Department of Data Science

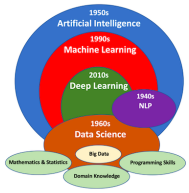
Course: Tools and Techniques for Data Science

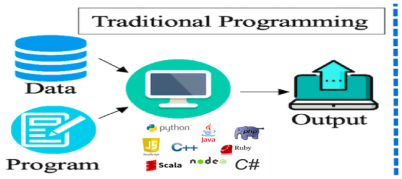
Instructor: Muhammad Arif Butt, Ph.D.

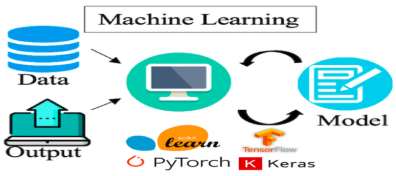
Lecture 6.6 (Linear Regression using Scikit-Learn Library)

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)



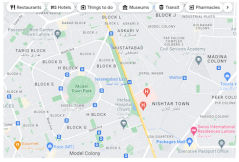




ML is the application of AI that gives machines the ability to learn without being explicitly programmed

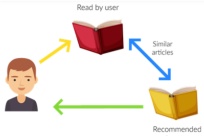
Face Detection

Forecasting



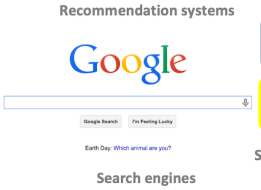
Navigation

Classification



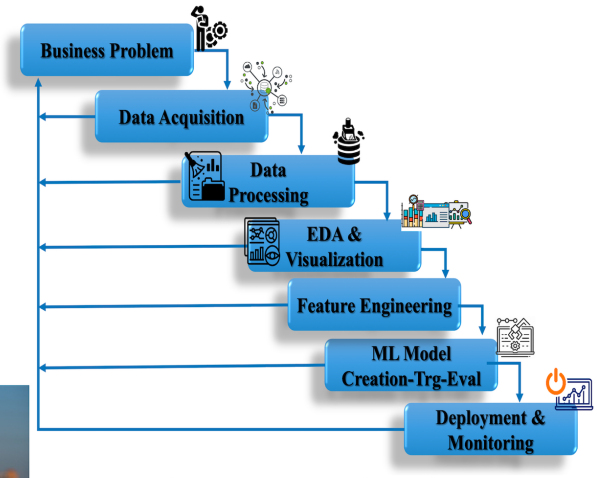
Recommendation systems

Predictions



Search engines

Social Apps



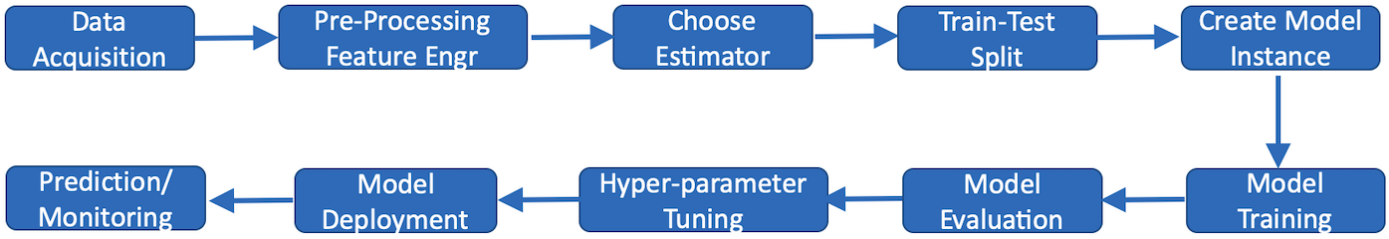
Machine Learning Life Cycle

Learning agenda of this notebook

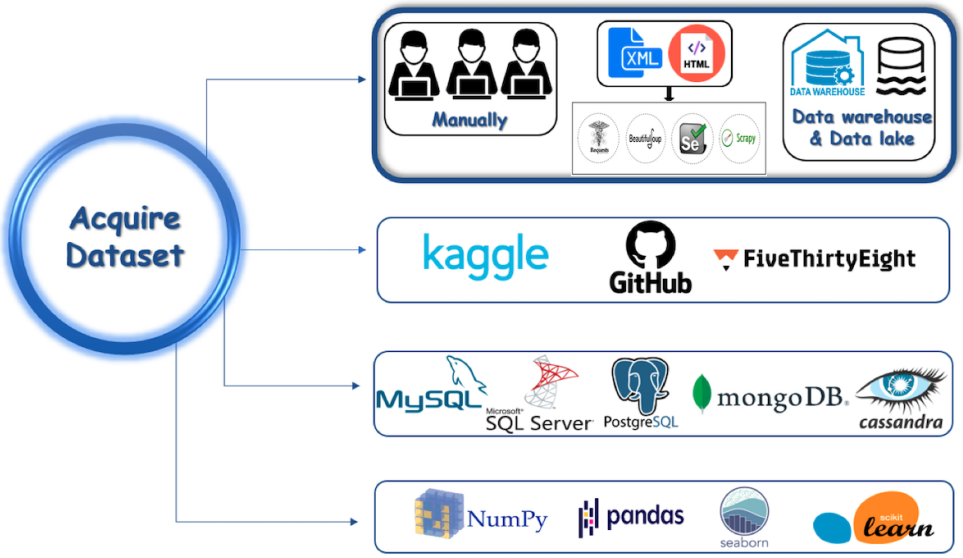
1. Scikit-Learn WorkFlow (Supervised ML)
   I. Data Acquisition
   II. Data Pre-Processing and Feature Engineering
   III. Choose an Appropriate Supervised Machine Learning Model

IV. Train-Test-Split
V. Create an Instance of Machine Learning Model/Estimator using Scikit-Learn
VI. Train the Model
VII. Carry out Prediction
VIII. Evaluate the Model
IX. Improve via Hyper-Parameter Tuning
X. Model Deployment

1. Scikit-Learn WorkFlow (Supervised ML)



I. Data Acquisition



- **Use Libraries Built-in Datasets:**
  - Seaborn: (iris, titanic, tips, flights, panguins, car_crashes)
  - Scikit-learn: (iris, digits, diabetes, Bostan housing)
  - NLTK: (movie-reviews, product_reviews, twitter_samples, gutenerg, genesis, timeit, voice, wordnet, sentiword)
- **Use Public Dataset Repositories:**
  - https://www.kaggle.com/ (https://www.kaggle.com/)
  - https://data.gov/ (https://data.gov/)
  - https://archive.ics.uci.edu/ml/index.php (https://archive.ics.uci.edu/ml/index.php)
  - https://github.com/ (https://github.com/)
- **Use Company's Database:** (SQL, NOSQL, Data warehouse, Data lake)
- **Generate your own Datasets:**
  - Use Web scraping or Web API
- IoT Devices
- Crowd Sourcing (Amazon Mechanical Turk, Lionbridge AI)
- Data Augmentation

Techniques to Label your Datasets:

> **Better data often beats better algorithms**

A Sample Advertising Dataset

In [1]:

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

Out[1]:

	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
199	232.1	8.6	8.7	13.4

200 rows × 4 columns

- How strong is the relationship between input features and output sales?
- Which ad types contribute more to sales?
- What is the effect of each ad type on sales?
- Given an advertising budget, can we predict future sales?

II. Data Pre-Processing and Feature Engineering

- `Data Preprocessing` involves actions that we need to perform on the dataset in order to make it ready to be fed to the machine learning model.
- `Feature Engineering` is the process of using domain knowledge to extract features from raw data via data mining techniques.

City	Size	Covered Area	No of bedrooms	Trees near by	No of bathrooms	Schools near by	Construction Date	Price
Lahore	2000	3500	3	1	3	1	25/10/2001	20.5 M
Karachi	2600	3000	2	0	4	1	16/05/1990	18 M
Islamabad	1800	2000	3	1	3	2	25/11/1995	20 M
Shaikhupura	1600	2600	1	2	NaN	0	08/06/2020	5 M
Lahore	2600	2000	3	3	1	1	03/09/2016	4 M
Karachi	3000	1000	2	2	1	NaN	19/01/1980	6 M
Islamabad	2000	3600	44	4	3	3	21/07/1999	30 M
Lahore	1000	2000	3	NaN	1	2	12/04/2015	10 M

- Pre-processing package of sklearn provides a bundle of utility functions and transformer classes for data preprocessing (will cover later).
  - **Detecting and handling outliers**
    - Univariate (Z-Score, IQR, Percentiles)
    - Multivariate Analysis (Depth-based, Distance-based, Density-based methods)
    - Trimming, Capping/Winsorization, Discritization
  - **Missing values Imputation**
    - Univariate Imputation (Panda's `fillna()` method, Sklearn's `SimpleImputer()` transformer)
    - Multivariate Imputation (Sklearn's `IterativeImputer()` and `KNNImputer()` transformers)
  - **Encoding Categorical Features**
    - Encode Nominal i/p features using Pandas `get_dummies()` and Scikit-Learn's `OneHotEncoder()`
    - Encode Ordinal i/p features using Scikit-Learn's `OrdinalEncoder()`
    - Encode categorical o/p column(s) using Scikit-Learn's `LabelEncoder()`
  - **Feature Scaling**
    - Use numPy to perform maxabs, minmax, standard and robust scaling
    - Use Sklearn's `MaxAbsScalar` , `MinMaxScalar` , `StandardScalar` , `RobustScalar` transformers
  - **Extracting Information**
    - Use Sklearn's `CountVectorizer` , `DictVectorizer` , `TfidfVectorizer` , and `TfidfTransformer`
  - **Combining Information**
    - Use `FeatureUnion` , `Pipeline` , `PCA`

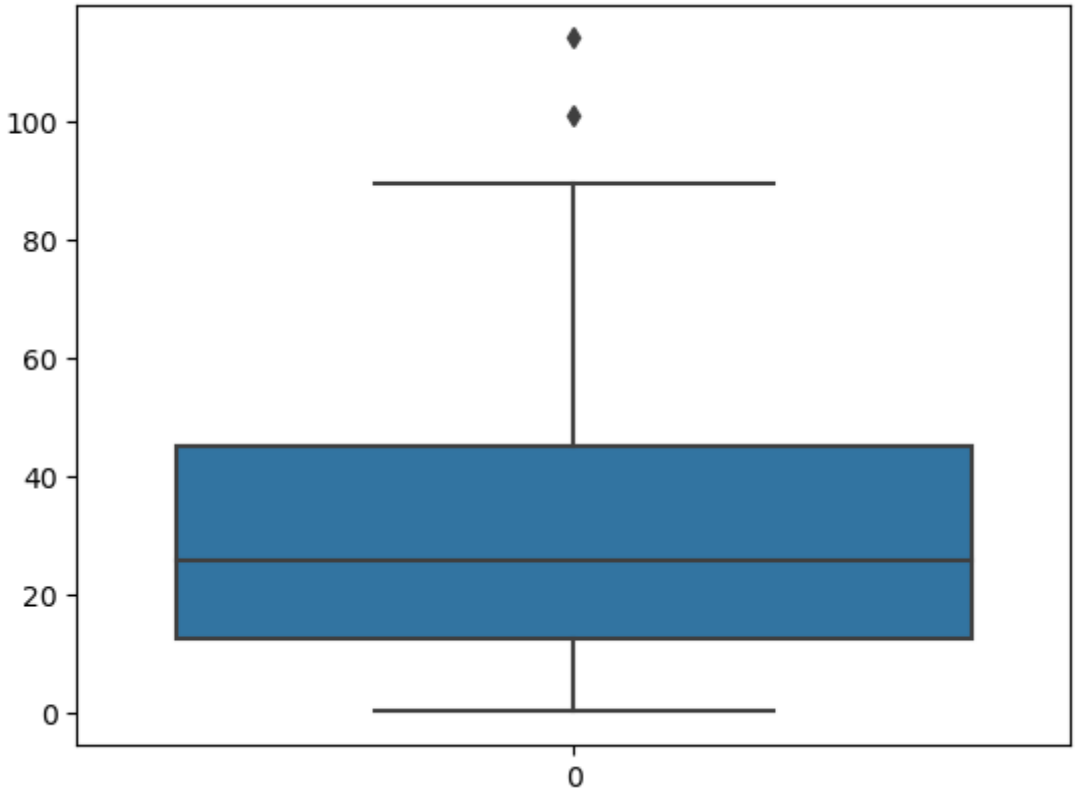
In [2]:

```
1 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
```

In [3]:

```
1 sns.boxplot(data=df['newspaper']);
```

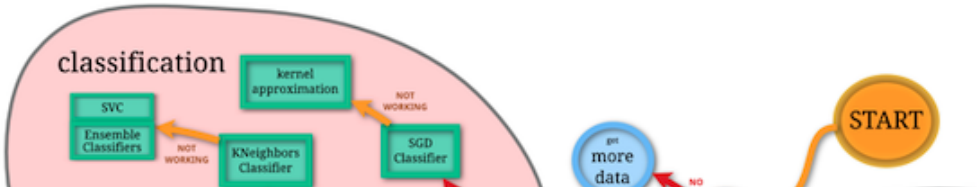


III. Choose an Appropriate Machine Learning Model (Estimator)

- There are many models that researchers and datascientists have created over the years.
- Some are very well suited for numerical data, some are good for textual data, while some are good for image data.

> **The right model for your data depends on the type of the data you have and what you want to do with it**

- This flowchart is designed to give users a rough guide to choose an estimator to try on your data.



Assumptions for Linear Regression

a. Linearity

b. Independence

c. Homoscedasticity

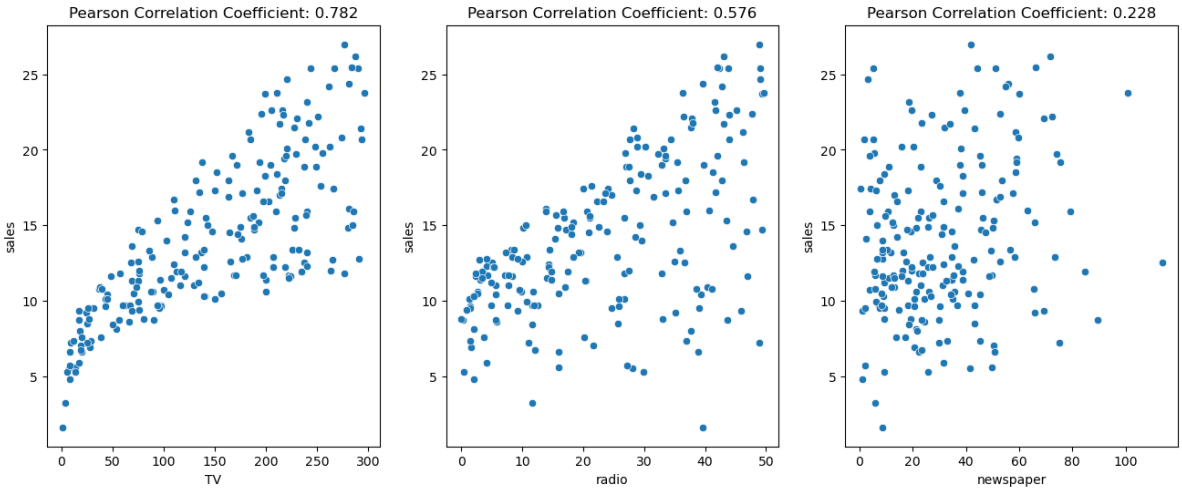
d. Normality

a. Linearity:

- What do you mean by this assumption?
  - All the feature variables should be somehow correlated with the output variable.
- How to determine if this assumption is met?
  - Can be checked using scatterplot and Pearson Correlation Coefficient among the feature/independent and the output/dependent variables.
- Problem Caused?
  - Voilation of this assumption will cause overfitting.
- What to do if this assumption is violated?
  - Add ploynomial features (i.e., add quadratic terms and interaction terms).
  - Apply a nonlinear transformation to the independent and/or dependent variable. For example, take the log/squareroot/reciprocal of the independent and/or dependent variable.

In [4]:

```
%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:

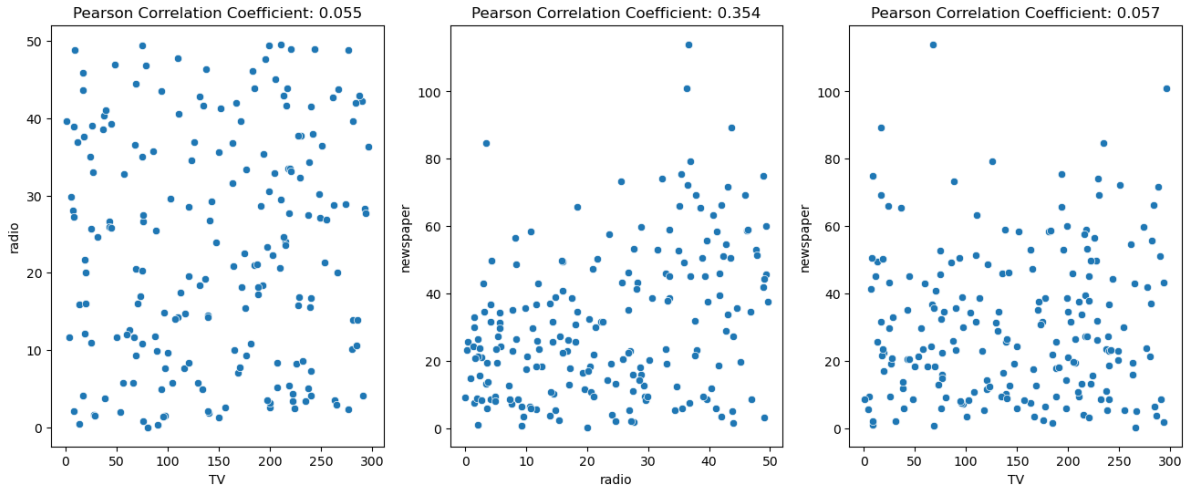
- What do you mean by this assumption?
  - All the feature variables should NOT have any correlation with eachother (should be independent).
- How to determine if this assumption is met?
  - Can be checked using scatterplot and Pearson Correlation Coefficient among the feature/independent variables.
- Problem Caused?

- 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.

In [5]:

```
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);
```

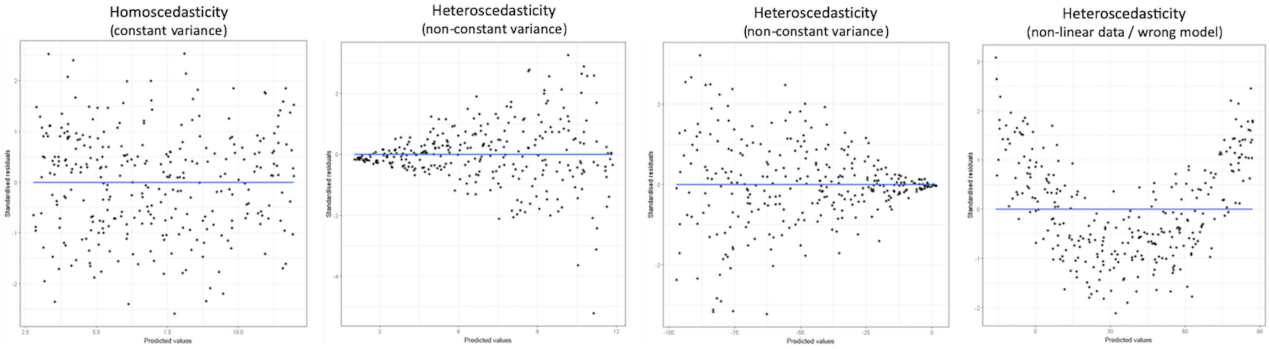


c. Homoscedasticity:

- What do you mean by this assumption?

- It is a condition where the variance of the residual errors in a regression model is constant.

- How to determine if this assumption is met?
  - 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



- Problem Caused?
  - Violation of this assumption makes your coefficients less accurate

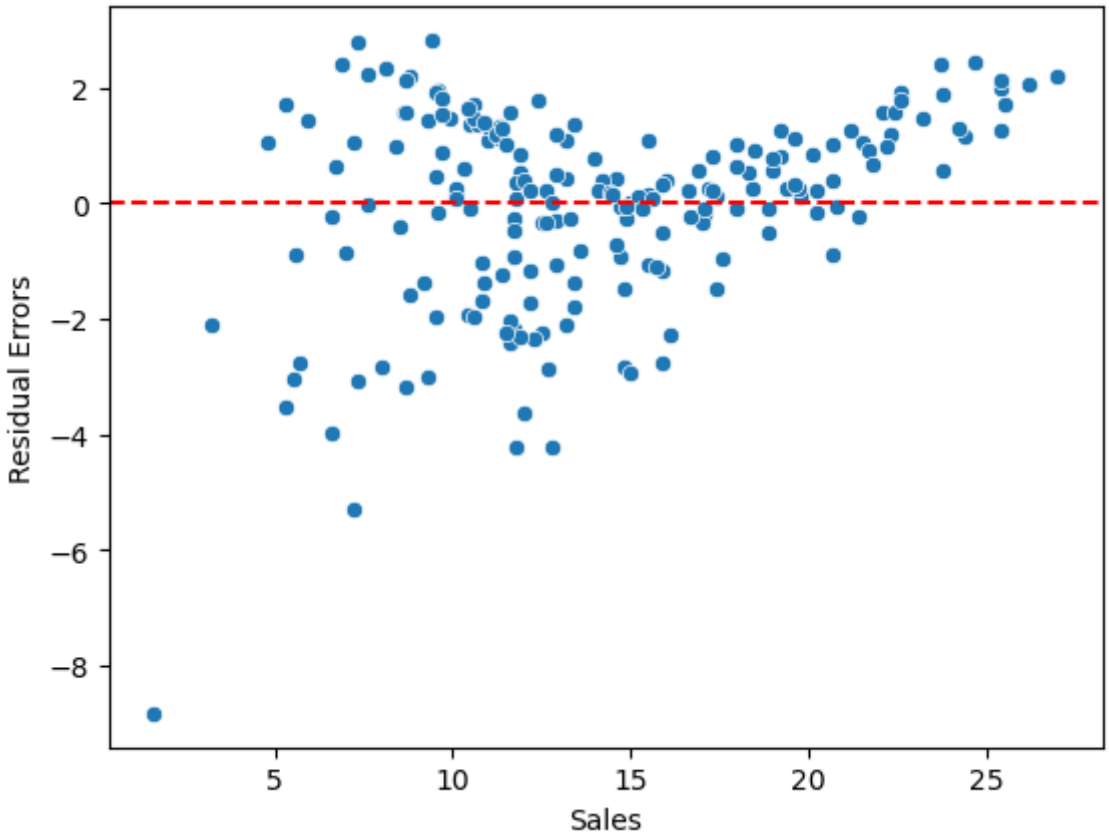
- What to do if this assumption is violated?
  - Transform the dependent variable. One common transformation is to simply take the log of the dependent variable.
  - Use weighted regression by assigning a weight to each data point based on the variance of its fitted value.

In [6]:

```
from sklearn.linear_model import LinearRegression
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[6]:

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



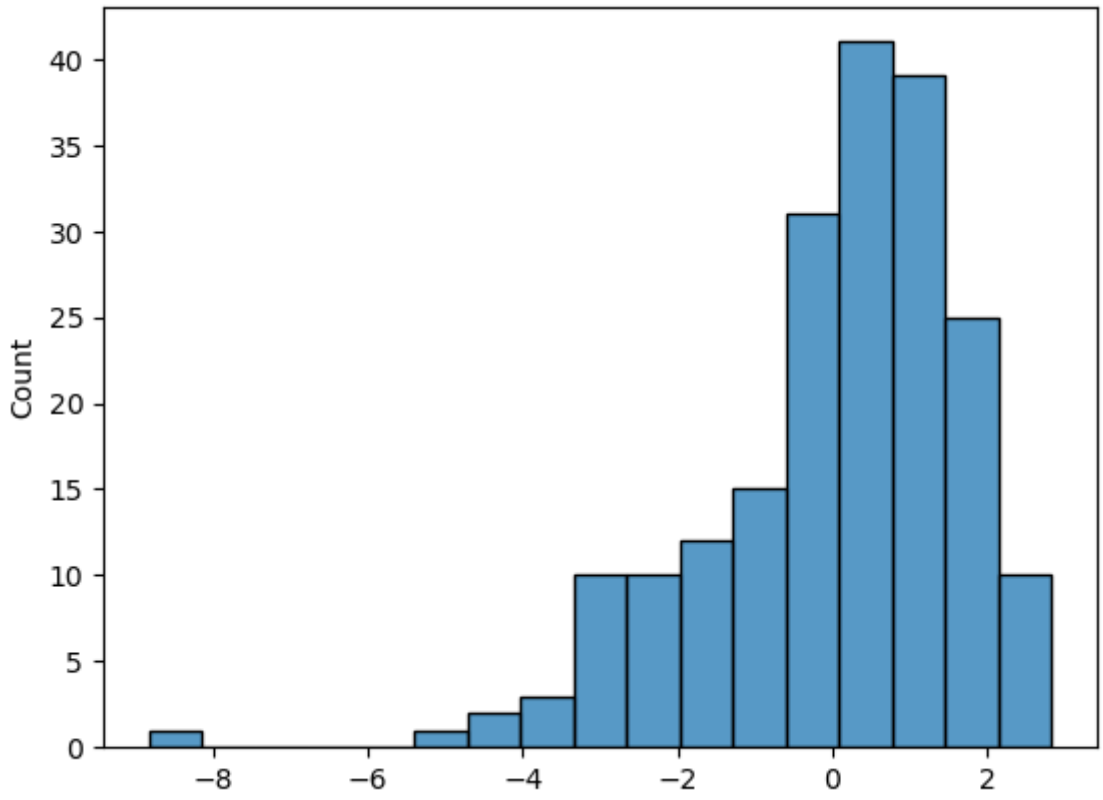
d. Normality:

- What do you mean by this assumption?
  - The residuals (errors) of the regression line are approximately normally distributed.
- How to determine if this assumption is met?
  - 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.
- Problem Caused?
  - Incorrect Regression coefficients.
- 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.

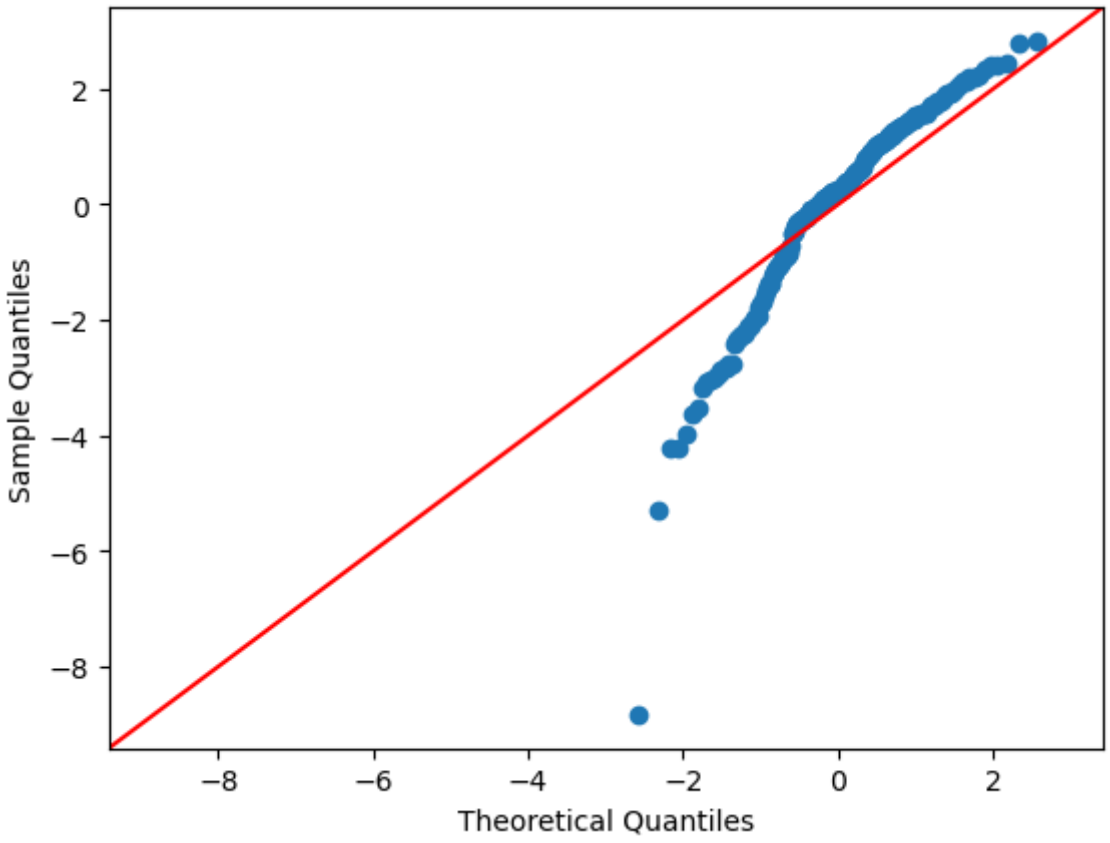
In [7]:

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



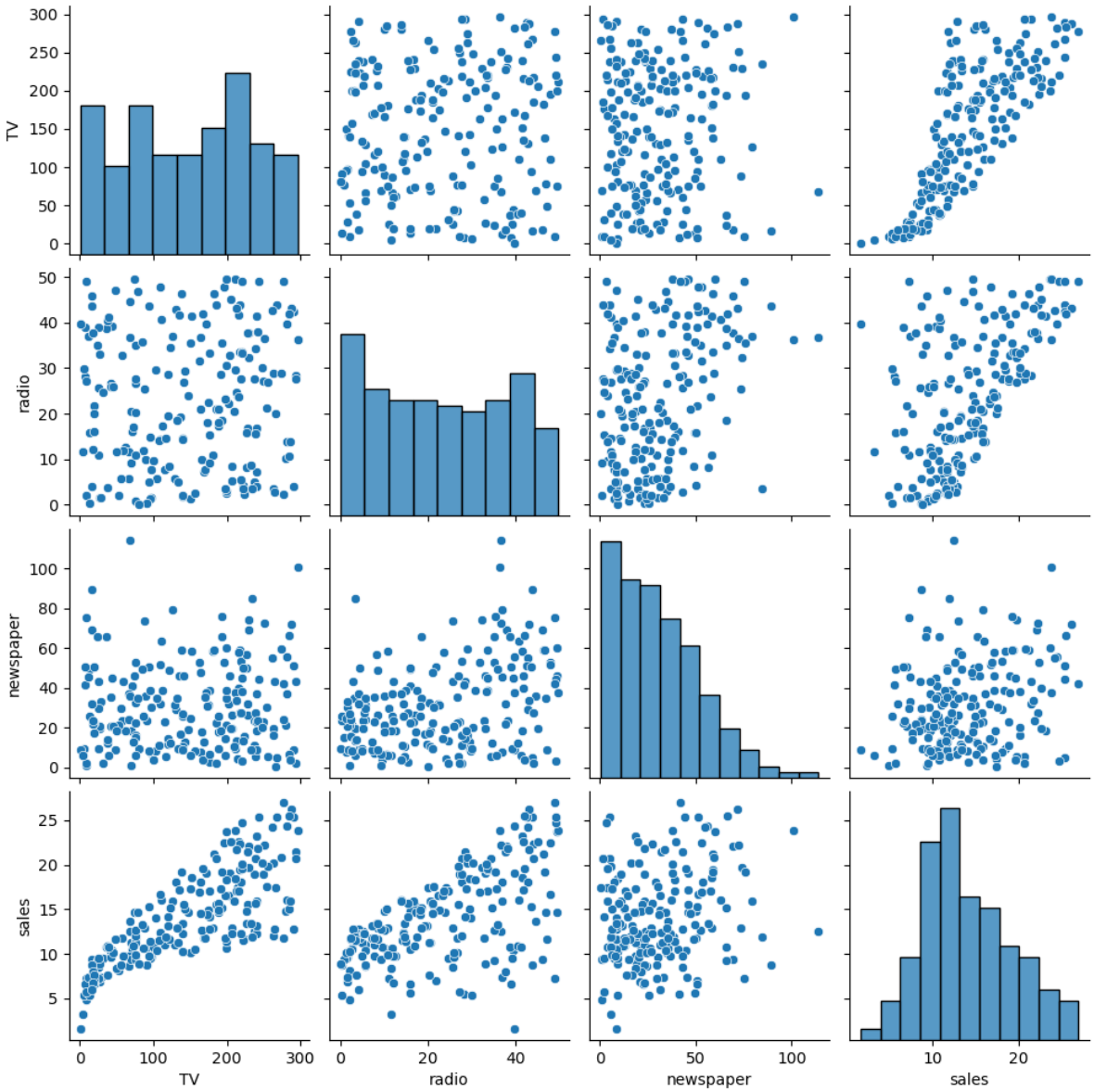
In [8]:

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

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



In [9]:

```
1 sns.pairplot(df);
```



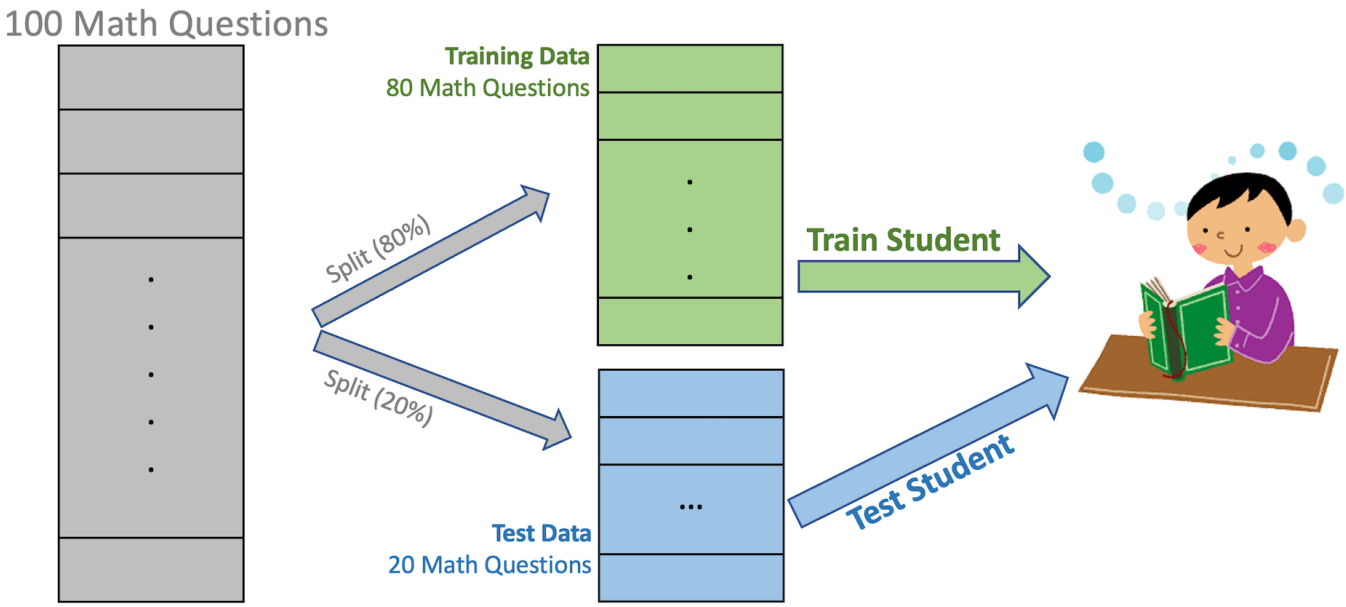
IV. Train-Test-Split

a. A Bad Approach

A terrible approach is to use all of the available data to train the model and then test the model using the same data

b. Train-Test-Split Method

- Holdout cross validation method involves removing a certain portion of data and using it as test data.
- The machine learning model is trained on the training data and then asked to predict the output on the test data




```
from sklearn.model_selection import train_test_split
X_train, X_test = train_test_split(X, test_size=0.2, shuffle=True)
```

Classification

#	Height (inches)	Weight (kgs)	B.P. Sys	B.P. Dia	Heart Disease
1	62	70	120	80	No
2	72	90	110	70	No
3	74	80	130	70	No
4	65	120	150	90	Yes
5	67	100	140	85	Yes
6	64	110	130	90	No
7	69	150	170	100	Yes
8	66	125	145	90	?
9	74	67	110	60	?

Features: Height (inches), Weight (kgs), B.P. Sys, B.P. Dia — Label: Heart Disease

4-D Feature Vector (x_i)

Training Data (X and Y)

Test Data

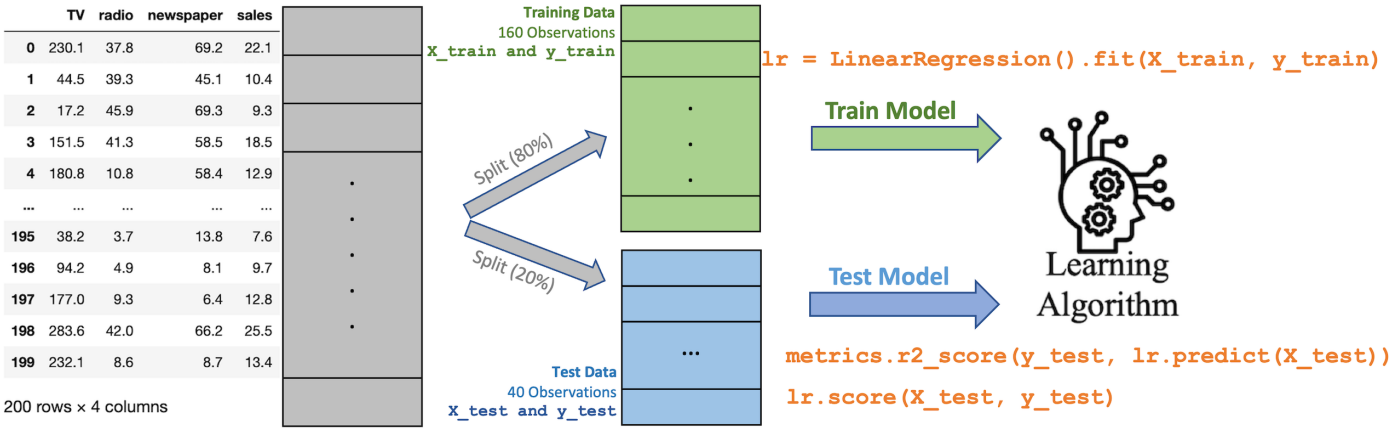
Regression

#	Height (inches)	Weight (kgs)	B.P. Sys	B.P. Dia	Cholesterol Level
1	62	70	120	80	150.35
2	72	90	110	70	160.46
3	74	80	130	70	130.05
4	65	120	150	90	200.19
5	67	100	140	85	190.26
6	64	110	130	90	130.33
7	69	150	170	100	250.10
8	66	125	145	90	?
9	74	67	110	60	?

Features: Height (inches), Weight (kgs), B.P. Sys, B.P. Dia — Label: Cholesterol Level

$$D = \{(\overrightarrow{x_1}, y_1), (\overrightarrow{x_2}, y_2), (\overrightarrow{x_3}, y_3), \cdots (\overrightarrow{x_n}, y_n)\} \subseteq X \times Y$$

c. Applying Train-Test-Split Method on Advertising Dataset




```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, shuffle=True)
```

Later we will see Cross Validation, which is a process that simulates multiple Train-Test-Splits on the available dataset

In [10]:

```
1 df
```

Out[10]:

	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
199	232.1	8.6	8.7	13.4

200 rows × 4 columns

In [11]:

```
1 X = df.drop('sales', axis=1)     # df.iloc[:,0:2]
2 X
```

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
...	...	...	...
195	38.2	3.7	13.8
196	94.2	4.9	8.1
197	177.0	9.3	6.4
198	283.6	42.0	66.2
199	232.1	8.6	8.7

200 rows × 3 columns

In [12]:

```
y = df['sales']     # df.iloc[:,-1]
y
```

Out[12]:

```
0      22.1
1      10.4
2       9.3
3      18.5
4      12.9
       ...
195     7.6
196     9.7
197    12.8
198    25.5
199    13.4
Name: sales, Length: 200, dtype: float64
```

In [13]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_
len(df), len(X_train), len(y_train), len(X_test), len(y_test)
```

Out[13]:

```
(200, 160, 160, 40, 40)
```

In [14]:

```
X_train
```

Out[14]:

	TV	radio	newspaper
7	120.2	19.6	11.6
191	75.5	10.8	6.0
59	210.7	29.5	9.3
34	95.7	1.4	7.4
175	276.9	48.9	41.8
...	...	...	...
23	228.3	16.9	26.2
15	195.4	47.7	52.9
130	0.7	39.6	8.7
69	216.8	43.9	27.2
111	241.7	38.0	23.2

160 rows × 3 columns

In [15]:

```
1 y_train
```

Out[15]:

```
7      13.2
191     9.9
59     18.4
34      9.5
175    27.0
       ...
23     15.5
15     22.4
130     1.6
69     22.3
111    21.8
Name: sales, Length: 160, dtype: float64
```

In [16]:

```
1 X_test.head()
```

Out[16]:

	TV	radio	newspaper
179	165.6	10.0	17.6
166	17.9	37.6	21.6
187	191.1	28.7	18.2
119	19.4	16.0	22.3
186	139.5	2.1	26.6

In [17]:

```
1 y_test.head()
```

Out[17]:

```
179    12.6
166     8.0
187    17.3
119     6.6
186    10.3
Name: sales, dtype: float64
```

V. Create Model/Estimator Instance

In sklearn, an estimator is a Python object that implements the *fit(X, y)* and *predict(X)* methods

```
from sklearn.model_family import Estimator
```

```
from sklearn.linear_model   import
LinearRegression,SGDRegressor,Lasso,Ridge,ElasticNet,LogisticRegression

from sklearn.neighbors   import KNeighborsRegressor, KNeighborsClassifier

from sklearn.svm         import LinearSVR, LinearSVC, SVR, SVC

from sklearn.naive_bayes import BernoulliNB, MultinomialNB,
CategoricalNB, GaussianNB

from sklearn.tree        import DecisionTreeRegressor,
DecisionTreeClassifier

from sklearn.ensemble   import RandomForestRegressor,
RandomForestClassifier, AdaBoostRegressor, AdaBoostClassifier,
GradientBoostingRegressor, GradientBoostingClassifier
```

In [18]:

```
from sklearn.linear_model import LinearRegression
```

In [19]:

```
#help(LinearRegression)
```

In [20]:

```
lr_model = LinearRegression()
```

VI Train the Model

In [21]:

```
lr_model.fit(X_train, y_train);
```

In [22]:

```
lr_model.coef_
```

Out[22]:

```
array([ 0.04597903,  0.18579595, -0.0030992 ])
```

In [23]:

```
lr_model.intercept_
```

Out[23]:

```
3.055121653762061
```

In [24]:

```
1 df.head()
```

Out[24]:

	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

- Synergy Effect or Interaction Effect is a phenomenon that arises in the multiple linear regression setting in machine learning, when increase in the value of one Independent variable increases the impact of another Independent variable on the dependent Variable.

VII. Evaluate the Model

Evaluation allows us to check the accuracy of our trained model by testing it on data that has never been used for training

In [25]:

```
1 X_test.head()
```

Out[25]:

	TV	radio	newspaper
179	165.6	10.0	17.6
166	17.9	37.6	21.6
187	191.1	28.7	18.2
119	19.4	16.0	22.3
186	139.5	2.1	26.6

In [26]:

```
1 y_test.head()
```

Out[26]:

```
179    12.6
166     8.0
187    17.3
119     6.6
186    10.3
Name: sales, dtype: float64
```

In [27]:

```
1 y_predicted = lr_model.predict(X_test)
2 y_predicted
```

Out[27]:

```
array([12.47266269, 10.79713122, 17.11765267,  6.85073786,  9.7769292
1,
       11.72016397, 19.24446371, 10.46662309, 15.44509966, 16.1313990
8,
        9.78080142, 14.04705019, 16.53392308, 17.37352548,  8.8499558
3,
       23.11499   ,  8.19088085, 13.19373337,  9.84007626, 11.3593746
,
       18.77622105, 12.03294526, 18.98214139, 20.44351881, 14.0199199
,
       13.49422896, 12.15922044, 24.04539408,  9.17661604, 14.7282961
2,
       20.02443673,  9.83109916, 19.37855769, 21.19271869, 18.1905889
,
       20.82406301,  6.60613385, 15.29816677, 20.363702  ,  6.1563066
2])
```

In [28]:

```
test_residuals = y_test - y_predicted
test_residuals
```

Out[28]:

```
179    0.127337
166   -2.797131
187    0.182347
119   -0.250738
186    0.523071
134   -0.920164
142    0.855536
12    -1.266623
118    0.454900
178   -4.331399
138   -0.180801
43    -1.147050
194    0.766077
11     0.026475
44    -0.349956
101    0.685010
49     1.509119
4     -0.293733
149    0.259924
31     0.540625
188   -2.876221
116    0.167055
33    -1.582141
0      1.656481
181   -1.819920
161   -0.194229
2     -2.859220
183    2.154606
125    1.423384
81    -2.428296
176    0.175563
145    0.468901
124    0.321442
55     2.507281
153    0.809411
137   -0.024063
182    2.093866
156    0.001833
104    0.336298
106    1.043693
Name: sales, dtype: float64
```

Regression Metrics:

(i) Mean Absolute Error:

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

(ii) Mean Squared Error:

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

(iii) Root Mean Squared Error:

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

In [29]:

```
from sklearn.metrics import mean_absolute_error, mean_squared_error
```

In [30]:

```
mae = mean_absolute_error(y_test, y_predicted)
mse = mean_squared_error(y_test, y_predicted)
rmse = np.sqrt(mse)
print("MAE:", mae)
print("MSE: ", mse)
print("RMSE: ", rmse)
```

```
MAE: 1.0602981028984602
MSE:  2.1533134134530854
RMSE:  1.4674172594913437
```

In [31]:

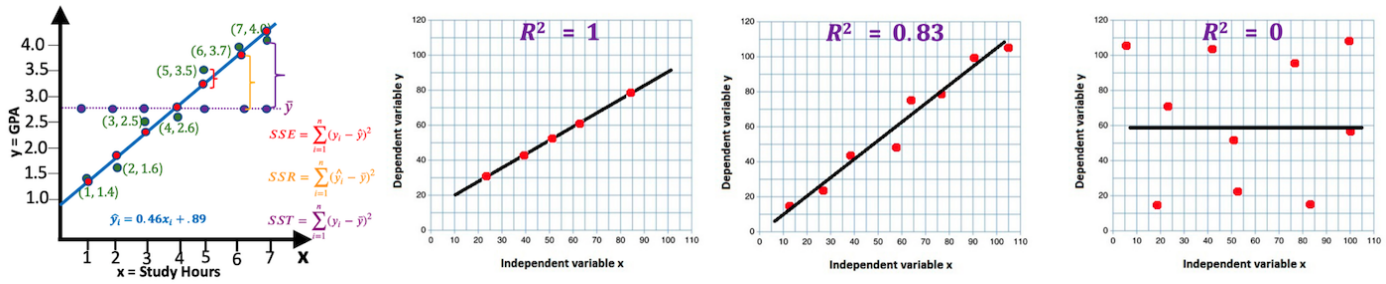
```
df['sales'].mean()
```

Out[31]:

```
14.0225
```

- Since the mean of the sales column is 14, so RMSE of 1.46 can be interpreted as roughly a 10% error.
- **100$ Question:** Does a model having 10% error is good to be deployed?

(iv) R^2 Score (Coefficient of Determination):



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

In [32]:

```
from sklearn.metrics import r2_score
r2 = r2_score(y_test, y_predicted)
print("R2 Score: ", r2)
```

```
R2 Score:  0.9175053931302267
```

> An R2 score of 0.917 means that 91.7% of the variance in the sales column of the advertising dataset is accounted for by the input features.

(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.

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

- where,
  - `n` is number of observations
  - `m` is number of features/predictors/independent variables in model

In [33]:

```
n = len(X_test)
m = 3
r2_adj = 1 - ((1-r2)*(n-1)/(n-m-1))
print("R2 Adjusted: ", r2_adj)
```

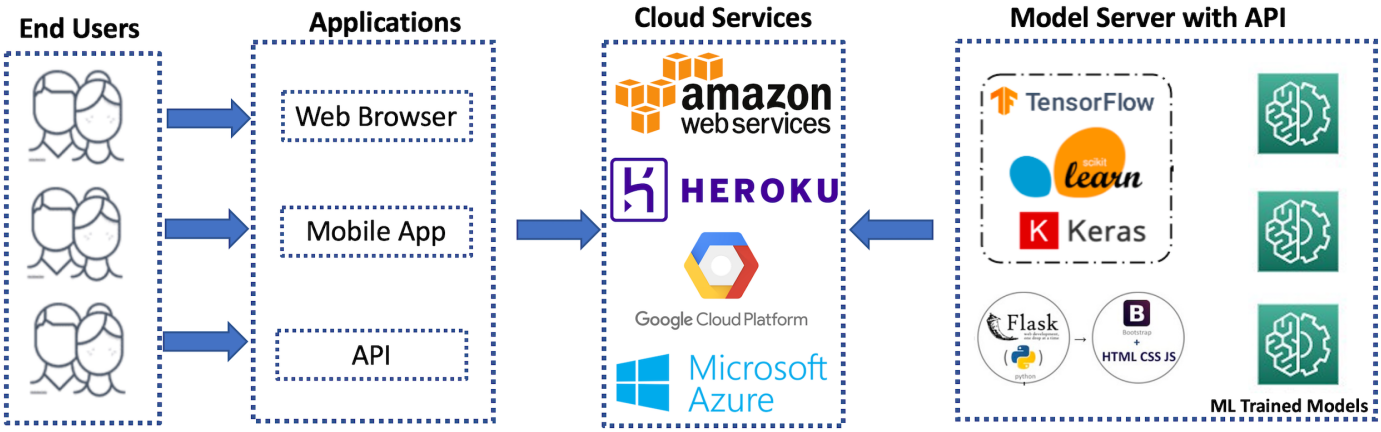
```
R2 Adjusted:  0.9106308425577456
```

VIII. Improve via Hyper-Parameter Tuning

Ser	Hyperparameters	Model Parameters
1	Hyperparameters are set manually by ML engineer/practitioner prior to the start of the model's training.	Model parameters are learnt by the learning algorithm during the training phase.
2	Hyperparameters are used to optimize machine learning model.	Model's parameters are later used for prediction.
3	They are internal to the model.	Thy are external to the model.
4	Examples: Learning rate and number of iterations in gradient descent or Neural Network,Value of K in KNN,, number of trees in RandomForrest, Number of layers in a Neural Network, Number of perceptrons per layer in a Neural Network.	Examples: Coefficients in a Linear or Logistic regression, support vectors in a support vector machine, and weights in an artificial neural network

IX. Model Deployment and Monitoring

- Once you are happy with your training and hyperparameters, guided by the evaluation steps



In [34]:

```
1 final_model = LinearRegression()
```

In [35]:

```
1 final_model.fit(np.array(X),y)
```

Out[35]:

```
LinearRegression()
```

In [36]:

```
1 from joblib import dump
2 dump(final_model, 'adv_model.joblib')
```

Out[36]:

```
['adv_model.joblib']
```

In [37]:

```
from joblib import load
loaded_model = load('adv_model.joblib')
```

In [38]:

```
TV = 200
radio = 150
newspaper = 0
test_input = np.array([[TV, radio, newspaper]])
test_input
```

Out[38]:

```
array([[200, 150,   0]])
```

In [39]:

```
loaded_model.predict(test_input)
```

Out[39]:

```
array([40.371321])
```

5. Task To Do

(i) Apply LinearRegression Model

(ii) Apply SGDRegressor Model

In [40]:

```
from sklearn.datasets import load_diabetes
diabetes = load_diabetes(as_frame=True)
df = pd.DataFrame(data=diabetes.data, columns=diabetes.feature_names)
df['target'] = diabetes.target
df
```

Out[40]:

	age	sex	bmi	bp	s1	s2	s3	s4	
0	0.038076	0.050680	0.061696	0.021872	-0.044223	-0.034821	-0.043401	-0.002592	0.01
1	-0.001882	-0.044642	-0.051474	-0.026328	-0.008449	-0.019163	0.074412	-0.039493	-0.06
2	0.085299	0.050680	0.044451	-0.005671	-0.045599	-0.034194	-0.032356	-0.002592	0.00
3	-0.089063	-0.044642	-0.011595	-0.036656	0.012191	0.024991	-0.036038	0.034309	0.02
4	0.005383	-0.044642	-0.036385	0.021872	0.003935	0.015596	0.008142	-0.002592	-0.03
...	...	...	...	...	...	...	...	...	
437	0.041708	0.050680	0.019662	0.059744	-0.005697	-0.002566	-0.028674	-0.002592	0.03
438	-0.005515	0.050680	-0.015906	-0.067642	0.049341	0.079165	-0.028674	0.034309	-0.01
439	0.041708	0.050680	-0.015906	0.017282	-0.037344	-0.013840	-0.024993	-0.011080	-0.04
440	-0.045472	-0.044642	0.039062	0.001215	0.016318	0.015283	-0.028674	0.026560	0.04
441	-0.045472	-0.044642	-0.073030	-0.081414	0.083740	0.027809	0.173816	-0.039493	-0.00

442 rows × 11 columns