In [ ]:

```
# Kyphosis Recurrence Predictor
# Capstone Assignment on ML Model Building
# Building a Classifier Model in Python
# Compare and Analyse various Classifier Algorithms

# Problem Statement:
# Given a DataSet of 81 patients who have undergone a spinal surgery for a deformation
 and the data if the condition recurred, Build a claddification Model to predict whethe
r a patient being admitted for the surgery has chance for recurrence. This model will h
elp the surgeons to plan appropriate level of treatment to prevent recurrence.
# Data Set Description :
# The kyphosis data frame has 81 rows and 4 columns. representing data on children who
 have had corrective spinal surgery
# This data frame contains the following columns/Features:
# Kyphosis :a factor with levels absent present indicating if a kyphosis (a type of def
ormation) was present after the operation.
# Age :in months
# Number :the number of vertebrae involved
# Start :the number of the first (topmost) vertebra operated on.
# Algorithms Suggested:
# Decision Trees
# Random Forest
# KNN Classifier
# Logistic Regression
# Let us compare the accuracy and suggest the best classifier.
```

In [2]:

```
#LIBRARIES
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

In [3]:

```
#DATA
df = pd.read_csv("C:\\Users\\Derrik Jerry\\Documents\\Python Scripts\\kyphosis.csv")
df.head()
```

Out[3]:

	Kyphosis	Age	Number	Start
0	absent	71	3	5
1	absent	158	3	14
2	present	128	4	5
3	absent	2	5	1
4	absent	1	4	15

In [4]:

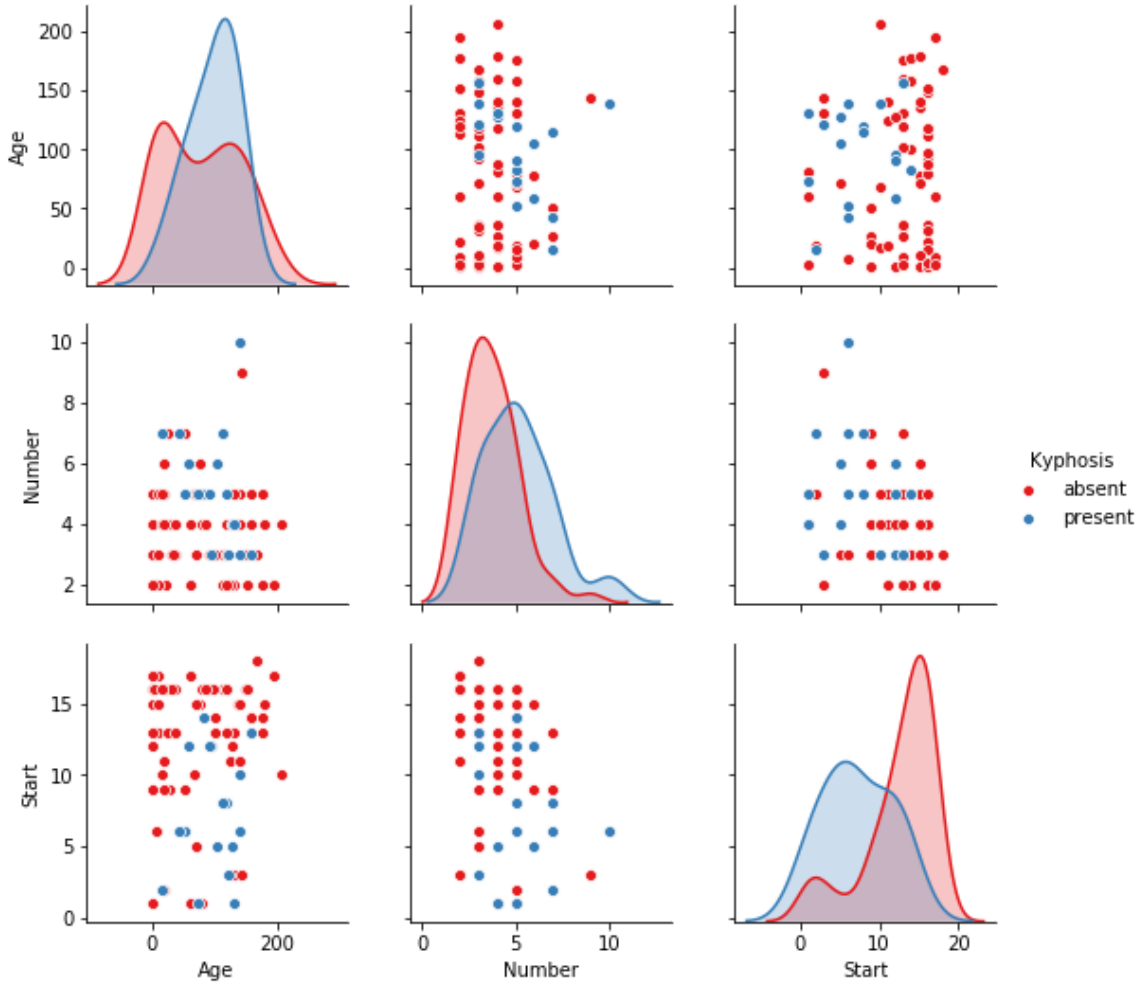
```
# EDA ANALAYSIS

sns.pairplot(df,hue='Kyphosis',palette='Set1')
```

```
C:\Users\Derrik Jerry\Anaconda3\lib\site-packages\scipy\stats\stats.py:171
3: FutureWarning: Using a non-tuple sequence for multidimensional indexing
is deprecated; use `arr[tuple(seq)]` instead of `arr[seq]`. In the future
this will be interpreted as an array index, `arr[np.array(seq)]`, which wi
ll result either in an error or a different result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```

Out[4]:

```
<seaborn.axisgrid.PairGrid at 0x2241dfd39e8>
```



In [6]:

```
# Train test split
from sklearn.model_selection import train_test_split
X = df.drop('Kyphosis',axis=1)
y = df['Kyphosis']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30,random_state=6
7)
```

In [7]:

```
# DECISION TREES
# We'll start just by training a single decision tree.

from sklearn.tree import DecisionTreeClassifier

dtree = DecisionTreeClassifier()
dtree.fit(X_train,y_train)
```

Out[7]:

```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=Non
e,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=Non
e,
            splitter='best')
```

In [9]:

```
# prediciton and Evaluation

predictions = dtree.predict(X_test)

from sklearn.metrics import classification_report,confusion_matrix

print(classification_report(y_test,predictions))
```

```
             precision    recall  f1-score   support

     absent       0.85      0.89      0.87        19
    present       0.60      0.50      0.55         6

avg / total       0.79      0.80      0.79        25
```

#precision we have 79% for decision tree

In [10]:

```
#CONFUSTION MATRIX
print(confusion_matrix(y_test,predictions))
```

```
[[17  2]
 [ 3  3]]
```

In [12]:

```
#RANDOM FOREST

from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier(n_estimators=100)
rfc.fit(X_train, y_train)
```

Out[12]:

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gin
i',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=100, n_jobs=1,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False)
```

In [13]:

```
# prediction and confusion matrix
rfc_pred = rfc.predict(X_test)
print(confusion_matrix(y_test,rfc_pred))
```

```
[[18  1]
 [ 3  3]]
```

In [14]:

```
print(classification_report(y_test,rfc_pred))
```

```
             precision    recall  f1-score   support

     absent       0.86      0.95      0.90        19
    present       0.75      0.50      0.60         6

avg / total       0.83      0.84      0.83        25
```

In [15]:

```
from sklearn.linear_model import LogisticRegression
logmodel = LogisticRegression()
logmodel.fit(X_train,y_train)
```

Out[15]:

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=Tru
e,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False)
```

In [16]:

```
#evaluate the Logistic Regression Classifier
predictions = logmodel.predict(X_test)

from sklearn.metrics import classification_report
print(classification_report(y_test,predictions))
```

```
             precision    recall  f1-score   support

     absent       0.86      0.95      0.90        19
    present       0.75      0.50      0.60         6

avg / total       0.83      0.84      0.83        25
```

In [18]:

```
##KNN CLASSIFICAION
from sklearn.preprocessing import StandardScaler
#Standardize  the data to a common scale
scaler= StandardScaler()
scaler.fit(df.drop('Kyphosis',axis=1))
scaled_features= scaler.transform(df.drop('Kyphosis',axis=1))
df_feat=pd.DataFrame(scaled_features,columns=df.columns[1:])
```

In [20]:

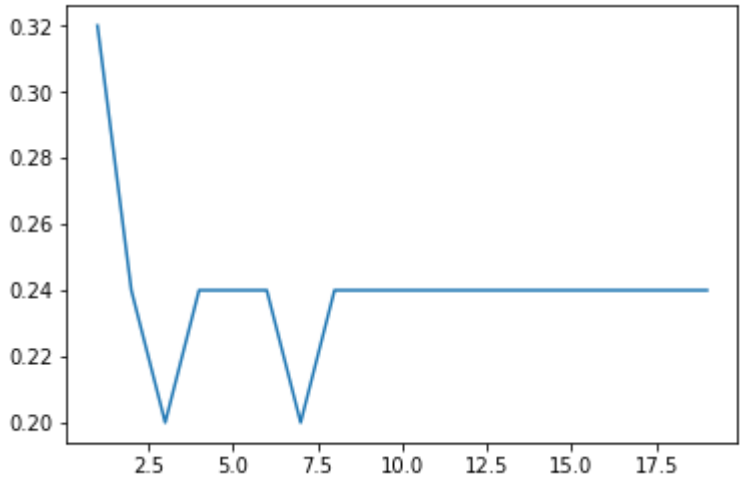
```
error_rate=[]
from sklearn.neighbors import KNeighborsClassifier
for i in range(1,20):
 knn=KNeighborsClassifier(n_neighbors=i)
 knn.fit(X_train,y_train)
 pred_i = knn.predict(X_test)
 error_rate.append(np.mean(pred_i != y_test))
```

In [21]:

```
plt.plot(range(1,20),error_rate)
```

Out[21]:

```
[<matplotlib.lines.Line2D at 0x2241ff83f60>]
```



In [22]:

```
#Build the Model with 3 as K
knn=KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train,y_train)

# Predict and Evaluate the Model
pred = knn.predict(X_test)
print(confusion_matrix(y_test,pred))
print(classification_report(y_test,pred))
```

```
[[19  0]
 [ 5  1]]
             precision    recall  f1-score   support

     absent       0.79      1.00      0.88        19
    present       1.00      0.17      0.29         6

avg / total       0.84      0.80      0.74        25
```

In [ ]:

```
#conclusion

The Accuracy Levels of the Models is observed to be:
1. Decision Tree Classifier - 79%
2. Random Forest Classifier - 83%
3. Logistic Regression Classifier - 83%
4. KNN Classifier - 84%
###Out of the four, knn Classifier gives out 84% Accuracy. SO For the Kyphosis Recurren
ce Predictor we are building, we sugges KNN Classifier Algorithm.
```