UL-Inclass-day-2

March 11, 2020

[85]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns; sns.set(style="ticks", color_codes=True)
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from scipy.stats import zscore
import sklearn.metrics
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
import  warnings
warnings.filterwarnings("ignore")
```

[86]:
```
df=pd.read_csv('2Classdata.csv')
df.head()
```

[86]:

	pelvic_incidence	pelvic_tilt numeric	lumbar_lordosis_angle	sacral_slope	pelvic_radius	degree_spondylolisthesis	class
0	63.027818	22.552586	39.609117	40.475232	98.672917	-0.254400	Abnormal
1	39.056951	10.060991	25.015378	28.995960	114.405425	4.564259	Abnormal
2	68.832021	22.218482	50.092194	46.613539	105.985135	-3.530317	Abnormal
3	69.297008	24.652878	44.311238	44.644130	101.868495	11.211523	Abnormal
4	49.712859	9.652075	28.317406	40.060784	108.168725	7.918501	Abnormal

1 reading of data

[87]:
```
df1=df.copy()
df1.head()
```

[87]:

	pelvic_incidence	pelvic_tilt numeric	lumbar_lordosis_angle	sacral_slope	\
0	63.027818	22.552586	39.609117	40.475232	
1	39.056951	10.060991	25.015378	28.995960	
2	68.832021	22.218482	50.092194	46.613539	
3	69.297008	24.652878	44.311238	44.644130	
4	49.712859	9.652075	28.317406	40.060784	

	pelvic_radius	degree_spondylolisthesis	class
0	98.672917	-0.254400	Abnormal
1	114.405425	4.564259	Abnormal
2	105.985135	-3.530317	Abnormal
3	101.868495	11.211523	Abnormal
4	108.168725	7.918501	Abnormal

[88]:
```
df1['class']=pd.get_dummies(df['class'])
```

[89]:
```
df1.head()
```

[89]:

	pelvic_incidence	pelvic_tilt numeric	lumbar_lordosis_angle	sacral_slope	\
0	63.027818	22.552586	39.609117	40.475232	
1	39.056951	10.060991	25.015378	28.995960	
2	68.832021	22.218482	50.092194	46.613539	
3	69.297008	24.652878	44.311238	44.644130	
4	49.712859	9.652075	28.317406	40.060784	

	pelvic_radius	degree_spondylolisthesis	class
0	98.672917	-0.254400	1
1	114.405425	4.564259	1
2	105.985135	-3.530317	1
3	101.868495	11.211523	1
4	108.168725	7.918501	1

[90]:
```
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 310 entries, 0 to 309
Data columns (total 7 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   pelvic_incidence         310 non-null    float64
 1   pelvic_tilt numeric      310 non-null    float64
 2   lumbar_lordosis_angle    310 non-null    float64
```

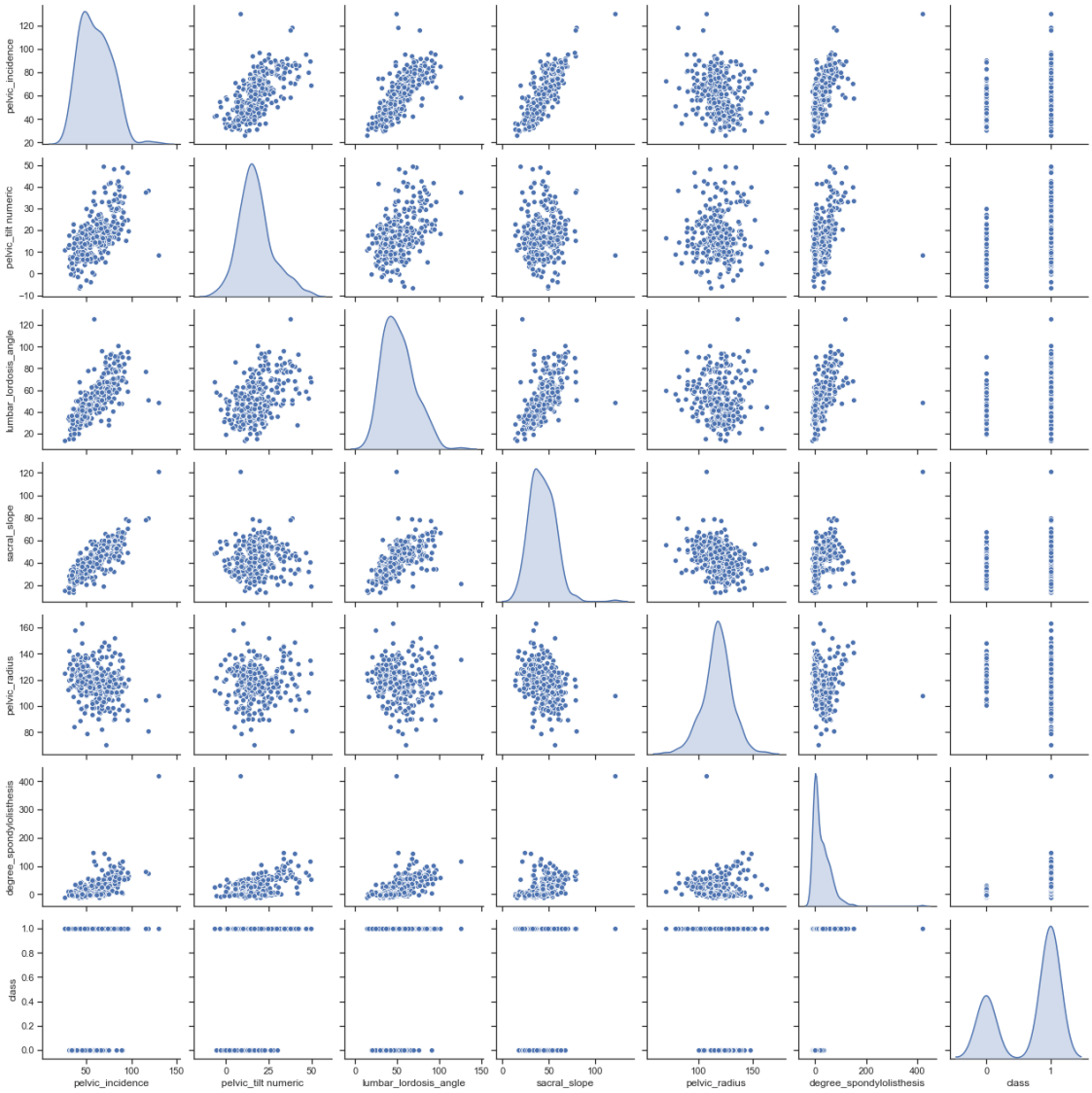
```
 3   sacral_slope              310 non-null    float64
 4   pelvic_radius             310 non-null    float64
 5   degree_spondylolisthesis  310 non-null    float64
 6   class                     310 non-null    uint8
dtypes: float64(6), uint8(1)
memory usage: 15.0 KB
```

[91]:
```
df1.isnull().sum()
```

[91]:
```
pelvic_incidence            0
pelvic_tilt numeric         0
lumbar_lordosis_angle       0
sacral_slope                0
pelvic_radius               0
degree_spondylolisthesis    0
class                       0
dtype: int64
```

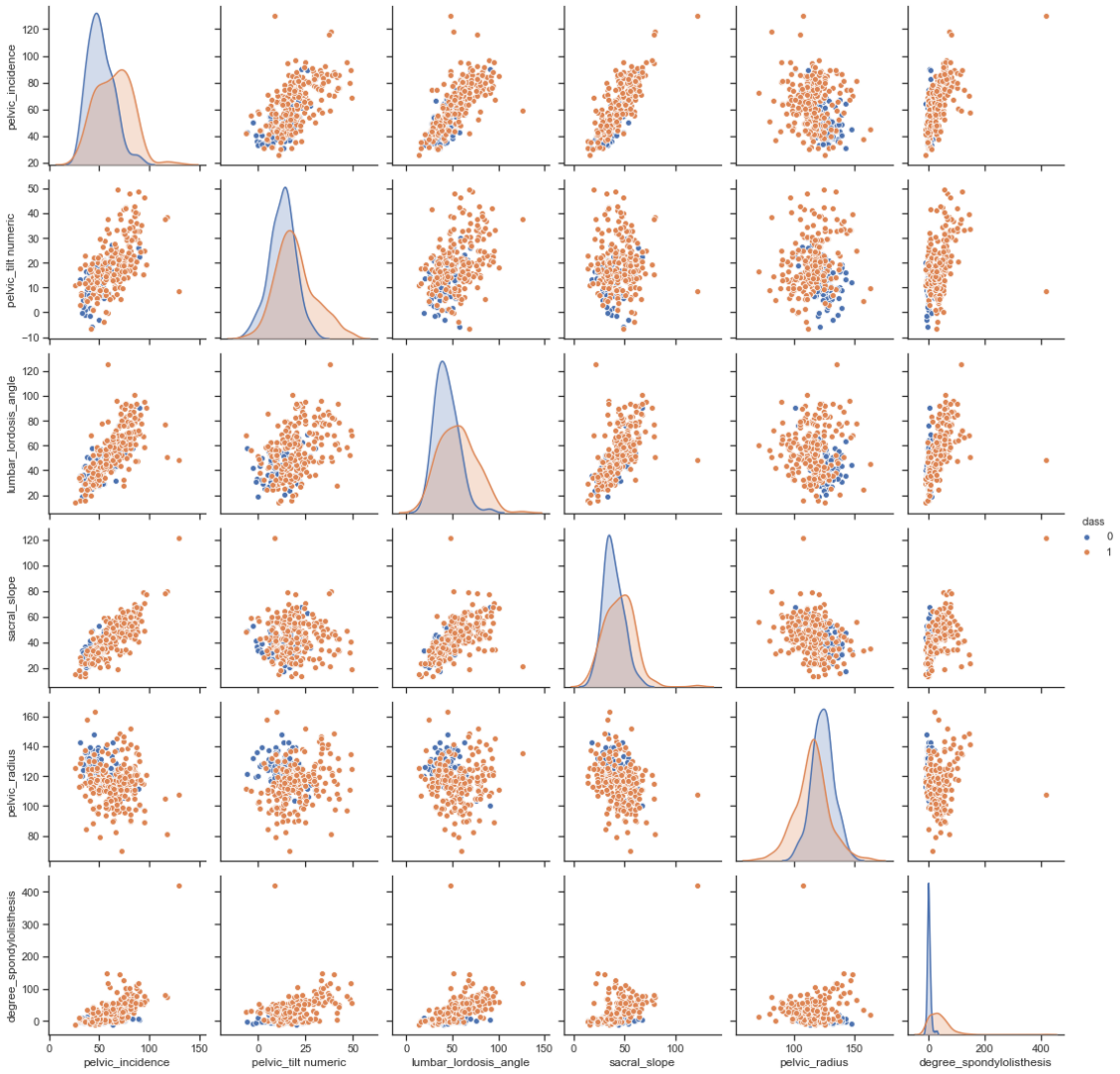
[92]:
```
import seaborn as sns
sns.pairplot(df1,diag_kind='kde')
```

[92]:
```
<seaborn.axisgrid.PairGrid at 0x23bda190400>
```



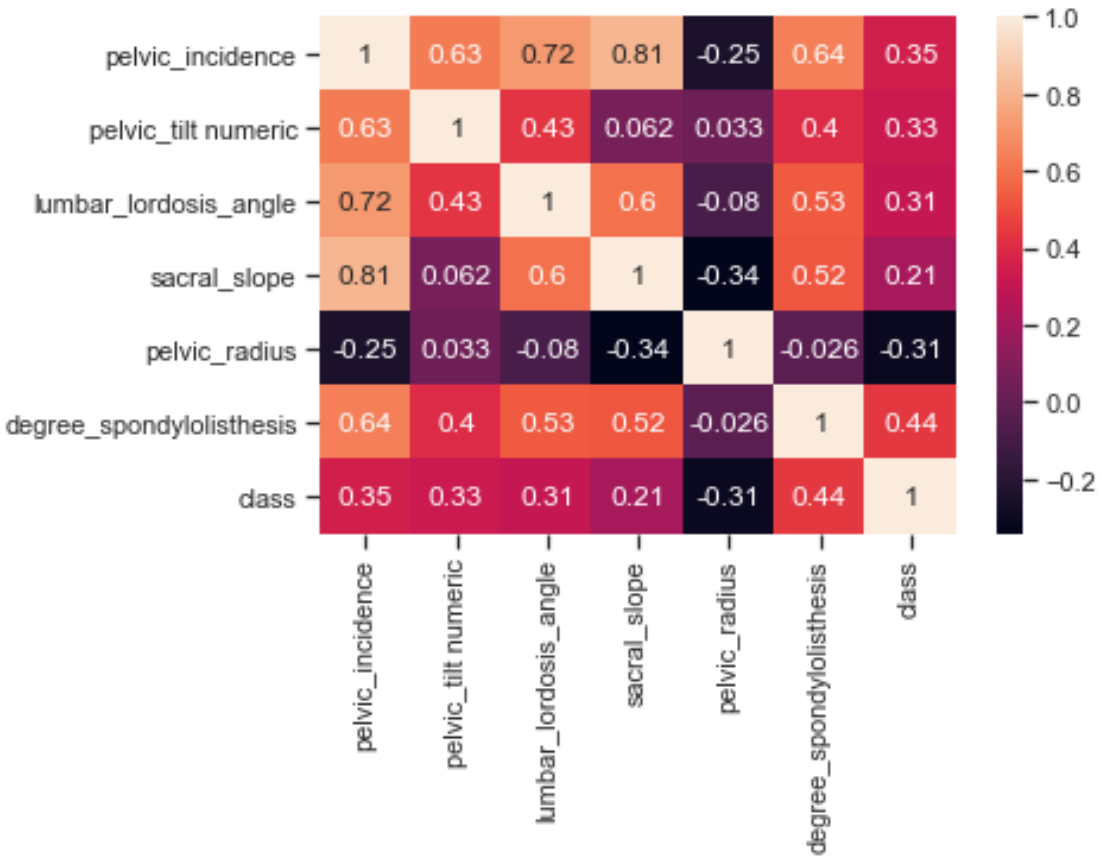
[93]:
```
sns.pairplot(df1,diag_kind='kde', hue='class')
```

[93]: <seaborn.axisgrid.PairGrid at 0x23bda184780>



[94]:
```
sns.heatmap(df1.corr(),annot=True)
```

[94]: <matplotlib.axes._subplots.AxesSubplot at 0x23bdd5f6a20>



[95]:
```
df_scaled = df1.apply(zscore)
```

[96]:
```
df_scaled.head()
```

[96]:

	pelvic_incidence	pelvic_tilt numeric	lumbar_lordosis_angle	sacral_slope	\
0	0.147086	0.501369	-0.665177	-0.184950	
1	-1.245864	-0.748769	-1.453001	-1.041521	
2	0.484370	0.467932	-0.099262	0.273083	
3	0.511390	0.711562	-0.411339	0.126128	
4	-0.626648	-0.789693	-1.274745	-0.215876	

	pelvic_radius	degree_spondylolisthesis	class
0	-1.447647	-0.708059	0.690066
1	-0.264385	-0.579556	0.690066
2	-0.897686	-0.795421	0.690066
3	-1.207303	-0.402288	0.690066
4	-0.733455	-0.490106	0.690066

K-MEANS CLUSTERING ALGORITHM:

Finding the best K value:

[97]:
```
model = KMeans(n_clusters = 3)
model
```

[97]:
```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
       n_clusters=3, n_init=10, n_jobs=None, precompute_distances='auto',
       random_state=None, tol=0.0001, verbose=0)
```

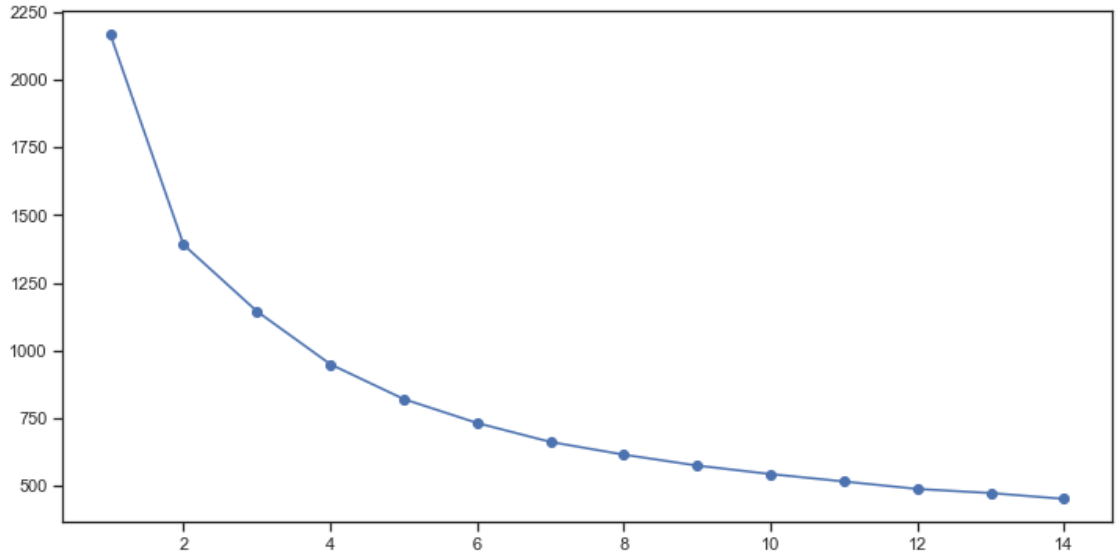
[184]:
```
cluster_range = range( 1, 15 )
cluster_errors = []
for num_clusters in cluster_range:
  clusters = KMeans( num_clusters, n_init = 10 )
  clusters.fit(df_scaled)
  labels = clusters.labels_
  centroids = clusters.cluster_centers_
  cluster_errors.append( clusters.inertia_ )
clusters_df = pd.DataFrame( { "num_clusters":cluster_range, "cluster_errors":␣
 ↪cluster_errors } )
clusters_df[0:15]
```

[184]:

	num_clusters	cluster_errors
0	1	2170.000000
1	2	1390.613323
2	3	1144.373088
3	4	948.686227
4	5	819.948533
5	6	731.346852
6	7	661.204500
7	8	614.232600
8	9	573.813822
9	10	542.402881
10	11	514.985561
11	12	487.326659
12	13	472.131678
13	14	450.485634

[185]:
```
# Elbow plot

plt.figure(figsize=(12,6))
plt.plot( clusters_df.num_clusters, clusters_df.cluster_errors, marker = "o" )
```

[185]:
```
[<matplotlib.lines.Line2D at 0x23bd9ed87b8>]
```



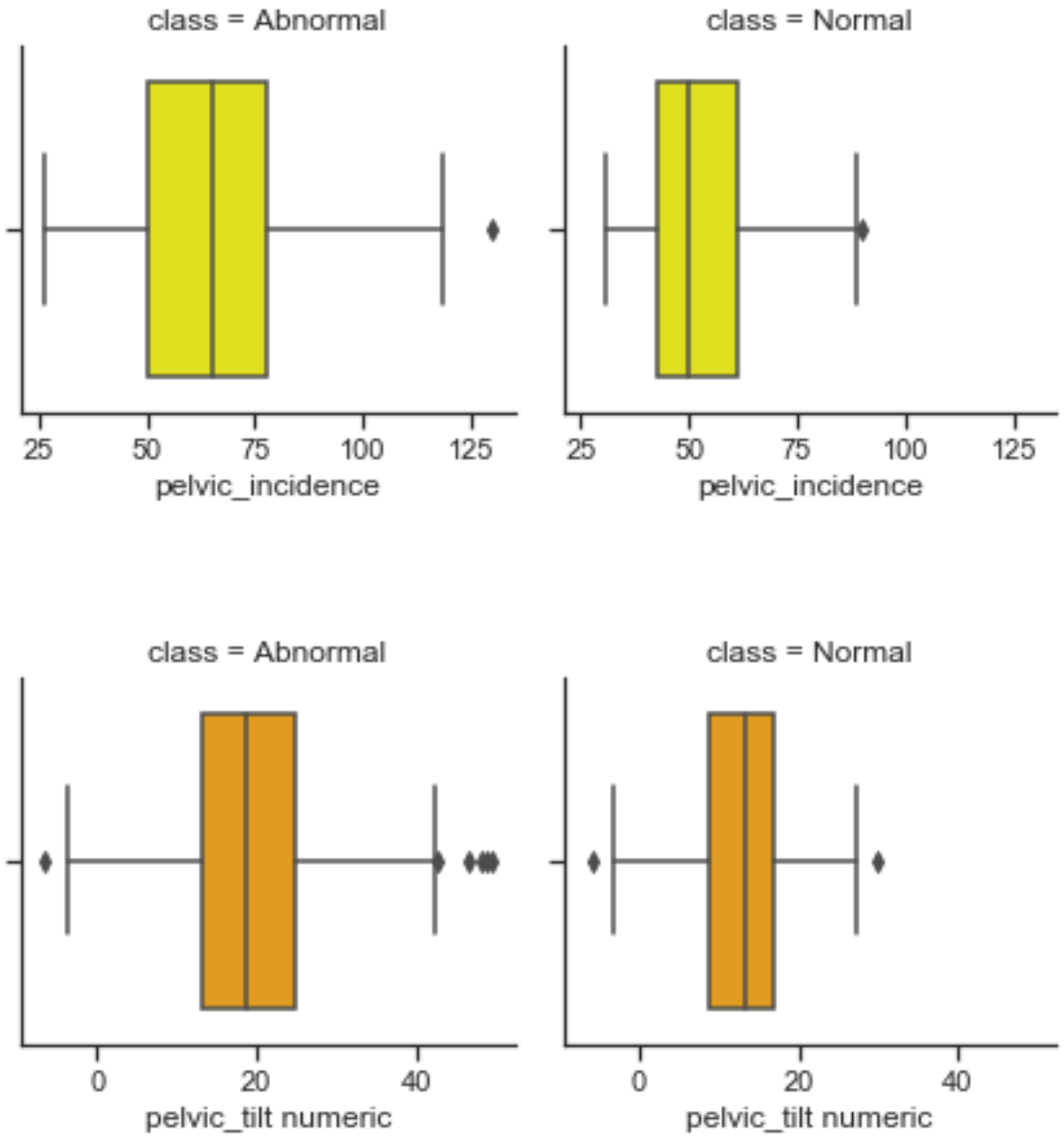
[186]:
```
df1.columns
```

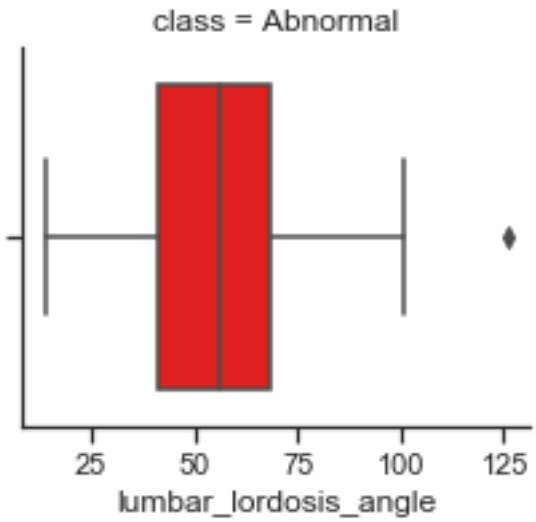
[186]:
```
Index(['pelvic_incidence', 'pelvic_tilt numeric', 'lumbar_lordosis_angle',
       'sacral_slope', 'pelvic_radius', 'degree_spondylolisthesis', 'class'],
      dtype='object')
```

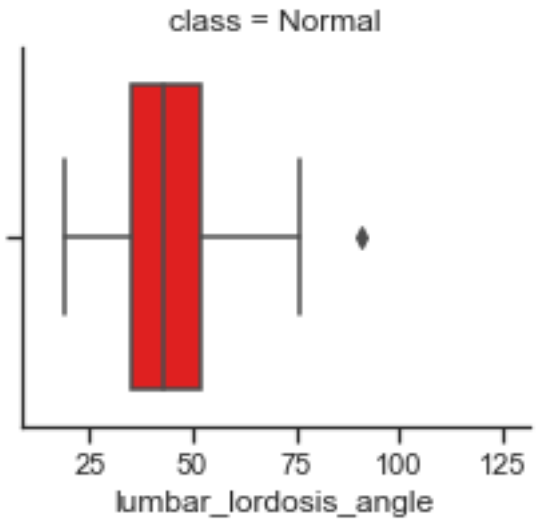
here we are taking the value of k as 2

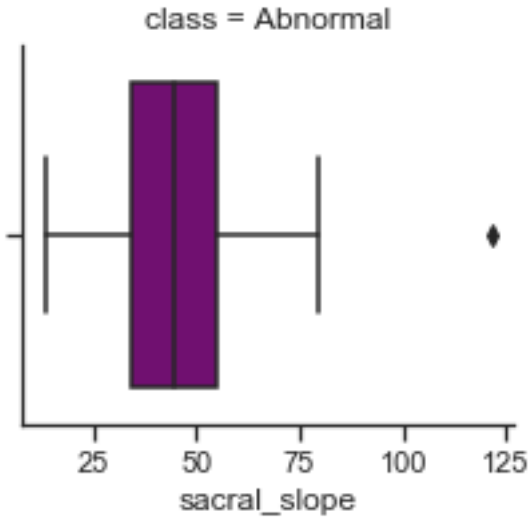
[187]:
```
pi = sns.FacetGrid(df, col='class')
pi.map(sns.boxplot, 'pelvic_incidence', color='yellow', order=['0', '1'])

ptn= sns.FacetGrid(df, col='class')
ptn.map(sns.boxplot, 'pelvic_tilt numeric', color='orange', order=['0', '1'])

lla = sns.FacetGrid(df, col='class')
lla.map(sns.boxplot, 'lumbar_lordosis_angle', color='red', order=['0', '1'])

ss = sns.FacetGrid(df, col='class')
ss.map(sns.boxplot, 'sacral_slope', color='purple', order=['0', '1'])

pr = sns.FacetGrid(df, col='class')
pr.map(sns.boxplot, 'pelvic_radius', color='blue', order=['0', '1'])

ds = sns.FacetGrid(df, col='class')
ds.map(sns.boxplot, 'degree_spondylolisthesis', color='cyan', order=['0', '1'])
```

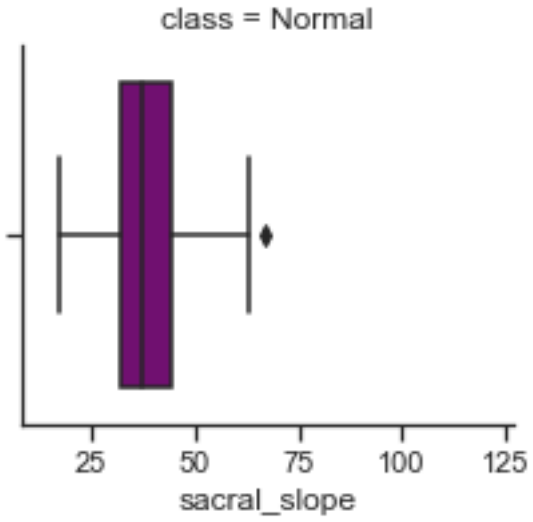
[187]:
```
<seaborn.axisgrid.FacetGrid at 0x23bd85a6128>
```

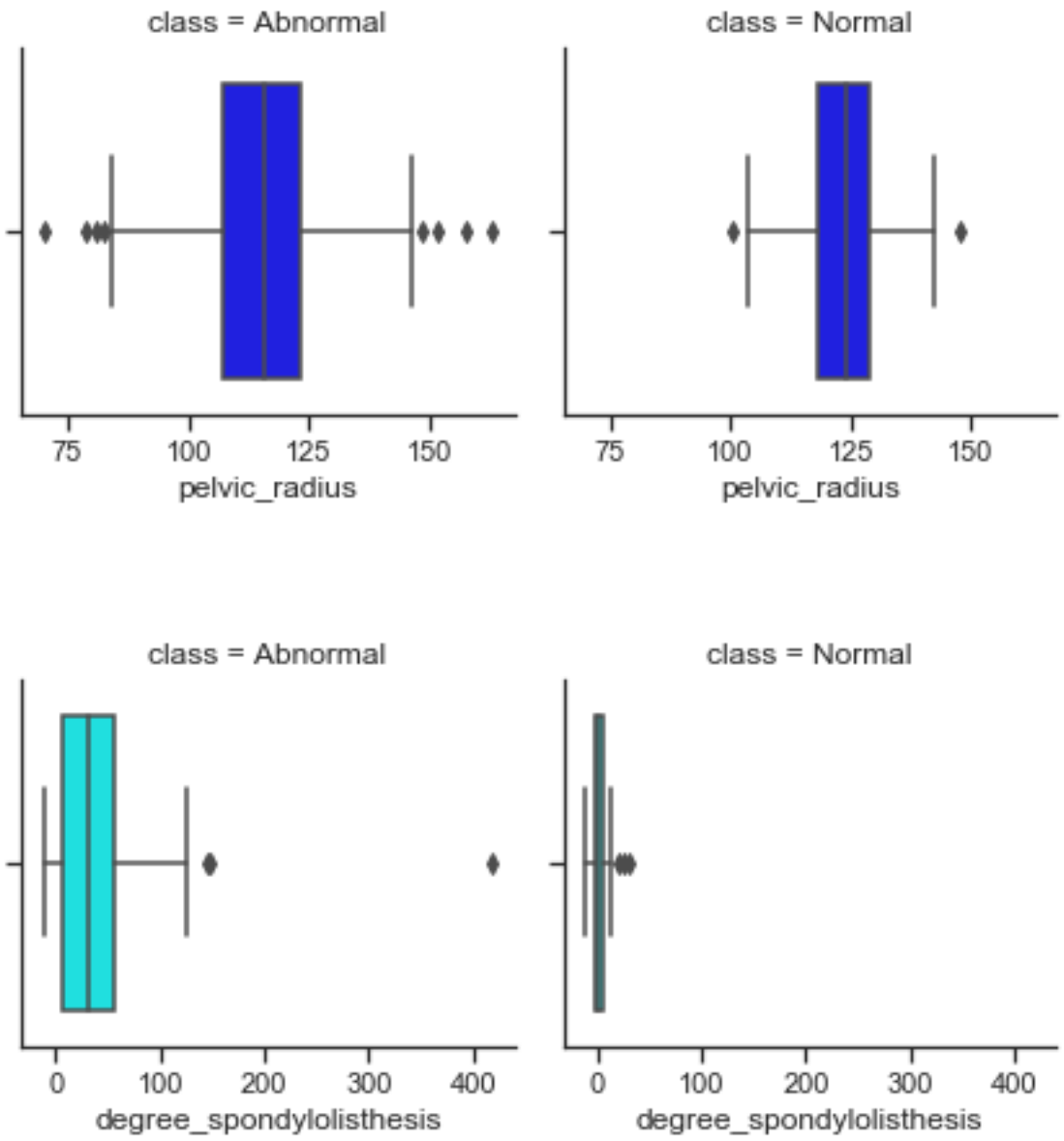

class = Abnormal
25
50
75
100
125
pelvic_incidence
class = Normal
25
50
75
100
125
pelvic_incidence
class = Abnormal
0
20
40
pelvic_tilt numeric
class = Normal
0
20
40
pelvic_tilt numeric


class = Abnormal
25
50
75
100
125
lumbar_lordosis_angle


class = Normal
25
50
75
100
125
lumbar_lordosis_angle

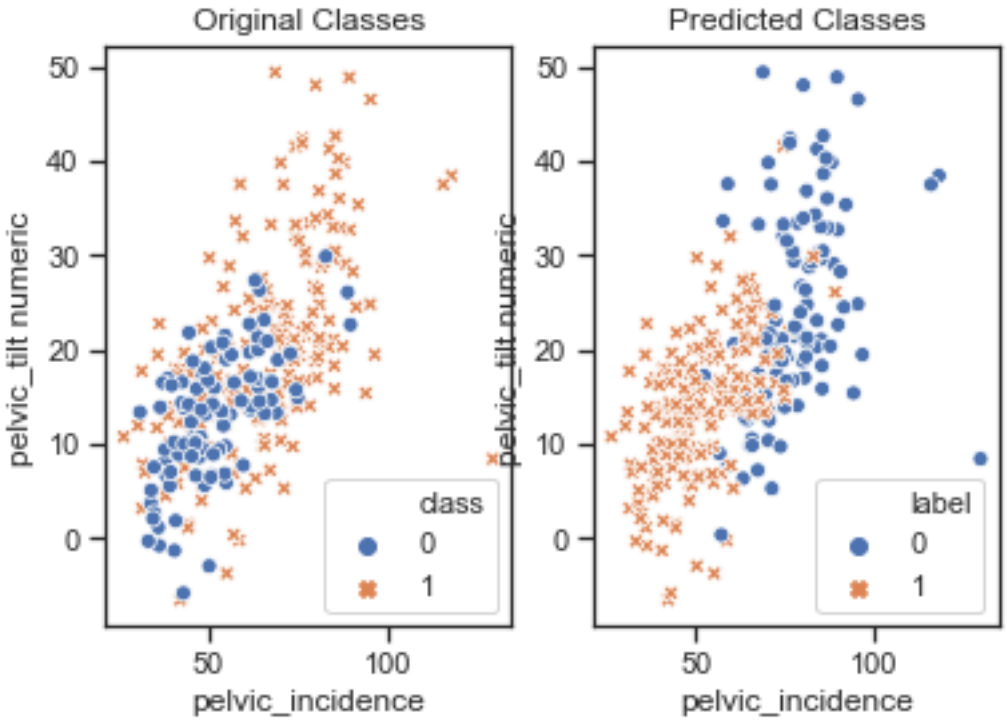

class = Abnormal
25
50
75
100
125
sacral_slope


class = Normal
25
50
75
100
125
sacral_slope



[188]:
```
# Now we know our best k value is 2, I am creating a new kmeans model:
kmeans2 = KMeans(n_clusters=2)

# Training the model:
clusters = kmeans2.fit_predict(df1)

# Adding a label feature with the predicted class values:
df_k = df1.copy(deep=True)
df_k['label'] = clusters
```

[190]:
```
fig, (ax1, ax2) = plt.subplots(1,2)

ax1 = plt.subplot(1,2,1)
plt.title('Original Classes')
sns.scatterplot(x='pelvic_incidence', y='pelvic_tilt numeric', hue='class',␣
 ↪style='class', data=df1, ax=ax1)

ax2 = plt.subplot(1,2,2)
plt.title('Predicted Classes')
sns.scatterplot(x='pelvic_incidence', y='pelvic_tilt numeric', hue='label',␣
 ↪style='label', data=df_k, ax=ax2)
plt.show()
```



[191]:
```
print('Original Data Classes:')
print(df1['class'].value_counts())
print('-' * 30)
print('Predicted Data Classes:')
print(df_k.label.value_counts())
```

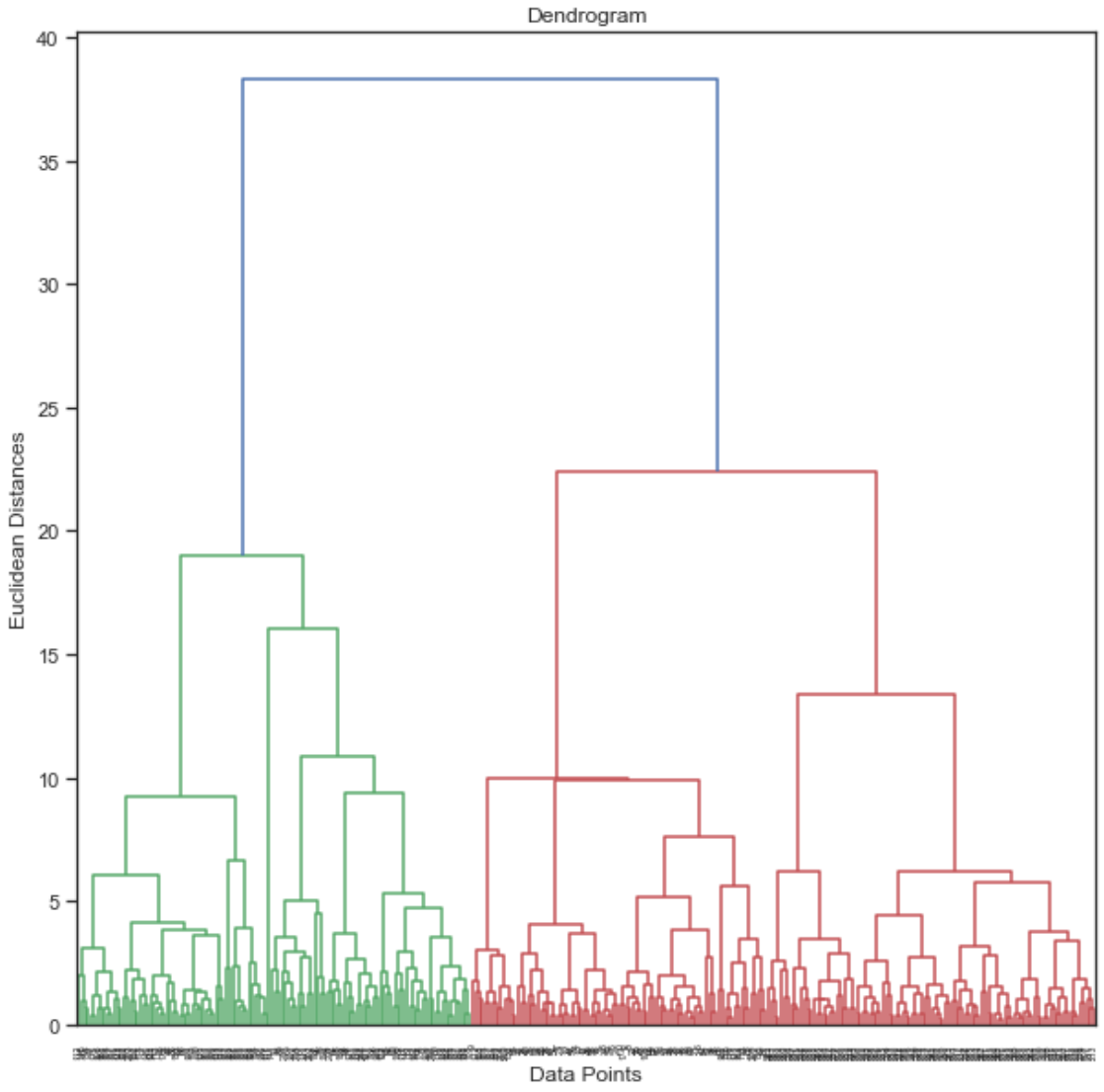
```
Original Data Classes:
1    210
0    100
Name: class, dtype: int64
------------------------------
```

```
Predicted Data Classes:
1    200
0    110
Name: label, dtype: int64
```

HIERARCHICAL CLUSTERING ALGORITHM:

Creating the Dendrogram:

[192]:
```
from scipy.cluster.hierarchy import linkage, dendrogram
plt.figure(figsize=[10,10])
merg = linkage(df_scaled, method='ward')
dendrogram(merg, leaf_rotation=90)
plt.title('Dendrogram')
plt.xlabel('Data Points')
plt.ylabel('Euclidean Distances')
plt.show()
```



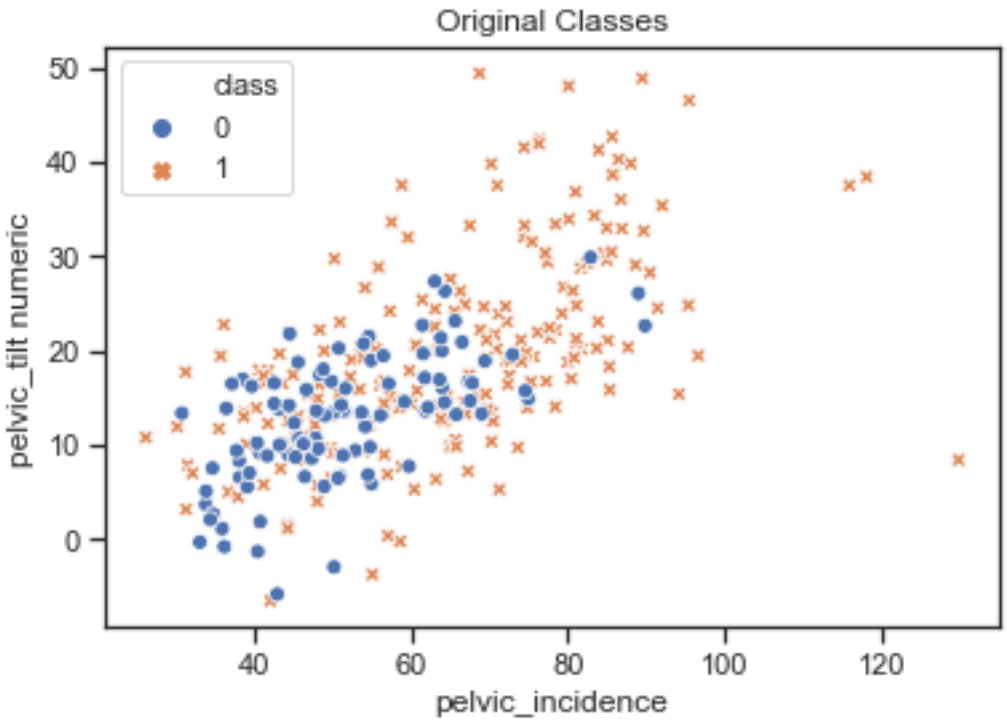
From the dendrogram we can read there are 2 classes in our data set.

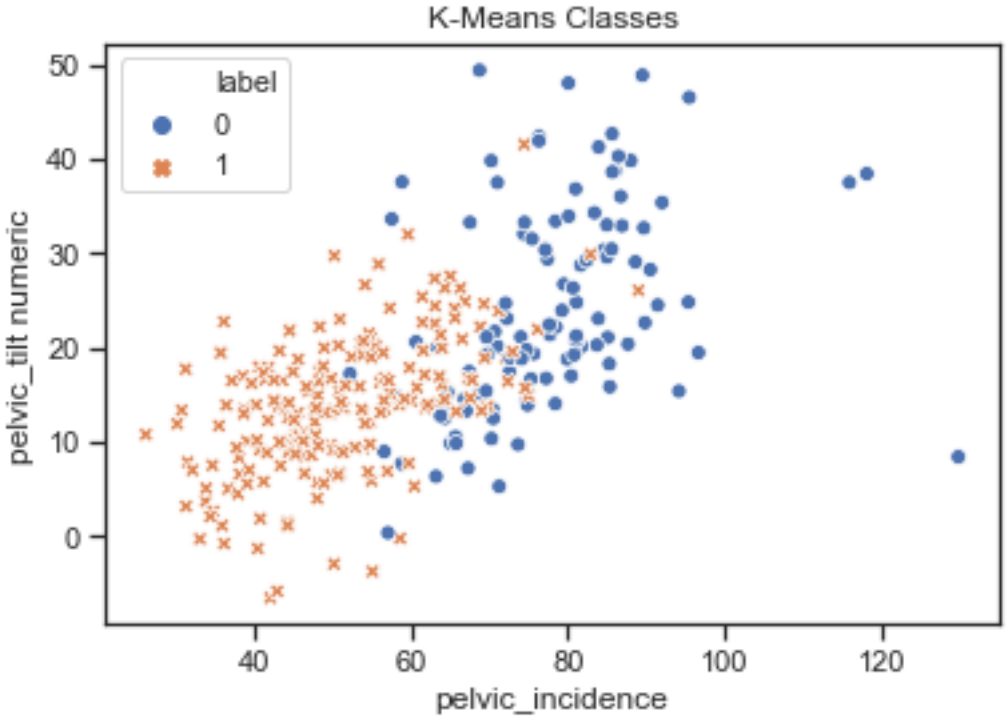
Hierarchical Clustering Algorithm:

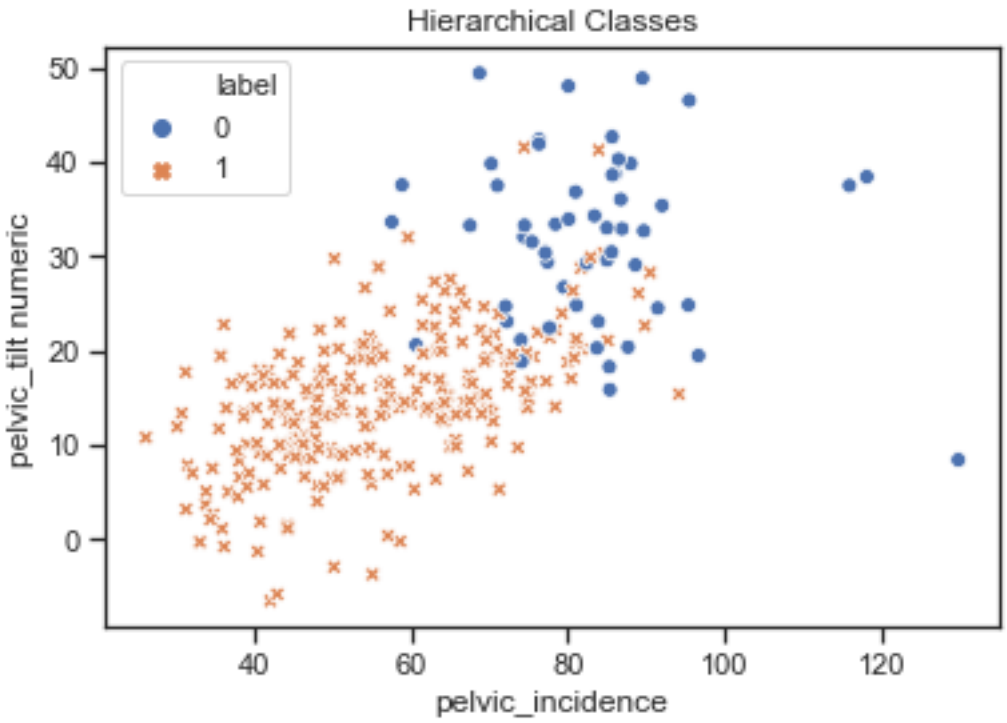
[193]:
```
from sklearn.cluster import AgglomerativeClustering

hie_clus = AgglomerativeClustering(n_clusters=2, affinity='euclidean',␣
↪linkage='ward')
cluster2 = hie_clus.fit_predict(df1)

df_h = df.copy(deep=True)
df_h['label'] = cluster2
```

Comparing Original, K-Means and Hierarchical Clustered Classes:

[194]:
```
plt.title('Original Classes')
sns.scatterplot(x='pelvic_incidence', y='pelvic_tilt numeric', hue='class',␣
↪style='class', data=df1)
plt.show()
plt.title('K-Means Classes')
sns.scatterplot(x='pelvic_incidence', y='pelvic_tilt numeric', hue='label',␣
↪style='label', data=df_k)
plt.show()
plt.title('Hierarchical Classes')
sns.scatterplot(x='pelvic_incidence', y='pelvic_tilt numeric', hue='label',␣
↪style='label', data=df_h)
plt.show()
```


Original Classes
class
0
1
pelvic_tilt numeric
pelvic_incidence
50
40
30
20
10
0
40
60
80
100
120


K-Means Classes
label
0
1
pelvic_tilt numeric
pelvic_incidence
50
40
30
20
10
0
40
60
80
100
120



[195]:
```
print('Original Data Classes:')
print(df1['class'].value_counts())
print('-' * 30)
print('K-Means Predicted Data Classes:')
print(df_k.label.value_counts())
print('-' * 30)
print('Hierarchical Predicted Data Classes:')
print(df_h.label.value_counts())
```

```
Original Data Classes:
1    210
0    100
Name: class, dtype: int64
------------------------------
K-Means Predicted Data Classes:
1    200
0    110
Name: label, dtype: int64
------------------------------
Hierarchical Predicted Data Classes:
1    256
0     54
Name: label, dtype: int64
```

We can see our models' differences from the comparision of our algorithms' class counts

2 Build An Classification model :-

3 K-Means

[196]:
```
df_k.sample(5)
```

[196]:
```
     pelvic_incidence  pelvic_tilt numeric  lumbar_lordosis_angle  \
133         81.754419            20.123466              70.560440
64          76.147212            21.936186              82.961502
200         63.364339            20.024621              67.498705
30          50.819268            15.402213              42.528939
227         61.540599            19.676957              52.892229

     sacral_slope  pelvic_radius  degree_spondylolisthesis  class  label
133     61.630954     119.425086                 55.506889      1      0
64      54.211027     123.932010                 10.431972      1      1
200     43.339718     130.999258                 37.556706      1      0
30      35.417055     112.192804                 10.869566      1      1
227     41.863642     118.686268                  4.815031      0      1
```

Data Preparation(Splitting the Dependent/Target Variable and the Independent Variables)

[198]:
```
x= df_k.drop('label',axis=1)
y= df_k['label']
```

[199]:
```
test_size = 0.30 # taking 70:30 training and test set
seed = 7  # Random numbmer seeding for reapeatability of the code
x_train, x_validate, y_train, y_validate = train_test_split(x, y,␣
↪test_size=test_size, random_state=seed)
```

[200]:
```
from sklearn.preprocessing import StandardScaler
independent_scalar = StandardScaler()
x_train = independent_scalar.fit_transform (x_train) #fit and transform
x_validate = independent_scalar.transform (x_validate) # only transform
```

Decision Tree Classifier

[201]:
```
from sklearn.tree import DecisionTreeClassifier
#DecisionTreeClassifier is the corresponding Classifier
Dtree = DecisionTreeClassifier(max_depth=3)
Dtree.fit (x_train, y_train)
```

[201]:
```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=3, 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='deprecated',
                       random_state=None, splitter='best')
```

[202]:
```
predictValues_train = Dtree.predict(x_train)
#print(predictValues_train)
accuracy_train=accuracy_score(y_train, predictValues_train)



predictValues_validate = Dtree.predict(x_validate)
#print(predictValues_validate)
accuracy_validate=accuracy_score(y_validate, predictValues_validate)

print("Train Accuracy  :: ",accuracy_train)
print("Validation Accuracy  :: ",accuracy_validate)
```

```
Train Accuracy  ::  0.9953917050691244
Validation Accuracy  ::  0.946236559139785
```

[203]:
```
print('Classification Report')
print(classification_report(y_validate, predictValues_validate))
```

```
Classification Report
              precision    recall  f1-score   support

           0       0.97      0.89      0.93        38
           1       0.93      0.98      0.96        55

    accuracy                           0.95        93
   macro avg       0.95      0.94      0.94        93
weighted avg       0.95      0.95      0.95        93
```

Random Forest

[204]:
```
RFclassifier = RandomForestClassifier(n_estimators = 100, random_state =␣
↪0,min_samples_split=5,criterion='gini',max_depth=5)
RFclassifier.fit(x_train, y_train)
```

[204]:
```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=5, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=5,
                       min_weight_fraction_leaf=0.0, n_estimators=100,
                       n_jobs=None, oob_score=False, random_state=0, verbose=0,
                       warm_start=False)
```

[205]:
```
predictValues_validate = RFclassifier.predict(x_validate)
#print(predictValues_validate)
accuracy_validate=accuracy_score(y_validate, predictValues_validate)



predictValues_train = RFclassifier.predict(x_train)
#print(predictValues_train)
accuracy_train=accuracy_score(y_train, predictValues_train)


print("Train Accuracy  :: ",accuracy_train)
print("Validation Accuracy  :: ",accuracy_validate)
```

```
Train Accuracy  ::  1.0
Validation Accuracy  ::  0.978494623655914
```

[206]:
```
RFclassifier = RandomForestClassifier(n_estimators = 11, random_state =␣
↪0,min_samples_split=5,criterion='gini',max_depth=5)
RFclassifier.fit(x_train, y_train)
```

[206]:
```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=5, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=5,
                       min_weight_fraction_leaf=0.0, n_estimators=11,
                       n_jobs=None, oob_score=False, random_state=0, verbose=0,
                       warm_start=False)
```

[207]:
```
predictValues_validate = RFclassifier.predict(x_validate)
#print(predictValues_validate)
accuracy_validate=accuracy_score(y_validate, predictValues_validate)



predictValues_train = RFclassifier.predict(x_train)
#print(predictValues_train)
accuracy_train=accuracy_score(y_train, predictValues_train)


print("Train Accuracy  :: ",accuracy_train)
print("Validation Accuracy  :: ",accuracy_validate)
```

```
Train Accuracy  ::  0.9953917050691244
Validation Accuracy  ::  0.967741935483871
```

[208]:
```
print('Classification Report')
print(classification_report(y_validate, predictValues_validate))
```

```
Classification Report
              precision    recall  f1-score   support

           0       1.00      0.92      0.96        38
           1       0.95      1.00      0.97        55

    accuracy                           0.97        93
   macro avg       0.97      0.96      0.97        93
weighted avg       0.97      0.97      0.97        93
```

KNN

[121]:
```
from sklearn.neighbors import KNeighborsClassifier
from scipy.stats import zscore
```

[143]:
```
df_k
```

[143]:

	pelvic_incidence	pelvic_tilt numeric	lumbar_lordosis_angle	sacral_slope	pelvic_radius	degree_spondylolisthesis	class
0	63.027818	22.552586	39.609117	40.475232	98.672917	-0.254400	1
1	39.056951	10.060991	25.015378	28.995960	114.405425	4.564259	1
2	68.832021	22.218482	50.092194	46.613539	105.985135	-3.530317	1
3	69.297008	24.652878	44.311238	44.644130	101.868495	11.211523	1
4	49.712859	9.652075	28.317406	40.060784	108.168725	7.918501	1
..	…	…	…	…	…	…	…
305	47.903565	13.616688	36.000000	34.286877	117.449062	-4.245395	0
306	53.936748	20.721496	29.220534	33.215251	114.365845	-0.421010	0
307	61.446597	22.694968	46.170347	38.751628	125.670725	-2.707880	0
308	45.252792	8.693157	41.583126	36.559635	118.545842	0.214750	0
309	33.841641	5.073991	36.641233	28.767649	123.945244	-0.199249	0

[310 rows x 7 columns]

[144]:
```
x= df_k.drop('class',axis=1)
y= df_k['class']
```

[134]:
```
x_standardize = x.apply(zscore)
```

[135]:
```
#KNN only takes array as input hence it is importanct to convert dataframe to␣
 ↪array
x1 = np.array(x_standardize)
y1 = np.array(y)
```

[136]:
```
test_size = 0.30 # taking 70:30 training and test set
seed = 7  # Random numbmer seeding for reapeatability of the code
x_train, x_validate, y_train, y_validate = train_test_split(x1, y1,␣
 ↪test_size=test_size, random_state=seed)
```

[137]:
```
KNN = KNeighborsClassifier(n_neighbors= 8 , weights = 'uniform',␣
 ↪metric='euclidean')
KNN.fit(x_train, y_train)
```

[137]:
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='euclidean',
                     metric_params=None, n_jobs=None, n_neighbors=8, p=2,
                     weights='uniform')
```

[138]:
```
predictValues_train = KNN.predict(x_train)
print(predictValues_train)
accuracy_train=accuracy_score(y_train, predictValues_train)
print("Train Accuracy  :: ",accuracy_train)
```

```
[1 1 1 1 1 0 1 1 1 0 1 1 1 1 1 0 0 0 0 0 1 1 1 1 0 1 0 1 0 0 1 0 1 1 0 1 0 1
 0 1 0 0 1 1 0 1 1 1 1 1 0 1 1 1 0 0 1 0 1 1 0 1 1 0 1 1 1 1 1 1 0 0 0 1 1 0
 1 1 0 1 1 1 1 1 0 1 0 0 1 0 0 0 1 0 1 1 1 1 1 0 0 0 0 1 0 1 1 1 1 1 1 1 1 1
 0 1 0 1 1 1 1 1 0 1 1 1 0 1 0 0 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 0 1 1 0 1 1 0 1
 1 1 1 1 1 0 0 1 1 1 1 1 1 1 0 1 0 1 1 0 0 1 0 1 1 1 1 1 1 1 0 0 0 1 1 0 1 1
 0 0 1 1 0 1 1 1 0 1 1 1 1 1 0 0 0 0 1 1 0 1 0 1 1 1 1 1 1 0 1 1 1 1]
Train Accuracy  ::  0.8387096774193549
```

[139]:
```
predictValues_validate = KNN.predict(x_validate)
print(predictValues_validate)
accuracy_validate=accuracy_score(y_validate, predictValues_validate)
print("Validation Accuracy  :: ",accuracy_validate)
```

```
[0 1 1 1 1 0 1 1 0 1 1 1 1 1 0 1 1 1 1 1 0 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 0 0 0 1 1 1 0 1 0 1 0 1 1 1 1 1 1 1 1 1 1 0 1 1 1 0 1 0 0 1 1 0 1 1
 0 1 1 0 0 1 1 1 0 1 0 0 1 0 0 1 1 1 1]
Validation Accuracy  ::  0.8279569892473119
```

[132]:
```
df_k.drop('label', axis=1, inplace=True)
```

Agglomerative clustering

[141]:
```
df_h.sample(5)
```

[141]:
```
     pelvic_incidence  pelvic_tilt numeric  lumbar_lordosis_angle  \
152         63.404481            14.115327              48.136806
70          72.560702            17.385191              52.000000
291         51.079833            14.209935              35.951229
22          63.073611            24.413803              54.000000
59          48.109236            14.930725              35.564683

     sacral_slope  pelvic_radius  degree_spondylolisthesis     class  label
152     49.289153     111.916007                 31.784495  Abnormal      1
70      55.175511     119.193724                 32.108537  Abnormal      1
291     36.869898     115.803711                  6.905090    Normal      1
22      38.659808     106.424329                 15.779697  Abnormal      1
59      33.178512     124.056452                  7.947905  Abnormal      1
```

[145]:
```
x= df_k.drop('class',axis=1)
y= df_k['class']
```

[146]:
```
test_size = 0.30 # taking 70:30 training and test set
seed = 7  # Random numbmer seeding for reapeatability of the code
x_train, x_validate, y_train, y_validate = train_test_split(x, y,␣
 ↪test_size=test_size, random_state=seed)
```

[147]:
```
from sklearn.preprocessing import StandardScaler
independent_scalar = StandardScaler()
x_train = independent_scalar.fit_transform (x_train) #fit and transform
x_validate = independent_scalar.transform (x_validate) # only transform
```

Decision Tree Classifier

[148]:
```
from sklearn.tree import DecisionTreeClassifier
#DecisionTreeClassifier is the corresponding Classifier
Dtree = DecisionTreeClassifier(max_depth=3)
Dtree.fit (x_train, y_train)
```

[148]:
```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=3, 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='deprecated',
                       random_state=None, splitter='best')
```

[149]:
```
predictValues_train = Dtree.predict(x_train)
#print(predictValues_train)
accuracy_train=accuracy_score(y_train, predictValues_train)
```

```
predictValues_validate = Dtree.predict(x_validate)
#print(predictValues_validate)
accuracy_validate=accuracy_score(y_validate, predictValues_validate)

print("Train Accuracy  :: ",accuracy_train)
print("Validation Accuracy  :: ",accuracy_validate)
```

```
Train Accuracy  ::  0.8847926267281107
Validation Accuracy  ::  0.7741935483870968
```

[150]:
```
print('Classification Report')
print(classification_report(y_validate, predictValues_validate))
```

```
Classification Report
              precision    recall  f1-score   support

           0       0.81      0.50      0.62        34
           1       0.76      0.93      0.84        59

    accuracy                           0.77        93
   macro avg       0.79      0.72      0.73        93
weighted avg       0.78      0.77      0.76        93
```

Random Forest

[151]:
```
RFclassifier = RandomForestClassifier(n_estimators = 100, random_state =␣
↪0,min_samples_split=5,criterion='gini',max_depth=5)
RFclassifier.fit(x_train, y_train)
```

[151]:
```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=5, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=5,
                       min_weight_fraction_leaf=0.0, n_estimators=100,
                       n_jobs=None, oob_score=False, random_state=0, verbose=0,
                       warm_start=False)
```

[152]:
```
predictValues_validate = RFclassifier.predict(x_validate)
#print(predictValues_validate)
accuracy_validate=accuracy_score(y_validate, predictValues_validate)
```

```
predictValues_train = RFclassifier.predict(x_train)
#print(predictValues_train)
accuracy_train=accuracy_score(y_train, predictValues_train)


print("Train Accuracy  :: ",accuracy_train)
print("Validation Accuracy  :: ",accuracy_validate)
```

```
Train Accuracy  ::  0.967741935483871
Validation Accuracy  ::  0.8709677419354839
```

[153]:
```
RFclassifier = RandomForestClassifier(n_estimators = 11, random_state =␣
↪0,min_samples_split=5,criterion='gini',max_depth=5)
RFclassifier.fit(x_train, y_train)
```

[153]:
```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=5, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=5,
                       min_weight_fraction_leaf=0.0, n_estimators=11,
                       n_jobs=None, oob_score=False, random_state=0, verbose=0,
                       warm_start=False)
```

[154]:
```
predictValues_validate = RFclassifier.predict(x_validate)
#print(predictValues_validate)
accuracy_validate=accuracy_score(y_validate, predictValues_validate)



predictValues_train = RFclassifier.predict(x_train)
#print(predictValues_train)
accuracy_train=accuracy_score(y_train, predictValues_train)


print("Train Accuracy  :: ",accuracy_train)
print("Validation Accuracy  :: ",accuracy_validate)
```

```
Train Accuracy  ::  0.9447004608294931
Validation Accuracy  ::  0.9032258064516129
```

[155]:
```
print('Classification Report')
print(classification_report(y_validate, predictValues_validate))
```

```
Classification Report
              precision    recall  f1-score   support

           0       1.00      0.74      0.85        34
```

```
           1       0.87      1.00      0.93        59

    accuracy                           0.90        93
   macro avg       0.93      0.87      0.89        93
weighted avg       0.92      0.90      0.90        93
```

[156]:
```
from sklearn.neighbors import KNeighborsClassifier
from scipy.stats import zscore
```

[158]:
```
x= df_k.drop('class',axis=1)
y= df_k['class']
```

[159]:
```
x_standardize = x.apply(zscore)
```

[160]:
```
#KNN only takes array as input hence it is importanct to convert dataframe to␣
 ↪array
x1 = np.array(x_standardize)
y1 = np.array(y)
```

[161]:
```
test_size = 0.30 # taking 70:30 training and test set
seed = 7  # Random numbmer seeding for reapeatability of the code
x_train, x_validate, y_train, y_validate = train_test_split(x1, y1,␣
 ↪test_size=test_size, random_state=seed)
```

[162]:
```
predictValues_train = KNN.predict(x_train)
print(predictValues_train)
accuracy_train=accuracy_score(y_train, predictValues_train)
print("Train Accuracy  :: ",accuracy_train)
```

```
[1 1 1 1 1 0 1 1 1 0 1 1 1 1 0 0 0 0 0 1 1 1 1 0 1 0 1 0 0 1 0 1 1 0 1 0 1
 0 1 0 0 1 1 0 1 1 1 1 1 0 1 1 1 0 0 1 0 1 1 0 1 1 0 1 1 1 1 1 0 0 0 1 1 0
 1 1 0 1 1 1 1 1 0 1 0 0 1 0 0 0 1 0 1 1 1 1 1 0 0 0 0 1 0 1 1 1 1 1 1 1 1
 0 1 0 1 1 1 1 1 0 1 1 1 0 1 0 0 1 1 1 1 1 1 1 0 1 1 1 1 1 0 1 1 0 1 1 0 1
 1 1 1 1 0 0 1 1 1 1 1 1 1 0 1 0 1 1 0 0 1 0 1 1 1 1 1 1 1 0 0 0 1 1 0 1 1
 0 0 1 1 0 1 1 1 0 1 1 1 1 0 0 0 0 1 1 0 1 0 1 1 1 1 1 1 0 1 1 1 1]
Train Accuracy  ::  0.8387096774193549
```

[163]:
```
predictValues_validate = KNN.predict(x_validate)
print(predictValues_validate)
accuracy_validate=accuracy_score(y_validate, predictValues_validate)
print("Validation Accuracy  :: ",accuracy_validate)
```

```
[0 1 1 1 1 0 1 1 0 1 1 1 1 0 1 1 1 1 0 1 1 1 1 1 1 0 0 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 0 0 0 1 1 1 0 1 0 1 0 1 1 1 1 1 1 1 1 1 1 0 1 1 1 0 1 0 0 1 1 0 1 1
 0 1 1 0 0 1 1 1 0 1 0 0 1 0 0 1 1 1 1]
Validation Accuracy  ::  0.8279569892473119
```

[ ]: