Using Optimization to improve the current course scheduling process at Marshall and quantify and report the potential gains

Marshall Course Scheduling

First Deliverable

Shiwen Xu, Nikhil Gupta, Justin Chandler, Li Na, Yan Zhang

The Opportunity

The Course Schedule Administration Office at USC wants to optimize the course schedule. One opportunity we identified for system improvement is to increase student satisfaction score through more student-centered class scheduling. Our interest in increasing student satisfaction stems from the fact that students drive the school. Every other stakeholder care about them. Based on our interviews, the pain point for full-time students who take more courses is having **too many classes in a day**. For part-time students taking less courses, the pain point is **coming to school too often**.

Based on a few assumptions on students' preferences, we computed student satisfaction scores based on the maximum course time per day and the number of days they must come to school. Part B describes this metric in detail.

Currently, the status quo (for average satisfaction score), based on previous years' data is 2.11 for full time students and 1.94 for part time students on a 1-3 scale. There is much space for improvement based on our analysis, and our goal is to move the average satisfaction score up to as close to 3 as possible via course scheduling.

Defining the "Metric"

We define the "goodness" for students as "reasonable course time allocation" in terms of maximum time spent on any given day in a week for full time students, and total number of days a student must attend school for part time students. We define "Student Satisfaction" as a metric to measure this "goodness" and followed a set of rules to compute it. The final student satisfaction scores are on a 1-3 scale. The rest of the Part B explains our assumptions, rules, data sources, and computation procedure in producing this metric.

Overarching assumptions:

- Given that the preferences regarding course scheduling can vary depending on the student status, full-time or part-time, the "goodness" for them should be defined separately.
- For full time students, the satisfaction scores are determined by maximum time spent on any given day in a week
- For part time students, the satisfaction scores are determined by the number of days in a week they must come to school.

Rule to determine full time vs. part time status:

- The university rule to differentiate part-time and full-time students is whether they take 12 units per semester or not for undergraduate level, and 8 units or not for undergraduate level.

Data sources:

- To compute our metric--student satisfaction score on class scheduling, we used two source data files "student_course_selection.xlsx" and "Marshall_Course_enrollment_1516_1617".

Data manipulation and Metric computation

In the following procedure, Step 1 and step 2 were done using SQL (SQL code see Appendix A), and Step 3-5 were done using python (Jupyter Notebook attached as Appendix B).

- Step 1: The first thing we did is to identify full time and part time students based on their past registration information in the "student_course_selection.xlsx". By aggregating total credits that students registered for per semester, we could determine student status. Following is the rules we used to compute student status:
    a. Undergraduate students:
        i. >= 12 units: full time
        ii. <12 units: part time
    b. Graduate students
        i. >=8 units: full time
        ii. <8 units: part time

- Step 2: We joined several fields from "Marshall_Course_enrollment_1516_1617" and "student_course_selection.xlsx" by course and course section using SQL code to create an intermediate data frame named "Total_Days_Per_Student.csv". The following is the table shell for illustration purposes:

Unique Student Identifier	Class	Term	Status	Total Units	Total Days coming to School
1000586	Undergraduate	20153	FT	12	2
1000586	Undergraduate	20161	PT	10	3
1001	Graduate	20163	FT	9	4
1001248	Undergraduate	20163	PT	11	4
1001248	Undergraduate	20171	FT	15	4

- Step 3: Using python code, we joined several fields in data files "student_course_selection.xlsx" and "Marshall_Course_enrollment_1516_1617" to get the course hours per weekday for each student. The following is the table shell for illustration purposes:

Unique Student Identifier	Term	Day	First Begin Time	First End Time	Duration
1001	20163	T	18.5	21.5	3
1001	20163	M	9.5	10.83	1.33
1001	20163	W	9.5	10.83	1.33
1001	20163	T	15.5	16.83	1.33
1001	20163	H	15.5	16.83	1.33

- Step 4. We used python to calculate the max duration any student spends in school on any given day in a week Then we joined "total days coming to school" and "max course duration" The following is the table shell for illustration purposes:

Unique Student Identifier	Class	Term	Status	Total Units	Total Days coming to School	Max Duration spent any given day at School
1000586	Undergraduate	20153	FT	12	2	3.66
1000586	Undergraduate	20161	PT	10	3	3.66
1001	Graduate	20163	FT	9	4	4.33
1001248	Undergraduate	20163	PT	11	4	3.66
1001248	Undergraduate	20171	FT	15	4	4.33

- Step 5. Depending on the number of days students must come to school and the max course hours on any given day, we computed the satisfaction score. The highest satisfaction score possible is 3, and lowest score possible is 1. Provided below are the computation rules for the "student satisfaction score". We have used our judgement to set the various cut-offs for assigning the score based on the class schedules and the parameters computed in steps 1-4. The final dataset looks the same as that from the previous step but adds another column having the "satisfaction scores".

  Full time students:
  - ➢ We assumed that full time students' don't want to have too many classes in a day.
  - ➢ If max time spent at school on any given day is >= 6 hours, deduct satisfaction score by 2

- ➢ If max time spent at school on any given day is between 4-6 hours (6 not included), deduct satisfaction score by 1

Part time students:

- ➢ We assumed that part time students' priority is to minimize the number of days they must come to school.
- ➢ If number of days coming to school >= 3 days, deduct satisfaction score by 2.
- ➢ If number of days coming to school is 2 days, deduct satisfaction score by 1.

Based on the above rules, we generated a "student satisfaction score" for each student. We then proceeded to compute the mean of the score for full time and part time students. This mean is the final metric that we are aiming to improve from its current levels. The metric computation is described in more detail in later sections of this report along with the Python code.

Why “Student Satisfaction Score”?

- ***Computable:*** Our metric is computable based on student registration data and course information with five simple steps. This is demonstrated in the data manipulation and metric computation procedure

- ***Actionable:*** Student satisfaction scores can be affected by course scheduling decision. For example, if a program restricts scheduling popular courses that are often taken in the same semester on the same day, it is possible to reduce the number of max course time per day. There is also a tradeoff, scheduling popular courses on different days is likely to increase the number of days students need to come to school. Finding the right balance can optimize students’ satisfaction score

- ***Simple:*** Student satisfaction score is a simple and interpretable metric, since it is computed based on two simple factors: maximum course time spent on any given day or number of days students must come to school

- ***Enlightening for the notion of “goodness”:*** The notion of “goodness” is majority of students can be satisfied with the class scheduling, that is, full time students’ maximum class load per day is under certain threshold and part time students don’t have to come to school for more than desired. And the satisfaction score is an intuitive aggregated metric to represent “goodness”.

Scope for Improvement in Computed Metric

After each student's satisfaction score is computed, we used it to arrive at the average satisfaction score for both FT and PT students separately. This is the status quo of student satisfaction level towards class scheduling. Provided below are some summary statistics for key fields in our final dataset prepared:

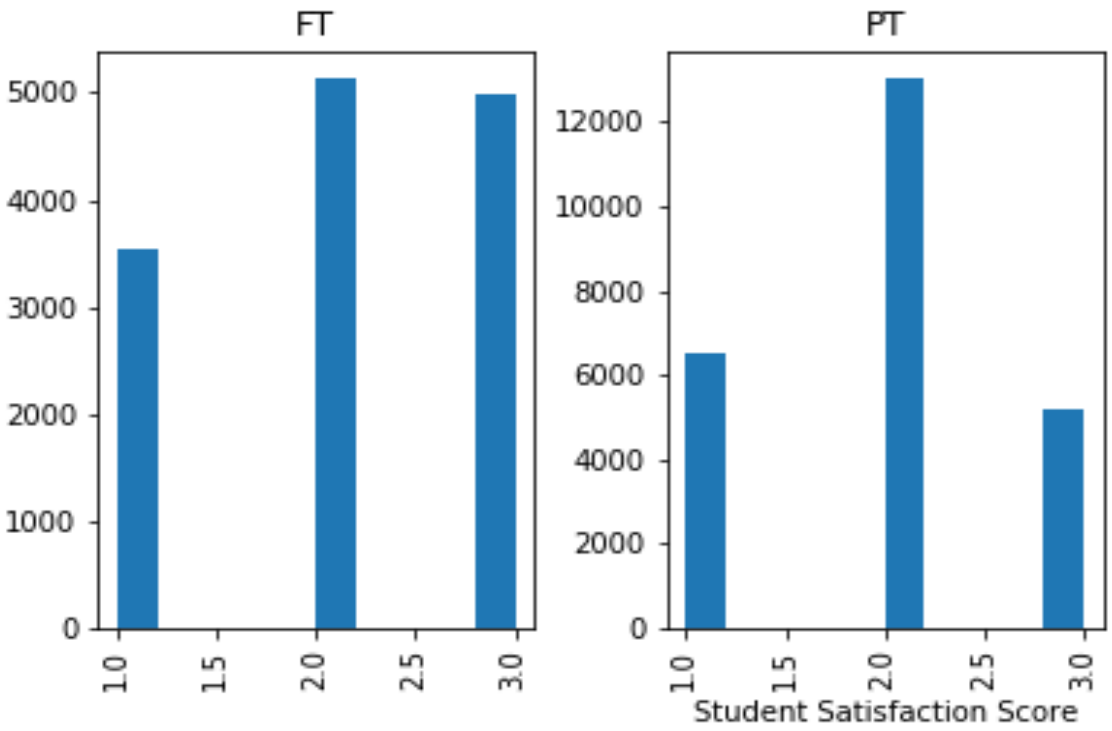
Full Time Students			
Parameters	Days that student comes to school	Max Course time spent any given day	Student Satisfaction Score
# of Records	13660	13378	13660
Average	3.42	5.48	2.11
Std. Dev.	1.15	2.80	0.78
Min	0	0	1
1st Quartile	2	3.66	1
Median	4	4.84	2
3rd Quartile	4	6.82	3
Max	6	16	3

Part Time Students			
Parameters	Days that student comes to school	Max Course time spent any given day	Student Satisfaction Score
# of Records	24736	24043	24736
Average	2.19	2.67	1.95
Std. Dev.	1.02	1.32	0.69
Min	0	0	1
1st Quartile	2	1.83	1
Median	2	1.83	2
3rd Quartile	3	3.33	2
Max	5	16	3

Based on the above dataset, provided below are the current values for our metric based on the previous year schedule. As may be seen, there is a scope for improvement in the metric for both full time and part time students.

Student Status	Average Score	Max Score Possible	Scope for Improvement
Part Time	1.94	3.00	43%
Full Time	2.10	3.00	55%

Score Distribution



Way Forward

We computed the current satisfaction scores based on the past data and determined the status quo that we aim to improve. This means that through our analysis, we target to increase the average student satisfaction scores to bring them as close to the max value possible i.e.3.00.

During the next phase of our project, we aim to generate substantial number of simulations and compute the average value of the metric across these which shall then be utilized to optimize our course schedule from Student perspective.

Appendix A. SQL code for step 1 and step 2 computation

Part B Step 2: SQL Query that produce the table showing the total days each student came to school:

```
select total_days.randomized_unique_identifier, total_days.class, total_days.term,
    total_days.status, total_days.total_units,
        (total_days.Monday
        +total_days.Tuesday
        +total_days.Wednesday
        +total_days.Thursday
        +total_days.Friday
        +total_days.Saturday
        +total_days.Sunday) as total_days
from
    (select aggregation.randomized_unique_identifier, aggregation.class, aggregation.term,
        case
            when (aggregation.class = 'Graduate' and aggregation.total_units >= 8) then 'FT'
            when (aggregation.class = 'Graduate' and aggregation.total_units < 8) then 'PT'
            when (aggregation.class = 'Undergraduate' and aggregation.total_units >= 12) then 'FT'
            when (aggregation.class = 'Undergraduate' and aggregation.total_units < 12 ) then 'PT'
        end as status,
        aggregation.total_units,
        aggregation.days as days,
        case when position('M' in aggregation.days) > 0 then 1 else 0 end as Monday,
        case when position('T' in aggregation.days) > 0 then 1 else 0 end as Tuesday,
        case when position('W' in aggregation.days) > 0 then 1 else 0 end as Wednesday,
        case when position('H' in aggregation.days) > 0 then 1 else 0 end as Thursday,
        case when position('F' in aggregation.days) > 0 then 1 else 0 end as Friday,
        case when position('S' in aggregation.days) > 0 then 1 else 0 end as Saturday,
        case when position('Z' in aggregation.days) > 0 then 1 else 0 end as Sunday
    from
        (select randomized_unique_identifier,
            case when class in ('G','L','O') then 'Graduate' else 'Undergraduate' end as class,
            student.term,
            group_concat(distinct replace(first_days,'SU','Z')) as days,
            sum(units) as total_units
        from student_course_selection_1516 student join marshall_course_enrollment_1516_1617 courses
            on (student.term = courses.Term and student.section = courses.Section)
        group by 1,2,3) as aggregation
    ) as total_days
order by 1,2,3,4
```

Computing the Metric

March 6, 2018

0.0.1 Appendix B

0.0.2 Import the two original data frames and explore the data frames

```
In [4]: import pandas as pd
        schedule=pd.read_excel('Marshall_Course_Enrollment_1516_1617.xlsx')
        students=pd.read_excel('Student_Course_Selection_1516.xlsx')

In [87]: #schedule.head()

In [88]: #students.head()
```

0.0.3 Select relavant columns and join two tables

```
In [7]: students2=students[["Randomized Unique Identifier", "Term", "Course", "Section", "Units
        schedule2=schedule[["Course",  "Term", "Section","First Begin Time", "First End Time",

In [89]: #schedule2.head()

In [90]: ##Merge two tables by "Section" and "Term"

         master=students2.merge(schedule2, how="left", on=["Section", "Term"])
         #master.head()
```

0.0.4 Mainpulate Data to get max course duration

```
In [14]: ##The function to manipulate time data, convert it to intergers

         import numpy as np
         def convert(inputTime):
             try:
                 hh, mm, ss=inputTime.split(':')
                 ans=round(int(hh)+int(mm)/60+int(ss)/3600,2)
             except:
                 ans=np.nan
             return ans

         ##test
         print(convert("11:30:00"))
```

11.5

In [17]:
```
##get student id, term, course, section, day, duration of the course from the master
##save the data in a dictionary

ans={}
for index, row in master.iterrows():
    id=row["Randomized Unique Identifier"]
    term=row["Term"]
    course=row["Course_x"]
    section=row["Section"]
    days=str((row["First Days"]))
    beg=convert(str(row["First Begin Time"]))
    end=convert(str(row["First End Time"]))
    duration=round(end-beg,2)
    for day in ['M','T','W','H','F','S','U']:
        if day in days:
            key=(id, term, day, beg, end)
            ##because there are some cases students' record show they take two course
            ##which have the same beginning and end time,
            ###but in fact, these two courses are the same one. This may be due to a
            ###has a different name/section in two different departments.
            ###In whichever case, we have to remove duplication.The following stateme
            if key in ans: continue
            ans[key]=duration
```

In [19]:
```
### save the needed data back into a data frame
lines=[]
for id, term, day, beg, end in ans:
    duration=ans[id, term, day, beg, end]
    lines.append([id, term,  day, beg, end, duration])


output=pd.DataFrame(lines, columns=["Randomized Unique Identifier", "Term", "Day",
                                    "First Begin Time", "First End Time", "Duration"]
```

In [91]:
```
## view the output table
#output.head()

###The output table lists course duration per student per semester per course
###we have to furthur aggregate it to get the max duration across weekdays
```

In [25]:
```
###first sum up the course time for each day each student
groups=output[["Randomized Unique Identifier", "Term", "Day", "Duration"]].groupby(['
course_duration=groups.sum()
```

In [92]:
```
#course_duration.head()
### course_duration table lists course duration per student per term per day
```

```
In [94]: ## The next aggreation is to find the max course time in a week
         groups2=course_duration.groupby(['Randomized Unique Identifier', "Term"])

In [95]: max_duration=groups2.max()
         #max_duration.head()
```

max_duration.head()

0.0.5 Read in the "Total_Days_per_Student.csv" file created by SQL manipulation

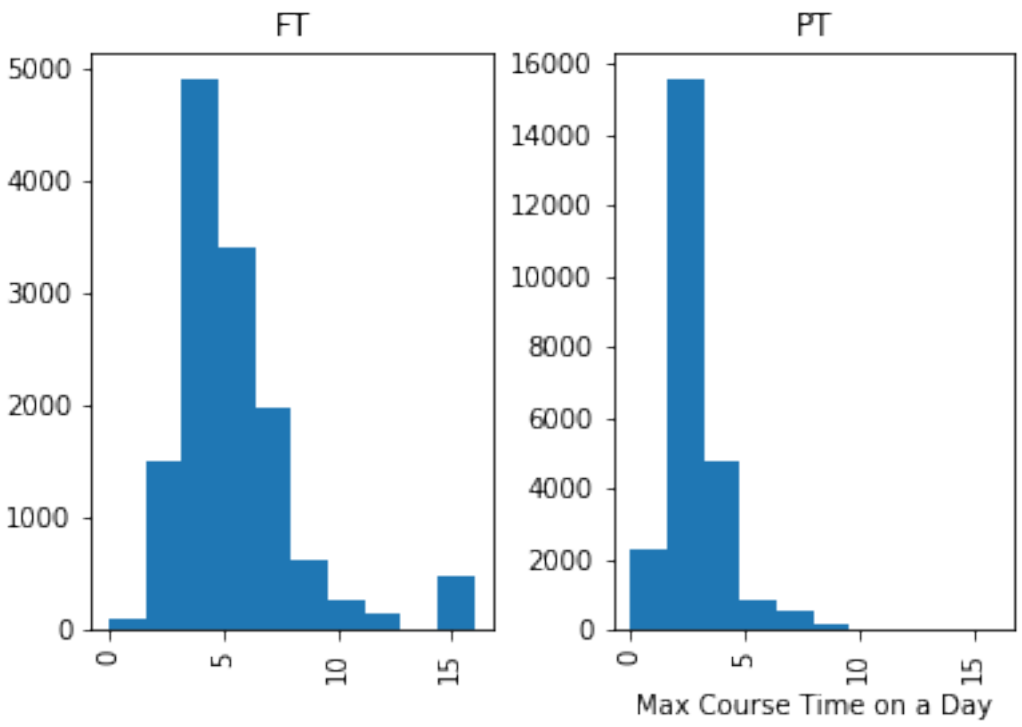
```
In [29]: total_days=pd.read_csv("Total_Days_per_Student.csv")

In [96]: #total_days.head()

In [51]: ### join the two tables

         combined=pd.merge(total_days, max_duration, how="left",
                           left_on=["randomized_unique_identifier", "term"],right_index=True)
         FTdurationMedian = combined[combined.status=="FT"]['Duration'].median()
         PTdurationMedian = combined[combined.status=="PT"]['Duration'].median()
         FTdaysMedian = combined[combined.status=="FT"]['total_days'].median()
         PTdaysMedian = combined[combined.status=="PT"]['total_days'].median()
         FTdurationMean = combined[combined.status=="FT"]['Duration'].mean()
         PTdurationMean = combined[combined.status=="PT"]['Duration'].mean()
         FTdaysMean = combined[combined.status=="FT"]['total_days'].mean()
         PTdaysMean = combined[combined.status=="PT"]['total_days'].mean()

In [58]: print("Days",FTdaysMean, FTdaysMedian,PTdaysMean,PTdaysMedian)
         print("Duration",FTdurationMean,FTdurationMedian,PTdurationMean,PTdurationMedian)
```

```
Days 3.4249633967789164 4.0 2.191219275549806 2.0
Duration 5.482706682613818 4.84 2.672308780103285 1.83
```

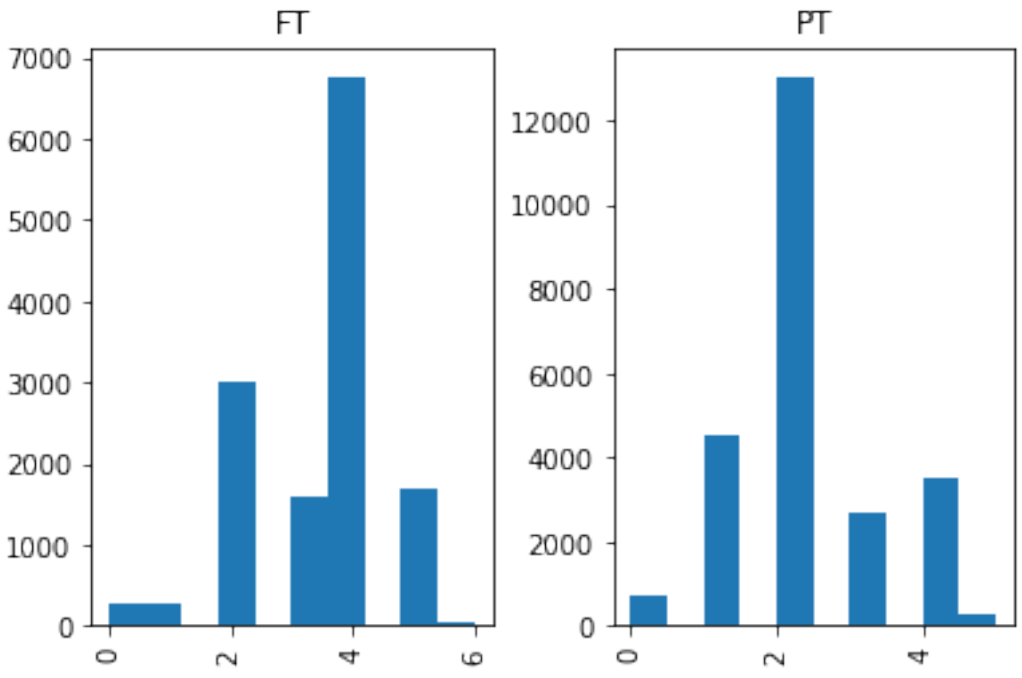
```
In [97]: ### Now we have the final table that lists the total_days a student coming to school
         #combined.head()

In [33]: ## save the final table
         combined.to_csv("student_totalDays_maxDur.csv")
```

0.0.6 Data Exploration

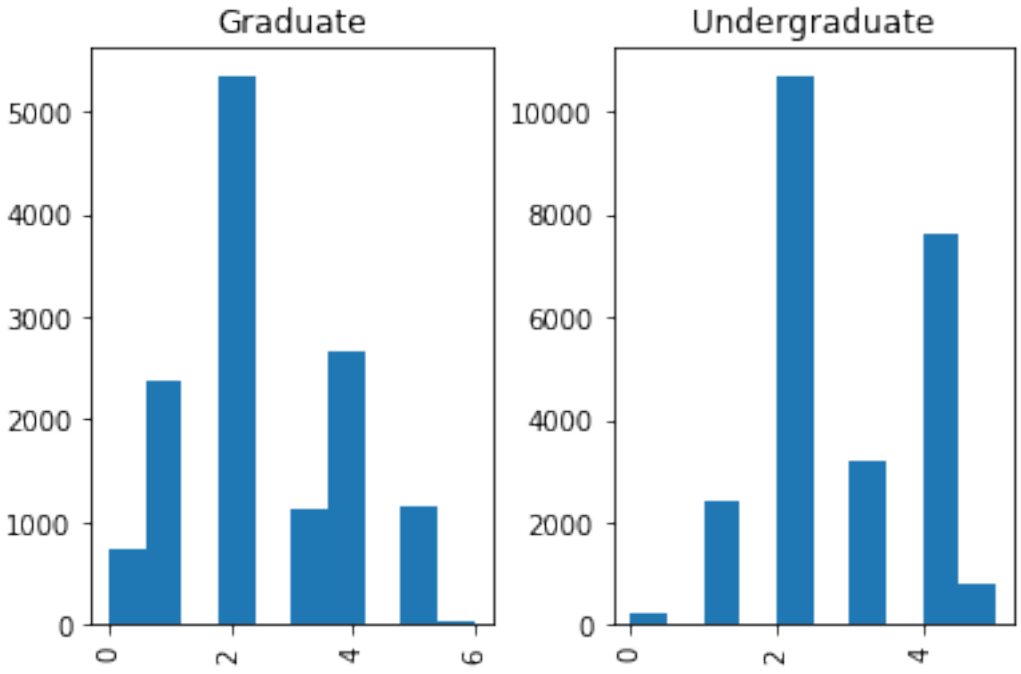
```
In [98]: import matplotlib.pyplot as plt
         combined.hist(column='Duration', by='status')
         plt.xlabel("Max Course Time on a Day")
         plt.show()
```



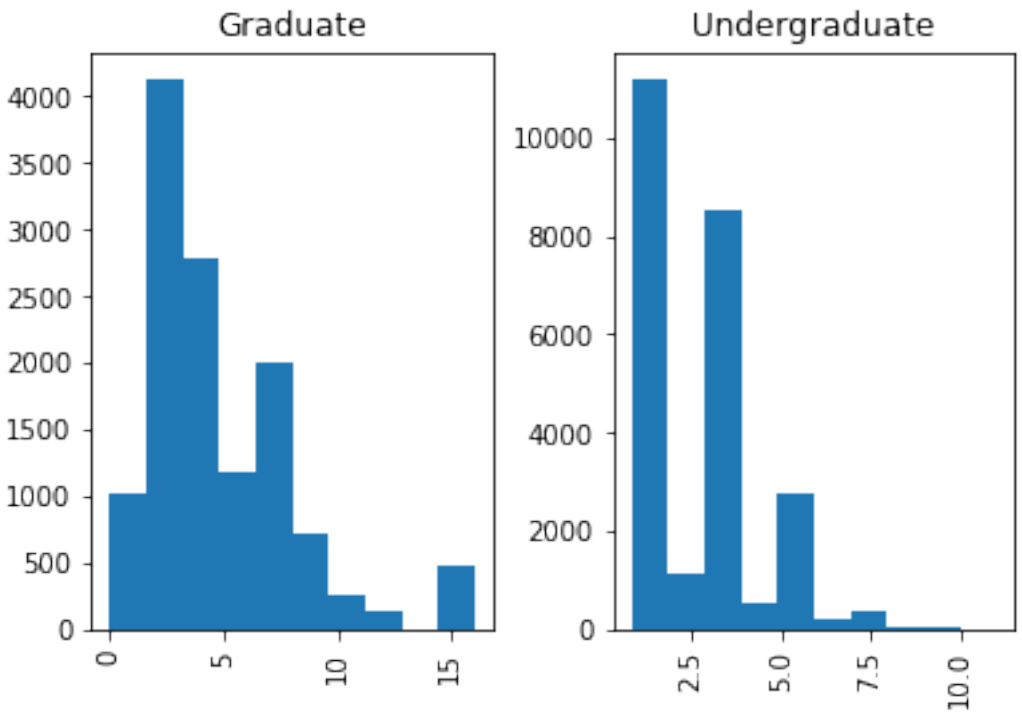

```
In [99]: combined.hist(column='total_days', by='status')
         plt.show()
```



In [100]:
```
combined.hist(column='total_days', by='class')
plt.show()
```



In [101]:
```
combined.hist(column='Duration', by='class')

plt.show()
```




```
In [102]: def sat (status,days,duration):
              if status == 'PT':
                  if days>=3:
                      return 1
                  elif days==2:
                      return 2
                  else:
                      return 3
              else:
                  if duration >= 6:
                      return 1
                  elif duration <4:
                      return 3
                  else:
                      return 2

          data=combined
          data['satisfaction'] = [sat(data['status'][i],data['total_days'][i],
               data['Duration'][i]) for i in range(len(data['class']))]
          data[data.status =="FT"]['satisfaction'].mean()
          #data[data.status =="PT"]['satisfaction'].mean()
          #data[data.status =="PT"].iloc[:,5:8].describe()

Out[102]: 2.1072474377745243
```

```
In [103]: #data[data.status =="FT"].iloc[:,5:8].describe()
```

```
In [86]: data.hist(column='satisfaction', by='status')
         plt.xlabel("Student Satisfaction Score")
         plt.savefig('distribution.png')
         plt.show()
```

```
<matplotlib.figure.Figure at 0x1a603e8a7f0>
```