Participant Detail

NAME : "AZKA SALEEM"

STATUS: "STUDENT (PhD.BioInformatics)"

EMAIL: "azkasaleem527@gmail.com"

MACHINE LEARNING IS ABOUT DATA DRIVEN DECISION MAKING

It has two parts ;

- TRAINING
- PREDICTION

Machine learning is a method of data analysis that automates analytical model building. It is a branch of artificial intelligence based on the idea that systems can learn from data, identify patterns and make decisions with minimal human intervention.

Its goal and usage is to build new and/or leverage existing algorithms to learn from data, in order to build generalizable models that give accurate predictions, or to find patterns, particularly with new and unseen similar data.

There are four types of machine learning algorithms:

- **SUPERVISED**
  - 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.
- **SEMI-SUPERVISED**
  - is an approach to machine learning that combines a small amount of labeled data with a large amount of unlabeled data during training. Semi-supervised learning falls between unsupervised learning (with no labeled training data) and supervised learning (with only labeled training data). It is a special instance of weak supervision.
- **UNSUPERVISED**
  - is a machine learning technique in which the users do not need to supervise the model. Instead, it allows the model to work on its own to discover patterns and information that was previously undetected. It mainly deals with the unlabelled data.
  - Unsupervised learning algorithms include clustering (K-means clustering,probabilictic clustering,

    - hieratical clustring), anomaly detection, neural networks, etc.
- **REINFORCEMENTt**
    - There is no supervisor, only a real number or reward signal
    - Sequential decision making
    - time plays a crucial role in Reinforcement problems
    - Feedback is always delayed, not instantaneous
    - Agent's actions determine the subsequent data it receives
    - Two types of reinforcement learning are 1) Positive 2) Negative
    - Two widely used learning model are 1) Markov Decision Process 2) Q learning

- Supervised learning can be separated into two types, classification and regression:

1. **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.

1. **Regression** is used to understand the relationship between dependent and independent **numeric** 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.

1. SIMPLE LINEAR REGRESSION

Example: National Unemployment Male Vs. Female

In the following data pairs X = national unemployment rate for adult males Y = national unemployment rate for adult females Reference: Statistical Abstract of the United States

In [ ]:
```
#import libararies and load dataset
import pandas as pd
import numpy as np
#pip install xlrd
df = pd.read_excel("slr04.xls", engine='xlrd')
df.head()
```

```
*** No CODEPAGE record, no encoding_override: will use 'iso-8859-1'
```

Out[ ]:

	X	Y
0	2.9	4.0
1	6.7	7.4
2	4.9	5.0
3	7.9	7.2
4	9.8	7.9

In [ ]:
```
#assigning x and y from the dataset
X = df.iloc[:, :-1].values #get a copy of dataset exclude last column
```

```
y = df.iloc[:, 1].values #get array of dataset in column 1st
```

In [ ]:
```
X
```

Out[ ]:
```
array([[2.9000001 ],
       [6.69999981],
       [4.9000001 ],
       [7.9000001 ],
       [9.80000019],
       [6.9000001 ],
       [6.0999999 ],
       [6.19999981],
       [6.        ],
       [5.0999999 ],
       [4.69999981],
       [4.4000001 ],
       [5.80000019]])
```

In [ ]:
```
y
```

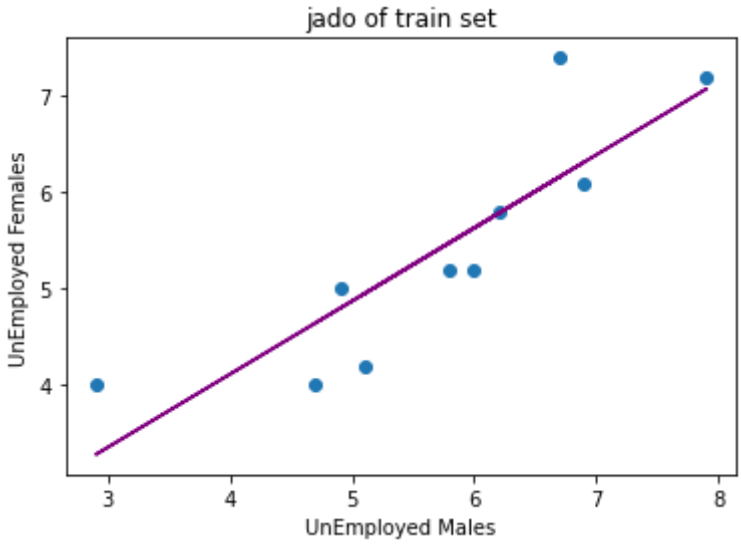
Out[ ]:
```
array([4.        , 7.4000001 , 5.        , 7.19999981, 7.9000001 ,
       6.0999999 , 6.        , 5.80000019, 5.19999981, 4.19999981,
       4.        , 4.4000001 , 5.19999981])
```

In [ ]:
```
# import libarary
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test= train_test_split(X ,y, test_size=1/5, random_sta
```

In [ ]:
```
#fit linear regression model
from sklearn.linear_model import LinearRegression
model= LinearRegression().fit(X_train, y_train)
model
```

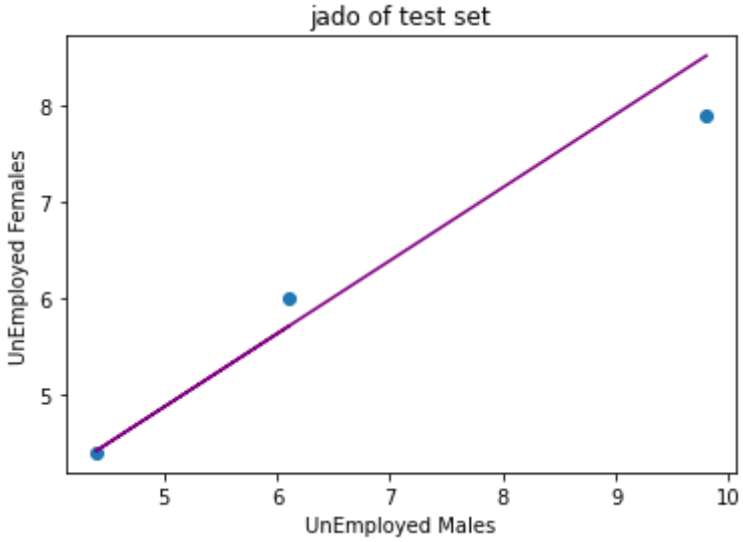
Out[ ]:
```
LinearRegression()
```

In [ ]:
```
#plotting/ visualization
import matplotlib.pyplot as plt
plt.scatter(X_train, y_train)
plt.plot((X_train), model.predict(X_train), color="Purple")
plt.xlabel("UnEmployed Males")
plt.ylabel("UnEmployed Females")
plt.title("jado of train set")
plt.show()
```



In [ ]:
```
#plotting/ visualization
```

```
import matplotlib.pyplot as plt
plt.scatter(X_test, y_test)
plt.plot(X_test,model.predict(X_test),color="Purple")
plt.xlabel("UnEmployed Males")
plt.ylabel("UnEmployed Females")
plt.title("jado of test set")
plt.show()
```



In [ ]:
```
#model fitness of test set
print("score of testing data = ",model.score(X_test, y_test) )
#model fitness of train set
print("score of training data = ",model.score(X_train, y_train))
```

```
score of training data =  0.9235294005913652
score of testing data =  0.7483123463214242
```

In [ ]:
```
#prediction of unknown values
model.predict([[5]])
```

Out[ ]:
```
array([4.87029591])
```

In [ ]:
```
model.predict([[20]])
```

Out[ ]:
```
array([16.27249384])
```

In [ ]:
```
model.predict([[5.9],[7.9]])
```

Out[ ]:
```
array([5.55442779, 7.07472085])
```

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

Out[ ]:
```
array([5.70645702, 4.41420807, 8.5189994 ])
```

---

2. MULTIPLE LINEAR REGRESSION

- more than two variables
- one dependent and more than one independent variables
- independent variables are also called as input data or features

- dependent variables are also called as prdiction or output

Example: Hollywood Movies

The data (X1, X2, X3, X4) are for each movie X1 = first year box office receipts/millions X2 = total production costs/millions X3 = total promotional costs/millions X4 = total book sales/millions

In [ ]:
```
#import libararies
from sklearn.linear_model import LinearRegression
import pandas as pd
import numpy as np
#pip install xlrd to read excel file
df = pd.read_excel("mlr04.xls", engine='xlrd')
df.head()
```

```
*** No CODEPAGE record, no encoding_override: will use 'iso-8859-1'
```

Out[ ]:

	X1	X2	X3	X4
0	85.099998	8.5	5.100000	4.7
1	106.300003	12.9	5.800000	8.8
2	50.200001	5.2	2.100000	15.1
3	130.600006	10.7	8.399999	12.2
4	54.799999	3.1	2.900000	10.6

In [ ]:
```
##assigning x and y from the dataset
X = df.iloc[:, :-1].values #get a copy of dataset exclude last column
y = df.iloc[:, 3].values #get array of dataset in last column
```

In [ ]:
```
#looking into X (its 2d array)
X
```

Out[ ]:
```
array([[ 85.09999847,   8.5       ,   5.0999999 ],
       [106.3000031 ,  12.89999962,   5.80000019],
       [ 50.20000076,   5.19999981,   2.0999999 ],
       [130.6000061 ,  10.69999981,   8.39999867],
       [ 54.79999924,   3.0999999 ,   2.9000001 ],
       [ 30.29999924,   3.5       ,   1.20000005],
       [ 79.40000153,   9.19999981,   3.70000005],
       [ 91.        ,   9.        ,   7.5999999 ],
       [135.3999939 ,  15.10000038,   7.69999981],
       [ 89.30000305,  10.19999981,   4.5       ]])
```

In [ ]:
```
#looking into y(its 1d array)
y
```

Out[ ]:
```
array([ 4.69999981,  8.80000019, 15.10000038, 12.19999981, 10.60000038,
        3.5       ,  9.69999981,  5.9000001 , 20.79999924,  7.9000001 ])
```

In [ ]:
```
# creat and fit your model
model= LinearRegression().fit(X,y)
model
```

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

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

Out[ ]:
```
array([ 0.34067981, -0.75651621, -2.75656564])
```

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

Out[ ]:
```
0.9995754298657626
```

In [ ]:
```
#looking for random predictions
model.predict([['50','5','2.2']])
```

```
C:\Users\Azka\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\base.py:566: FutureWarning: Arrays of bytes/strings is being converted to decimal numbers if dtype='numeric'. This behavior is deprecated in 0.24 and will be removed in 1.1 (renaming of 0.26). Please convert your data to numeric values explicitly instead.
  X = check_array(X, **check_params)
```

Out[ ]:
```
array([8.1865406])
```

In [ ]:
```
#efficacy of model
print("efficacy score= ",model.score(X, y))
```

```
efficacy score=  0.194623258328661
```

In [ ]:
```
# try splitting 80/20
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test= train_test_split(X ,y, test_size=1/5, random_sta
```

In [ ]:
```
#fitting again
from sklearn.linear_model import LinearRegression
model= LinearRegression().fit(X_train, y_train)
model
```

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

In [ ]:
```
#prediction for Xtest data
y_pred= model.predict(X_test)
y_pred
```

Out[ ]:
```
array([[ 6.79228712],
       [10.53327692]])
```

In [ ]:
```
#evaluate model through R2 score
from sklearn.metrics import r2_score
print("r2_score = ", r2_score(y_test,y_pred))
```

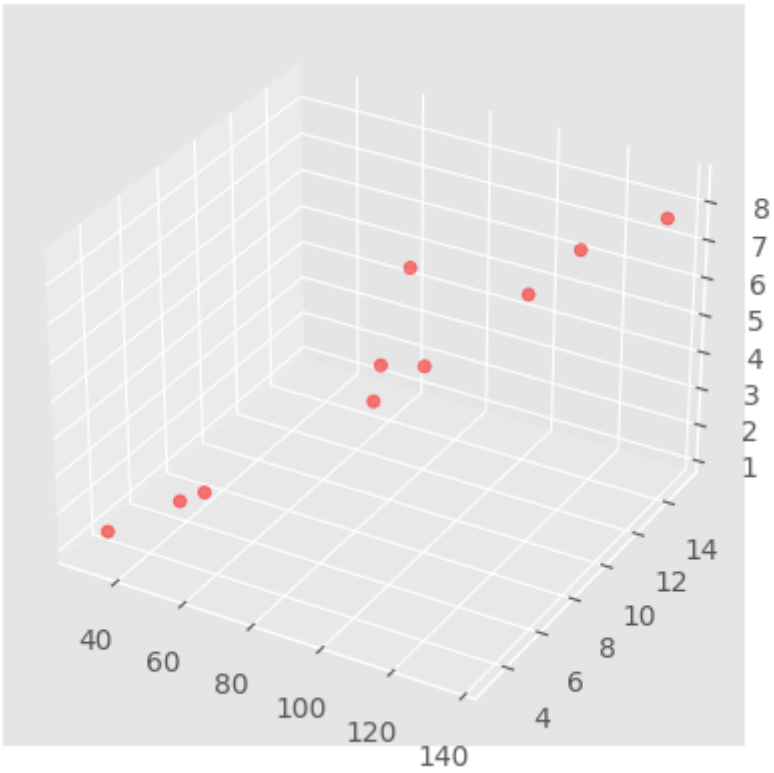
```
r2_score =  -9.737073352032414
```

In [ ]:
```
#playing with shapes and dimensions of both arrays
X.shape
y.shape
y=y[:,np.newaxis]
X.ndim
y.ndim
```

Out[ ]:
```
(10, 1)
```

In [ ]:
```
#plotting the data of multivars on 3d projection
fig = plt.figure()
ax=fig.add_subplot(111,projection='3d')
ax.scatter(X[:, 0],X[:, 1],X[:, 2],y,c='red',alpha=0.5)
plt.show()
```

```
C:\Users\Azka\AppData\Local\Programs\Python\Python310\lib\site-packages\mpl_toolki
ts\mplot3d\art3d.py:906: FutureWarning: elementwise comparison failed; returning s
calar instead, but in the future will perform elementwise comparison
  if zdir == 'x':
C:\Users\Azka\AppData\Local\Programs\Python\Python310\lib\site-packages\mpl_toolki
ts\mplot3d\art3d.py:908: FutureWarning: elementwise comparison failed; returning s
calar instead, but in the future will perform elementwise comparison
  elif zdir == 'y':
C:\Users\Azka\AppData\Local\Programs\Python\Python310\lib\site-packages\mpl_toolki
ts\mplot3d\art3d.py:910: FutureWarning: elementwise comparison failed; returning s
calar instead, but in the future will perform elementwise comparison
  elif zdir[0] == '-':
```



---

3. Decision Tree classification

it is a supervised learning algorithm. it is a non paramatric supervised learning method used for classification and regression

- if then elese decision rule
- easy use and interpret
- little data prep needed
- handle numeric and catagorical
- multi output problems can be handled
- use boolean logic to model the prediction
- perform well

In [ ]:
```
#import libarary
import pandas as pd
#load dataset
```

```
df=pd.read_csv("multivarBiryani.csv")
df.head()
```

Out[ ]:

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

In [ ]:

```
#replacing male and female from gender column with 1,0
df['gender']= df['gender'].replace('Male',1)
df['gender']= df['gender'].replace('Female',0)
df.tail()
```

Out[ ]:

	age	height	weight	gender	likeness
240	31	160.0	60.0	1	Pakora
241	26	172.0	70.0	1	Biryani
242	40	178.0	80.0	1	Biryani
243	25	5.7	65.0	1	Biryani
244	33	157.0	56.0	0	Samosa

In [ ]:

```
#droping extra columns
df= df.drop(['age', 'height'],axis=1)
df.head()
```

Out[ ]:

	weight	gender	likeness
0	76.0	1	Biryani
1	70.0	1	Biryani
2	80.0	1	Biryani
3	102.0	1	Biryani
4	67.0	1	Biryani

In [ ]:

```
##assigning x and y from the dataset
X = df.iloc[:, :-1].values #get a copy of dataset exclude last column
y = df.iloc[:, 2].values #get array of dataset in column 1st
```

In [ ]:

```
#look into X and y
#X
#y
```

In [ ]:

```
# import machine learning algorithm
from sklearn.tree import DecisionTreeClassifier

#creat and fit our model
model= DecisionTreeClassifier().fit(X,y)

#predictions
model.predict([["67.0","1"]])
```

Out[ ]:
```
array(['Biryani'], dtype=object)
```

In [ ]:
```
#split data into test and train (80/20)%rule
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

#creat a model
model.fit(X_train, y_train)

#prediction
predicted_values =model.predict(X_test)
predicted_values

#accuracy_score of the model
from sklearn.metrics import accuracy_score
score = accuracy_score(y_test, predicted_values)
print("accuracy_score of the model = ",score)
```

```
accuracy_score of the model =  0.5102040816326531
```

In [ ]:
```
#how to train and save your model
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
import joblib
model = DecisionTreeClassifier().fit(X,y)

joblib.dump(model, "foodie.joblib")
```

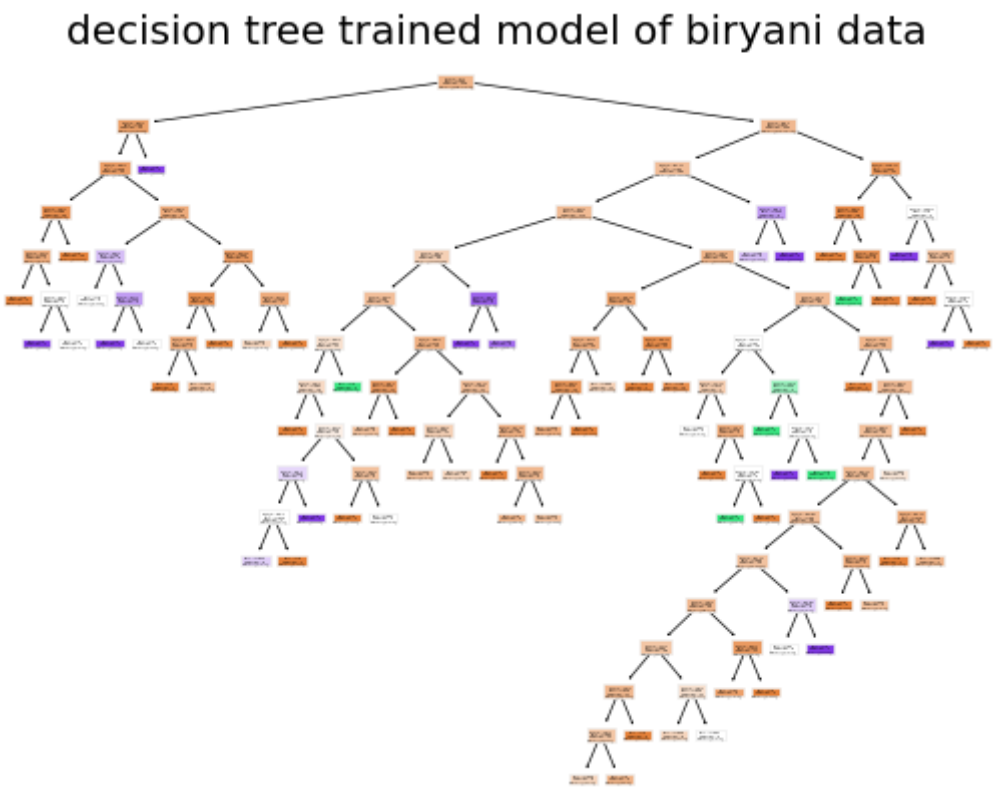
Out[ ]:
```
['foodie.joblib']
```

In [ ]:
```
# how to import/run a stored/saved model on our data
# some time later...

# load the model from disk
loaded_model = joblib.load("foodie.joblib")

#calculating model score
score = loaded_model.score(X_test, y_test)
print('model score = ',score)
```

```
model score =  0.6938775510204082
```

In [ ]:
```
#export a decission tree graph
from sklearn import tree
model= DecisionTreeClassifier().fit(X,y)

#graphic evaluation
tree.export_graphviz(model,
                     out_file="foodie.dot",
                     feature_names=["age","gender"],
                     class_names=sorted(y),
                     label= "all",
                     rounded=True,
                     filled=True)
```

In [ ]:
```
#plot a decission tree plot
from sklearn.tree import plot_tree

plot_tree(model, filled=True)
plt.title("decision tree trained model of biryani data")
plt.show()
```




```
<Figure size 1500x1500 with 0 Axes>
```

---

4. k-nearest neighbors (KNN)

The k-nearest neighbors (KNN) algorithm is a simple, supervised machine learning algorithm that can be used to solve both classification and regression problems. It's easy to implement and understand, but has a major drawback of becoming significantly slows as the size of that data in use grows.

> it mainly depends upon 4 factors;
>
> - point
> - k value
> - jamhoriyat
> - rishtydari

```
- k= number of neighbors
- k should not be low nor too high
- predict the response value based on the neighbors which is
nearest and more in numbers (minkowski distance)
- can also be used for numerical data/ regression
```

k-nearest neighbor accuracy measurement

1. jaccard index
2. F1_score
3. log loss
4. some others also

A. classification accuracy
B. confusion matrix
C. area under curve
D. mean absolute error
E. mean squared error

- accuracy_score can be replaced by
- precision _score
- recall_score
- f1_score ## pros of KNN
- training phase is faster
- instance based learning algorithm
- can be used with non linear data ## cons of KNN
- testing phase is slower
- costly for memory and computation
- not suitable for large dimensions ## how to improve:
- data wrangling and scaling
- missing value
- normalization on same scale for everything (-1-0-1)
- reduce dimensions to improve performance

lets get hands on!

In [ ]:
```
#import libarary
import pandas as pd

#load dataset
df = pd.read_csv("multivarBiryani.csv")

#replacing male and female from gender column with 1,0
df['gender']= df['gender'].replace('Male',1)
df['gender']= df['gender'].replace('Female',0)

df.tail()
```

Out[ ]:

	age	height	weight	gender	likeness
240	31	160.0	60.0	1	Pakora
241	26	172.0	70.0	1	Biryani
242	40	178.0	80.0	1	Biryani
243	25	5.7	65.0	1	Biryani
244	33	157.0	56.0	0	Samosa

In [ ]:
```
#selection of input and output vars
X = df[["weight","gender"]]
y = df["likeness"]
```

In [ ]:
```
#look into X and y
#X
#y
```

In [ ]:
```
#creat the model
```

```
from sklearn.neighbors import KNeighborsClassifier
model = KNeighborsClassifier(n_neighbors=5)

#train the model
model.fit(X,y)

#predict output
predicted= model.predict([[70,1]])
predicted
```

```
C:\Users\Azka\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\ba
se.py:450: UserWarning: X does not have valid feature names, but KNeighborsClassif
ier was fitted with feature names
  warnings.warn(
```

Out[ ]:

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

In [ ]:

```
model.predict(X)
```

Out[ ]:

```
array(['Biryani', 'Biryani', 'Biryani', 'Biryani', 'Biryani', 'Biryani',
       'Biryani', 'Biryani', 'Biryani', 'Biryani', 'Biryani', 'Biryani',
       'Biryani', 'Biryani', 'Biryani', 'Biryani', 'Biryani', 'Biryani',
       'Biryani', 'Biryani', 'Biryani', 'Biryani', 'Biryani', 'Biryani',
       'Biryani', 'Biryani', 'Biryani', 'Biryani', 'Biryani', 'Biryani',
       'Biryani', 'Biryani', 'Samosa', 'Biryani', 'Biryani', 'Biryani',
       'Biryani', 'Biryani', 'Biryani', 'Biryani', 'Biryani', 'Samosa',
       'Biryani', 'Biryani', 'Biryani', 'Biryani', 'Pakora', 'Biryani',
       'Biryani', 'Biryani', 'Biryani', 'Biryani', 'Biryani', 'Biryani',
       'Biryani', 'Biryani', 'Biryani', 'Biryani', 'Biryani', 'Biryani',
       'Biryani', 'Biryani', 'Biryani', 'Biryani', 'Biryani', 'Biryani',
       'Biryani', 'Biryani', 'Biryani', 'Biryani', 'Biryani', 'Biryani',
       'Biryani', 'Biryani', 'Biryani', 'Biryani', 'Biryani', 'Biryani',
       'Biryani', 'Biryani', 'Biryani', 'Biryani', 'Biryani', 'Biryani',
       'Biryani', 'Biryani', 'Biryani', 'Biryani', 'Biryani', 'Samosa',
       'Biryani', 'Biryani', 'Biryani', 'Biryani', 'Biryani', 'Biryani',
       'Biryani', 'Biryani', 'Biryani', 'Biryani', 'Biryani', 'Biryani',
       'Biryani', 'Biryani', 'Biryani', 'Biryani', 'Biryani', 'Biryani',
       'Biryani', 'Biryani', 'Biryani', 'Biryani', 'Biryani', 'Biryani',
       'Biryani', 'Biryani', 'Biryani', 'Biryani', 'Biryani', 'Biryani',
       'Biryani', 'Biryani', 'Biryani', 'Biryani', 'Biryani', 'Biryani',
       'Biryani', 'Biryani', 'Biryani', 'Biryani', 'Biryani', 'Biryani',
       'Biryani', 'Biryani', 'Biryani', 'Biryani', 'Biryani', 'Biryani',
       'Biryani', 'Biryani', 'Biryani', 'Biryani', 'Biryani', 'Biryani',
       'Biryani', 'Biryani', 'Biryani', 'Biryani', 'Biryani', 'Biryani',
       'Biryani', 'Biryani', 'Biryani', 'Biryani', 'Biryani', 'Biryani',
       'Biryani', 'Biryani', 'Biryani', 'Biryani', 'Biryani', 'Biryani',
       'Biryani', 'Samosa', 'Pakora', 'Biryani', 'Biryani', 'Biryani',
       'Biryani', 'Biryani', 'Biryani', 'Biryani', 'Samosa', 'Biryani',
       'Biryani', 'Samosa', 'Biryani', 'Biryani', 'Biryani', 'Biryani',
       'Pakora', 'Biryani', 'Biryani', 'Biryani', 'Biryani', 'Biryani',
       'Biryani', 'Samosa', 'Biryani', 'Biryani', 'Biryani', 'Biryani',
       'Biryani', 'Biryani', 'Biryani', 'Biryani', 'Biryani', 'Biryani',
       'Biryani', 'Biryani', 'Samosa', 'Biryani', 'Biryani', 'Biryani',
       'Biryani', 'Biryani', 'Biryani', 'Biryani', 'Samosa', 'Biryani',
       'Biryani', 'Biryani', 'Biryani', 'Biryani', 'Biryani', 'Biryani',
       'Biryani', 'Biryani', 'Biryani', 'Pakora', 'Samosa', 'Biryani',
       'Biryani', 'Biryani', 'Samosa', 'Biryani', 'Biryani', 'Biryani',
       'Biryani', 'Biryani', 'Biryani', 'Biryani', 'Biryani', 'Biryani',
       'Biryani', 'Biryani', 'Biryani', 'Biryani', 'Biryani'],
      dtype=object)
```

In [ ]:

```
#split data into train and test (80/20)
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)

#creat a model
model = KNeighborsClassifier()

#fitting a  model
model.fit(X_train, y_train)

predicted_values = model.predict(X_test)
predicted_values

#checking score/evaluation
from sklearn.metrics import accuracy_score
score = accuracy_score(y_test,predicted_values)
print("the accuracy score of the model is = ",score)
```

```
the accuracy score of the model is =  0.7142857142857143
```

---

5. Logistic Regression in Machine Learning

- popular algorithm of Supervised Learning technique.
- It is used for predicting the categorical dependent variable using a given set of independent variables.
- The outcome must be a categorical or discrete value but instead of giving the exact value as 0 and 1, it gives the probabilistic values which lie between 0 and 1.
- Solves the classification problems.
- Fit an "S" shaped logistic function, which predicts two maximum values (0 or 1).
- The curve from the logistic function indicates the likelihood of something such as whether the cells are cancerous or not, a mouse is obese or not based on its weight, etc.
- The sigmoid function is a mathematical function used to map the predicted values to probabilities.

Let's hands on!

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

In [ ]:
```
#load dataset
from sklearn.datasets import load_digits
digits = load_digits()
```

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

Out[ ]:
```
(1797, 64)
```

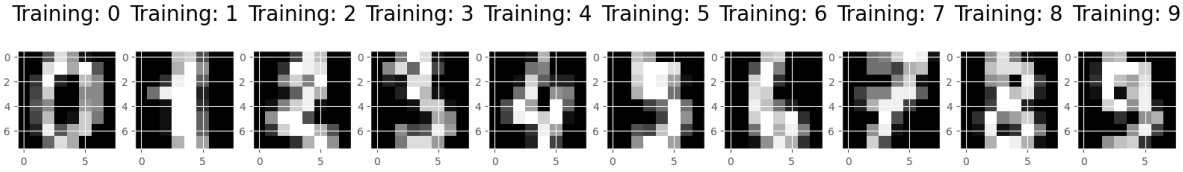
In [ ]:
```
X= digits.data
```

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

Out[ ]:
```
(1797,)
```

In [ ]:
```
y = digits.target
```

In [ ]:
```
#visualization
plt.figure(figsize=(20,4))
for index,(image, label)in enumerate(zip(X[0:10],y[0:10])):
    plt.subplot(1,10,index+1)
    plt.imshow(np.reshape(image,(8,8)),cmap=plt.cm.gray)
    plt.title("Training: %i\n" % label , fontsize = 20)
```



In [ ]:
```
#split data
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test= train_test_split(X ,y, test_size=1/5, random_sta

#creat and fit model
from sklearn.linear_model import LogisticRegression
model= LogisticRegression().fit(X_train,y_train)
model

#look into predictions
predictions = model.predict(X_test)
predictions

# check accuracy score of model
score = model.score(X_test,y_test)
print("the accuracy score is: ",score)
```

```
the accuracy score is:  0.9666666666666667
```

```
C:\Users\Azka\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\li
near_model\_logistic.py:814: ConvergenceWarning: lbfgs failed to converge (status=
1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

In [ ]:
```
#if you want to check the shape/size of test train data
# print("shape of train input data: ",X_train.shape)
# print("shape of test input data: ",X_test.shape)
# print("shape of train output data: ",y_train.shape)
# print("shape of test output data: ",y_test.shape)
```

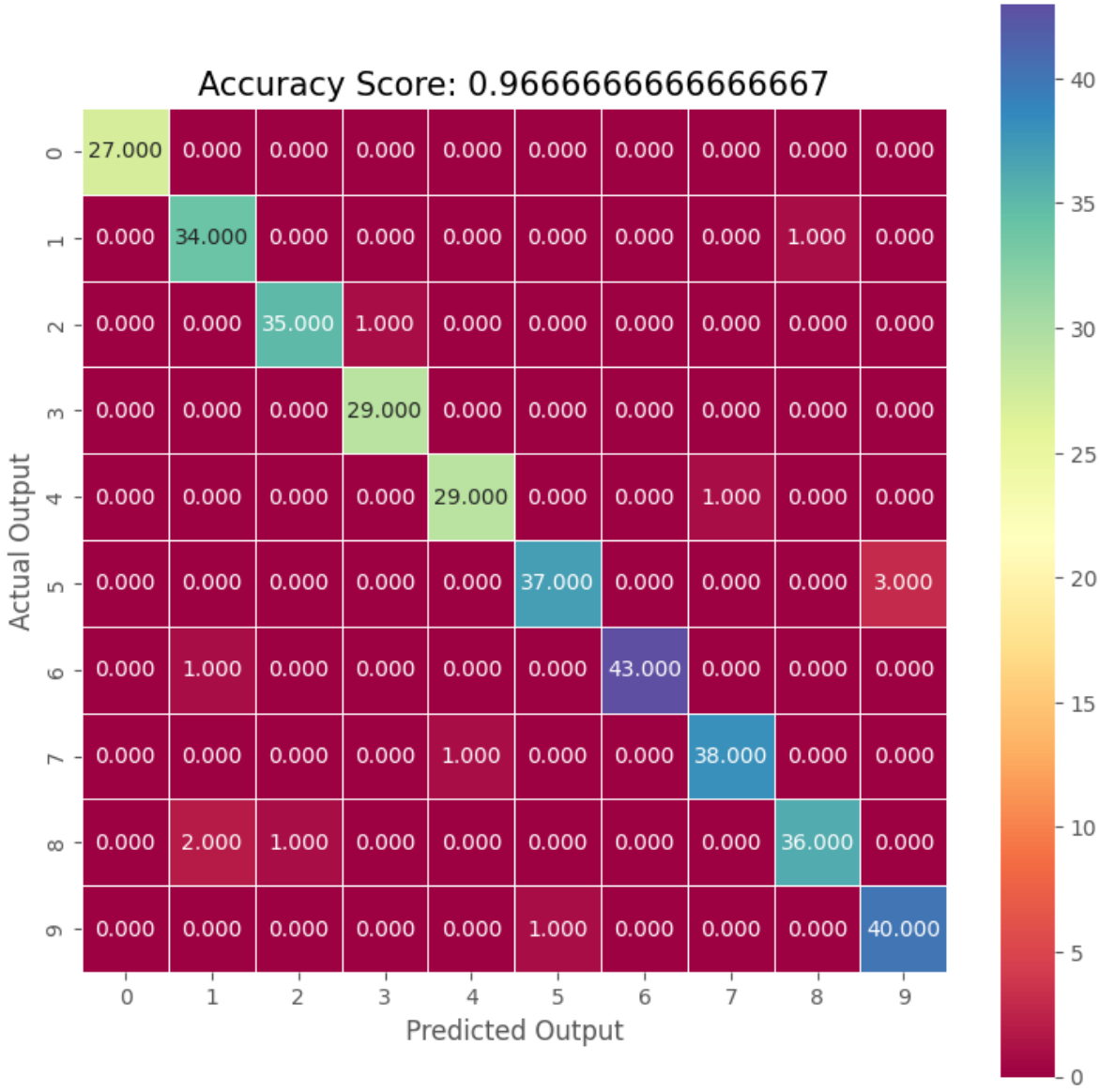
In [ ]:
```
#see prediction for first ten rows of x test
model.predict(X_test[0:10])
```

Out[ ]:
```
array([2, 8, 2, 6, 6, 7, 1, 9, 8, 5])
```

In [ ]:
```
# confusion metrix
from sklearn import metrics
cm = metrics.confusion_matrix(y_test, predictions)
cm
```

Out[ ]:
```
array([[27,  0,  0,  0,  0,  0,  0,  0,  0,  0],
       [ 0, 34,  0,  0,  0,  0,  0,  0,  1,  0],
       [ 0,  0, 35,  1,  0,  0,  0,  0,  0,  0],
       [ 0,  0,  0, 29,  0,  0,  0,  0,  0,  0],
       [ 0,  0,  0,  0, 29,  0,  0,  1,  0,  0],
       [ 0,  0,  0,  0,  0, 37,  0,  0,  0,  3],
       [ 0,  1,  0,  0,  0,  0, 43,  0,  0,  0],
       [ 0,  0,  0,  0,  1,  0,  0, 38,  0,  0],
       [ 0,  2,  1,  0,  0,  0,  0,  0, 36,  0],
       [ 0,  0,  0,  0,  0,  1,  0,  0,  0, 40]], dtype=int64)
```

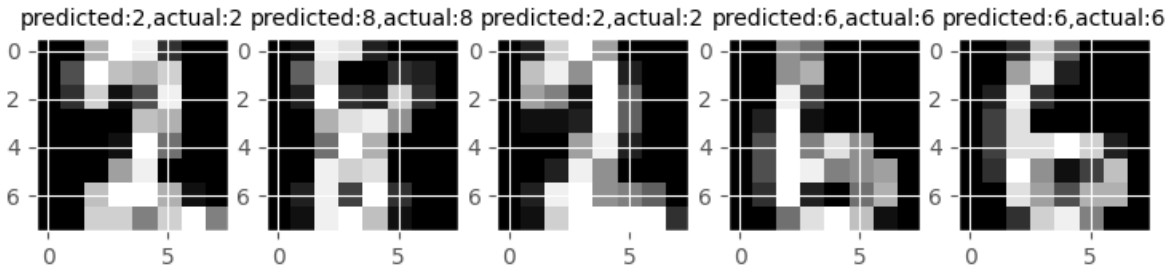
In [ ]:
```
# draw in table format

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



In [ ]:
```
# Getting miss classified labels

import numpy as np
import matplotlib.pyplot as plt
index = 0
```

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

In [ ]:
```
# plotting missclassified labels with known
plt.figure(figsize=(9,9))
for plotIndex , badIndex in enumerate( misclassifiedIndexes[0:5]):
    plt.subplot(1,5, plotIndex + 1)
    plt.imshow(np.reshape(X_test[badIndex],(8,8)),cmap=plt.cm.gray)
    plt.title("predicted:{},actual:{}".format(predictions[badIndex],y_test[badIndex
```



---

6. Random Forest Alogorithm

- Random forest is a Supervised Machine Learning Algorithm that is used widely in Classification and Regression problems.
- It builds decision trees on different samples and takes their majority vote for classification and average in case of regression.
- One of the most important features of the Random Forest Algorithm is that it can handle the data set containing continuous variables as in the case of regression and categorical variables as in the case of classification.
- It performs better results for classification problems.

Let's hands on!

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

#load sample data set
df = pd.read_csv("multivarBiryani.csv")
df.head()
```

Out[ ]:

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

In [ ]:
```
#replacing male and females of gender column with 1 and 0 respectively.
df['gender']= df['gender'].replace('Male',1)
df['gender']= df['gender'].replace('Female',0)
df.head()
```

Out[ ]:

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

In [ ]:
```
#assigning x and y from df
X= df.iloc[ : ,:-1] # all columns except the last one
y= df.iloc[ : ,-1:] # only the last one
```

In [ ]:
```
#import algorithm
from sklearn.ensemble import RandomForestClassifier

#creat model
model= RandomForestClassifier(n_estimators=100)
model

#fit model
model.fit(X,np.ravel(y))

#make prediction
model.predict([[29,164.000,67.0,1]])
```

```
C:\Users\Azka\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\ba
se.py:450: UserWarning: X does not have valid feature names, but RandomForestClass
ifier was fitted with feature names
  warnings.warn(
```

Out[ ]:
```
array(['Biryani'], dtype=object)
```

In [ ]:
```
#split data
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test= train_test_split(X ,y, test_size=1/5, random_sta

#creat and fit model
from sklearn.ensemble import RandomForestClassifier
model= RandomForestClassifier(n_estimators=100)
model.fit(X_train,np.ravel(y_train))

#look into predictions
predictions = model.predict(X_test)
```

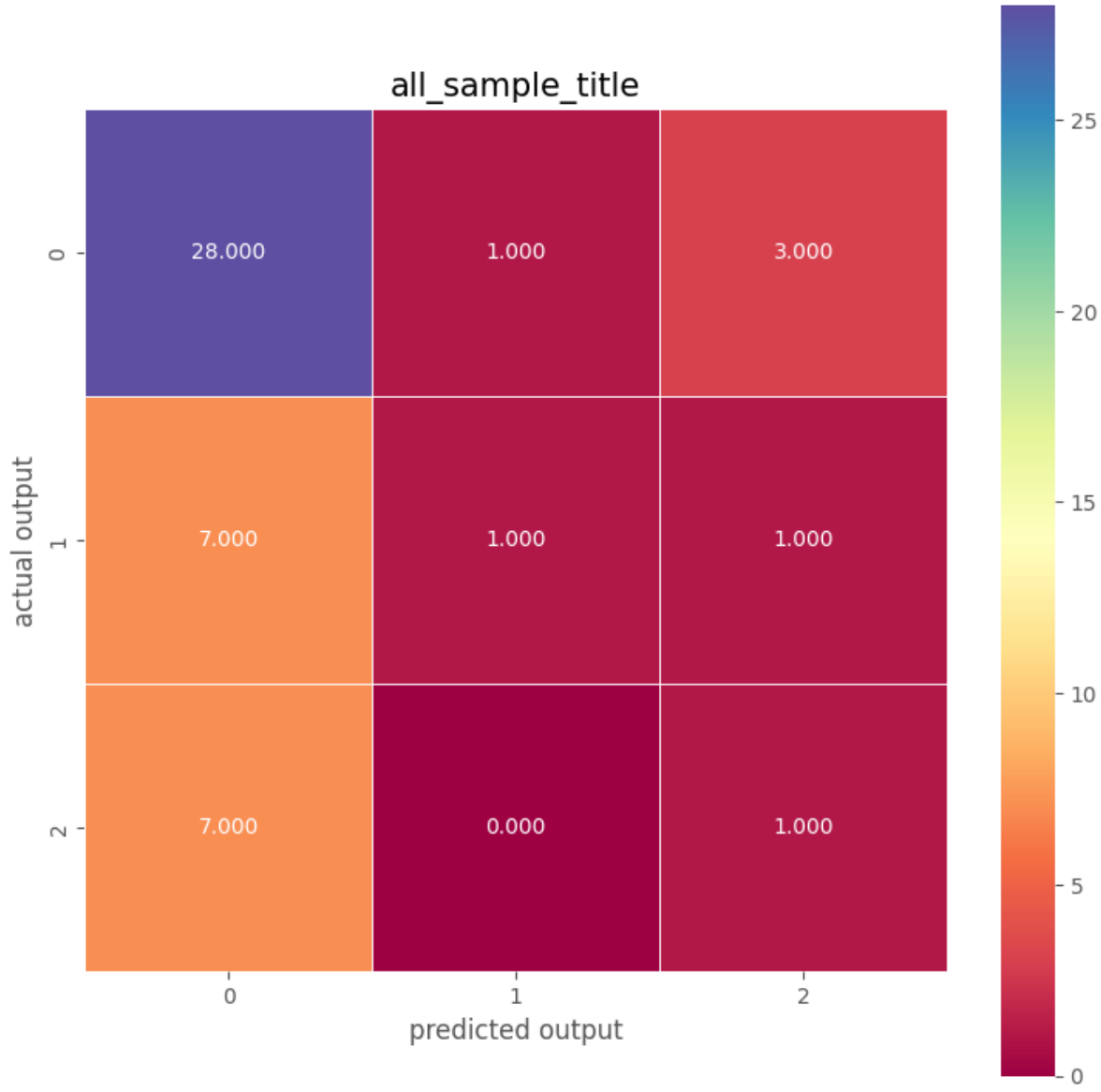
```
predictions

# check accuracy score of model
score = model.score(X_test,y_test)
print("The Accuracy score of model = ",score)
```

```
The Accuracy score of model =  0.6122448979591837
```

In [ ]:
```
#confusion metrics for model accuracy
from sklearn import metrics
cm= metrics.confusion_matrix(y_test,predictions)
cm
```

Out[ ]:
```
array([[28,  1,  3],
       [ 7,  1,  1],
       [ 7,  0,  1]], dtype=int64)
```

In [ ]:
```
#plotting confussion metrics
plt.figure(figsize=(9,9))
sns.heatmap(cm, annot=True,fmt= ".3f",linewidths=.5,square=True,cmap="Spectral");
plt.ylabel("actual output");
plt.xlabel("predicted output");
all_sample_title= "Accuracy Score : {0}".format(score)
plt.title("all_sample_title", size= 15);
```



7. Polynomial Regression

- Polynomial Regression is a regression algorithm that models the relationship between a dependent(y) and independent variable(x) as nth degree polynomial.

> y= b0+b1x1+ b2x12+ b2x13+..... bnx1n

- It is a linear model with some modification in order to increase the accuracy.
- The dataset used in Polynomial regression for training is of non-linear nature.
- It makes use of a linear regression model to fit the complicated and non-linear functions and datasets.

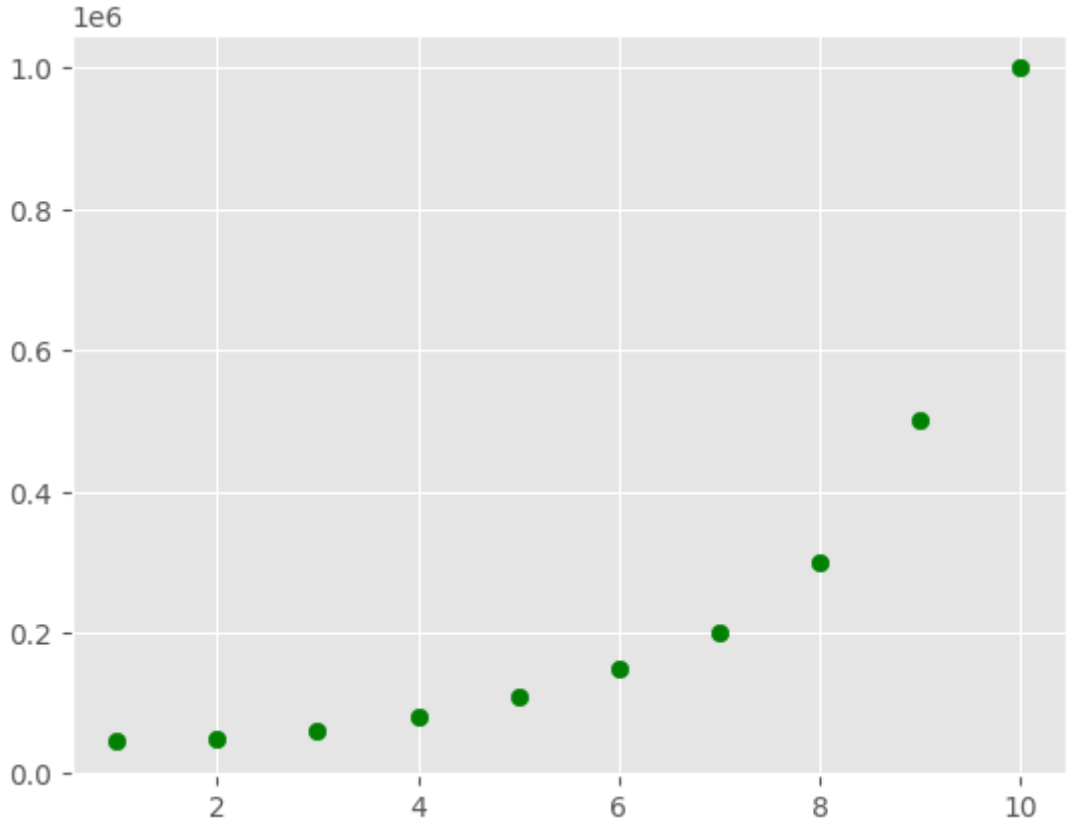
In [ ]:
```
# import libararies
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
# load data
df= pd.read_csv("position_salaries.csv")
df.head()
```

Out[ ]:

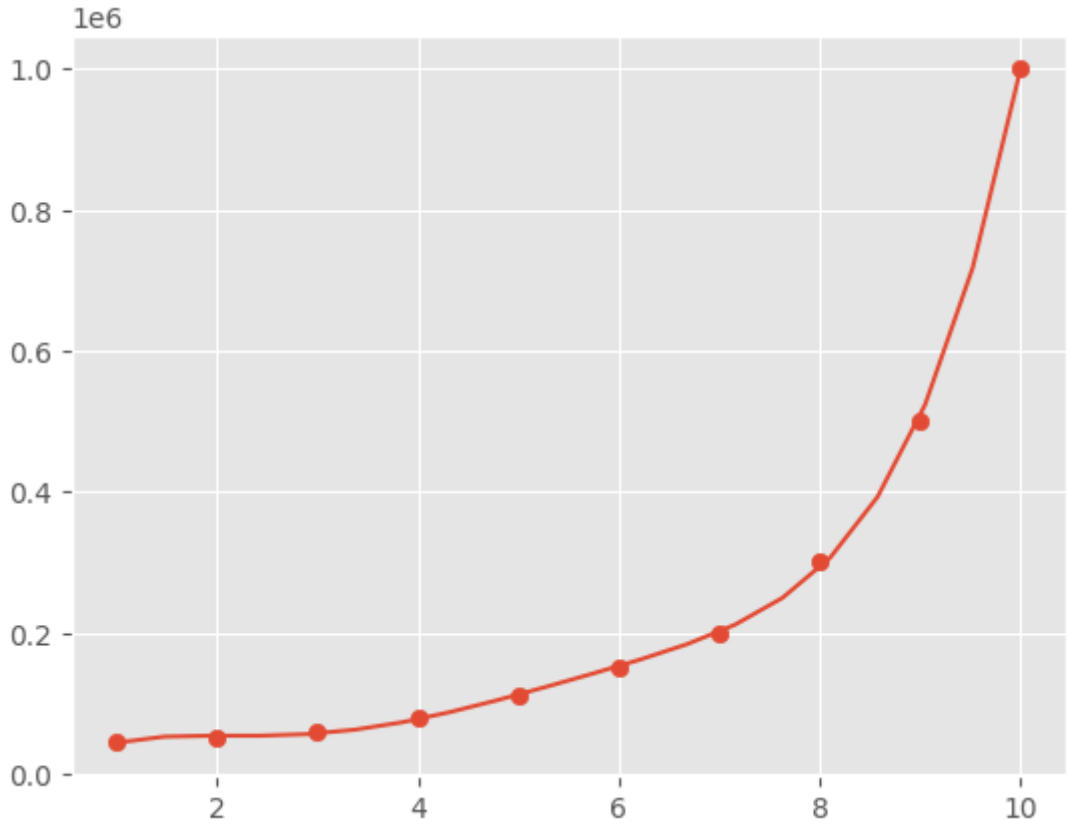
	Position	Level	Salary
0	Business Analyst	1	45000
1	Junior Consultant	2	50000
2	Senior Consultant	3	60000
3	Manager	4	80000
4	Country Manager	5	110000

In [ ]:
```
#assign x and y
X= df.iloc[ : ,1] #second column
y= df.iloc[ : ,-1] #last column
```

In [ ]:
```
#step 1 look into data by plotting
plt.scatter(X,y,color='green')
plt.show()
```



In [ ]:
```
#step 2 draw the line
mymodel= np.poly1d(np.polyfit(X,y ,5))
myline= np.linspace(1,10,20)
plt.scatter(X,y)
plt.plot(myline,mymodel(myline))
plt.show()
```



In [ ]:
```
#step3 r-squared
model= np.poly1d(np.polyfit(X,y ,5))
```

```
print(r2_score(y,model(X)))
```

```
0.9997969027099755
```

In [ ]:
```
#step 4 prediction (Salary for a person with 14 years of experience)
prediction = mymodel(14)
prediction
```

Out[ ]:
```
12233501.165501155
```

---

8. Naive Bayes Classifier

- It is a probabilistic classifier, which means it predicts on the basis of the probability of an object.
- Bayes theorem, named after Thomas Bayes from the 1700s. The Naive Bayes classifier works on the principle of conditional probability, as given by the Bayes theorem.

> Probability = P(A|B) = [P(B|A) * P(A)] / P(B)

- Where is Naive Bayes Used?

1. Face Recognition
2. Weather Prediction
3. Medical Diagnosis
4. News Classification
5. Classifying objects on the base of its features as its an Apple / Banana

- Advantages of Naive Bayes Classifier

1. It is simple and easy to implement
2. It doesn't require as much training data
3. It handles both continuous and discrete data
4. It is highly scalable with the number of predictors and data points
5. It is fast and can be used to make real-time predictions
6. It is not sensitive to irrelevant features

lets hands on!

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

In [ ]:
```
#load dataset
df = pd.read_csv("Social_Network_Ads.csv")
df.head()
```

Out[ ]:

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0

In [ ]:
```
#replacing male and females of gender column with 1 and 0 respectively.
df['Gender']= df['Gender'].replace('Male',1)
df['Gender']= df['Gender'].replace('Female',0)
df.head()
```

Out[ ]:

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	1	19	19000	0
1	15810944	1	35	20000	0
2	15668575	0	26	43000	0
3	15603246	0	27	57000	0
4	15804002	1	19	76000	0

In [ ]:
```
#assigning X and y
X=df.iloc[: , 1:4] #second, third and forth column
y=df.iloc[: , 4]  #last column
```

In [ ]:
```
# #looking into x and y
# y
# X
```

In [ ]:
```
#creat and fit model
from sklearn.naive_bayes import GaussianNB
model = GaussianNB().fit(X,y)
model
```

Out[ ]:
```
GaussianNB()
```

In [ ]:
```
#split data into test and train (80/20)%rule
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

#training model on training set
from sklearn.naive_bayes import GaussianNB
model = GaussianNB().fit(X_train,y_train)

# making predictions on the testing set
prediction= model.predict(X_test)
prediction
```

Out[ ]:
```
array([0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1,
       1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0,
       1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1,
       0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0], dtype=int64)
```

In [ ]:
```
#Accuracy score of model
from sklearn import metrics
```

```
score = metrics.accuracy_score(y_test,prediction)*100
print("Accuracy score of model = ", score)
```

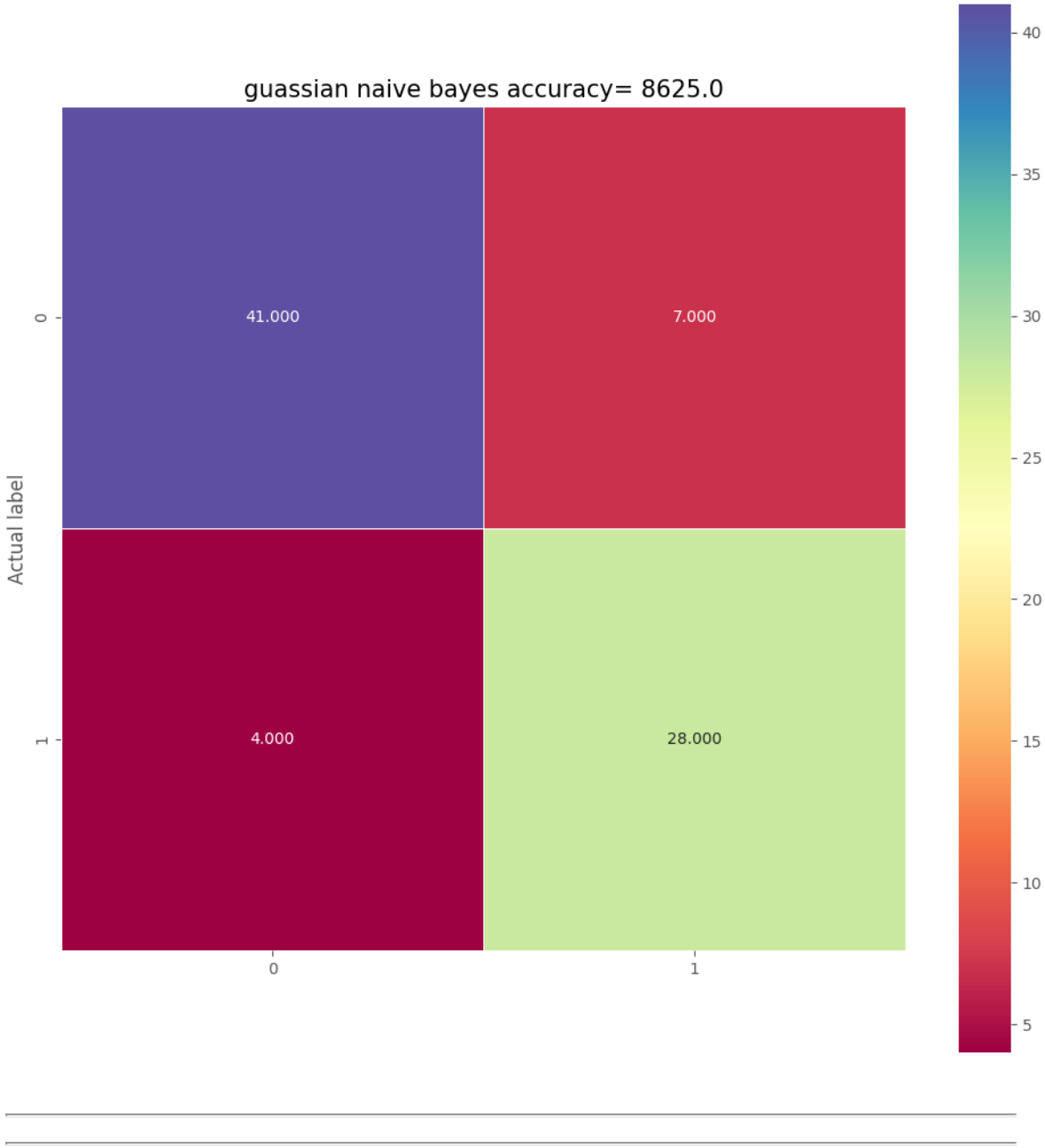
```
Accuracy score of model =  86.25
```

In [ ]:
```
#confusion metrix
from sklearn import metrics
cm = metrics.confusion_matrix(y_test,prediction)
cm
```

Out[ ]:
```
array([[41,  7],
       [ 4, 28]], dtype=int64)
```

In [ ]:
```
#confusion metrix plot
plt.figure(figsize=(12,12))
sns.heatmap(cm,annot=True,fmt=".3f",linewidths=.5,square=True,cmap = "Spectral")
plt.ylabel('Actual label')
all_sample_title= "guassian naive bayes accuracy= {0}".format(score*100)
plt.title (all_sample_title,size=15)
```

Out[ ]:
```
Text(0.5, 1.0, 'guassian naive bayes accuracy= 8625.0')
```



9. Support Vector Machine?

- SVM is a supervised machine learning algorithm that can be used for both classification or regression challenges. However, it is mostly used in classification problems.
- SVM algorithm, plot each data item as a point in n-dimensional space (where n is a number of features you have) with the value of each feature being the value of a particular coordinate. Then, we perform classification by finding the hyper-plane that differentiates the two classes very well.
- Support Vectors are simply the coordinates of individual observation. The SVM classifier is a frontier that best segregates the two classes (hyper-plane/ line).

In [ ]:
```
#import libararies
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
```

In [ ]:
```
#load dataset
df = pd.read_csv("fish.csv")
df.head()
```

Out[ ]:

	Species	Weight	Length1	Length2	Length3	Height	Width
0	Bream	242.0	23.2	25.4	30.0	11.5200	4.0200
1	Bream	290.0	24.0	26.3	31.2	12.4800	4.3056
2	Bream	340.0	23.9	26.5	31.1	12.3778	4.6961
3	Bream	363.0	26.3	29.0	33.5	12.7300	4.4555
4	Bream	430.0	26.5	29.0	34.0	12.4440	5.1340

In [ ]:
```
#assign X and y
X = df.drop(['Species'], axis = 'columns')
y = df.Species
```

In [ ]:
```
# #look into X and y
# X
# y
```

In [ ]:
```
#split data into test and train (80/20)%rule
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=

#import svm model
from sklearn import svm

#creat a svm classifier
clf= svm.SVC(kernel="linear") #linear kernal

#train the model using the training sets
clf.fit(X_train,y_train)

#predic th response
prediction= clf.predict(X_test)
prediction
```

Out[ ]:
```
array(['Bream', 'Smelt', 'Perch', 'Roach', 'Perch', 'Perch', 'Perch',
       'Pike', 'Bream', 'Perch', 'Perch', 'Pike', 'Perch', 'Perch',
       'Parkki', 'Roach', 'Roach', 'Pike', 'Perch', 'Bream', 'Parkki',
       'Whitefish', 'Perch', 'Pike', 'Parkki', 'Bream', 'Bream', 'Roach',
       'Perch', 'Smelt', 'Bream', 'Perch'], dtype=object)
```

In [ ]:
```
#Accuracy score of model
from sklearn import metrics
score = metrics.accuracy_score(y_test,prediction)*100
print("Accuracy score of model = ", score)
```

```
Accuracy score of model =  81.25
```

In [ ]:
```
#precision score of model
from sklearn import metrics
score = metrics.precision_score(y_test,prediction ,average='macro')*100
print("Precision score of model = ", score)
```

```
Precision score of model =  71.42857142857143
```

In [ ]:
```
#Recall score of model
from sklearn import metrics
score = metrics.recall_score(y_test,prediction ,average='macro')*100
print("Recall score of model = ", score)
```

```
Recall score of model =  77.14285714285714
```

1. **Accuracy** - Accuracy is the most intuitive performance measure and it is simply a ratio of correctly predicted observation to the total observations. One may think that, if we have high accuracy then our model is best.

   > `Accuracy = TP+TN/TP+FP+FN+TN`

2. **Precision** - Precision is the ratio of correctly predicted positive observations to the total predicted positive observations.

   > `Precision = TP/TP+FP`

3. **Recall** (Sensitivity) - Recall is the ratio of correctly predicted positive observations to the all observations in actual class.

   > `Recall = TP/TP+FN`

4. **F1 score** - F1 Score is the weighted average of Precision and Recall. Therefore, this score takes both false positives and false negatives into account. Intuitively it is not as easy to understand as accuracy, but F1 is usually more useful than accuracy.

   > `F1 Score = 2*(Recall * Precision) / (Recall + Precision)`