In [ ]:
```
!pip install chart_studio
```

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

# plotly library
#import plotly.plotly as py
import chart_studio.plotly as py
from plotly.offline import init_notebook_mode, iplot
init_notebook_mode(connected=True)
import plotly.graph_objs as go
import warnings
warnings.filterwarnings("ignore", category=FutureWarning)

import os
```

In [ ]:
```
data = pd.read_csv('C:/Users/banke/Desktop/Breastmodel/data.csv')
```

In [ ]:
```
data.info()
```

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

In [ ]:
```
# Drop the unnecessery columns for the prediction
data = data.drop(['Unnamed: 32', 'id'], axis=1)
data.head()
```

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

In [ ]:
```
data.info()
```

In [ ]:
```
data['diagnosis'].value_counts()
```

In [ ]:
```
color_list = ['red' if i == 'M' else 'blue' for i in data.loc[:,'diagnosis']]
pd.plotting.scatter_matrix(data.iloc[:, 7:13],
                                       c=color_list,
                                       figsize= [10,15],
                                       diagonal='hist',
                                       alpha=0.5,
                                       s = 200,
                                       marker = '.',
                                       edgecolor= "black")
plt.show()
```

In [ ]:
```
data['diagnosis'] = [1 if x=='M' else 0 for x in  data['diagnosis']]
```

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

In [ ]:
```
#Choosing x and y values

#x is our features except diagnosis (classification columns)
#y is diagnosis
```

```python
x_data = data.iloc[:,1:]
y = data['diagnosis']
```

In [ ]:
```python
# Normalization our dataNormalization in machine learning refers to the process of scali
# and transforming numeric feature values to a standard range.
#This is typically done to ensure that all features contribute equally to the analysis a
#to prevent features with larger magnitudes from dominating those with smaller magnitude
x = (x_data - np.min(x_data) / (np.max(x_data) - np.min(x_data)))
```

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

In [ ]:
```python
#train test spilt

from sklearn.model_selection import train_test_split
x_train, x_test,y_train,y_test = train_test_split(x,y,test_size =0.3, random_state=1)
```

In [ ]:
```python
print('x-train shape : ', x_train.shape)
print('y-train shape : ', y_train.shape)
print('x-test shape : ', x_test.shape)
print('y-test shape : ', y_test.shape)
```

In [ ]:
```python
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors =3)
knn.fit(x_train, y_train)
predcted_value = knn.predict (x_test)
correct_value= np.array (y_test)

print ('KNN(with k=3 ) accuracy is: ', knn.score(x_test, y_test))
```

In [ ]:
```python
#best_neig= (1,25)
#train_accuracy_list =[]
#test_accuracy_list =[]

#for each in best_neig:
   # knn = KNeighborsClassifier(n_neighbors = each)
   # knn.fit(x_train, y_train)
   # train_accuracy_list.append(knn.score(x_train, y_train))
   # test_accuracy_list.append(knn.score(x_test, y_test))

#print('best k for knn: {}, best accuracy: {}'.format (test_acuracy_list.index(np.max(te

#plt.figure(figsize = [13,8])
#plt.plot(best_neig, train_accuracy_list, label = 'Train Accuracy')
#plt.plot(best_neig, test_accuracy_list, label = 'Test Accuracy')
#plt.title('Neighbors vs accuracy')
#plt.xlabel ('Number of Neighbors')
#plt.ylabel ('Accuracy')
#plt.legend()
#plt.grid()
#plt.xticks(best_neig)
#plt.show()
```

In [ ]:
```python
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors =7)
knn.fit(x_train, y_train)
predcted_value = knn.predict (x_test)
correct_value= np.array (y_test)
```

```
print ('KNN(with k=7 ) accuracy is: ', knn.score(x_test, y_test))
```

In [ ]:
```
from sklearn.svm import SVC
#svm = SVC(random_state = 1, gamma='auto')
svm = SVC()
svm.fit(x_train, y_train)
print ("accuracy of SVM : ", svm.score(x_test, y_test))
```

In [ ]:
```
from sklearn.naive_bayes import GaussianNB
nb = GaussianNB()
nb.fit(x_train, y_train)
print ("accuracy of naive bayes: ", nb.score(x_test, y_test))
```

In [ ]:
```
#Decision Tree
from sklearn.tree import DecisionTreeClassifier
dt = DecisionTreeClassifier()
dt.fit(x_train, y_train)
print ("accuracy of Decision Tree Classification: ", dt.score(x_test,y_test))
```

In [ ]:
```
#Rondom Forest
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(n_estimators=200, random_state=1)
rf.fit(x_train, y_train)
print ("accuracy of Random Forest Classification: ", rf.score(x_test,y_test))
```

In [ ]:
```
```