Q.3. Use the dataset Data1.csv to implement the list of classification models below in order to predict whether a tumor is malign or beginning. The dependent variable in this dataset is Class, it is equal to 2 if the tumor is benign and 4 if it is malign. The remaining columns are features that describe different characteristics of each tumor.

Import the libraries

In [1]:
```python
from sklearn import svm
import pandas as pd
from sklearn.model_selection import train_test_split
import numpy as np
import matplotlib.pyplot as plt
from sklearn import metrics
import seaborn as sb
```

Read the data from the csv.

In [2]:
```python
dataset = pd.read_csv('Data1.csv')
x = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values
```

Now will split the dataset into the training set and test set

In [3]:
```python
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2, random_state = 0)
```

Feature scaling

In [4]:
```python
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
x_train = sc.fit_transform(x_train)
x_test = sc.transform(x_test)
```

1. Logistic Regression

Import the Logistic Regression library

In [5]:
```python
from sklearn.linear_model import LogisticRegression
```

Create the Logistic Regression object.

In [6]:
```python
logisticRegr = LogisticRegression(random_state=0)
```

Training the model on the data, storing the information learned from the data Model is learning the relationship between digits (x_train) and labels (y_train).

In [7]:
```python
logisticRegr.fit(x_train, y_train)
```

Out[7]:
```
LogisticRegression(random_state=0)
```

Predict labels for new data. Uses the information the model learned during the model training process.

In [8]:
```python
logisticRegr.predict(x_test[0].reshape(1,-1))
```

Out[8]:
```
array([2], dtype=int64)
```

Predict for Multiple Observations at Once

In [9]:
```python
logisticRegr.predict(x_test[0:10])
```

Out[9]:
```
array([2, 2, 4, 4, 2, 2, 2, 4, 2, 2], dtype=int64)
```

Make predictions on entire test data.

In [10]:
```python
predictions = logisticRegr.predict(x_test)
```

Let's see the accuracy of the model using the score method:

In [11]:
```python
score = logisticRegr.score(x_test, y_test)
print("Accuracy: {:.2f} %".format(score*100))
```

```
Accuracy: 95.62 %
```

In [12]:
```python
cm = metrics.confusion_matrix(y_test, predictions)
print(cm)
```

```
[[84  3]
 [ 3 47]]
```

According to the results above the accuracy of the model is not quite good - 95.62%. There were 6 values out of 137 that were not predicted correctly.

Using K-fold cross-validation method (cv = 10) will ensure that the accuracy score calculated only on one test set was not just a lucky occurence. To be more exactly in k-fold, with K=10, the dataset is devided into 10 subsets. At the first iteration the model is trained on the first 9 subsets and tested on the 10th subset. In the second iteration, the model is trained on the first 8 subsets and 10th, and is tested on the 9th subset. In this way there are 10 iterations and we obtain 10 accuracies of the model. Will print the mean accuracy score and the standard deviation of the score computed.

In [13]:
```python
from sklearn.model_selection import cross_val_score
logisticRegr_accuracies = cross_val_score(estimator = logisticRegr, X = x_train, y = y_train, cv = 10)
print("Accuracy: {:.2f} %".format(logisticRegr_accuracies.mean()*100))
print("Standard Deviation: {:.2f} %".format(logisticRegr_accuracies.std()*100))
logisticRegr_accuracies
```

```
Accuracy: 96.70 %
Standard Deviation: 1.97 %
```

Out[13]:
```
array([0.94545455, 0.96363636, 0.96363636, 1.        , 0.94545455,
       1.        , 0.96296296, 0.96296296, 0.98148148, 0.94444444])
```

The mean accurracy for 10 test sets is 96.7% that is not much bigger that the accuracy of the model calculated on only one train data set (95.62%).

2. Decision Tree Classifier

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

In [14]:
```python
from sklearn.tree import DecisionTreeClassifier
```

Create Decision Tree classifer object.

In [15]:
```python
dtc = DecisionTreeClassifier( random_state=0)
```

Train Decision Tree Classifer.

In [16]:
```python
dtc = dtc.fit(x_train, y_train)
```

Predict the response for test dataset.

In [17]:
```python
y_pred = dtc.predict(x_test)
```

Let's estimate, how accurately the classifier or model can predict whether a tumor is malign or bening. Accuracy can be computed by comparing actual test set values and predicted values.

In [18]:
```python
print("Accuracy: {:.2f} %".format(metrics.accuracy_score(y_test, y_pred)*100))
```

```
Accuracy: 92.70 %
```

In [19]:
```python
cm = metrics.confusion_matrix(y_test, y_pred)
print(cm)
```

```
[[80  7]
 [ 3 47]]
```

We see that this model has a big accuracy that means it is good to use for predicting whether a tumor is malign or benign. The confusion matrix also shows that only 10 out of 137 objects are not predicted as they had to.

Using K-fold cross-validation method (cv = 10) will ensure that the accuracy score calculated only on one test set was not just a lucky occurence. To be more exactly in k-fold, with K=10, the dataset is devided into 10 subsets. At the first iteration the model is trained on the first 9 subsets and tested on the 10th subset. In the second iteration, the model is trained on the first 8 subsets and 10th, and is tested on the 9th subset. In this way there are 10 iterations and we obtain 10 accuracies of the model. Will print the mean accuracy score and the standard deviation of the score computed.

In [20]:
```python
from sklearn.model_selection import cross_val_score
dtc_accuracies = cross_val_score(estimator = dtc, X = x_train, y = y_train, cv = 10)
print("Accuracy: {:.2f} %".format(dtc_accuracies.mean()*100))
print("Standard Deviation: {:.2f} %".format(dtc_accuracies.std()*100))
dtc_accuracies
```

```
Accuracy: 93.59 %
Standard Deviation: 2.21 %
```

Out[20]:
```
array([0.92727273, 0.90909091, 0.92727273, 0.96363636, 0.94545455,
       0.92727273, 0.90740741, 0.92592593, 0.98148148, 0.94444444])
```

The mean accuracy calculated for 10 subsets of data is bigger with 0.89% that the accuracy of the model calculated on only one test set.

3. Random Forest Classifier (with nb_trees = 10)

Import Random Forest Model.

In [21]:
```python
from sklearn.ensemble import RandomForestClassifier
```

Create a Gaussian Classifier.

In [22]:
```python
rfc=RandomForestClassifier(n_estimators=10)
```

Train the model using the training sets y_pred=clf.predict(X_test)

In [23]:
```python
rfc.fit(x_train,y_train)
```

Out[23]:
```
RandomForestClassifier(n_estimators=10)
```

Predict the response for test dataset.

In [24]:
```python
y_pred=rfc.predict(x_test)
```

After training, check the accuracy using actual and predicted values.

In [25]:
```python
print("Accuracy: {:.2f} %".format(metrics.accuracy_score(y_test, y_pred)*100))
```

```
Accuracy: 95.62 %
```

In [26]:
```python
cm = metrics.confusion_matrix(y_test, y_pred)
print(cm)
```

```
[[84  3]
 [ 3 47]]
```

We can observe that the accuracy of the model is pretty high - 96,35%. There are only 5 values that were not predicted to the correct group.

Using K-fold cross-validation method (cv = 10) will ensure that the accuracy score calculated only on one test set was not just a lucky occurence. To be more exactly in k-fold, with K=10, the dataset is devided into 10 subsets. At the first iteration the model is trained on the first 9 subsets and tested on the 10th subset. In the second iteration, the model is trained on the first 8 subsets and 10th, and is tested on the 9th subset. In this way there are 10 iterations and we obtain 10 accuracies of the model. Will print the mean accuracy score and the standard deviation of the score computed.

In [27]:
```python
from sklearn.model_selection import cross_val_score
rfc_accuracies = cross_val_score(estimator = rfc, X = x_train, y = y_train, cv = 10)
print("Accuracy: {:.2f} %".format(rfc_accuracies.mean()*100))
print("Standard Deviation: {:.2f} %".format(rfc_accuracies.std()*100))
rfc_accuracies
```

```
Accuracy: 97.07 %
Standard Deviation: 2.03 %
```

Out[27]:
```
array([0.92727273, 0.96363636, 0.98181818, 0.98181818, 0.96363636,
       1.        , 0.96296296, 0.96296296, 1.        , 0.96296296])
```

Here we can observe that the mean value of the accuracy for the model calculated on 10 test datasets (96.15%) is bigger than the value of the accuracy on the model calculated on only one test dataset (95.62%).

4. K- Nearest Neighbors (K-NN)

Import the KNeighborsClassifier library

In [28]:
```python
from sklearn.neighbors import KNeighborsClassifier
```

Create the object of KNeighborsClassifier type with number of neighbors =3.

In [29]:
```python
neigh = KNeighborsClassifier(n_neighbors=3)
```

Train the KNeighbors Classifier.

In [30]:
```python
neigh.fit(x_train,y_train)
```

Out[30]:
```
KNeighborsClassifier(n_neighbors=3)
```

Predict the response for test dataset.

In [31]:
```python
y_pred = neigh.predict(x_test)
```

After training, check the accuracy using actual and predicted values.

In [32]:
```python
print("Accuracy: {:.2f} %".format(metrics.accuracy_score(y_test, y_pred)*100))
```

```
Accuracy: 97.08 %
```

In [33]:
```python
cm = metrics.confusion_matrix(y_test, y_pred)
print(cm)
```

```
[[85  2]
 [ 2 48]]
```

We can observe that this model is better that the previous ones. This has 97.08% of accuracy and only 4 values out of 137 are predicted wrongly. For sure is one of the best candidates for the best models.

Using K-fold cross-validation method (cv = 10) will ensure that the accuracy score calculated only on one test set was not just a lucky occurence. To be more exactly in k-fold, with K=10, the dataset is devided into 10 subsets. At the first iteration the model is trained on the first 9 subsets and tested on the 10th subset. In the second iteration, the model is trained on the first 8 subsets and 10th, and is tested on the 9th subset. In this way there are 10 iterations and we obtain 10 accuracies of the model. Will print the mean accuracy score and the standard deviation of the score computed.

In [34]:
```python
from sklearn.model_selection import cross_val_score
neigh_accuracies = cross_val_score(estimator = neigh, X = x_train, y = y_train, cv = 10)
print("Accuracy: {:.2f} %".format(neigh_accuracies.mean()*100))
print("Standard Deviation: {:.2f} %".format(neigh_accuracies.std()*100))
neigh_accuracies
```

```
Accuracy: 96.34 %
Standard Deviation: 2.44 %
```

Out[34]:
```
array([0.90909091, 0.94545455, 0.98181818, 0.98181818, 0.94545455,
       1.        , 0.96296296, 0.96296296, 0.98148148, 0.96296296])
```

We can observe that the accuracy for the model calculated on only one test dataset is bigger than the mean accuracy for the model calculated on 10 smaller test data sets with 0.74%. Here we can say that there was a little lucky occurences.

5. Naive Bayes

Import the GaussianNB library

In [35]:
```python
from sklearn.naive_bayes import GaussianNB
```

Create the GaussianNB object and train it on the train data.

In [36]:
```python
gnb = GaussianNB()
gnb.fit(x_train, y_train)
```

Out[36]:
```
GaussianNB()
```

Making predictions on the testing set

In [37]:
```python
y_pred = gnb.predict(x_test)
```

Comparing actual response values (y_test) with predicted response values (y_pred)

In [38]:
```python
from sklearn import metrics
print("Accuracy: {:.2f}%".format(metrics.accuracy_score(y_test, y_pred)*100))
```

```
Accuracy: 94.89%
```

In [39]:
```python
cm = metrics.confusion_matrix(y_test, y_pred)
print(cm)
```

```
[[80  7]
 [ 0 50]]
```

The model has a pretty good accuracy- 94.89%. There are only 7 values that were predicted wrongly.

Using K-fold cross-validation method (cv = 10) will ensure that the accuracy score calculated only on one test set was not just a lucky occurence. To be more exactly in k-fold, with K=10, the dataset is devided into 10 subsets. At the first iteration the model is trained on the first 9 subsets and tested on the 10th subset. In the second iteration, the model is trained on the first 8 subsets and 10th, and is tested on the 9th subset. In this way there are 10 iterations and we obtain 10 accuracies of the model. Will print the mean accuracy score and the standard deviation of the score computed.

In [40]:
```python
from sklearn.model_selection import cross_val_score
gnb_accuracies = cross_val_score(estimator = gnb, X = x_train, y = y_train, cv = 10)
print("Accuracy: {:.2f} %".format(gnb_accuracies.mean()*100))
print("Standard Deviation: {:.2f} %".format(gnb_accuracies.std()*100))
gnb_accuracies
```

```
Accuracy: 96.16 %
Standard Deviation: 1.91 %
```

Out[40]:
```
array([0.92727273, 0.96363636, 0.96363636, 0.96363636, 0.96363636,
       0.96363636, 0.98148148, 0.94444444, 1.        , 0.94444444])
```

There can be observed that the accuracy score of the model calculated on only one test set is smaller that the accuracy mean calculated on 10 smaller data sets. (94.89% < 96.16%)

6. Support Vector Machine (SVM)

Train the SVC classifier

In [41]:
```python
clf = svm.SVC(random_state=0)
clf.fit(x_train, y_train)
```

Out[41]:
```
SVC(random_state=0)
```

Make predictions and compare the results by displaying the predicted values of y next to the test values of y in a two-dimensional array.

In [42]:
```python
y_pred = clf.predict(x_test)
y_pred
```

Out[42]:
```
array([2, 2, 4, 4, 2, 2, 2, 4, 2, 2, 4, 2, 4, 2, 2, 4, 4, 4, 4, 2, 2, 2,
       4, 2, 4, 4, 2, 2, 2, 4, 2, 4, 4, 2, 2, 2, 4, 4, 2, 4, 2, 2, 2, 2,
       2, 2, 2, 4, 2, 2, 4, 2, 4, 2, 2, 2, 4, 4, 2, 4, 2, 2, 2, 2, 2, 2,
       2, 2, 4, 4, 2, 2, 2, 2, 2, 2, 4, 2, 2, 2, 4, 2, 4, 2, 2, 4, 2, 4,
       4, 2, 4, 2, 4, 4, 2, 4, 4, 4, 4, 2, 2, 2, 4, 4, 2, 2, 4, 4, 2, 2,
       4, 2, 2, 4, 2, 2, 2, 4, 2, 2, 2, 4, 2, 2, 4, 4, 2, 4, 2, 4, 4, 2,
       4, 2, 2, 4, 2], dtype=int64)
```

Create and print the Confusion Matrix and the accuracy score of the model and interpret the two.

In [43]:
```python
from sklearn.metrics import confusion_matrix,accuracy_score
cm = confusion_matrix(y_test, y_pred)
print(cm)
print("SVC model accuracy: {:.2f}%".format(accuracy_score(y_test, y_pred)*100))
```

```
[[82  5]
 [ 1 49]]
SVC model accuracy: 95.62%
```

We can observe that this is a good model with an accuracy of 95.62%. There are 6 values that were not predicted correctly.

Using K-fold cross-validation method (cv = 10) will ensure that the accuracy score calculated only on one test set was not just a lucky occurence. To be more exactly in k-fold, with K=10, the dataset is devided into 10 subsets. At the first iteration the model is trained on the first 9 subsets and tested on the 10th subset. In the second iteration, the model is trained on the first 8 subsets and 10th, and is tested on the 9th subset. In this way there are 10 iterations and we obtain 10 accuracies of the model. Will print the mean accuracy score and the standard deviation of the score computed.

In [44]:
```python
from sklearn.model_selection import cross_val_score
svm_accuracies = cross_val_score(estimator = clf, X = x_train, y = y_train, cv = 10)
print("Accuracy: {:.2f} %".format(svm_accuracies.mean()*100))
print("Standard Deviation: {:.2f} %".format(svm_accuracies.std()*100))
svm_accuracies
```

```
Accuracy: 96.53 %
Standard Deviation: 1.91 %
```

Out[44]:
```
array([0.92727273, 0.96363636, 0.96363636, 0.98181818, 0.96363636,
       0.96363636, 0.98148148, 0.96296296, 1.        , 0.94444444])
```

We can observe that the accuracy score calculated for a only one test set is smaller with 0.92% than the accuracy mean calculatd for 10 test sets.

Q.4. Choose the best performing model based on the results from performing the k-fold cross-validation. Discuss your choice.

In [55]:
```python
lg_accuracy = logisticRegr_accuracies.mean()*100
lg_std = logisticRegr_accuracies.std()*100
dtc_accuracy = dtc_accuracies.mean()*100
dtc_std = dtc_accuracies.std()*100
rfc_accuracy = rfc_accuracies.mean()*100
rfc_std = rfc_accuracies.std()*100
neigh_accuracy = neigh_accuracies.mean()*100
neigh_std = neigh_accuracies.std()*100
gnb_accuracy = gnb_accuracies.mean()*100
gnb_std = gnb_accuracies.std()*100
svm_accuracy = svm_accuracies.mean()*100
svm_std = svm_accuracies.std()*100

print("Mean accuracy for Logistic Regression model: {:.2f} %, Std: {:.2f}.".format(lg_accuracy, lg_std))
print("Mean accuracy for Decision Tree Classifier model: {:.2f}%, Std: {:.2f}.".format(dtc_accuracy, dtc_std))
print("Mean accuracy for Random Forest Classifier with 10 nb_trees: {:.2f} %, Std: {:.2f}.".format(rfc_accuracy
print("Mean accuracy for K-Nearest Neighbors: {:.2f} %, Std: {:.2f}.".format(neigh_accuracy, neigh_std))
print("Mean accuracy for Naive Bayes: {:.2f} %, Std: {:.2f}.".format(gnb_accuracy, gnb_std))
print("Mean accuracy for Support Vector Machine: {:.2f} %, Std: {:.2f}.".format(svm_accuracy, svm_std))

lg1 = lg_accuracy-lg_std
dtc1 = dtc_accuracy - dtc_std
rfc1 = rfc_accuracy - rfc_std
neigh1 = neigh_accuracy - neigh_std
gnb1 = gnb_accuracy - gnb_std
svm1 = svm_accuracy - svm_std

map = {"Logistic Regression":lg1, "Decision Tree Classifier" : dtc1, "Random Forest Classifier" : rfc1, "K-Nea
        "Naive Bayes" : gnb1,
        "Support Vector Machine" : svm1}

print ("\n Below we can see the results for all 6 models of the difference calculation between mean accuracy ar
print(map)
max = max(map.values())

print("The best result is: {:.2f}".format(max))
```

```
Mean accuracy for Logistic Regression model: 96.70 %, Std: 1.97.
Mean accuracy for Decision Tree Classifier model: 93.59%, Std: 2.21.
Mean accuracy for Random Forest Classifier with 10 nb_trees: 97.07 %, Std: 2.03.
Mean accuracy for K-Nearest Neighbors: 96.34 %, Std: 2.44.
Mean accuracy for Naive Bayes: 96.16 %, Std: 1.91.
Mean accuracy for Support Vector Machine: 96.53 %, Std: 1.91.

 Below we can see the results for all 6 models of the difference calculation between mean accuracy and standard
deviation.
{'Logistic Regression': 94.73053901089192, 'Decision Tree Classifier': 91.38701005825551, 'Random Forest Classi
fier': 95.03858734142017, 'K-Nearest Neighbors': 93.90304295607386, 'Naive Bayes': 94.2457764262916, 'Support V
ector Machine': 94.61833923684192}
The best result is: 95.04
```

We can observe that the best mean accuracy is for Random Forest Classifier model.

Q. 5. For the chosen best performing model select two hyper parameters you would like to tune using the grid search method. Learn what the two chosen hyper parameters do to your model. Look into the description of hyper parameters for your respective model by using the scikit-org API https://scikitlearn.org/stable/modules/classes.html. Discuss here the two hyper parameters you chose and how they impact the model. Choose between 2 and 4 different values for each of these hyper parameters to test with grid search. Explain your choice and, if you have any, make some expectations (you could speculate for example which values of the chosen hyper parameters you believe will make the model perform best and why). Use grid search to find the best model. Print the accuracy of this model and the hyper parameter values of the best model. Do these values confirm your intuition?

Firstly let's see what parameters has the classifier of Random Forest Classifier model.

In [46]:
```python
rfc.get_params().keys()
```

Out[46]:
```
dict_keys(['bootstrap', 'ccp_alpha', 'class_weight', 'criterion', 'max_depth', 'max_features', 'max_leaf_node
s', 'max_samples', 'min_impurity_decrease', 'min_samples_leaf', 'min_samples_split', 'min_weight_fraction_lea
f', 'n_estimators', 'n_jobs', 'oob_score', 'random_state', 'verbose', 'warm_start'])
```

In [47]:
```python
params = {"n_estimators": [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
          "criterion": ["gini", "entropy"],
          "max_depth": [1, 2, 3, 4, 5, 6, 7, 8, 9, None],
          "max_features": ["auto", "sqrt", "log2"]}
```

Description of some hyper parameters that I chose:

1. n_estimators - Number of trees in the forest.

2. criterion - The function to measure the quality of a split. Supported criteria are "gini" for the Gini impurity and "entropy" for the information gain. Gini Impurity is a measurement of the likelihood of an incorrect classification of a new instance of a random variable, if that new instance were randomly classified according to the distribution of class labels from the data set. Entropy is a measure of information that indicates the disorder of the features with the target.

3. max_depth - The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples.

4. max_features - The number of features to consider when looking for the best split:

- If int, then consider max_features features at each split.

- If float, then max_features is a fraction and round(max_features * n_features) features are considered at each split.

- If “auto”, then max_features=sqrt(n_features).

- If “sqrt”, then max_features=sqrt(n_features) (same as “auto”).

- If “log2”, then max_features=log2(n_features).

- If None, then max_features=n_features.

In [48]:
```python
from sklearn.model_selection import GridSearchCV
grid_search = GridSearchCV(estimator = rfc,
                           param_grid= params,
                           cv = 10,
                          scoring = 'accuracy',
                          n_jobs = -1)
grid_search.fit(x_train, y_train)
best_accuracy = grid_search.best_score_
best_parameters = grid_search.best_params_
print("Best Accuracy: {:.2f} %".format(best_accuracy*100))
print("Best Parameters:", best_parameters)
```

```
Best Accuracy: 97.98 %
Best Parameters: {'criterion': 'gini', 'max_depth': 3, 'max_features': 'log2', 'n_estimators': 9}
```

We can observe that for the best accuracy we must have the values for parameters as: "criterion" = "gini", "max_depth" = 3, "max_features" = sqrt, "n_estimators": 6. This means that the number of trees in the forest must be 3. The maximum depth of the tree is 3. The criterion for measure the quality of the split is "gini". The nuumber of features to consider when looking for the best split is "sqrt" of the total number of features. Using these parameters will obtain an accuracy of 97.98%.