Wharton Classes: When, Where and Why

OIDD 321 Final Project Motivated by Stories of Frustrated Wharton Kids

Grant Cho | David Fan | Kevin Tan | Sam Abello | Patrick Rich

I. Introduction

Problem and Model Objective

As Business Analytics (BUAN) gains popularity at Wharton, the demand for STAT and OIDD classes needed to fulfill BUAN concentration requirements has increased. This results in **two problems that we wanted to address with our model:** (1) inefficient allocation of classrooms to classes based on enrollment, and (2) conflicts in scheduling that prevent students from taking courses that they would want to take in the same semester.

The first part of our model is linear and allocates classrooms in Huntsman Hall and Steinberg-Dietrich Hall to classes to **maximize classroom utilization**. The second part of our model is non-linear and schedules STAT and OIDD classes separately to minimize covariances between classes scheduled at the same times. We wanted to **minimize overlap between related classes** because based on our experiences, students tend to want to take classes that are similar (e.g. classes that cover computer programming for data roles) to gain deeper knowledge in certain fields. Further explanations of our model will be provided in the "Model" section.

Assumptions

Our team made 9 simplifying assumptions to overcome the lack of available data and to avoid over-complicating our model for Solver to run smoothly.

1. **Classes held in Huntsman and Steinberg-Dietrich halls only**
   This was a fair assumption given that most STAT and OIDD lectures are held in these two halls. We further assumed that no BUAN classes take place in conference rooms, above the 3rd floor of Huntsman or on the ground floor of Steinberg-Dietrich, which is generally the case.

2. **Only undergraduate students included**
   Some classes have seperate quotas for MBA students, but we decided to ignore these as MBA quotas were complicated to account for and MBA course demand data was not easy to obtain.

3. **Recitations not included**
   This was a simplifying assumption that allowed us to focus on lecture scheduling, and was reasonable given that not many BUAN classes have recitations.

4. **No lectures happen on Fridays**
   This allowed us to focus our model on Monday-Thursday time slots only, which is typically when BUAN lectures take place.

5. **Optimizing for Spring 2020 semester only**
   Because of the data that we had immediate access to and our desire for findings to be readily applicable, we structured our model to predict and optimize for the upcoming Spring semester, using only classes offered in the Spring.

6. **Constant course enrollment capacities**
   The maximum number of students that can be enrolled in a certain class in Spring 2020 was assumed to be the same as in previous Spring semesters, giving us a simpler way to predict enrollment and apply our model.

7. **Courses with unknown enrollment capacity assumed to hold a maximum of 60 students**
   This was a representative average of enrollment capacity for courses which did not have available data on PennInTouch.

8. **6 time slots for classes each day**
   Specifically, we had 1.5 hour time slots from 9am-6pm, which we believe is a realistic picture of Wharton class scheduling.

9. **Weighted average to find course demand**
   We recognize that the past enrollment data we obtained will unlikely be truly representative of the actual future demand of a class. Thus, **our predicted course demand was based on a weighted average of 70% historical demand and 30% Penn Course Review-based regression**, which we refer to as "prior demand." A class's prior demand was calculated by weighting its Penn Course Review ratings (eg. amount learned, course quality, etc.) by arbitrary coefficients reflecting the importance of each of these ratings (Appendix D). The coefficients used were based on our own experiences. For instance, we agreed that course quality was the most important factor for classes and gave it a coefficient of 30 compared to amount learned, which was assigned a coefficient of 15. It should be noted these coefficients can be easily adjusted to fit individual preferences.

II. Data Collection

Data Sources

The majority of the data used in our model was obtained through API requests (Appendices A, B, C) while the rest was manually obtained online. We gathered course descriptions and rating data from Penn Course Review and course offerings, enrollment capacities and historical demands from Wharton Syllabi and PennInTouch. We also extracted classroom sizes from Wharton Utilities.

Measuring Class Covariance

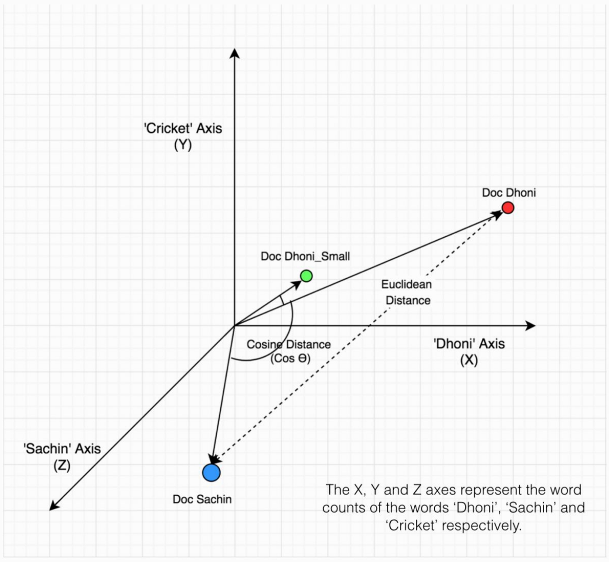
To achieve goal of better scheduling classes by avoiding conflicting class placements for heavily related classes, we needed to come up a measure for class similarity and minimize that said metric through our model. To do so, we decided to use text mining techniques on course descriptions obtained from the aforementioned Penn Labs API to quantify class similarity or covariance. The specific method we chose was cosine text similarity.

Cosine Text Similarity

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}$$

Cosine Similarity calculation for two vectors A and B [source]

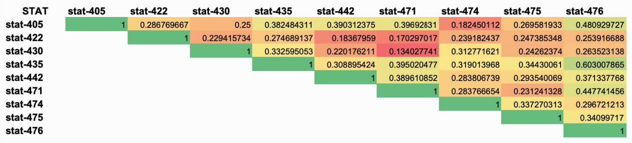
Cosine similarity calculates similarity by measuring the cosine of angle between two vectors. In the case of text, these vectors are simply count representations of the different words that exist in the corpus (paragraphs). Mathematically speaking, cosine similarity is a measure of similarity between two non-zero vectors of an inner product space that measures the cosine of the angle between them. The cosine of 0° is 1, and it is less than 1 for any angle within the interval (0,π] radians. It is thus a judgment of orientation and not magnitude: two vectors with the same orientation have a cosine similarity of 1, two vectors oriented at 90° relative to each other have a similarity of 0, and two vectors diametrically opposed have a similarity of -1, independent of their magnitude. Cosine similarity is advantageous because even if the two similar documents are far apart by the Euclidean distance (i.e. due to the sizes of the documents), chances are they may still be oriented closer together. **The smaller the angle, higher the cosine similarity, with 1 being the most similar and 0 being the least.**



A graphical representation of Cosine Text Similarity

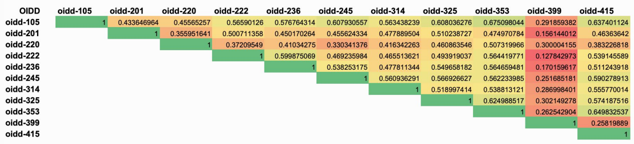
Results

After applying the cosine method, **we obtained the following similarity/covariance matrices**:



STAT	stat-405	stat-422	stat-430	stat-435	stat-442	stat-471	stat-474	stat-475	stat-476
stat-405	1	0.286769667	0.25	0.382484311	0.390312375	0.39692831	0.182450112	0.269581933	0.480929727
stat-422		1	0.229415734	0.274689137	0.18367959	0.170297017	0.239182437	0.247385348	0.253916688
stat-430			1	0.332595053	0.220176211	0.134027741	0.312771621	0.24262374	0.263523138
stat-435				1	0.308895424	0.395020477	0.319013968	0.34430061	0.603007865
stat-442					1	0.389610852	0.283806739	0.293540069	0.371337768
stat-471						1	0.283766654	0.231241328	0.447741456
stat-474							1	0.337270313	0.296721213
stat-475								1	0.34099717
stat-476									1

STAT Class Similarity Matrix



OIDD	oidd-105	oidd-201	oidd-220	oidd-222	oidd-236	oidd-245	oidd-314	oidd-325	oidd-353	oidd-399	oidd-415
oidd-105	1	0.433646964	0.45565257	0.56590126	0.576764314	0.607930557	0.563438239	0.608036276	0.675098044	0.291859382	0.637401124
oidd-201		1	0.355951641	0.500711358	0.450170264	0.455624334	0.477889504	0.510238727	0.474970784	0.156144012	0.46363642
oidd-220			1	0.37209549	0.41034275	0.330341376	0.416342263	0.460863546	0.507319966	0.300004155	0.383226818
oidd-222				1	0.599875069	0.469235984	0.465513621	0.493919037	0.564419771	0.127842973	0.539145589
oidd-236					1	0.538253175	0.477811344	0.549658182	0.564659481	0.170159617	0.511243918
oidd-245						1	0.560936291	0.566926627	0.562233985	0.251685181	0.590278913
oidd-314							1	0.518997414	0.538813121	0.286998401	0.555770014
oidd-325								1	0.624988517	0.302149278	0.574187516
oidd-353									1	0.262542904	0.649832537
oidd-399										1	0.25819889
oidd-415											1

OIDD Class Similarity Matrix

The figures above indicate that classes such as STAT471 and STAT430, STAT474 and STAT405, OIDD245 and OIDD220 (i.e. redder highlighted cells) are not similar while OIDD353 and OIDD105, STAT435 and STAT476 (i.e.greener highlighted cells) are similar. OIDD 399 appears to be different from all classes, likely because it is an independent study.

The results seem generally acceptable; however, some minor corrections are needed to improve the model. For instance, STAT422 had a very short and uninformative class description, and, using personal experiences, we have corrected our results to make it more highly correlated to other predictive analytics classes such as STAT471.

III. Model

Note: Due to Solver's 200-decision variable limit, we broke our model into two parts: (1) Classroom allocation and (2) Class scheduling (see Excel file for visuals).

Part 1: Classroom Allocation

The objective of this part was to **maximize utilization of classroom capacity** by assigning classes with enrollment sizes most similar to classroom capacities. We used linear optimization to minimize the difference between assigned classroom capacities and expected class enrollments. In this case, we took class enrollment as the lower of our forecasted demand for a class and the class's maximum enrollment capacity. Thus, **the decision variables were integer variables in a table of classes by classroom capacities.** The decision variable table indicated how many sections of a class were assigned to a classroom of a specific capacity.

Parameters

3 main parameters were used in this part: **room type, time slots available per room type, and forecasted demand.** Room types were designated according to room capacity and summed into the categories. Time slots were calculated based on the 1.5-hour time slots from 9am-6pm on either the Mon-Wed or Tues-Thurs schedule per room. This was then multiplied to the total number of rooms per category. Forecasted demand was obtained using the weighted method discussed in the "Introduction".

Constraints

We incorporated 3 constraints in this part:

1. **Every section must be assigned a classroom**
   The sum of decision variables per class was set to be equal to the given sections per class offered in the Spring semester.

2. **Classes can only be assigned to rooms that are large enough to accommodate their demand**
   Classroom capacity was set to be greater than or equal to the forecasted demand per section (i.e. forecasted demand of a class divided by number of sections of the class).

3. **Rooms cannot have more classes scheduled than there are time slots for**
   The sum of decision variables (i.e. number of sections) per room type was set to be less than or equal to the number of time slots available per room type.

Objective

For each class, we took the difference between total capacity of allocated classrooms and total predicted enrollment, and added the numbers together to find the total unused classroom capacity, which we want to minimize. Total classroom capacity the sumproduct of a class's column on the decision table and a parameter column with the corresponding classroom capacities.

Part 2: Class Scheduling

This part of the model utilized nonlinear optimization to assign time slots to classes in a manner that would **minimize schedule conflicts between high-demand classes**.The similarities/covariances of class pairings were used as proxies for class demand as we assumed that students would be most interested in taking similar classes, and our model sought to minimize the sum of covariances between classes that would occur at the same time slots. Due to the decision variable limits of Solver, we had to separate STAT and OIDD classes and ran department-specific (i.e. mutually exclusive) optimization models.

There were 2 sets of decision variables for each iteration. The first was a table of classes by time slots to determine class assignments to time slots. **The second was a covariance decision table between classes**. The decision variables in this covariance table were binary and set to be equal to 1 if a class pair was scheduled at the same time slot and 0 otherwise.

Parameters

The **class similarity/covariance matrices** we obtained using the method discussed in the "Data Collection" section were the parameters used here.

Constraints

We enforced 2 constraints in this part:

1. **Each class must be assigned one time slot**
   The sum of decision variables per class in the time slot table were set to be equal to be at least 1.

2. **Linking decision variables in the time slot table to those in the covariance table**
   The corresponding covariance decision cell (i.e. covariance of a class pairing) was set to be greater than or equal to the sumproduct of both paired classes' columns of decision variables in the time slot table. This way, the sumproduct only equals 1 if both classes are scheduled at the same time slot. This constraint forces the corresponding covariance decision variable to equal 1 if and only if the two corresponding classes are scheduled together.

Objective

To find the total covariance of all classes scheduled at the same time slots, **we simply took the sumproduct of the covariance decision table and the covariance matrix parameter.** This was done separately for both STAT and OIDD iterations of the model.

IV. Results Interpretation

Findings and Takeaways

After running solver, we were able to find a feasible solution that minimized our objectives and satisfied all constraints (Appendix E). Upon analysis, we arrived with 3 key findings:

1. **Plenty of unused classroom capacity**

   The minimum difference in classroom capacity and class enrollment that we can achieve at the optimal solution is 192.885. This means that there will still be at least ~193 unoccupied classroom seats after assigning classes in a way that would maximize seating. We noticed that **there are 3 classes that significantly contribute to unoccupied seating**: STAT422, OIDD105 and OIDD236, which each have ~31, ~28 and ~37 unoccupied seats respectively. This is because the predicted enrollment counts for all three classes are largely misaligned with the available classroom capacities. For instance, OIDD236, which offers 1 section, has a total enrollment of ~87, but the smallest classroom available to fit that many students has a capacity of 123. To further minimize unoccupied seats, Wharton should optimize the number of sections offered for these 3 classes (holding all else equal and assuming no construction of new classrooms). For instance, by offering 2 sections of OIDD236, each section will have ~44 students and can each use classrooms with 50 seats, minimizing total unoccupied seats from this class to ~12.

2. **Smaller classrooms mostly needed**

   At the optimal solution, **12 sections require classrooms with 25 seats, 7 sections require classrooms with 50 seats, while no sections require classrooms with either 150 or 299 seats.** This is primarily because of the abundance of seminar-style and SAIL (i.e. Structured Active In-class Learning) classes in BUAN, which typically limit the number of students enrolled per section to maximize in-class learning, discussion and interaction. PennInTouch confirms this point, as the majority of BUAN classes listed have a maximum enrollment capacity ranging from around 35 to 45 students.

3. **Same time slots for unrelated class pairings**

   At the optimal solution, **the following classes can be scheduled simultaneously:**

   - STAT422 and STAT435
   - STAT430 and STAT471
   - STAT405 and STAT475
   - OIDD105 and OIDD201
   - OIDD220 and OIDD245
   - OIDD314 and OIDD325
   - OIDD222 and OIDD353
   - OIDD399 and OIDD415

   Because we are trying to minimize the total covariance of classes scheduled together, it makes sense that **the classes that were paired together at the same times are relatively unrelated** (i.e. low covariance) at the optimal solution. For instance, STAT405 teaches programming in the R computer language whereas STAT475 covers designing different types of sample surveys, two rather dissimilar topics in statistics (i.e. covariance = 0.270).

   Additionally, it is worth noting that **the minimum total covariance of OIDD classes paired together is higher (2.106) than that of STAT classes (0.587)**. This likely stems from the fact that there are 11 OIDD classes but only 9 STAT classes in BUAN, resulting in more overlaps in OIDD given the same 6 time slots (i.e. 5 class pairings for OIDD versus 3 for STAT). Moreover, on average, OIDD classes seem to be more related to each other compared to STAT classes, as the average covariance of all OIDD pairings is 0.466 versus 0.305 for STAT pairings.

   Going one step further, we conclude that **all other classes not listed above should not be paired together as they are relatively related** (i.e. high covariance). Furthermore, no class can be paired with itself, and this rule was upheld at the optimal solution despite us not explicitly specifying it as a constraint. This is likely because Solver automatically ruled out class pairings with a covariance of 1 (i.e. the highest possible covariance for a pair) when attempting to minimize total covariance among pairs.

V. Risks & Improvements

Academic Restrictions and Size led to Oversimplifications in the Final Model

Running the initial model wasn't feasible given the thousands of decision variable cells Solver couldn't run and the academic restrictions and bugginess of OpenSolver. As such, we had to simplify the model extensively, resulting in a few limitations.

Limitations

We identified three main risks and limitations to using this model:

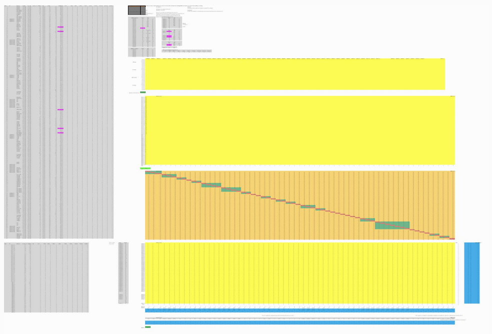
1. **Doesn't account for overlap**
   Firstly, the model does NOT account for STAT and OIDD classes potentially overlapping with covariances. This is because we've only accounted for STAT classes overlapping with other STAT classes and OIDD classes doing the same with other OIDD classes. As a result, we aren't able to determine which BUAN classes should coexist with each other, rather **we're only able to select the most in-demand STAT/OIDD classes should these sections be incapable of coexisting**.

2. **Excludes MBA students**
   Secondly, the current model does not include MBA students, which **could result in excluding graduate level classes that are high sought after by graduate students but not necessarily by undergraduate students**. For example, STAT 476 is an advanced class that reserves 50 seats for graduate students while only 25 for undergraduates. If, say, 5 undergraduates enrolled in that course, while 50 graduate students did the same, the model would not recommend including STAT 476 into the course schedule despite having an enrollment rate of 79%.

3. **Does not account for irregular schedules**
   Thirdly, model does not take into consideration classes that meet for 3 hours straight and don't meet twice a week, which can lead to an inaccurate assortment of classes, such as STAT 475.

Initial Model

The initial model was designed to optimize/solve for which STAT and OIDD classes belonged to which classrooms throughout the school week. This differs from our current model in that the initial model was designed to efficiently allocate all STAT and OIDD classes, not just those related to BUAN, in all relevant classrooms. Doing so, however, required the usage of over 3000 decision variable cells, which is too much for Solver to handle. We then attempted to utilize OpenSolver but found that **Wharton had limited the capabilities of the software to the point where the problem still could not be solved**.


The initial model had well over 3,000 decision variable cells, rendering it unsolvable for Solver

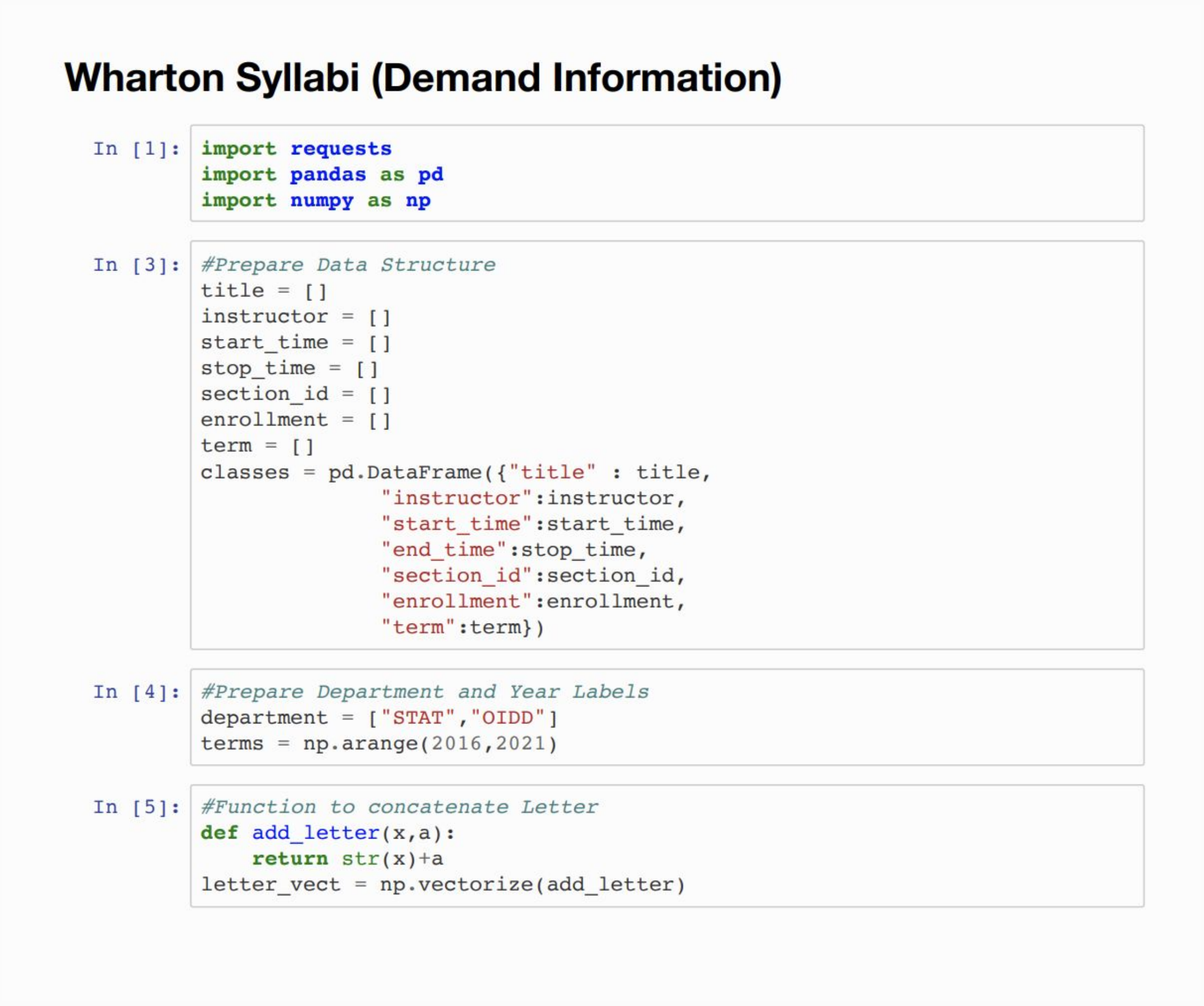
Future Model Improvements

Moving forward, we could implement better estimates for covariances as cosine similarity is not the most accurate estimate and have two time slots allocated for each class. We would also use Crystal Ball and OptQuest to simulate and optimize for uncertainty so that the model wouldn't only be relevant for one semester. Instead of using moving averages to forecast class demand, we would opt for exponential smoothing which we would use to emphasize the weight on recent demand. Finally, we would account for overlaps between STAT and OIDD classes instead of limit the model to the classes respective departments.

Applications

Finally, this model is only a prototype with extremely rough estimates and assumptions. Should Wharton ever use this model, **the institution could greatly enhance its functionality and accuracy by inputting the actual enrollment data and updating the covariance method to account for whether or not students take certain classes simultaneously.** The school would also not need to use moving averages to forecast demand anymore as it already possesses the enrollment data. Eliminating forecasts would greatly improve the accuracy to best optimize/maximize enrollment rates.

VI. Appendix A



Wharton Syllabi (Demand Information)

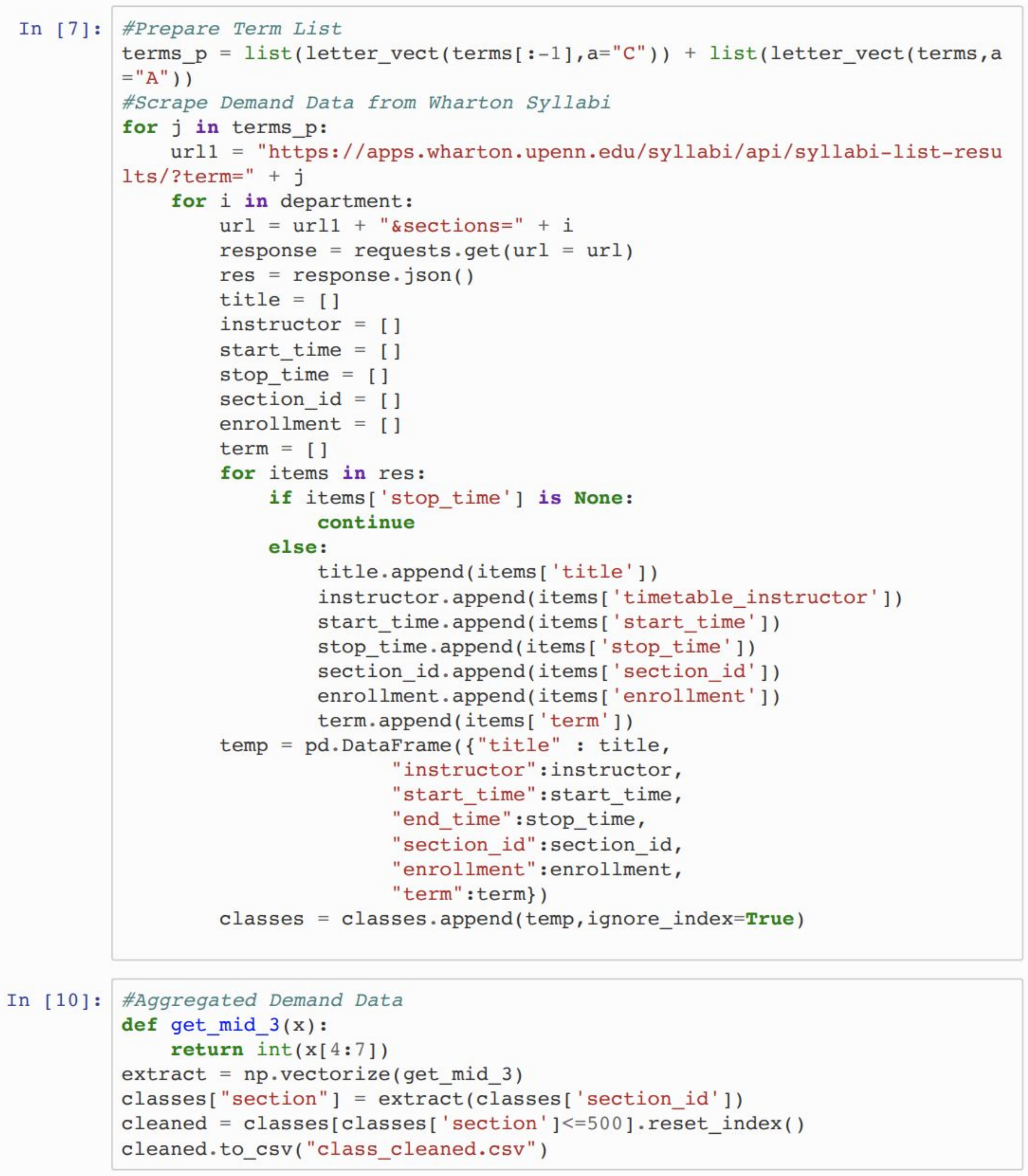
In [1]:
```
import requests
import pandas as pd
import numpy as np
```

In [3]:
```
#Prepare Data Structure
title = []
instructor = []
start_time = []
stop_time = []
section_id = []
enrollment = []
term = []
classes = pd.DataFrame({"title" : title,
              "instructor":instructor,
              "start_time":start_time,
              "end_time":stop_time,
              "section_id":section_id,
              "enrollment":enrollment,
              "term":term})
```

In [4]:
```
#Prepare Department and Year Labels
department = ["STAT","OIDD"]
terms = np.arange(2016,2021)
```

In [5]:
```
#Function to concatenate Letter
def add_letter(x,a):
    return str(x)+a
letter_vect = np.vectorize(add_letter)
```

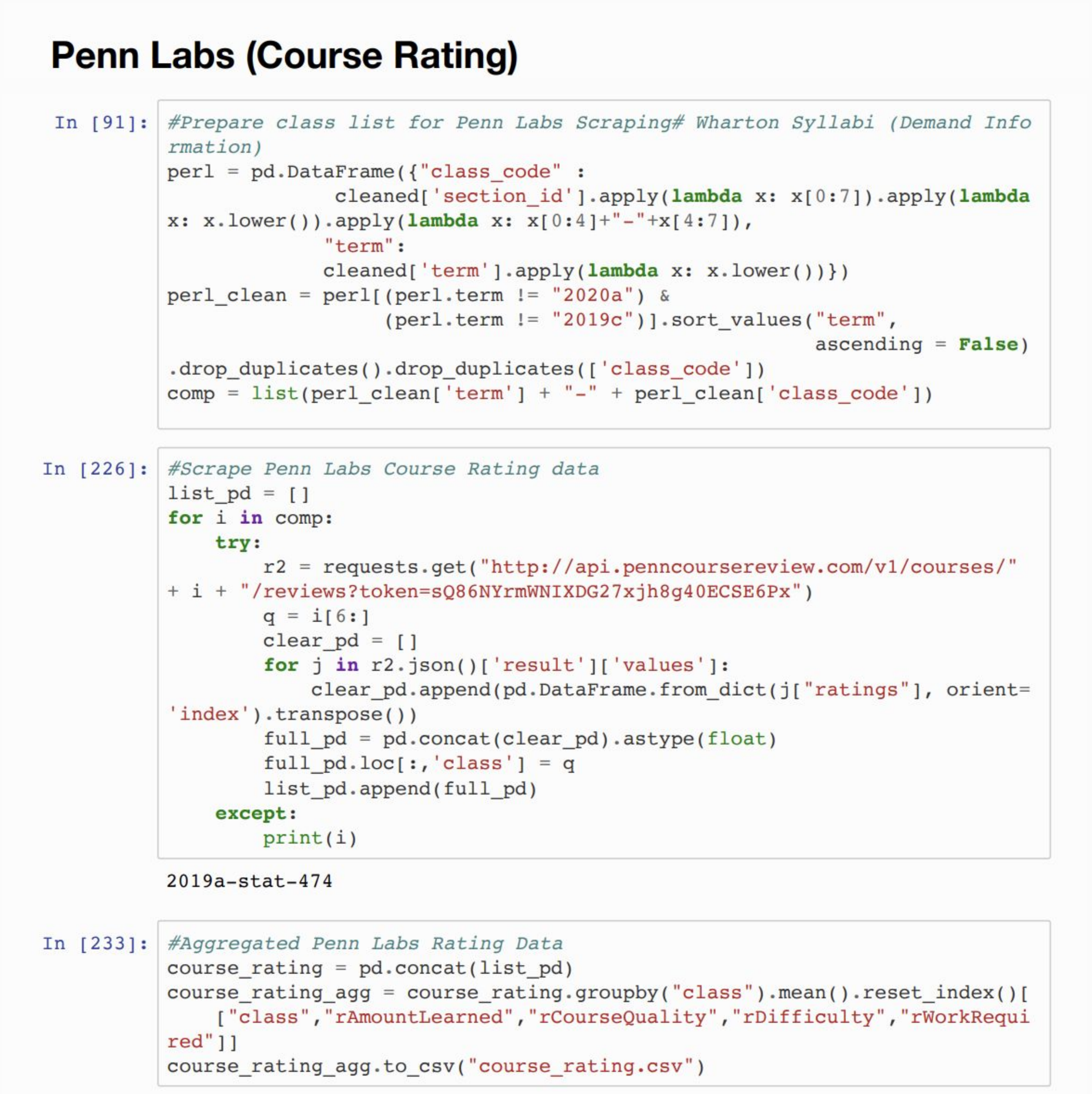
VI. Appendix A



```
In [7]:  #Prepare Term List
         terms_p = list(letter_vect(terms[:-1],a="C")) + list(letter_vect(terms,a
         ="A"))
         #Scrape Demand Data from Wharton Syllabi
         for j in terms_p:
             url1 = "https://apps.wharton.upenn.edu/syllabi/api/syllabi-list-resu
         lts/?term=" + j
             for i in department:
                 url = url1 + "&sections=" + i
                 response = requests.get(url = url)
                 res = response.json()
                 title = []
                 instructor = []
                 start_time = []
                 stop_time = []
                 section_id = []
                 enrollment = []
                 term = []
                 for items in res:
                     if items['stop_time'] is None:
                         continue
                     else:
                         title.append(items['title'])
                         instructor.append(items['timetable_instructor'])
                         start_time.append(items['start_time'])
                         stop_time.append(items['stop_time'])
                         section_id.append(items['section_id'])
                         enrollment.append(items['enrollment'])
                         term.append(items['term'])
                 temp = pd.DataFrame({"title" : title,
                                      "instructor":instructor,
                                      "start_time":start_time,
                                      "end_time":stop_time,
                                      "section_id":section_id,
                                      "enrollment":enrollment,
                                      "term":term})
                 classes = classes.append(temp,ignore_index=True)
```

```
In [10]: #Aggregated Demand Data
         def get_mid_3(x):
             return int(x[4:7])
         extract = np.vectorize(get_mid_3)
         classes["section"] = extract(classes['section_id'])
         cleaned = classes[classes['section']<=500].reset_index()
         cleaned.to_csv("class_cleaned.csv")
```

VI. Appendix B



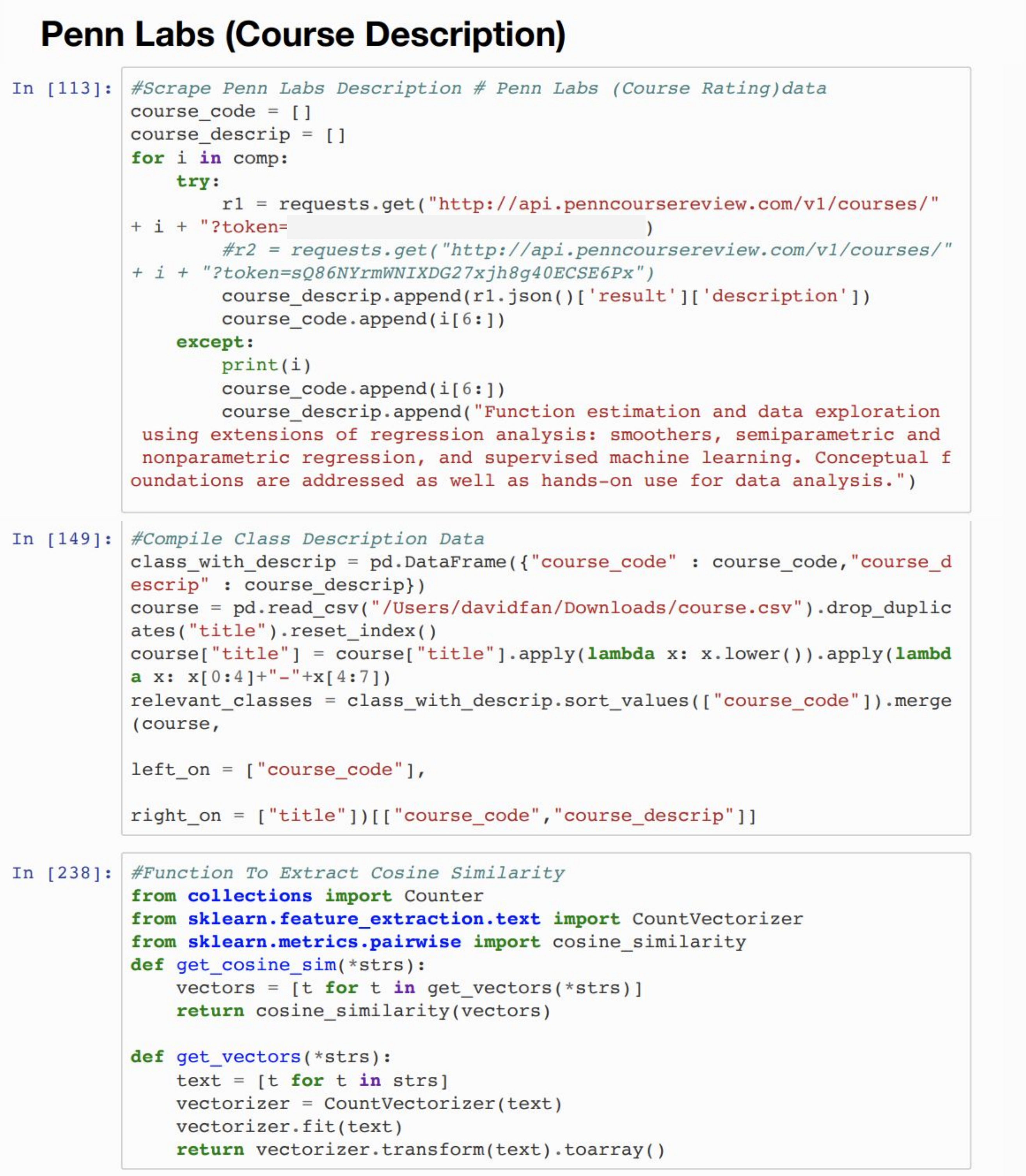
Penn Labs (Course Rating)

```
In [91]: #Prepare class list for Penn Labs Scraping# Wharton Syllabi (Demand Info
         rmation)
         perl = pd.DataFrame({"class_code" :
                       cleaned['section_id'].apply(lambda x: x[0:7]).apply(lambda
         x: x.lower()).apply(lambda x: x[0:4]+"-"+x[4:7]),
                      "term":
                      cleaned['term'].apply(lambda x: x.lower())})
         perl_clean = perl[(perl.term != "2020a") &
                           (perl.term != "2019c")].sort_values("term",
                                                               ascending = False)
         .drop_duplicates().drop_duplicates(['class_code'])
         comp = list(perl_clean['term'] + "-" + perl_clean['class_code'])
```

```
In [226]: #Scrape Penn Labs Course Rating data
          list_pd = []
          for i in comp:
              try:
                  r2 = requests.get("http://api.penncoursereview.com/v1/courses/"
          + i + "/reviews?token=sQ86NYrmWNIXDG27xjh8g40ECSE6Px")
                  q = i[6:]
                  clear_pd = []
                  for j in r2.json()['result']['values']:
                      clear_pd.append(pd.DataFrame.from_dict(j["ratings"], orient=
          'index').transpose())
                  full_pd = pd.concat(clear_pd).astype(float)
                  full_pd.loc[:,'class'] = q
                  list_pd.append(full_pd)
              except:
                  print(i)

          2019a-stat-474
```

```
In [233]: #Aggregated Penn Labs Rating Data
          course_rating = pd.concat(list_pd)
          course_rating_agg = course_rating.groupby("class").mean().reset_index()[
              ["class","rAmountLearned","rCourseQuality","rDifficulty","rWorkRequi
          red"]]
          course_rating_agg.to_csv("course_rating.csv")
```

VI. Appendix C



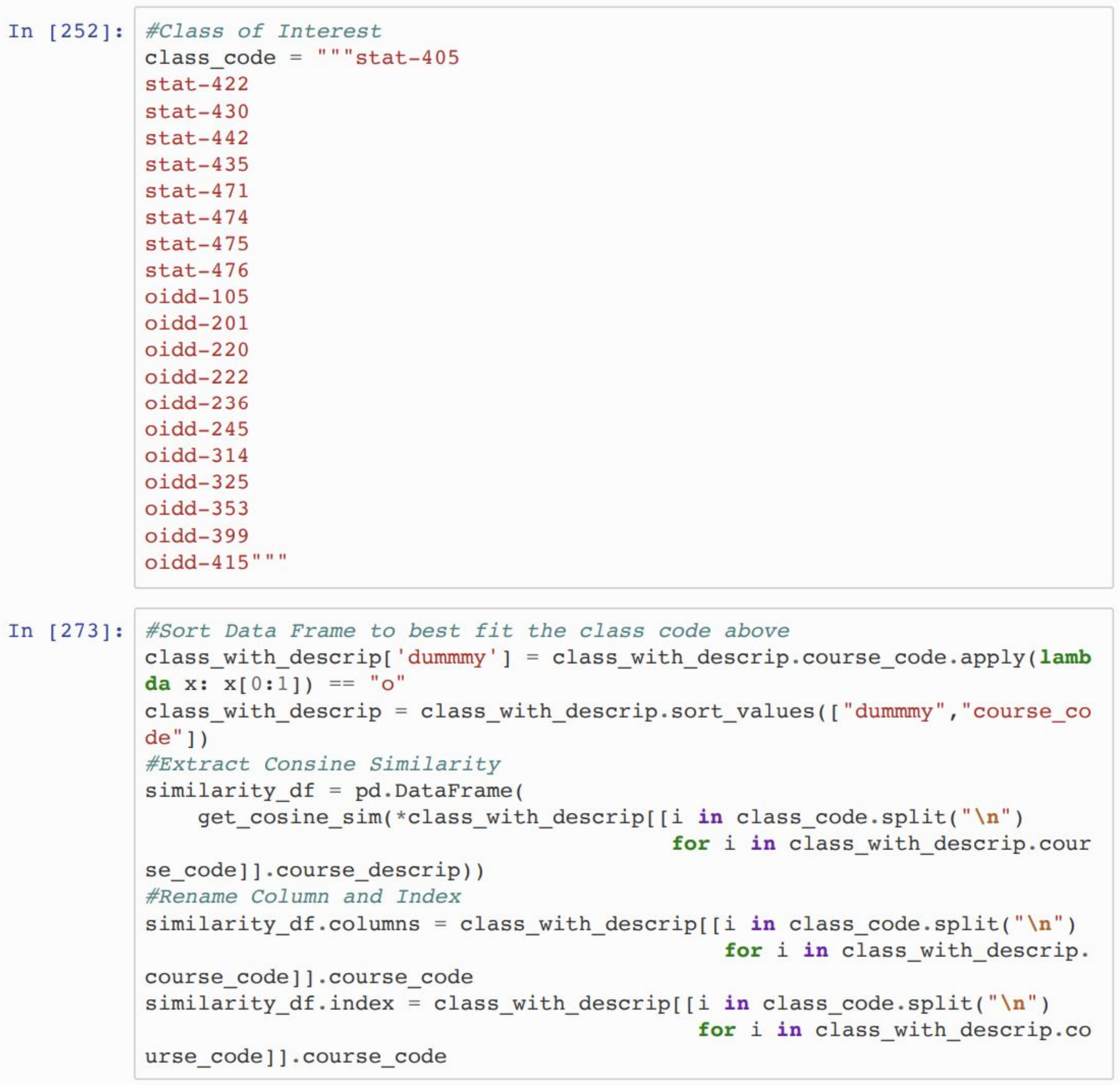
Penn Labs (Course Description)

```
In [113]: #Scrape Penn Labs Description # Penn Labs (Course Rating)data
          course_code = []
          course_descrip = []
          for i in comp:
              try:
                  r1 = requests.get("http://api.penncoursereview.com/v1/courses/"
          + i + "?token=                                   )
                  #r2 = requests.get("http://api.penncoursereview.com/v1/courses/"
          + i + "?token=sQ86NYrmWNIXDG27xjh8g40ECSE6Px")
                  course_descrip.append(r1.json()['result']['description'])
                  course_code.append(i[6:])
              except:
                  print(i)
                  course_code.append(i[6:])
                  course_descrip.append("Function estimation and data exploration
           using extensions of regression analysis: smoothers, semiparametric and
           nonparametric regression, and supervised machine learning. Conceptual f
          oundations are addressed as well as hands-on use for data analysis.")
```

```
In [149]: #Compile Class Description Data
          class_with_descrip = pd.DataFrame({"course_code" : course_code,"course_d
          escrip" : course_descrip})
          course = pd.read_csv("/Users/davidfan/Downloads/course.csv").drop_duplic
          ates("title").reset_index()
          course["title"] = course["title"].apply(lambda x: x.lower()).apply(lambd
          a x: x[0:4]+"-"+x[4:7])
          relevant_classes = class_with_descrip.sort_values(["course_code"]).merge
          (course,

          left_on = ["course_code"],

          right_on = ["title"])[["course_code","course_descrip"]]
```

```
In [238]: #Function To Extract Cosine Similarity
          from collections import Counter
          from sklearn.feature_extraction.text import CountVectorizer
          from sklearn.metrics.pairwise import cosine_similarity
          def get_cosine_sim(*strs):
              vectors = [t for t in get_vectors(*strs)]
              return cosine_similarity(vectors)

          def get_vectors(*strs):
              text = [t for t in strs]
              vectorizer = CountVectorizer(text)
              vectorizer.fit(text)
              return vectorizer.transform(text).toarray()
```

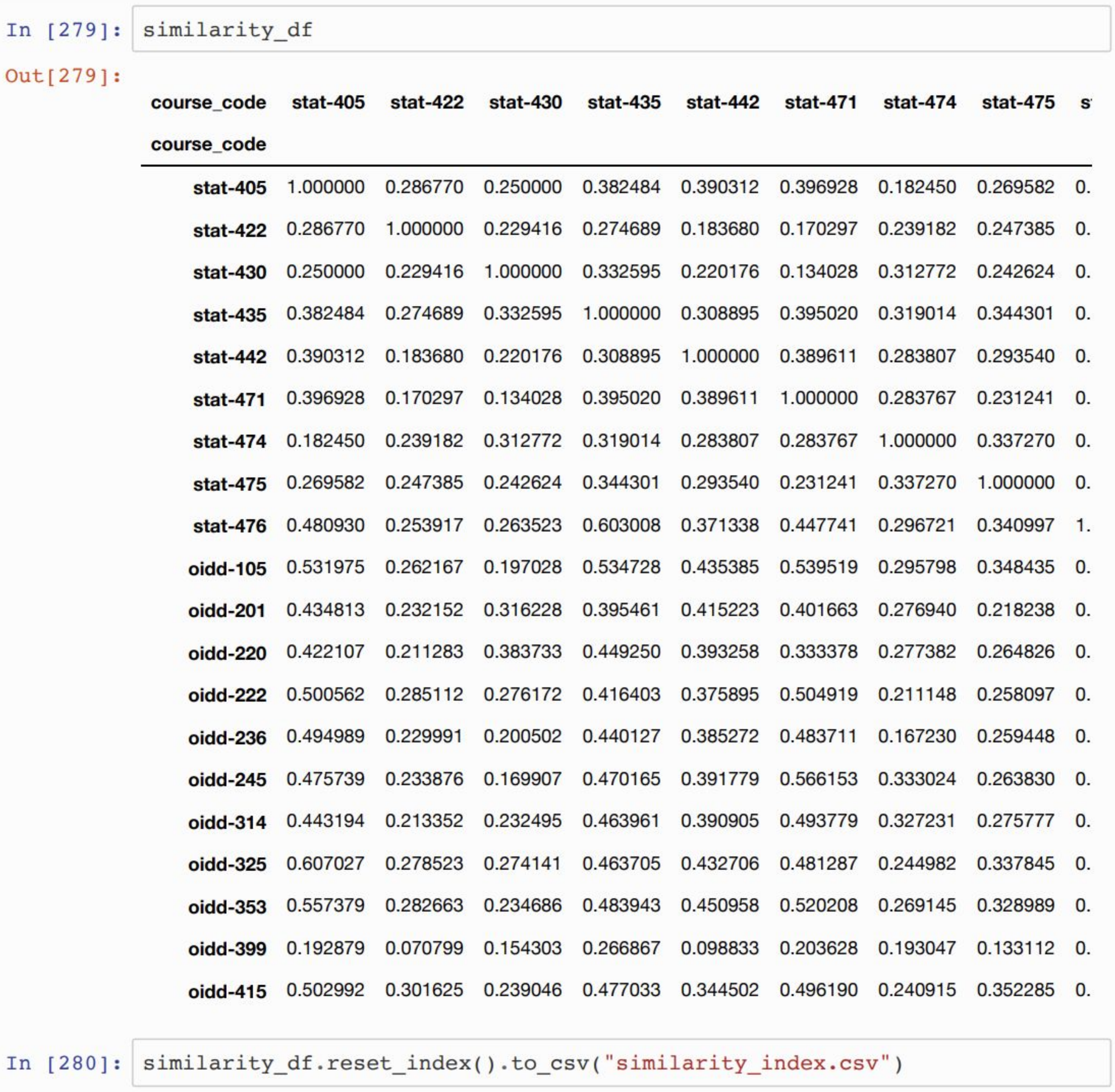
VI. Appendix C



```
In [252]: #Class of Interest
          class_code = """stat-405
          stat-422
          stat-430
          stat-442
          stat-435
          stat-471
          stat-474
          stat-475
          stat-476
          oidd-105
          oidd-201
          oidd-220
          oidd-222
          oidd-236
          oidd-245
          oidd-314
          oidd-325
          oidd-353
          oidd-399
          oidd-415"""
```

```
In [273]: #Sort Data Frame to best fit the class code above
          class_with_descrip['dummmy'] = class_with_descrip.course_code.apply(lamb
          da x: x[0:1]) == "o"
          class_with_descrip = class_with_descrip.sort_values(["dummmy","course_co
          de"])
          #Extract Consine Similarity
          similarity_df = pd.DataFrame(
              get_cosine_sim(*class_with_descrip[[i in class_code.split("\n")
                                                  for i in class_with_descrip.cour
          se_code]].course_descrip))
          #Rename Column and Index
          similarity_df.columns = class_with_descrip[[i in class_code.split("\n")
                                                      for i in class_with_descrip.
          course_code]].course_code
          similarity_df.index = class_with_descrip[[i in class_code.split("\n")
                                                    for i in class_with_descrip.co
          urse_code]].course_code
```

VI. Appendix C



```
In [279]: similarity_df
```

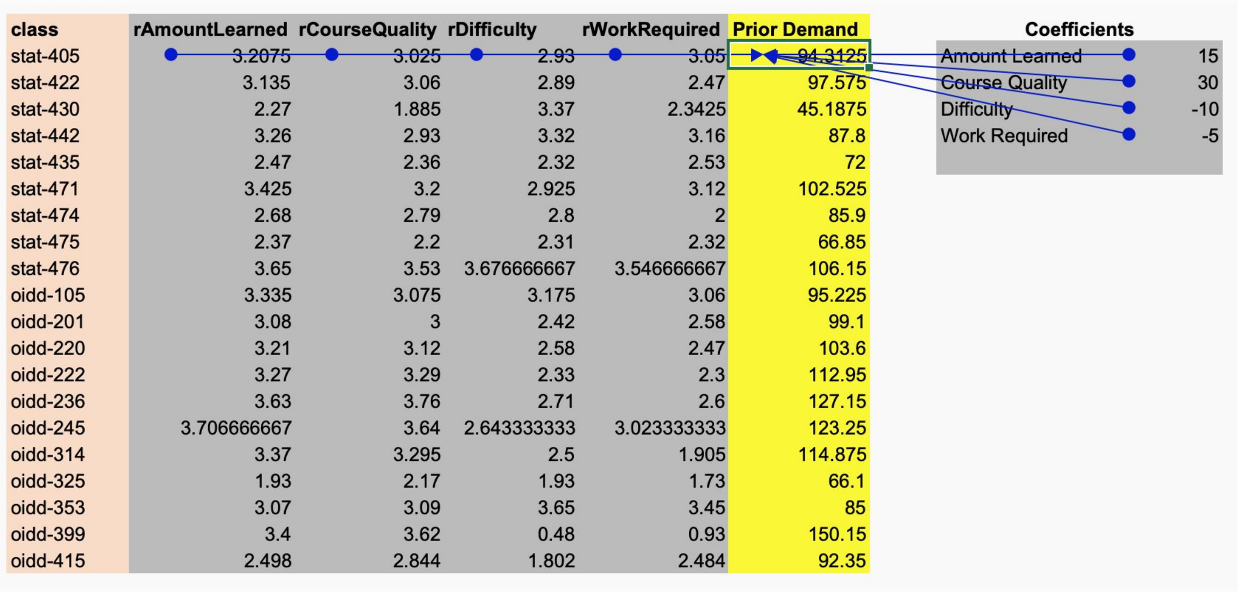
Out[279]:

course_code	stat-405	stat-422	stat-430	stat-435	stat-442	stat-471	stat-474	stat-475	s
course_code									
stat-405	1.000000	0.286770	0.250000	0.382484	0.390312	0.396928	0.182450	0.269582	0.
stat-422	0.286770	1.000000	0.229416	0.274689	0.183680	0.170297	0.239182	0.247385	0.
stat-430	0.250000	0.229416	1.000000	0.332595	0.220176	0.134028	0.312772	0.242624	0.
stat-435	0.382484	0.274689	0.332595	1.000000	0.308895	0.395020	0.319014	0.344301	0.
stat-442	0.390312	0.183680	0.220176	0.308895	1.000000	0.389611	0.283807	0.293540	0.
stat-471	0.396928	0.170297	0.134028	0.395020	0.389611	1.000000	0.283767	0.231241	0.
stat-474	0.182450	0.239182	0.312772	0.319014	0.283807	0.283767	1.000000	0.337270	0.
stat-475	0.269582	0.247385	0.242624	0.344301	0.293540	0.231241	0.337270	1.000000	0.
stat-476	0.480930	0.253917	0.263523	0.603008	0.371338	0.447741	0.296721	0.340997	1.
oidd-105	0.531975	0.262167	0.197028	0.534728	0.435385	0.539519	0.295798	0.348435	0.
oidd-201	0.434813	0.232152	0.316228	0.395461	0.415223	0.401663	0.276940	0.218238	0.
oidd-220	0.422107	0.211283	0.383733	0.449250	0.393258	0.333378	0.277382	0.264826	0.
oidd-222	0.500562	0.285112	0.276172	0.416403	0.375895	0.504919	0.211148	0.258097	0.
oidd-236	0.494989	0.229991	0.200502	0.440127	0.385272	0.483711	0.167230	0.259448	0.
oidd-245	0.475739	0.233876	0.169907	0.470165	0.391779	0.566153	0.333024	0.263830	0.
oidd-314	0.443194	0.213352	0.232495	0.463961	0.390905	0.493779	0.327231	0.275777	0.
oidd-325	0.607027	0.278523	0.274141	0.463705	0.432706	0.481287	0.244982	0.337845	0.
oidd-353	0.557379	0.282663	0.234686	0.483943	0.450958	0.520208	0.269145	0.328989	0.
oidd-399	0.192879	0.070799	0.154303	0.266867	0.098833	0.203628	0.193047	0.133112	0.
oidd-415	0.502992	0.301625	0.239046	0.477033	0.344502	0.496190	0.240915	0.352285	0.

```
In [280]: similarity_df.reset_index().to_csv("similarity_index.csv")
```

VI. Appendix D

Prior Demand Calculations



class	rAmountLearned	rCourseQuality	rDifficulty	rWorkRequired	Prior Demand
stat-405	3.2075	3.025	2.93	3.05	94.3125
stat-422	3.135	3.06	2.89	2.47	97.575
stat-430	2.27	1.885	3.37	2.3425	45.1875
stat-442	3.26	2.93	3.32	3.16	87.8
stat-435	2.47	2.36	2.32	2.53	72
stat-471	3.425	3.2	2.925	3.12	102.525
stat-474	2.68	2.79	2.8	2	85.9
stat-475	2.37	2.2	2.31	2.32	66.85
stat-476	3.65	3.53	3.676666667	3.546666667	106.15
oidd-105	3.335	3.075	3.175	3.06	95.225
oidd-201	3.08	3	2.42	2.58	99.1
oidd-220	3.21	3.12	2.58	2.47	103.6
oidd-222	3.27	3.29	2.33	2.3	112.95
oidd-236	3.63	3.76	2.71	2.6	127.15
oidd-245	3.706666667	3.64	2.643333333	3.023333333	123.25
oidd-314	3.37	3.295	2.5	1.905	114.875
oidd-325	1.93	2.17	1.93	1.73	66.1
oidd-353	3.07	3.09	3.65	3.45	85
oidd-399	3.4	3.62	0.48	0.93	150.15
oidd-415	2.498	2.844	1.802	2.484	92.35

Coefficients	
Amount Learned	15
Course Quality	30
Difficulty	-10
Work Required	-5

VI. Appendix E

Model Results

Part 1: Classroom Allocation

	STAT405	STAT422	STAT430	STAT442	STAT435	STAT471	STAT474	STAT475	STAT476	OIDD105	OIDD201	OIDD220	OIDD222	OIDD236	OIDD245	OIDD314	OIDD325	OIDD353	OIDD399	OIDD415
12-25	0	4	1	1	0	1	2	0	2	0	0	0	0	0	0	0	0	0	1	0
25-40	0	0	1	1	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0
40-50	2	0	1	0	1	1	0	0	0	0	1	0	1	0	0	0	0	0	0	0
50-60	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0
60-78	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	1	1
78-100	0	0	0	0	0	0	0	0	0	1	0	0	0	1	1	0	0	0	0	0
123-150	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
150-299	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Differences	1.00625	30.8275	2.34375	5.76	9.5	5.7425	11.63	8.945	8.454993	27.9325	7.67	3.72	7.015	36.555	5.525	0.8375	5.17	3.3	1.955	8.995
Objective	192.885																			

Part 2a: Class Scheduling (STAT Iteration)

	STAT405	STAT422	STAT430	STAT442	STAT435	STAT471	STAT474	STAT475	STAT476
STAT405	0	0	0	0	0	0	0	1	0
STAT422		0	0	0	1	0	0	0	0
STAT430			0	0	0	1	0	0	0
STAT442				0	0	0	0	0	0
STAT435					0	0	0	0	0
STAT471						0	0	0	0
STAT474							0	0	0
STAT475								0	0
STAT476									0

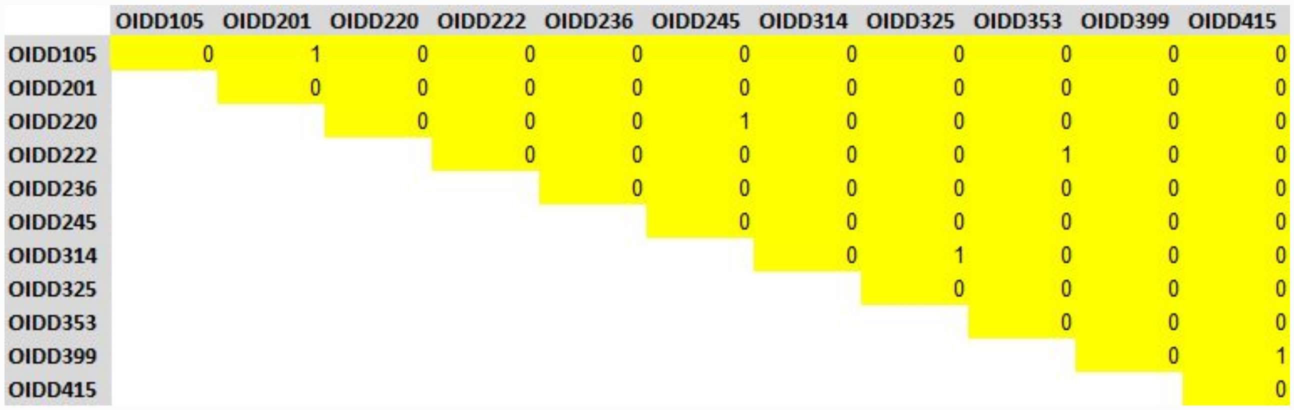
	STAT405	STAT422	STAT430	STAT442	STAT435	STAT471	STAT474	STAT475	STAT476
Time1	0	0	0	0	0	0	0	0	1
Time2	0	0	0	0	0	0	1	0	0
Time3	0	0	1	0	0	1	0	0	0
Time4	0	1	0	0	1	0	0	0	0
Time5	1	0	0	0	0	0	0	1	0
Time6	0	0	0	1	0	0	0	0	0

Objective	0.587289

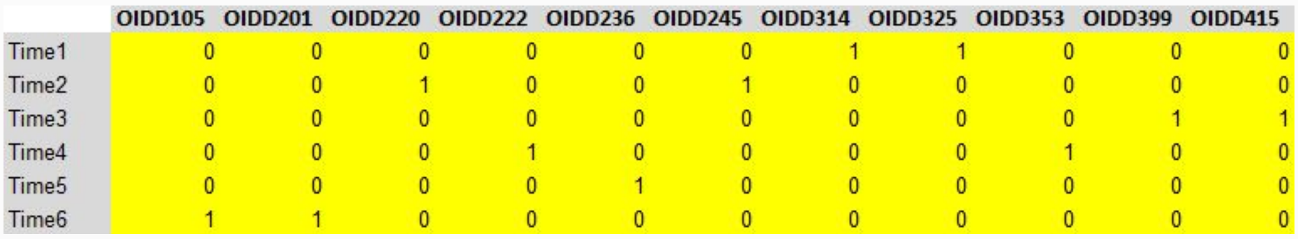
VI. Appendix E

Model Results

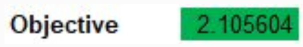
Part 2b: Class Scheduling (OIDD Iteration)



	OIDD105	OIDD201	OIDD220	OIDD222	OIDD236	OIDD245	OIDD314	OIDD325	OIDD353	OIDD399	OIDD415
OIDD105	0	1	0	0	0	0	0	0	0	0	0
OIDD201		0	0	0	0	0	0	0	0	0	0
OIDD220			0	0	0	1	0	0	0	0	0
OIDD222				0	0	0	0	0	1	0	0
OIDD236					0	0	0	0	0	0	0
OIDD245						0	0	0	0	0	0
OIDD314							0	1	0	0	0
OIDD325								0	0	0	0
OIDD353									0	0	0
OIDD399										0	1
OIDD415											0



	OIDD105	OIDD201	OIDD220	OIDD222	OIDD236	OIDD245	OIDD314	OIDD325	OIDD353	OIDD399	OIDD415
Time1	0	0	0	0	0	0	1	1	0	0	0
Time2	0	0	1	0	0	1	0	0	0	0	0
Time3	0	0	0	0	0	0	0	0	0	1	1
Time4	0	0	0	1	0	0	0	0	1	0	0
Time5	0	0	0	0	1	0	0	0	0	0	0
Time6	1	1	0	0	0	0	0	0	0	0	0



Objective	2.105604