MACHINE LEARNING


```
Name:           Junaid Latif\
Designation:    Ph.D Scholar (China)\
Email:          hafizjunaidlatif@gmail.com\
Contact #:      +92-315-3411950\
```


In [ ]:

```

```

Simple Linear Regression

1. Relationship between two variables
2. Prediction on the basis of Relationships y= a+Bx

   y is dependent and x is independent (example of bridle dress) a = Constant / Intercept b = Function / Slope of x

In [ ]:

```
import pandas as pd
df =pd.read_csv('homeprices.csv')
df.head(2)
```

Out[ ]:

	area	price
0	2600	550000
1	3000	565000

Step-2 Splitting dataset into training data and testing data

In [ ]:

```
# X = df["YearsExperience"] # is 1d array
X = df[["area"]] # 2d array
y = df['price']
```

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 [ ]:

```
# Import Library and 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=0.2, random_state=0) # 0.2 means 1/5 or 20%
```

Step-3 Fit Linear Regression Model on training data

In [ ]:

```
from sklearn.linear_model import LinearRegression
model = LinearRegression()
model = model.fit(X_train, y_train)
# can do this in one line " model = LinearRegression().fit(X_train, Y_train)"
# here we will convert X into 2D array [[]] upper columns
model
```

Out[ ]:

```
LinearRegression()
```

Step-4 Ploting Scatter plot

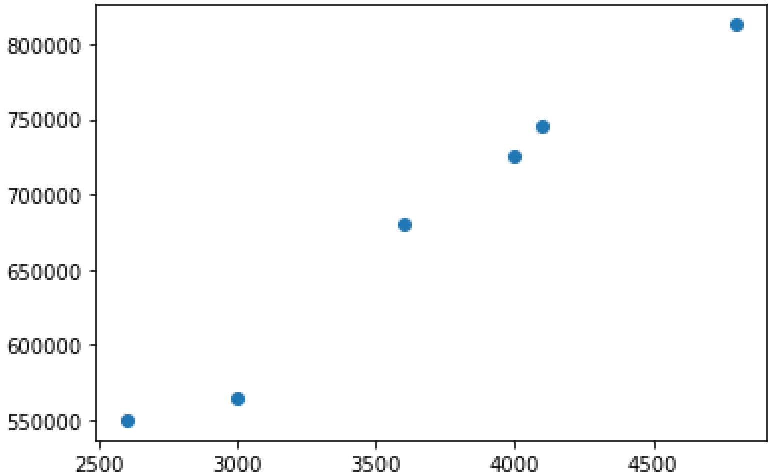
import matplotlib.pyplot as plt plt.scatter(X_train, Y_train)

In [ ]:

```
import matplotlib.pyplot as plt
plt.scatter(X_train, y_train)
```

Out[ ]:

```
<matplotlib.collections.PathCollection at 0x208ea7ab700>
```

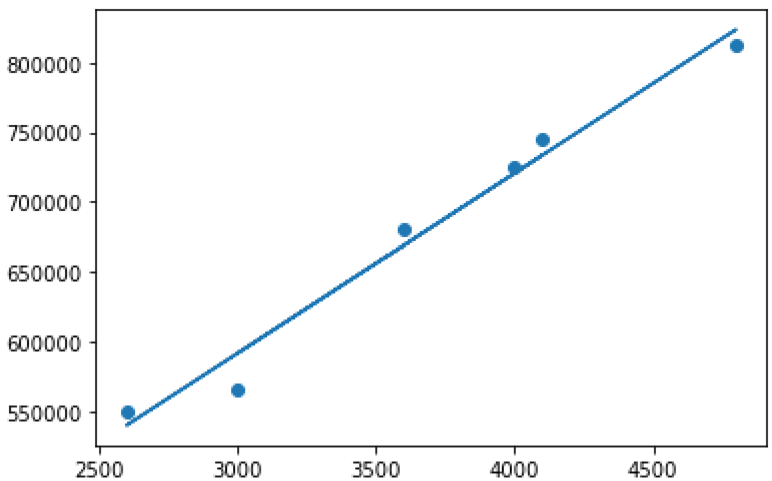


In [ ]:

```
import matplotlib.pyplot as plt

plt.scatter(X_train, y_train)
plt.plot(X_train, model.predict(X_train))
```

Out[ ]:

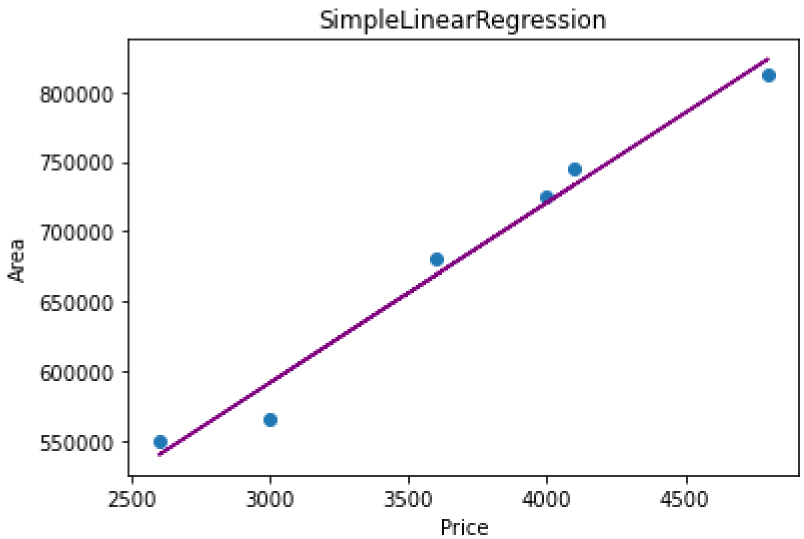
```
[<matplotlib.lines.Line2D at 0x208ea80dc10>]
```



In [ ]:

```
import matplotlib.pyplot as plt
plt.scatter(X_train, y_train)
plt.plot(X_train, model.predict(X_train), color="Purple")
plt.xlabel("Price")
```

```
plt.ylabel("Area")
plt.title("SimpleLinearRegression")
plt.show()
```



Step-5 Testing or Evaluating your model

In [ ]:

```
# Model Fitness
model.score(X_test, y_test)
```

Out[ ]: 0.9912837310746618

In [ ]:

```
print('Score for Training data = ', model.score(X_train, y_train))
print('Score for testing data = ', model.score(X_test, y_test))
```

```
Score for Training data =  0.9778653626410652
Score for testing data =  0.9912837310746618
```

Step-6 Prediction of unknown values

In [ ]:

```
print('Predict_1 = ' , model.predict([[3000]])) # used 2d array checked from error
print('Predict_X = ' , model.predict(X_test))
```

```
Predict_1 =  [591397.68542872]
```

```
Predict_X =  [785158.86375592 617232.50920568]
```

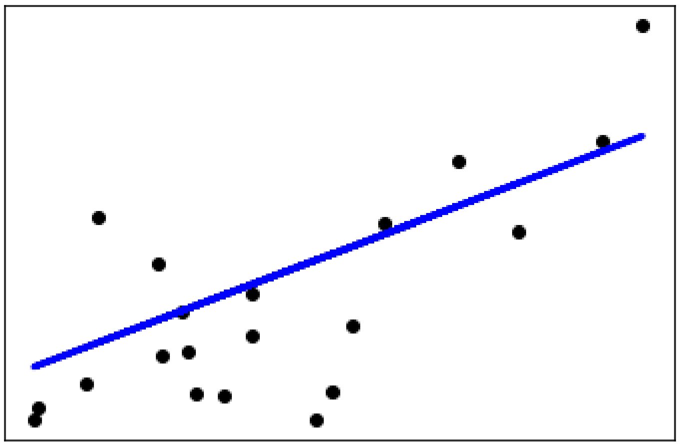
Example: Simple Linear Regression

In [ ]:

```
# Code source: Jaques Grobler
# License: BSD 3 clause

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn import datasets, linear_model
from sklearn.metrics import mean_squared_error, r2_score

# Load the diabetes dataset
diabetes_X, diabetes_y = datasets.load_diabetes(return_X_y=True)

# Use only one feature
diabetes_X = diabetes_X[:, np.newaxis, 2]

# Split the data into training/testing sets
diabetes_X_train = diabetes_X[:-20]
diabetes_X_test = diabetes_X[-20:]

# Split the targets into training/testing sets
diabetes_y_train = diabetes_y[:-20]
diabetes_y_test = diabetes_y[-20:]

# Create linear regression object
regr = linear_model.LinearRegression()

# Train the model using the training sets
regr.fit(diabetes_X_train, diabetes_y_train)

# Make predictions using the testing set
diabetes_y_pred = regr.predict(diabetes_X_test)

# The coefficients
print("Coefficients: \n", regr.coef_)
# The mean squared error
print("Mean squared error: %.2f" % mean_squared_error(diabetes_y_test, diabetes_y_pred))
# The coefficient of determination: 1 is perfect prediction
print("Coefficient of determination: %.2f" % r2_score(diabetes_y_test, diabetes_y_pred))

# Plot outputs
```

```
plt.scatter(diabetes_X_test, diabetes_y_test, color="black")
plt.plot(diabetes_X_test, diabetes_y_pred, color="blue", linewidth=3)

plt.xticks(())
plt.yticks(())

plt.show()
```

```
Coefficients:
 [938.23786125]
Mean squared error: 2548.07
Coefficient of determination: 0.47
```



Multiple linear Regression

Input data or independent variables or features\ Outputdata or dependent variables or prediction\ Equation: salary = i1 *age + i2* distance + i3 * yearsexperience + b\ I are coefficient and b in intercept\ Using scikit learn library

In [ ]:

```
# import library
import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression
```

In [ ]:

```
# import data
```

```
df = pd.read_csv('ml_salary_data.csv')
df.head()
```

Out[ ]:

	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 [ ]:

```
##assigning x and y from the dataset
x = df [['age', 'distance', 'YearsExperience']]
y = df ['Salary']
```

In [ ]:

```
# creat and fit model
model = LinearRegression().fit(x,y)
model
```

Out[ ]:

```
LinearRegression()
```

In [ ]:

```
model.coef_   # i are coefficients
```

Out[ ]:

```
array([-2.68055892e+15,  1.06092560e+15,  2.82449143e+13])
```

In [ ]:

```
model.intercept_   # b is intercept
```

Out[ ]:

```
847347429532075.5
```

In [ ]:

```
# 'open', 'close', 'low', 'high'
response= model.predict([['32.2','80.50','2.2']])
response
# 32.2 is age, 80.5 is distance , 2.2 is experience
```

```
C:\Users\Junaid\anaconda3\lib\site-packages\sklearn\utils\validation.py:63: FutureWarning: Arrays of bytes/strings is bei
ng converted to decimal numbers if dtype='numeric'. This behavior is deprecated in 0.24 and will be removed in 1.1 (renam
ing of 0.26). Please convert your data to numeric values explicitly instead.
  return f(*args, **kwargs)
```

Out[ ]:
```
array([46614.75])
```

Efficacy of model

In [ ]:
```python
model.predict([[31.1, 77.75, 1.1]])   # any required data we can predict
```

Out[ ]:
```
array([36217.125])
```

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

Out[ ]:
```
0.9569520722791693
```

In [ ]:
```python
# how it predicted this price 36217.125? we will see here from equation
(((-2.68055892e+15) * (31.1)) + ((1.06092560e+15) * (77.75)) + ((2.82449143e+13) * (1.1)) + (847347429532075.5))
```

Out[ ]:
```
-176737924.5
```

Assignments:\

- how to plot multiple linear regression model?\
- how to test the efficiecy of model?

Graphically representation of multiple linear regression is not possible until we have 4 dimentional plot it is very trickey and maximum 3 treatments. We will have to use different plot to represent our data, ie x1 with y , x2 with y. But accuracy score and other things can be test on all data. But still we used some examples for other dataset to check these in different species.

In [ ]:
```python
df = pd.read_csv('ml_salary_data.csv')
x = df [['age', 'distance', 'YearsExperience']]
x1= df[['age']]
x2= df[['distance']]
x3= df[['YearsExperience']]
y = df[['Salary']]
model = LinearRegression().fit(x1,y)
```

In [ ]:

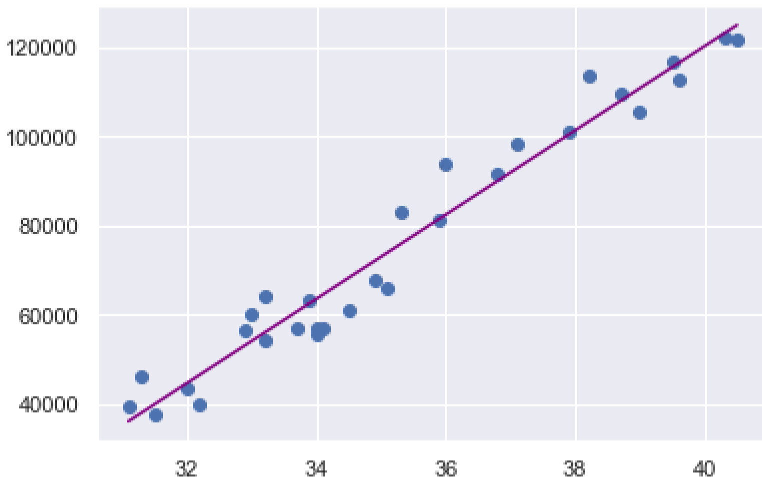
```
df.head()
# x1.head()
```

Out[ ]:

	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 [ ]:

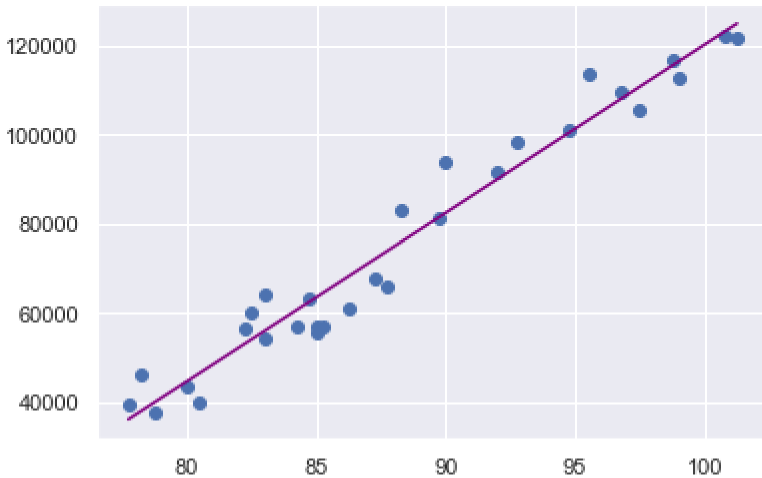
```
import matplotlib.pyplot as plt
plt.scatter(x1, y)
plt.plot(x1, model.predict(x1), color="Purple")
# plt.xlabel("yearsExperience")
# plt.ylabel("salary")
# plt.title("Magic of train set")
plt.show()
```



In [ ]:

```
modelx2 = LinearRegression().fit(x2,y)
import matplotlib.pyplot as plt
```

```
plt.scatter(x2, y)
plt.plot(x2, modelx2.predict(x2), color="Purple")
# plt.xlabel("yearsExperience")
# plt.ylabel("salary")
# plt.title("Magic of train set")
plt.show()
```



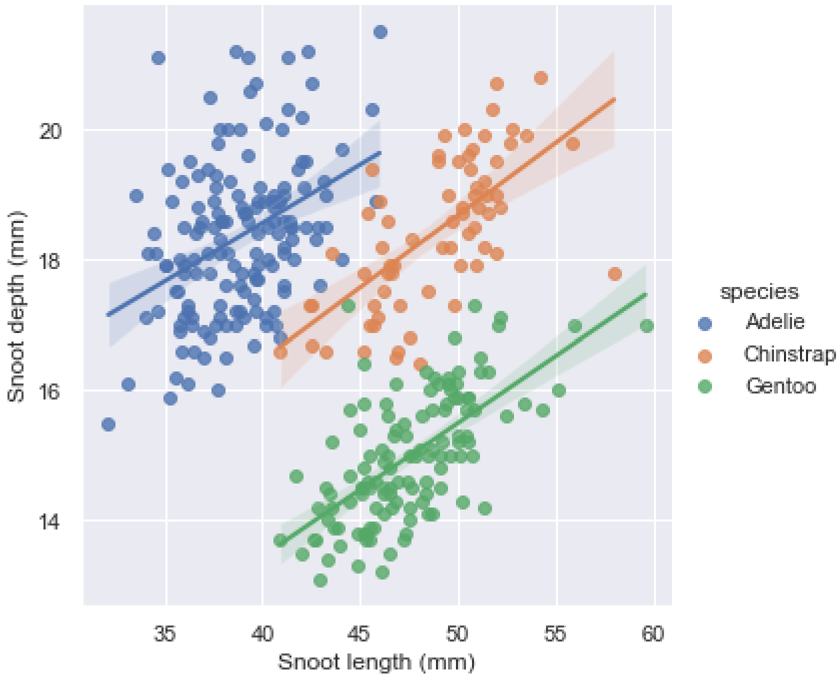
Example:

In [ ]:

```
import seaborn as sns
sns.set_theme()

# Load the penguins dataset
penguins = sns.load_dataset("penguins")

# Plot sepal width as a function of sepal_length across days
g = sns.lmplot(
    data=penguins,
    x="bill_length_mm", y="bill_depth_mm", hue="species",
    height=5
)

# Use more informative axis labels than are provided by default
g.set_axis_labels("Snoot length (mm)", "Snoot depth (mm)")
```

Out[ ]:

```
<seaborn.axisgrid.FacetGrid at 0x208ec657f10>
```

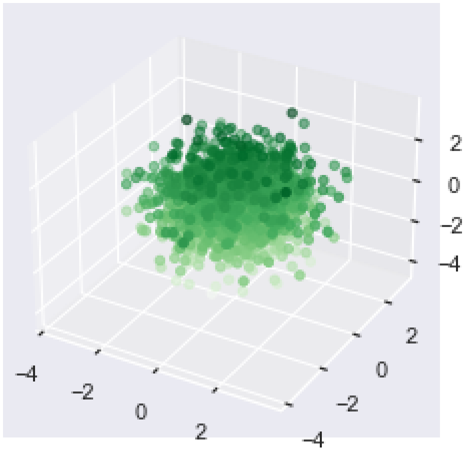


In [ ]:

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D


#create some test data and simulate results
x_data = np.random.randn(2000,3)
w_real = [0.3,0.5,0.1]
b_real = -0.2

noise = np.random.randn(1,2000)*0.1
y_data = np.matmul(w_real,x_data.T) + b_real + noise

print(len(x_data))
print(len(y_data[0]))

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

x1 = x_data[:,0]
x2 = x_data[:,1]
```

```
x3 = x_data[:,2]
ax.scatter3D(x1, x2, x3, c=x3, cmap='Greens');

plt.show()
```

```
2000
2000
```



Decission Tree Classification

In [ ]:

```
# import libraries
import pandas as pd
df= pd.read_csv('biryani.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

Convert Gender into dummies variables \ ***With replace function*** Male = 1 , Female = 0

In [ ]:
```
df['gender'] = df['gender'].replace('Male',1)
df['gender'] = df['gender'].replace('Female',0)
df.head()
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

Likeness is catagorical or discrete variable we have to define it we use desission tree

In [ ]:
```
# Selection of input and output variable
x = df[['weight', 'gender']]
y= df['likeness']
```

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

Out[ ]:

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

In [ ]:
```
# machine learning algrothim
from sklearn.tree import DecisionTreeClassifier
```

```
# creat and fit our model
model = DecisionTreeClassifier()
model.fit(x,y)
# or use one line code " model = DecisionTreeClassifier().fit(x,y) ""

# prediction ... age 80 , 1 male
model.predict([[80, 1]])
```

Out[ ]:

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

In [ ]:

```
# prediction ... age 30 , 0 female
model.predict([[30, 0]])
```

Out[ ]:

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

In [ ]:

```
# how to measure accuracy of our model
## split data into test and train rule
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

x_train, x_test, y_train, y_test = train_test_split(x,y, test_size=0.2, random_state=1) # 80% training data and 20 % test
# random_stat = 1 or 0 , if we add this score value will eb fixed every time otherwise it would be random scores every ti

# creat a model
model = DecisionTreeClassifier()

# fitting a model
model.fit(x_train, y_train)
predicted_values = model.predict(x_test)
predicted_values


# checking score
# y_test = actual_values :  but we cannot write this

score = accuracy_score(y_test, predicted_values)
score
```

Out[ ]:

```
0.5102040816326531
```

Assignment

Accuracy score check on linear and MLR

In [ ]:
```
# how to train and save our model

import pandas as pd
from sklearn.tree import DecisionTreeClassifier
import joblib
model = DecisionTreeClassifier().fit(x,y)   # here we will made a model on whole dataset , while above we did 80/20 data
joblib.dump(model, "foodie.joblib") # this command is for store/save model  # joblib is extenion of model without this we
```

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

In [ ]:
```
# how to run a stores or saved model on our data?
```

In [ ]:
```
# # how to load our model

# joblib.load('foodie.joblib')
```

In [ ]:
```
# what is decission tree classifier

# graph
from sklearn import tree
model = DecisionTreeClassifier().fit(x,y)

# graphic evulation

tree.export_graphviz(model,
                        out_file='foodiee.dot',
                        feature_names=['age','gender'],
                        class_names=sorted(y.unique()),
                        label='all',
                        rounded=True,
                        filled=True)

  # y.unique mean sort all unique values in y
```

Another practice

In [ ]:

```
import pandas as pd
import numpy as np
import seaborn as sns
df = sns.load_dataset("iris")
df.head()
```

Out[ ]:

	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 [ ]:

```
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeClassifier
X= df.iloc[ : ,:-1] # all rows + all columns select except last one
y= df.iloc[ : ,-1:]
```

In [ ]:

```
X.head()
```

Out[ ]:

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

In [ ]:

```
y.head()
```

Out[ ]:

	species
0	setosa
1	setosa
2	setosa
3	setosa
4	setosa

In [ ]:

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import plot_tree

model = DecisionTreeClassifier().fit(X,y)
plot_tree(model, filled=True)
plt.title("decision tree trained model of IRIS data")

# how to save this plot in tif, png, and pdf files, in HD quality?
#plt.savefig("DecessionTree.png", dpi=300)
plt.savefig("tiff_compressed.tiff", dpi=600, format="tiff", facecolor='white', edgecolor='none', pil_kwargs={"compressior
plt.show()
```

Assignment\ check accuracy, for this we will divide data into 20/30 , 30/70, 10/90. \ Any unknown sample value should be 5-10 make prediction.

DecisionTree at whole data set

In [ ]:

```
# import libraries
import pandas as pd
df= pd.read_csv('biryani.csv')
df['gender'] = df['gender'].replace('Male',1)
df['gender'] = df['gender'].replace('Female',0)

x = df[['weight', 'age', 'height','gender']]
y= df['likeness']

# machine learning algrothim
from sklearn.tree import DecisionTreeClassifier

# creatand fit our model
model = DecisionTreeClassifier()
```

```
model.fit(x,y)

# # prediction ... age 80 , 1 male
model.predict([[45, 160, 50, 1]])
```

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

In [ ]:
```
# how to measure accuracy of our model
## split data into test and train rule

from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

x_train, x_test, y_train, y_test = train_test_split(x,y, test_size=0.2, random_state=1) # 80% training data and 20 % test
# random_stat = 1 or 0 , if we add this score value will eb fixed every time otherwise it would be random scores every ti

# creat a model
model = DecisionTreeClassifier()

# fitting a model
model.fit(x_train, y_train)
predicted_values = model.predict(x_test)
predicted_values

# checking score
score = accuracy_score(y_test, predicted_values)
score
```

Out[ ]:
```
0.46938775510204084
```

Random Forest Classification

In [ ]:
```
#load sample data set
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
df = sns.load_dataset("iris")
```

```
df1 = sns.load_dataset("iris")
df.head()
```

Out[ ]:

	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 [ ]:

```
from sklearn.tree import DecisionTreeClassifier
X= df.iloc[ : ,:-1]
y= df.iloc[ : ,-1:]
```

In [ ]:

```
from sklearn.ensemble import RandomForestClassifier
model= RandomForestClassifier(n_estimators=100)  # The number of trees in the forest
model.fit(X,y)
model.predict([[10,4,2,6]])
```

```
C:\Users\Junaid\AppData\Local\Temp/ipykernel_4440/2584235027.py:3: DataConversionWarning: A column-vector y was passed wh
en a 1d array was expected. Please change the shape of y to (n_samples,), for example using ravel().
  model.fit(X,y)
```

Out[ ]:

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

In [ ]:

```
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_state=0)
predictions = model.predict(X_test)
predictions
```

Out[ ]:

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

In [ ]:

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

```
score of accuracy is:  1.0
```

In [ ]:

```
from sklearn import metrics
print("Accuracy =",metrics.accuracy_score(y_test,predictions))
```

```
Accuracy = 1.0
```

In [ ]:

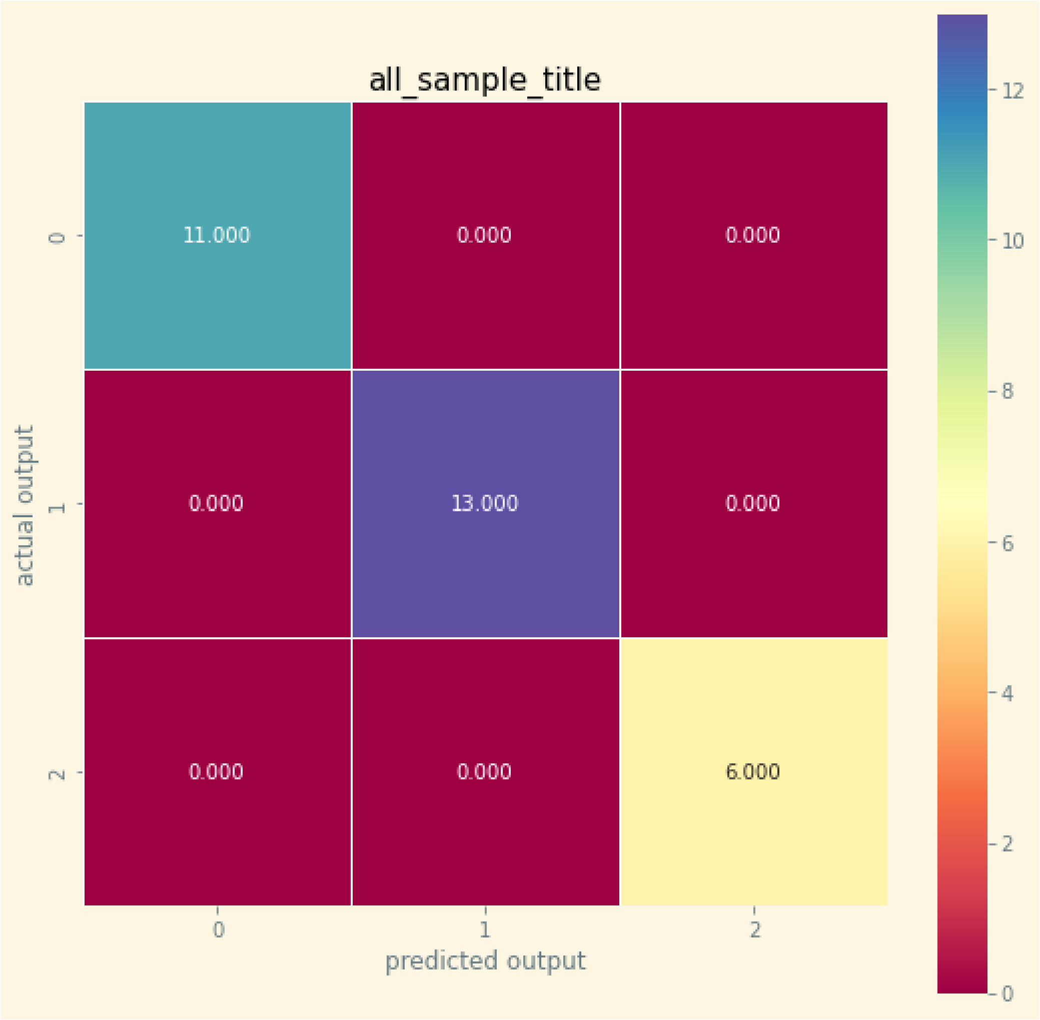
```
from sklearn import metrics
cm= metrics.confusion_matrix(y_test,predictions)
cm
```

Out[ ]:

```
array([[11,  0,  0],
       [ 0, 13,  0],
       [ 0,  0,  6]], dtype=int64)
```

In [ ]:

```
plt.style.use('Solarize_Light2')
import seaborn as sns
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);
```



Classification vs Regression

Classification predictive modeling problems are different from regression predictive modeling problems.

Classification is the task of predicting a discrete class label.\ Regression is the task of predicting a continuous quantity.\ There is some overlap between the algorithms for classification and regression; for example:

A classification algorithm may predict a continuous value, but the continuous value is in the form of a probability for a class label. A regression algorithm may predict a discrete value, but the discrete value in the form of an integer quantity.

Example of Regressor at Salary data

In [ ]:

```
# import libararies
import numpy as np
import pandas as pd
import seaborn as sns
import scipy as sc
import matplotlib.pyplot as plt
```

In [ ]:

```
#import data
sd = pd.read_csv('ml_salary_data.csv')
X= sd.iloc[ : ,:-1]
y= sd.iloc[ : ,-1:]
sd.head(5)
```

Out[ ]:

	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 [ ]:

```
y.head()
```

Out[ ]:

	Salary
0	39343
1	46205
2	37731
3	43525
4	39891

In [ ]:
```
from sklearn.ensemble import RandomForestRegressor
model= RandomForestRegressor(n_estimators=100)
model.fit(X,y)
```

```
C:\Users\Junaid\AppData\Local\Temp/ipykernel_4440/4160705919.py:3: DataConversionWarning: A column-vector y was passed wh
en a 1d array was expected. Please change the shape of y to (n_samples,), for example using ravel().
  model.fit(X,y)
```

Out[ ]:
```
RandomForestRegressor()
```

In [ ]:
```
model.predict([[30,80,2]])
```

Out[ ]:
```
array([41913.26])
```

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

Out[ ]:

	species
137	virginica
84	versicolor
27	setosa
127	virginica
132	virginica

In [ ]:
```
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_state=0)
predictions = model.predict(X_test)
predictions
```

Out[ ]:
```
array([ 40598.68      , 121460.01      ,  56993.0975    ,  60843.81333333,
       114851.81      , 108625.5       ])
```

In [ ]:
```
score = model.score(X_test,y_test)
print("score of accuracy is: ",score)
```

```
score of accuracy is:  0.9969453515010206
```

Regression Example with RandomForestRegressor in Python

Random forest is an ensemble learning algorithm based on decision tree learners. The estimator fits multiple decision trees on randomly extracted subsets from the dataset and averages their prediction.

```
Scikit-learn API provides the RandomForestRegressor class included in ensemble module to implement the
random forest for regression problem.

We'll briefly learn how to fit and predict regression data by using the RandomForestRegressor class in
Python. It will covers:
```

- Preparing the data
- Training the model
- Predicting and accuracy check
- Boston dataset prediction

Import libraries and dataset

In [ ]:

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.datasets import load_boston
from sklearn.datasets import make_regression
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
# from sklearn.preprocessing import scale
import matplotlib.pyplot as plt
from sklearn import set_config
import numpy as np
import pandas as pd
import seaborn as sns
import scipy as sc
```

In [ ]:

```
#import data
sd = pd.read_csv('ml_salary_data.csv')
X= sd.iloc[ : ,:-1]
y= sd.iloc[ : ,-1:]
sd.head(5)
```

Out[ ]:

	age	distance	YearsExperience	Salary
0	31.1	77.75	1.1	39343

	age	distance	YearsExperience	Salary
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

Preparing the dataset

In [ ]:
```
from sklearn.ensemble import RandomForestRegressor
model= RandomForestRegressor(n_estimators=100)
model.fit(X,y)
```

```
C:\Users\Junaid\AppData\Local\Temp/ipykernel_4440/4160705919.py:3: DataConversionWarning: A column-vector y was passed wh
en a 1d array was expected. Please change the shape of y to (n_samples,), for example using ravel().
  model.fit(X,y)
```

Out[ ]:
```
RandomForestRegressor()
```

Split data into training and testing

In [ ]:
```
X_train, X_test, y_train, y_test= train_test_split(X ,y, test_size=1/5, random_state=0)
predictions = model.predict(X_test)
predictions
```

Out[ ]:
```
array([ 40143.85      , 120974.76      ,  57033.71833333,  61026.145     ,
       114542.84      , 108884.98      ])
```

In [ ]:
```
# Model Score and Accuracy and Prediction
score = model.score(X_test,y_test)
print("score of accuracy is: ",score)
```

```
score of accuracy is:  0.9971064488216127
```

In [ ]:
```
model.predict([[30,80,2]])
```

Out[ ]:
```
array([42237.69])
```

Training model and print to see

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

```
RandomForestRegressor()
```

In [ ]:
```
# fit the model on train data and check the model accuracy score.
model.fit(X_train, y_train)

score = model.score(X_train, y_train)
print("R-squared:", score)
```

```
R-squared: 0.9878820657479063
```

```
C:\Users\Junaid\AppData\Local\Temp/ipykernel_4440/2109200087.py:2: DataConversionWarning: A column-vector y was passed wh
en a 1d array was expected. Please change the shape of y to (n_samples,), for example using ravel().
  model.fit(X_train, y_train)
```

Predicting and accuracy check

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

mse = mean_squared_error(y_test, predictions)
print("MSE: ", mse)
print("RMSE: ", mse*(1/2.0))
```

```
MSE:  25217852.339958936
RMSE:  12608926.169979468
```

Visualize the original and predicted data in a plot

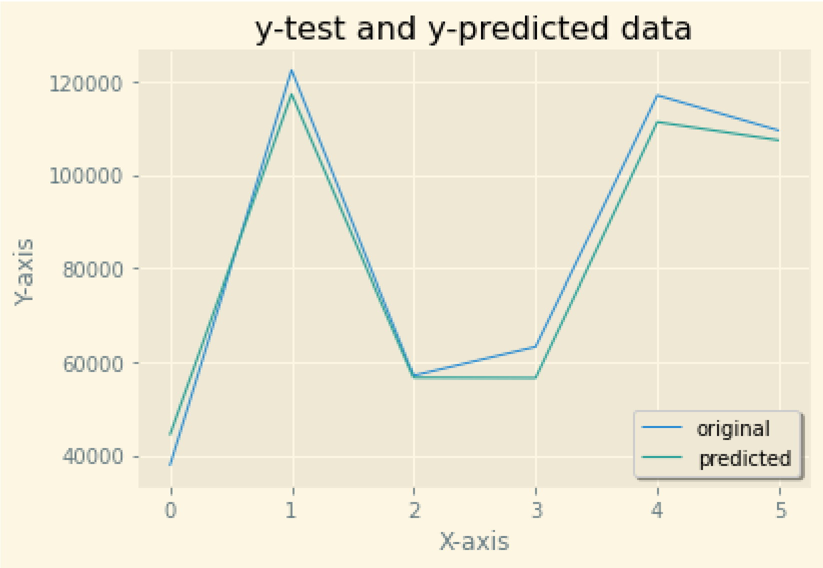
In [ ]:
```
y_test
```

Out[ ]:

	Salary
2	37731
28	122391
13	57081
10	63218
26	116969
24	109431

In [ ]:

```
x_ax = range(len(y_test))
plt.plot(x_ax, y_test, linewidth=1, label="original")
plt.plot(x_ax, predictions, linewidth=1.1, label="predicted")
plt.title("y-test and y-predicted data")
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.legend(loc='best',fancybox=True, shadow=True)
plt.grid(True)
plt.show()
```

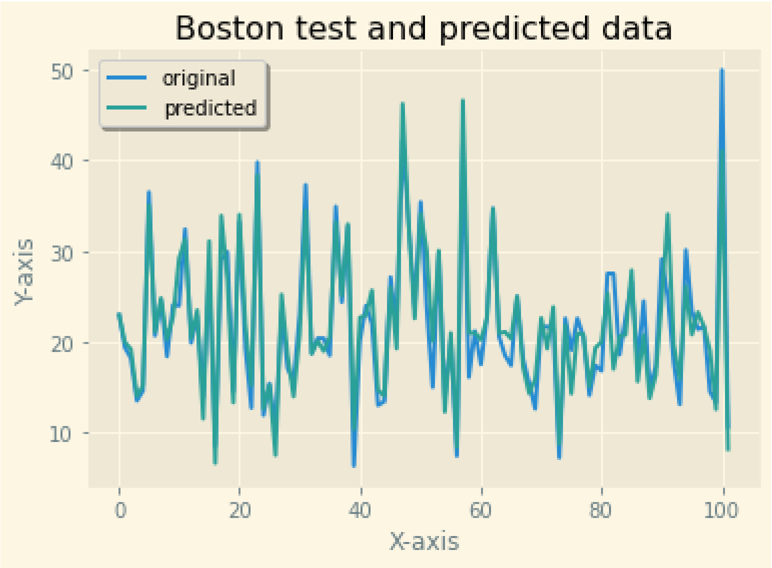


In [ ]:

```
print("Boston housing dataset prediction.")
boston = load_boston()
X, y = boston.data, boston.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=1/5)

model = RandomForestRegressor()
model.fit(X_train, y_train)

score = model.score(X_train, y_train)
print("R-squared:", score)

predictions = model.predict(X_test)
```

```
mse = mean_squared_error(y_test, predictions)
print("MSE: ", mse)
print("RMSE: ", mse*(1/2.0))

x_ax = range(len(y_test))
plt.plot(x_ax, y_test, label="original")
plt.plot(x_ax, predictions, label="predicted")
plt.title("Boston test and predicted data")
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.legend(loc='best',fancybox=True, shadow=True)
plt.grid(True)
plt.show()
```

```
Boston housing dataset prediction.
R-squared: 0.982954725898816
MSE:  8.053077647058817
RMSE:  4.026538823529409
```



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 librarya and dataset

import pandas as pd
df= pd.read_csv("biryani.csv")
df['gender'] = df['gender'].replace('Male',1)
df['gender'] = df['gender'].replace('Female',0)
```

In [ ]:

```
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 [ ]:

```
# selection of input and output variable

X = df[['weight', 'gender']]
Y = df['likeness']
```

In [ ]:

```
X.head()
```

Out[ ]:

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

In [ ]:

```
Y.head()
```

Out[ ]:

```
0    Biryani
1    Biryani
2    Biryani
3    Biryani
4    Biryani
Name: likeness, dtype: object
```

In [ ]:

```
# model and prediction
from sklearn.neighbors import KNeighborsClassifier
model = KNeighborsClassifier (n_neighbors=5)  # K points 3, 5, 7 etc

# train the model using the training sets
model.fit(X,Y)

#predict output
Predicted = model.predict([[70,1]])  # 70: weight , 1: female
Predicted
```

Out[ ]:

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

In [ ]:

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

In [ ]:

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

from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
```

```
X_train, X_test, Y_train, Y_test = train_test_split(X,Y, test_size=0.2) # 80% training data and 20% test

# creat a model
model = KNeighborsClassifier()
# fitting model
model._fit(X_train,Y_train)

Predicted_values = model.predict(X_test)
Predicted_values

# checking score
# Y-test = actual_values

score = accuracy_score(Y_test, Predicted_values)
print(" The accuracy score for our model is =", score)

# everytime it accuracy score will be changed or random ,, here we can change K number 3/5/7 etc
```

```
The accuracy score for our model is = 0.6530612244897959
```

PolyNomila Regression

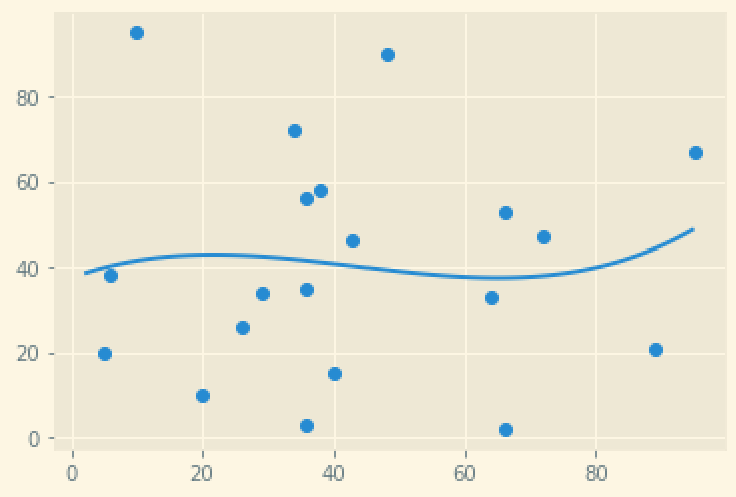
Example of bad fit plot line

In [ ]:

```
# import libs
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
# load data and assign x and y
X= [89,43,36,36,95,10,66,34,38,20,26,29,48,64,6,5,36,66,72,40]
y= [21,46,3,35,67,95,53,72,58,10,26,34,90,33,38,20,56,2,47,15]
```

In [ ]:

```
#look into it
mymodel= np.poly1d(np.polyfit(X,y ,3))
myline= np.linspace(2,95,100)
plt.scatter(X,y)
plt.plot(myline,mymodel(myline))
plt.show()
```



In [ ]:

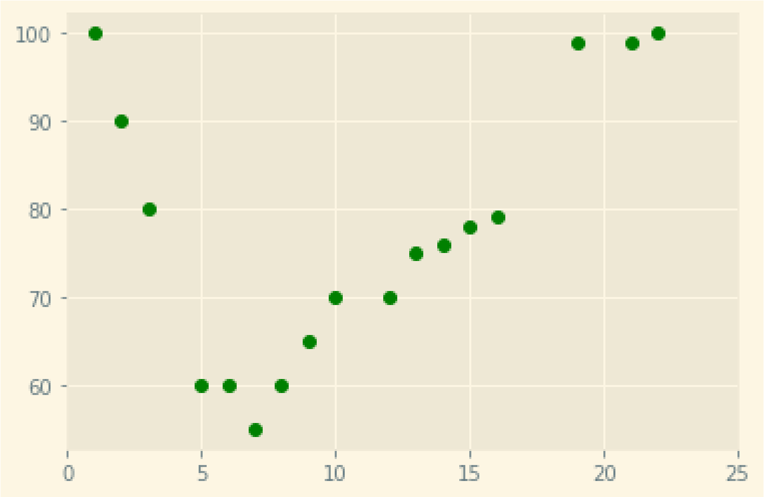
```
#r squared for bad fit
from sklearn.metrics import r2_score
X= [89,43,36,36,95,10,66,34,38,20,26,29,48,64,6,5,36,66,72,40]
y= [21,46,3,35,67,95,53,72,58,10,26,34,90,33,38,20,56,2,47,15]
model= np.poly1d(np.polyfit(X,y ,3))
print(r2_score(y,model(X)))
```

```
0.009952707566680652
```

polynomial regression with numpy

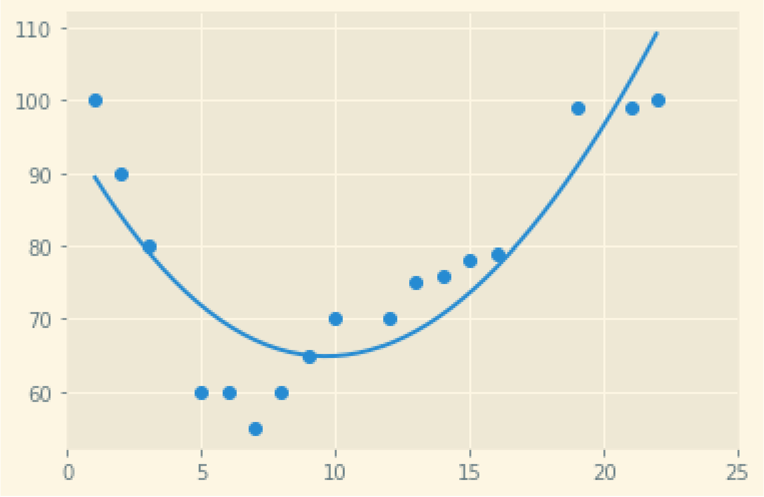
In [ ]:

```
#step 1 look in to data

X = [1,2,3,5,6,7,8,9,10,12,13,14,15,16,178,19,21,22]
y=[100,90,80,60,60,55,60,65,70,70,75,76,78,79,90,99,99,100]

plt.scatter(X,y,color='green')
plt.xlim(0,25)
plt.show()
```



In [ ]:

```
#step 2 draw the line
mymodel= np.poly1d(np.polyfit(X,y ,3))
myline= np.linspace(1,22,50)
plt.scatter(X,y)
plt.plot(myline,mymodel(myline))
plt.xlim(0,25)
plt.show()
```



In [ ]:

```
#step3 r-squared
model= np.poly1d(np.polyfit(X,y ,3))
print(r2_score(y,model(X)))
```

```
0.7855684300264448
```

In [ ]:
```
#step 4 prediction
speed = mymodel(18)
print(speed)
```

```
85.74417683003881
```

Another Example

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


dataset= pd.read_csv('https://s3.us-west-2.amazonaws.com/public.gamelab.fun/dataset/position_salaries.csv')
X= dataset.iloc[:, 1:2].values
y= dataset.iloc[:, 2].values
```

In [ ]:
```
dataset.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

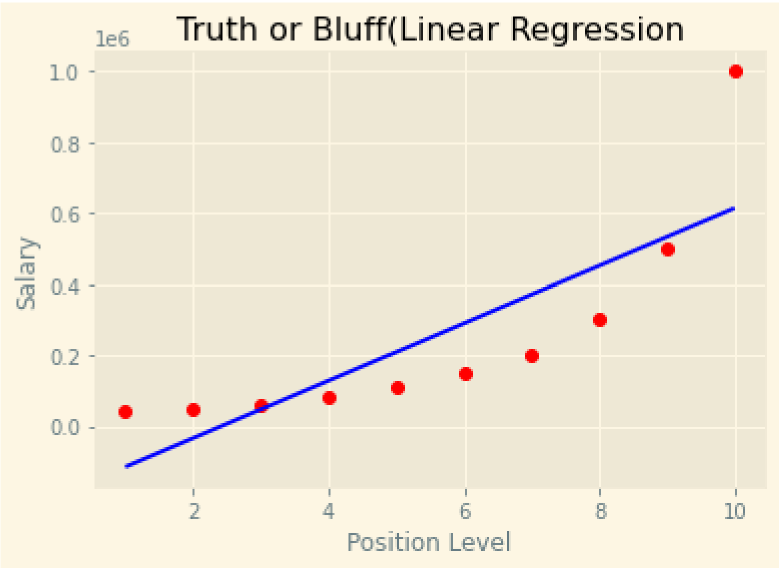
In [ ]:
```
# Split the dataset into training set and test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test =train_test_split(X,y,test_size=0.2, random_state=0)
```

In [ ]:
```
# Fitting Linear Regression to thr dataset
from sklearn.linear_model import LinearRegression
```

```
lin_reg = LinearRegression()
lin_reg.fit(X,y)
```

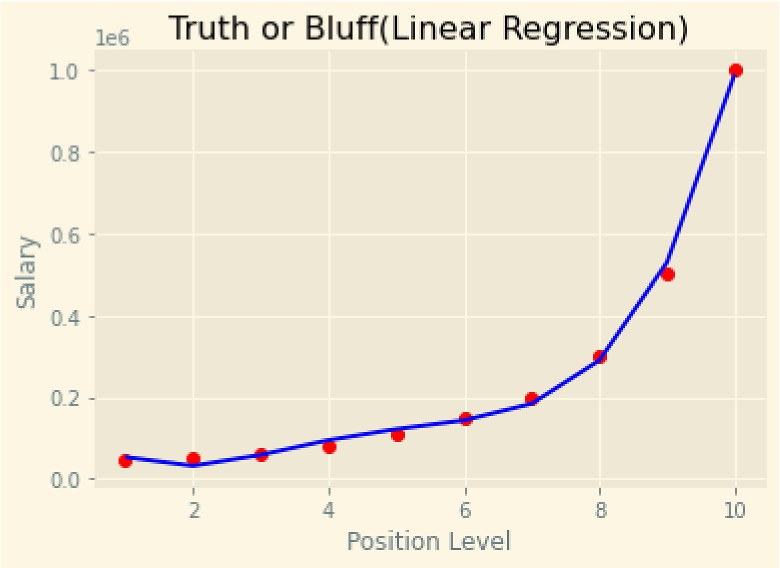
Out[ ]:
```
LinearRegression()
```

In [ ]:
```
#visualizing the linear regression results

def viz_linear():
    plt.scatter(X,y,color="red")
    plt.plot(X,lin_reg.predict(X),color="blue")
    plt.title("Truth or Bluff(Linear Regression")
    plt.xlabel('Position Level')
    plt.ylabel('Salary')
    plt.show()
    return
viz_linear()
```



In [ ]:
```
#fitting polynomial regression to the dataset
from sklearn.preprocessing import PolynomialFeatures
poly_reg = PolynomialFeatures(degree=4)
X_poly= poly_reg.fit_transform(X)
pol_reg= LinearRegression()
pol_reg.fit(X_poly,y)
```

```
#visualizing the polynomial regression results

def viz_polynomial():
    plt.scatter(X,y,color="red")
    plt.plot(X, pol_reg.predict(poly_reg.fit_transform(X)),color="blue")
    plt.title("Truth or Bluff(Linear Regression)")
    plt.xlabel('Position Level')
    plt.ylabel('Salary')
    plt.show()
    return
viz_polynomial()
```



In [ ]:

```
# Predict a new result with linear regression
pred_linear = lin_reg.predict([[11]])
```

In [ ]:

```
# Predict a new result with polynomial regression
pred_polynomial = pol_reg.predict(poly_reg.fit_transform([[11]]))
```

In [ ]:

```
print('Linear Regressionresult = ', pred_linear)
print('Linear Regressionresult = ', pred_polynomial)

print('Difference is =', pred_linear - pred_polynomial)
```

```
Linear Regressionresult =  [694333.33333333]
Linear Regressionresult =  [1780833.33333284]
Difference is = [-1086499.99999951]
```

Logistic Regression

Predicting if a person would buy life insurnace based on his age using logistic regression\ Above is a binary logistic regression problem as there are only two possible outcomes (i.e. if person buys insurance or he/she doesn't).

In [ ]:

```
import pandas as pd
from matplotlib import pyplot as plt
%matplotlib inline
df = pd.read_csv("insurance_data.csv")
df.head()
```

Out[ ]:

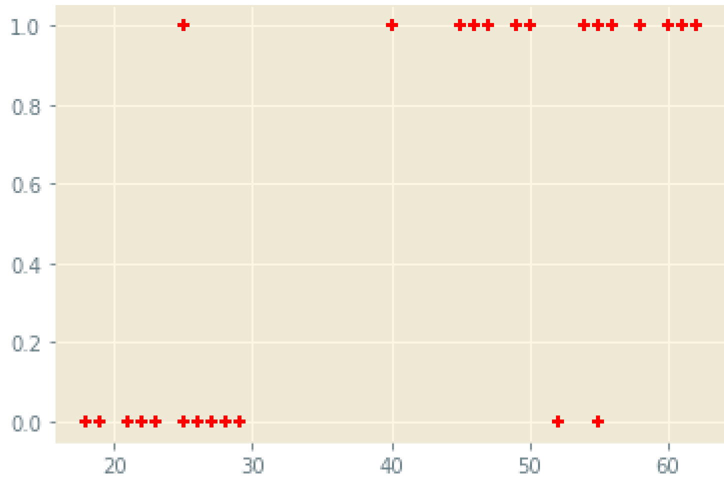
	age	bought_insurance
0	22	0
1	25	0
2	47	1
3	52	0
4	46	1

In [ ]:

```
plt.scatter(df.age,df.bought_insurance,marker='+',color='red')
```

Out[ ]:

```
<matplotlib.collections.PathCollection at 0x2ba316b1f40>
```



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

In [ ]:
```
X_train, X_test, y_train, y_test = train_test_split(df[['age']],df.bought_insurance,train_size=0.8)
```

In [ ]:
```
X_test
```

Out[ ]:

	age
16	25
2	47
4	46
8	62
14	49
18	19

In [ ]:
```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
```

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

Out[ ]:
```
LogisticRegression()
```

In [ ]:
```
X_test
```

Out[ ]:

	age
16	25
2	47
4	46
8	62
14	49
18	19

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

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

Out[ ]:
```
array([[0.95462525, 0.04537475],
       [0.38275554, 0.61724446],
       [0.42124216, 0.57875784],
       [0.05311321, 0.94688679],
       [0.31039927, 0.68960073],
       [0.98214609, 0.01785391]])
```

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

Out[ ]:
```
0.8333333333333334
```

In [ ]:
```
y_predicted
```

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

In [ ]:

```
X_test
```

Out[ ]:

	age
16	25
2	47
4	46
8	62
14	49
18	19

model.coef_ indicates value of m in y=m*x + b equation

In [ ]:

```
model.coef_
```

Out[ ]:

```
array([[0.16019235]])
```

model.intercept_ indicates value of b in y=m*x + b equation

In [ ]:

```
model.intercept_
```

Out[ ]:

```
array([-7.05117198])
```

Lets defined sigmoid function now and do the math with hand

In [ ]:

```
import math
def sigmoid(x):
  return 1 / (1 + math.exp(-x))
```

In [ ]:

```
def prediction_function(age):
    z = 0.042 * age - 1.53 # 0.04150133 ~ 0.042 and -1.52726963 ~ -1.53
    y = sigmoid(z)
    return y
```

In [ ]:

```
age = 35
```

```
prediction_function(age)
```

Out[ ]:

```
0.4850044983805899
```

0.485 is less than 0.5 which means person with 35 age will *not* buy insurance

In [ ]:

```
age = 43
prediction_function(age)
```

Out[ ]:

```
0.568565299077705
```

0.485 is more than 0.5 which means person with 43 will buy the insurance

---

Naïve Bayes Classifier

Naïve Bayes Classifier is one of the simple and most effective Classification algorithms which helps in building the fast machine learning models that can make quick predictions. It is a probabilistic classifier, which means it predicts on the basis of the probability of an object.

Naive Bayes algorithm falls under classification of Supervised learning.

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 pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [ ]:

```
phool = sns.load_dataset("iris")
phool.head()
```

Out[ ]:

	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 [ ]:

```
X=phool.iloc[: , :-1]
y=phool.iloc[: , -1:]
```

In [ ]:

```
from sklearn.naive_bayes import GaussianNB
model = GaussianNB().fit(X,y)
model
```

```
C:\Users\Junaid\anaconda3\lib\site-packages\sklearn\utils\validation.py:63: DataConversionWarning: A column-vector y was
passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  return f(*args, **kwargs)
GaussianNB()
```

Out[ ]:

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=1)
```

In [ ]:

```
#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
```

```
C:\Users\Junaid\anaconda3\lib\site-packages\sklearn\utils\validation.py:63: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  return f(*args, **kwargs)
```

Out[ ]:

```
array(['setosa', 'versicolor', 'versicolor', 'setosa', 'virginica',
       'versicolor', 'virginica', 'setosa', 'setosa', 'virginica',
       'versicolor', 'setosa', 'virginica', 'versicolor', 'versicolor',
       'setosa', 'versicolor', 'versicolor', 'setosa', 'setosa',
       'versicolor', 'versicolor', 'virginica', 'setosa', 'virginica',
       'versicolor', 'setosa', 'setosa', 'versicolor', 'virginica'],
      dtype='<U10')
```

In [ ]:

```
from sklearn import metrics
score = metrics.accuracy_score(y_test,prediction)*100
score
```

Out[ ]:

```
96.66666666666667
```

In [ ]:

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

Out[ ]:

```
array([[11,  0,  0],
       [ 0, 12,  1],
       [ 0,  0,  6]], dtype=int64)
```

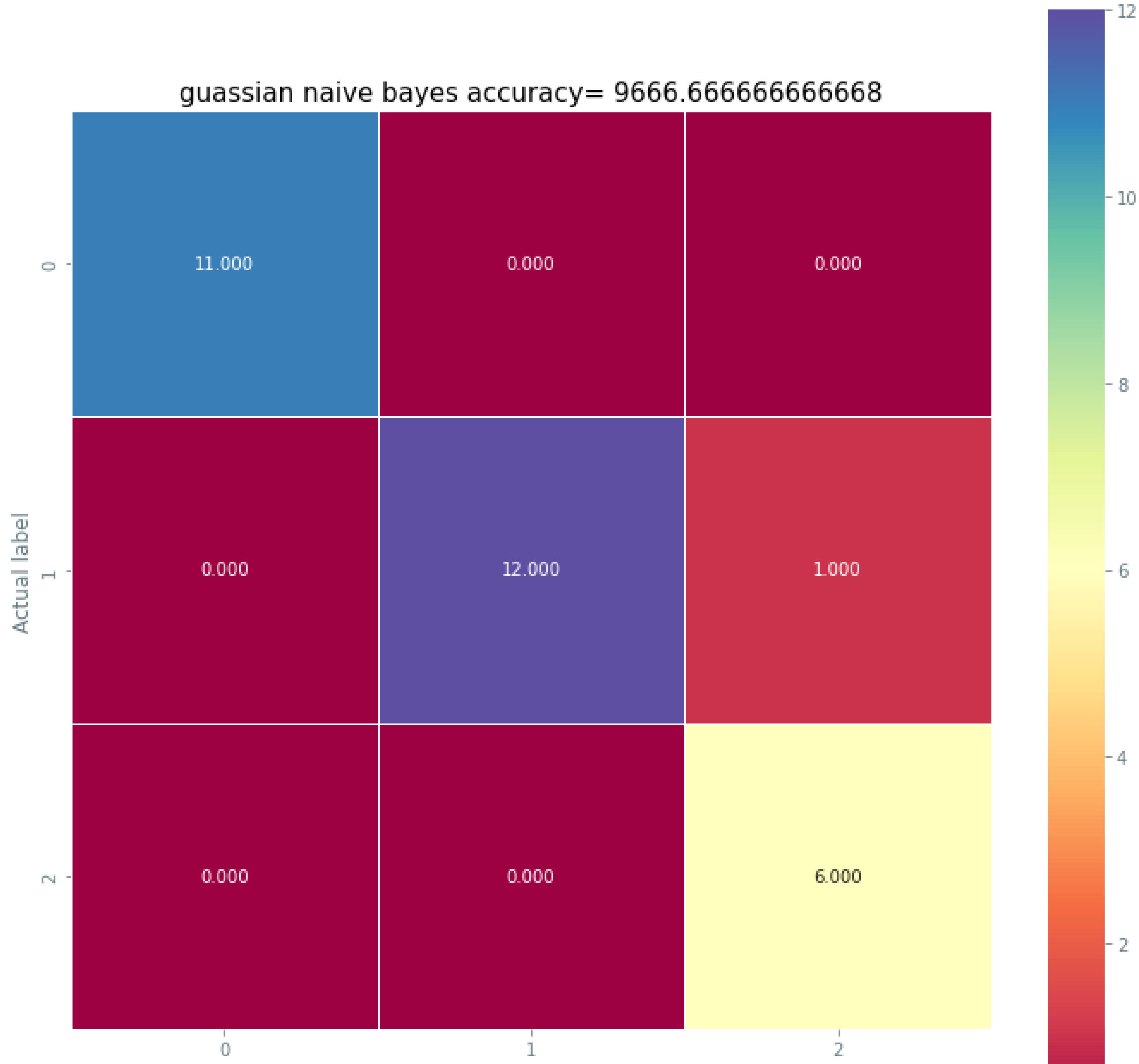
In [ ]:

```
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= 9666.666666666668')
```


guassian naive bayes accuracy= 9666.666666666668
11.000
0.000
0.000
0.000
12.000
1.000
0.000
0.000
6.000
Actual label
0
1
2
12
10
8
6
4
2
0

Support Vector Machine

In [ ]:

```
#import sciket learn
from sklearn import datasets
#load dataset
cancer= datasets.load_breast_cancer()
```

In [ ]:

```
#print the names of the 30 features
print("features: ", cancer.feature_names)
```

```
features:  ['mean radius' 'mean texture' 'mean perimeter' 'mean area'
 'mean smoothness' 'mean compactness' 'mean concavity'
 'mean concave points' 'mean symmetry' 'mean fractal dimension'
 'radius error' 'texture error' 'perimeter error' 'area error'
 'smoothness error' 'compactness error' 'concavity error'
 'concave points error' 'symmetry error' 'fractal dimension error'
 'worst radius' 'worst texture' 'worst perimeter' 'worst area'
 'worst smoothness' 'worst compactness' 'worst concavity'
 'worst concave points' 'worst symmetry' 'worst fractal dimension']
```

In [ ]:

```
#print the label type of cancer ('malignant','benign')
print("labels: ", cancer.target_names)
```

```
labels:  ['malignant' 'benign']
```

In [ ]:

```
#print data
cancer.data.shape
```

Out[ ]:

```
(569, 30)
```

In [ ]:

```
print(cancer.data.shape[0:5])
```

```
(569, 30)
```

In [ ]:

```
print(cancer.data[0:5])
```

```
[[1.799e+01 1.038e+01 1.228e+02 1.001e+03 1.184e-01 2.776e-01 3.001e-01
  1.471e-01 2.419e-01 7.871e-02 1.095e+00 9.053e-01 8.589e+00 1.534e+02
```

```
  6.399e-03 4.904e-02 5.373e-02 1.587e-02 3.003e-02 6.193e-03 2.538e+01
  1.733e+01 1.846e+02 2.019e+03 1.622e-01 6.656e-01 7.119e-01 2.654e-01
  4.601e-01 1.189e-01]
 [2.057e+01 1.777e+01 1.329e+02 1.326e+03 8.474e-02 7.864e-02 8.690e-02
  7.017e-02 1.812e-01 5.667e-02 5.435e-01 7.339e-01 3.398e+00 7.408e+01
  5.225e-03 1.308e-02 1.860e-02 1.340e-02 1.389e-02 3.532e-03 2.499e+01
  2.341e+01 1.588e+02 1.956e+03 1.238e-01 1.866e-01 2.416e-01 1.860e-01
  2.750e-01 8.902e-02]
 [1.969e+01 2.125e+01 1.300e+02 1.203e+03 1.096e-01 1.599e-01 1.974e-01
  1.279e-01 2.069e-01 5.999e-02 7.456e-01 7.869e-01 4.585e+00 9.403e+01
  6.150e-03 4.006e-02 3.832e-02 2.058e-02 2.250e-02 4.571e-03 2.357e+01
  2.553e+01 1.525e+02 1.709e+03 1.444e-01 4.245e-01 4.504e-01 2.430e-01
  3.613e-01 8.758e-02]
 [1.142e+01 2.038e+01 7.758e+01 3.861e+02 1.425e-01 2.839e-01 2.414e-01
  1.052e-01 2.597e-01 9.744e-02 4.956e-01 1.156e+00 3.445e+00 2.723e+01
  9.110e-03 7.458e-02 5.661e-02 1.867e-02 5.963e-02 9.208e-03 1.491e+01
  2.650e+01 9.887e+01 5.677e+02 2.098e-01 8.663e-01 6.869e-01 2.575e-01
  6.638e-01 1.730e-01]
 [2.029e+01 1.434e+01 1.351e+02 1.297e+03 1.003e-01 1.328e-01 1.980e-01
  1.043e-01 1.809e-01 5.883e-02 7.572e-01 7.813e-01 5.438e+00 9.444e+01
  1.149e-02 2.461e-02 5.688e-02 1.885e-02 1.756e-02 5.115e-03 2.254e+01
  1.667e+01 1.522e+02 1.575e+03 1.374e-01 2.050e-01 4.000e-01 1.625e-01
  2.364e-01 7.678e-02]]
```

In [ ]:

```
print(cancer.target)
```

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 1 0 0 0 0 0 0 0 0 1 0 1 1 1 1 1 0 0 1 0 0 1 1 1 1 0 1 0 0 1 1 1 1 0 1 0 0
 1 0 1 0 0 1 1 1 0 0 1 0 0 0 1 1 1 0 1 1 0 0 1 1 1 0 0 1 1 1 1 0 1 1 0 1 1
 1 1 1 1 1 1 0 0 0 1 0 0 1 1 1 0 0 1 0 1 0 0 1 0 0 1 1 0 1 1 0 1 1 1 1 0 1
 1 1 1 1 1 1 1 1 0 1 1 1 1 0 0 1 0 1 1 0 0 1 1 0 0 1 1 1 1 0 1 1 0 0 0 1 0
 1 0 1 1 1 0 1 1 0 0 1 0 0 0 0 1 0 0 0 1 0 1 0 1 1 0 1 0 0 0 0 1 1 0 0 1 1
 1 0 1 1 1 1 1 0 0 1 1 0 1 1 0 0 1 0 1 1 1 1 0 1 1 1 1 1 0 1 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 1 1 1 1 1 1 0 1 0 1 1 0 1 1 0 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1
 1 0 1 1 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 0 1 0 1 1 1 1 0 0 0 1 1
 1 1 0 1 0 1 0 1 1 1 0 1 1 1 1 1 1 1 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 1 0 0
 0 1 0 0 1 1 1 1 1 0 1 1 1 1 1 0 1 1 1 0 1 1 0 0 1 1 1 1 1 1 0 1 1 1 1 1 1
 1 0 1 1 1 1 1 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 0 1 0 0 1 0 1 1 1 1 1 0 1 1
 0 1 0 1 1 0 1 0 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 0 1
 1 1 1 1 1 1 0 1 0 1 1 0 1 1 1 1 1 0 0 1 0 1 0 1 1 1 1 1 0 1 1 0 1 0 1 0 0
 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 0 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 0 0 0 0 0 0 1]
```

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(cancer.data,cancer.target,test_size=0.2, random_state=0)
```

In [ ]:

```
#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
y_pred= clf.predict(X_test)
```

In [ ]:

```
# import scikit.learn metrics module for accuracy calculation
from sklearn import metrics
score = metrics.accuracy_score(y_test,y_pred)*100
# model accuracy: how often is the classifier correct?
print('Accuracy', metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy 0.956140350877193
```

In [ ]:

```
# model precision: what percentage of positive tuples are labled as such?
print('Accuracy', metrics.accuracy_score(y_test, y_pred))

# model recall: what percentage of positive tuples are labled as such?
print('Recall', metrics.recall_score(y_test, y_pred))
```

```
Accuracy 0.956140350877193
Recall 0.9402985074626866
```

In [ ]:

```
# confusion metrics
from sklearn import metrics
cm= metrics.confusion_matrix(y_test, y_pred)
cm
```

Out[ ]:

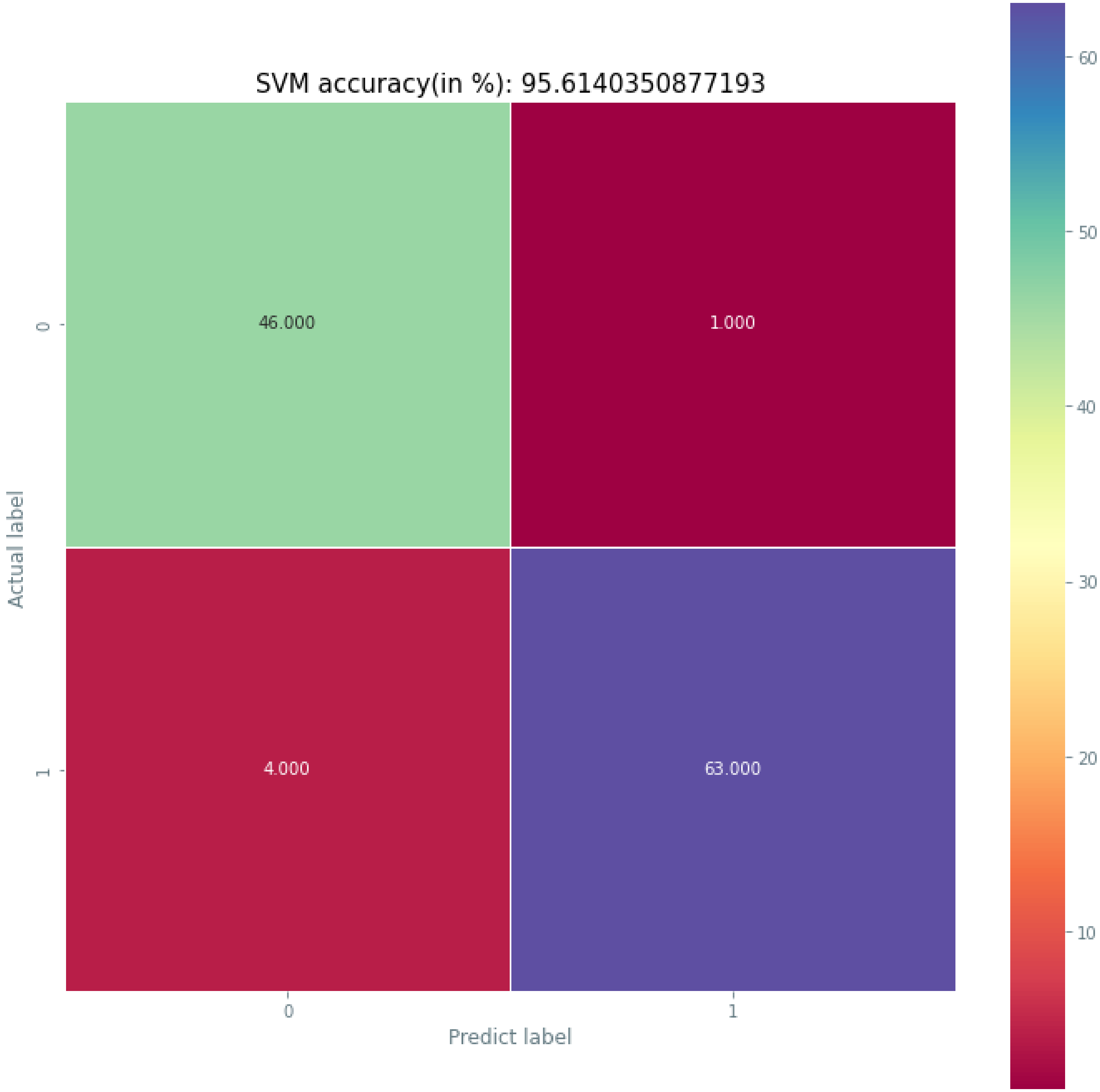
```
array([[46,  1],
       [ 4, 63]], dtype=int64)
```

In [ ]:

```
import seaborn as sns
import matplotlib.pyplot as plt
plt.figure(figsize=(12,12))
sns.heatmap(cm, annot=True, fmt='.3f', linewidths=.5, square=True, cmap="Spectral")
plt.ylabel('Actual label')
plt.xlabel('Predict label')
all_sample_title= 'SVM accuracy(in %): {0}'.format(score)
plt.title(all_sample_title, size=15)
```

Out[ ]:

```
Text(0.5, 1.0, 'SVM accuracy(in %): 95.6140350877193')
```



SVM Example

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

```
from sklearn.datasets import load_iris
iris = load_iris()
```

In [ ]:

```
iris.feature_names
```

Out[ ]:

```
['sepal length (cm)',
 'sepal width (cm)',
 'petal length (cm)',
 'petal width (cm)']
```

In [ ]:

```
iris.target_names
```

Out[ ]:

```
array(['setosa', 'versicolor', 'virginica'], dtype='<U10')
```

In [ ]:

```
df = pd.DataFrame(iris.data,columns=iris.feature_names)
df.head()
```

Out[ ]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

In [ ]:

```
df['target'] = iris.target
df.head()
```

Out[ ]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
4	5.0	3.6	1.4	0.2	0

In [ ]:

```
df[df.target==1].head()
```

Out[ ]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
50	7.0	3.2	4.7	1.4	1
51	6.4	3.2	4.5	1.5	1
52	6.9	3.1	4.9	1.5	1
53	5.5	2.3	4.0	1.3	1
54	6.5	2.8	4.6	1.5	1

In [ ]:

```
df[df.target==2].head()
```

Out[ ]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
100	6.3	3.3	6.0	2.5	2
101	5.8	2.7	5.1	1.9	2
102	7.1	3.0	5.9	2.1	2
103	6.3	2.9	5.6	1.8	2
104	6.5	3.0	5.8	2.2	2

In [ ]:

```
df['flower_name'] =df.target.apply(lambda x: iris.target_names[x])
df.head()
```

Out[ ]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target	flower_name
0	5.1	3.5	1.4	0.2	0	setosa
1	4.9	3.0	1.4	0.2	0	setosa
2	4.7	3.2	1.3	0.2	0	setosa

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target	flower_name
3	4.6	3.1	1.5	0.2	0	setosa
4	5.0	3.6	1.4	0.2	0	setosa

In [ ]:
```
df[45:55]
```

Out[ ]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target	flower_name
45	4.8	3.0	1.4	0.3	0	setosa
46	5.1	3.8	1.6	0.2	0	setosa
47	4.6	3.2	1.4	0.2	0	setosa
48	5.3	3.7	1.5	0.2	0	setosa
49	5.0	3.3	1.4	0.2	0	setosa
50	7.0	3.2	4.7	1.4	1	versicolor
51	6.4	3.2	4.5	1.5	1	versicolor
52	6.9	3.1	4.9	1.5	1	versicolor
53	5.5	2.3	4.0	1.3	1	versicolor
54	6.5	2.8	4.6	1.5	1	versicolor

In [ ]:
```
df0 = df[:50]
df1 = df[50:100]
df2 = df[100:]
```

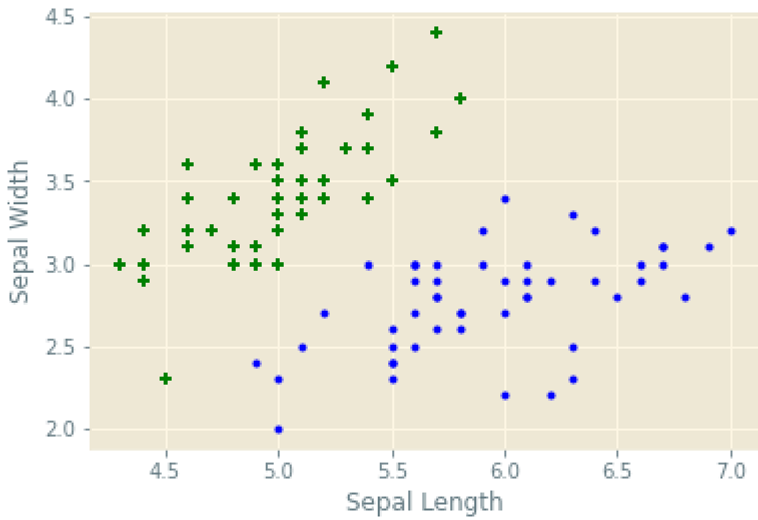
In [ ]:
```
import matplotlib.pyplot as plt
%matplotlib inline
```

Sepal length vs Sepal Width (Setosa vs Versicolor)

In [ ]:
```
plt.xlabel('Sepal Length')
plt.ylabel('Sepal Width')
```

```
plt.scatter(df0['sepal length (cm)'], df0['sepal width (cm)'],color="green",marker='+')
plt.scatter(df1['sepal length (cm)'], df1['sepal width (cm)'],color="blue",marker='.')
```

Out[ ]: <matplotlib.collections.PathCollection at 0x2ba339747f0>

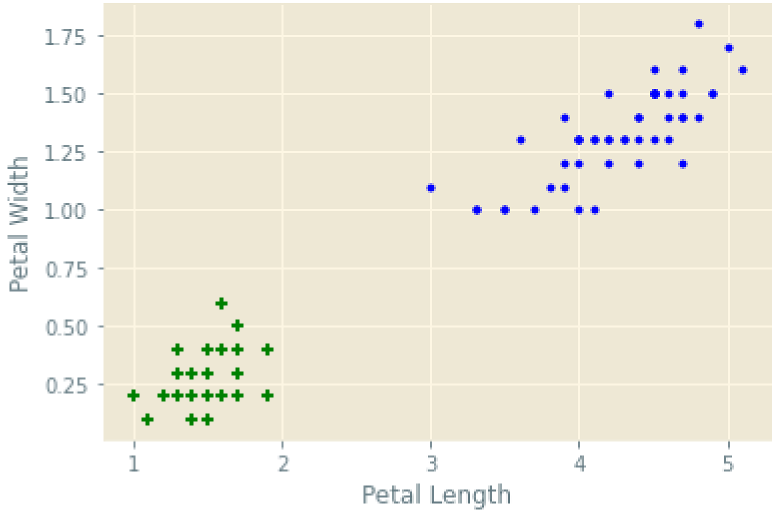


Petal length vs Pepal Width (Setosa vs Versicolor)

In [ ]:

```
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
plt.scatter(df0['petal length (cm)'], df0['petal width (cm)'],color="green",marker='+')
plt.scatter(df1['petal length (cm)'], df1['petal width (cm)'],color="blue",marker='.')
```

Out[ ]: <matplotlib.collections.PathCollection at 0x2ba33c30400>



Train Using Support Vector Machine (SVM)

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

In [ ]:
```
X = df.drop(['target','flower_name'], axis='columns')
y = df.target
```

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

In [ ]:
```
len(X_train)
```

Out[ ]:
```
120
```

In [ ]:
```
len(X_test)
```

Out[ ]:
```
30
```

In [ ]:
```
from sklearn.svm import SVC
model = SVC()
```

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

Out[ ]:
```
SVC()
```

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

Out[ ]:
```
1.0
```

In [ ]:
```
model.predict([[4.8,3.0,1.5,0.3]])
```

Out[ ]:
```
array([0])
```

Tune parameters

1. Regularization (C)

In [ ]:
```
model_C = SVC(C=1)
model_C.fit(X_train, y_train)
model_C.score(X_test, y_test)
```

Out[ ]:
```
1.0
```

In [ ]:
```
model_C = SVC(C=10)
model_C.fit(X_train, y_train)
model_C.score(X_test, y_test)
```

Out[ ]:
```
1.0
```

2. Gamma

In [ ]:
```
model_g = SVC(gamma=10)
model_g.fit(X_train, y_train)
model_g.score(X_test, y_test)
```

Out[ ]:
```
1.0
```

3. Kernel

In [ ]:
```
model_linear_kernal = SVC(kernel='linear')
model_linear_kernal.fit(X_train, y_train)
```

Out[ ]:
```
SVC(kernel='linear')
```

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

Out[ ]:
```
1.0
```