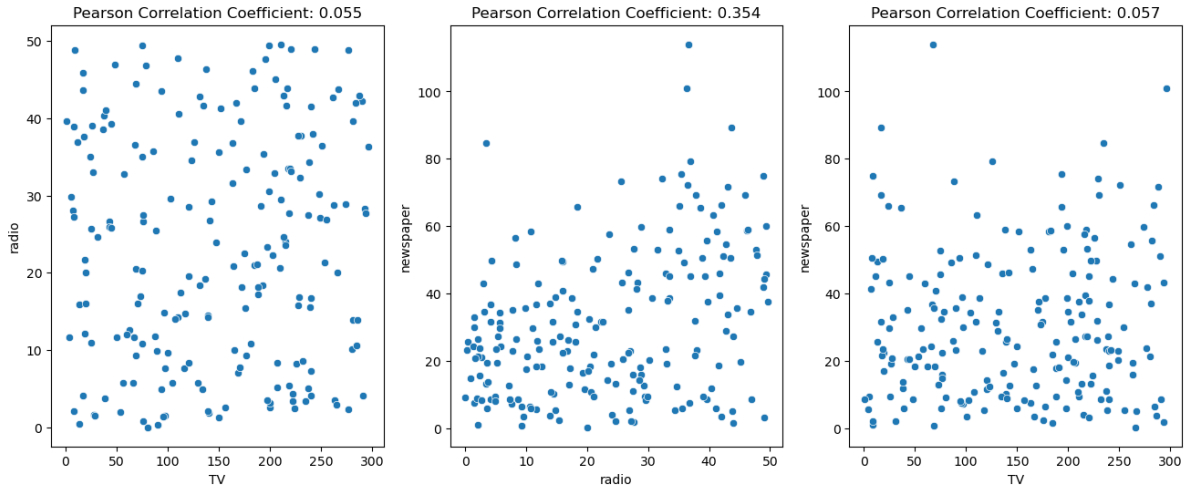


- One may be interested in answering questions such as:
  - Which media are associated with sales?
  - Which media generate the biggest boost in sales? or
  - How large of an increase in sales is associated with a given increase in TV advertising?

b. Relationship Among Feature Variables

In [4]:

```
tv = np.array(df['TV'])
radio = np.array(df['radio'])
newspaper = np.array(df['newspaper'])

x=tv
y = radio
numerator = sum(np.multiply(x - np.mean(x), y - np.mean(y)))
denominator = np.multiply((np.sqrt(np.sum((x-np.mean(x))**2))), np.sqrt(np.sum((
corr_coef_tv_radio = numerator / denominator

x=radio
y = newspaper
numerator = sum(np.multiply(x - np.mean(x), y - np.mean(y)))
denominator = np.multiply((np.sqrt(np.sum((x-np.mean(x))**2))), np.sqrt(np.sum((
corr_coef_radio_newspaper = numerator / denominator

x=tv
y = newspaper
numerator = sum(np.multiply(x - np.mean(x), y - np.mean(y)))
denominator = np.multiply((np.sqrt(np.sum((x-np.mean(x))**2))), np.sqrt(np.sum((
corr_coef_tv_newspaper = numerator / denominator



fig, (ax1,ax2, ax3) = plt.subplots(1,3, figsize=(16,6))
ax1.set_title("Pearson Correlation Coefficient: {:.3f}".format(corr_coef_tv_radi
sns.scatterplot(x='TV', y='radio', data=df, ax=ax1)
ax2.set_title("Pearson Correlation Coefficient: {:.3f}".format(corr_coef_radio_r
sns.scatterplot(x='radio', y='newspaper', data=df, ax=ax2)
ax3.set_title("Pearson Correlation Coefficient: {:.3f}".format(corr_coef_tv_news
sns.scatterplot(x='TV', y='newspaper', data=df, ax=ax3);
```



c. Create X matrix and y Vector

Create the y Vector:

In [5]:

```
y = np.array(df['sales'])
y
```

Out[5]:

```
array([22.1, 10.4,  9.3, 18.5, 12.9,  7.2, 11.8, 13.2,  4.8, 10.6,  8.
6,
       17.4,  9.2,  9.7, 19. , 22.4, 12.5, 24.4, 11.3, 14.6, 18. , 12.
5,
        5.6, 15.5,  9.7, 12. , 15. , 15.9, 18.9, 10.5, 21.4, 11.9,  9.
6,
       17.4,  9.5, 12.8, 25.4, 14.7, 10.1, 21.5, 16.6, 17.1, 20.7, 12.
9,
        8.5, 14.9, 10.6, 23.2, 14.8,  9.7, 11.4, 10.7, 22.6, 21.2, 20.
2,
       23.7,  5.5, 13.2, 23.8, 18.4,  8.1, 24.2, 15.7, 14. , 18. ,  9.
3,
        9.5, 13.4, 18.9, 22.3, 18.3, 12.4,  8.8, 11. , 17. ,  8.7,  6.
9,
       14.2,  5.3, 11. , 11.8, 12.3, 11.3, 13.6, 21.7, 15.2, 12. , 16.
,
       12.9, 16.7, 11.2,  7.3, 19.4, 22.2, 11.5, 16.9, 11.7, 15.5, 25.
4,
       17.2, 11.7, 23.8, 14.8, 14.7, 20.7, 19.2,  7.2,  8.7,  5.3, 19.
8,
       13.4, 21.8, 14.1, 15.9, 14.6, 12.6, 12.2,  9.4, 15.9,  6.6, 15.
5,
        7. , 11.6, 15.2, 19.7, 10.6,  6.6,  8.8, 24.7,  9.7,  1.6, 12.
7,
        5.7, 19.6, 10.8, 11.6,  9.5, 20.8,  9.6, 20.7, 10.9, 19.2, 20.
1,
       10.4, 11.4, 10.3, 13.2, 25.4, 10.9, 10.1, 16.1, 11.6, 16.6, 19.
,
       15.6,  3.2, 15.3, 10.1,  7.3, 12.9, 14.4, 13.3, 14.9, 18. , 11.
9,
       11.9,  8. , 12.2, 17.1, 15. ,  8.4, 14.5,  7.6, 11.7, 11.5, 27.
,
       20.2, 11.7, 11.8, 12.6, 10.5, 12.2,  8.7, 26.2, 17.6, 22.6, 10.
3,
       17.3, 15.9,  6.7, 10.8,  9.9,  5.9, 19.6, 17.3,  7.6,  9.7, 12.
8,
       25.5, 13.4])
```

Create the X matrix:

In [6]:

```
features = np.array(df.drop("sales", axis=1))
features
```

Out[6]:

```
array([[230.1,  37.8,  69.2],
       [ 44.5,  39.3,  45.1],
       [ 17.2,  45.9,  69.3],
       [151.5,  41.3,  58.5],
       [180.8,  10.8,  58.4],
       [  8.7,  48.9,  75. ],
       [ 57.5,  32.8,  23.5],
       [120.2,  19.6,  11.6],
       [  8.6,   2.1,   1. ],
       [199.8,   2.6,  21.2],
       [ 66.1,   5.8,  24.2],
       [214.7,  24. ,   4. ],
       [ 23.8,  35.1,  65.9],
       [ 97.5,   7.6,   7.2],
       [204.1,  32.9,  46. ],
       [195.4,  47.7,  52.9],
       [ 67.8,  36.6, 114. ],
       [281.4,  39.6,  55.8],
```

In [7]:

```
# We need to create a matrix X, with a column vector of ones appended on the lei
X = np.insert(features, 0, 1, axis=1)
X
```

Out[7]:

```
array([[  1. , 230.1,  37.8,  69.2],
       [  1. ,  44.5,  39.3,  45.1],
       [  1. ,  17.2,  45.9,  69.3],
       [  1. , 151.5,  41.3,  58.5],
       [  1. , 180.8,  10.8,  58.4],
       [  1. ,   8.7,  48.9,  75. ],
       [  1. ,  57.5,  32.8,  23.5],
       [  1. , 120.2,  19.6,  11.6],
       [  1. ,   8.6,   2.1,   1. ],
       [  1. , 199.8,   2.6,  21.2],
       [  1. ,  66.1,   5.8,  24.2],
       [  1. , 214.7,  24. ,   4. ],
       [  1. ,  23.8,  35.1,  65.9],
       [  1. ,  97.5,   7.6,   7.2],
       [  1. , 204.1,  32.9,  46. ],
       [  1. , 195.4,  47.7,  52.9],
       [  1. ,  67.8,  36.6, 114. ],
       [  1. , 281.4,  39.6,  55.8],
```

In [8]:

```
# We need to create a matrix X, with a column vector of ones appended on the lei
rows = df['radio'].shape[0]
ones = np.ones([rows, 1]) # specify number of rows and columns as a list
X = np.hstack((ones, features))
X
```

Out[8]:

```
array([[  1. , 230.1,  37.8,  69.2],
       [  1. ,  44.5,  39.3,  45.1],
       [  1. ,  17.2,  45.9,  69.3],
       [  1. , 151.5,  41.3,  58.5],
       [  1. , 180.8,  10.8,  58.4],
       [  1. ,   8.7,  48.9,  75. ],
       [  1. ,  57.5,  32.8,  23.5],
       [  1. , 120.2,  19.6,  11.6],
       [  1. ,   8.6,   2.1,   1. ],
       [  1. , 199.8,   2.6,  21.2],
       [  1. ,  66.1,   5.8,  24.2],
       [  1. , 214.7,  24. ,   4. ],
       [  1. ,  23.8,  35.1,  65.9],
       [  1. ,  97.5,   7.6,   7.2],
       [  1. , 204.1,  32.9,  46. ],
       [  1. , 195.4,  47.7,  52.9],
       [  1. ,  67.8,  36.6, 114. ],
       [  1. , 281.4,  39.6,  55.8],
```

d. Calculate the Regression Coefficients

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

In [9]:

```
X.T@X   # np.dot(X.T, X)
```

Out[9]:

```
array([[2.00000000e+02, 2.94085000e+04, 4.65280000e+03, 6.11080000e+0
3],
       [2.94085000e+04, 5.79111839e+06, 6.98061980e+05, 9.19625280e+0
5],
       [4.65280000e+03, 6.98061980e+05, 1.52107860e+05, 1.64946550e+0
5],
       [6.11080000e+03, 9.19625280e+05, 1.64946550e+05, 2.81096740e+0
5]])
```

In [10]:

```
1 np.linalg.inv(X.T@X)
```

Out[10]:

```
array([[ 3.42444998e-02, -9.35348333e-05, -3.92647297e-04,
        -2.08036831e-04],
       [-9.35348333e-05,  6.84890750e-07, -1.57355890e-07,
        -1.14959987e-07],
       [-3.92647297e-04, -1.57355890e-07,  2.61016474e-05,
        -6.26574016e-06],
       [-2.08036831e-04, -1.14959987e-07, -6.26574016e-06,
         1.21328472e-05]])
```

In [11]:

```
1 X.T@y
```

Out[11]:

```
array([  2804.5 , 482108.34,  74126.39,  90851.03])
```

In [12]:

```
1 betas = np.linalg.inv(X.T@X) @ X.T@y
2 print("Beta0: {:.3f}, Beta1: {:.3f}, Beta2: {:.3f}, Beta3: {:.3f}".format(betas[
```

```
Beta0: 2.939, Beta1: 0.046, Beta2: 0.189, Beta3: -0.001
```

e. Carry out the Prediction

Predict the expected sales for the advertisement budget of [180,10,58]

$$\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x_1 + \hat{\beta}_2 x_2 + \hat{\beta}_3 x_3$$

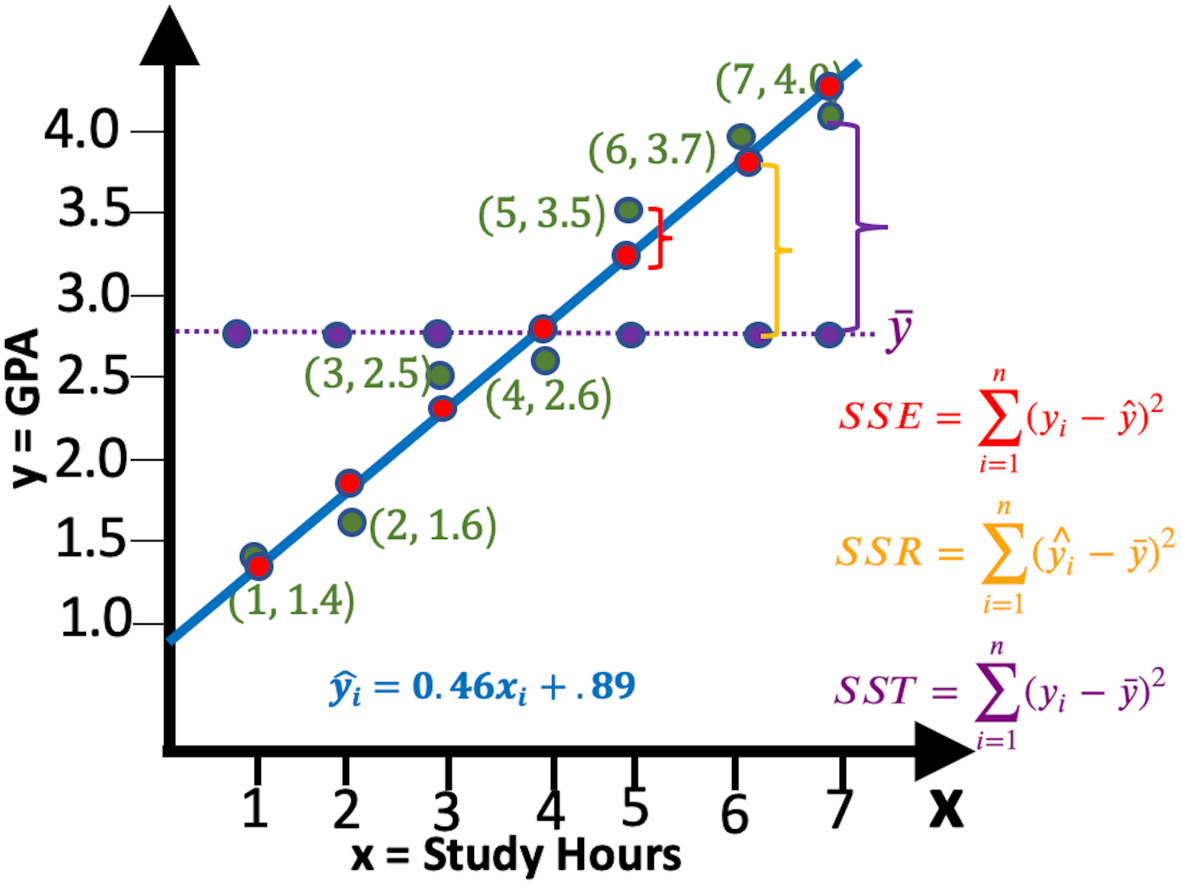
$$\hat{y} = 2.939 + 0.046 * \text{tv} + 0.189 * \text{radio} + (-0.001) * \text{newspaper}$$

In [13]:

```
1 tv=180
2 radio=10
3 newspaper=58
4 pred_y = betas[0]   +    betas[1] * tv   +      betas[2] * radio +      betas[3]
5 print("Predicted Sales value for input features: {:.3f}".format(pred_y))
```

```
Predicted Sales value for input features: 13.002
```

7. Evaluate the Model



(i) Mean Absolute Error (MAE):

- Absolute means we do not consider the sign, i.e., $d_i = |y_i - \hat{y}_i|$

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}|$$

- The **advantages** of MAE are that the unit is same and it works well in case of outliers as well.
- The **disadvantage** is that the modulus function is not differentiable.

(ii) Mean Squared Error (MSE):

- To overcome the limitation of MAE, i.e., to make the cost function differentiable
- In Mean Squared Error (MSE), instead of taking the modulus, we take the mean of all the residual errors and take their average:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y})^2$$

- The **advantages** of MSE is that the cost function is differentiable.
- The **limitation** is that the unit are different. For example, if the unit of output variable (y) is Rs, then the MSE unit is RS^2. So if MSE of a point is 10 that means the actual error is square of 10.
- The other limitation is that it doesn't work well in case of outliers, because in case of a outlier the error is squared and is a big value. So the loss function is effected much by outliers. So if there are lot of outliers in yor dataset, you should go with MAE instead of MSE.

(iii) Root Mean Squared Error (RMSE):

Using above Formulae:

In [39]:

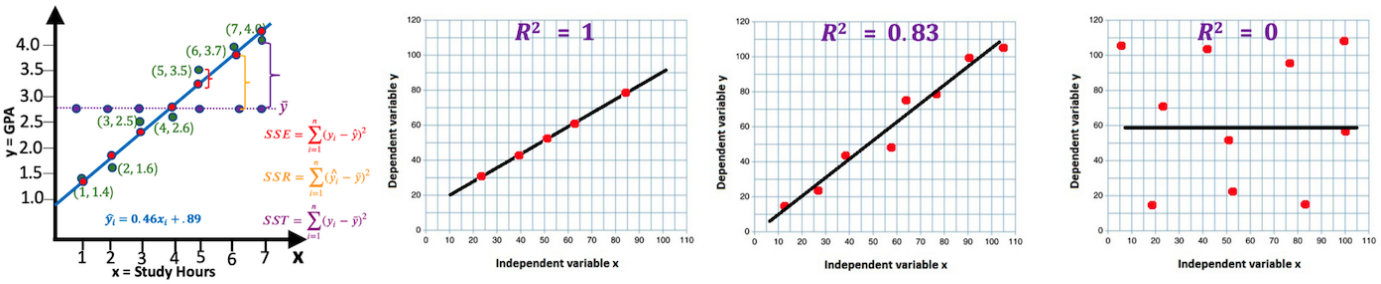
```
ybar = np.mean(y)
yhat = betas[0] + betas[1]*df['TV'] + betas[2]*df['radio'] +  betas[3] * df['new
mae = (np.sum(abs(y - yhat)))/len(y)
mse = (np.sum((y - yhat)**2))/len(y)
rmse = np.sqrt((np.sum((y - yhat)**2))/len(y))

print("MAE: {:.3f}".format(mae))
print("MSE: {:.3f}".format(mse))
print("RMSE: {:.3f}".format(rmse))
```

```
MAE: 1.252
MSE: 2.784
RMSE: 1.669
```

In [ ]:

```

```

(iv) R^2 Score:



$$R^2 = 1 - \frac{SSE}{SST}$$

- Where,

$$SSE = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$
$$SSR = \sum_{i=1}^{n}(\hat{y}_i - \bar{y})^2$$
$$SST = \sum_{i=1}^{n}(y_i - \bar{y})^2$$

Using above Formulae:

In [41]:

```
ybar = np.mean(y)
sse = np.sum((y - yhat)**2)
sst = np.sum((y - ybar)**2)
r2 = 1 - (sse/sst)
print("R2 Score: {:.4f}".format(r2))
```

```
R2 Score: 0.8972
```

(v) Adjusted R^2 Score:**

- The limitation or R2 score is that, if you add more features to your model, the R2 score will either increase or stay the same, but will never decrease.
- Adjusted R2 score is a modified form of R2 score, whose value increases if new predictors tend to improve model's performance and decreases if new predictors do not improve performance as expected.

$$R_a^2 = 1 - [\frac{(1-R^2)(n-1)}{n-m-1}]$$

$$R_a^2 = 1 - (\frac{n-1}{SST})MSE$$

$$R_a^2 = R^2 - \frac{m(1-R^2)}{n-m-1}$$

> - where n is number of observations in sample and m is number of features/predictors/independent variables in model
> - All the three formulas give you the same result, actually if your reorganize them algebbraically you can end up with the same formula

In [43]:

```
n = len(X)
m = 3
r2_adj = 1 - ((1-r2)*(n-1)/(n-m-1))
print("R2 Score: {:.4f}".format(r2_adj))
```

```
R2 Score: 0.8956
```

Assumptions for Linear Regression

a. Linearity

b. Independence

c. Homoscedasticity

d. Normality

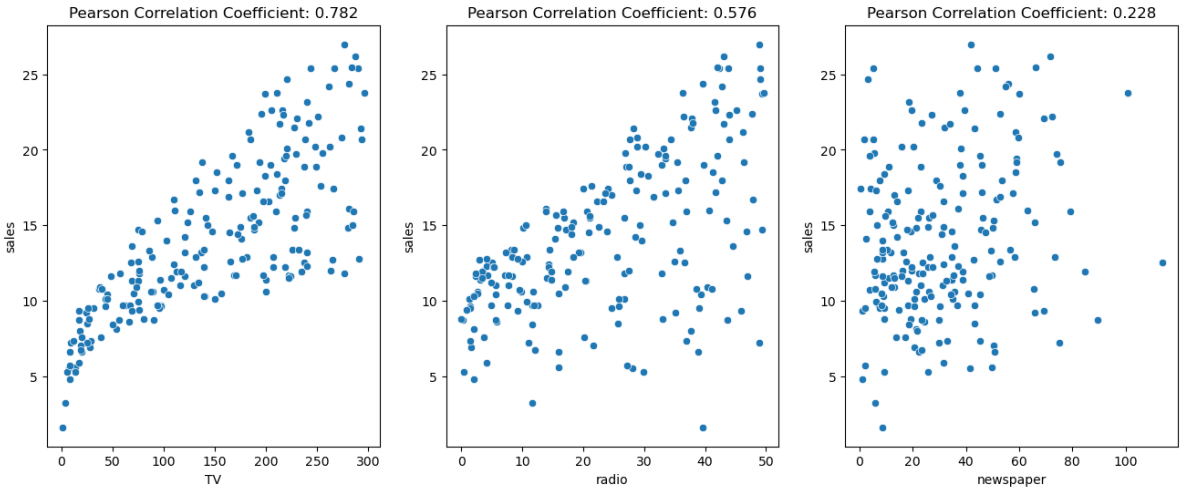
> - What do you mean by this assumption?
> - How to determine if this assumption is met?
> - What to do if this assumption is violated?

a. Linearity:

- All the feature variables should be somehow correlated with the output variable.
- Can be checked using scatterchart and Pearson Correlation Coefficient.
- Voilation of this assumption will cause overfitting
- `What to do if this assumption is violated?` If you create a scatter plot of values for x and y and see that there is not a linear relationship between the two variables, then you have a couple options:
  - Apply a nonlinear transformation to the independent and/or dependent variable. Common examples include taking the log, the square root, or the reciprocal of the independent and/or dependent variable.
  - Add another independent variable to the model. For example, if the plot of x vs. y has a parabolic shape then it might make sense to add X2 as an additional independent variable in the model.

In [44]:

```
%matplotlib inline
tv = np.array(df['TV'])
radio = np.array(df['radio'])
newspaper = np.array(df['newspaper'])

x=tv
y = np.array(df['sales'])
numerator = sum(np.multiply(x - np.mean(x), y - np.mean(y)))
denominator = np.multiply((np.sqrt(np.sum((x-np.mean(x))**2))), np.sqrt(np.sum((
corr_coef_tv = numerator / denominator

x=radio
y = np.array(df['sales'])
numerator = sum(np.multiply(x - np.mean(x), y - np.mean(y)))
denominator = np.multiply((np.sqrt(np.sum((x-np.mean(x))**2))), np.sqrt(np.sum((
corr_coef_radio = numerator / denominator

x=newspaper
y = np.array(df['sales'])
numerator = sum(np.multiply(x - np.mean(x), y - np.mean(y)))
denominator = np.multiply((np.sqrt(np.sum((x-np.mean(x))**2))), np.sqrt(np.sum((
corr_coef_newspaper = numerator / denominator


fig, (ax1,ax2, ax3) = plt.subplots(1,3, figsize=(16,6))
ax1.set_title("Pearson Correlation Coefficient: {:.3f}".format(corr_coef_tv))
sns.scatterplot(x='TV', y='sales', data=df, ax=ax1)
ax2.set_title("Pearson Correlation Coefficient: {:.3f}".format(corr_coef_radio))
sns.scatterplot(x='radio', y='sales', data=df, ax=ax2)
ax3.set_title("Pearson Correlation Coefficient: {:.3f}".format(corr_coef_newspap
sns.scatterplot(x='newspaper', y='sales', data=df, ax=ax3);
```



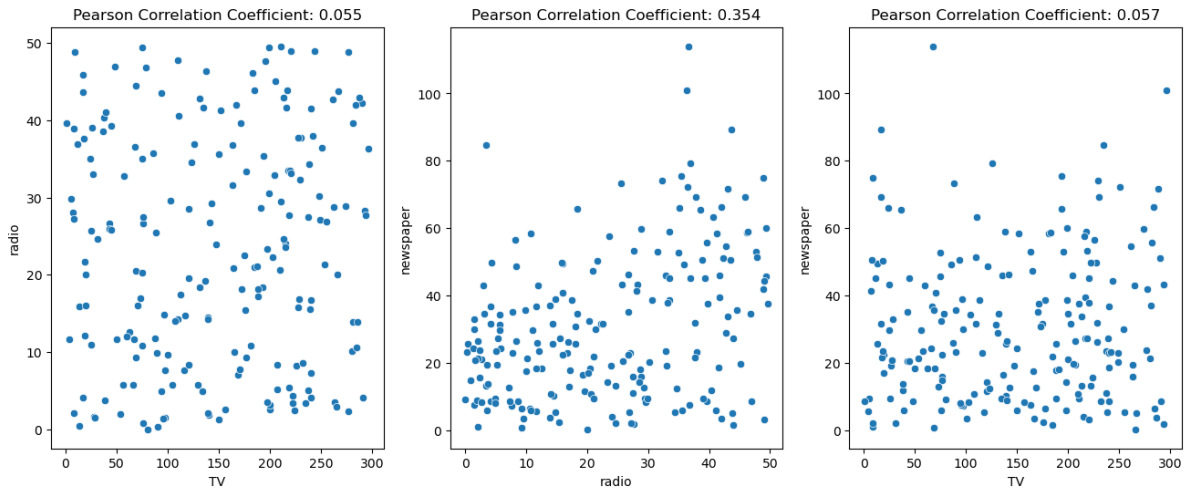
b. Independence:

- All the feature variables should NOT have any correlation with eachother (should be independent).
- This can be checked using scatterchart.
- Violation of this assumption will cause multicollinearity, due to which regression coefficients will become unstable and difficult to interpret
- `What to do if this assumption is violated?`
  - For positive serial correlation, consider adding lags of the dependent and/or independent variable to the model.

- For negative serial correlation, check to make sure that none of your variables are overdifferenced.
- For seasonal correlation, consider adding seasonal dummy variables to the model.

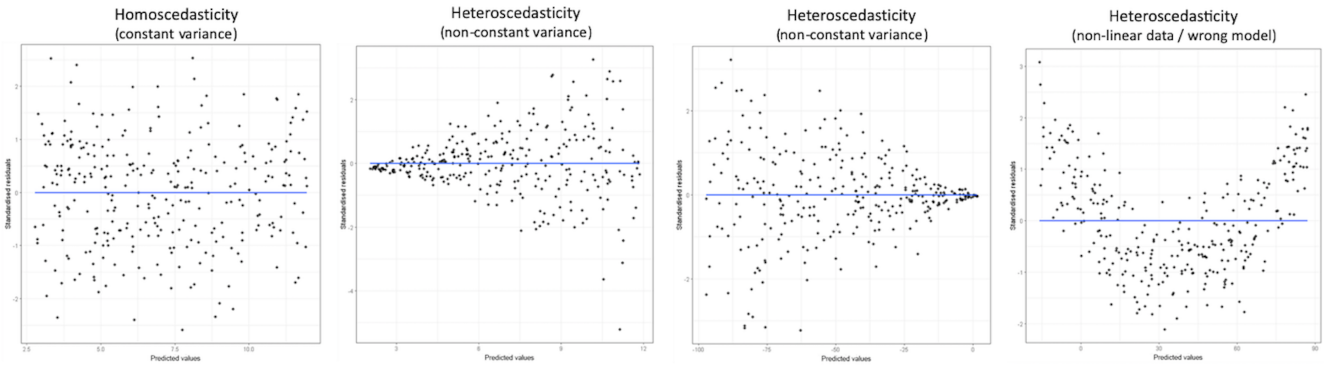
In [45]:

```
tv = np.array(df['TV'])
radio = np.array(df['radio'])
newspaper = np.array(df['newspaper'])

x=tv
y = radio
numerator = sum(np.multiply(x - np.mean(x), y - np.mean(y)))
denominator = np.multiply((np.sqrt(np.sum((x-np.mean(x))**2))), np.sqrt(np.sum((
corr_coef_tv_radio = numerator / denominator

x=radio
y = newspaper
numerator = sum(np.multiply(x - np.mean(x), y - np.mean(y)))
denominator = np.multiply((np.sqrt(np.sum((x-np.mean(x))**2))), np.sqrt(np.sum((
corr_coef_radio_newspaper = numerator / denominator

x=tv
y = newspaper
numerator = sum(np.multiply(x - np.mean(x), y - np.mean(y)))
denominator = np.multiply((np.sqrt(np.sum((x-np.mean(x))**2))), np.sqrt(np.sum((
corr_coef_tv_newspaper = numerator / denominator



fig, (ax1,ax2, ax3) = plt.subplots(1,3, figsize=(16,6))
ax1.set_title("Pearson Correlation Coefficient: {:.3f}".format(corr_coef_tv_radi
sns.scatterplot(x='TV', y='radio', data=df, ax=ax1)
ax2.set_title("Pearson Correlation Coefficient: {:.3f}".format(corr_coef_radio_r
sns.scatterplot(x='radio', y='newspaper', data=df, ax=ax2)
ax3.set_title("Pearson Correlation Coefficient: {:.3f}".format(corr_coef_tv_news
sns.scatterplot(x='TV', y='newspaper', data=df, ax=ax3);
```



c. Homoscedasticity:

- It is a condition where the variance of the residual errors in a regression model is constant.
- This can be checked using residual plot, that shows the residual errors on the vertical axis and the independent variable on the horizontal axis. If the residuals do not fan out in a triangular fashion that means that the homoscedasticity (equal variance) assumption is met
- Violation of this assumption makes your coefficients less accurate



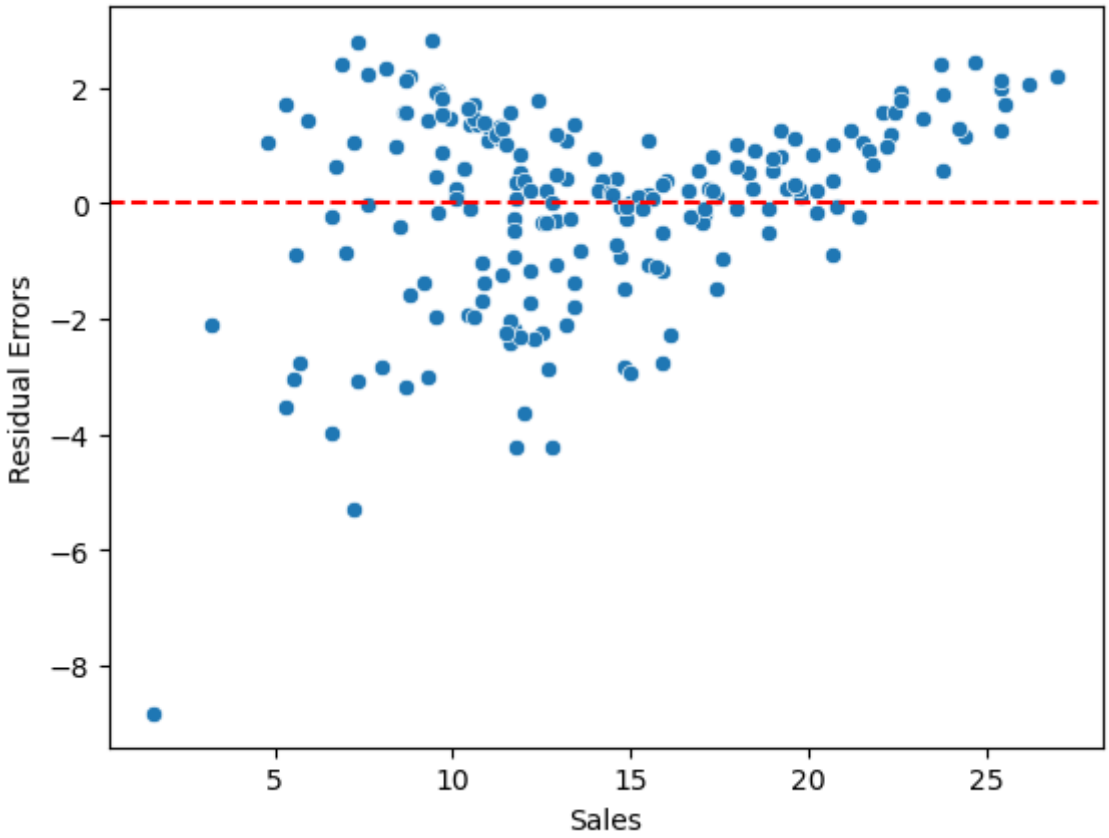
- `What to do if this assumption is violated?`
  - Transform the dependent variable. One common transformation is to simply take the log of the dependent variable. For example, if we are using population size (independent variable) to predict the number of flower shops in a city (dependent variable), we may instead try to use population size to predict the log of the number of flower shops in a city. Using the log of the dependent variable, rather than the original dependent variable, often causes heteroskedasticity to go away.
  - Redefine the dependent variable. One common way to redefine the dependent variable is to use a rate, rather than the raw value. For example, instead of using the population size to predict the number of flower shops in a city, we may instead use population size to predict the number of flower shops per capita. In most cases, this reduces the variability that naturally occurs among larger populations since we're measuring the number of flower shops per person, rather than the sheer amount of flower shops.
  - Use weighted regression. Another way to fix heteroscedasticity is to use weighted regression. This type of regression assigns a weight to each data point based on the variance of its fitted value. Essentially, this gives small weights to data points that have higher variances, which shrinks their squared residuals. When the proper weights are used, this can eliminate the problem of heteroscedasticity.

In [46]:

```
df = pd.read_csv("datasets/advertising4D.csv")
X = df.drop('sales', axis=1)
y = np.array(df['sales'])
model = LinearRegression()
model.fit(X, y)

yhat = model.predict(X)
residual_errors = y - yhat
sns.scatterplot(x=y, y=residual_errors)
plt.axhline(y=0, color='r', ls='--')

plt.xlabel('Sales')
plt.ylabel("Residual Errors")
```

Out[46]:

```
Text(0, 0.5, 'Residual Errors')
```



d. Normality:

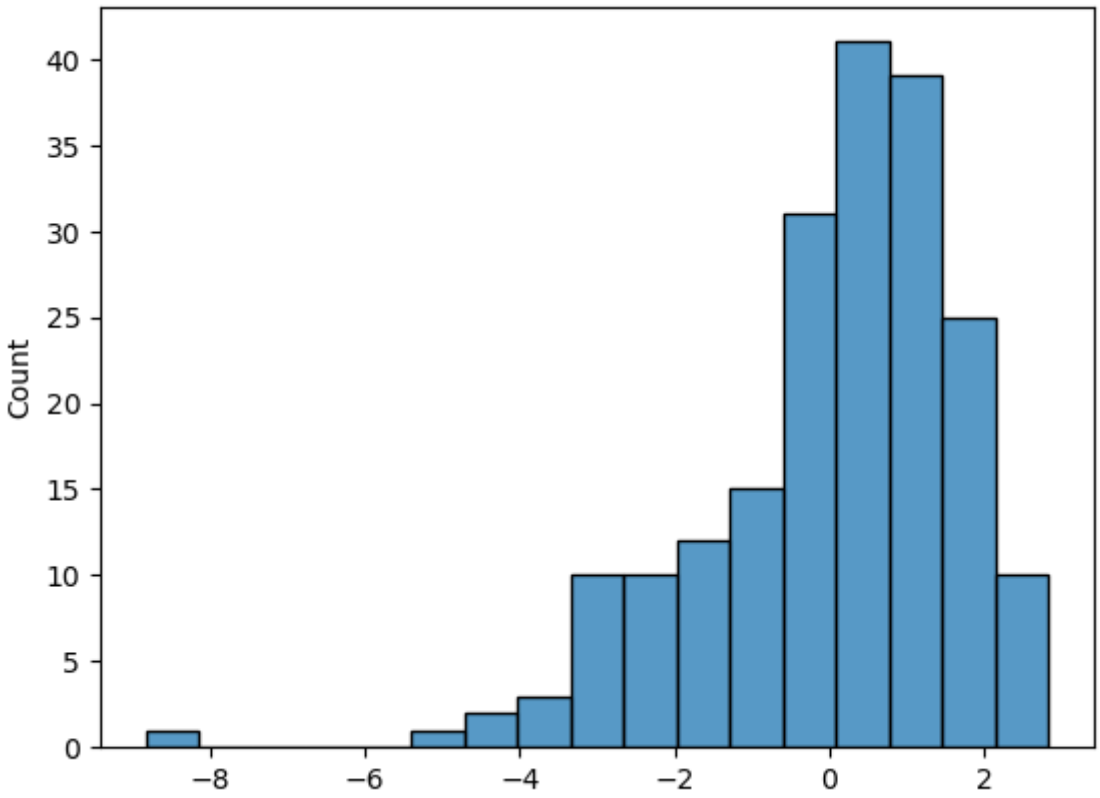
- The residuals (errors) of the regression line are approximately normally distributed.
- This can be checked by drawing the histogram of the residuals. If it is not skewed, that means the assumption is satisfied. OR, you can use Q-Q plot, short for quantile-quantile plot. If the points on the plot roughly form a straight diagonal line, then the normality assumption is met.
- What to do if this assumption is violated
  - First, verify that any outliers aren't having a huge impact on the distribution. If there are outliers present, make sure that they are real values and that they aren't data entry errors.
  - Next, you can apply a nonlinear transformation to the independent and/or dependent variable. Common examples include taking the log, the square root, or the reciprocal of the independent and/or dependent variable.

In [47]:

```
1 sns.histplot(data=residual_errors)
```

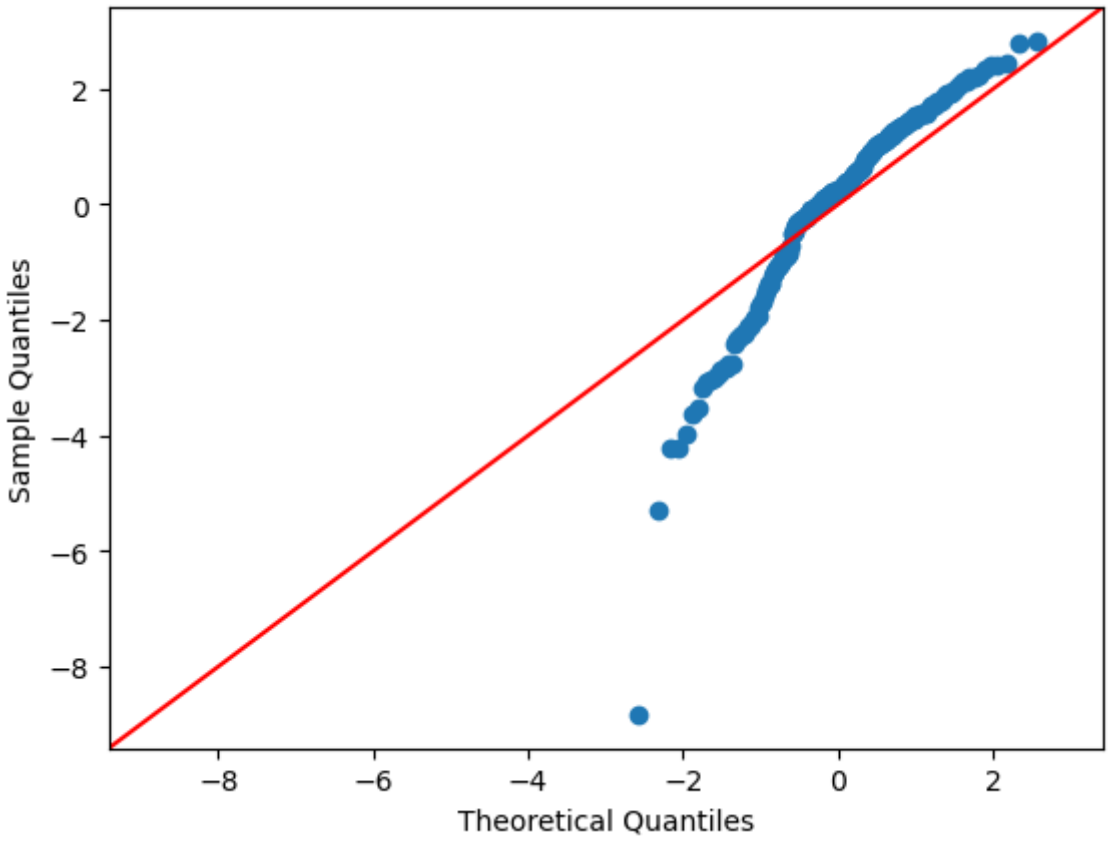
Out[47]:

```
<AxesSubplot:ylabel='Count'>
```



In [48]:

```
import statsmodels.api as sm
import pylab as py

sm.qqplot(residual_errors, line ='45')
py.show()
```



6. Task To Do

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib
from matplotlib import pyplot as plt
import seaborn as sns
df = pd.read_csv("datasets/insurance.csv")
df
```

Out[1]:

	age	sex	bmi	children	smoker	region	charges
0	19	female	27.900	0	yes	southwest	16884.92400
1	18	male	33.770	1	no	southeast	1725.55230
2	28	male	33.000	3	no	southeast	4449.46200
3	33	male	22.705	0	no	northwest	21984.47061
4	32	male	28.880	0	no	northwest	3866.85520
...	...	...	...	...	...	...	...
1333	50	male	30.970	3	no	northwest	10600.54830
1334	18	female	31.920	0	no	northeast	2205.98080
1335	18	female	36.850	0	no	southeast	1629.83350
1336	21	female	25.800	0	no	southwest	2007.94500
1337	61	female	29.070	0	yes	northwest	29141.36030

1338 rows × 7 columns

In [ ]:

```

```