Student Detials

> Title= "Mr"\ Name= "Ali Nawaz"\ email = "nawazktk99@gmail.com"\ whatsapp = "03358043653"

In this Notebook I am are going to combine all the codes and stuff which I have done in Machine learning Part in Python Chilla with Baba Ammar

How is machine learning used in data science?

Image result for data science with machine learning Machine Learning basically automates the process of Data Analysis and makes data-informed predictions in real-time without any human intervention. A Data Model is built automatically and further trained to make real-time predictions. This is where the Machine Learning Algorithms are used in the Data Science Lifecycle

What is machine learning?

Machine learning creates a useful model or program by autonomously testing many solutions against the available data and finding the best fit for the problem. This means machine learning can be great for solving problems that are extremely labor intensive for humans. It can inform decisions and make predictions about complex topics in an efficient and reliable way.

These strengths make machine learning useful in a huge number of different industries. The possibilities for machine learning are vast. This technology has the potential to save lives and solve important problems in healthcare, computer security and more.

How Machine Learning Works?

Machine Learning is, undoubtedly, one of the most exciting subsets of Artificial Intelligence. It completes the task of learning from data with specific inputs to the machine. It’s important to understand what makes Machine Learning work and, thus, how it can be used in the future.

The Machine Learning process starts with inputting training data into the selected algorithm. Training data being known or unknown data to develop the final Machine Learning algorithm. The type of training data input does impact the algorithm, and that concept will be covered further momentarily.

New input data is fed into the machine learning algorithm to test whether the algorithm works correctly. The prediction and results are then checked against each other.

If the prediction and results don’t match, the algorithm is re-trained multiple times until the data scientist gets the desired outcome. This enables the machine learning algorithm to continually learn on its own and produce the optimal answer, gradually increasing in accuracy over time.

What are the Different Types of Machine Learning?

Machine Learning is complex, which is why it has been divided into two primary areas, supervised learning and unsupervised learning. Each one has a specific purpose and action, yielding results and utilizing various forms of data. Approximately 70 percent of machine learning is supervised learning, while unsupervised learning accounts for anywhere from 10 to 20 percent. The remainder is taken up by reinforcement learning.

1. Supervised Learning

In supervised learning, we use known or labeled data for the training data. Since the data is known, the learning is, therefore, supervised, i.e., directed into successful execution. The input data goes through the Machine Learning algorithm and is used to train the model. Once the model is trained based on the known data, you can use unknown data into the model and get a new response.

Supervised Learning

In this case, the model tries to figure out whether the data is an apple or another fruit. Once the model has been trained well, it will identify that the data is an apple and give the desired response.

Here is the list of top algorithms currently being used for supervised learning are:

- Polynomial regression
- Random forest
- Linear regression
- Logistic regression
- Decision trees
- K-nearest neighbors
- Naive Bayes
- The following part of the What is Machine Learning article focuses on unsupervised learning.

2. Semi-Supervised

Semi-supervised machine learning is a combination of supervised and unsupervised machine learning methods.

With more common supervised machine learning methods, you train a machine learning algorithm on a “labeled” dataset in which each record includes the outcome information. This allows the algorithm to deduce patterns and identify relationships between your target variable and the rest of the dataset based on information it already has. In contrast, unsupervised machine learning algorithms learn from a dataset without the outcome variable. In semi-supervised learning, an algorithm learns from a dataset that includes both labeled and unlabeled data, usually mostly unlabeled.

3. Unsupervised Learning

In unsupervised learning, the training data is unknown and unlabeled - meaning that no one has looked at the data before. Without the aspect of known data, the input cannot be guided to the algorithm, which is where the unsupervised term originates from. This data is fed to the Machine Learning algorithm and is used to train the model. The trained model tries to search for a pattern and give the desired response. In this case, it is often like the algorithm is trying to break code like the Enigma machine but without the human mind directly involved but rather a machine.

Unsupervised Learning

In this case, the unknown data consists of apples and pears which look similar to each other. The trained model tries to put them all together so that you get the same things in similar groups.

The top 7 algorithms currently being used for unsupervised learning are:

- Partial least squares
- Fuzzy means
- Singular value decomposition
- K-means clustering
- Apriori
- Hierarchical clustering
- Principal component analysis

4. Reinforcement Learning

Like traditional types of data analysis, here, the algorithm discovers data through a process of trial and error and then decides what action results in higher rewards. Three major components make up reinforcement learning: the agent, the environment, and the actions. The agent is the learner or decision-maker, the environment includes everything that the agent interacts with, and the actions are what the agent does.

Reinforcement learning occurs when the agent chooses actions that maximize the expected reward over a given time. This is easiest to achieve when the agent is working within a sound policy framework.

Herein, we share few examples of machine learning that we use everyday and perhaps have no idea that they are driven by ML.

- Virtual Personal Assistants. ...
- Predictions while Commuting. ...
- Videos Surveillance. ...
- Social Media Services. ...
- Email Spam and Malware Filtering. ...
- Online Customer Support. ...
- Search Engine Result Refining.
- etc

In this Notebook we are going to know about Simple Linear Regression

In [ ]:
```
# import all the lib
import pandas as pd
import seaborn as sns
import numpy as np
import sklearn.metrics as sm
import matplotlib.pyplot as plt
```

1. Loading data Firstly, we will use the Python Pandas library to read our CSV data.

In [ ]:
```
# import all the lib
import pandas as pd
import seaborn as sns
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn import linear_model
import sklearn.metrics as sm
import matplotlib.pyplot as plt

# Importing Linear Regression model from scikit learn
from sklearn.linear_model import LinearRegression
# Importing metrics for the evaluation of the model
from sklearn.metrics import r2_score,mean_squared_error
#Import knearest neighbors Classifier model
from sklearn.neighbors import KNeighborsClassifier
# Importing Linear Regression model from scikit learn
from sklearn.linear_model import LinearRegression
# Importing metrics for the evaluation of the model
from sklearn.metrics import r2_score,mean_squared_error
from sklearn.tree import DecisionTreeClassifier # Import Decision Tree Classifier
from sklearn.model_selection import train_test_split # Import train_test_split function
from sklearn import metrics #Import scikit-learn metrics module for accuracy calculatio
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import auc
import matplotlib.pyplot as plt
# read the dataset using pandas

# read the dataset using pandas
data = pd.read_csv('D:/Python ka Chilla/python_chilla/data/Salary_Data.csv')
```

In [ ]:
```
# This displays the top 5 rows of the data
data.head()
```

	YearsExperience	Salary
0	1.1	39343.0
1	1.3	46205.0

	YearsExperience	Salary
2	1.5	37731.0
3	2.0	43525.0
4	2.2	39891.0

In [ ]:
```
# Provides some information regarding the columns in the data
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30 entries, 0 to 29
Data columns (total 2 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   YearsExperience  30 non-null     float64
 1   Salary           30 non-null     float64
dtypes: float64(2)
memory usage: 608.0 bytes
```

In [ ]:
```
# this describes the basic stat behind the dataset used
data.describe()
```

	YearsExperience	Salary
count	30.000000	30.000000
mean	5.313333	76003.000000
std	2.837888	27414.429785
min	1.100000	37731.000000
25%	3.200000	56720.750000
50%	4.700000	65237.000000
75%	7.700000	100544.750000
max	10.500000	122391.000000

In [ ]:
```
plt.figure(figsize=(10,5))
plt.title('Years of Experience vs Salary', fontsize=13)
plt.xlabel('Experience (yrs)', fontsize=12)
plt.ylabel('Salary', fontsize=12)
plt.scatter(data.YearsExperience, data.Salary, s=75, color='black')
plt.show()
```

In [ ]:
```
# Cooking the data
X = data['YearsExperience']
X.head()
```

```
0    1.1
1    1.3
2    1.5
3    2.0
4    2.2
Name: YearsExperience, dtype: float64
```

In [ ]:
```
# Cooking the data
y = data['Salary']
y.head()
```

```
0    39343.0
1    46205.0
2    37731.0
3    43525.0
4    39891.0
Name: Salary, dtype: float64
```

In [ ]:
```
# Split the data for train and test
X_train,X_test,y_train,y_test = train_test_split(X,y,train_size=0.7,random_state=100)
```

In [ ]:
```
# Create new axis for x column
X_train = X_train[:,np.newaxis]
X_test = X_test[:,np.newaxis]
```

```
C:\Users\Ali\anaconda3\envs\python-chilla\lib\site-packages\ipykernel_launcher.py:2: Fut
ureWarning: Support for multi-dimensional indexing (e.g. `obj[:, None]`) is deprecated a
nd will be removed in a future version.  Convert to a numpy array before indexing instea
d.

C:\Users\Ali\anaconda3\envs\python-chilla\lib\site-packages\ipykernel_launcher.py:3: Fut
ureWarning: Support for multi-dimensional indexing (e.g. `obj[:, None]`) is deprecated a
```

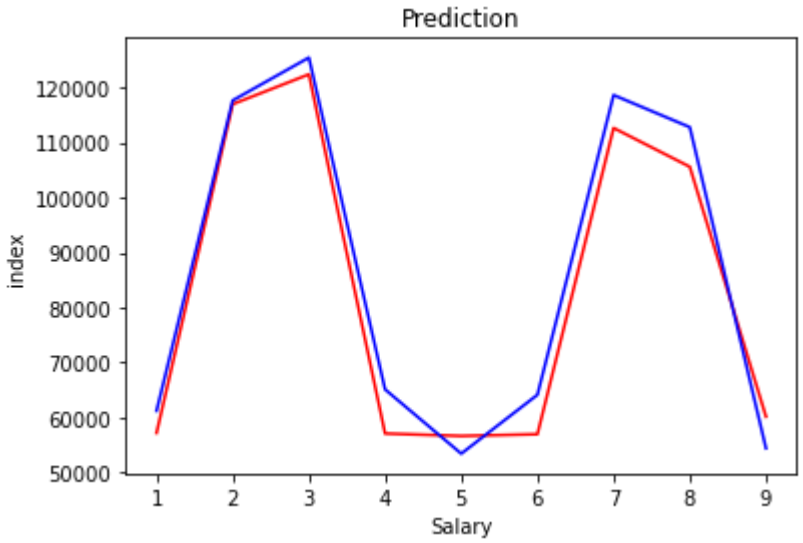
```
nd will be removed in a future version.  Convert to a numpy array before indexing instea
d.
  This is separate from the ipykernel package so we can avoid doing imports until
```

In [ ]:
```
# Fitting the model
lr = LinearRegression()
lr.fit(X_train,y_train)
```

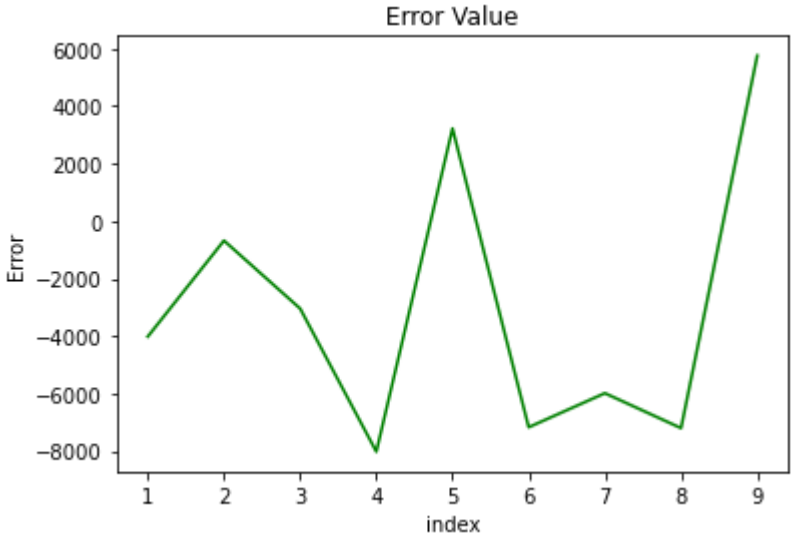
```
LinearRegression()
```

In [ ]:
```
# Predicting the Salary for the Test values
y_pred = lr.predict(X_test)
```

In [ ]:
```
# Plotting the actual and predicted values

c = [i for i in range (1,len(y_test)+1,1)]
plt.plot(c,y_test,color='r',linestyle='-')
plt.plot(c,y_pred,color='b',linestyle='-')
plt.xlabel('Salary')
plt.ylabel('index')
plt.title('Prediction')
plt.show()
```



In [ ]:
```
# plotting the error
c = [i for i in range(1,len(y_test)+1,1)]
plt.plot(c,y_test-y_pred,color='green',linestyle='-')
plt.xlabel('index')
plt.ylabel('Error')
plt.title('Error Value')
plt.show()
```



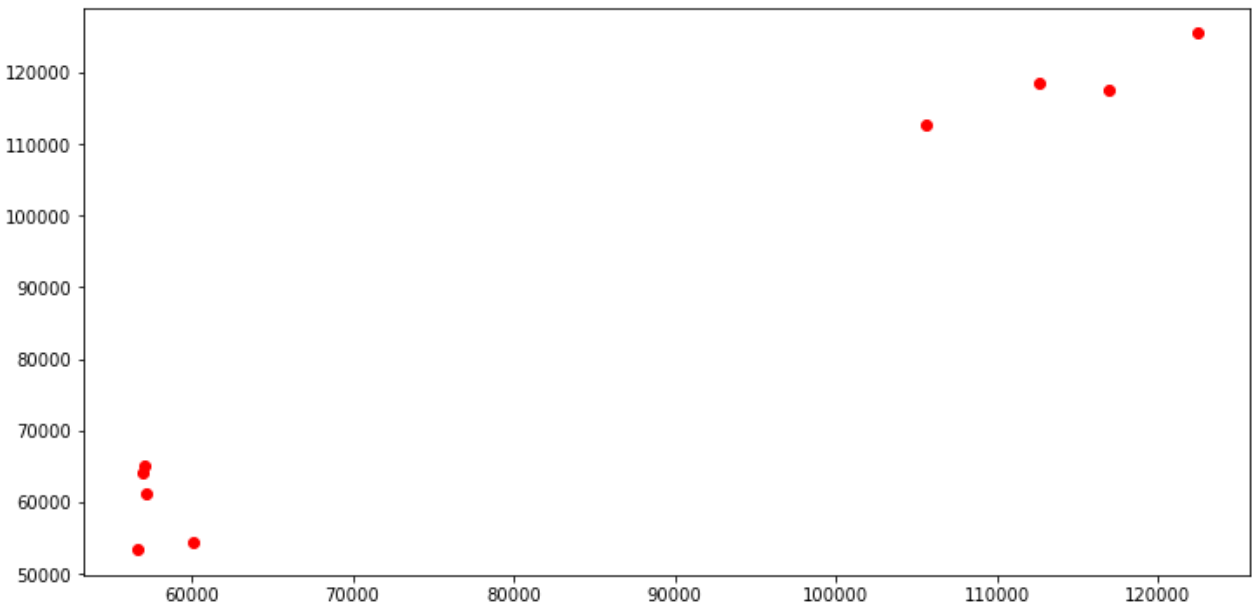
In [ ]:
```
# calculate Mean square error
mse = mean_squared_error(y_test,y_pred)
```

In [ ]:
```
# Calculate R square vale
rsq = r2_score(y_test,y_pred)
```

In [ ]:
```
print('mean squared error :',mse)
print('r square :',rsq)
```

```
mean squared error : 30310299.043402452
r square : 0.9627668685473267
```

In [ ]:
```
# Just plot actual and predicted values for more insights
plt.figure(figsize=(12,6))
plt.scatter(y_test,y_pred,color='r',linestyle='-')
plt.show()
```



In [ ]:
```
# Intecept and coeff of the line
print('Intercept of the model:',lr.intercept_)
print('Coefficient of the line:',lr.coef_)
```

```
Intercept of the model: 25202.887786154883
Coefficient of the line: [9731.20383825]
```

Class Students Data with Machine Learning

In [ ]:
```
# load dataset
df = pd.read_csv("D:/Python ka Chilla/python_chilla/data/mldata.csv")
df['age'] = df['age'].astype(float)
df['weight'] = df['weight'].astype(float)
df['height'] = df['height'].astype(float)
df.head()
```

	age	weight	gender	likeness	height
0	27.0	76.0	Male	Biryani	170.688
1	41.0	70.0	Male	Biryani	165.000
2	29.0	80.0	Male	Biryani	171.000
3	27.0	102.0	Male	Biryani	173.000
4	29.0	67.0	Male	Biryani	164.000

In [ ]:
```
# Take relevant data
workshop_data = df[["age","weight","height"]]
workshop_data.head()
```

	age	weight	height
0	27.0	76.0	170.688
1	41.0	70.0	165.000
2	29.0	80.0	171.000
3	27.0	102.0	173.000
4	29.0	67.0	164.000

In [ ]:
```
X = workshop_data.iloc[:, :-1].values #get a copy of dataset exclude last column
y = workshop_data.iloc[:, 1].values #get array of dataset in column 1st
```

In [ ]:
```
import plotly.express as px
fig = px.imshow(workshop_data.corr())
fig.show()
```

In [ ]:
```
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=.33,random_state=42)
```

In [ ]:
```
col = ['X_train', 'X_test', 'y_train', 'y_test']
data = [X_train, X_test, y_train, y_test]
for i in range(len(col)):
    print(f'Shape of {col[i]}: {data[i].shape}')
```

```
Shape of X_train: (196, 2)
Shape of X_test: (49, 2)
Shape of y_train: (196,)
Shape of y_test: (49,)
```

In [ ]:
```
from sklearn.linear_model import LinearRegression
lr = LinearRegression()
lr.fit(X_train, y_train)
```

```
LinearRegression()
```

In [ ]:
```
#Prediciting Y from Linear regression Model
y_pred = lr.predict(X_test)
```

In [ ]:
```
x_range = np.linspace(X.min(), X.max(), 100)
y_range = lr.predict(x_range.reshape(-1, 1))

fig = go.Figure([
        go.Scatter(x=X_train.squeeze(), y=y_train,
                   name='train', mode='markers'),
        go.Scatter(x=X_test.squeeze(), y=y_test,
                   name='test', mode='markers'),
        go.Scatter(x=x_range, y=y_range,
                   name='prediction')
    ])

fig.show()
```

In [ ]:
```
#Assigning Coefficient (slope) to b
b = lr.coef_
```

In [ ]:
```
print("Coefficient  :" , b)
```

```
Coefficient  : [-8.3603836e-17  1.0000000e+00]
```

In [ ]:
```
#Assigning Y-intercept to a
a = lr.intercept_
```

In [ ]:
```
print("Intercept : ", a)
```

```
Intercept :  -1.4210854715202004e-14
```

In [ ]:
```
#y_predict(11)
print(lr.predict([[11]]))
```

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

```
regr.fit(X_train, y_train)
print(f'Coefficients: {regr.coef_}')
print(f'Intercepts: {regr.intercept_}')
```

```
Coefficients: [-4.6705726e-16  1.0000000e+00]
Intercepts: -1.4210854715202004e-14
```

In [ ]:

```
fig, ax = plt.subplots(1, 2, figsize=(15, 5))
ax[0].set_title('Training Set', fontsize=15)
ax[0].scatter(X_train, y_train, color='black')
ax[0].plot(X_train, regr.coef_*X_train + regr.intercept_, '-r')
ax[0].set_xlabel('Experience (yrs)')
ax[0].set_ylabel('Salary')
ax[1].set_title('Testing Set', fontsize=15)
ax[1].scatter(X_test, y_test,  color='black')
ax[1].plot(X_test, regr.coef_*X_test + regr.intercept_, '-r')
ax[1].set_xlabel('Experience (yrs)')
ax[1].set_ylabel('Salary')
plt.show()
```

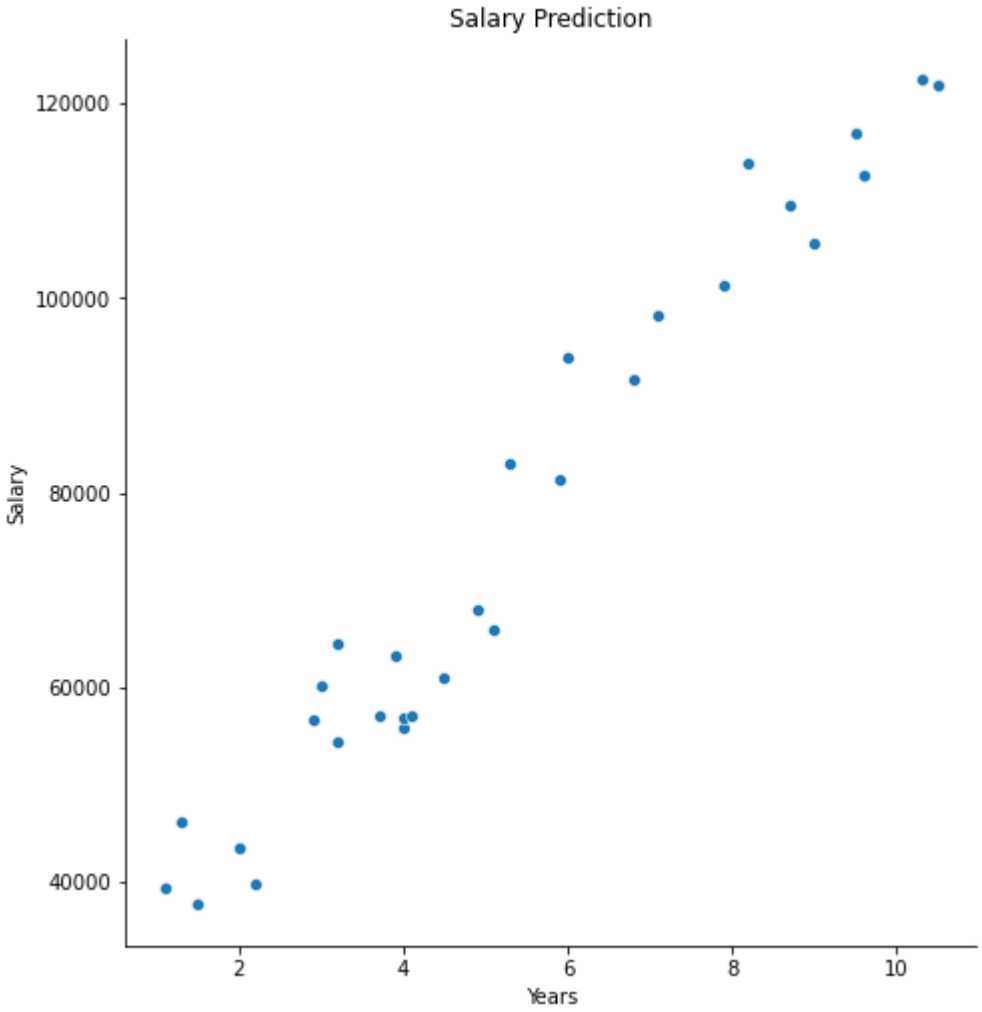
In [ ]:

```
fig, ax = plt.subplots(1, 2, figsize=(15, 5))
ax[0].set_title('Training Set', fontsize=15)
ax[0].scatter(X_train, y_train, s=75, color='black')
ax[0].set_xlabel('Experience (yrs)')
ax[0].set_ylabel('Salary')
ax[1].set_title('Testing Set', fontsize=15)
ax[1].scatter(X_test, y_test, s=75, color='black')
ax[1].set_xlabel('Experience (yrs)')
ax[1].set_ylabel('Salary')
plt.show()
```

In [ ]:

```
# These Plots help to explain the values and how they are scattered

plt.figure(figsize=(12,6))
sns.pairplot(data,x_vars=['YearsExperience'],y_vars=['Salary'],size=7,kind='scatter')
plt.xlabel('Years')
plt.ylabel('Salary')
plt.title('Salary Prediction')
plt.show()
```

```
C:\Users\Ali\anaconda3\envs\python-chilla\lib\site-packages\seaborn\axisgrid.py:2076: Us
erWarning: The `size` parameter has been renamed to `height`; please update your code.
  warnings.warn(msg, UserWarning)
<Figure size 864x432 with 0 Axes>
```

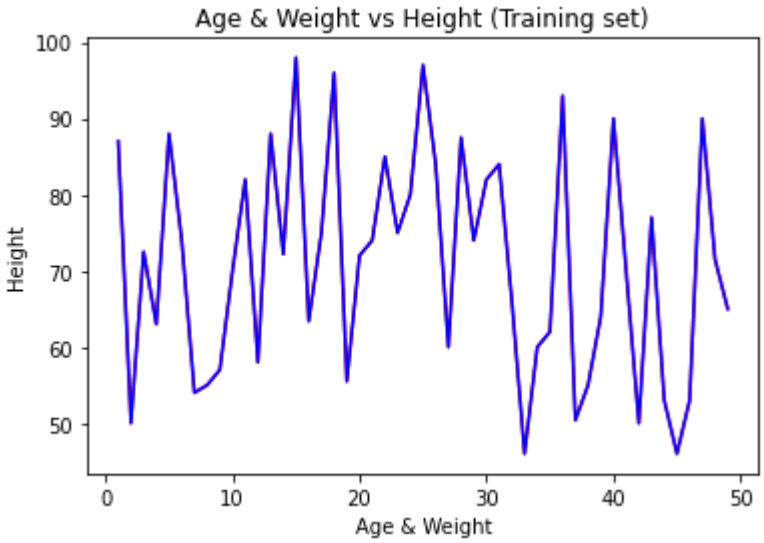


In [ ]:
```
regressor = LinearRegression()
regressor.fit(X_train, y_train)
```

```
LinearRegression()
```

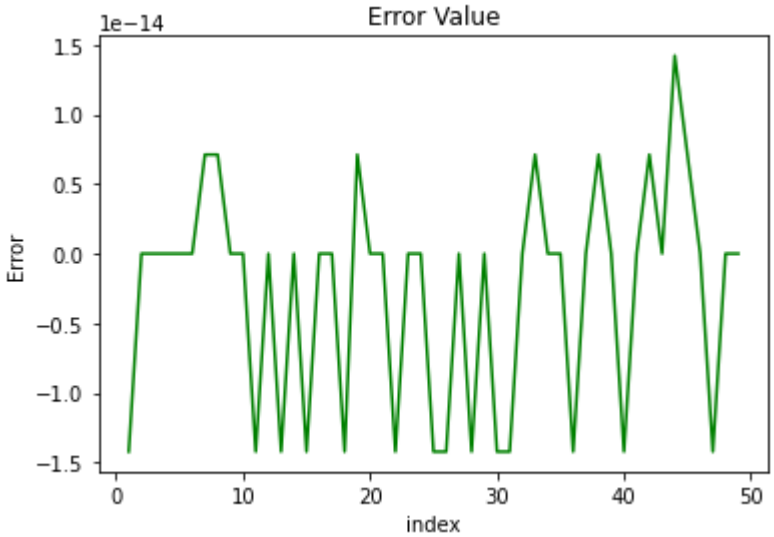
In [ ]:
```
# Predicting the Test set results
y_pred = regressor.predict(X_test)
```

In [ ]:
```
# Plotting the actual and predicted values

c = [i for i in range (1,len(y_test)+1,1)]
plt.plot(c,y_test,color='r',linestyle='-')
plt.plot(c,y_pred,color='b',linestyle='-')
plt.xlabel('Age & Weight')
plt.ylabel('Height')
plt.title('Age & Weight vs Height (Training set)')
plt.show()
```



In [ ]:
```
# Visualizing the Training set results
viz_train = plt
viz_train.scatter(X_train[:,0], y_train, color='red')
viz_train.plot(X_train[:,0], regressor.predict(X_train[:,0]), color='blue')
viz_train.title('Age & Weight vs Height (Training set)')
viz_train.xlabel('Age & Weight')
viz_train.ylabel('Height')
viz_train.show()
```

In [ ]:
```
# plotting the error
c = [i for i in range(1,len(y_test)+1,1)]
plt.plot(c,y_test-y_pred,color='green',linestyle='-')
plt.xlabel('index')
plt.ylabel('Error')
plt.title('Error Value')
plt.show()
```



Linear Regression Assignment

Model Fiting

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

Ploting train model

In [ ]:
```
plt.scatter(X_train, y_train)
plt.plot(X_train, model.predict(X_train))
plt.ylabel("Salary")
plt.xlabel("Experince")
plt.title("Salary vs Experience Plot")
```

```
Text(0.5, 1.0, 'Salary vs Experience Plot')
```

In [ ]:
```
plt.scatter(X_test, y_test)
plt.plot(X_train, model.predict(X_train))
plt.ylabel("Salary")
plt.xlabel("Experince")
plt.title("Salary vs Experience Plot on Test Data")
```

```
Text(0.5, 1.0, 'Salary vs Experience Plot on Test Data')
```

Evaluating model or Testing model

In [ ]:
```
print('Testing Score of Model:', model.score(X_test, y_test))

print('Training Score of Model:',model.score(X_train, y_train))
```

```
Testing Score of Model: 0.988169515729126
Training Score of Model: 0.9411949620562126
```

In [ ]:
```
# Predict on new data and data should be 2 D
x = [[5]]
model.predict(x)
```

```
array([[73342.97478427]])
```

What is a Regression

In Regression, we plot a graph between the variables which best fit the given data points. The machine learning model can deliver predictions regarding the data. In naïve words, "Regression shows a line or curve that passes through all the data points on a target-predictor graph in such a way that the vertical distance between the data points and the regression line is minimum." It is used principally for prediction, forecasting, time series modeling, and determining the causal-effect relationship between variables.

Types of Regression models

1. Linear Regression
2. Polynomial/Multi Regression
3. Logistics Regression

Multi Linear Regression

Multiple Linear Regression is basically indicating that we will be having many features Such as f1, f2, f3, f4, and our output feature f5. If we take the same example as above we discussed, suppose:

f1 is the size of the house.

f2 is bad rooms in the house.

f3 is the locality of the house.

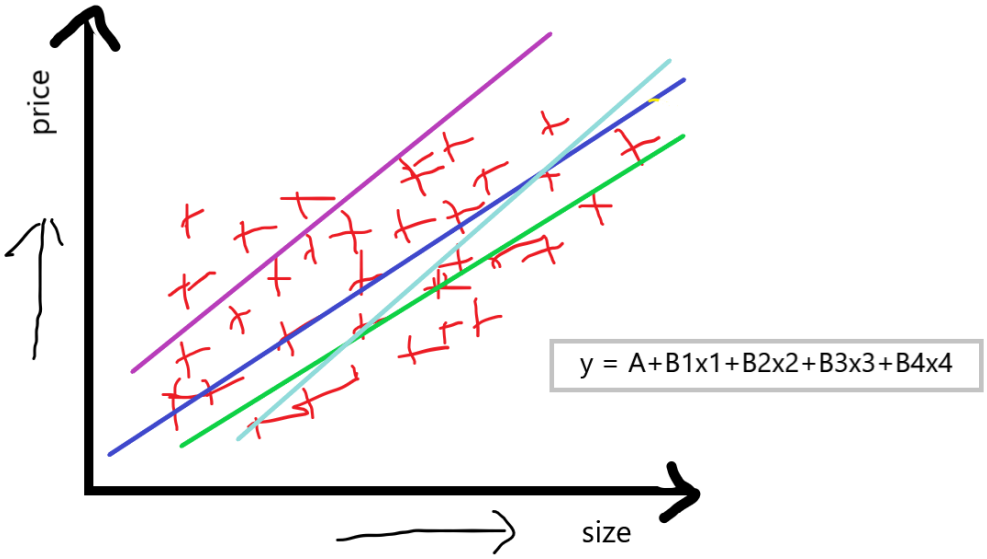
f4 is the condition of the house and,

f5 is our output feature which is the price of the house.

Now, you can see that multiple independent features also make a huge impact on the price of the house, price can vary from feature to feature. When we are discussing multiple linear regression then the equation of simple linear regression y=A+Bx is converted to something like:

```
equation:  y = A+B1x1+B2x2+B3x3+B4x4
```

"If we have one dependent feature and multiple independent features then basically call it a multiple linear regression."



Now, our aim to using the multiple linear regression is that we have to compute A which is an intercept, and B1 B2 B3 B4 which are the slops or coefficient concerning this independent feature, that basically indicates that if we increase the value of x1 by 1 unit then B1 says that how much value it will affect int he price of the house, and this was similar concerning others B2 B3 B4

So, this is a small theoretical description of multiple linear regression now we will use the scikit learn linear regression library to solve the multiple linear regression problem.

x in independent but y is depends on x values so it's the correct form.

In [ ]:
```
df = pd.read_csv('D:/Python ka Chilla/python_chilla/data/ml_data_salary.csv')
df.head()
```

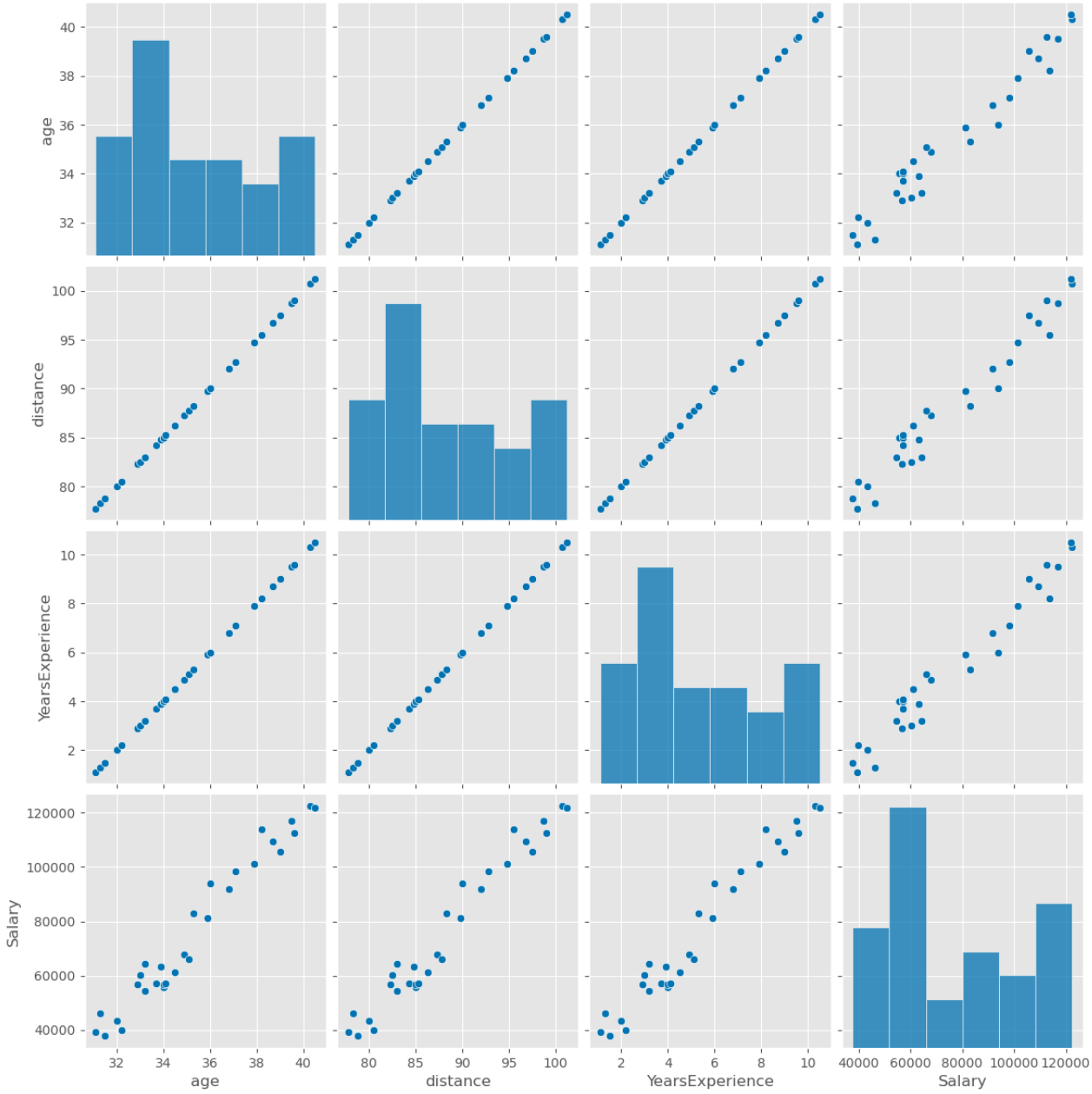
	age	distance	YearsExperience	Salary
0	31.1	77.75	1.1	39343
1	31.3	78.25	1.3	46205
2	31.5	78.75	1.5	37731
3	32.0	80.00	2.0	43525
4	32.2	80.50	2.2	39891

In [ ]:
```
X = df[['age','distance','YearsExperience']]
y = df[['Salary']]
```

In [ ]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0
```

In [ ]:
```
sns.set_palette('colorblind')
sns.pairplot(data=df, height=3)
```

```
<seaborn.axisgrid.PairGrid at 0x23dda1ef630>
```



Model Fiting

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

In [ ]:
```
model.coef_
```

```
array([[-7.88818945e+15,  1.95412592e+14,  7.39965797e+15]])
```

In [ ]:
```
model.intercept_
```

```
array([2.21989739e+17])
```

In [ ]:
```
print('Intercept:', model.intercept_)
print('Coefficients:', model.coef_)
```

```
Intercept: [2.21989739e+17]
Coefficients: [[-7.88818945e+15  1.95412592e+14  7.39965797e+15]]
```

Predict on Test Data

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

```
0.9411819089155802
```

In [ ]:
```
model.score(X_test, y_test)
```

```
0.9882568142614382
```

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

Ploting train model

Evaluating model or Testing model

In [ ]:
```
print('Testing Score of Model:', model.score(X_test, y_test))

print('Training Score of Model:',model.score(X_train, y_train))
```

```
Testing Score of Model: 0.9882568142614382
Training Score of Model: 0.9411819089155802
```

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

	age	distance	YearsExperience	Salary
0	31.1	77.75	1.1	39343
1	31.3	78.25	1.3	46205
2	31.5	78.75	1.5	37731
3	32.0	80.00	2.0	43525
4	32.2	80.50	2.2	39891

In [ ]:
```
# Values to predict
distance = input('What is the Distance you convered? \n')
exp = input('What is the Years Experience? \n')
age = input('Enter Your Age? \n')

pred = model.predict([[float(distance), float(exp), float(age)]])
print("We predict ", pred , " for your Data")
```

```
We predict  [[2.49578913e+17]]  for your Data
```

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

	age	distance	YearsExperience	Salary
0	31.1	77.75	1.1	39343
1	31.3	78.25	1.3	46205
2	31.5	78.75	1.5	37731
3	32.0	80.00	2.0	43525
4	32.2	80.50	2.2	39891

In [ ]:
```
X = df[['age', ]]
X1 = df[['distance', ]]
X2 = df[['YearsExperience', ]]
Y = df['Salary']
```

Model Evavulting

In [ ]:
```
X_test
```

	age	YearsExperience
2	31.5	1.5
28	40.3	10.3
13	34.1	4.1
10	33.9	3.9
26	39.5	9.5
24	38.7	8.7

In [ ]:
```
from sklearn import linear_model
regr = linear_model.LinearRegression()
X = np.asanyarray(X_train[['age','distance','YearsExperience']])
y = np.asanyarray(y_train[['Salary']])
regr.fit(X,y)

print('Coefficients: ', regr.coef_)
```

```
Coefficients:  [[-7.88818945e+15  1.95412592e+14  7.39965797e+15]]
```

Making Predictions

In [ ]:
```
r_square=model.score(X_train, y_train)

print('Coefficient of determination(R square):',r_square)
# pred = model.predict(X_test)
y_pred = model.predict(X_test)
from sklearn import metrics
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
```

```
print('Mean Squared Error:', metrics.mean_squared_error(y_test,y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

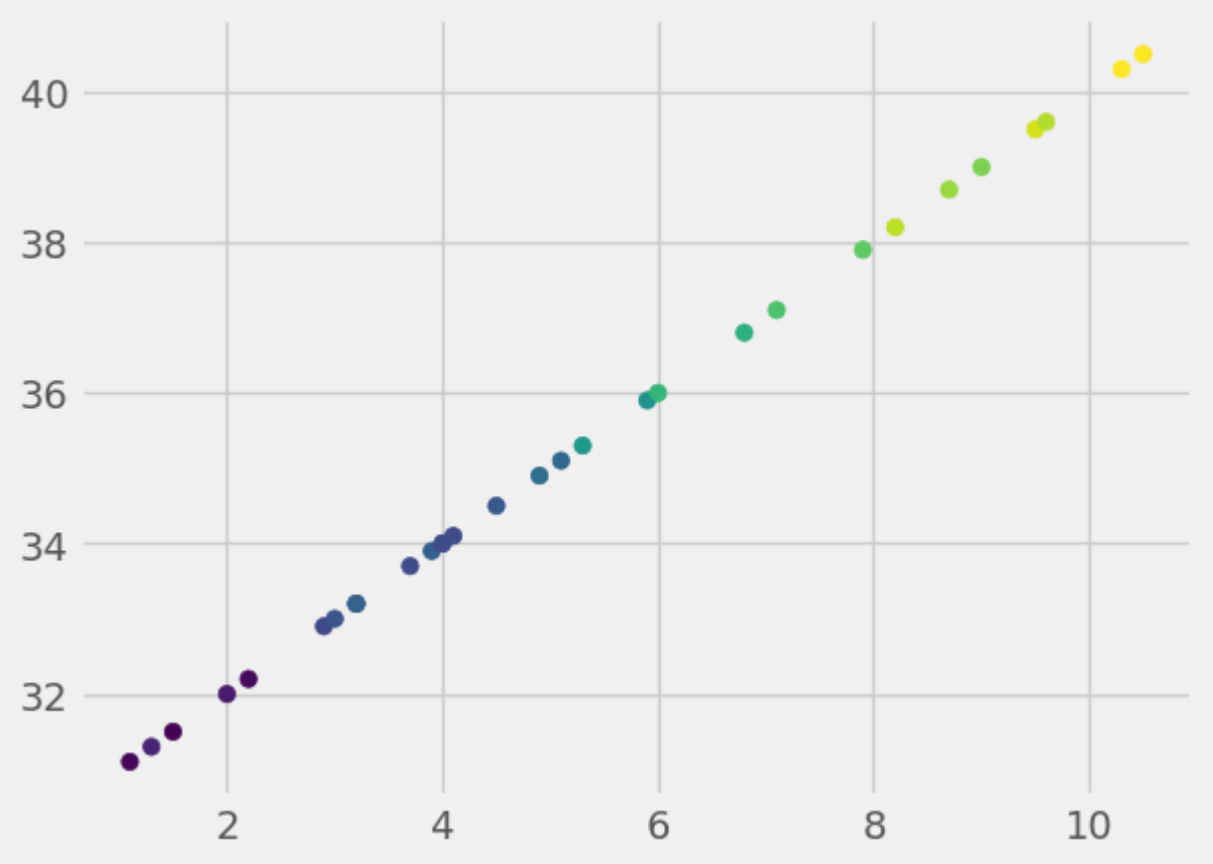
```
Coefficient of determination(R square): 0.9411819089155802
Mean Absolute Error: 2442.5
Mean Squared Error: 12728786.833333334
Root Mean Squared Error: 3567.742540225308
```

Multi Variable Ploting

the ploting can't be done for more than two varible but we can just print single single varible with the dependent variable

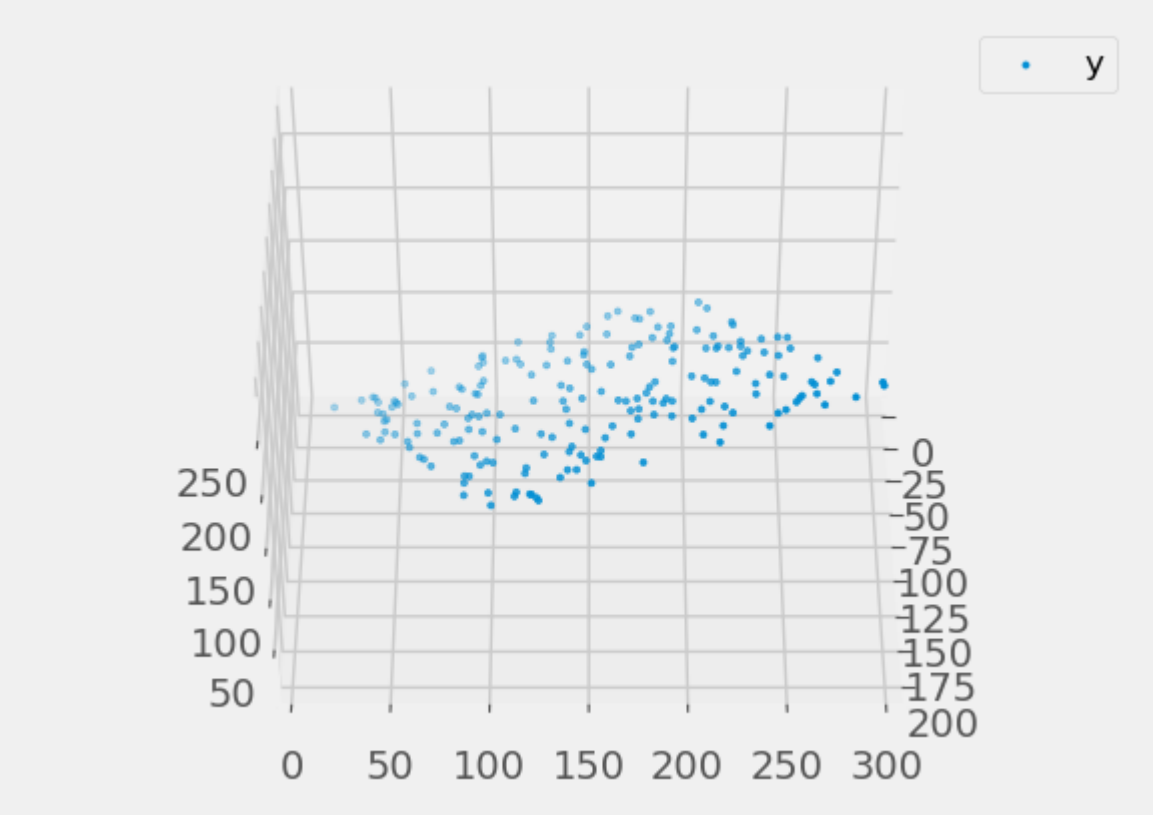
In [ ]:
```
plt.scatter(X2, X, c=Y, cmap='viridis')
```

```
<matplotlib.collections.PathCollection at 0x23ddb4896d8>
```



In [ ]:
```
def generate_dataset(n):
    x = []
    y = []
    random_x1 = np.random.rand()
    random_x2 = np.random.rand()
    for i in range(n):
        x1 = i
        x2 = i/2 + np.random.rand()*n
        x.append([1, x1, x2])
        y.append(random_x1 * x1 + random_x2 * x2 + 1)
    return np.array(x), np.array(y)

x, y = generate_dataset(200)
```

```
plt.rcParams['legend.fontsize'] = 12

fig = plt.figure()
ax = fig.gca(projection ='3d')

ax.scatter(x[:, 1], x[:, 2], y, label ='y', s = 5)
ax.legend()
ax.view_init(45, 0)

plt.show()
```



In [ ]:

```
# Prepare data
X = df[['age', 'YearsExperience']]
Y = df['Salary']

X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, random_state=0


xx_pred = np.linspace(4, 9, 30)  # range of price values
yy_pred = np.linspace(2, 5, 30)  # range of advertising values
xx_pred, yy_pred = np.meshgrid(xx_pred, yy_pred)
model_viz = np.array([xx_pred.flatten(), yy_pred.flatten()]).T

# Predict using model built on previous step
ols = linear_model.LinearRegression()
model = ols.fit(X_train, y_train)
predicted = model.predict(X_test)

# Evaluate model by using it's R^2 score
r2 = model.score(X_train, y_train)

# Plot model visualization
```

```
plt.style.use('fivethirtyeight')

fig = plt.figure(figsize=(12, 4))

ax1 = fig.add_subplot(131, projection='3d')
ax2 = fig.add_subplot(132, projection='3d')
ax3 = fig.add_subplot(133, projection='3d')

axes = [ax1, ax2, ax3]

for ax in axes:
    ax.plot(x, y, z, color='k', zorder=15, linestyle='none', marker='o', alpha=0.5)
    ax.scatter(xx_pred.flatten(), yy_pred.flatten(), predicted, facecolor=(0,0,0,0), s=
    ax.set_xlabel('Price', fontsize=12)
    ax.set_ylabel('Advertising', fontsize=12)
    ax.set_zlabel('Pie Sales', fontsize=12)
    ax.locator_params(nbins=4, axis='x')
    ax.locator_params(nbins=5, axis='x')

ax1.view_init(elev=25, azim=-60)
ax2.view_init(elev=15, azim=15)
ax3.view_init(elev=25, azim=60)

fig.suptitle('Multi-Linear Regression Model Visualization ($R^2 = %.2f$)' % r2, fontsiz

fig.tight_layout()
```

KNN

In [ ]:

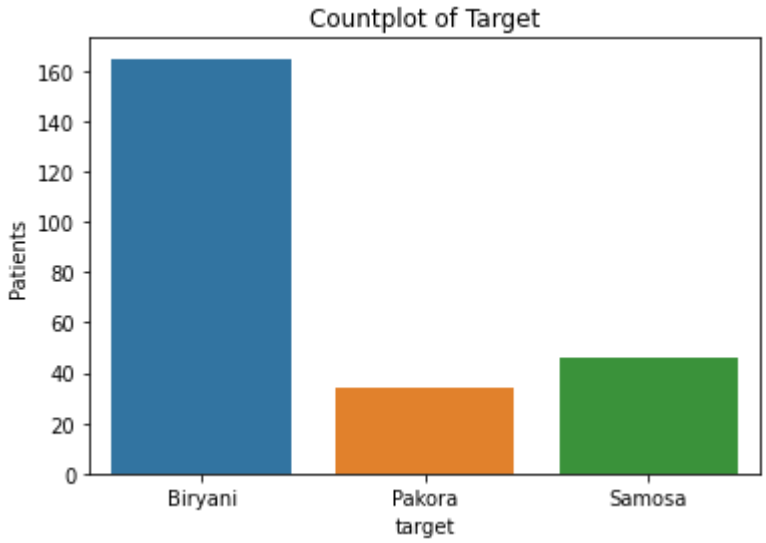
```
# print the names of the features
df = pd.read_csv('D:/Python ka Chilla/python_chilla/data/mldata.csv')

print(df.columns)
```

```
Index(['age', 'weight', 'gender', 'height', 'likeness'], dtype='object')
```

In [ ]:

```
# print the label species(class_0, class_1, class_2)
#2. distribution of target variable.
sns.countplot(df['likeness'])

# Add labels
plt.title('Countplot of Target')
plt.xlabel('target')
plt.ylabel('Patients')
plt.show()
```

```
C:\Users\Ali\anaconda3\envs\python-chilla\lib\site-packages\seaborn\_decorators.py:43: F
utureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the on
ly valid positional argument will be `data`, and passing other arguments without an expl
icit keyword will result in an error or misinterpretation.
  FutureWarning
```



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

	age	weight	gender	height	likeness
0	27	76.0	Male	170.688	Biryani
1	41	70.0	Male	165.000	Biryani
2	29	80.0	Male	171.000	Biryani
3	27	102.0	Male	173.000	Biryani
4	29	67.0	Male	164.000	Biryani

In [ ]:
```
 #split dataset in features and target variable
df['gender'] = df['gender'].replace('Male',  1)
df['gender'] = df['gender'].replace('Female',  0)
feature_cols = ['age', 'gender', 'weight']
X = df[feature_cols] # Features
y = df.likeness # Target variable
```

In [ ]:
```
# Split dataset into training set and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1
```

In [ ]:
```
#Create KNN Classifier
knn = KNeighborsClassifier(n_neighbors=5)

#Train the model using the training sets
knn.fit(X_train, y_train)

#Predict the response for test dataset
y_pred = knn.predict(X_test)
```

In [ ]:
```
train_score = []
test_score = []
k_vals = []
```

```
for k in range(1, 21):
    k_vals.append(k)
    knn = KNeighborsClassifier(n_neighbors = k)
    knn.fit(X_train, y_train)

    tr_score = knn.score(X_train, y_train)
    train_score.append(tr_score)

    te_score = knn.score(X_test, y_test)
    test_score.append(te_score)
```

In [ ]:
```
## score that comes from the testing set only
max_test_score = max(test_score)
test_scores_ind = [i for i, v in enumerate(test_score) if v == max_test_score]
print('Max test score {} and k = {}'.format(max_test_score * 100, list(map(lambda x: x
```

```
Max test score 64.86486486486487 and k = [11, 13, 14, 15, 16, 17, 18, 19, 20]
```

In [ ]:
```
#Import scikit-learn metrics module for accuracy calculation
from sklearn import metrics
# Model Accuracy, how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.6081081081081081
```

Re-generating Model for K=7

Let's build KNN classifier model for k=7.

In [ ]:
```
#Import knearest neighbors Classifier model
from sklearn.neighbors import KNeighborsClassifier

#Create KNN Classifier
knn = KNeighborsClassifier(n_neighbors=7)

#Train the model using the training sets
knn.fit(X_train, y_train)

#Predict the response for test dataset
y_pred = knn.predict(X_test)
```

In [ ]:
```
#Import scikit-learn metrics module for accuracy calculation
from sklearn import metrics
# Model Accuracy, how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.6351351351351351
```

In [ ]:
```
y_pred = knn.predict(X_test)
print(metrics.accuracy_score(y_test, y_pred))
```

```
0.6351351351351351
```

In [ ]:

```
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
print(metrics.accuracy_score(y_test, y_pred))
```

```
0.5675675675675675
```

Locate K value for our dataset

In [ ]:

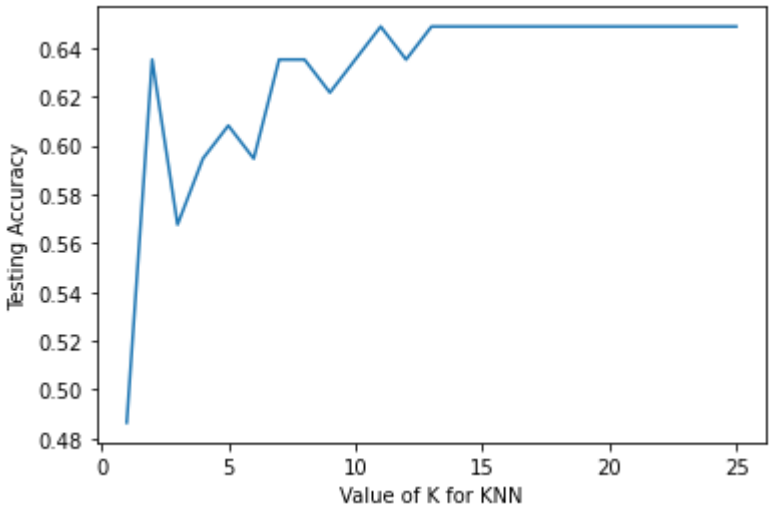
```
# try K=1 through K=25 and record testing accuracy
k_range = range(1, 26)

# We can create Python dictionary using [] or dict()
scores = []

# We use a loop through the range 1 to 26
# We append the scores in the dictionary
for k in k_range:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    y_pred = knn.predict(X_test)
    scores.append(metrics.accuracy_score(y_test, y_pred))

print(scores)
```

```
[0.4864864864864865, 0.6351351351351351, 0.5675675675675675, 0.5945945945945946, 0.6081081081081081, 0.5945945945945946, 0.6351351351351351, 0.6351351351351351, 0.6216216216216216, 0.6351351351351351, 0.6486486486486487, 0.6351351351351351, 0.6486486486486487, 0.6486486486486487, 0.6486486486486487, 0.6486486486486487, 0.6486486486486487, 0.6486486486486487, 0.6486486486486487, 0.6486486486486487, 0.6486486486486487, 0.6486486486486487, 0.6486486486486487]
```

In [ ]:

```
# allow plots to appear within the notebook
%matplotlib inline

# plot the relationship between K and testing accuracy
# plt.plot(x_axis, y_axis)
plt.plot(k_range, scores)
plt.xlabel('Value of K for KNN')
plt.ylabel('Testing Accuracy')
```

```
Text(0, 0.5, 'Testing Accuracy')
```



In [ ]:

```
```

Other Eval Measure

In [ ]:

```
y_pred = knn.predict(X_test)
confusion_matrix(y_test,y_pred)
pd.crosstab(y_test, y_pred, rownames = ['Actual'], colnames =['Predicted'], margins = T
```

Predicted	Biryani	All
Actual		
Biryani	48	48
Pakora	12	12
Samosa	14	14
All	74	74

In [ ]:

```
y_pred_proba = knn.predict_proba(X_test)[:,1]
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)
```

In [ ]:

```
roc_auc_score(y_test, y_pred_proba)
```

In [ ]:

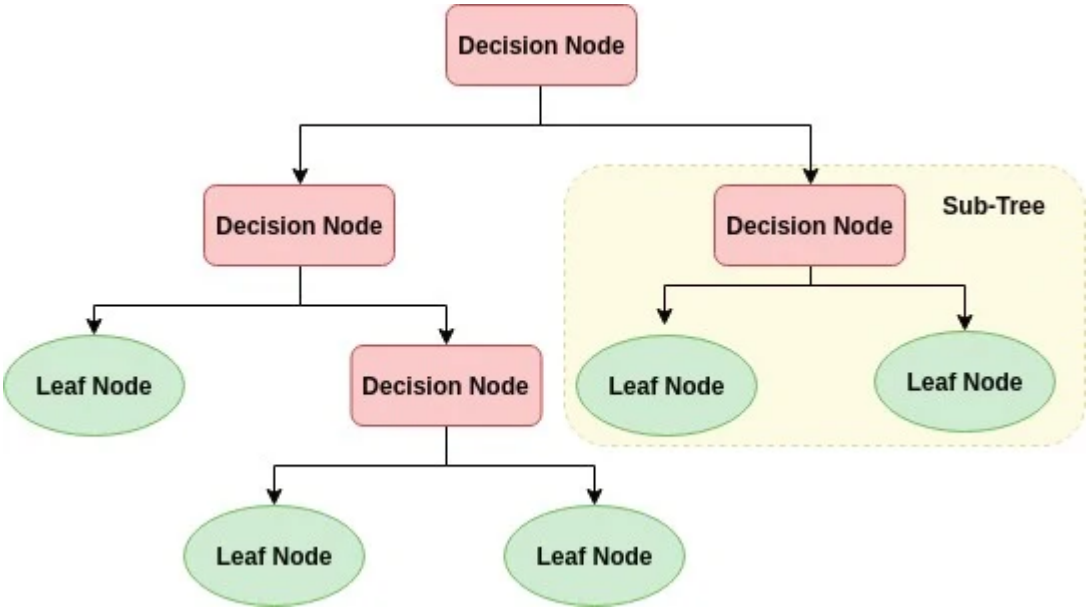
```
precision, recall, thresholds = precision_recall_curve(y_test, y_pred_proba)

plt.figure(figsize = (10,8))
plt.plot([0, 1], [0.5, 0.5],'k--')
plt.plot(recall, precision, label = 'Knn')
plt.xlabel('recall')
plt.ylabel('precision')
plt.title('Knn(n_neighbors = 8) PRC curve')
plt.show()
```

In [ ]:

```
# calculate precision-recall AUC
auc_prc = auc(recall, precision)
print(auc_prc)
```

Decision Tree Algorithm

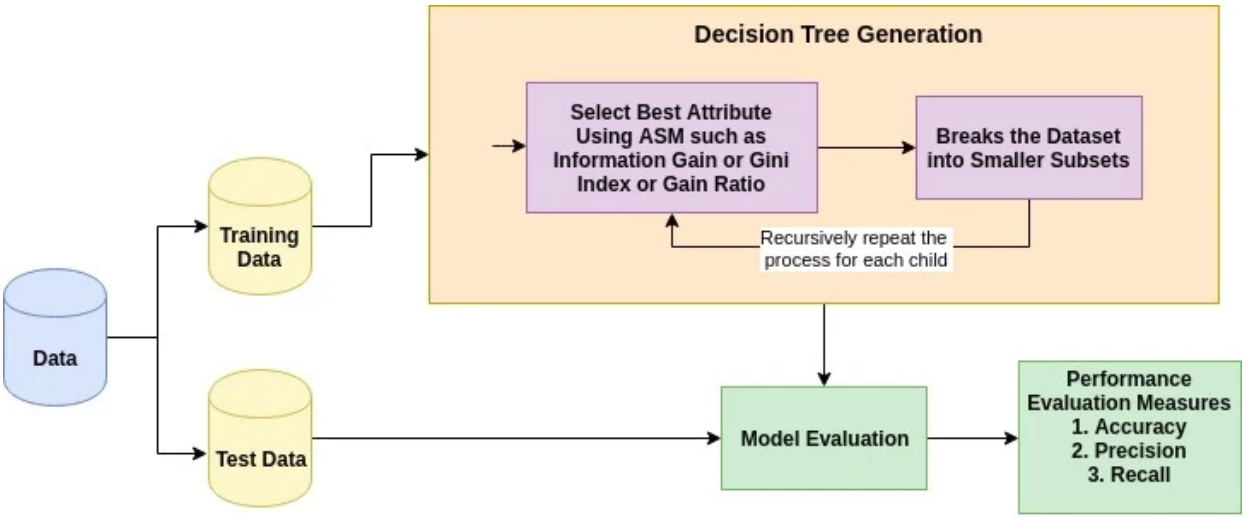
A decision tree is a flowchart-like tree structure where an internal node represents feature(or attribute), the branch represents a decision rule, and each leaf node represents the outcome. The topmost node in a decision tree is known as the root node. It learns to partition on the basis of the attribute value. It partitions the tree in recursively manner call recursive partitioning. This flowchart-like structure helps you in decision making. It's visualization like a flowchart diagram which easily mimics the human level thinking. That is why decision trees are easy to understand and interpret.



How does the Decision Tree algorithm work?

The basic idea behind any decision tree algorithm is as follows:

Select the best attribute using Attribute Selection Measures(ASM) to split the records. Make that attribute a decision node and breaks the dataset into smaller subsets. Starts tree building by repeating this process recursively for each child until one of the condition will match: All the tuples belong to the same attribute value. There are no more remaining attributes. There are no more instances.



For More Information visit (More Details)

In [ ]:
```
# import all the lib
import pandas as pd
import seaborn as sns
import numpy as np
import sklearn.metrics as sm
import matplotlib.pyplot as plt
```

```
# Importing Linear Regression model from scikit learn
from sklearn.linear_model import LinearRegression
# Importing metrics for the evaluation of the model
from sklearn.metrics import r2_score,mean_squared_error
from sklearn.tree import DecisionTreeClassifier # Import Decision Tree Classifier
from sklearn.model_selection import train_test_split # Import train_test_split function
from sklearn import metrics #Import scikit-learn metrics module for accuracy calculatio

# read the dataset using pandas
df = pd.read_csv('D:/Python ka Chilla/python_chilla/data/mldata.csv')
```

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

	age	weight	gender	height	likeness
0	27	76.0	Male	170.688	Biryani
1	41	70.0	Male	165.000	Biryani
2	29	80.0	Male	171.000	Biryani
3	27	102.0	Male	173.000	Biryani
4	29	67.0	Male	164.000	Biryani

Feature Selection

In [ ]:
```
 #split dataset in features and target variable
df['gender'] = df['gender'].replace('Male',  1)
df['gender'] = df['gender'].replace('Female',  0)
feature_cols = ['age', 'gender', 'weight']
X = df[feature_cols] # Features
y = df.likeness # Target variable
```

In [ ]:
```
df['likeness'].value_counts()
```

```
Biryani    165
Samosa      46
Pakora      34
Name: likeness, dtype: int64
```

Splitting Data

To understand model performance, dividing the dataset into a training set and a test set is a good strategy.

In [ ]:
```
# Split dataset into training set and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1
```

Building Decision Tree Model

Let's create a Decision Tree Model using Scikit-learn.

In [ ]:
```
# Create Decision Tree classifer object
clf = DecisionTreeClassifier()

# Train Decision Tree Classifer
clf = clf.fit(X_train,y_train)

#Predict the response for test dataset
y_pred = clf.predict(X_test)
```

In [ ]:
```
# Model Accuracy, how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.5
```

In [ ]:
```
# Create Decision Tree classifer object
clf = DecisionTreeClassifier(criterion="entropy", max_depth=3)

# Train Decision Tree Classifer
clf = clf.fit(X_train,y_train)

#Predict the response for test dataset
y_pred = clf.predict(X_test)

# Model Accuracy, how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.6081081081081081
```

Assignments for Decsion Tree

1. Decision Tree Model Saving

Let's Save a Decision Tree Model

In [ ]:
```
import pickle
import joblib
filename = 'finalized_model_descsion_tree.joblib'
joblib.dump(clf, filename)
```

```
['finalized_model_descsion_tree.joblib']
```

In [ ]:
```
# using pickle
filename = 'finalized_model.sav'
pickle.dump(clf, open(filename, 'wb'))
```

2. Decision Tree Model Loading

Let's load the saved a Decision Tree Model using pickle.

In [ ]:
```
# using job lib
# load the model from disk
loaded_model = joblib.load(filename)
result = loaded_model.score(X_test, y_test)
print(result)
```

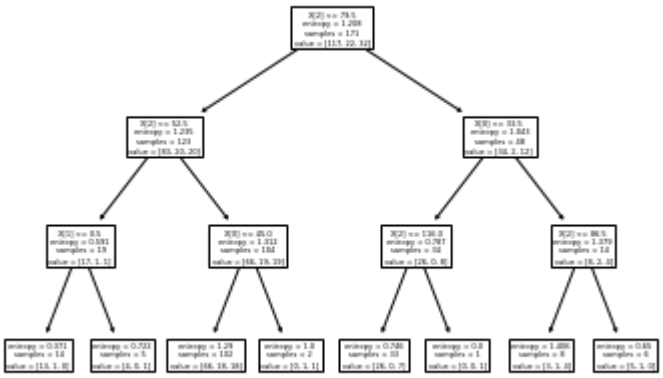
```
0.6081081081081081
```

In [ ]:
```
# using pickle

# load the model from disk
loaded_model = pickle.load(open(filename, 'rb'))
result = loaded_model.score(X_test, y_test)
print(result)
```

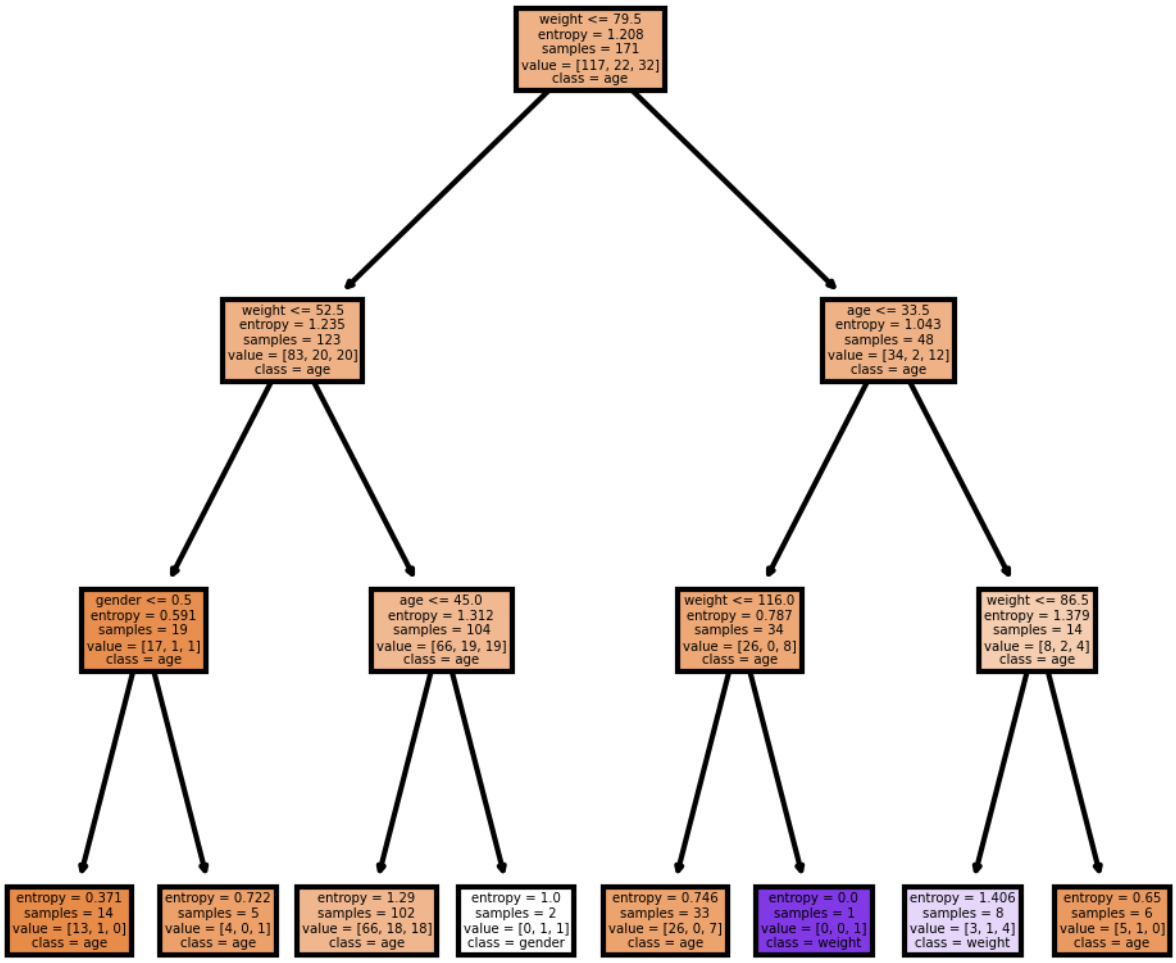
```
0.6081081081081081
```

3. Visalize the Tree of Our Model

In [ ]:
```
from sklearn import tree
tree.plot_tree(clf);
```



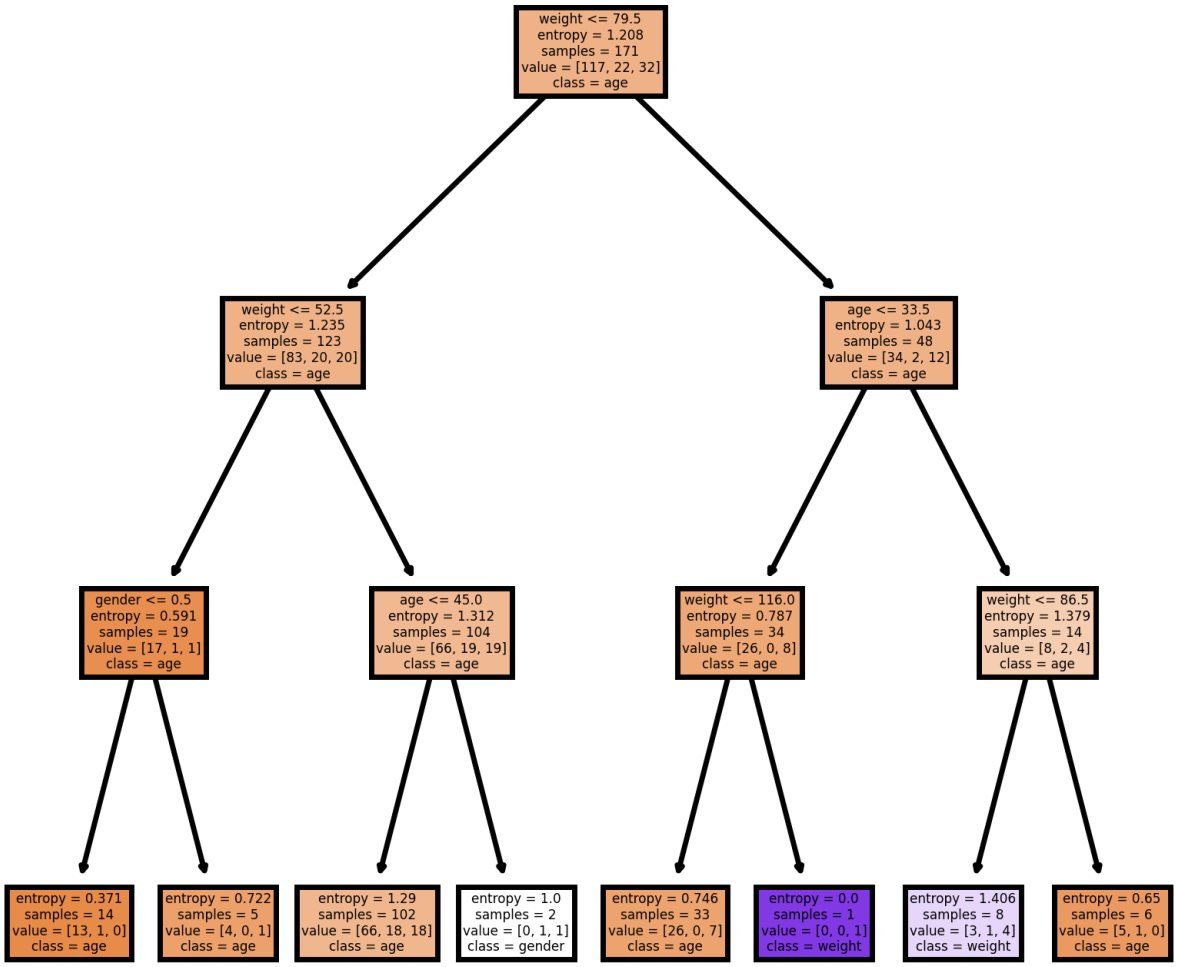
In [ ]:
```
fn=['age', 'gender', 'weight']
cn=['age', 'gender', 'weight']
fig, axes = plt.subplots(nrows = 1,ncols = 1,figsize = (4,4), dpi=300)
tree.plot_tree(clf,
               feature_names = fn,
               class_names=cn,
               filled = True)
fig.savefig('descion_tree.png')
```



In [ ]:
```
tree.export_graphviz(clf, out_file='tree.dot')
```

4. Visalize the Tree and Show Image in HD

In [ ]:
```
fn=['age', 'gender', 'weight']
cn=['age', 'gender', 'weight']
fig, axes = plt.subplots(nrows = 1,ncols = 1,figsize = (4,4), dpi=500)
tree.plot_tree(clf,
               feature_names = fn,
               class_names=cn,
               filled = True)
fig.savefig('descion_tree.png')
```



What is supervised learning?

Supervised learning, also known as supervised machine learning, is a subcategory of machine learning and artificial intelligence. It is defined by its use of labeled datasets to train algorithms that to classify data or predict outcomes accurately. As input data is fed into the model, it adjusts its weights until the model has been fitted appropriately, which occurs as part of the cross validation process. Supervised learning helps organizations solve for a variety of real-world problems at scale, such as classifying spam in a separate folder from your inbox.

How supervised learning works

Supervised learning uses a training set to teach models to yield the desired output. This training dataset includes inputs and correct outputs, which allow the model to learn over time. The algorithm measures its accuracy through the loss function, adjusting until the error has been sufficiently minimized.

Supervised learning can be separated into two types of problems when data mining—classification and regression:

Classification:

uses an algorithm to accurately assign test data into specific categories. It recognizes specific entities within the dataset and attempts to draw some conclusions on how those entities should be labeled or defined. Common classification algorithms are linear classifiers, support vector machines (SVM), decision trees, k-nearest neighbor, and random forest, which are described in more detail below.

Regression:

is used to understand the relationship between dependent and independent variables. It is commonly used to make projections, such as for sales revenue for a given business. Linear regression, logistical regression, and polynomial regression are popular regression algorithms.

Supervised learning algorithms

Various algorithms and computation techniques are used in supervised machine learning processes. Below are brief explanations of some of the most commonly used learning methods, typically calculated through use of programs like R or Python:

1. Neural networks

Primarily leveraged for deep learning algorithms, neural networks process training data by mimicking the interconnectivity of the human brain through layers of nodes. Each node is made up of inputs, weights, a bias (or threshold), and an output. If that output value exceeds a given threshold, it “fires” or activates the node, passing data to the next layer in the network. Neural networks learn this mapping function through supervised learning, adjusting based on the loss function through the process of gradient descent. When the cost function is at or near zero, we can be confident in the model’s accuracy to yield the correct answer.

2. Naive Bayes

Naive Bayes is classification approach that adopts the principle of class conditional independence from the Bayes Theorem. This means that the presence of one feature does not impact the presence of another in the probability of a given outcome, and each predictor has an equal effect on that result. There are three types of Naïve Bayes classifiers: Multinomial Naïve Bayes, Bernoulli Naïve Bayes, and Gaussian Naïve Bayes. This technique is primarily used in text classification, spam identification, and recommendation systems.

3. Linear regression

Linear regression is used to identify the relationship between a dependent variable and one or more independent variables and is typically leveraged to make predictions about future outcomes. When there is only one independent variable and one dependent variable, it is known as simple linear regression. As the number of independent variables increases, it is referred to as multiple linear regression. For each type of linear regression, it seeks to plot a line of best fit, which is calculated through the method of least squares. However, unlike other regression models, this line is straight when plotted on a graph.

4. Logistic regression

While linear regression is leveraged when dependent variables are continuous, logistical regression is selected when the dependent variable is categorical, meaning they have binary outputs, such as "true" and "false" or "yes" and "no." While both regression models seek to understand relationships between data inputs, logistic regression is mainly used to solve binary classification problems, such as spam identification.

5. Support vector machine (SVM)

A support vector machine is a popular supervised learning model developed by Vladimir Vapnik, used for both data classification and regression. That said, it is typically leveraged for classification problems, constructing a hyperplane where the distance between two classes of data points is at its maximum. This hyperplane is known as the decision boundary, separating the classes of data points (e.g., oranges vs. apples) on either side of the plane.

6. K-nearest neighbor

K-nearest neighbor, also known as the KNN algorithm, is a non-parametric algorithm that classifies data points based on their proximity and association to other available data. This algorithm assumes that similar data points can be found near each other. As a result, it seeks to calculate the distance between data points, usually through Euclidean distance, and then it assigns a category based on the most frequent category or average.

Its ease of use and low calculation time make it a preferred algorithm by data scientists, but as the test dataset grows, the processing time lengthens, making it less appealing for classification tasks. KNN is typically used for recommendation engines and image recognition.

7. Random forest

Random forest is another flexible supervised machine learning algorithm used for both classification and regression purposes. The "forest" references a collection of uncorrelated decision trees, which are then merged together to reduce variance and create more accurate data predictions.

In this Notebook I am are going to Explore Logistic Regression

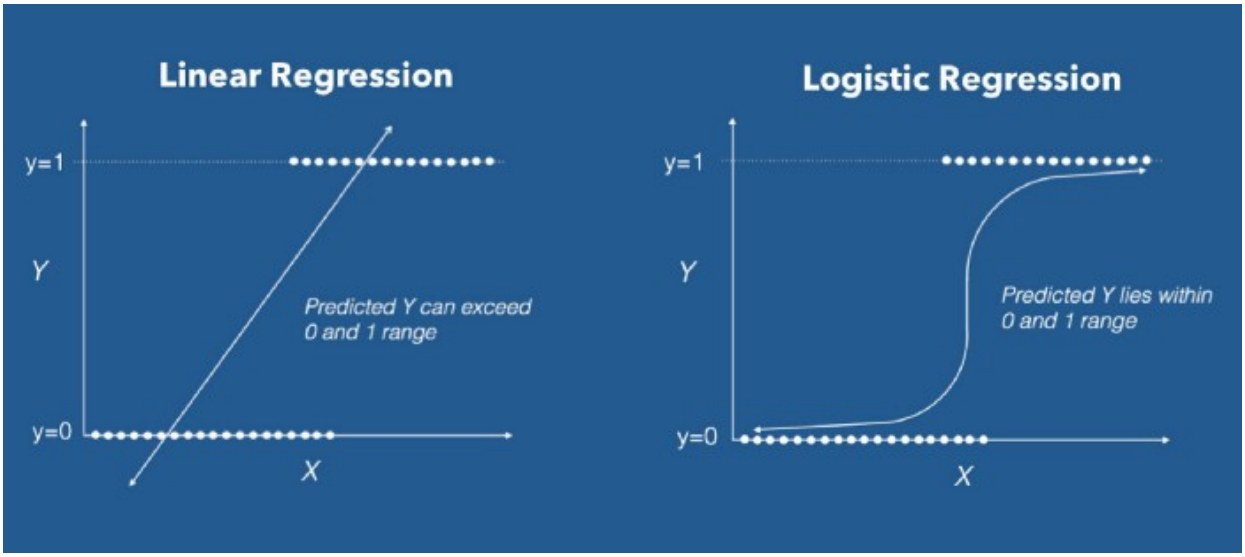
Logistic regression

Its is a classification algorithm used to assign observations to a discrete set of classes. Some of the examples of classification problems are Email spam or not spam, Online transactions Fraud or not Fraud, Tumor Malignant or Benign. Logistic regression transforms its output using the logistic sigmoid function to return a probability value. What are the types of logistic regression

> 1. Binary (eg. Tumor Malignant or Benign)
> 2. Multi-linear functions failsClass (eg. Cats, dogs or Sheep's)

Logistic Regression

Logistic Regression is a Machine Learning algorithm which is used for the classification problems, it is a predictive analysis algorithm and based on the concept of probability.



We can call a Logistic Regression a Linear Regression model but the Logistic Regression uses a more complex cost function, this cost function can be defined as the 'Sigmoid function' or also known as the 'logistic function' instead of a linear function.

The hypothesis of logistic regression tends it to limit the cost function between 0 and 1. Therefore linear functions fail to represent it as it can have a value greater than 1 or less than 0 which is not possible as per the hypothesis of logistic regression

What is the Sigmoid Function?

In order to map predicted values to probabilities, we use the Sigmoid function. The function maps any real value into another value between 0 and 1. In machine learning, we use sigmoid to map predictions to probabilities.

In [ ]:
```
import matplotlib.pyplot as plt
import numpy as np
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix
```

In [ ]:
```
x = np.arange(10).reshape(-1, 1)
y = np.array([0, 0, 0, 0, 1, 1, 1, 1, 1, 1])
```

In [ ]:
```
model = LogisticRegression(solver='liblinear', random_state=0)
```

In [ ]:
```
model.fit(x, y)
```

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=0, solver='liblinear',
          tol=0.0001, verbose=0, warm_start=False)
```

In [ ]:
```
model = LogisticRegression(solver='liblinear', random_state=0).fit(x, y)
```

In [ ]:
```
model.classes_
```

```
array([0, 1])
```

In [ ]:
```
print(model.intercept_)

print(model.coef_)
```

```
[-1.04608067]
[[0.51491375]]
```

In [ ]:
```
model.predict_proba(x)
```

```
array([[0.74002157, 0.25997843],
       [0.62975524, 0.37024476],
       [0.5040632 , 0.4959368 ],
       [0.37785549, 0.62214451],
       [0.26628093, 0.73371907],
       [0.17821501, 0.82178499],
       [0.11472079, 0.88527921],
       [0.07186982, 0.92813018],
       [0.04422513, 0.95577487],
       [0.02690569, 0.97309431]])
```

In [ ]:
```
model.predict(x)
```

```
array([0, 0, 0, 1, 1, 1, 1, 1, 1, 1])
```

In [ ]:
```
model.score(x, y)
```

```
0.9
```

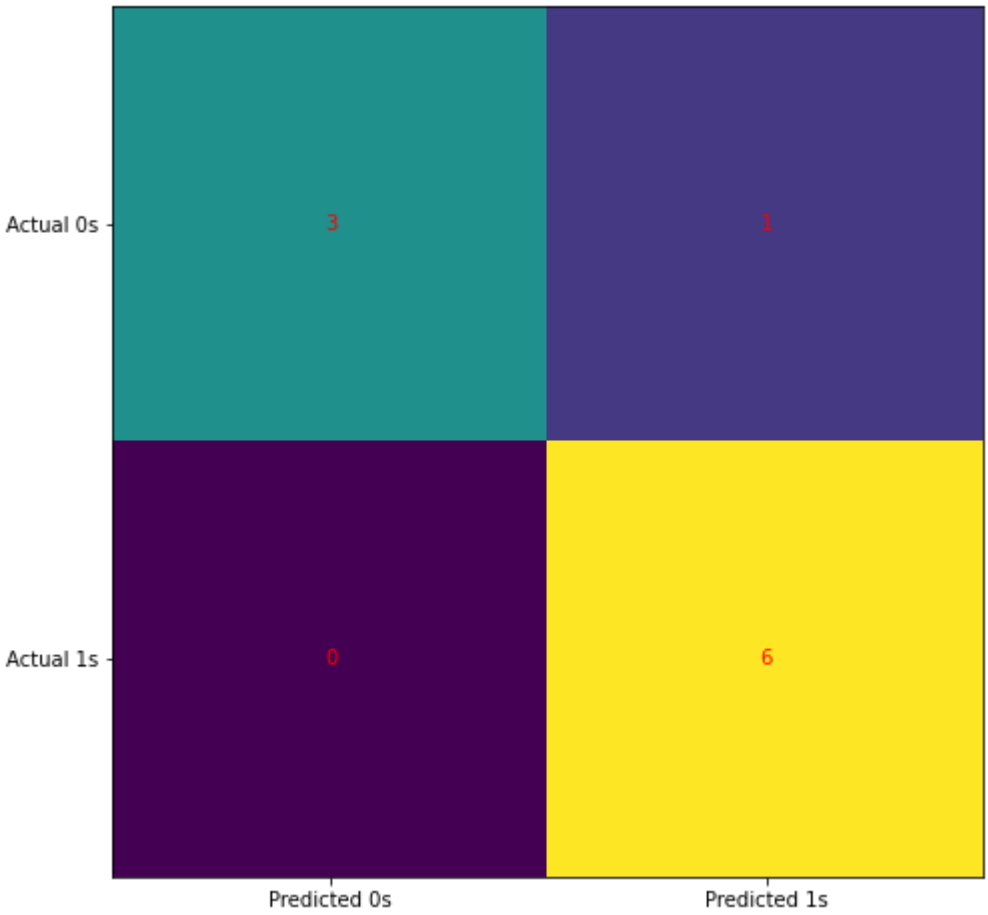
In [ ]:
```
confusion_matrix(y, model.predict(x))
```

```
array([[3, 1],
```

```
       [0, 6]], dtype=int64)
```

In [ ]:

```
cm = confusion_matrix(y, model.predict(x))

fig, ax = plt.subplots(figsize=(8, 8))
ax.imshow(cm)
ax.grid(False)
ax.xaxis.set(ticks=(0, 1), ticklabels=('Predicted 0s', 'Predicted 1s'))
ax.yaxis.set(ticks=(0, 1), ticklabels=('Actual 0s', 'Actual 1s'))
ax.set_ylim(1.5, -0.5)
for i in range(2):
    for j in range(2):
        ax.text(j, i, cm[i, j], ha='center', va='center', color='red')
plt.show()
```



In [ ]:

```
print(classification_report(y, model.predict(x)))
```

```
              precision    recall  f1-score   support

           0       1.00      0.75      0.86         4
           1       0.86      1.00      0.92         6

   micro avg       0.90      0.90      0.90        10
   macro avg       0.93      0.88      0.89        10
weighted avg       0.91      0.90      0.90        10
```

Improve the Model

In [ ]:
```
model = LogisticRegression(solver='liblinear', C=10.0, random_state=0)
model.fit(x, y)
```

```
LogisticRegression(C=10.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=0, solver='liblinear',
          tol=0.0001, verbose=0, warm_start=False)
```

In [ ]:
```
print(model.intercept_)

print(model.coef_)

print(model.predict_proba(x))
print(model.predict(x))
```

```
[-3.51335372]
[[1.12066084]]
[[0.97106534 0.02893466]
 [0.9162684  0.0837316 ]
 [0.7810904  0.2189096 ]
 [0.53777071 0.46222929]
 [0.27502212 0.72497788]
 [0.11007743 0.88992257]
 [0.03876835 0.96123165]
 [0.01298011 0.98701989]
 [0.0042697  0.9957303 ]
 [0.00139621 0.99860379]]
[0 0 0 0 1 1 1 1 1 1]
```

In [ ]:
```
model.score(x, y)

confusion_matrix(y, model.predict(x))


print(classification_report(y, model.predict(x)))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         4
           1       1.00      1.00      1.00         6

   micro avg       1.00      1.00      1.00        10
   macro avg       1.00      1.00      1.00        10
weighted avg       1.00      1.00      1.00        10
```

On Chilla Data

In [ ]:
```
import matplotlib.pyplot as plt
import numpy as np
from sklearn.datasets import load_digits
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import pandas as pd
```

In [ ]:
```
df = pd.read_csv('D:/Python ka Chilla/python_chilla/data/mldata.csv')

 #split dataset in features and target variable
feature_cols = ['age', 'gender', 'weight']
X = df[feature_cols] # Features
y = df.likeness # Target variable
```

In [ ]:
```
x_train, x_test, y_train, y_test =\
    train_test_split(X, y, test_size=0.2, random_state=0)
```

In [ ]:
```
# import the class
from sklearn.linear_model import LogisticRegression

# instantiate the model (using the default parameters)
logreg = LogisticRegression()

# fit the model with data
logreg.fit(x_train,y_train)

#
y_pred=logreg.predict(x_test)
```

```
C:\Users\Ali\anaconda3\envs\python-chilla\lib\site-packages\sklearn\linear_model\logisti
c.py:433: FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a so
lver to silence this warning.
  FutureWarning)
C:\Users\Ali\anaconda3\envs\python-chilla\lib\site-packages\sklearn\linear_model\logisti
c.py:460: FutureWarning: Default multi_class will be changed to 'auto' in 0.22. Specify
the multi_class option to silence this warning.
  "this warning.", FutureWarning)
```

In [ ]:
```
# import the metrics class
from sklearn import metrics
cnf_matrix = metrics.confusion_matrix(y_test, y_pred)
cnf_matrix
```

```
array([[32,  0,  0],
       [ 9,  0,  0],
       [ 8,  0,  0]], dtype=int64)
```

In [ ]:
```
LogisticRegression(C=0.05, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, max_iter=100,
                   multi_class='ovr', n_jobs=None, penalty='l2', random_state=0,
                   solver='liblinear', tol=0.0001, verbose=0, warm_start=False)
```

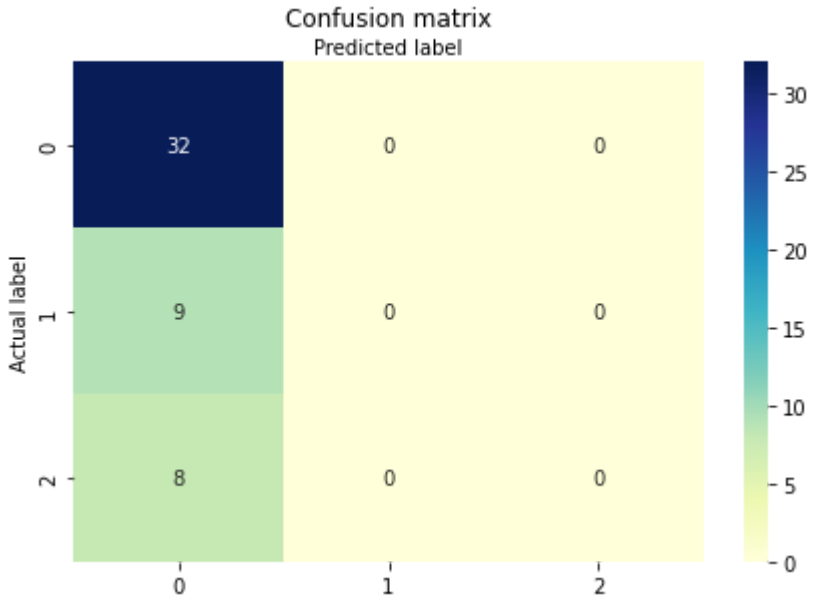
```
LogisticRegression(C=0.05, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr',
          n_jobs=None, penalty='l2', random_state=0, solver='liblinear',
          tol=0.0001, verbose=0, warm_start=False)
```

In [ ]:
```
confusion_matrix(y_test, y_pred)
```

```
array([[32,  0,  0],
       [ 9,  0,  0],
       [ 8,  0,  0]], dtype=int64)
```

In [ ]:
```
import seaborn as sns
class_names=[0,1] # name  of classes
fig, ax = plt.subplots()
tick_marks = np.arange(len(class_names))
plt.xticks(tick_marks, class_names)
plt.yticks(tick_marks, class_names)
# create heatmap
sns.heatmap(pd.DataFrame(cnf_matrix), annot=True, cmap="YlGnBu" ,fmt='g')
ax.xaxis.set_label_position("top")
plt.tight_layout()
plt.title('Confusion matrix', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
```

```
Text(0.5, 257.44, 'Predicted label')
```



In [ ]:
```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

     Biryani       0.65      1.00      0.79        32
      Pakora       0.00      0.00      0.00         9
      Samosa       0.00      0.00      0.00         8

   micro avg       0.65      0.65      0.65        49
   macro avg       0.22      0.33      0.26        49
weighted avg       0.43      0.65      0.52        49

C:\Users\Ali\anaconda3\envs\python-chilla\lib\site-packages\sklearn\metrics\classificati
on.py:1143: UndefinedMetricWarning: Precision and F-score are ill-defined and being set
to 0.0 in labels with no predicted samples.
  'precision', 'predicted', average, warn_for)
```

In [ ]:
```
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
# print("Precision:",metrics.precision_score(y_test, y_pred))
# print("Recall:",metrics.recall_score(y_test, y_pred))
```

```
Accuracy: 0.6530612244897959
```

Model is Underfit due to class imblance so that why this is biased towards biryani output

In [ ]:

In [ ]:

In this Notebook I am are going to Explore Least Squre Error

What Is the Least Squares Method?

The least-squares method is a form of mathematical regression analysis used to determine the line of best fit for a set of data, providing a visual demonstration of the relationship between the data points. Each point of data represents the relationship between a known independent variable and an unknown dependent variable.

What is the Least Squares Regression Method?

The least-squares regression method is a technique commonly used in Regression Analysis. It is a mathematical method used to find the best fit line that represents the relationship between an independent and dependent variable.

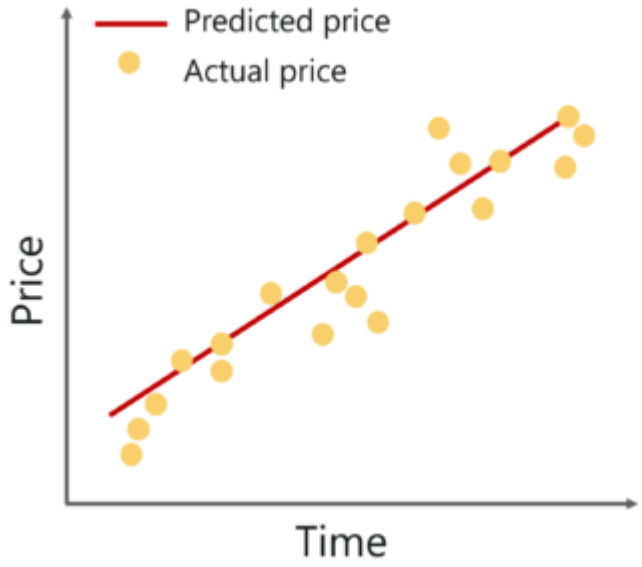
To understand the least-squares regression method lets get familiar with the concepts involved in formulating the line of best fit.

What is the Line Of Best Fit?

Line of best fit is drawn to represent the relationship between 2 or more variables. To be more specific, the best fit line is drawn across a scatter plot of data points in order to represent a relationship between those data points.

Regression analysis makes use of mathematical methods such as least squares to obtain a definite relationship between the predictor variable (s) and the target variable. The least-squares method is one of the most effective ways used to draw the line of best fit. It is based on the idea that the square of the errors obtained must be minimized to the most possible extent and hence the name least squares method.

If we were to plot the best fit line that shows the depicts the sales of a company over a period of time, it would look something like this:



In [ ]:
```
# import all the lib
import pandas as pd
import seaborn as sns
import numpy as np
import sklearn.metrics as sm
import matplotlib.pyplot as plt
```

In [ ]:
```
# Reading Data
data = pd.read_csv('D:/Python ka Chilla/python_chilla/data/ml_data_salary.csv')
# data = pd.read_csv('D:/Python ka Chilla/python_chilla/data/mldata.csv')


print(data.shape)
data.head()
```

```
(30, 4)
```

	age	distance	YearsExperience	Salary
0	31.1	77.75	1.1	39343
1	31.3	78.25	1.3	46205
2	31.5	78.75	1.5	37731
3	32.0	80.00	2.0	43525
4	32.2	80.50	2.2	39891

In [ ]:
```
 #split dataset in features and target variable
# data['gender'] = data['gender'].replace('Male',  1)
# data['gender'] = data['gender'].replace('Female',  0)
feature_cols = ['age', ]#'gender', 'weight'
X = data[feature_cols].values # Features
Y = data.YearsExperience.values # Target variable
```

In [ ]:
```
# Mean X and Y
```

```
mean_x = np.mean(X)
mean_y = np.mean(Y)

# Total number of values
n = len(X)
```

In [ ]:
```
print(mean_x)
print(mean_y)
print(n)
```

```
35.31333333333334
5.3133333333333335
30
```

Calculate the values of the slope and y-intercept

In [ ]:
```
# Using the formula to calculate 'm' and 'c'
numer = 0
denom = 0
for i in range(n):
    numer = (X[i] - mean_x) * (Y[i] - mean_y)
    numer = numer + 1
    denom = (X[i] - mean_x) ** 2
    denom = denom + 1

m = numer / denom
c = mean_y - (m * mean_x)

# Printing coefficients
print("Coefficients")
print(m, c)
```

```
Coefficients
[1.] [-30.]
```

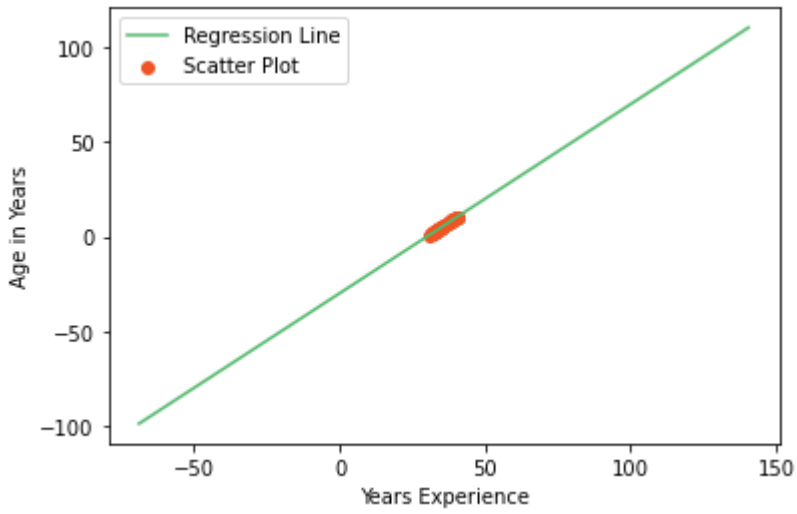
Plotting the line of best fit

In [ ]:
```
print(X.shape)
print(Y.shape)
```

```
(30, 1)
(30,)
```

In [ ]:
```
# Plotting Values and Regression Line

max_x = np.max(X) + 100
min_x = np.min(X) - 100

# Calculating line values x and y
x = np.linspace(min_x, max_x, 1000)
y = c + m * x

# Ploting Line
plt.plot(x, y, color='#58b970', label='Regression Line')
# Ploting Scatter Points
```

```
plt.scatter(X, Y, c='#ef5423', label='Scatter Plot')

plt.xlabel('Years Experience')
plt.ylabel('Age in Years')
plt.legend()
plt.show()
```



In [ ]:

```
# Calculating Root Mean Squares Error
rmse = 0
for i in range(n):
    y_pred = c + m * X[i]
    rmse += (Y[i] - y_pred) ** 2
rmse = np.sqrt(rmse/n)
print("RMSE")
print(rmse)
```

```
RMSE
[1.13849929e-14]
```

Calculating R2 Score

In [ ]:

```
# Calculating R2 Score
ss_tot = 0
ss_res = 0
for i in range(n):
    y_pred = c + m * X[i]
    ss_tot += (Y[i] - mean_y) ** 2
    ss_res += (Y[i] - y_pred) ** 2
r2 = 1 - (ss_res/ss_tot)
print("R2 Score")
print(r2)
```

```
R2 Score
[1.]
```

In this Notebook I am are going to Explore Image classifcation using Machine Learning with Python

In this Notebook I am are going to Explore Actication Function

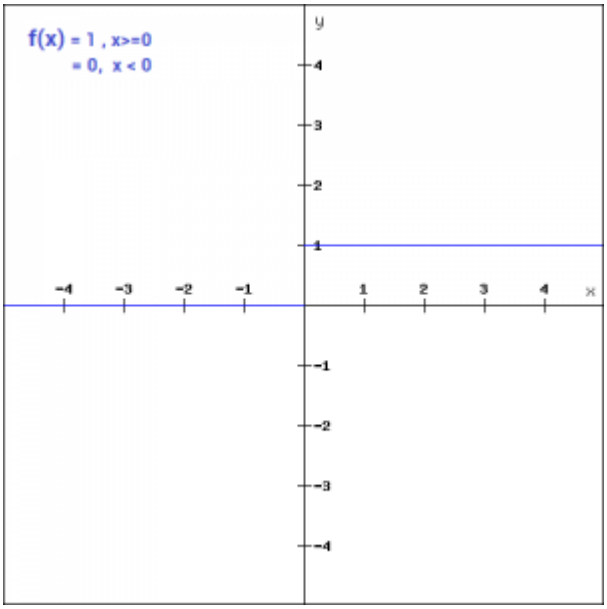
used in Machine Learning

1. Binary Step Function

The first thing that comes to our mind when we have an activation function would be a threshold based classifier i.e. whether or not the neuron should be activated based on the value from the linear transformation.

In other words, if the input to the activation function is greater than a threshold, then the neuron is activated, else it is deactivated, i.e. its output is not considered for the next hidden layer. Let us look at it mathematically-

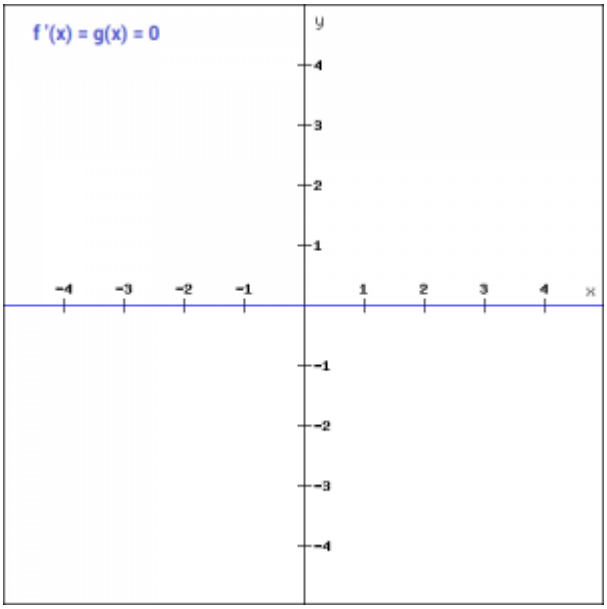
```
f(x) = 1, x>=0
     = 0, x<0
```



This is the simplest activation function, which can be implemented with a single if-else condition in python

```python
def binary_step(x):
    if x<0:
        return 0
    else:
        return 1
```

`python binary_step(5), binary_step(-1)

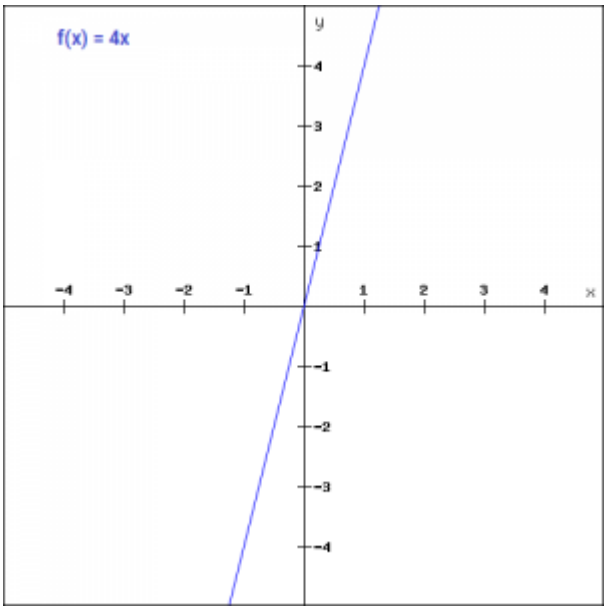


Gradients are calculated to update the weights and biases during the backprop process. Since the gradient of the function is zero, the weights and biases don't update.

2. Linear Function

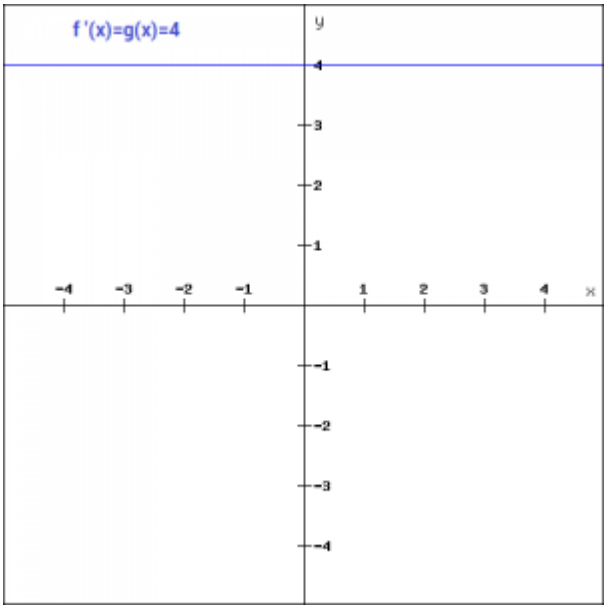
We saw the problem with the step function, the gradient of the function became zero. This is because there is no component of x in the binary step function. Instead of a binary function, we can use a linear function. We can define the function as-

$f(x)=ax$



Here the activation is proportional to the input.The variable 'a' in this case can be any constant value. Let's quickly define the function in python:

```
def linear_function(x):
    return 4*x
linear_function(4), linear_function(-2)
```



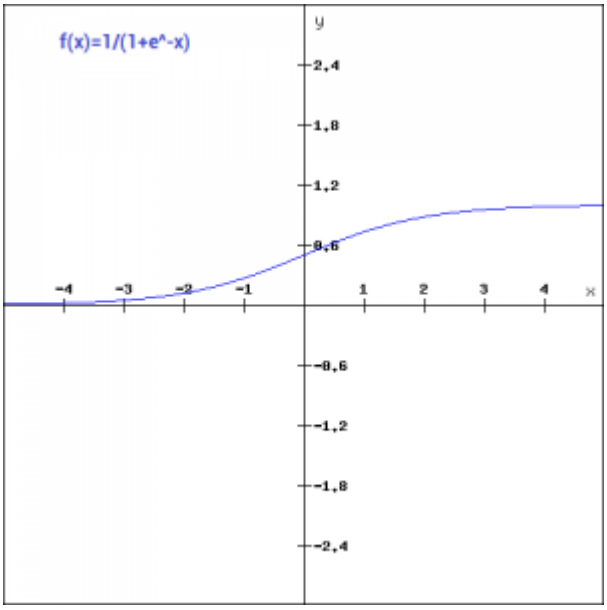
Although the gradient here does not become zero, but it is a constant which does not depend upon the input value x at all. This implies that the weights and biases will be updated during the backpropagation process but the updating factor would be the same.

In this scenario, the neural network will not really improve the error since the gradient is the same for every iteration. The network will not be able to train well and capture the complex patterns from the data. Hence, linear function might be ideal for simple tasks where interpretability is highly desired.

3. Sigmoid

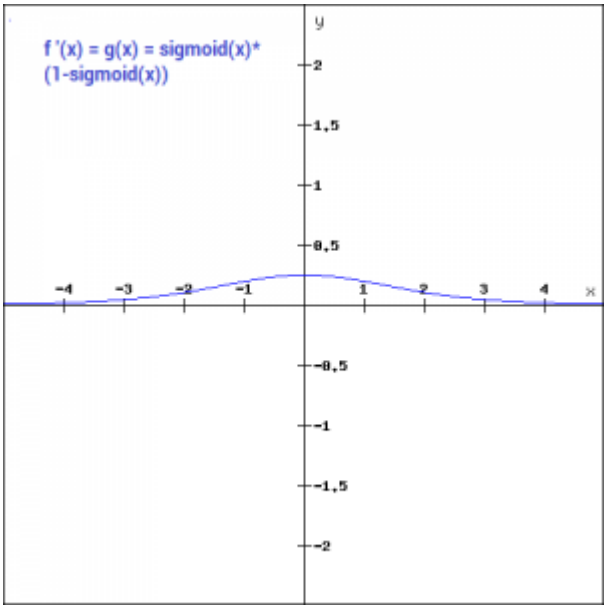
The next activation function that we are going to look at is the Sigmoid function. It is one of the most widely used non-linear activation function. Sigmoid transforms the values between the range 0 and 1. Here is the mathematical expression for sigmoid-

```
f(x) = 1/(1+e^-x)
```



A noteworthy point here is that unlike the binary step and linear functions, sigmoid is a non-linear function. This essentially means -when I have multiple neurons having sigmoid function as their activation function,the output is non linear as well. Here is the python code for defining the function in python-

```
import numpy as np
def sigmoid_function(x):
    z = (1/(1 + np.exp(-x)))
    return z
sigmoid_function(7),sigmoid_function(-22)
```



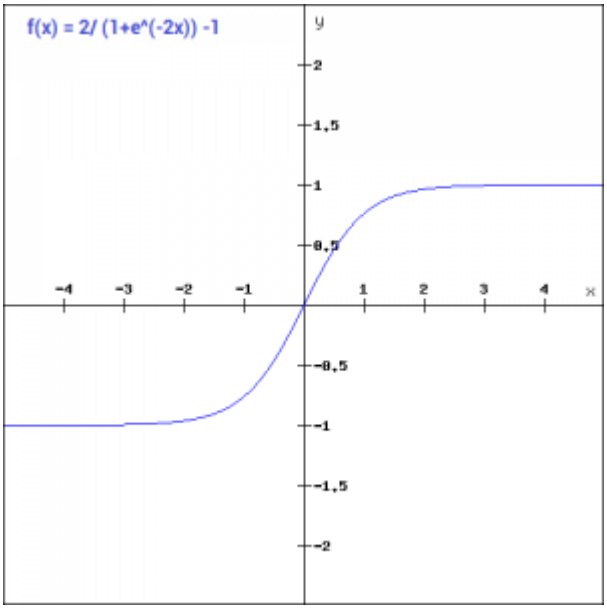
The gradient values are significant for range -3 and 3 but the graph gets much flatter in other regions. This implies that for values greater than 3 or less than -3, will have very small gradients. As the gradient value approaches zero, the network is not really learning.

Additionally, the sigmoid function is not symmetric around zero. So output of all the neurons will be of the same sign. This can be addressed by scaling the sigmoid function which is exactly what happens in the tanh function. Let's read on.

4. Tanh

The tanh function is very similar to the sigmoid function. The only difference is that it is symmetric around the origin. The range of values in this case is from -1 to 1. Thus the inputs to the next layers will not always be of the same sign. The tanh function is defined as-

```
tanh(x)=2sigmoid(2x)-1
```

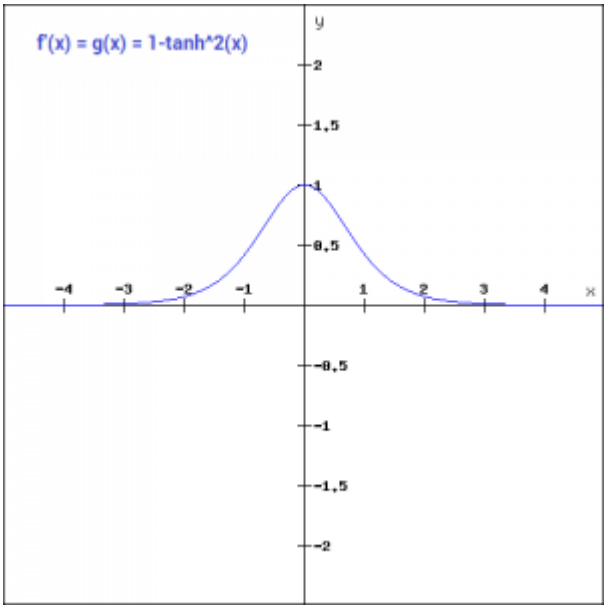


In order to code this is python, let us simplify the previous expression.

`tanh(x) = 2sigmoid(2x)-1 \ tanh(x) = 2/(1+e^(-2x)) -1` And here is the python code for the same:

```
def tanh_function(x):
    z = (2/(1 + np.exp(-2*x))) -1
    return z
tanh_function(0.5), tanh_function(-1)
```

the range of values is between -1 to 1. Apart from that, all other properties of tanh function are the same as that of the sigmoid function. Similar to sigmoid, the tanh function is continuous and differentiable at all points.



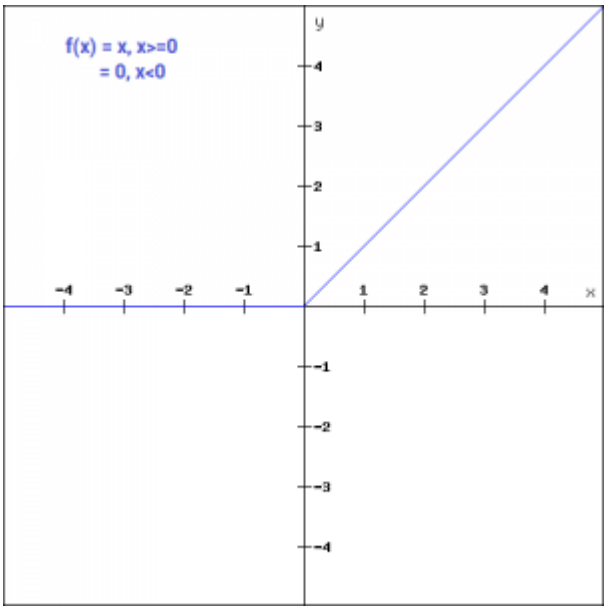
5. ReLU

The ReLU function is another non-linear activation function that has gained popularity in the deep learning domain. ReLU stands for Rectified Linear Unit. The main advantage of using the ReLU

function over other activation functions is that it does not activate all the neurons at the same time.

This means that the neurons will only be deactivated if the output of the linear transformation is less than 0. The plot below will help you understand this better-

```
f(x)=max(0,x)
```



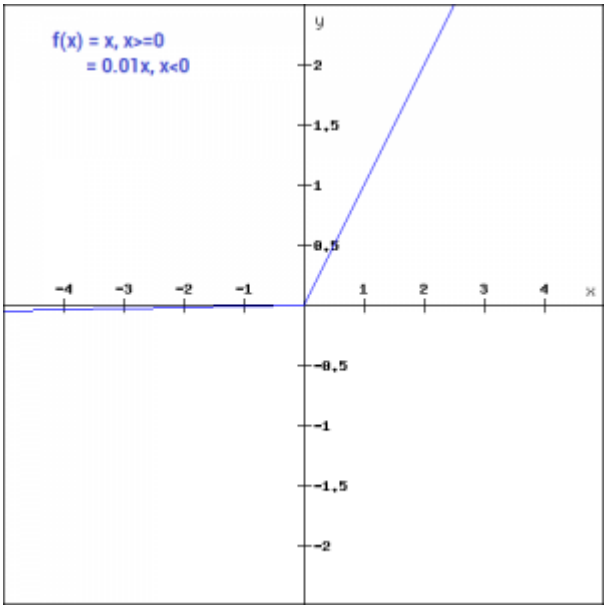
For the negative input values, the result is zero, that means the neuron does not get activated. Since only a certain number of neurons are activated, the ReLU function is far more computationally efficient when compared to the sigmoid and tanh function. Here is the python function for ReLU:

```
def relu_function(x):
    if x<0:
        return 0
    else:
        return x
relu_function(7), relu_function(-7)
```

6. Leaky ReLU

Leaky ReLU function is nothing but an improved version of the ReLU function. As we saw that for the ReLU function, the gradient is 0 for x<0, which would deactivate the neurons in that region.

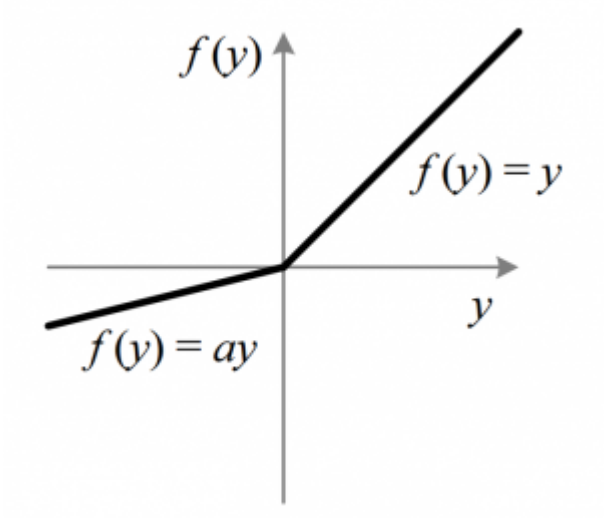
Leaky ReLU is defined to address this problem. Instead of defining the Relu function as 0 for negative values of x, we define it as an extremely small linear component of x. Here is the mathematical expression



By making this small modification, the gradient of the left side of the graph comes out to be a non zero value. Hence we would no longer encounter dead neurons in that region. Here is the derivative of the Leaky ReLU function

7. Parameterised ReLU

This is another variant of ReLU that aims to solve the problem of gradient's becoming zero for the left half of the axis. The parameterised ReLU, as the name suggests, introduces a new parameter as a slope of the negative part of the function. Here's how the ReLU function is modified to incorporate the slope parameter-



When the value of a is fixed to 0.01, the function acts as a Leaky ReLU function. However, in case of a parameterised ReLU function, 'a' is also a trainable parameter. The network also learns the value of 'a' for faster and more optimum convergence.

8. Softmax

Softmax function is often described as a combination of multiple sigmoids. We know that sigmoid returns values between 0 and 1, which can be treated as probabilities of a data point belonging to a particular class. Thus sigmoid is widely used for binary classification problems.

The softmax function can be used for multiclass classification problems. This function returns the probability for a datapoint belonging to each individual class. Here is the mathematical expression of the same-

While building a network for a multiclass problem, the output layer would have as many neurons as the number of classes in the target. For instance if you have three classes, there would be three neurons in the output layer. Suppose you got the output from the neurons as [1.2 , 0.9 , 0.75].

Applying the softmax function over these values, you will get the following result – [0.42 , 0.31, 0.27]. These represent the probability for the data point belonging to each class. Note that the sum of all the values is 1. Let us code this in python

```
def softmax_function(x):
    z = np.exp(x)
    z_ = z/z.sum()
    return z_
softmax_function([0.8, 1.2, 3.1])
```

Choosing the right Activation Function

Now that we have seen so many activation functions, we need some logic / heuristics to know which activation function should be used in which situation. Good or bad – there is no rule of thumb.

However depending upon the properties of the problem we might be able to make a better choice for easy and quicker convergence of the network.

- Sigmoid functions and their combinations generally work better in the case of classifiers
- Sigmoids and tanh functions are sometimes avoided due to the vanishing gradient problem
- ReLU function is a general activation function and is used in most cases these days
- If we encounter a case of dead neurons in our networks the leaky ReLU function is the best choice
- Always keep in mind that ReLU function should only be used in the hidden layers
- As a rule of thumb, you can begin with using ReLU function and then move over to other activation functions in case ReLU doesn't provide with optimum results

In this Notebook I am are going to Explore Image classifcation using Machine Learning with Python

What is Pixel:

A pixel is a contraction if the term PIcture ELement. Digital images are made up of small squares, just like a tile mosaic on a wall. Though a digital photograph looks smooth and continuous just like

a regular photograph, it's actually composed of millions of tiny squares

What exactly is a pixel?

The pixel (a word invented from "picture element") is the basic unit of programmable color on a computer display or in a computer image. Think of it as a logical - rather than a physical - unit. The physical size of a pixel depends on how you've set the resolution for the display screen

Types of Images:

The binary image

The binary image as it name states, contain only two pixel values.

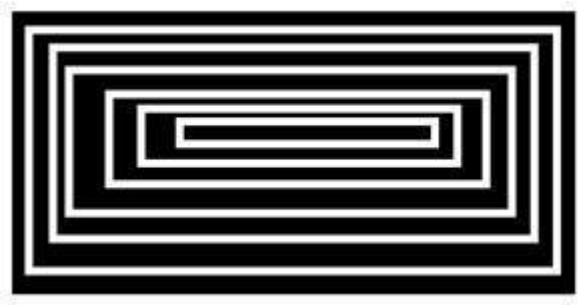
0 and 1.

In our previous tutorial of bits per pixel, we have explained this in detail about the representation of pixel values to their respective colors.

Here 0 refers to black color and 1 refers to white color. It is also known as Monochrome.

Black and white image:

The resulting image that is formed hence consist of only black and white color and thus can also be called as Black and White image.



black and white

No gray level One of the interesting this about this binary image that there is no gray level in it. Only two colors that are black and white are found in it.

Format Binary images have a format of PBM (Portable bit map)

2, 3, 4,5, 6 bit color format The images with a color format of 2, 3, 4, 5 and 6 bit are not widely used today. They were used in old times for old TV displays, or monitor displays.

But each of these colors have more then two gray levels, and hence has gray color unlike the binary image.

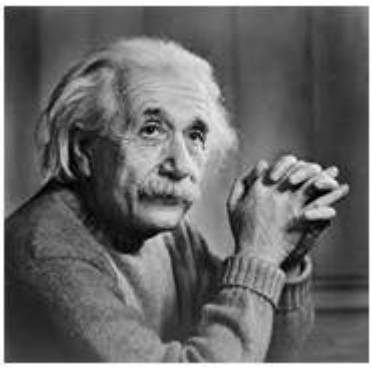
In a 2 bit 4, in a 3 bit 8, in a 4 bit 16, in a 5 bit 32, in a 6 bit 64 different colors are present.

8 bit color format

8 bit color format is one of the most famous image format. It has 256 different shades of colors in it. It is commonly known as Grayscale image.

The range of the colors in 8 bit vary from 0-255. Where 0 stands for black, and 255 stands for white, and 127 stands for gray color.

This format was used initially by early models of the operating systems UNIX and the early color Macintoshes.

A grayscale image of Einstein is shown below: 

einstein Format The format of these images are PGM (Portable Gray Map).

This format is not supported by default from windows. In order to see gray scale image, you need to have an image viewer or image processing toolbox such as Matlab.

Behind gray scale image: As we have explained it several times in the previous tutorials, that an image is nothing but a two dimensional function, and can be represented by a two dimensional array or matrix. So in the case of the image of Einstein shown above, there would be two dimensional matrix in behind with values ranging between 0 and 255.

But thats not the case with the color images.

16 bit color format

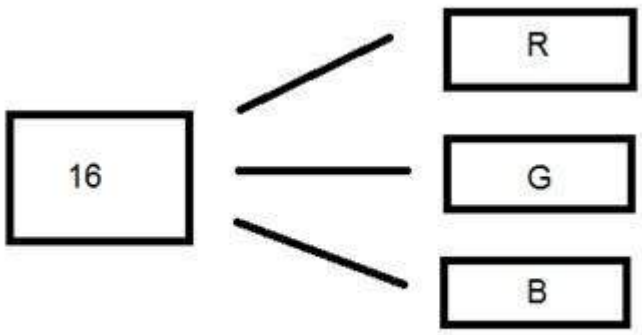
It is a color image format. It has 65,536 different colors in it. It is also known as High color format.

It has been used by Microsoft in their systems that support more then 8 bit color format. Now in this 16 bit format and the next format we are going to discuss which is a 24 bit format are both color format.

The distribution of color in a color image is not as simple as it was in grayscale image.

A 16 bit format is actually divided into three further formats which are Red , Green and Blue. The famous (RGB) format.

It is pictorially represented in the image below.



16-bit Now the question arises, that how would you distribute 16 into three. If you do it like this,

5 bits for R, 5 bits for G, 5 bits for B

Then there is one bit remains in the end.

So the distribution of 16 bit has been done like this.

5 bits for R, 6 bits for G, 5 bits for B.

The additional bit that was left behind is added into the green bit. Because green is the color which is most soothing to eyes in all of these three colors.

Note this is distribution is not followed by all the systems. Some have introduced an alpha channel in the 16 bit.

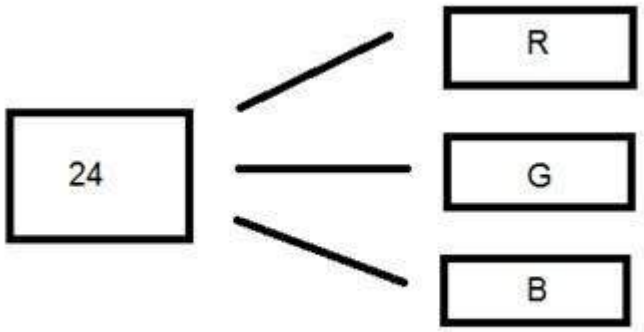
Another distribution of 16 bit format is like this: 4 bits for R, 4 bits for G, 4 bits for B, 4 bits for alpha channel.

Or some distribute it like this

5 bits for R, 5 bits for G, 5 bits for B, 1 bits for alpha channel.

24 bit color format

24 bit color format also known as true color format. Like 16 bit color format, in a 24 bit color format, the 24 bits are again distributed in three different formats of Red, Green and Blue.



24-bit Since 24 is equally divided on 8, so it has been distributed equally between three different color channels.

Their distribution is like this.

8 bits for R, 8 bits for G, 8 bits for B.

Behind a 24 bit image.

Unlike a 8 bit gray scale image, which has one matrix behind it, a 24 bit image has three different matrices of R, G, B. 

what is image

Format It is the most common used format. Its format is PPM (Portable pixMap) which is supported by Linux operating system. The famous windows has its own format for it which is BMP (Bitmap).

RGB (red, green, blue):

RGB image is a three-dimensional byte array that explicitly stores a color value for each pixel. RGB image arrays are made up of width, height, and three channels of color information. Scanned photographs are commonly stored as RGB images

CMYK (Cyan, Magenta, Yellow, and Key) (black):

CMYK represent Cyan, Magenta, Yellow, and Key (black). By mixing the four colors in varying amounts, millions of others shades are produced in the printed material. These links are utilized

when printing photos in books and magazines. RGB describes the colors of images that are viewed on the monitor

Recognizing hand-written digits

In [ ]:
```
# import all the lib
import pandas as pd
import seaborn as sns
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn import linear_model
import sklearn.metrics as sm
import matplotlib.pyplot as plt

# Importing Linear Regression model from scikit learn
from sklearn.linear_model import LinearRegression
# Importing metrics for the evaluation of the model
from sklearn.metrics import r2_score,mean_squared_error
```

Data Exploration

In [ ]:
```
digits = load_digits()
digits.data.shape
```

```
(1797, 64)
```

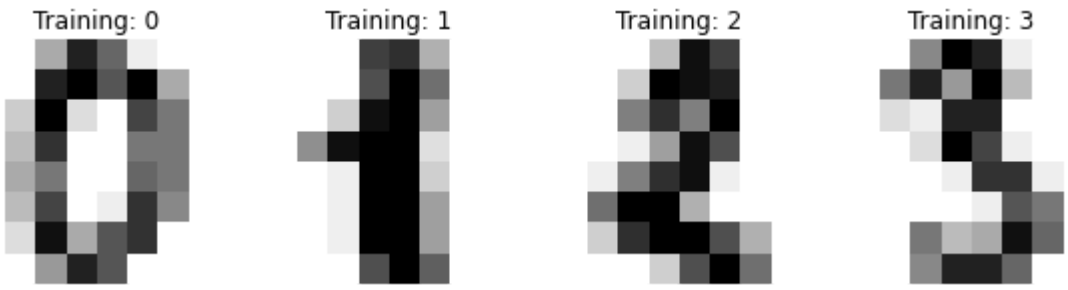
In [ ]:
```
digits.target.shape
```

```
(1797,)
```

In [ ]:
```
digits.images.shape
```

```
(1797, 8, 8)
```

In [ ]:
```
_, axes = plt.subplots(nrows=1, ncols=4, figsize=(10, 3))
for ax, image, label in zip(axes, digits.images, digits.target):
    ax.set_axis_off()
    ax.imshow(image, cmap=plt.cm.gray_r, interpolation="nearest")
    ax.set_title("Training: %i" % label)
```



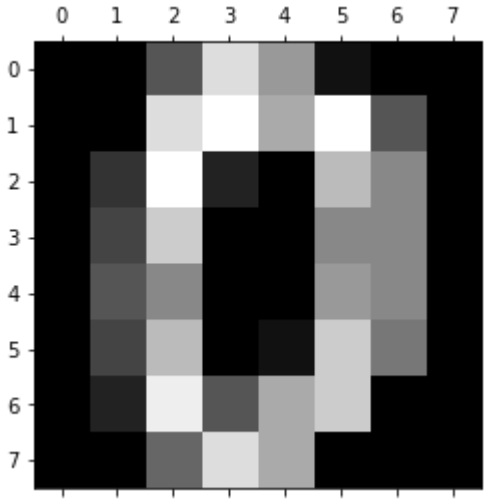
In [ ]:
```
print( digits.images[0])
```

```
[[ 0.  0.  5. 13.  9.  1.  0.  0.]
 [ 0.  0. 13. 15. 10. 15.  5.  0.]
 [ 0.  3. 15.  2.  0. 11.  8.  0.]
 [ 0.  4. 12.  0.  0.  8.  8.  0.]
 [ 0.  5.  8.  0.  0.  9.  8.  0.]
 [ 0.  4. 11.  0.  1. 12.  7.  0.]
 [ 0.  2. 14.  5. 10. 12.  0.  0.]
 [ 0.  0.  6. 13. 10.  0.  0.  0.]]
```

In [ ]:

```
plt.gray()
plt.matshow(digits.images[0])

plt.show()
```

```
<Figure size 432x288 with 0 Axes>
```



In [ ]:

```
digits.target
```

```
array([0, 1, 2, ..., 8, 9, 8])
```

In [ ]:

```
def plot_multi(i):
    '''Plots 16 digits, starting with digit i'''
    nplots = 16
    fig = plt.figure(figsize=(15,15))
    for j in range(nplots):
        plt.subplot(4,4,j+1)
        plt.imshow(digits.images[i+j], cmap='binary')
        plt.title(digits.target[i+j])
        plt.axis('off')
    plt.show()
```

In [ ]:

```
plot_multi(0)
```

Building the network and preparing the input data

In [ ]:
```
# Print to show there are 1797 images (8 by 8 images for a dimensionality of 64)
print('Image Data Shape' , digits.data.shape)
# Print to show there are 1797 labels (integers from 0-9)
print("Label Data Shape", digits.target.shape)
```

```
Image Data Shape (1797, 64)
Label Data Shape (1797,)
```

In [ ]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(digits.data, digits.target, test_si
```

In [ ]:
```
print('Input Shape of X Train',X_train.shape)
print('Input Shape of Y Train',y_train.shape)
```

```
print('Input Shape of X Test',X_test.shape)
print('Input Shape of Y Test',y_test.shape)
```

```
Input Shape of X Train (1347, 64)
Input Shape of Y Train (1347,)
Input Shape of X Test (450, 64)
Input Shape of Y Test (450,)
```

In [ ]:
```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
model.fit(X_train, y_train)
```

```
C:\Users\Ali\anaconda3\envs\python-chilla\lib\site-packages\sklearn\linear_model\logisti
c.py:433: FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a so
lver to silence this warning.
  FutureWarning)
C:\Users\Ali\anaconda3\envs\python-chilla\lib\site-packages\sklearn\linear_model\logisti
c.py:460: FutureWarning: Default multi_class will be changed to 'auto' in 0.22. Specify
the multi_class option to silence this warning.
  "this warning.", FutureWarning)
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None, solver='warn',
          tol=0.0001, verbose=0, warm_start=False)
```

In [ ]:
```
# Learn the digits on the train subset
model.fit(X_train, y_train)

# Predict the value of the digit on the test subset
predicted = model.predict(X_test)
```

```
C:\Users\Ali\anaconda3\envs\python-chilla\lib\site-packages\sklearn\linear_model\logisti
c.py:433: FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a so
lver to silence this warning.
  FutureWarning)
C:\Users\Ali\anaconda3\envs\python-chilla\lib\site-packages\sklearn\linear_model\logisti
c.py:460: FutureWarning: Default multi_class will be changed to 'auto' in 0.22. Specify
the multi_class option to silence this warning.
  "this warning.", FutureWarning)
```

In [ ]:
```
# Returns a NumPy Array
# Predict for One Observation (image)
model.predict(X_test[0].reshape(1,-1))
```

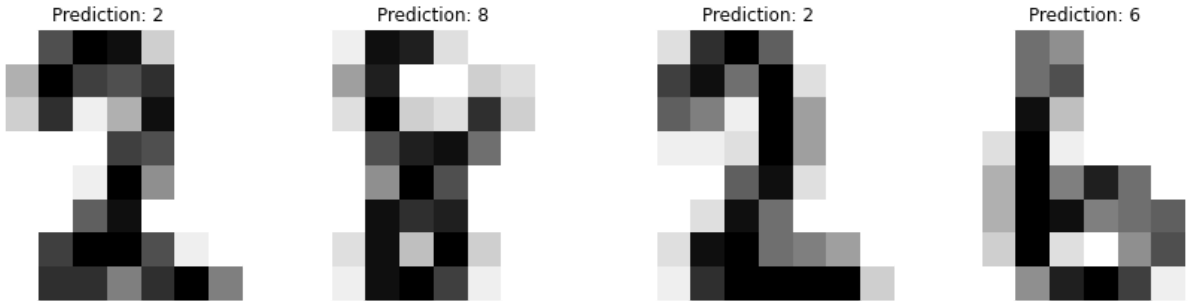
```
array([2])
```

In [ ]:
```
predictions = model.predict(X_test)
score = model.score(X_test, y_test)
# print("Prediction are:", pred)
print("Accuracy Score is :", score)
```

```
Accuracy Score is : 0.9533333333333334
```

In [ ]:
```
_, axes = plt.subplots(nrows=1, ncols=4, figsize=(15, 5))
for ax, image, prediction in zip(axes, X_test, predicted):
    ax.set_axis_off()
    image = image.reshape(8, 8)
```

```
    ax.imshow(image, cmap=plt.cm.gray_r, interpolation="nearest")
    ax.set_title(f"Prediction: {prediction}")
```



In [ ]:
```
# Returns a NumPy Array
# Predict for One Observation (image)
model.predict(X_test[0].reshape(1,-1))
```

```
array([2])
```

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

In [ ]:
```
# Use score method to get accuracy of model
score = model.score(X_test, y_test)
print(score)
```

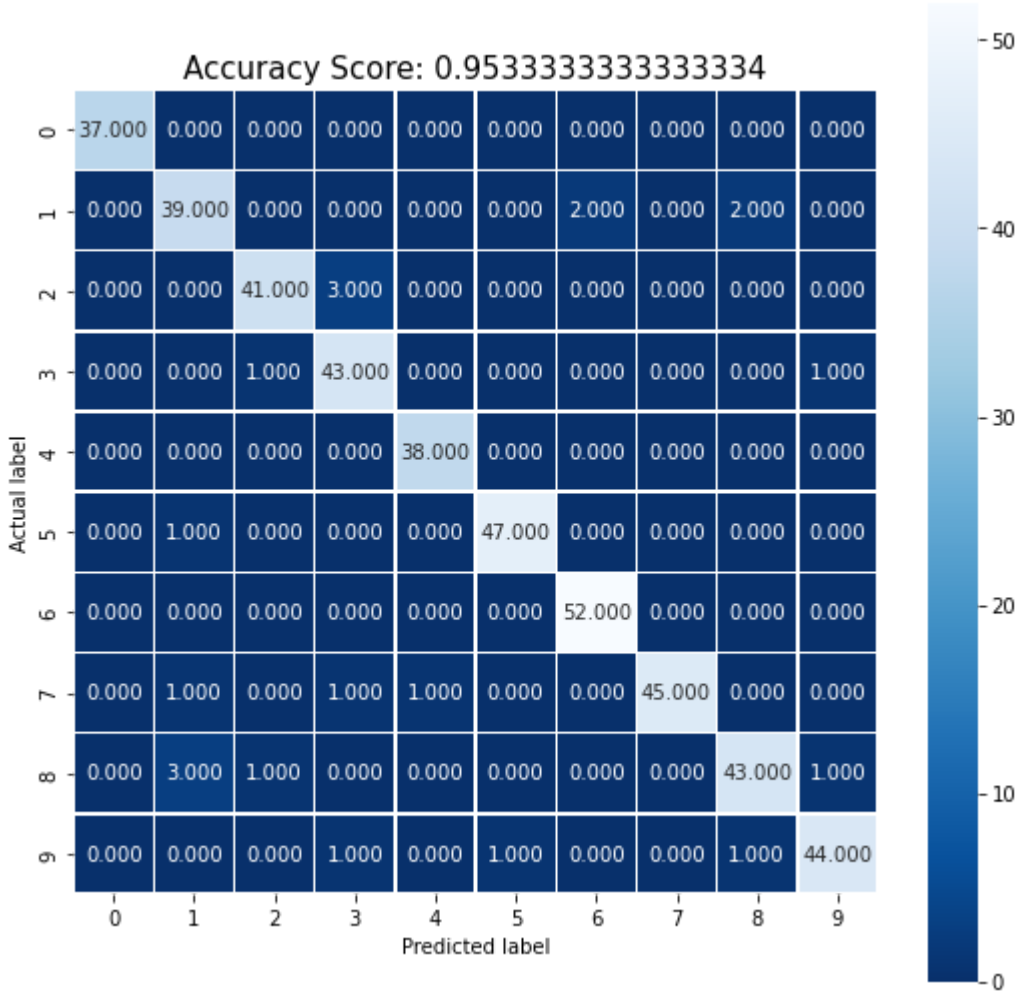
```
0.9533333333333334
```

In [ ]:
```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import metrics
```

In [ ]:
```
cm = metrics.confusion_matrix(y_test, predictions)
print(cm)
```

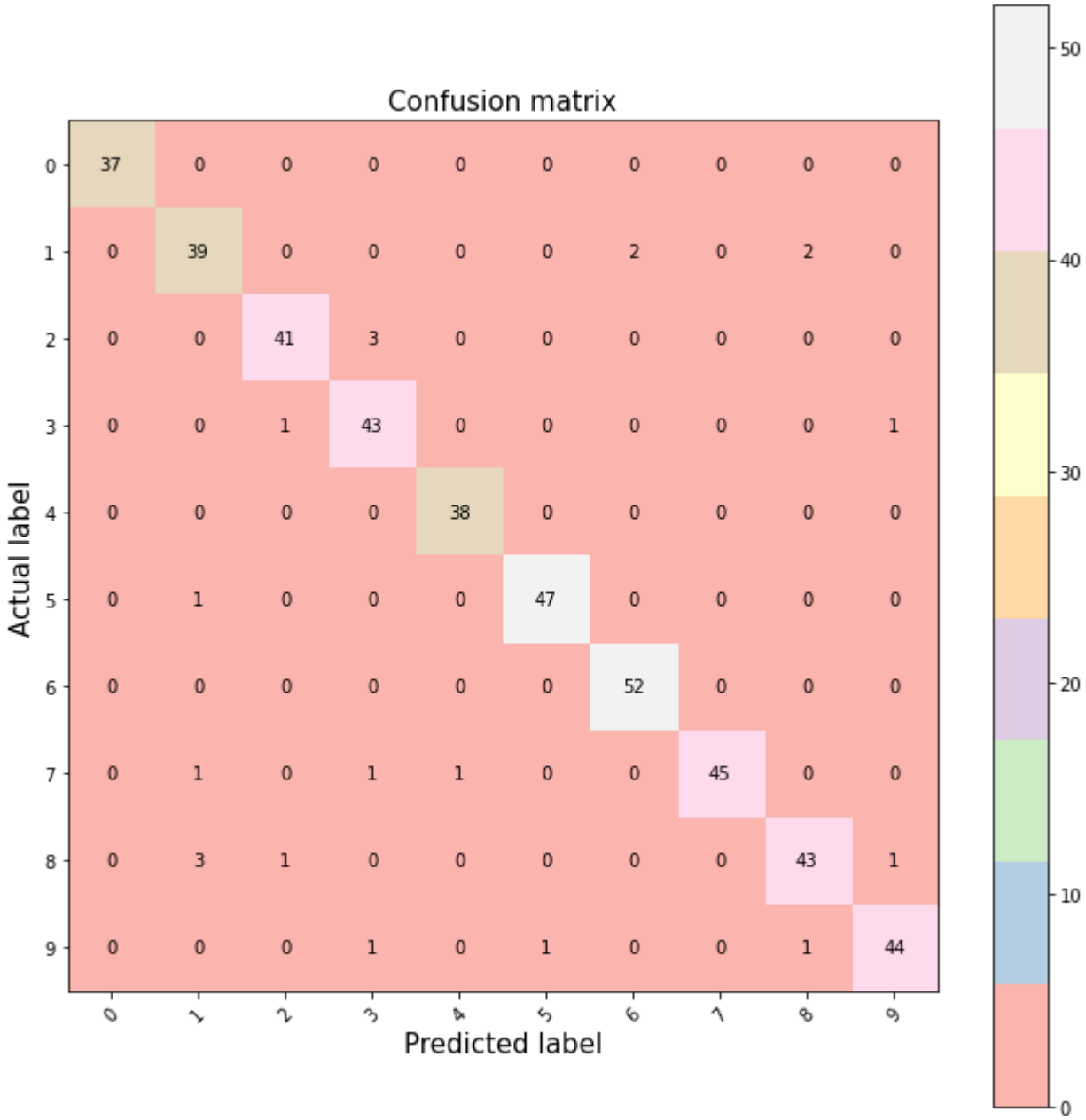
```
[[37  0  0  0  0  0  0  0  0  0]
 [ 0 39  0  0  0  0  2  0  2  0]
 [ 0  0 41  3  0  0  0  0  0  0]
 [ 0  0  1 43  0  0  0  0  0  1]
 [ 0  0  0  0 38  0  0  0  0  0]
 [ 0  1  0  0  0 47  0  0  0  0]
 [ 0  0  0  0  0  0 52  0  0  0]
 [ 0  1  0  1  1  0  0 45  0  0]
 [ 0  3  1  0  0  0  0  0 43  1]
 [ 0  0  0  1  0  1  0  0  1 44]]
```

In [ ]:
```
plt.figure(figsize=(9,9))
sns.heatmap(cm, annot=True, fmt=".3f", linewidths=.5, square = True, cmap = 'Blues_r');
plt.ylabel('Actual label');
plt.xlabel('Predicted label');
all_sample_title = 'Accuracy Score: {0}'.format(score)
plt.title(all_sample_title, size = 15);
```



In [ ]:

```
plt.figure(figsize=(9,9))
plt.imshow(cm, interpolation='nearest', cmap='Pastel1')
plt.title('Confusion matrix', size = 15)
plt.colorbar()
tick_marks = np.arange(10)
plt.xticks(tick_marks, ["0", "1", "2", "3", "4", "5", "6", "7", "8", "9"], rotation=45,
plt.yticks(tick_marks, ["0", "1", "2", "3", "4", "5", "6", "7", "8", "9"], size = 10)
plt.tight_layout()
plt.ylabel('Actual label', size = 15)
plt.xlabel('Predicted label', size = 15)
width, height = cm.shape
for x in range(width):
 for y in range(height):
  plt.annotate(str(cm[x][y]), xy=(y, x),
  horizontalalignment='center',
  verticalalignment='center')
```



Display Misclassified images with Predicted Labels Assignment

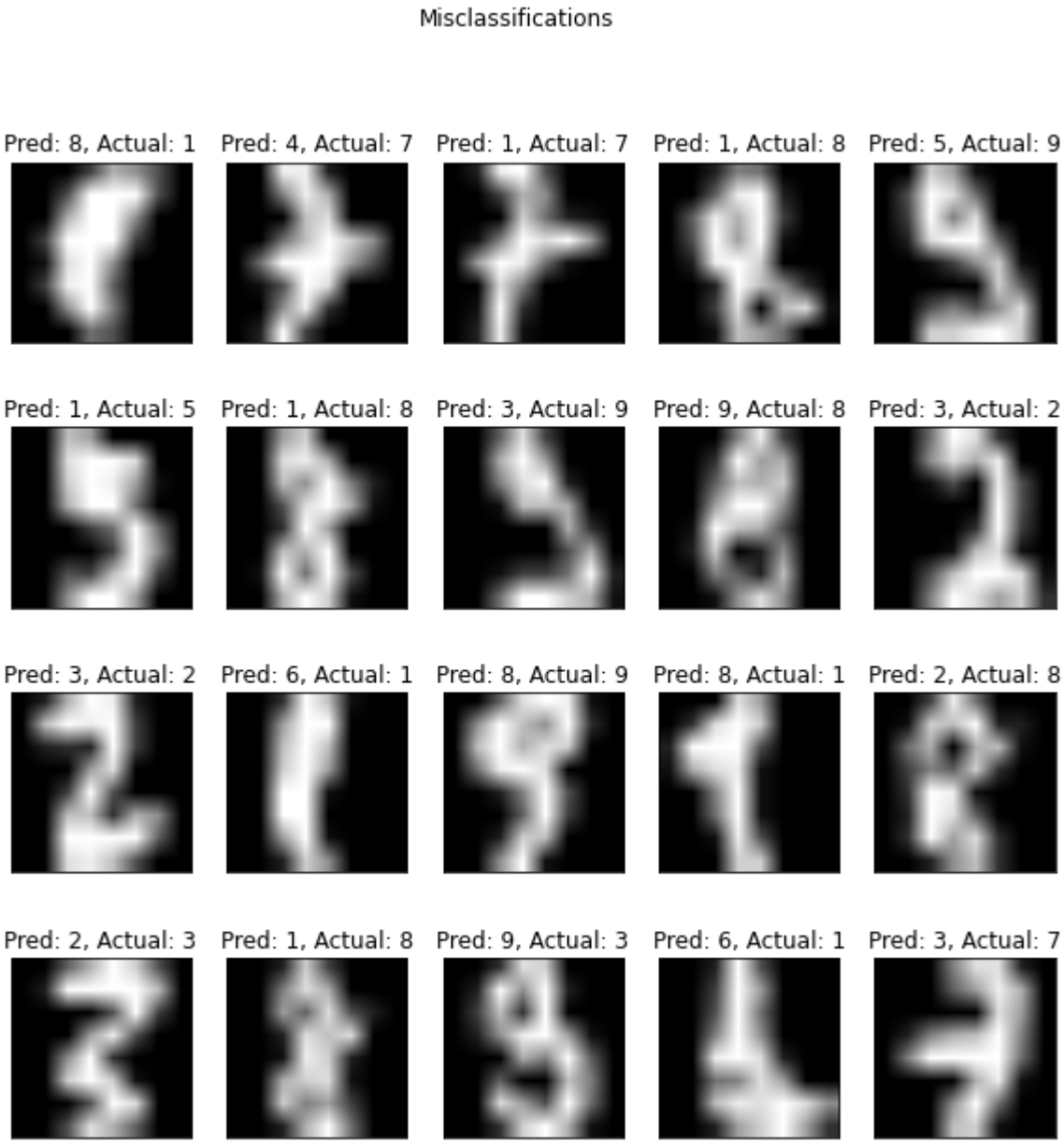
In [ ]:
```
index = 0
misclassified_images = []
for label, predict in zip(y_test, predictions):
    if label != predict:
        misclassified_images.append(index)
    index +=1
```

In [ ]:
```
print(misclassified_images)
```

```
[37, 94, 109, 124, 130, 181, 196, 211, 218, 235, 251, 301, 312, 331, 335, 378, 398, 413,
416, 425, 440]
```

In [ ]:

```
plt.figure(figsize=(10,10))
plt.suptitle('Misclassifications');

for plot_index, bad_index in enumerate(misclassified_images[0:20]):
    p = plt.subplot(4,5, plot_index+1) # 4x5 plot

    p.imshow(X_test[bad_index].reshape(8,8), cmap=plt.cm.gray,
             interpolation='bilinear')
    p.set_xticks(()); p.set_yticks(()) # remove ticks

    p.set_title(f'Pred: {predictions[bad_index]}, Actual: {y_test[bad_index]}');
```



In [ ]:

||Random forest algorthim||-

Random Forest is a **flexible**, **easy** to use machine learning algorithm that produces, even without

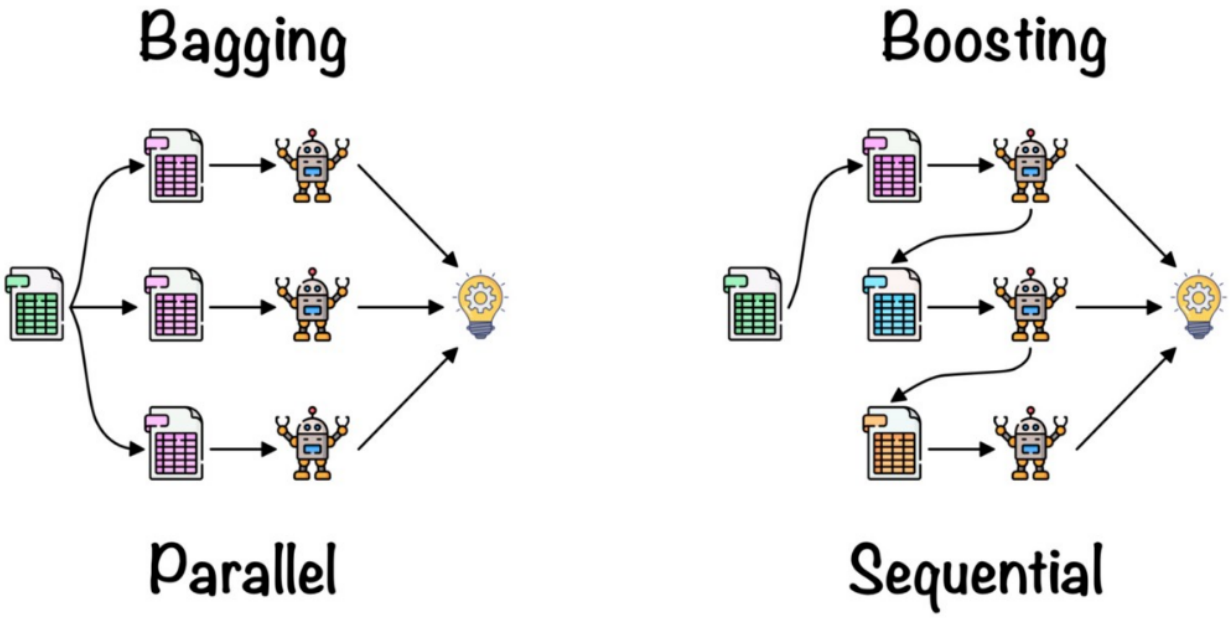
hyper-parameter tuning, a great result most of the time. It is also one of the most used algorithms, because it's simplicity and the fact that it can be used for both classification and regression tasks.

Working of Random Forest Algorithm

Before understanding the working of the random forest we must look into the ensemble technique. Ensemble simply means combining multiple models. Thus a collection of models is used to make predictions rather than an individual model.

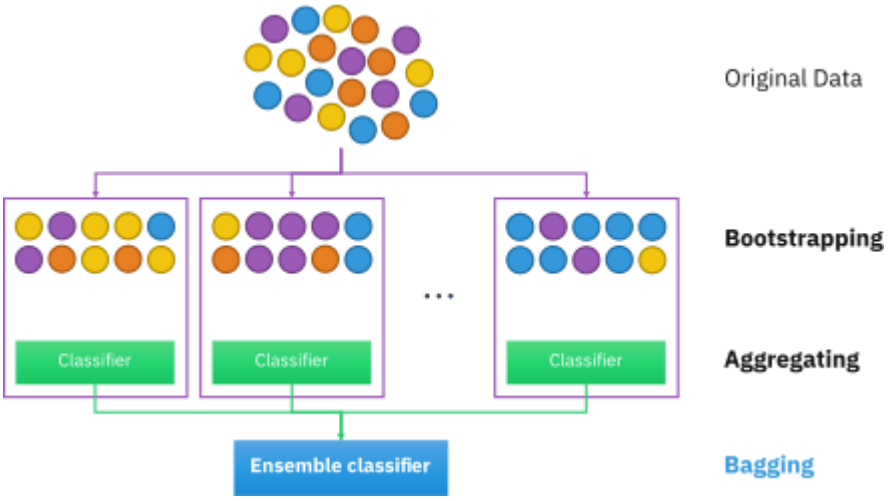
Ensemble uses two types of methods:

1. Bagging– It creates a different training subset from sample training data with replacement & the final output is based on majority voting. For example, Random Forest.
2. Boosting– It combines weak learners into strong learners by creating sequential models such that the final model has the highest accuracy. For example, ADA BOOST, XG BOOST

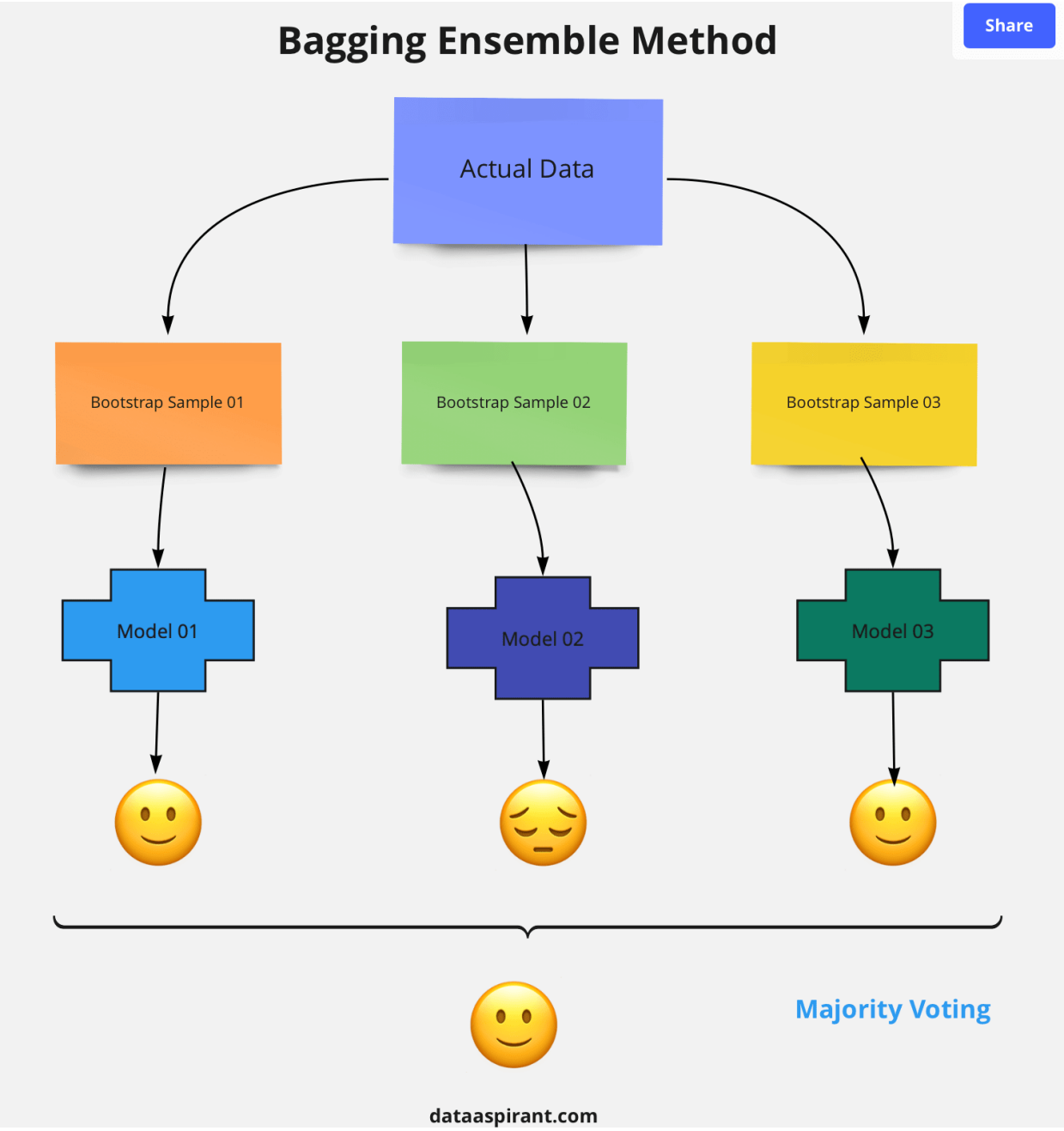


Bagging

Bagging, also known as Bootstrap Aggregation is the ensemble technique used by random forest. Bagging chooses a random sample from the data set. Hence each model is generated from the samples (Bootstrap Samples) provided by the Original Data with replacement known as row sampling. This step of row sampling with replacement is called bootstrap. Now each model is trained independently which generates results. The final output is based on majority voting after combining the results of all models. This step which involves combining all the results and generating output based on majority voting is known as aggregation.

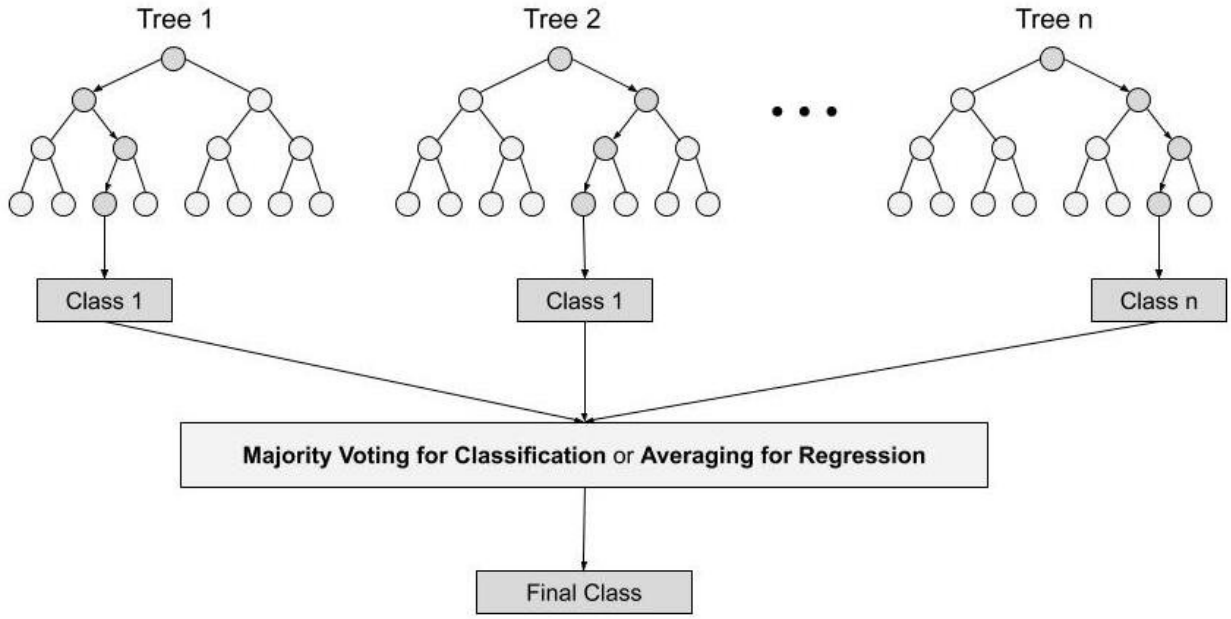


Now let's look at an example by breaking it down with the help of the following figure. Here the bootstrap sample is taken from actual data (Bootstrap sample 01, Bootstrap sample 02, and Bootstrap sample 03) with a replacement which means there is a high possibility that each sample won't contain unique data. Now the model (Model 01, Model 02, and Model 03) obtained from this bootstrap sample is trained independently. Each model generates results as shown. Now Happy emoji is having a majority when compared to sad emoji. Thus based on majority voting final output is obtained as Happy emoji.



Steps involved in random forest algorithm:

- Step 1: In Random forest n number of random records are taken from the data set having k number of records.
- Step 2: Individual decision trees are constructed for each sample.
- Step 3: Each decision tree will generate an output.
- Step 4: Final output is considered based on Majority Voting or Averaging for Classification and regression respectively.



A prediction on a regression problem is the average of the prediction across the trees in the ensemble. A prediction on a classification problem is the majority vote for the class label across the trees in the ensemble.

> Regression: Prediction is the average prediction across the decision trees.\
> Classification: Prediction is the majority vote class label predicted across the decision trees.

Random forest involves constructing a large number of decision trees from bootstrap samples from the training dataset, like bagging.

Unlike bagging, random forest also involves selecting a subset of input features (columns or variables) at each split point in the construction of trees. Typically, constructing a decision tree involves evaluating the value for each input variable in the data in order to select a split point. By reducing the features to a random subset that may be considered at each split point, it forces each decision tree in the ensemble to be more different

For More Info Visit Link

||Step 1-Data Preprocessing||

In [ ]:
```
#import required libraries
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import seaborn as sns
import matplotlib.pyplot as plt
```

In [ ]:
```
#Reading csv(comma separated values) file.......
# iris=pd.read_csv("../input/Iris.csv")
iris = sns.load_dataset('iris')
```

In [ ]:
```
#Check dataset.......
iris.head()
```

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

In [ ]:
```
#Checking the dimensions....
iris.shape
```

```
(150, 5)
```

In [ ]:
```
#checking whether a dataset contain a missing value or not/.....
iris.isnull().sum()
```

```
sepal_length    0
sepal_width     0
petal_length    0
petal_width     0
species         0
dtype: int64
```

In [ ]:
```
'''checking if there is any inconsistency in the dataset as we see there
are no null values in the dataset, so the data can be processed...../'''
iris.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   sepal_length  150 non-null    float64
 1   sepal_width   150 non-null    float64
 2   petal_length  150 non-null    float64
 3   petal_width   150 non-null    float64
 4   species       150 non-null    object 
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
```

In [ ]:
```
#Checkig the unique values in species column which is our target variable..
iris["species"].unique()
```

```
array(['setosa', 'versicolor', 'virginica'], dtype=object)
```

In [ ]:
```
'''dropping the Id column as it is unnecessary, axis=1 specifies that
it should be column wise, inplace =1 means
the changes should be reflected into the dataframe'''
# iris.drop('id',axis=1,inplace=True)
```

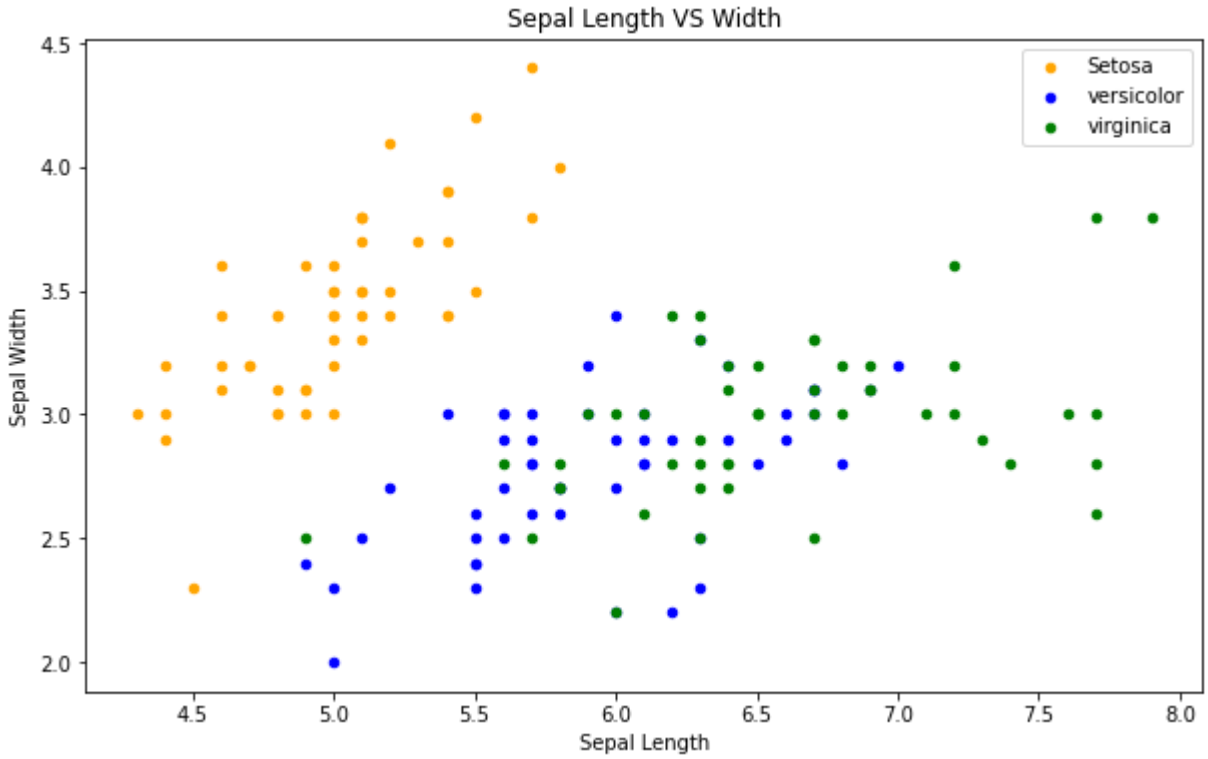
```
'dropping the Id column as it is unnecessary, axis=1 specifies that \nit should be colum
n wise, inplace =1 means \nthe changes should be reflected into the dataframe'
```

In [ ]:
```
#checking data after droping "ID column".....
iris.head()
```

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

||Visualize Sepal Length VS Sepal Width||

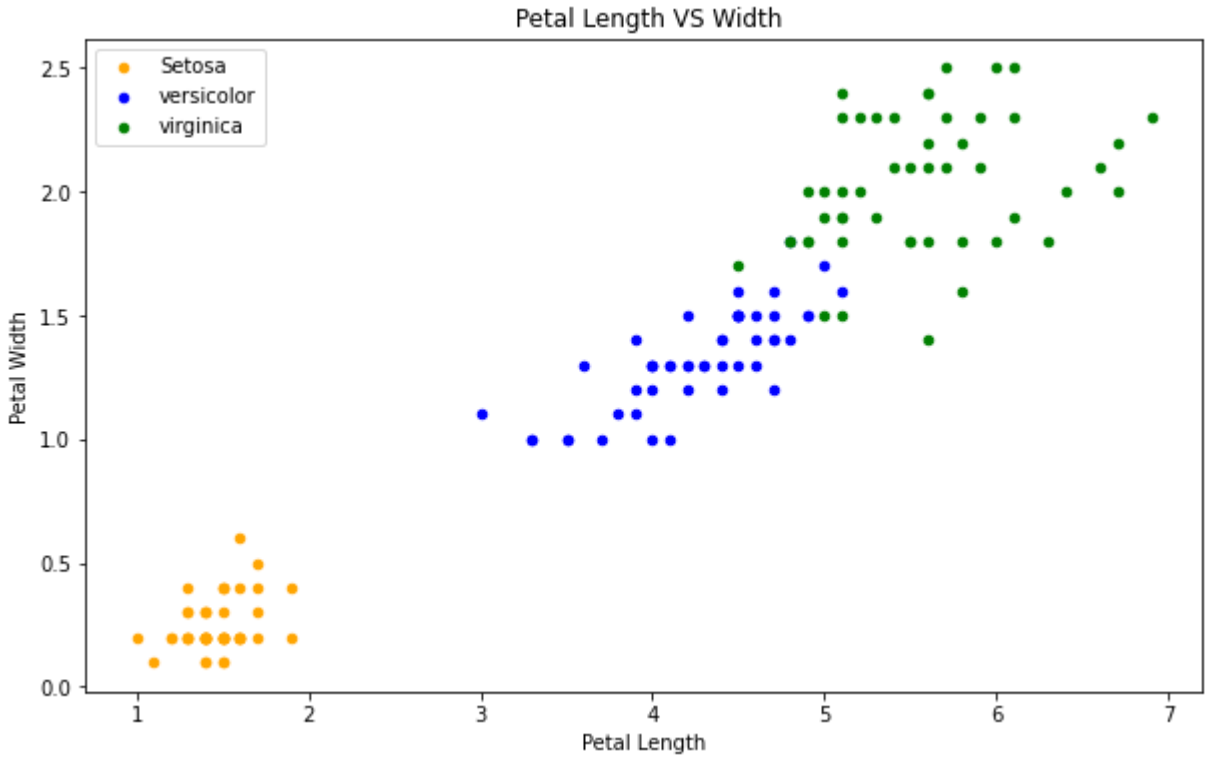
In [ ]:
```
'''Some Exploratory Data Analysis With Iris'''

fig = iris[iris.species=='setosa'].plot(kind='scatter',x='sepal_length',y='sepal_width'
iris[iris.species=='versicolor'].plot(kind='scatter',x='sepal_length',y='sepal_width',c
iris[iris.species=='virginica'].plot(kind='scatter',x='sepal_length',y='sepal_width',co
fig.set_xlabel("Sepal Length")
fig.set_ylabel("Sepal Width")
fig.set_title("Sepal Length VS Width")
fig=plt.gcf()
fig.set_size_inches(10,6)
plt.show()
fig.savefig("Sepal Length VS Width.png")
```



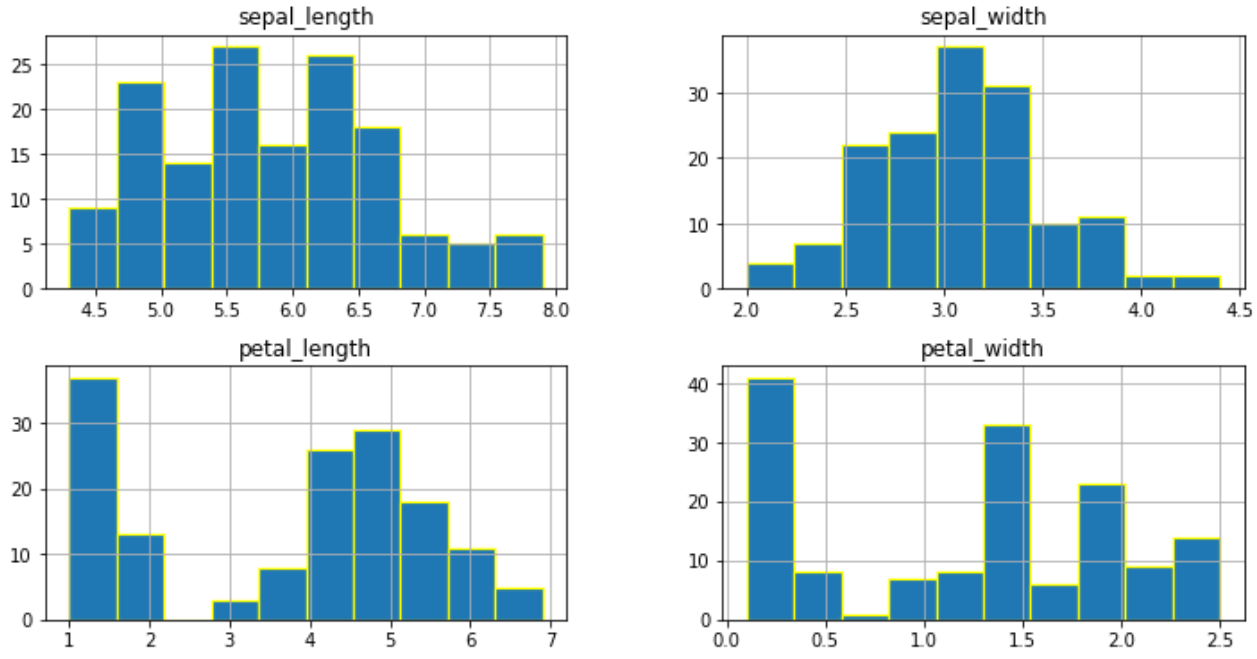
||Visualize Petal Length VS Petal Width||

In [ ]:

```
'''Some Exploratory Data Analysis With Iris'''
fig = iris[iris.species=='setosa'].plot(kind='scatter',x='petal_length',y='petal_width'
iris[iris.species=='versicolor'].plot(kind='scatter',x='petal_length',y='petal_width',c
iris[iris.species=='virginica'].plot(kind='scatter',x='petal_length',y='petal_width',co
fig.set_xlabel("Petal Length")
fig.set_ylabel("Petal Width")
fig.set_title(" Petal Length VS Width")
fig=plt.gcf()
fig.set_size_inches(10,6)
plt.show()
fig.savefig("Petal Length VS Width.png")
```



In [ ]:
```
'''let us see how are the length and width are distributed'''
iris.hist(edgecolor='Yellow', linewidth=1.2)
fig=plt.gcf()
fig.set_size_inches(12,6)
plt.show()
```

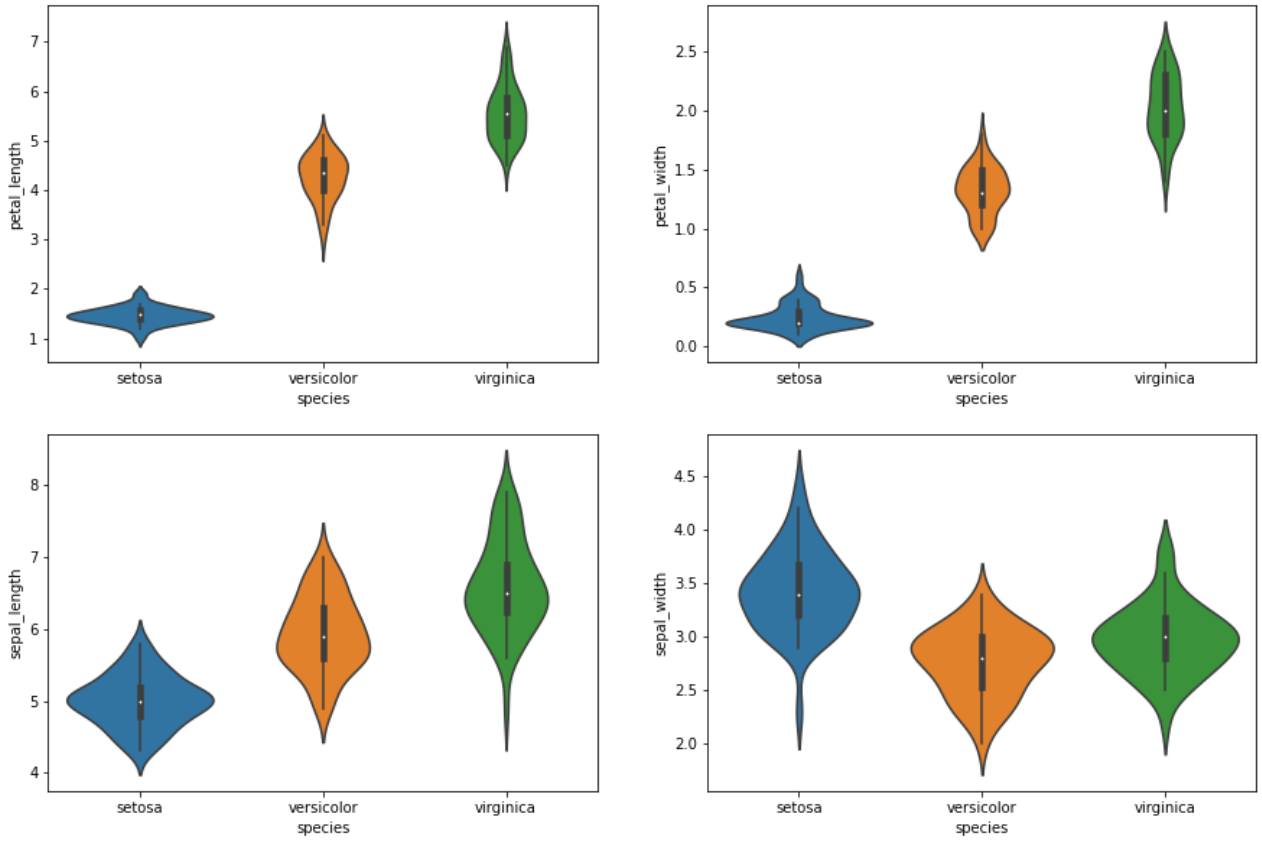


In [ ]:
```
iris.head()
```

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa

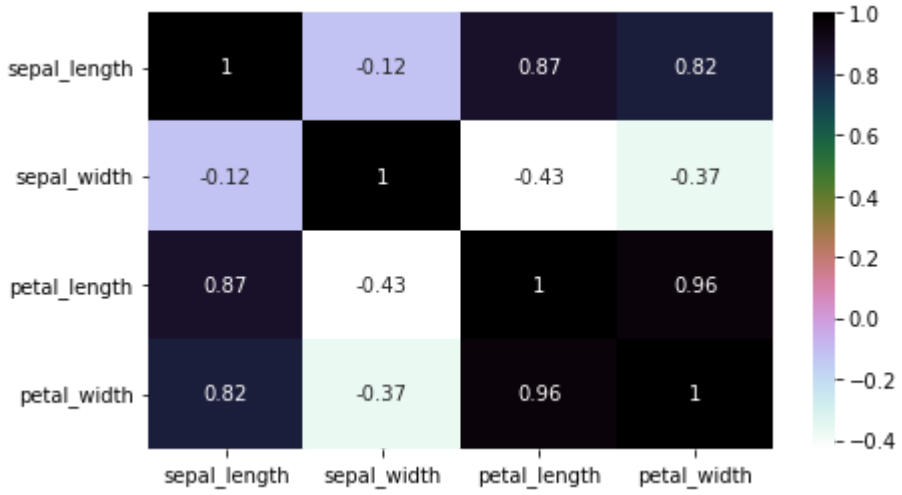
	sepal_length	sepal_width	petal_length	petal_width	species
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

In [ ]:
```
'''Let us see how the length and width vary according to the species'''
plt.figure(figsize=(15,10))
plt.subplot(2,2,1)
sns.violinplot(x='species',y='petal_length',data=iris)
plt.subplot(2,2,2)
sns.violinplot(x='species',y='petal_width',data=iris)
plt.subplot(2,2,3)
sns.violinplot(x='species',y='sepal_length',data=iris)
plt.subplot(2,2,4)
sns.violinplot(x='species',y='sepal_width',data=iris)
fig.savefig("variable with species.png")
```



'''Now, when we train any algorithm, the number of features and their correlation plays an important role. If there are features and many of the features are highly correlated, then training an algorithm with all the featues will reduce the accuracy. Thus features selection should be done carefully. This dataset has less featues but still we will see the correlation.'''

In [ ]:
```
plt.figure(figsize=(7,4)) #7 is the size of the width and 4 is parts....
sns.heatmap(iris.corr(),annot=True,cmap='cubehelix_r') #draws  heatmap with input as th
plt.show()
```



In [ ]:
```
#Separating dependent and independent values..
X=iris.iloc[:, :-1].values
X[:10]
```

```
array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2],
       [4.6, 3.1, 1.5, 0.2],
       [5. , 3.6, 1.4, 0.2],
       [5.4, 3.9, 1.7, 0.4],
       [4.6, 3.4, 1.4, 0.3],
       [5. , 3.4, 1.5, 0.2],
       [4.4, 2.9, 1.4, 0.2],
       [4.9, 3.1, 1.5, 0.1]])
```

In [ ]:
```
y=iris.iloc[:, -1].values
y[:10]
```

```
array(['setosa', 'setosa', 'setosa', 'setosa', 'setosa', 'setosa',
       'setosa', 'setosa', 'setosa', 'setosa'], dtype=object)
```

In [ ]:
```
#splitting into training set and test.
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_state =
```

||Step 2-Building a model||

In [ ]:
```
# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

In [ ]:
```
# Fitting Random Forest Classification to the Training set
from sklearn.ensemble import RandomForestClassifier
```

```
classifier = RandomForestClassifier(n_estimators = 10, criterion = 'entropy', random_st
classifier.fit(X_train, y_train)
```

```
RandomForestClassifier(criterion='entropy', n_estimators=10, random_state=0)
```

In [ ]:

```
# Predicting the Test set results
y_pred = classifier.predict(X_test)
```

In [ ]:

```
# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
cm
```

```
array([[13,  0,  0],
       [ 0,  5,  1],
       [ 0,  2,  9]], dtype=int64)
```

In [ ]:

```
from sklearn import metrics #for checking the model accuracy
print('The accuracy of the Random forest is:',metrics.accuracy_score(y_pred,y_test))
```

```
The accuracy of the Random forest is: 0.9
```

In [ ]:

In [ ]:

In [ ]:

Happy Learning