```
{
  "pelvic_incidence": 33.03,
  "pelvic_tilt": 12.55,
  "lumbar_lordosis_angle": 49.61,
  "sacral_slope":  60.48,
  "pelvic_radius": 88.67,
  "grade_of_spondylolisthesis": -0.75
}
```

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn import metrics
from sklearn.preprocessing import normalize,scale
from sklearn.decomposition import PCA

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from sklearn.decomposition import PCA as sklearnPCA
from pandas.plotting import parallel_coordinates

def knn(report, sample_path='./vertebral_column_data/column_3C.dat'):
    #importing dataset and converting to datasframe
    header = ['pelvic_incidence', 'pelvic_tilt', 'lumbar_lordosis_angle', 'sacral_slope', 'pel
vic_radius', 'grade_of_spondylolisthesis', 'label']
    df = pd.read_csv(sample_path, sep=' ', header=None, names=header)

    features = df.iloc[:,0:6]
    labels = df.iloc[:,6]
    model = KNeighborsClassifier(n_neighbors=5, weights='distance')
    model.fit(features,labels)

    predicted = model.predict(report)
    predicted_proba = model.predict_proba(report)

    return(predicted, predicted_proba)
```

```
import smtplib
from configparser import ConfigParser
from email.mime.text import MIMEText

def sendmail(to, message):
    config = ConfigParser()
    config.read('credentials.ini')

    gmail_user = config.get('email', 'username')
    gmail_password = config.get('email', 'pwd')
    sent_from = gmail_user

    try:
        msg = MIMEText(message)
        msg['Subject'] = 'Medical Report for Vertebral Column Test'
        msg['From'] = gmail_user
        msg['To'] = to

        server = smtplib.SMTP_SSL('smtp.gmail.com', 465)
        server.ehlo()
        server.login(gmail_user, gmail_password)
        server.sendmail(sent_from, to, msg.as_string())
        server.close()

        print('Email sent!')
    except:
        print('Unable to process mail...')
```

```
import json

class Patient:

    def __init__(self, cpf, name, address, telephone, email):
        self.cpf = cpf
        self.name = name
        self.address = address
        self.telephone = telephone
        self.email = email

    @classmethod
    def from_input(cls):
        return cls(
            input('CPF: '),
            input('NAME: '),
            input('Address: '),
            input('Telephone: '),
            input('Email: ')
            )


class VertebralColumnReport:
    def __init__(self, patient, report):
        self.id = patient
        self.pelvic_incidence = report['pelvic_incidence']
        self.pelvic_tilt = report['pelvic_tilt']
        self.lumbar_lordosis_angle = report['lumbar_lordosis_angle']
        self.sacral_slope = report['sacral_slope']
        self.pelvic_radius = report['pelvic_radius']
        self.grade_of_spondylolisthesis = report['grade_of_spondylolisthesis']

    def tolist(self):
        return [self.pelvic_incidence,
            self.pelvic_tilt,
            self.lumbar_lordosis_angle,
            self.sacral_slope,
            self.pelvic_radius,
            self.grade_of_spondylolisthesis]

    def todict(self):
        return {
            'pelvic_incidence' : self.pelvic_incidence,
            'pelvic_tilt' : self.pelvic_tilt,
            'lumbar_lordosis_angle' : self.lumbar_lordosis_angle,
            'sacral_slope' : self.sacral_slope,
            'pelvic_radius' : self.pelvic_radius,
            'grade_of_spondylolisthesis' : self.grade_of_spondylolisthesis
        }
```

```
from patient import Patient, VertebralColumnReport
from FuzzyKnn import knn
from headers import *
from mail import sendmail

label = {'DH' : 'Disk Hernia', 'NO' : 'Normal', 'SL' : 'Spondilolysthesis'}

def printdetails(user, predicted, probability):
    print()
    print('DETAILED REPORT FOR PATIENT {}'.format(user.name))
    print('Diagnosed: {}'.format(label[predicted]))
    accuracy = probability[predicted]
    print('Accuracy: {}%'.format(accuracy))
    print()

    if predicted != 'NO' and accuracy > 75:
        action = 'Booked Follow-up Appointment\nReference for Speciality Hospital also forward
ed.\n'
        print('-'*60)
        # call Appointment Scheduler Module
        print('Follow Up Appointment Scheduled with priority : HIGH')
        print('-'*60)
        # call reference builder
        print('Forwarded report for reference with Speciality Hospital')
        print('-'*60)

    elif (predicted != 'NO' and accuracy > 50):
        action = 'Booked Follow-up Appointment'
        # call Appointment Scheduler Module
        print('-'*60)
        print('Follow Up Appointment Scheduled with priority : MODERATE')
        print('-'*60)

    else :
        action = 'Booked Follow-up Appointment'
        print('-'*60)
        # call Appointment Scheduler Module
        print('Follow Up Appointment Scheduled with priority : LOW')
        print('-'*60)

    msg = 'Diagnosed: {}\nAccuracy: {}\n{}'.format(label[predicted], accuracy, action)
    # print(msg, user.email)
    sendmail(user.email, msg)

def main():
    user = Patient.from_input()
    report_path = input('Report Path: ')

    with open(report_path) as file:
        report = json.load(file)

    vcr = VertebralColumnReport(user, report)
    data = pd.DataFrame(vcr.todict(), index=[0])

    predicted, predicted_proba = knn(data)

    probability = {'DH' : predicted_proba[0][0]*100, 'NO' : predicted_proba[0][1]*100, 'SL' :
predicted_proba[0][2]*100}

    printdetails(user, predicted[0], probability)

if __name__ == '__main__':
    main()
```