DATA SCIENTIST

In this tutorial, I only explain you what you need to be a data scientist neither more nor less.

Data scientist need to have these skills:

1. Basic Tools: Like python, R or SQL. You do not need to know everything. What you only need is to learn how to use **python**
2. Basic Statistics: Like mean, median or standart deviation. If you know basic statistics, you can use **python** easily.
3. Data Munging: Working with messy and difficult data. Like a inconsistent date and string formatting. As you guess, **python** helps us.
4. Data Visualization: Title is actually explanatory. We will visualize the data with **python** like matplot and seaborn libraries.
5. Machine Learning: You do not need to understand math behind the machine learning technique. You only need is understanding basics of machine learning and learning how to implement it while using **python**.

As a summary we will learn python to be data scientist !!!

For parts 1, 2, 3, 4, 5 and 6, look at DATA SCIENCE TUTORIAL for BEGINNERS

https://www.kaggle.com/kanncaa1/data-sciencetutorial-for-beginners/ (https://www.kaggle.com/kanncaa1/data-sciencetutorial-for-beginners/)

In this tutorial, I am not going to learn machine learning to you, I am going to explain how to learn something by yourself.

Confucius: Give a man a fish, and you feed him for a day. Teach a man to fish, and you feed him for a lifetime

Content:

In [1]:
```
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

import warnings
# ignore warnings
warnings.filterwarnings("ignore")

# Any results you write to the current directory are saved as output.
```

In [2]:
```
# read csv (comma separated value) into data
data = pd.read_csv('column_2C_weka.csv')
print(plt.style.available) # look at available plot styles
plt.style.use('ggplot')
```

```
['fast', 'seaborn-deep', 'bmh', 'seaborn-talk', 'seaborn-pastel',
'seaborn', 'ggplot', 'classic', 'seaborn-bright', 'seaborn-poster
', 'seaborn-muted', 'seaborn-notebook', 'seaborn-white', 'fivethi
rtyeight', 'seaborn-whitegrid', 'seaborn-dark-palette', 'grayscal
e', 'seaborn-colorblind', 'Solarize_Light2', 'seaborn-darkgrid',
'_classic_test', 'tableau-colorblind10', 'seaborn-dark', 'dark_ba
ckground', 'seaborn-paper', 'seaborn-ticks']
```

8. MACHINE LEARNING (ML)

In python there are some ML libraries like sklearn, keras or tensorflow. We will use sklearn.

A. SUPERVISED LEARNING

- Supervised learning: It uses data that has labels. Example, there are orthopedic patients data that have labels *normal* and *abnormal*.
    - There are features(predictor variable) and target variable. Features are like *pelvic radius* or *sacral slope*(If you have no idea what these are like me, you can look images in google like what I did :))Target variables are labels *normal* and *abnormal*
    - Aim is that as given features(input) predict whether target variable(output) is *normal* or *abnormal*
    - Classification: target variable consists of categories like normal or abnormal
    - Regression: target variable is continious like stock market
    - If these explanations are not enough for you, just google them. However, be careful about terminology: features = predictor variable = independent variable = columns = inputs. target variable = responce variable = class = dependent variable = output = result

EXPLORATORY DATA ANALYSIS (EDA)

- In order to make something in data, as you know you need to explore data. Detailed exploratory data analysis is in my Data Science Tutorial for Beginners
- I always start with *head()* to see features that are *pelvic_incidence, pelvic_tilt numeric, lumbar_lordosis_angle, sacral_slope, pelvic_radius* and *degree_spondylolisthesis* and target variable that is *class*
- head(): default value of it shows first 5 rows(samples). If you want to see for example 100 rows just write head(100)

In [3]:
```
# to see features and target variable
data.head()
```

Out[3]:

	pelvic_incidence	pelvic_tilt numeric	lumbar_lordosis_angle	sacral_slope	pelvic_radius	degree
0	63.027818	22.552586	39.609117	40.475232	98.672917	
1	39.056951	10.060991	25.015378	28.995960	114.405425	
2	68.832021	22.218482	50.092194	46.613539	105.985135	
3	69.297008	24.652878	44.311238	44.644130	101.868495	
4	49.712859	9.652075	28.317406	40.060784	108.168725	

In [4]:
```
# Well known question is is there any NaN value and length of thi
s data so lets look at info
data.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    object 
dtypes: float64(6), object(1)
memory usage: 17.1+ KB
```

In [6]:
```
list(set(data['class']))
```

Out[6]:
```
['Normal', 'Abnormal']
```

As you can see:

- length: 310 (range index)
- Features are float
- Target variables are object that is like string

In [5]:
```
data.describe()
```

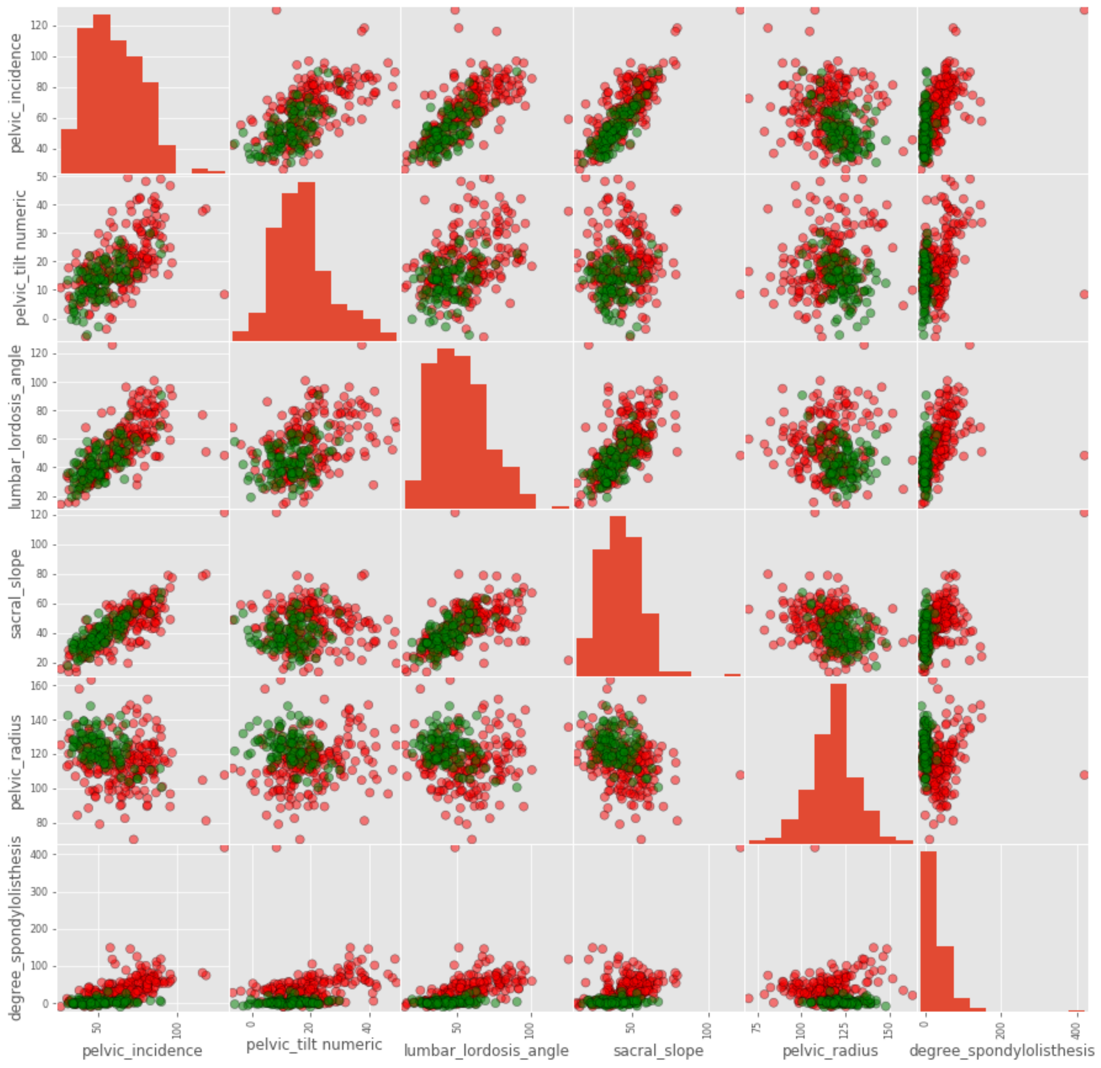
Out[5]:

	pelvic_incidence	pelvic_tilt numeric	lumbar_lordosis_angle	sacral_slope	pelvic_radius	d
count	310.000000	310.000000	310.000000	310.000000	310.000000	
mean	60.496653	17.542822	51.930930	42.953831	117.920655	
std	17.236520	10.008330	18.554064	13.423102	13.317377	
min	26.147921	-6.554948	14.000000	13.366931	70.082575	
25%	46.430294	10.667069	37.000000	33.347122	110.709196	
50%	58.691038	16.357689	49.562398	42.404912	118.268178	
75%	72.877696	22.120395	63.000000	52.695888	125.467674	
max	129.834041	49.431864	125.742385	121.429566	163.071041	

pd.plotting.scatter_matrix:

- green: *normal* and red: *abnormal*
- c: color
- figsize: figure size
- diagonal: histohram of each features
- alpha: opacity
- s: size of marker
- marker: marker type

In [9]:
```
color_list = ['red' if i=='Abnormal' else 'green' for i in data.l
oc[:,'class']]
pd.plotting.scatter_matrix(data.loc[:, data.columns != 'class'],
                                       c=color_list,
                                       figsize= [15,15],
                                       diagonal='hist',
                                       alpha=0.5,
                                       s = 200,
                                       marker = '.',
                                       edgecolor= "black")
plt.show()
```



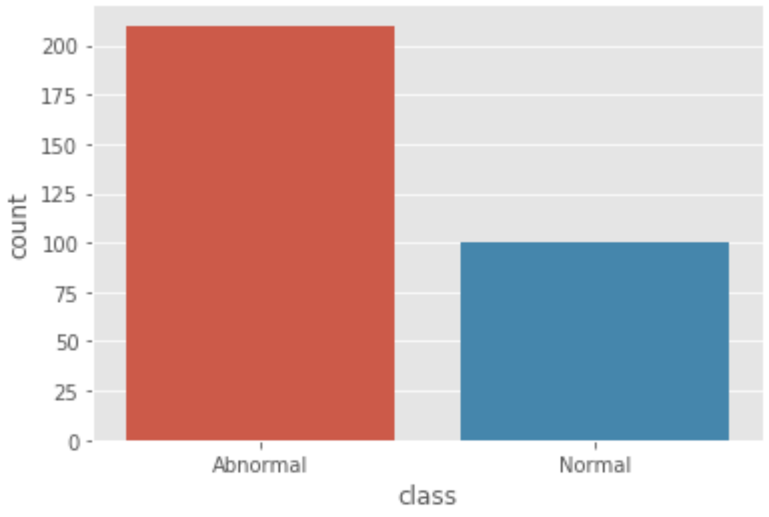
Okay, as you understand in scatter matrix there are relations between each feature but how many *normal(green)* and *abnormal(red)* classes are there.

- Searborn library has *countplot()* that counts number of classes
- Also you can print it with *value_counts()* method

This data looks like balanced. Actually there is no definiton or numeric value of balanced data but this data is balanced enough for us.
Now lets learn first classification method KNN

In [10]:
```
sns.countplot(x="class", data=data)
data.loc[:,'class'].value_counts()
```

Out[10]:
```
Abnormal    210
Normal      100
Name: class, dtype: int64
```



K-NEAREST NEIGHBORS (KNN)

- KNN: Look at the K closest labeled data points
- Classification method.
- First we need to train our data. Train = fit
- fit(): fits the data, train the data.
- predict(): predicts the data
  If you do not understand what is KNN, look at youtube there are videos like 4-5 minutes. You can understand better with it.
  Lets learn how to implement it with sklearn
- x: features
- y: target variables(normal, abnormal)
- n_neighbors: K. In this example it is 3. it means that Look at the 3 closest labeled data points

In [ ]:
```
# KNN
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors = 3)
x,y = data.loc[:,data.columns != 'class'], data.loc[:,'class']
knn.fit(x,y)
prediction = knn.predict(x)
print('Prediction: {}'.format(prediction))
```

- Well, we fit the data and predict it with KNN.
- So, do we predict correct or what is our accuracy or the accuracy is best metric to evaluate our result? Lets give answer of this questions
  Measuring model performance:
- Accuracy which is fraction of correct predictions is commonly used metric. We will use it know but there is another problem

As you see I train data with x (features) and again predict the x(features). Yes you are reading right but yes you are right again it is absurd :)

Therefore we need to split our data train and test sets.

- train: use train set by fitting
- test: make prediction on test set.
- With train and test sets, fitted data and tested data are completely different
- train_test_split(x,y,test_size = 0.3,random_state = 1)
  - x: features
  - y: target variables (normal,abnormal)
  - test_size: percentage of test size. Example test_size = 0.3, test size = 30% and train size = 70%
  - random_state: sets a seed. If this seed is same number, train_test_split() produce exact same split at each time
- fit(x_train,y_train): fit on train sets
- score(x_test,y_test)): predict and give accuracy on test sets

In [12]:
```
from sklearn.neighbors import KNeighborsClassifier
x,y = data.loc[:,data.columns != 'class'], data.loc[:,'class']

# train test split
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)
knn = KNeighborsClassifier(n_neighbors = 3)
x,y = data.loc[:,data.columns != 'class'], data.loc[:,'class']
knn.fit(x_train,y_train)
prediction = knn.predict(x_test)
#print('Prediction: {}'.format(prediction))
print('With KNN (K=3) accuracy is: ',knn.score(x_test,y_test)) # 
accuracy
```

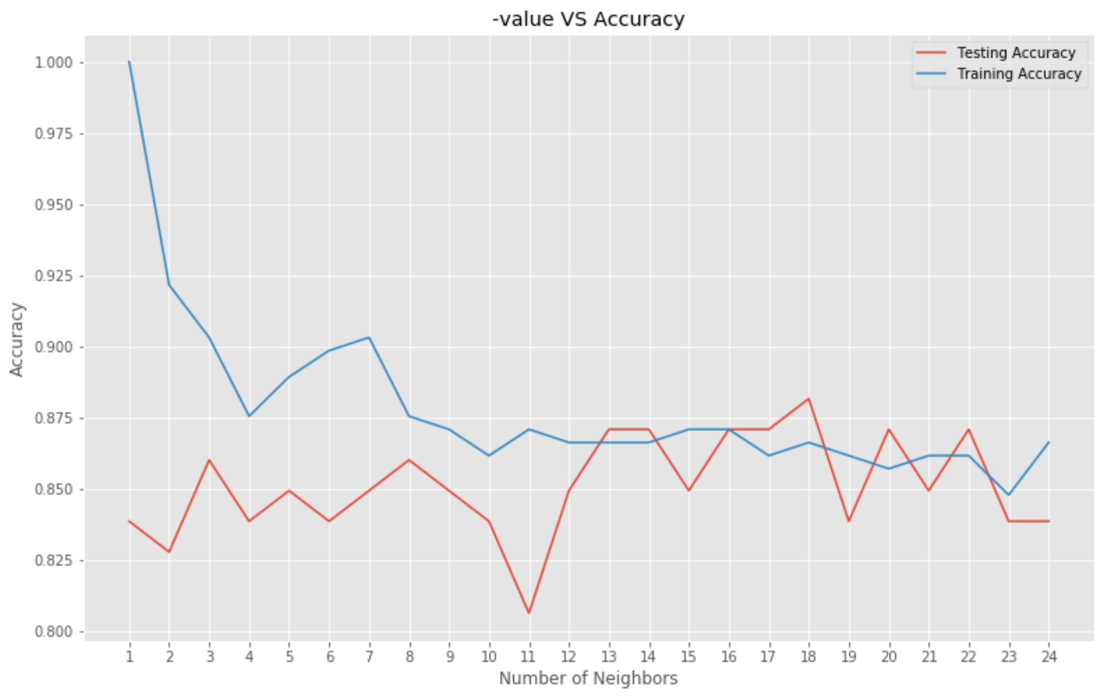
```
With KNN (K=3) accuracy is:  0.8602150537634409
```

Accuracy is 86% so is it good ? I do not know actually, we will see at the end of tutorial.
Now the question is why we choose K = 3 or what value we need to choose K. The answer is in model complexity

Model complexity:

- K has general name. It is called a hyperparameter. For now just know K is hyperparameter and we need to choose it that gives best performace.
- Literature says if k is small, model is complex model can lead to overfit. It means that model memorizes the train sets and cannot predict test set with good accuracy.
- If k is big, model that is less complex model can lead to underfit.
- At below, I range K value from 1 to 25(exclude) and find accuracy for each K value. As you can see in plot, when K is 1 it memozize train sets and cannot give good accuracy on test set (overfit). Also if K is 18, model is lead to underfit. Again accuracy is not enough. However look at when K is 18(best performance), accuracy has highest value almost 88%.

In [13]:
```
# Model complexity
neig = np.arange(1, 25)
train_accuracy = []
test_accuracy = []
# Loop over different values of k
for i, k in enumerate(neig):
    # k from 1 to 25(exclude)
    knn = KNeighborsClassifier(n_neighbors=k)
    # Fit with knn
    knn.fit(x_train,y_train)
    #train accuracy
    train_accuracy.append(knn.score(x_train, y_train))
    # test accuracy
    test_accuracy.append(knn.score(x_test, y_test))

# Plot
plt.figure(figsize=[13,8])
plt.plot(neig, test_accuracy, label = 'Testing Accuracy')
plt.plot(neig, train_accuracy, label = 'Training Accuracy')
plt.legend()
plt.title('-value VS Accuracy')
plt.xlabel('Number of Neighbors')
plt.ylabel('Accuracy')
plt.xticks(neig)
plt.savefig('graph.png')
plt.show()
print("Best accuracy is {} with K = {}".format(np.max(test_accura
cy),1+test_accuracy.index(np.max(test_accuracy))))
```

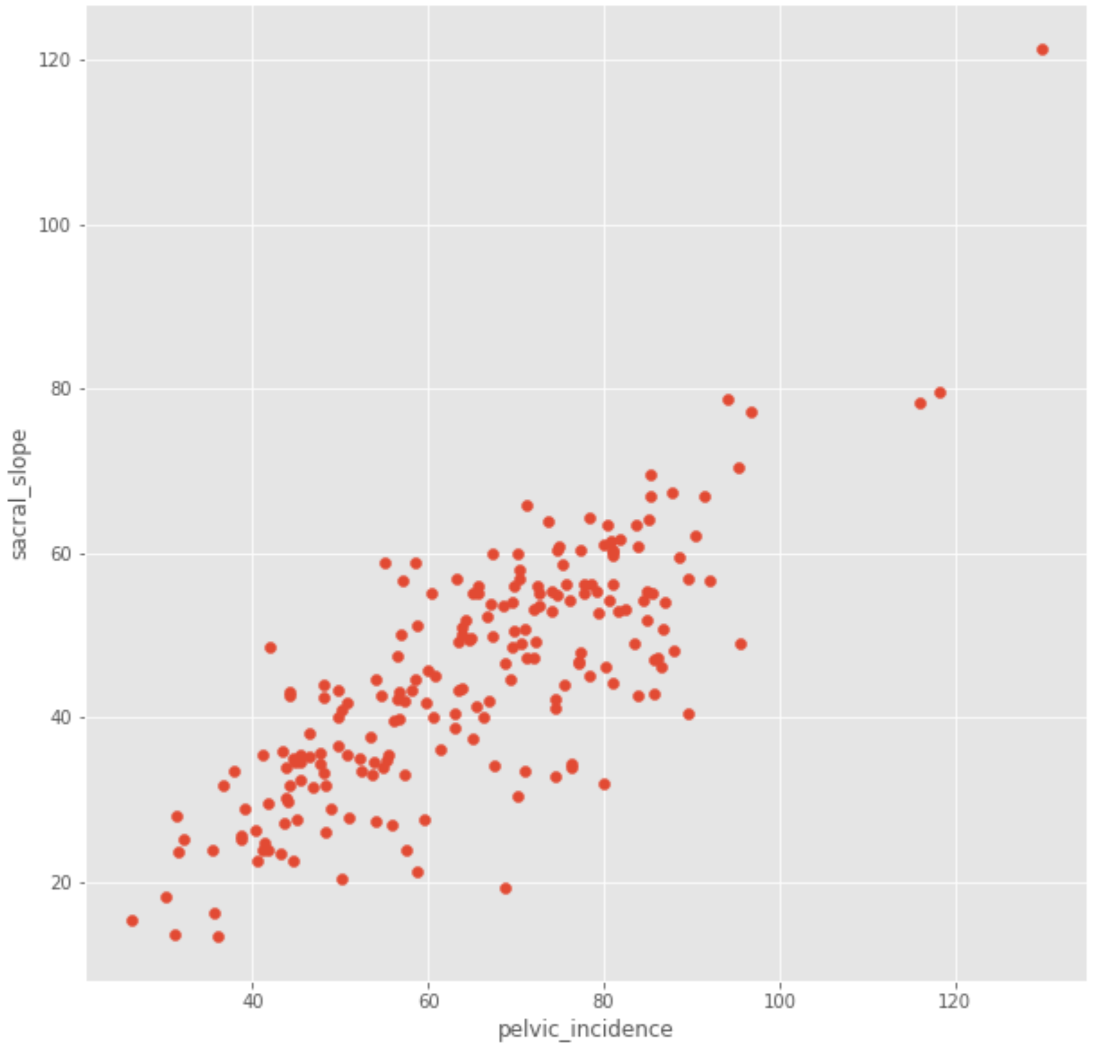


```
Best accuracy is 0.8817204301075269 with K = 18
```

REGRESSION

- Supervised learning
- We will learn linear and logistic regressions
- This orthopedic patients data is not proper for regression so I only use two features that are *sacral_slope* and *pelvic_incidence* of abnormal
  - I consider feature is pelvic_incidence and target is sacral_slope
  - Lets look at scatter plot so as to understand it better
  - reshape(-1,1): If you do not use it shape of x or y becaomes (210,) and we cannot use it in sklearn, so we use shape(-1,1) and shape of x or y be (210, 1).

In [14]:
```
# create data1 that includes pelvic_incidence that is feature and
sacral_slope that is target variable
data1 = data[data['class'] =='Abnormal']
x = np.array(data1.loc[:,'pelvic_incidence']).reshape(-1,1)
y = np.array(data1.loc[:,'sacral_slope']).reshape(-1,1)
# Scatter
plt.figure(figsize=[10,10])
plt.scatter(x=x,y=y)
plt.xlabel('pelvic_incidence')
plt.ylabel('sacral_slope')
plt.show()
```



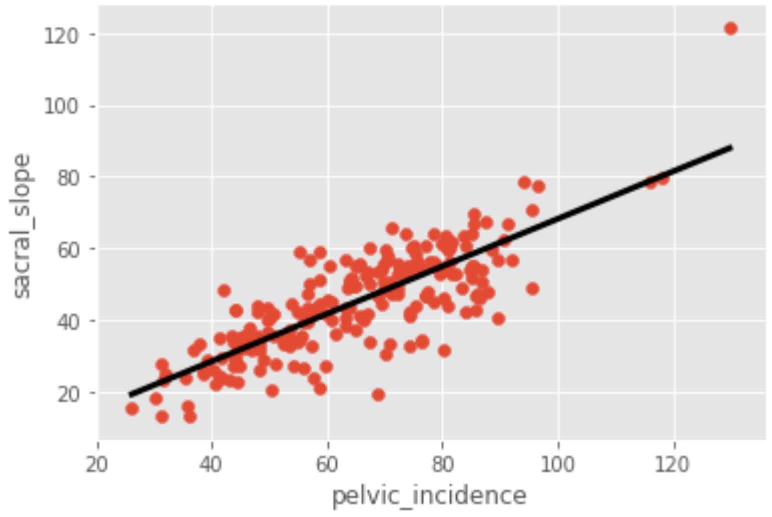
Now we have our data to make regression. In regression problems target value is continuously varying variable such as price of house or sacral_slope. Lets fit line into this points.

Linear regression

- y = ax + b where y = target, x = feature and a = parameter of model
- We choose parameter of model(a) according to minimum error function that is lost function
- In linear regression we use Ordinary Least Square (OLS) as lost function.
- OLS: sum all residuals but some positive and negative residuals can cancel each other so we sum of square of residuals. It is called OLS
- Score: Score uses R^2 method that is ((y_pred - y_mean)^2)/(y_actual - y_mean)^2

In [15]:
```python
# LinearRegression
from sklearn.linear_model import LinearRegression
reg = LinearRegression()
# Predict space
predict_space = np.linspace(min(x), max(x)).reshape(-1,1)
# Fit
reg.fit(x,y)
# Predict
predicted = reg.predict(predict_space)
# R^2
print('R^2 score: ',reg.score(x, y))
# Plot regression line and scatter
plt.plot(predict_space, predicted, color='black', linewidth=3)
plt.scatter(x=x,y=y)
plt.xlabel('pelvic_incidence')
plt.ylabel('sacral_slope')
plt.show()
```

```
R^2 score:  0.6458410481075871
```



Metrics Sklearn: [https://scikit-learn.org/stable/modules/model_evaluation.html (https://scikit-learn.org/stable/modules/model_evaluation.html)](https://scikit-learn.org/stable/modules/model_evaluation.html)

```
In [35]: from sklearn.metrics import r2_score,mean_squared_error
print("R2 Score = ",r2_score(y, reg.predict(x)))
print("MSE Score = ",mean_squared_error(y, reg.predict(x), square
d=True))
print("RMSE Score = ",mean_squared_error(y, reg.predict(x), squar
ed=False))
```

```
R2 Score =  0.6458410481075871
MSE Score =  74.26647086122357
RMSE Score =  8.617799653114684
```

CROSS VALIDATION

As you know in KNN method we use train test split with random_state that split exactly same at each time. However, if we do not use random_state, data is split differently at each time and according to split accuracy will be different. Therefore, we can conclude that model performance is dependent on train_test_split. For example you split, fit and predict data 5 times and accuracies are 0.89, 0.9, 0.91, 0.92 and 0.93, respectively. Which accuracy do you use? Do you know what accuracy will be at 6th times split, train and predict. The answer is I do not know but if I use cross validation I can find acceptable accuracy.
Cross Validation (CV)

- K folds = K fold CV.
- Look at this image it defines better than me :)
- When K is increase, computationally cost is increase
- cross_val_score(reg,x,y,cv=5): use reg(linear regression) with x and y that we define at above and K is 5. It means 5 times(split, train,predict)

In [39]:
```python
import sklearn.metrics
sorted(sklearn.metrics.SCORERS.keys())
```

Out[39]:
```
['accuracy',
 'adjusted_mutual_info_score',
 'adjusted_rand_score',
 'average_precision',
 'balanced_accuracy',
 'completeness_score',
 'explained_variance',
 'f1',
 'f1_macro',
 'f1_micro',
 'f1_samples',
 'f1_weighted',
 'fowlkes_mallows_score',
 'homogeneity_score',
 'jaccard',
 'jaccard_macro',
 'jaccard_micro',
 'jaccard_samples',
 'jaccard_weighted',
 'max_error',
 'mutual_info_score',
 'neg_brier_score',
 'neg_log_loss',
 'neg_mean_absolute_error',
 'neg_mean_gamma_deviance',
 'neg_mean_poisson_deviance',
 'neg_mean_squared_error',
 'neg_mean_squared_log_error',
 'neg_median_absolute_error',
 'neg_root_mean_squared_error',
 'normalized_mutual_info_score',
 'precision',
 'precision_macro',
 'precision_micro',
 'precision_samples',
 'precision_weighted',
 'r2',
 'recall',
 'recall_macro',
 'recall_micro',
 'recall_samples',
 'recall_weighted',
 'roc_auc',
 'roc_auc_ovo',
 'roc_auc_ovo_weighted',
 'roc_auc_ovr',
 'roc_auc_ovr_weighted',
 'v_measure_score']
```

CROSS VALIDATION PERMET D ESTIMER AU PLUS PROCHE LA PERFORMANCE D UN MODELE MAIS CE NEST PAS UNE OPTIMISATION DE SES HYPERPARAMETRES

In [48]:
```
# CV
from sklearn.model_selection import cross_val_score
reg = LinearRegression()
k = 5
cv_result = cross_val_score(reg,x,y,cv=k, scoring='neg_mean_squar
ed_error') # uses MSE as score
print('CV Scores: ',cv_result)


print("neg_mean_squared_error: %0.2f (+/- %0.2f)" % (cv_result.me
an(), cv_result.std() * 2))

print('CV scores average: ',np.sum(cv_result)/k)


"It outputs the negative of the MSE, as it always tries to maximi
ze the score."
```

```
CV Scores:  [ -41.41499622  -60.13518443 -112.94078597  -78.76705
303 -107.08695672]
neg_mean_squared_error: -80.07 (+/- 54.43)
CV scores average:  -80.06899527512078
```

Out[48]:
```
'It outputs the negative of the MSE, as it always tries to maximi
ze the score.'
```

In [50]:
```
from sklearn.pipeline import make_pipeline
from sklearn import preprocessing
reg = LinearRegression()
k = 5
clf = make_pipeline(preprocessing.StandardScaler(), reg)
cv_result = cross_val_score(clf,x,y,cv=k, scoring='neg_mean_squar
ed_error') # uses MSE as score
print('CV Scores: ',cv_result)


print("neg_mean_squared_error: %0.2f (+/- %0.2f)" % (cv_result.me
an(), cv_result.std() * 2))

print('CV scores average: ',np.sum(cv_result)/k)


"It outputs the negative of the MSE, as it always tries to maximi
ze the score."
```

```
CV Scores:  [ -41.41499622  -60.13518443 -112.94078597  -78.76705
303 -107.08695672]
neg_mean_squared_error: -80.07 (+/- 54.43)
CV scores average:  -80.06899527512083
```

Out[50]:
```
'It outputs the negative of the MSE, as it always tries to maximi
ze the score.'
```

Regularized Regression

As we learn linear regression choose parameters (coefficients) while minimizing lost function. If linear regression thinks that one of the feature is important, it gives high coefficient to this feature. However, this can cause overfitting that is like memorizing in KNN. In order to avoid overfitting, we use regularization that penalize large coefficients.

- Ridge regression: First regularization technique. Also it is called L2 regularization.
    - Ridge regression lost fuction = OLS + alpha * sum(parameter^2)
    - alpha is parameter we need to choose to fit and predict. Picking alpha is similar to picking K in KNN. As you understand alpha is hyperparameter that we need to choose for best accuracy and model complexity. This process is called hyperparameter tuning.
    - What if alpha is zero? lost function = OLS so that is linear rigression :)
    - If alpha is small that can cause overfitting
    - If alpha is big that can cause underfitting. But do not ask what is small and big. These can be change from problem to problem.
- Lasso regression: Second regularization technique. Also it is called L1 regularization.
    - Lasso regression lost fuction = OLS + alpha * sum(absolute_value(parameter))
    - It can be used to select important features od the data. Because features whose values are not shrinked to zero, is chosen by lasso regression
    - In order to choose feature, I add new features in our regression data

Linear vs Ridge vs Lasso First impression: Linear Feature Selection: 1.Lasso 2.Ridge Regression model: 1.Ridge 2.Lasso 3.Linear

In [51]:
```
# Ridge
from sklearn.linear_model import Ridge
x_train,x_test,y_train,y_test = train_test_split(x,y,random_state
= 2, test_size = 0.3)
ridge = Ridge(alpha = 0.1, normalize = True)
ridge.fit(x_train,y_train)
ridge_predict = ridge.predict(x_test)
print('Ridge score: ',ridge.score(x_test,y_test))
```

```
Ridge score:  0.5608287918841997
```

In [62]:
```
from sklearn.pipeline import make_pipeline
from sklearn import preprocessing
from sklearn.linear_model import Ridge

data1 = data[data['class'] =='Abnormal']
x = np.array(data1.loc[:,'pelvic_incidence']).reshape(-1,1)
y = np.array(data1.loc[:,'sacral_slope']).reshape(-1,1)


ridge = Ridge(alpha = 0.1, normalize = True)
k = 5
clf = make_pipeline(preprocessing.StandardScaler(), ridge)
cv_result = cross_val_score(clf,x,y,cv=k, scoring='r2') # uses MS
E as score
print('CV Scores: ',cv_result)
# scoring='neg_mean_squared_error'

print("R2 score: %0.2f (+/- %0.2f)" % (cv_result.mean(), cv_resul
t.std() * 2))

### THE DEFAULT SCORING IS scoring='r2'
```

```
CV Scores:  [0.13349995 0.6099293  0.50238422 0.23509033 0.311251
]
R2 score: 0.36 (+/- 0.35)
```

In [70]:
```
# Ridge
from sklearn.linear_model import Ridge

data1 = data[data['class'] =='Abnormal']
x = np.array(data1.loc[:,'pelvic_incidence']).reshape(-1,1)
x = np.array(data1.loc[:,['pelvic_incidence','pelvic_tilt numeric
','lumbar_lordosis_angle','pelvic_radius']])
y = np.array(data1.loc[:,'sacral_slope']).reshape(-1,1)

x_train,x_test,y_train,y_test = train_test_split(x,y,random_state
= 2, test_size = 0.3)
ridge = Ridge(alpha = 0.1, normalize = True)
ridge.fit(x_train,y_train)
ridge_predict = ridge.predict(x_test)
print('Ridge score: ',ridge.score(x_test,y_test))
print('Ridge coefficients: ',ridge.coef_)
```

```
Ridge score:  0.9114728424477411
Ridge coefficients:  [[ 0.69936866 -0.66867475  0.13768552 -0.083
45027]]
```

In [67]:
```
# Lasso
from sklearn.linear_model import Lasso
x = np.array(data1.loc[:,['pelvic_incidence','pelvic_tilt numeric
','lumbar_lordosis_angle','pelvic_radius']])
y = np.array(data1.loc[:,'sacral_slope']).reshape(-1,1)

lasso = Lasso(alpha = 0.1, normalize = True)
k = 5
clf = make_pipeline(preprocessing.StandardScaler(), lasso)
cv_result = cross_val_score(clf,x,y,cv=k, scoring='r2') # uses MS
E as score
print('CV Scores: ',cv_result)
# scoring='neg_mean_squared_error'

print("R2 score: %0.2f (+/- %0.2f)" % (cv_result.mean(), cv_resul
t.std() * 2))

### THE DEFAULT SCORING IS scoring='r2'

#print('Lasso coefficients: ',lasso.coef_)
```

```
CV Scores:  [0.8263005  0.96102355 0.92970193 0.93730646 0.932706
79]
R2 score: 0.92 (+/- 0.09)
```

In [68]:
```
# Lasso
from sklearn.linear_model import Lasso
x = np.array(data1.loc[:,['pelvic_incidence','pelvic_tilt numeric
','lumbar_lordosis_angle','pelvic_radius']])
y = np.array(data1.loc[:,'sacral_slope']).reshape(-1,1)

x_train,x_test,y_train,y_test = train_test_split(x,y,test_size =
0.3,random_state = 1)
lasso = Lasso(alpha = 0.1, normalize = True)
k = 5
lasso.fit(x_train,y_train)
lasso_predict = lasso.predict(x_test)
print('Lasso score: ',lasso.score(x_test,y_test))
print('Lasso coefficients: ',lasso.coef_)
```

```
Lasso score:  0.9591648736911897
Lasso coefficients:  [ 0.83807916 -0.71789736  0.         -0.
]
```

As you can see *pelvic_incidence* and *pelvic_tilt numeric* are important features but others are not important

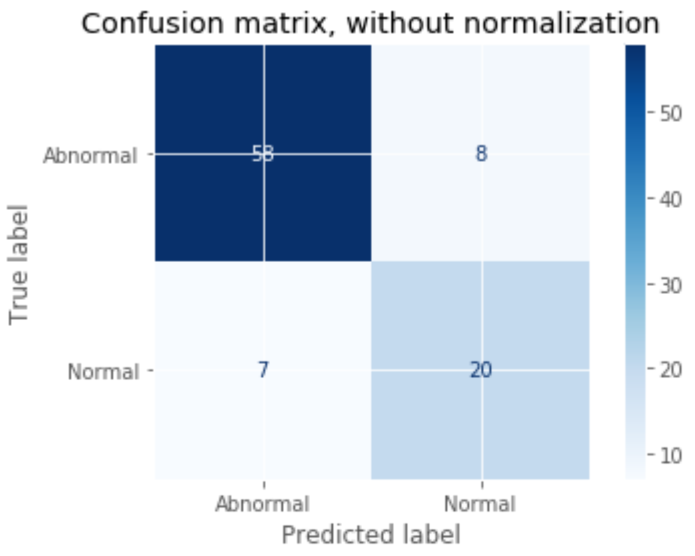
Now lets discuss accuracy. Is it enough for measurement of model selection. For example, there is a data that includes 95% normal and 5% abnormal samples and our model uses accuracy for measurement metric. Then our model predict 100% normal for all samples and accuracy is 95% but it classify all abnormal samples wrong. Therefore we need to use confusion matrix as a model measurement matris in imbalance data.
While using confusion matrix lets use Random forest classifier to diversify classification methods.

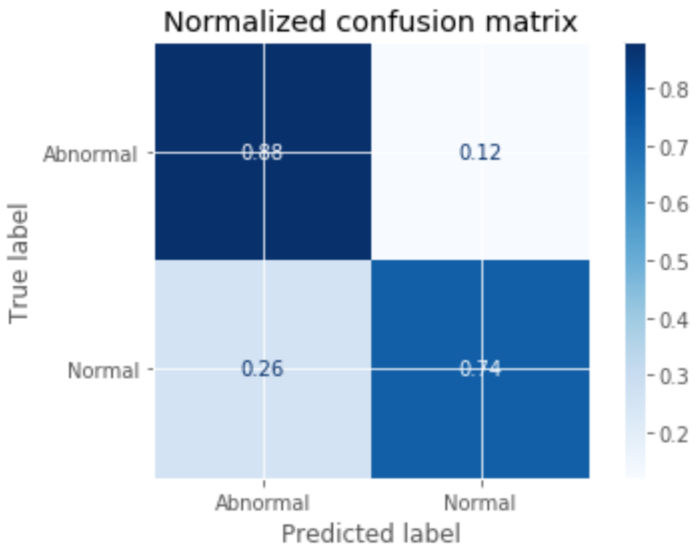
- tp = true positive(20), fp = false positive(7), fn = false negative(8), tn = true negative(58)
- tp = Prediction is positive(normal) and actual is positive(normal).
- fp = Prediction is positive(normal) and actual is negative(abnormal).
- fn = Prediction is negative(abnormal) and actual is positive(normal).
- tn = Prediction is negative(abnormal) and actual is negative(abnormal)
- precision = tp / (tp+fp)
- recall = tp / (tp+fn)
- f1 = 2 *precision* recall / (precision + recall)

CONFUSION MATRIX with Random Forest

In [207]:
```
# Confusion matrix with random forest
from sklearn.metrics import classification_report, confusion_matr
ix
from sklearn.ensemble import RandomForestClassifier
import numpy as np
import matplotlib.pyplot as plt

from sklearn import svm, datasets
from sklearn.model_selection import train_test_split
from sklearn.metrics import plot_confusion_matrix

if 'class_binary' in data.columns:
    print("ok")
x,y = data.loc[:,data.columns != 'class'], data.loc[:,'class']

if 'class_binary' in data.columns:
    x,y = data[data.columns.difference(['class','class_binary
'])], data.loc[:,'class']


x_train,x_test,y_train,y_test = train_test_split(x,y,test_size =
0.3,random_state = 1)

rf = RandomForestClassifier(random_state = 4)
rf.fit(x_train,y_train)
y_pred = rf.predict(x_test)
cm = confusion_matrix(y_test,y_pred)


np.set_printoptions(precision=2)

# Plot non-normalized confusion matrix
titles_options = [("Confusion matrix, without normalization", Non
e),
                  ("Normalized confusion matrix", 'true')]
for title, normalize in titles_options:
    disp = plot_confusion_matrix(rf, x_test, y_test,
                                 display_labels=rf.classes_,
                                 cmap=plt.cm.Blues,
                                 normalize=normalize)
    disp.ax_.set_title(title)

    print(title)
    print(disp.confusion_matrix)

plt.show()

print('Classification report: \n',classification_report(y_test,y_
pred))
```

```
ok
Confusion matrix, without normalization
[[58  8]
 [ 7 20]]
Normalized confusion matrix
[[0.88 0.12]
 [0.26 0.74]]
```



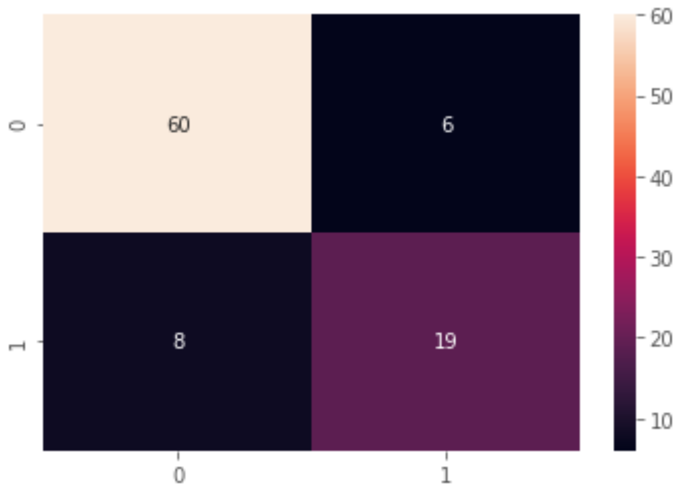


```
Classification report:
              precision    recall  f1-score   support

    Abnormal       0.89      0.88      0.89        66
      Normal       0.71      0.74      0.73        27

    accuracy                           0.84        93
   macro avg       0.80      0.81      0.81        93
weighted avg       0.84      0.84      0.84        93
```

In [80]:
```
rf.classes_
```

Out[80]:
```
array(['Abnormal', 'Normal'], dtype=object)
```

In [87]:
```
# visualize with seaborn library
sns.heatmap(cm,annot=True,fmt="d")
plt.show()
```



In [208]:
```
y_pred
```

Out[208]:
```
array(['Abnormal', 'Abnormal', 'Abnormal', 'Abnormal', 'Abnormal
',
       'Abnormal', 'Normal', 'Abnormal', 'Abnormal', 'Normal', 'A
bnormal',
       'Normal', 'Abnormal', 'Normal', 'Normal', 'Abnormal', 'Abn
ormal',
       'Normal', 'Normal', 'Abnormal', 'Abnormal', 'Normal', 'Nor
mal',
       'Abnormal', 'Abnormal', 'Abnormal', 'Abnormal', 'Abnormal
',
       'Normal', 'Abnormal', 'Abnormal', 'Normal', 'Normal', 'Abn
ormal',
       'Abnormal', 'Abnormal', 'Abnormal', 'Abnormal', 'Abnormal
',
       'Abnormal', 'Normal', 'Normal', 'Abnormal', 'Normal', 'Abn
ormal',
       'Abnormal', 'Normal', 'Abnormal', 'Abnormal', 'Abnormal',
       'Abnormal', 'Normal', 'Abnormal', 'Abnormal', 'Abnormal',
       'Abnormal', 'Abnormal', 'Abnormal', 'Normal', 'Abnormal',
'Normal',
       'Abnormal', 'Abnormal', 'Normal', 'Abnormal', 'Abnormal',
       'Abnormal', 'Abnormal', 'Abnormal', 'Abnormal', 'Abnormal
',
       'Abnormal', 'Abnormal', 'Normal', 'Normal', 'Abnormal', 'A
bnormal',
       'Abnormal', 'Abnormal', 'Normal', 'Normal', 'Abnormal', 'A
bnormal',
       'Normal', 'Abnormal', 'Abnormal', 'Abnormal', 'Abnormal',
'Normal',
       'Normal', 'Abnormal', 'Normal', 'Abnormal'], dtype=object)
```

ROC Curve with Logistic Regression

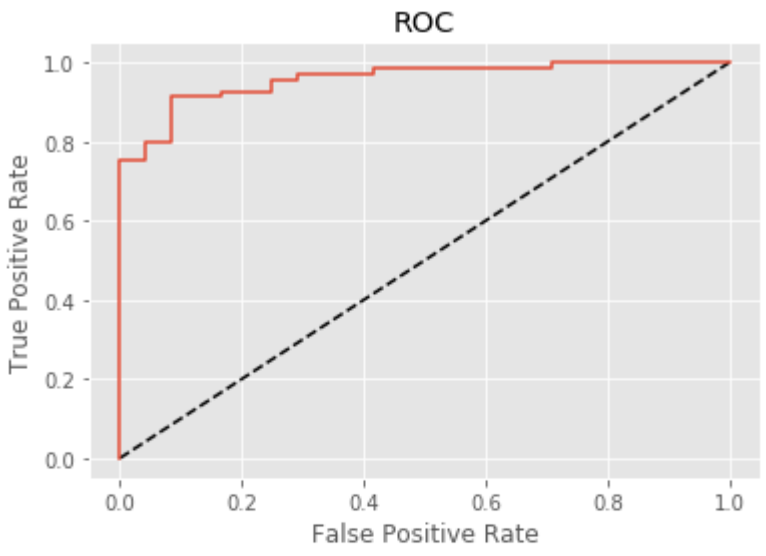
- logistic regression output is probabilities
- If probability is higher than 0.5 data is labeled 1(abnormal) else 0(normal)
- By default logistic regression threshold is 0.5
- ROC is receiver operationg characteristic. In this curve x axis is false positive rate and y axis is true positive rate
- If the curve in plot is closer to left-top corner, test is more accurate.
- Roc curve score is auc that is computation area under the curve from prediction scores
- We want auc to closer 1
- fpr = False Positive Rate
- tpr = True Positive Rate
- If you want, I made ROC, Random forest and K fold CV in this tutorial. https://www.kaggle.com/kanncaa1/roc-curve-with-k-fold-cv/ (https://www.kaggle.com/kanncaa1/roc-curve-with-k-fold-cv/)

In [135]:
```
data['class'][data['class'] == 'Abnormal'].count()
```

Out[135]:
```
pelvic_incidence            210
pelvic_tilt numeric         210
lumbar_lordosis_angle       210
sacral_slope                210
pelvic_radius               210
degree_spondylolisthesis    210
class                       210
class_binary                210
dtype: int64
```

In [209]:
```
# ROC Curve with logistic regression
from sklearn.metrics import roc_curve
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, classification_repo
rt

# abnormal = 1 and normal = 0
data['class_binary'] = data.loc[:,'class'].map({'Abnormal':1, 'No
rmal':0})
### ON MET 1 = ANORMAL car on veut détecter ANORMALITE  même si l
'échantillon ANORMAL est plus présent que NORMAL
nombre_abnormal = data['class'][data['class'] == 'Abnormal'].coun
t()
nombre_total = data['class'].count()

print("Proportion de Abnormal: {}%".format(int(nombre_abnormal*10
0/nombre_total)))


x = data.loc[:,(data.columns != 'class') & (data.columns != 'clas
s_binary')]
y = data.loc[:,'class_binary']

x_train, x_test, y_train, y_test = train_test_split(x, y, test_si
ze = 0.3, random_state=42)
logreg = LogisticRegression()
logreg.fit(x_train,y_train)

y_pred_prob = logreg.predict_proba(x_test)[:,1]
y_score = logreg.decision_function(x_test) ### Plus le chiffre es
t négatif plus la proba est 0, plus le chiffre est gros plus la p
roba est proche de 1. Si c'est proche de 0 c'est indécision
fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob)

# Plot ROC curve
plt.plot([0, 1], [0, 1], 'k--')
plt.plot(fpr, tpr)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC')
plt.show()
```

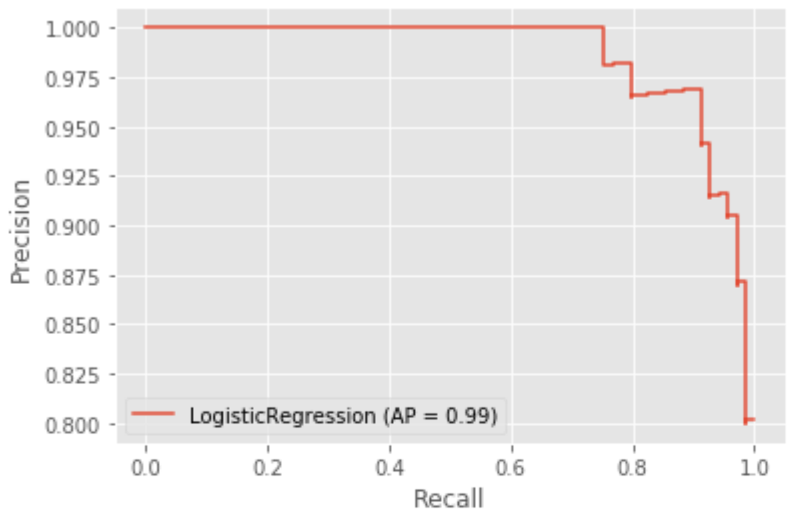
```
Proportion de Abnormal: 67%
```



In [158]:
```python
len(y_pred_prob)
```

Out[158]:
```
93
```

In [210]:
```python
from sklearn.metrics import precision_recall_curve, precision_rec
all_fscore_support, plot_precision_recall_curve

precision, recall, thresholds = precision_recall_curve(y_test, y_
pred_prob)
disp = plot_precision_recall_curve(logreg, x_test, y_test)
```



Meilleur Threshold pour optimiser F1 score dans classification logistique

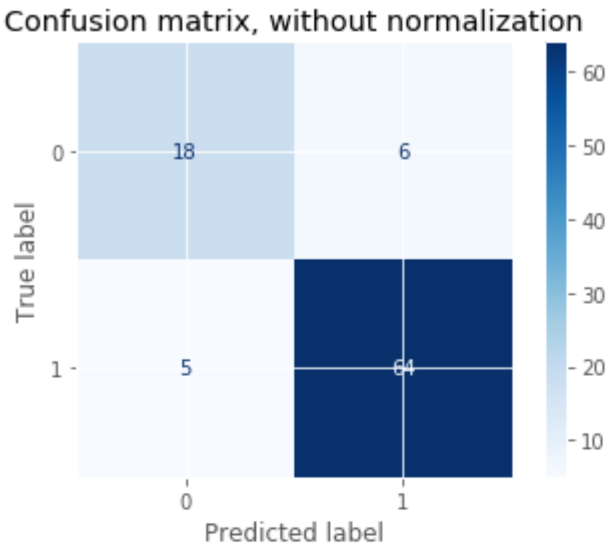
In [211]:

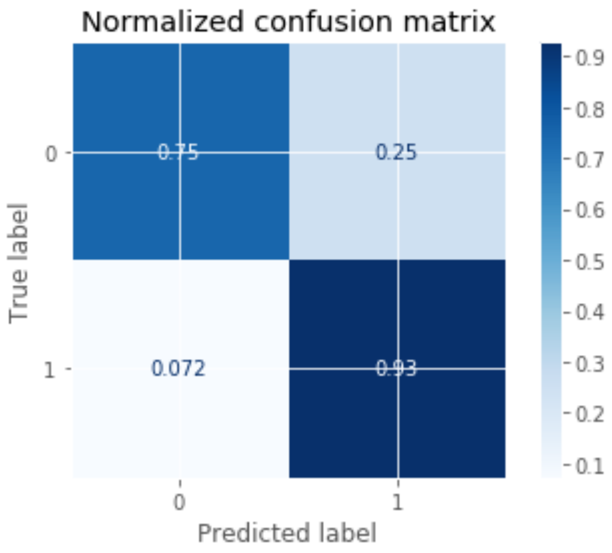
```
from sklearn.metrics import classification_report

def f1_score(precision, recall):
    return 2*(precision*recall)/(precision+recall)

pr = [(precision[i], recall[i]) for i in range(len(precision))]
f1_scores = [f1_score(element[0], element[1]) for element in pr]
best_index = np.argmax(f1_scores)
best_threshold = thresholds[best_index]
print("Best Threshold =  ",best_threshold)
print("Reaching a F1 score = ",np.max(f1_scores))

y_pred = [1 if prob>=best_threshold else 0 for prob in y_pred_pro
b]  ### prédictions finales
print(classification_report(y_test, y_pred))
```

```
Best Threshold =   0.5808941709146563
Reaching a F1 score =  0.9402985074626865
              precision    recall  f1-score   support

           0       0.79      0.92      0.85        24
           1       0.97      0.91      0.94        69

    accuracy                           0.91        93
   macro avg       0.88      0.91      0.89        93
weighted avg       0.92      0.91      0.92        93
```

In [212]:
```
# Plot non-normalized confusion matrix
titles_options = [("Confusion matrix, without normalization", None),
                  ("Normalized confusion matrix", 'true')]
for title, normalize in titles_options:
    disp = plot_confusion_matrix(logreg, x_test, y_test,
                                 display_labels=logreg.classes_,
                                 cmap=plt.cm.Blues,
                                 normalize=normalize)
    disp.ax_.set_title(title)

    print(title)
    print(disp.confusion_matrix)

plt.show()

print('Classification report: \n',classification_report(y_test,y_pred))
```

```
Confusion matrix, without normalization
[[18  6]
 [ 5 64]]
Normalized confusion matrix
[[0.75 0.25]
 [0.07 0.93]]
```





```
Classification report:
               precision    recall  f1-score   support

           0       0.79      0.92      0.85        24
           1       0.97      0.91      0.94        69

    accuracy                           0.91        93
   macro avg       0.88      0.91      0.89        93
weighted avg       0.92      0.91      0.92        93
```

In [ ]:
```
"For the smaller threshold, 0.09, if we predict that all proba >=
0.09 is abnormal, we have 80% precision and 100% recall"
```

HYPERPARAMETER TUNING

As I mention at KNN there are hyperparameters that are need to be tuned

- For example:
  - k at KNN
  - alpha at Ridge and Lasso
  - Random forest parameters like max_depth
  - linear regression parameters(coefficients)
- Hyperparameter tuning:
  - try all of combinations of different parameters
  - fit all of them
  - measure prediction performance
  - see how well each performs
  - finally choose best hyperparameters
- This process is most difficult part of this tutorial. Because we will write a lot of for loops to iterate all combinations. Just I am kidding sorry for this :) (We actually did it at KNN part)
- We only need is one line code that is GridSearchCV
  - grid: K is from 1 to 50(exclude)
  - GridSearchCV takes knn and grid and makes grid search. It means combination of all hyperparameters. Here it is k.

In [214]:
```
# grid search cross validation with 1 hyperparameter
from sklearn.model_selection import GridSearchCV
grid = {'n_neighbors': np.arange(1,50)}

knn = KNeighborsClassifier()
knn_cv = GridSearchCV(knn, grid, cv=3) # GridSearchCV
knn_cv.fit(x,y)# Fit


# Print hyperparameter
print("Tuned hyperparameter k: {}".format(knn_cv.best_params_))
print("Best score: {}".format(knn_cv.best_score_))
```

```
Tuned hyperparameter k: {'n_neighbors': 4}
Best score: 0.7559434901667911
```

Other grid search example with 2 hyperparameter

- First hyperparameter is C:logistic regression regularization parameter
  - If C is high: overfit
  - If C is low: underfit
- Second hyperparameter is penalty(lost function): l1 (Lasso) or l2(Ridge) as we learnt at linear regression part.

In [217]:

```
# grid search cross validation with 2 hyperparameter
# 1. hyperparameter is C:logistic regression regularization param
eter
# 2. penalty l1 or l2
# Hyperparameter grid

param_grid = {'C': np.logspace(-3, 3, 7), 'penalty': ['l1', 'l2
']}

x_train, x_test, y_train, y_test = train_test_split(x,y,test_size
= 0.3,random_state = 12)

logreg = LogisticRegression()
logreg_cv = GridSearchCV(logreg,param_grid,cv=10)
logreg_cv.fit(x_train,y_train)

# Print the optimal parameters and best score
print("Tuned hyperparameters : {}".format(logreg_cv.best_params
_))
print("Best Accuracy: {}".format(logreg_cv.best_score_))
```

```
Tuned hyperparameters : {'C': 0.01, 'penalty': 'l2'}
Best Accuracy: 0.8525974025974026
```

PRE-PROCESSING DATA

- In real life data can include objects or categorical data in order to use them in sklearn we need to encode them into numerical data
- In data, class is *abnormal* and *normal*. Lets convert them into numeric value (actually I did it in logistic regression part with different method)
- 2 different feature is created with the name *class_Abnormal* and *class_Normal*
- However we need to drop one of the column because they are duplicated

In [219]:
```
# Load data
data = pd.read_csv('column_2C_weka.csv')
# get_dummies
print(data.head())

df = pd.get_dummies(data, drop_first = True)
df.head(10)
```

```
   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
```

Out[219]:

	pelvic_incidence	pelvic_tilt numeric	lumbar_lordosis_angle	sacral_slope	pelvic_radius	degree
0	63.027818	22.552586	39.609117	40.475232	98.672917	
1	39.056951	10.060991	25.015378	28.995960	114.405425	
2	68.832021	22.218482	50.092194	46.613539	105.985135	
3	69.297008	24.652878	44.311238	44.644130	101.868495	
4	49.712859	9.652075	28.317406	40.060784	108.168725	
5	40.250200	13.921907	25.124950	26.328293	130.327871	
6	53.432928	15.864336	37.165934	37.568592	120.567523	
7	45.366754	10.755611	29.038349	34.611142	117.270068	
8	43.790190	13.533753	42.690814	30.256437	125.002893	
9	36.686353	5.010884	41.948751	31.675469	84.241415	

In [ ]:
```
# drop one of the feature
df.drop("class_Normal",axis = 1, inplace = True)
df.head(10)
# instead of two steps we can make it with one step pd.get_dummie
s(data,drop_first = True)
```

Other preprocessing step is centering, scaling or normalizing

- If you listen my advice and watch KNN in youtube, you have noticed that KNN uses form of distance for classificaiton like some oher methods. Therefore, we need to scale data. For this reason, we use
  - standardization: (x - x.mean) / x.variance or x - x.min / x.range
- pipeline: The purpose of the pipeline is to assemble several steps like svm(classifier) and standardization(pre-processing)
- How we create parameters name: for example SVM_ _C : stepName__parameterName
- Then grid search to find best parameters

Explanation here: [https://www.quora.com/What-are-C-and-gamma-with-regards-to-a-support-vector-machine (https://www.quora.com/What-are-C-and-gamma-with-regards-to-a-support-vector-machine)](https://www.quora.com/What-are-C-and-gamma-with-regards-to-a-support-vector-machine)

In [231]:
```
# SVM, pre-process and pipeline
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
steps = [('scalar', StandardScaler()),
         ('SVM', SVC())]
pipeline = Pipeline(steps)
parameters = {'SVM__C':[1, 10, 100],
              'SVM__gamma':[0.1, 0.01]}
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size
=0.2,random_state = 1)
cv = GridSearchCV(pipeline,param_grid=parameters,cv=3)
cv.fit(x_train,y_train)

y_pred = cv.predict(x_test)

print("Accuracy: {}".format(cv.score(x_test, y_test)))
print("Tuned Model Parameters: {}".format(cv.best_params_))
```

```
Accuracy: 0.8548387096774194
Tuned Model Parameters: {'SVM__C': 100, 'SVM__gamma': 0.01}
```

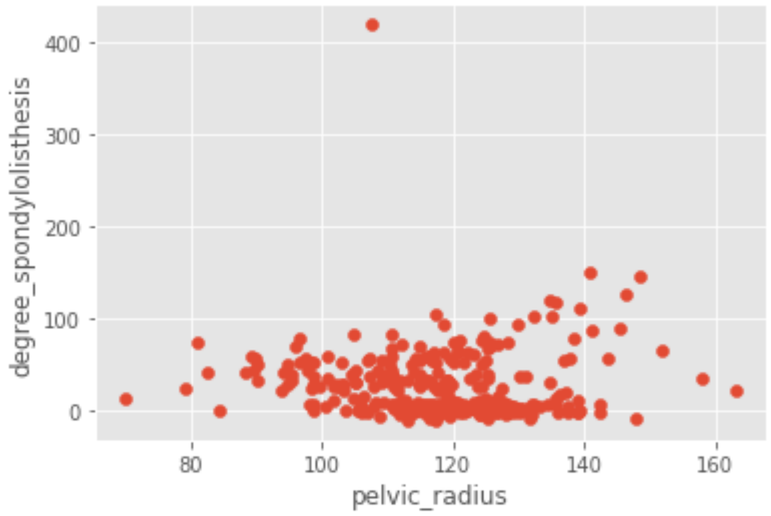
UNSUPERVISED LEARNING

- Unsupervised learning: It uses data that has unlabeled and uncover hidden patterns from unlabeled data. Example, there are orthopedic patients data that do not have labels. You do not know which orthopedic patient is normal or abnormal.
- As you know orthopedic patients data is labeled (supervised) data. It has target variables. In order to work on unsupervised learning, lets drop target variables and to visualize just consider *pelvic_radius* and *degree_spondylolisthesis*

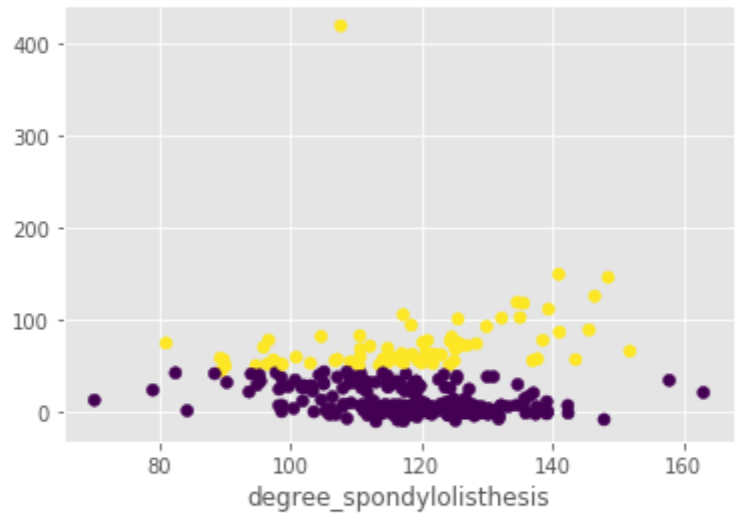
KMEANS

- Lets try our first unsupervised method that is KMeans Cluster
- KMeans Cluster: The algorithm works iteratively to assign each data point to one of K groups based on the features that are provided. Data points are clustered based on feature similarity
- KMeans(n_clusters = 2): n_clusters = 2 means that create 2 cluster

In [232]:
```
# As you can see there is no labels in data
data = pd.read_csv('column_2C_weka.csv')
plt.scatter(data['pelvic_radius'],data['degree_spondylolisthesis
'])
plt.xlabel('pelvic_radius')
plt.ylabel('degree_spondylolisthesis')
plt.show()
```



In [233]:
```
# KMeans Clustering
data2 = data.loc[:,['degree_spondylolisthesis','pelvic_radius']]
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters = 2)
kmeans.fit(data2)
labels = kmeans.predict(data2)
plt.scatter(data['pelvic_radius'],data['degree_spondylolisthesis
'],c = labels)
plt.xlabel('pelvic_radius')
plt.xlabel('degree_spondylolisthesis')
plt.show()
```



EVALUATING OF CLUSTERING

We cluster data in two groups. Okey well is that correct clustering? In order to evaluate clustering we will use cross tabulation table.

- There are two clusters that are *0* and *1*
- First class *0* includes 138 abnormal and 100 normal patients
- Second class *1* includes 72 abnormal and 0 normal patiens *The majority of two clusters are abnormal patients.

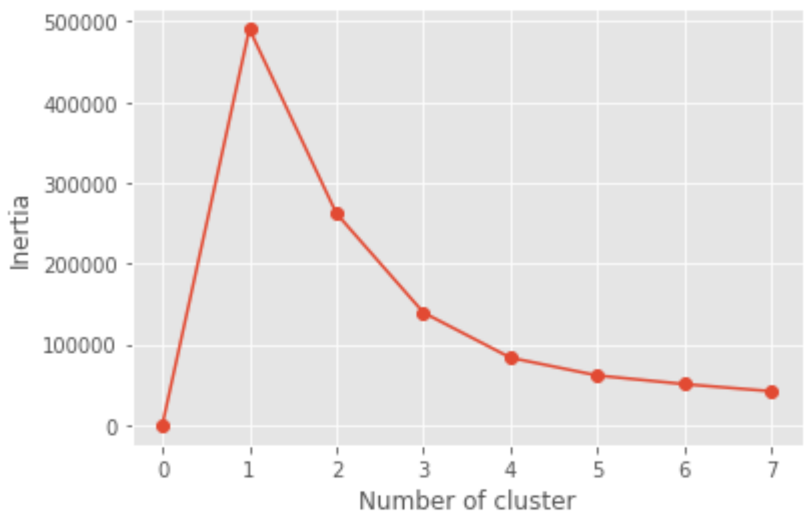
In [234]:
```
# cross tabulation table
df = pd.DataFrame({'labels':labels,"class":data['class']})
ct = pd.crosstab(df['labels'],df['class'])
print(ct)
```

```
class   Abnormal  Normal
labels
0            138     100
1             72       0
```

The new question is that we know how many class data includes, but what if number of class is unknow in data. This is kind of like hyperparameter in KNN or regressions.

- inertia: how spread out the clusters are distance from each sample
- lower inertia means more clusters
- What is the best number of clusters ? *There are low inertia and not too many cluster trade off so we can choose elbow

In [235]:
```
# inertia
inertia_list = np.empty(8)
for i in range(1,8):
    kmeans = KMeans(n_clusters=i)
    kmeans.fit(data2)
    inertia_list[i] = kmeans.inertia_
plt.plot(range(0,8),inertia_list,'-o')
plt.xlabel('Number of cluster')
plt.ylabel('Inertia')
plt.show()
```



STANDARDIZATION

- Standardizaton is important for both supervised and unsupervised learning
- Do not forget standardization as pre-processing
- As we already have visualized data so you got the idea. Now we can use all features for clustering.
- We can use pipeline like supervised learning.

In [236]:
```
data = pd.read_csv('column_2C_weka.csv')
data3 = data.drop('class',axis = 1)
```

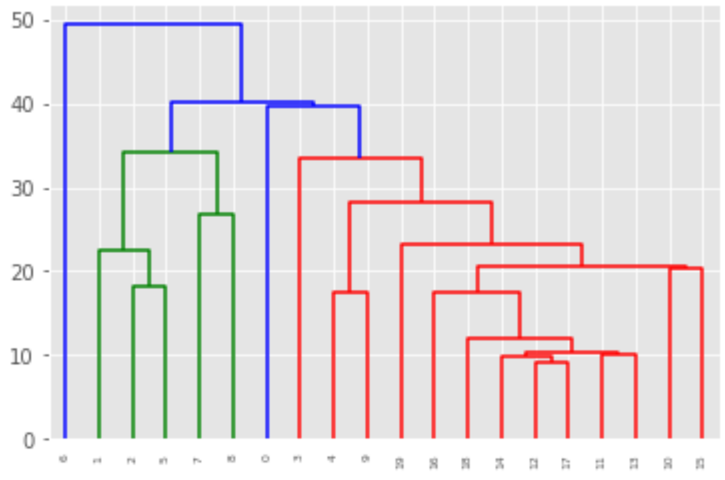
In [237]:
```python
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline
scalar = StandardScaler()
kmeans = KMeans(n_clusters = 2)
pipe = make_pipeline(scalar,kmeans)
pipe.fit(data3)
labels = pipe.predict(data3)
df = pd.DataFrame({'labels':labels,"class":data['class']})
ct = pd.crosstab(df['labels'],df['class'])
print(ct)
```

```
class   Abnormal  Normal
labels
0             94      90
1            116      10
```

HIERARCHY

- vertical lines are clusters
- height on dendogram: distance between merging cluster
- method= 'single' : closest points of clusters

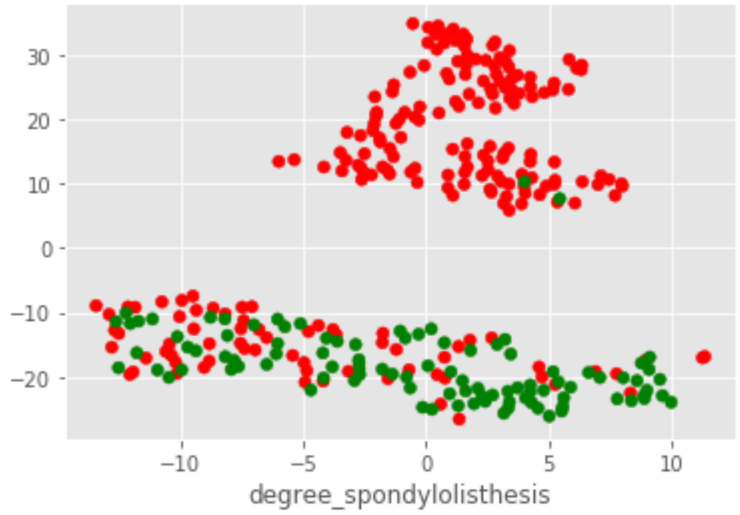
In [238]:
```python
from scipy.cluster.hierarchy import linkage,dendrogram

merg = linkage(data3.iloc[200:220,:],method = 'single')
dendrogram(merg, leaf_rotation = 90, leaf_font_size = 6)
plt.show()
```



T - Distributed Stochastic Neighbor Embedding (T - SNE)

- learning rate: 50-200 in normal
- fit_transform: it is both fit and transform. t-sne has only have fit_transform
- Varieties have same position relative to one another

In [239]:
```
from sklearn.manifold import TSNE
model = TSNE(learning_rate=100)
transformed = model.fit_transform(data2)
x = transformed[:,0]
y = transformed[:,1]
plt.scatter(x,y,c = color_list )
plt.xlabel('pelvic_radius')
plt.xlabel('degree_spondylolisthesis')
plt.show()
```



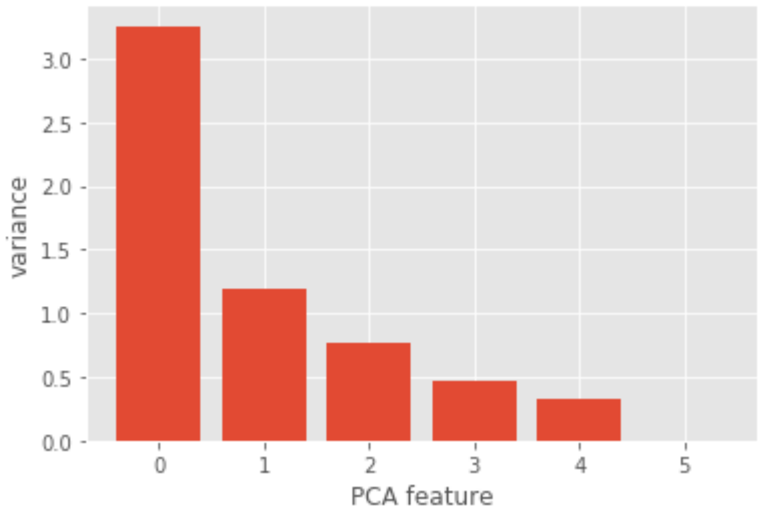
PRINCIPLE COMPONENT ANALYSIS (PCA)

- Fundemental dimension reduction technique
- first step is decorrelation:
  - rotates data samples to be aligned with axes
  - shifts data asmples so they have mean zero
  - no information lost
  - fit() : learn how to shift samples
  - transform(): apply the learned transformation. It can also be applies test data
- Resulting PCA features are not linearly correlated
- Principle components: directions of variance

In [240]:
```
# PCA
from sklearn.decomposition import PCA
model = PCA()
model.fit(data3)
transformed = model.transform(data3)
print('Principle components: ',model.components_)
```

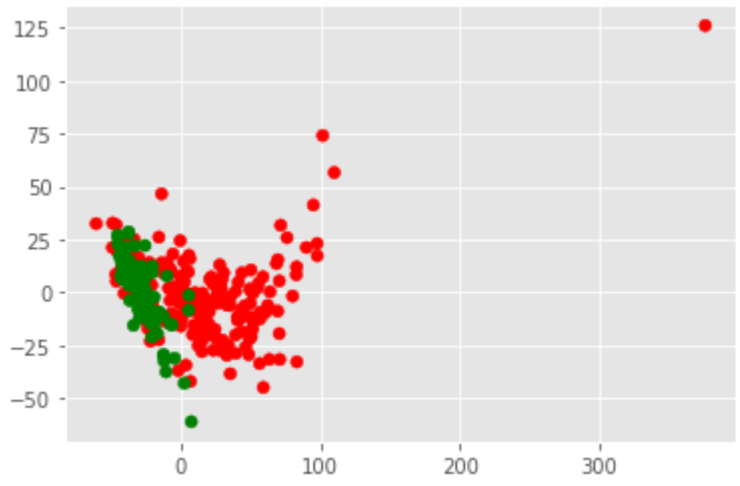
```
Principle components:  [[ 3.24e-01  1.13e-01  3.04e-01  2.10e-01
-3.00e-02  8.63e-01]
 [-4.77e-01 -9.86e-02 -5.33e-01 -3.78e-01  3.22e-01  4.82e-01]
 [-1.54e-03 -2.65e-01 -4.97e-01  2.63e-01 -7.75e-01  1.19e-01]
 [ 3.74e-01  7.54e-01 -3.39e-01 -3.80e-01 -1.75e-01 -3.29e-02]
 [-4.42e-01  7.35e-02  5.12e-01 -5.15e-01 -5.15e-01  8.36e-02]
 [ 5.77e-01 -5.77e-01 -1.09e-11 -5.77e-01 -3.59e-12  3.07e-12]]
```

In [241]:
```
# PCA variance
scaler = StandardScaler()
pca = PCA()
pipeline = make_pipeline(scaler,pca)
pipeline.fit(data3)

plt.bar(range(pca.n_components_), pca.explained_variance_)
plt.xlabel('PCA feature')
plt.ylabel('variance')
plt.show()
```



- Second step: intrinsic dimension: number of feature needed to approximate the data essential idea behind dimension reduction
- PCA identifies intrinsic dimension when samples have any number of features
- intrinsic dimension = number of PCA feature with significant variance
- In order to choose intrinsic dimension try all of them and find best accuracy
- Also check intuitive way of PCA with this example: https://www.kaggle.com/kanncaa1/tutorial-pca-intuition-and-image-completion (https://www.kaggle.com/kanncaa1/tutorial-pca-intuition-and-image-completion)

In [242]:
```
# apply PCA
pca = PCA(n_components = 2)
pca.fit(data3)
transformed = pca.transform(data3)
x = transformed[:,0]
y = transformed[:,1]
plt.scatter(x,y,c = color_list)
plt.show()
```



CONCLUSION

This is the end of DATA SCIENCE tutorial. The first part is here: https://www.kaggle.com/kanncaa1/data-sciencetutorial-for-beginners/ (https://www.kaggle.com/kanncaa1/data-sciencetutorial-for-beginners/)
If you have any question or suggest, I will be happy to hear it.