CS3481 Homework 2

In this homework, I am going to provide a report on constructing a Random Forest and Naïve Bayes Classifiers to predict whether a person is categorized ***disk hernia (DH)***, ***spondylolisthesis (SL)*** or ***Normal (NO)***.

Here are the features that I have to consider for classification:

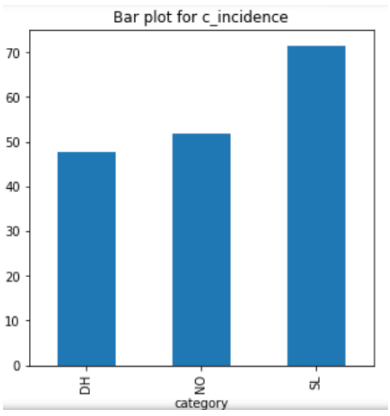
```
In [3]: import pandas as pd

        df = pd.read_excel('./data.xlsx')
        df.head()
```

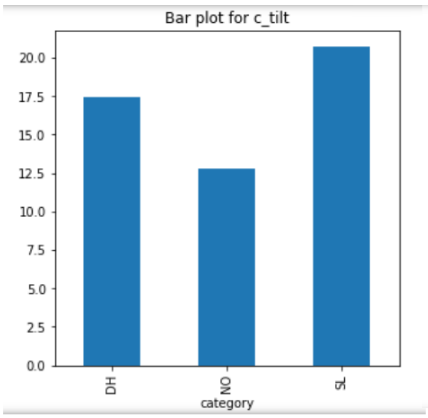
Out[3]:

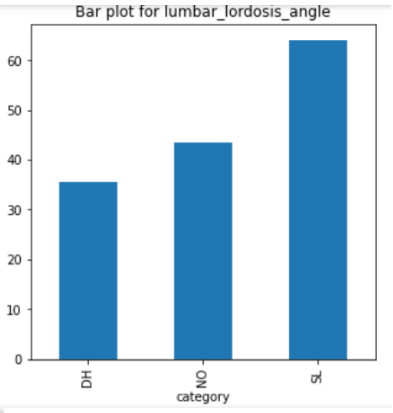
	c_incidence	c_tilt	lumbar_lordosis_angle	sacral_slope	pelvic_radius	degree_spondylolisthesis	category
0	63.03	22.55	39.61	40.48	98.67	-0.25	DH
1	39.06	10.06	25.02	29.00	114.41	4.56	DH
2	68.83	22.22	50.09	46.61	105.99	-3.53	DH
3	69.30	24.65	44.31	44.64	101.87	11.21	DH
4	49.71	9.65	28.32	40.06	108.17	7.92	DH

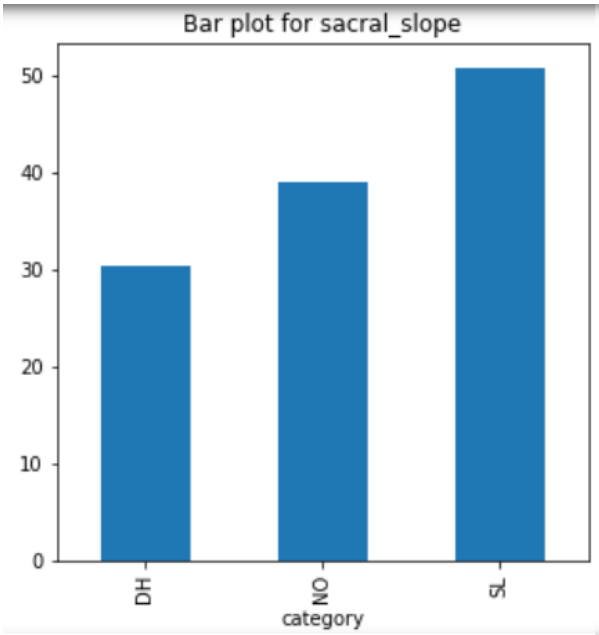
As we can see, [c_incidence, c_tilt, lumbar_lordosis_angle, sacral_slope, pelvic_radius, degree spondylolisthesis] are the features in our dataset. It is also worth noting that, when we group the mean values of columns by category (e.g pandas groupby() method) and visualize, we get the following insight:

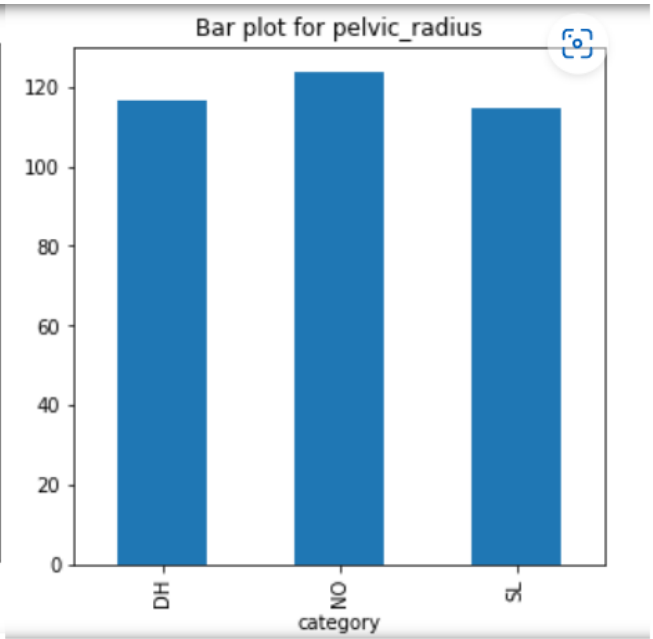
```
grouped_df = df.groupby(['category']).mean()

for col in grouped_df.columns:
    plt.figure(figsize = (5,5))
    plt.title(f'Bar plot for {col}')
    grouped_df[col].plot(kind = 'bar')
    plt.show()
```



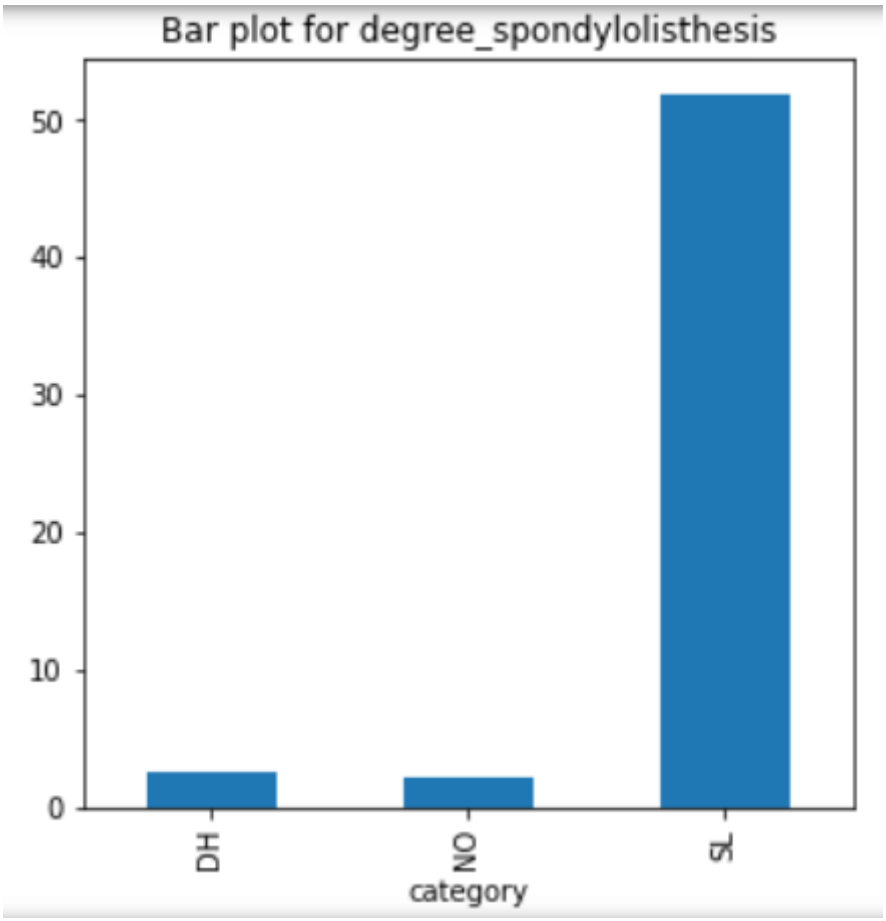








As seen by the barplots above, we can make a conclusion that there is no important feature that might more likely classify a certain category except the feature = [degree_spondylolisthesis]:



We can clearly tell that the high degree of spondylolisthesis of a patient makes it clear that the patient has SL category, although it can be already assumed from the name of the feature that the higher degree of spondylolisthesis, the more likely it is SL.

A) *Construct random forest models using different numbers of component trees based on a specific training set/test set partition, and analyze the resulting change in classification performance*

For this part, firstly, we split the dataset into train and test sets. I have partitioned it by 80% train set and 20% test set. The reason I favored more portion to train set is that I further want to use cross-validation to evaluate the performance of the best model with respect to hyperparameters:

In [14]:
```
from sklearn.model_selection import train_test_split

X = df.drop(['category'], axis = 1)
y = df['category']

X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2, random_state=42)
```

In [15]:
```
print(len(X), len(X_train), len(X_test))
```

```
310 248 62
```

I have used GridSearchCV as a cross validation method in order to tune hyperparameters and find the best Random Forest Classifier (5-fold cross-validation).

```
from sklearn.model_selection import GridSearchCV

def print_results(results):
    print('BEST PARAMS: {}\n'.format(results.best_params_))

    means = results.cv_results_['mean_test_score']
    stds = results.cv_results_['std_test_score']
    for mean, std, params in zip(means, stds, results.cv_results_['params']):
        print('{} (+/-{}) for {}'.format(round(mean, 3), round(std * 2, 3), params))
```

```
from sklearn.ensemble import RandomForestClassifier

clf = RandomForestClassifier()

params = {
    'n_estimators' : [5,50,250],
    'max_depth' : [2,4,8,16,32, None]
}

cv_clf = GridSearchCV(clf,params, cv=5)

cv_clf.fit(X_train, y_train)

print_results(cv_clf)
```

Here, it is worth noting that I have tuned only ‘n_estimators’ (number of decision trees in the forest) and ‘max_depth’ (max depth of trees) for tuning. This is because each of them has 3 and 6 values respectively, that might end up in 18 combinations of models. If I have used even more hyperparameters to tune with respective varieties of values, I would have increased the training time drastically, and for the purpose of this assignment, I have selected two hyperparameters above.

I also have used ‘print_results()’ helper function to print accuracy of each model. Here is the result:

```
BEST PARAMS: {'max_depth': 4, 'n_estimators': 50}

0.791 (+/-0.162) for {'max_depth': 2, 'n_estimators': 5}
0.775 (+/-0.097) for {'max_depth': 2, 'n_estimators': 50}
0.767 (+/-0.133) for {'max_depth': 2, 'n_estimators': 250}
0.79 (+/-0.112) for {'max_depth': 4, 'n_estimators': 5}
0.847 (+/-0.082) for {'max_depth': 4, 'n_estimators': 50}
0.835 (+/-0.1) for {'max_depth': 4, 'n_estimators': 250}
0.827 (+/-0.073) for {'max_depth': 8, 'n_estimators': 5}
0.843 (+/-0.083) for {'max_depth': 8, 'n_estimators': 50}
0.839 (+/-0.066) for {'max_depth': 8, 'n_estimators': 250}
0.798 (+/-0.076) for {'max_depth': 16, 'n_estimators': 5}
0.839 (+/-0.066) for {'max_depth': 16, 'n_estimators': 50}
0.839 (+/-0.061) for {'max_depth': 16, 'n_estimators': 250}
0.807 (+/-0.114) for {'max_depth': 32, 'n_estimators': 5}
0.839 (+/-0.107) for {'max_depth': 32, 'n_estimators': 50}
0.843 (+/-0.068) for {'max_depth': 32, 'n_estimators': 250}
0.807 (+/-0.114) for {'max_depth': None, 'n_estimators': 5}
0.835 (+/-0.069) for {'max_depth': None, 'n_estimators': 50}
0.843 (+/-0.046) for {'max_depth': None, 'n_estimators': 250}
```

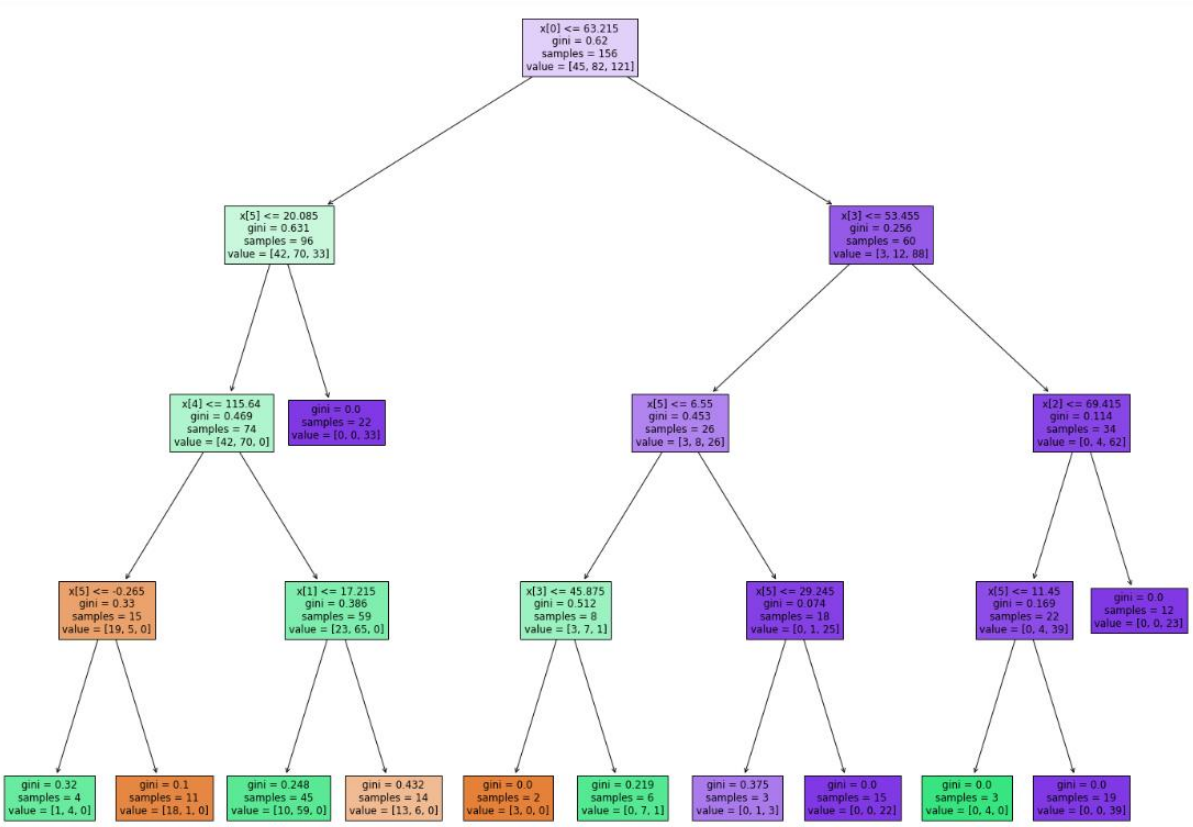
We can clearly tell that there are ‘bad’ hyperparameter values that we have to avoid and ‘good’ values to favor in order to increase the accuracy of the model and select the best model. As a result, we can select a Random Forest Classifier with ‘max_depth’ = 4 and ‘n_estimators’ = 50. A relatively small depth and small number of trees in the forest helps avoiding overfitting. This is the reason I did not choose a model with ‘max_depth’ = None and ‘n_estimators’ = 250, as it it clear that although it has high accuracy, that might be because it is prone to overfit.

B) *For the random forest model corresponding to the best classification performance, select different component decision trees in the model, and compare the classification performances of these trees with that of the original random forest model.*

For this part, we can select a decision tree and visualize:

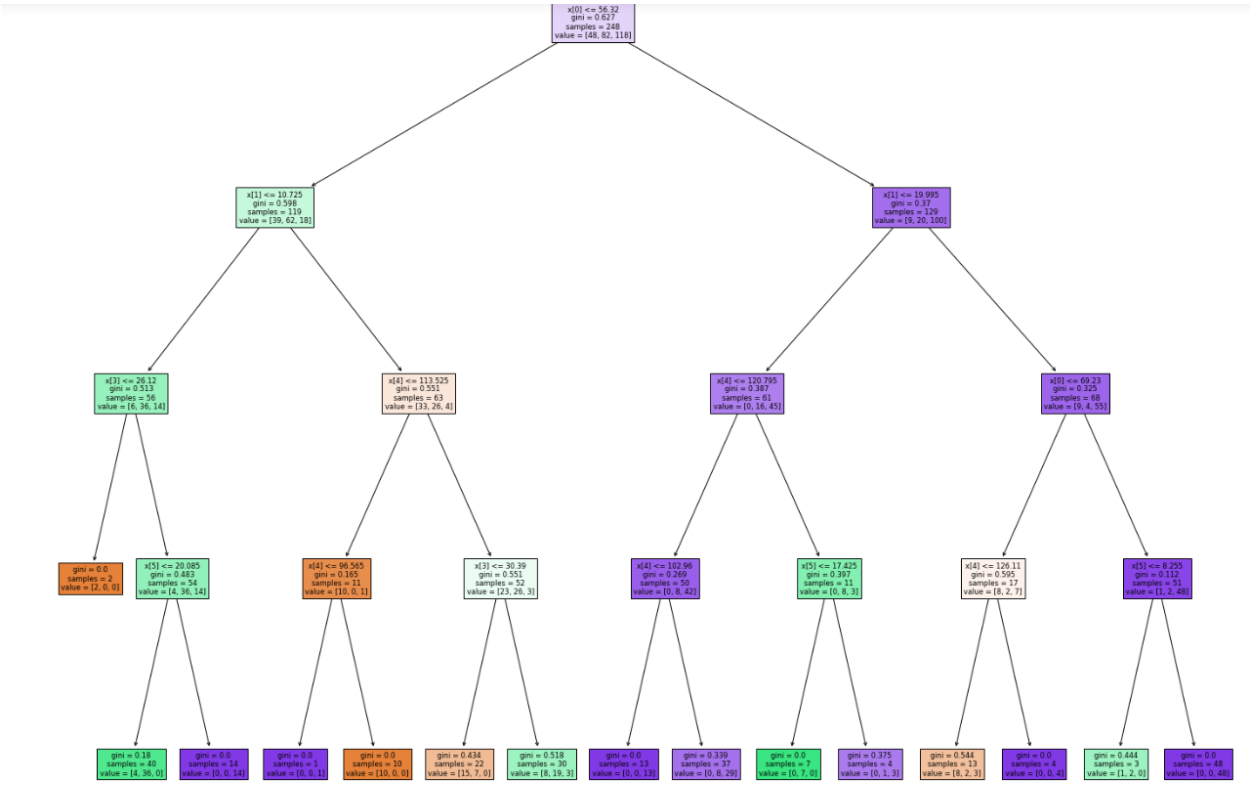
```
best_clf = cv_clf.best_estimator_
```

```
from sklearn import tree


best_clf = cv_clf.best_estimator_
plt.figure(figsize=(25,20))
tree.plot_tree(best_clf.estimators_[random.randint(0,len(best_clf.estimators_))],filled=True)
```



As we can see, this is the certain component of the random forest and it has maximum depth = 4 and has its individual way of deciding which attributes should be selected first and split. For comparison, this

is another component tree and it has different way of splitting and making decisions:



Next, I have selected the best decision tree model in the forest for the purpose of comparing its performance to the decision tree model:

```
decision_trees = best_clf.estimators_
len(decision_trees)

acc_scores = {}
for dtree in decision_trees:
    dtree.fit(X_train,y_train)
    preds = dtree.predict(X_test)
    acc = round(accuracy_score(y_test,preds),3)
    acc_scores[acc] = dtree
best_decision_tree = acc_scores[max(acc_scores.keys())]
```

Here, I have iterated over the estimators (trees) and selected the best one by its accuracy score.

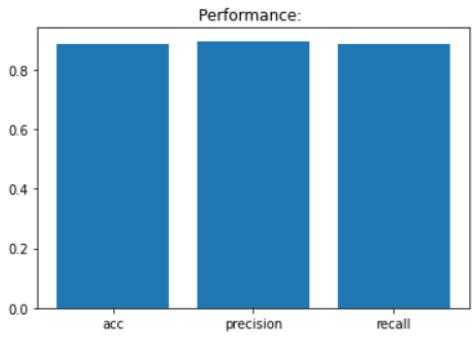
Next, I have used ‘display_scores’ helper function to display the accuracy, precision and recall scores of each model and visualized it using barchart:

```
from sklearn.metrics import accuracy_score, recall_score, precision_score



def display_scores(model):

    y_pred = model.predict(X_test)

    accuracy = round(accuracy_score(y_test,y_pred),3)
    recall = round(recall_score(y_test, y_pred, average = 'weighted'),3)
    precision = round(precision_score(y_test,y_pred, average = 'weighted'),3)

    scores = {'acc': accuracy,
              'precision' : precision,
              'recall': recall}

    print(f'''Accuracy: {accuracy}
Precision: {precision}
Recall: {recall}
    ''')
    plt.figure()
    plt.title('Performance: ')
    plt.bar(scores.keys(),scores.values())
    plt.show()
```

Now, let's display the scores of best decision tree and our initial random forest respectively:

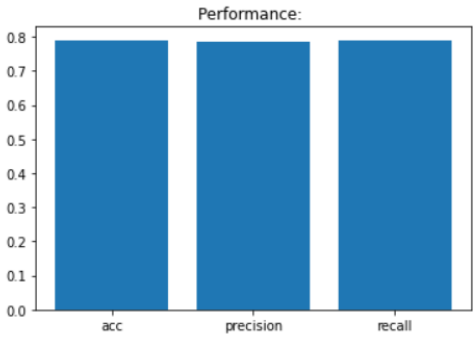
```
display_scores(best_decision_tree)
```

```
Accuracy: 0.887
Precision: 0.897
Recall: 0.887
```




```
display_scores(best_clf)
```

```
Accuracy: 0.79
Precision: 0.787
Recall: 0.79
```



We can see that the decision tree has higher accuracy, precision, and recall scores. It might happen because:

- Overfitting. It is common that individual decision trees are prone to overfitting. Random forests aims to reduce the overfitting by combining predictions of multiple decision trees
- Randomness. Random forests introduce randomness in the decision tree construction by sampling the features and observations. This randomness can sometimes result in decision trees with lower accuracy.

It is essential to consider various factors other than accuracy, recall, and precision scores when deciding on the preferred machine learning model for a given problem. While an individual decision tree in a random forest may have higher accuracy, recall, and precision scores, it may not necessarily be the best choice for the task at hand. Factors such as the size of the dataset, the generalization performance of the

models, and the interpretability of the model must be considered. Random forests are known for their ability to perform well in terms of generalization performance, making them suitable for handling large datasets with many features. Therefore, choosing a random forest over an individual decision tree with higher scores on some metrics may be a better choice in certain scenarios.

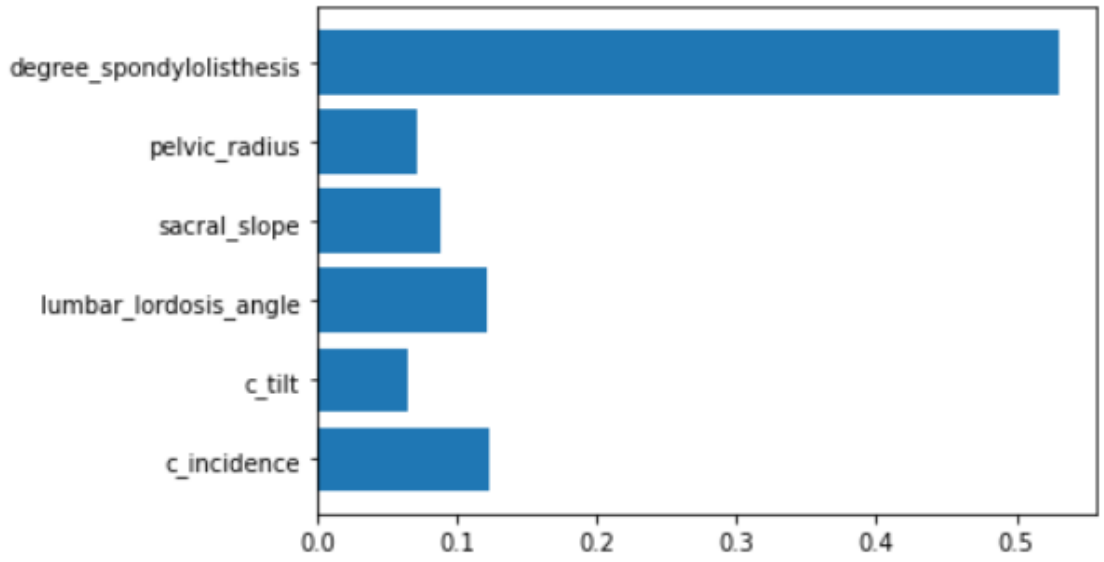
C) *For a random forest classifier (or one of its component trees), the relative importance of the attributes can be measured through the feature_importances_ field of the classifier. For selected component trees in (b), compare their associated lists of relative attribute importance values*

For this part, I have used a helper function 'display_feature_importances()' to display the feature importanes:

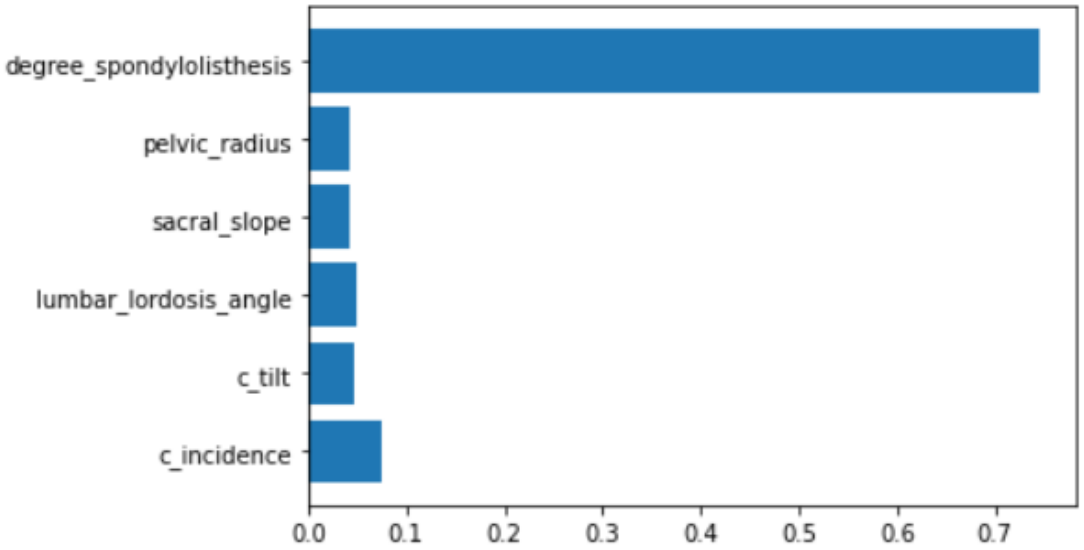
```
def display_feature_importances(model):
    feature_importances = model.feature_importances_
    #recall: from cell #3 -> X = df.drop(['category'], axis = 1)
    plt.barh(X.columns, feature_importances)
```

As discussed in the **last sentence of page 2**, a feature 'degree_spondylolithesis' might have the most importance in classifying whether a person is **SL** or **not.** Hence, it can be clearly seen from both selected decision tree and random forest (from b) that 'degree_spondylolithesis' has largest importance:

```
display_feature_importances(best_clf) #random_forest
```



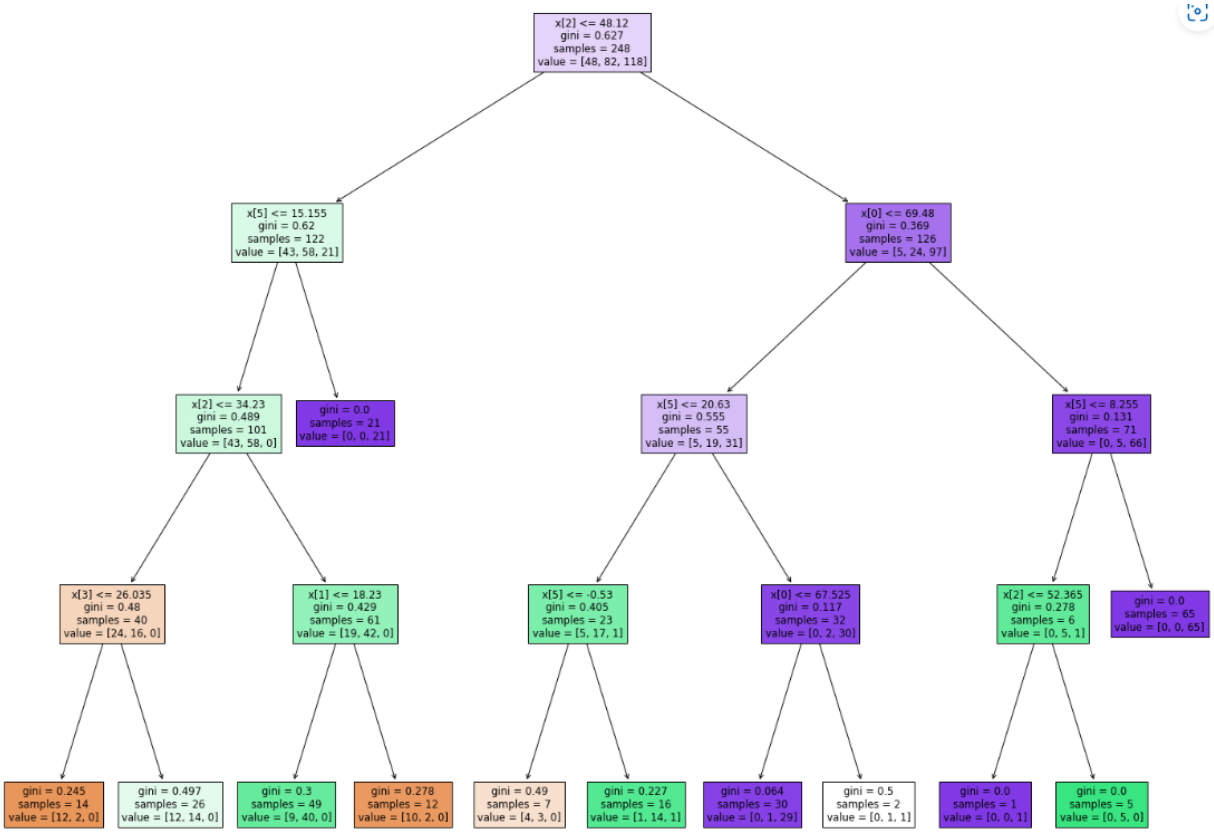
```
display_feature_importances(best_decision_tree) #decision_tree
```



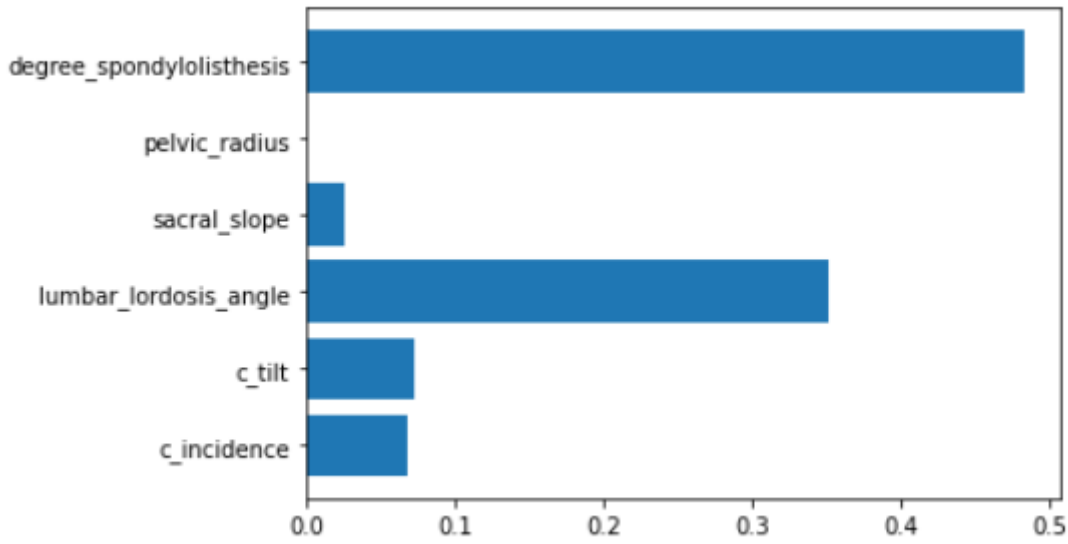
Additionally, it is also noticeable that 'c_incidence' has second largest importance in both models. However, It is insufficient to make conclusions about the importance of other features, as the permutation based method can have problem with highly-correlated features, it can report them as unimportant. If we select decision tree #10 in our random forest to see its feature importances,

```
random_index = random.randint(0,len(best_clf.estimators_))
random_tree = best_clf.estimators_[random_index]


display_feature_importances(random_tree)


plt.figure(figsize=(25,20))
tree.plot_tree(random_tree,filled=True)
print(f'Decision tree #{random_index}')
```

```
Decision tree #10
```



It displays the following result:



It is clearly visible that ‘lumbar_lordosis_angle’ has second biggest importance, while ‘pelvic_radius’ has zero importance.

D) *Construct a naïve Bayes classifier model based on our data set, and compare the classification performance with that of the random forest model*

For this part, I will use Gaussian naïve bayes classifer GaussianNB as provided from the assignment paper:

```
from sklearn.naive_bayes import GaussianNB

nb = GaussianNB()

params_nb = {'var_smoothing': [1e-9, 1e-8, 1e-7, 1e-6, 1e-1]}

cv_nb = GridSearchCV(nb,params_nb,cv=5)

cv_nb.fit(X_train,y_train)

print_results(cv_nb)
```

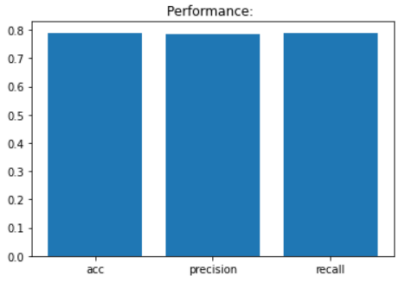
```
BEST PARAMS: {'var_smoothing': 1e-09}

0.827 (+/-0.147) for {'var_smoothing': 1e-09}
0.827 (+/-0.147) for {'var_smoothing': 1e-08}
0.827 (+/-0.147) for {'var_smoothing': 1e-07}
0.827 (+/-0.147) for {'var_smoothing': 1e-06}
0.77 (+/-0.177) for {'var_smoothing': 0.1}
```

```
cls_nb = cv_nb.best_estimator_
```

Using display_scores() functions, can compare accuracy, precision, recall scores of Naïve Bayes and Random Forest models:

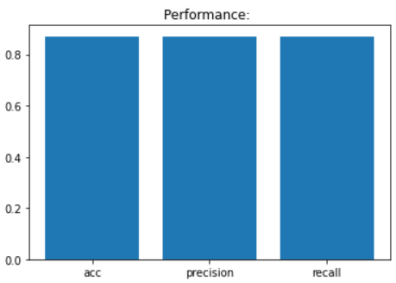
```
display_scores(best_clf) #random_forest
```

```
Accuracy: 0.79
Precision: 0.787
Recall: 0.79
```




```
display_scores(cls_nb) #naive bayes
```

```
Accuracy: 0.871
Precision: 0.871
Recall: 0.871
```

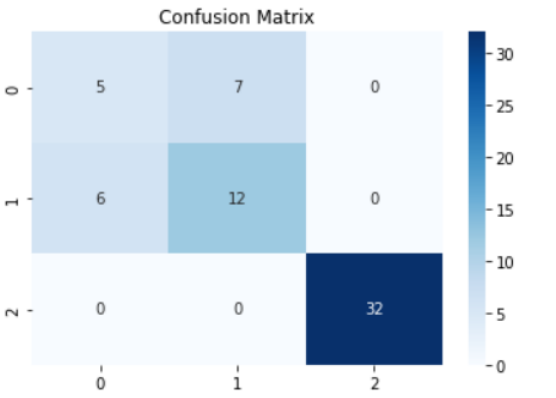


We can see that naïve bayes classifier has better performance for this particular dataset.

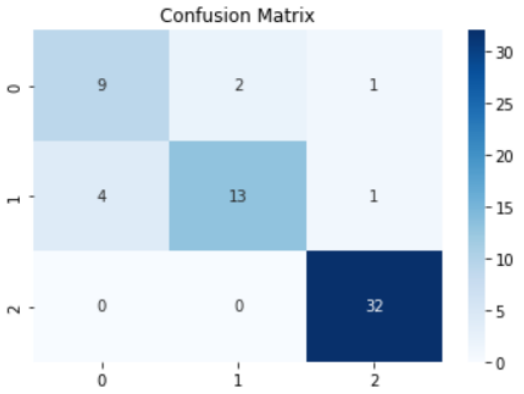
Additionally, I have used seaborn's heatmap function to plot the confusion matrices for the Naive Bayes and Random Forest models. The confusion matrix provides information about the number of true positives, true negatives, false positives, and false negatives for each class in the classification task:

```
from sklearn.metrics import confusion_matrix

def show_confusion_matrix(model):

    y_pred = model.predict(X_test)

    sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, cmap="Blues")
    plt.title("Confusion Matrix")
    plt.show()
```

```
show_confusion_matrix(best_clf) #random_forest
```




```
show_confusion_matrix(cls_nb) #naive bayes
```



Conclusion:

It can be concluded that the degree of spondylolisthesis is the most important feature for categorizing patients as normal, with a disk hernia, or with spondylolisthesis. Random Forest Classifier models were constructed with different numbers of component trees based on a specific training set/test set partition (80/20), and the resulting change in classification performance was analyzed. The best model was selected based on hyperparameters such as the number of decision trees in the forest = 50 and the maximum depth of the trees = 4.

The performance of individual decision trees in the model was compared to the overall performance of the Random Forest Classifier, and it was found that while individual decision trees may have higher accuracy, recall, and precision scores, Random Forest Classifiers are better suited for handling large datasets with many features. Finally, the feature_importances_ field of the Random Forest Classifier was used to measure the relative importance of the attributes, and it was found that the degree of spondylolisthesis had the highest importance for classification.

Finally, after comparing the performance of Naïve Bayes Classifier and Random Forest Classifier, it was concluded that Naïve Bayes classifier is more suitable to select as a model for this particular dataset, as it has better accuracy, recall and precision scores.