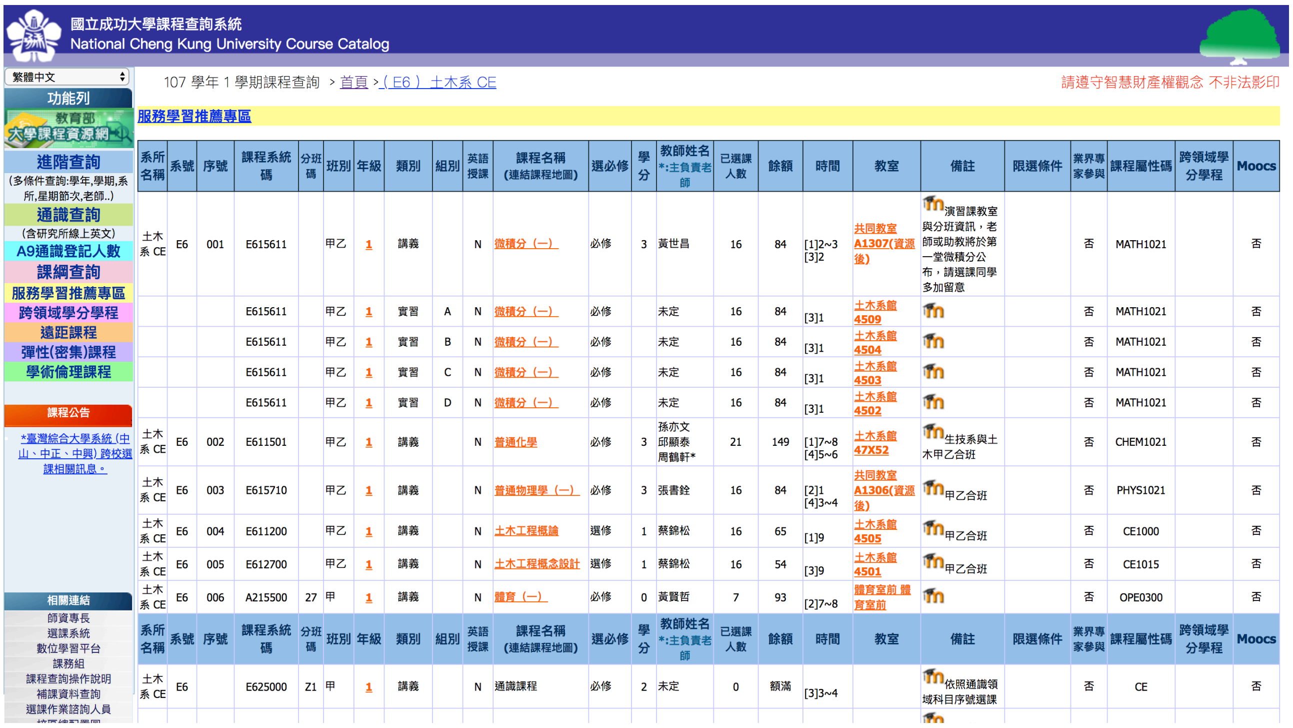
成功大學大學部一到三年級各系必修學分與通識課程和體育課程的撞課分析

我是成功大學土木系110級林友鈞。本文將分析107學年上學期大一到大三（因大三以上許多科系有分組，不做統計）各系平均必修學分與通識課及體育課撞課數之間的關係。並統計通識課及體育課分佈的時間。

分析範圍：成功大學大一至大三各系必修學分，通識課及體育課上課時間。
資料來源：成功大學課程查詢系統。
使用語言：Python 3.6
使用套件：

requests-html==0.9.0
matplotlib==2.2.2
scipy==1.1.0
numpy==1.14.3
pandas==0.23.0



國立成功大學課程查詢系統
National Cheng Kung University Course Catalog

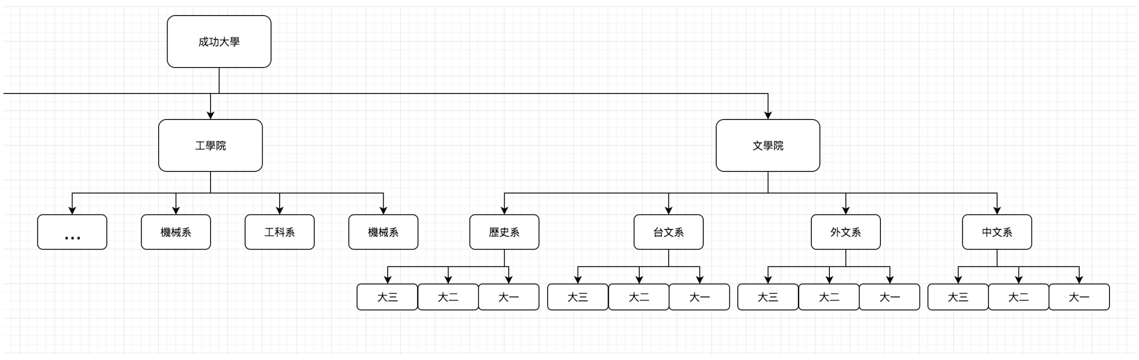
107 學年 1 學期課程查詢 > 首頁 > (E6) 土木系 CE

請遵守智慧財產權觀念 不非法影印

服務學習推薦專區

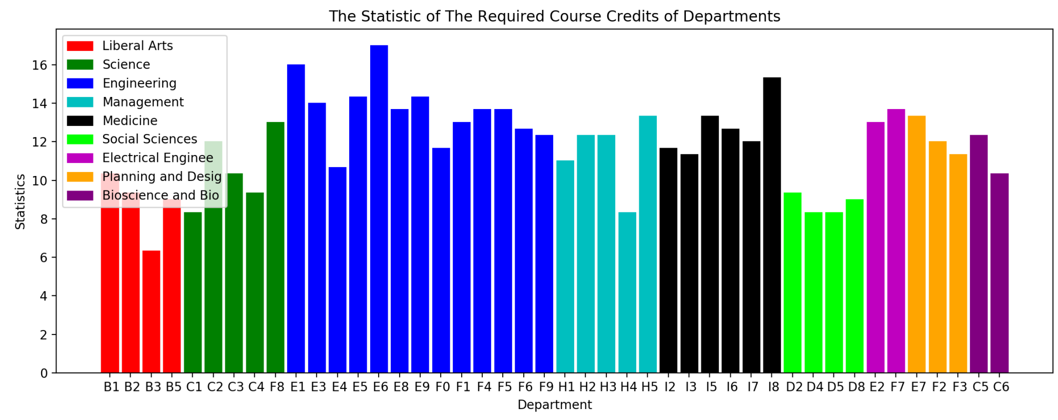
系所名稱	系號	序號	課程系統碼	分班碼	班別	年級	類別	組別	英語授課	課程名稱 (連結課程地圖)	選必修	學分	教師姓名 *:主負責老師	已選課人數	餘額	時間	教室	備註	限選條件	業界專家參與	課程屬性碼	跨領域學分學程	Moocs
土木系 CE	E6	001	E615611		甲乙	1	講義		N	微積分（一）	必修	3	黃世昌	16	84	[1]2~3 [3]2	共同教室 A1307(資源後)	演習課教室與分班資訊，老師或助教將於第一堂微積分公布，請選課同學多加留意		否	MATH1021		否
			E615611		甲乙	1	實習	A	N	微積分（一）	必修		未定	16	84	[3]1	土木系館 4509			否	MATH1021		否
			E615611		甲乙	1	實習	B	N	微積分（一）	必修		未定	16	84	[3]1	土木系館 4504			否	MATH1021		否
			E615611		甲乙	1	實習	C	N	微積分（一）	必修		未定	16	84	[3]1	土木系館 4503			否	MATH1021		否
			E615611		甲乙	1	實習	D	N	微積分（一）	必修		未定	16	84	[3]1	土木系館 4502			否	MATH1021		否
土木系 CE	E6	002	E611501		甲乙	1	講義		N	普通化學	必修	3	孫亦文 邱顯泰 周鶴軒*	21	149	[1]7~8 [4]5~6	土木系館 47X52	生技系與土木甲乙合班		否	CHEM1021		否
土木系 CE	E6	003	E615710		甲乙	1	講義		N	普通物理學（一）	必修	3	張書銓	16	84	[2]1 [4]3~4	共同教室 A1306(資源後)	甲乙合班		否	PHYS1021		否
土木系 CE	E6	004	E611200		甲乙	1	講義		N	土木工程概論	選修	1	蔡錦松	16	65	[1]9	土木系館 4505	甲乙合班		否	CE1000		否
土木系 CE	E6	005	E612700		甲乙	1	講義		N	土木工程概念設計	選修	1	蔡錦松	16	54	[3]9	土木系館 4501	甲乙合班		否	CE1015		否
土木系 CE	E6	006	A215500	27	甲	1	講義		N	體育（一）	必修	0	黃賢哲	7	93	[2]7~8	體育室前 體育室前			否	OPE0300		否
系所名稱	系號	序號	課程系統碼	分班碼	班別	年級	類別	組別	英語授課	課程名稱 (連結課程地圖)	選必修	學分	教師姓名 *:主負責老師	已選課人數	餘額	時間	教室	備註	限選條件	業界專家參與	課程屬性碼	跨領域學分學程	Moocs
土木系 CE	E6		E625000	Z1	甲	1	講義		N	通識課程	必修	2	未定	0	額滿	[3]3~4		依照通識領域科目序號選課		否	CE		否

首先，蒐集各系必修必選課程資料，再計算各系平均學分。資料結構示意圖如圖（一）。（程式碼：附件一）

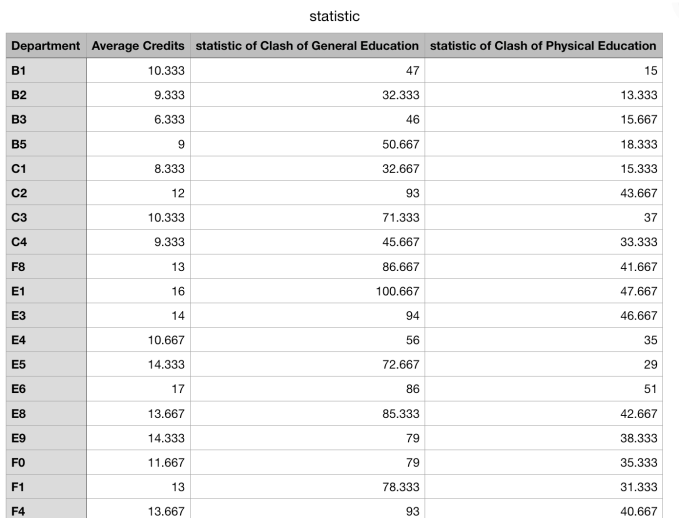


圖一

視覺化分析後（圖一）。X座標為個系所代碼。Y座標為107年上學期大一至大三平均個系所開必修學分數。顏色的區分為不同學院。結果顯示E6土木系平均必修學分最高。（程式碼：附件二）

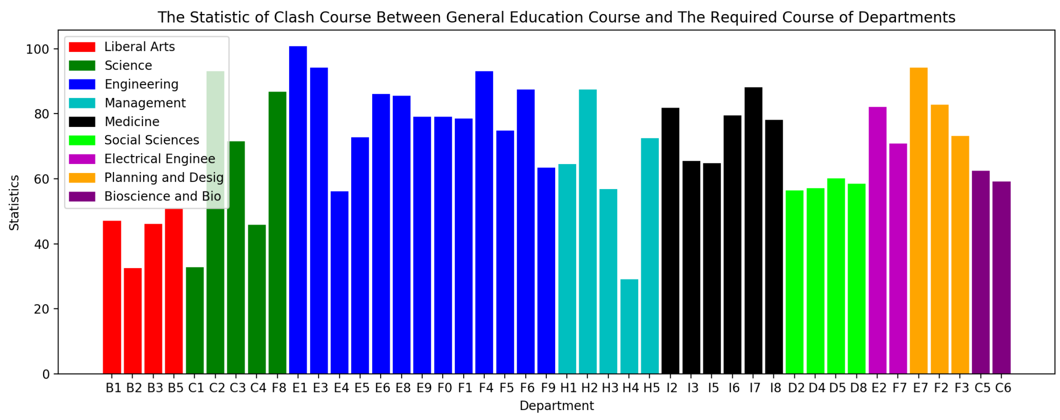


第二步，蒐集接著蒐集通識課及體育課課程資料（程式碼:附件三）。並統計撞課數（程式碼:附件四）
。部分資料如下表所示。（程式碼:附件五）


statistic

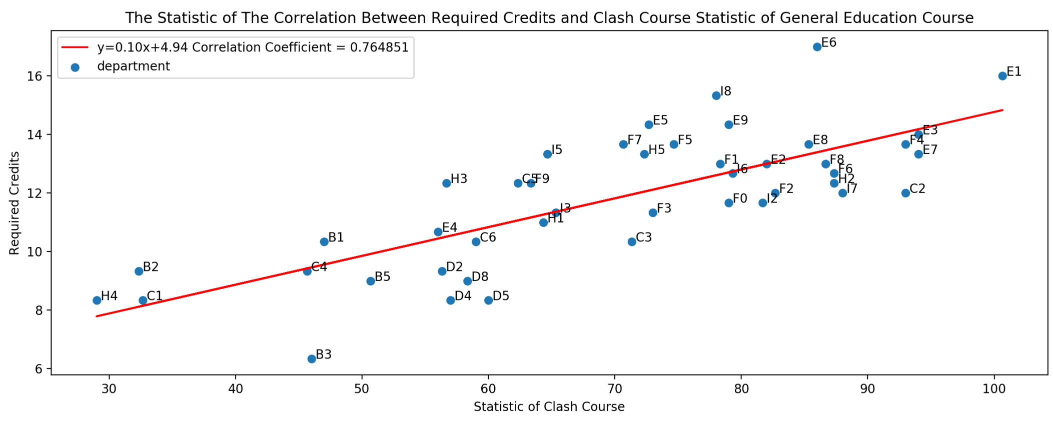
Department	Average Credits	statistic of Clash of General Education	statistic of Clash of Physical Education
B1	10.333	47	15
B2	9.333	32.333	13.333
B3	6.333	46	15.667
B5	9	50.667	18.333
C1	8.333	32.667	15.333
C2	12	93	43.667
C3	10.333	71.333	37
C4	9.333	45.667	33.333
F8	13	86.667	41.667
E1	16	100.667	47.667
E3	14	94	46.667
E4	10.667	56	35
E5	14.333	72.667	29
E6	17	86	51
E8	13.667	85.333	42.667
E9	14.333	79	38.333
F0	11.667	79	35.333
F1	13	78.333	31.333
F4	13.667	93	40.667

各系通識課撞課統計視覺化如下圖



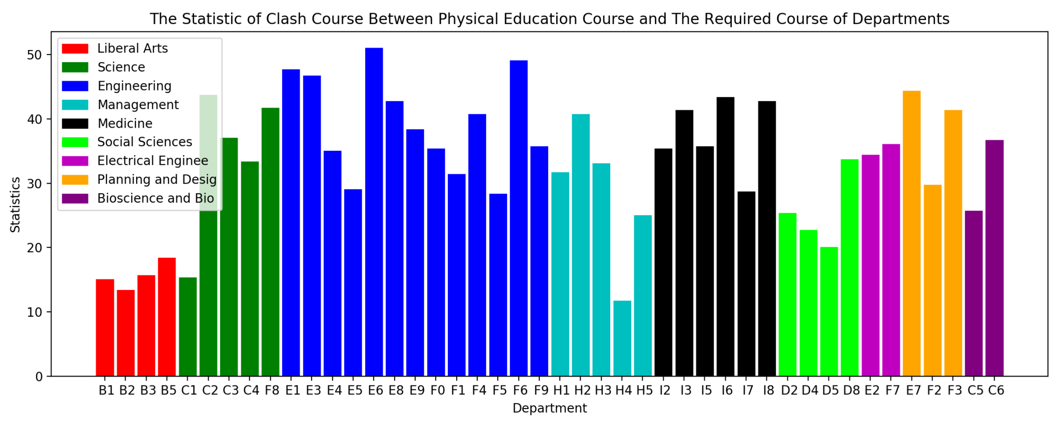
程式碼：附件六

若將通識撞課統計數據與各系必修平均學分做圖如下：

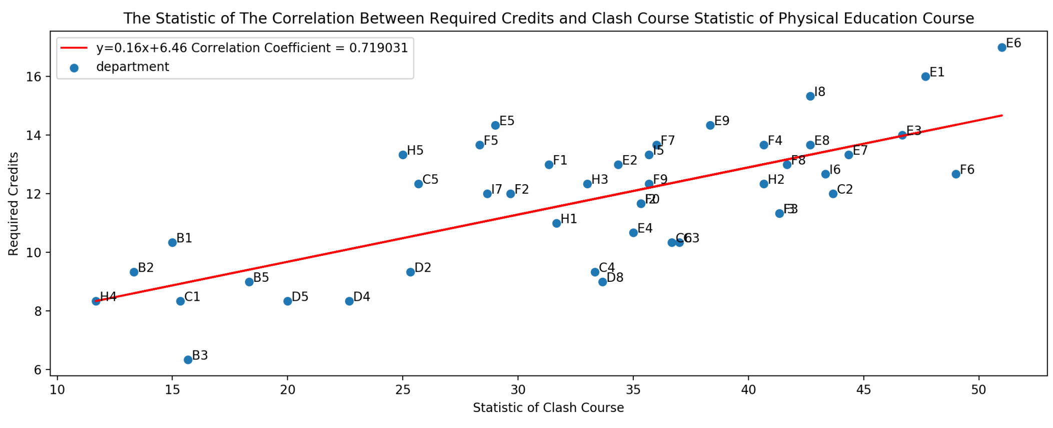


由圖左上所列相關係數0.76可知，必修學分與通識課撞課之間的關係為**高度正相關**。
程式碼：附件七

各系體育課撞課統計如下

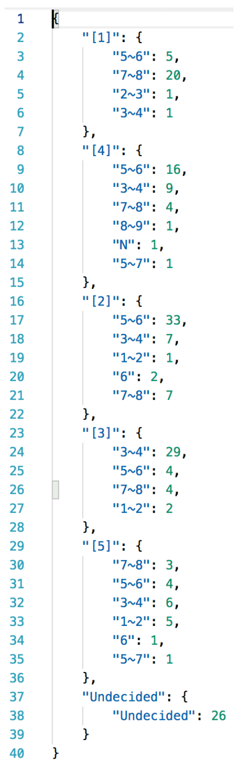


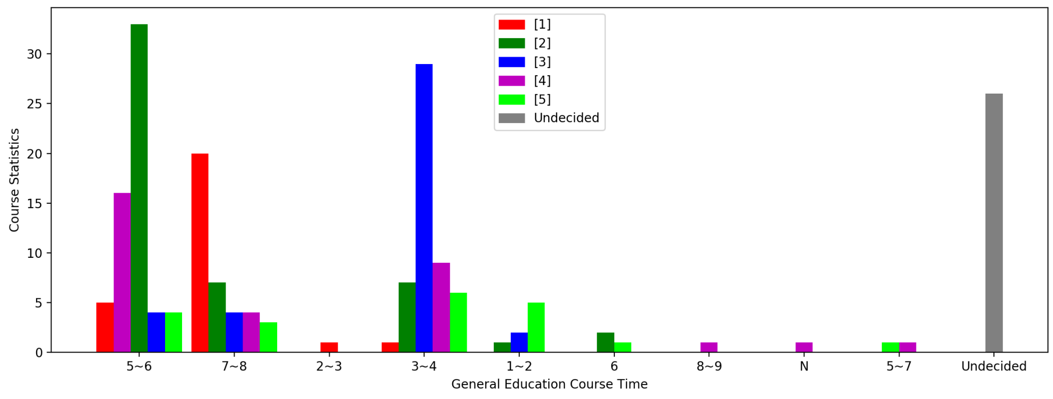
若將體育撞課統計數據與各系必修平均學分做圖如下：



由圖左上所列相關係數0.71可知，必修學分與體育課撞課之間的關係為**高度正相關**。

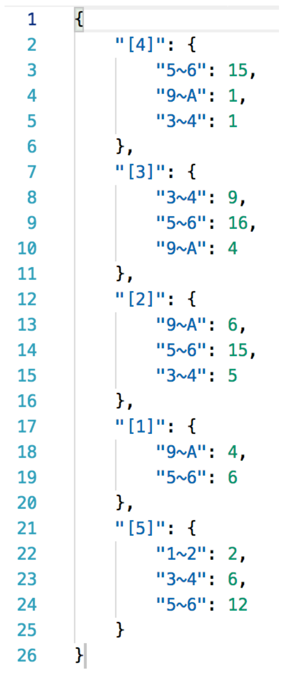
由以上結果得知，若各系必修學分愈多，撞課數越高。各系應如何排必修課的上課時間才能讓學生修到想修的通識課呢？我做了通識課與體育課上課時間統計。部分內容如右圖：
程式碼：附件八

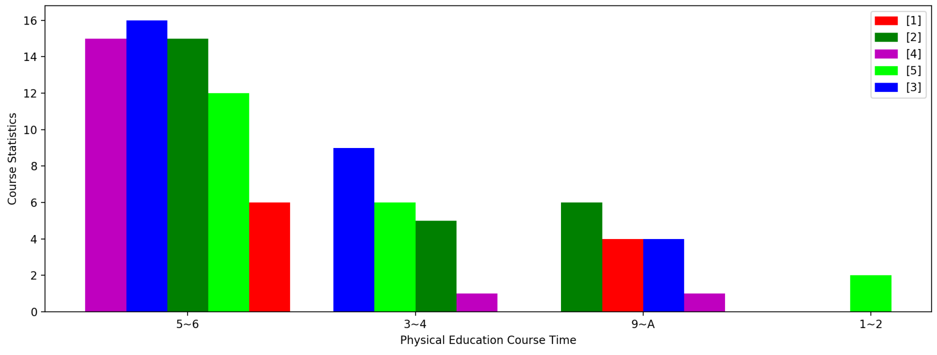

```
{
    "[1]": {
        "5~6": 5,
        "7~8": 20,
        "2~3": 1,
        "3~4": 1
    },
    "[4]": {
        "5~6": 16,
        "3~4": 9,
        "7~8": 4,
        "8~9": 1,
        "N": 1,
        "5~7": 1
    },
    "[2]": {
        "5~6": 33,
        "3~4": 7,
        "1~2": 1,
        "6": 2,
        "7~8": 7
    },
    "[3]": {
        "3~4": 29,
        "5~6": 4,
        "7~8": 4,
        "1~2": 2
    },
    "[5]": {
        "7~8": 3,
        "5~6": 4,
        "3~4": 6,
        "1~2": 5,
        "6": 1,
        "5~7": 1
    },
    "Undecided": {
        "Undecided": 26
    }
}
```



上圖是通識課的上課時間統計圖。各顏色所代表的是星期幾，例如紅色是星期一。X軸是上課的時間，Y軸是同時上課的通識課數目。由圖可知，星期一7~8、星期二5~6、星期三3~4、星期四5~6都是通識課上課的高峰。如果各系能避開這些時段排必修課的時間，就能減低撞課的機率。

同樣的，我也做了體育課上課時間的統計。部分內容如左下圖。視覺化後如下圖。


```
{
    "[4]": {
        "5~6": 15,
        "9~A": 1,
        "3~4": 1
    },
    "[3]": {
        "3~4": 9,
        "5~6": 16,
        "9~A": 4
    },
    "[2]": {
        "9~A": 6,
        "5~6": 15,
        "3~4": 5
    },
    "[1]": {
        "9~A": 4,
        "5~6": 6
    },
    "[5]": {
        "1~2": 2,
        "3~4": 6,
        "5~6": 12
    }
}
```



體育課上課的時間大多為下午，若比較上下兩視覺化後圖表，大可看出，星期二5~6節是體育課與通識課上課的高峰。各系若能避開這個時段排必修課，學生就能如願修到想修的課。

附件一：抓取成功大學課程查詢系統上的資料，並做前處理與建立存放資料的結構。

main.py:

```
import student
ncku=student.Ncku()
ncku.Search()
```

student.py:

```
import re
from requests_html import HTMLSession
class grade:
    def __init__(self):
        self.__course=[]
        self.__credit=0

    def Course(self,name,necessary,credits,hours,time):
        for i in self.__course:
            if name == i['Name']:
                return
        pattern1=r'GENERAL EDUCATION'
        pattern2=r'PHYSICAL EDUCATION (3)'
        match1=re.search(pattern1,name)
        if match1 is not None:
            return
        if pattern2 == name:
            return
        if credits is '':
            credits=0
        else:
            credits=float(credits)
        pattern=r'Elective'
        match=re.search(pattern,necessary)
        if match is not None:
            return
        else:
            yn=True
        Subject={
            'Name':name,
            'Necessary':yn,
            'Credits':float(credits),
            'Hours':hours,
            'Time':time
        }
        crash=self.Crash(Subject)
        if crash is True:
            self.__course.append(Subject)
            self.__credit+=credits
        else:
            return

    def credit(self):
        return self.__credit

    def course(self):
        return self.__course

    def Crash(self,check_course):
        for i in self.__course:
            judge=self.__crash(i,check_course)
            if judge is False:
                return False
        return True

    def __crash(self,course,check):#use intersection to judge the crash class
        course_day=set(list(course['Time']))
        check_course_day=set(list(check['Time']))
        day_intersrction=course_day & check_course_day
        if len(day_intersrction) == 0:
            return True
        else:
            for i in day_intersrction:
                course_time=set(list(course['Time'][i]))
                check_time=set(list(check['Time'][i]))
                time_intersection=course_time & check_time
                if len(time_intersection) != 0:
                    return False
            return True


def times(time):    #times function is to change the time data get from website.ex:'[1]5~6' become '[1]':[5,6]
also compute the course time
    a=time[0:3]
    if len(time)>3:
        b=time[3:].split('~')
        num=[]
        for i in b:
            if i is 'N':
                i=4.5
                num.append(float(i))
            elif i is 'A':
                i=10
                num.append(i)
            elif i is 'B':
                i=11
                num.append(i)
            elif i is 'C':
                i=12
                num.append(i)
            elif i is 'D':
                i=13
                num.append(i)
            elif i is 'E':
                i=14
                num.append(i)
            else:
                num.append(int(i))
        if len(num)>=2:
            if num[1]-num[0]>1:
                final=num.pop()
                for i in range(int(num[0])+1,int(final+1)):
                    num.append(i)
        hours=len(num)

        return [a,num],hours
    else:
        return [a,[0]],0

class department():
    def __init__(self):
        self.__grade_data={}
        self.__grade={}
        self.__crash_num={}
        self.__grade_data['total']=0
        self.__grade_data['average']=1

    def Grade(self,course,grade):
        self.__grade_data[grade]=course.course()
        self.__grade[grade]=course
        self.__grade_data['total']+=course.credit()
        self.__crash_num[grade]=0
        self.__average_credits()

    def __average_crash_num(self,num):
        return round(num/len(list(self.__grade)),3)

    def Crash(self,course):
        grades=list(self.__grade)
        time=0
        for i in grades:
            crash=self.__grade[i].Crash(course)
            if crash is False:
                self.__crash_num[i]+=1
            time+=self.__crash_num[i]

        self.__crash_num['total']=time
        self.__crash_num['average']=self.__average_crash_num(time)

        return self.__crash_num

    def grade(self):
        return self.__grade

    def Crash_data(self):
        return self.__crash_num

    def Crash_data_init(self):
        grades=list(self.__grade)
        for i in grades:
            self.__crash_num[i]=0

    def __average_credits(self):
        num=len(list(self.__grade_data))-2
        if num!=0:
            self.__grade_data['average']=round(self.__grade_data['total']/num,3)

    def total_credits(self):
        return self.__grade_data['total']

    def average_credits(self):
        self.__average_credits()
        return self.__grade_data['average']

    def grade_data(self):
        return self.__grade_data
```

```python
class school():
    def __init__(self):
        self.__department_data={}
        self.__department={}
        self.__school_credits=0
        self.__dept_num=0
        self.__crash_data={}
        self.__CrashStatic={}
    def Department(self,department,fule_name):
        name=fule_name[1:4]
        self.__department_data[name]=department.grade_data()
        self.__department[name]=department
        self.__school_credits+=department.total_credits()
        self.__crash_data[name]={}
        self.__dept_num+=1
    def avg_credits(self):
        self.__avgcredits=round(self.__school_credits/self.__dept_num,3)
        return(self.__avgcredits)
    def Crash(self,course):
        departments=list(self.__department)
        for i in departments:
            self.__crash_data[i]=self.__department[i].Crash(course)
        return self.__crash_data
    def Crash_data(self):
        return self.__crash_data
    def Crash_data_init(self):
        departments=list(self.__department)
        for i in departments:
            self.__department[i].Crash_data_init()
    def dept_amount(self):
        return self.__dept_num
    def department_data(self):
        return self.__department_data
    def department(self):
        return self.__department



class Ncku():
    def __init__(self):
        self.__school_data={}
        self.__school={}
        self.__crash_data={}

    def School(self,school,name):
        self.__school_data[name]=school.department_data()
        self.__school[name]=school
        self.__crash_data[name]={}

    def Search(self):#get the information
        url='http://course-query.acad.ncku.edu.tw/qry/'
        en_url='http://course-query.acad.ncku.edu.tw/qry/index.php?lang=en'
        session=HTMLSession()
        response=session.get(en_url)
        blocks=response.html.find('ul[id=dept_list] li')
        pattern=r'College'
        for i in blocks[3:]:
            dept=i.find('.dept')
            schoo=school()
            for j in dept:
                element=j.find('a')
                url_dept=url+element[0].attrs['href']
                response2=session.get(url_dept)
                depmnt=department()
                match=re.search(pattern,j.text)
                if match is not None or len(j.text)<=7:
                    continue
                for k in range(1,4):
                    Grade=grade()
                    elements=response2.html.find('.course_y{}'.format(k))
                    if len(elements) is 0:
                        break
                    for l in elements:#l is a subject
                        class_time={}
                        hours=0
                        h=0
                        sub=l.find('td')
                        classtime=sub[16].find('br')
                        for m in classtime:
                            cltime,h=times(m.text)
                            hours+=h
                            class_time[cltime[0]]=cltime[1]
                        Grade.Course(sub[10].text,sub[11].text,sub[12].text,hours,class_time)
                    depmnt.Grade(Grade,str(k))
                schoo.Department(depmnt,str(j.text))
            self.School(schoo,i.find('.theader')[0].text)
        return self

    def Crash(self,course):
        schools=list(self.__school)
        for i in schools:
            self.__crash_data[i]=self.__school[i].Crash(course)

        return self.__crash_data

    def Crash_data(self):
        return self.__crash_data

    def Crash_data_init(self):
        schools=list(self.__school)
        for i in schools:
            self.__school[i].Crash_data_init()

    def school(self):
        return self.__school

    def school_data(self):
        return self.__school_data
```

附件二：各系學分視覺化

main.py:

```python
import pattern


Department_data=pattern.DepartmentVisualize(ncku.school(),'The Statistic of The Required Course Credits of Departments','Department','Statistics')
```

pattern.py:

```python
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.patches as matchs
from scipy import stats
__my_color=['r','g','b','c','k','lime','m','orange','purple','r','g','b']
def CrashVisualize(dic_data,for_what,title=None,xlabel=None,ylabel=None):
    x=[]
    y=[]
    k=0
    global __my_color
    school_color=[]
    barcolor=[]
    school_name=[]
    for i in list(dic_data):
        school_name.append(i)
        school_color.append(__my_color[k])
        for j in list(dic_data[i]):
            barcolor.append(__my_color[k])
            x.append(j)
            y.append(dic_data[i][j][for_what])
        k+=1
    graph(x,y,barcolor,school_name,school_color,title,xlabel,ylabel)

    return [x,y]

def DepartmentVisualize(dict_data,title,xlabel,ylabel):
    x=[]
    y=[]
    barcolor=[]
    school_name=[]
    school_color=[]
    k=0
    global __my_color
    for i in list(dict_data):
        school_name.append(i)
        school_color.append(__my_color[k])
        for j in list(dict_data[i].department()):
            barcolor.append(__my_color[k])
            y.append(dict_data[i].department()[j].average_credits())
            x.append(j)
        k+=1
    graph(x,y,barcolor,school_name,school_color,title,xlabel,ylabel)

    return [x,y]

def graph(x,y,barcolor,school,school_color,title,xlabel,ylabel):
    plt.figure(figsize=(20,5))
    barls=plt.bar(x,y)
    k=0
    while k<len(list(barls)):
        barls[k].set_color(barcolor[k])
        k+=1
    label=__match_color(school,school_color)
    plt.legend(handles=label)
    if xlabel is not None:
        plt.xlabel(xlabel)
    if ylabel is not None:
        plt.ylabel(ylabel)
    if title is not None:
        plt.title(title)
    plt.show()


def __match_color(school,school_color):
    match=[]
    try:
        for i in range(len(school_color)):
            match.append(matchs.Patch(color=school_color[i],label=school[i]))
    except IndexError as p:
        print('barcolor',len(school_color))
        print('school',len(school))
        print(school)
        print(school_color)
    return match

def RelativeVisulize(x,y,text=None,title=None,xlabel=None,ylabel=None):
    plt.figure(figsize=(20,5))
```

```
ax=plt.subplot()
ax.scatter(x,y)
if text is not None:
    for i,txt in enumerate(text):
        ax.annotate(txt,(x[i],y[i]))
slope,intercept,r_value,p_value,std_err=stats.linregress(x,y)
plt.plot(np.array(x),np.array(x)*slope+intercept,'r',label='y={:.2f}x+{:.2f}'.format(slope,intercept))
plt.legend(['y={:.2f}x+{:.2f} Correlation Coefficient = {:2f}'.format(slope,intercept,r_value),'department'])
#print(r_value)
if title is not None:
    plt.title(title)
if xlabel is not None:
    plt.xlabel(xlabel)
if ylabel is not None:
    plt.ylabel(ylabel)
plt.show()
```

附件三：抓取通識課及體育課課程資料

main.py:

```
from common_sense import General_edu

Gurl='http://course-query.acad.ncku.edu.tw/qry/qry001.php?dept_no=A9'
GE=General_edu(Gurl)
GE.Search('.course_y0')

Purl='http://course-query.acad.ncku.edu.tw/qry/qry001.php?dept_no=A2'
PE=General_edu(Purl)
PE.Search('.course_y2')
```

common_sense.py：

```
import re
import student
import pandas as pd
import json
import numpy as np
import matplotlib.patches as matchs
from pattern import graph
from matplotlib import pyplot as plt
from requests_html import HTMLSession
class General_edu(student.grade):
    def __init__(self,url):
        self.courses=[]
        self.amount=0
        self.url=url
    def Course(self,name,necessary,credits,hours,time,time_str,time_re):
        if credits is '':
            credits=0
        else:
            credits=float(credits)
        Subject={
            'Name':name,
            'Necessary':True,
            'Credits':float(credits),
            'Hours':hours,
            'Time':time,
            'Time_str':time_str,
            'Time_re':time_re
        }
        self.courses.append(Subject)
        self.amount+=1

    def course(self):
        return self.courses

    def Search(self,u_want):
        #url='http://course-query.acad.ncku.edu.tw/qry/qry001.php?dept_no=A9'
        session=HTMLSession()
        response=session.get(self.url)
        elements=response.html.find(u_want)
        #GE=General_edu()
        for i in elements:
            course=i.find('td')
            day_time,hours=student.times(course[16].text)
            self.Course(course[10].text,course[11].text,course[12].text,hours,{day_time[0]:day_time[1]},
{course[16].text[3:]:course[16].text[0:3]},{course[16].text[0:3]:course[16].text[3:]})
        return self

    def Crash(self,university):
        university.Crash_data_init()
        for i in self.courses:
            university.Crash(i)
        return university

    def Statistic(self,name):
        self.__statisitc={}
        for i in self.courses:
            self.__judge(i['Time_str'],self.__statisitc)
        self.__re_statistic={}
        for i in self.courses:
            self.__judge(i['Time_re'],self.__re_statistic)
        with open('{} statistic.json'.format(name),'w') as f:
            f.write(json.dumps(self.__re_statistic,indent=4))
        #print(json.dumps(self.__statisitc,ensure_ascii=False,indent=4))
    def __judge(self,course_time,statistic):
        key = list(course_time)[0]
        class_time=course_time[key]
        if class_time == "未定" or class_time=='':
            class_time="Undecided"
        if key=='' or key=='未定':
            key='Undecided'
        days=list(statistic)
        for day in days:
```

```python
            if key == day:
                for j in list(statistic[day]):
                    if j == class_time:
                        statistic[day][class_time]+=1
                        return
                statistic[day][class_time]=1
                return
        statistic[key]={}
        statistic[key][class_time] = 1
        return
    def Visualize(self,title):
        x=[]
        bar_statistic={}
        colors=['k','r','g','b','m','lime','m','orange','k','grey']
        for i in list(self.__statisitc):
            for j in list(self.__statisitc[i]):
                x.append(j)
        lsSetx=list(set(x))
        lsx=self.__sort(lsSetx)
        time=list(self.__statisitc)
        ind=0
        bar_width = 0.18  # the width of the bars
        fig, ax = plt.subplots(figsize=(30,5))
        for i in time:
            position=ind-((len(list(self.__statisitc[i])))/2)*bar_width
            for j in list(self.__statisitc[i]):
                try:
                    ax.bar(position,self.__statisitc[i][j],width=bar_width,color=colors[int(j[1])])
                    position+=bar_width
                except:
                    ax.bar(position,self.__statisitc[i][j],width=bar_width,color='grey')
            ind+=1
        ind = np.arange(len(time))
        ax.set_xticks(ind-bar_width/2)
        ax.set_xticklabels(time)
        match=[]
        for i in lsx:
            try:
                match.append(matchs.Patch(color=colors[int(i[1])],label=i))
            except:
                match.append(matchs.Patch(color='grey',label=i))
        plt.legend(handles=match)
        plt.xlabel('{} Course Time'.format(title))
        plt.ylabel('Course Statistics')
        plt.show()

    def __sort(self,lsX):
        size=len(lsX)
        for i in range(size):
            try:
                int(lsX[i][1])
            except:
                temp=lsX[i]
                lsX[i]=lsX[size-1]
                lsX[size-1]=temp
                break

        for i in range(size-1):
            for j in range(size-2):
                try:

                    if int(lsX[j][1]) > int(lsX[j+1][1]):
                        temp=lsX[j]
                        lsX[j]=lsX[j+1]
                        lsX[j+1]=temp
                except:
                    print("j",j)
                    print("j+1",j+1)
                    print("lsX[j+1][1]",lsX[j+1][1])
                    print("size",size)
                    input()

        return lsX
```

附件四：統計撞課數

main.py:

```
import csvfile
import copy

GECrash=copy.deepcopy(GE.Crash(ncku))
PECrash=PE.Crash(ncku)
```

附件五：

main.py:

```
csvfile.form(GECrash,PECrash)
```

csvfile.py:

```
import csv

def form(clash1,clash2):
    with open("statistic.csv","w",encoding='utf-8') as f:
        filewriter=csv.writer(f,delimiter=',')
        filewriter.writerow(['Department','Average Credits','statistic of Clash of General Education','statistic of Clash of Physical Education'])
        for i in list(clash1.school()):
            for j in list(clash1.school()[i].department()):
                filewriter.writerow([j,clash1.school()[i].department()[j].average_credits(),clash1.school()[i].department()[j].Crash_data()['average'],clash2.school()[i].department()[j].Crash_data()['average']])
```

附件六：撞課視覺化：

main.py:

```
Crash_data=pattern.CrashVisualize(GECrash.Crash_data(),'average','The Statistic of Clash Course Between General Education Course and The Required Course of Departments','Department','Statistics')

Crash_data=pattern.CrashVisualize(PECrash.Crash_data(),'average','The Statistic of Clash Course Between Physical Education Course and The Required Course of Departments','Department','Statistics')
```

附件七：撞課數與必修學分關係作圖

main.py:

```
pattern.RelativeVisulize(Crash_data[1],Department_data[1],Crash_data[0],'The Statistic of The Correlation Between Required Credits and Clash Course Statistic of General Education Course','Statistic of Clash Course','Required Credits')


pattern.RelativeVisulize(Crash_data[1],Department_data[1],Crash_data[0],'The Statistic of The Correlation Between Required Credits and Clash Course Statistic of Physical Education Course','Statistic of Clash Course','Required Credits')
```

附件八：統計通識課及體育課上課時間

main.py:

```
GE.Statistic('GE')
GE.Visualize('General Education')

PE.Statistic('PE')
PE.Visualize("Physical Education")
```