**A blue and white hexagon with white text

Description automatically generated**

***Final Project in Data Analytics***

**Course Popularity on Udemy**

**Weilin Chen**

**Table of Contents**

1. **Use Case 3**
2. **Principal Findings of the Report 4**
3. **Data Collection 5**
4. **Data Cleaning and Data Preparation 7**
5. **The Dataset 10**
6. **Database Type 12**
7. **EDA and Data Visualization 12**

**Use Case**

The rise of MOOC (Massive Open Online Course) platforms has taken the world of digital learning by storm. It has transformed the face of education and will continue to do so in the foreseeable future as technology evolves. MOOC platforms have become an indispensable tool for not only individuals seeking continuous learning but also for companies seeking to update the skillset of their workforce.

My project focuses on one important KPI for the digital learning sector: the popularity of a course as measured by the number of subscribers. This is an essential metric for MOOC platforms seeking to evaluate their course offerings, for individuals choosing courses on these platforms, and for future educators trying to understand the needs of potential students. Focusing on Udemy, a leading MOOC platform, my project includes collecting data directly from Udemy with API calls, preparing and storing the data, and performing exploratory data analysis to identity key features that influence course popularity. Furthermore, both the data and the data analysis will serve as important stepping stones for the machine learning part of my project which aims to predict the popularity of planned or early stage courses.

**Principal Findings of the Report**

The main findings of this report are the following:

1. The top 3 course categories in terms of the number of subscribers are Development, Marketing, and Office Productivity. When it comes to rating, which is an essential metric due to its importance for the decision making of users in course selection, there is still much room for improvement: nearly half of the categories have an average rating less than 4 and no category has an average rating above 4.25. Moreover, the category IT & Software, which ranks high in popularity, performs particularly poorly in rating. This suggests that improving user feedback is a potential strategy of self-differentiation among competitors, both for IT & Software courses and in general.
2. The top 3 course subcategories in terms of the number of subscribers are Influence, Seach Engine Optimization, and Programming Languages. The average ratings are mostly good for the top 15 most popular course subcategories. Subcategories like Growth Hacking and Programming Languages, however, have high popularity and low ratings. Again, improving user feedback can be a potential strategy of self-differentiation in these two subcategories.
3. The data suggests that the courses with high potential are likely to lie in the fields of Programming Languages and Generative AI, thus indicating future trends for the field of digital learning.
4. Price elasticity is likely high in the digital learning landscape. While balancing price and quality remains an important task, low price tolerance suggests that strategic pricing remains important for staying competitive in the market.
5. The popular courses generally have a high ratio of good reviews (above 4). This highlights the importance of user feedback for staying competitive.

**Data Collection**

**Data Sources and API use**

The data used for this project are collected directly from Udemy with the use of the Udemy Affiliate API. It is a REST API that takes HTTP requests and responds to those requests in the JSON format.[[1]](#footnote-1) The Udemy API allows for four types of API requests, for four types of information: a list of courses and information about them, course reviews (including ratings and sometimes comments by the reviewer), course curriculum, and course details. The course list API request allows for obtaining information about different courses in bulk whereas the other three types of API requests only allow for calls based on a specific course.

For this project I used the course list request, the course curriculum request, and the course reviews request. Since the information obtainable from the course details request overlaps with that of the course list request, I did not need to use the course details request for this project.

**Methods and Process**

My main collection tool was the course list API request which returns a list of courses and information about them. I adjusted the settings of the API call to maximize the amount of information (the number of courses) in the response, as well as ask for only the types of information needed (the future columns in my table). I created two Python functions to facilitate the process: one function for transforming the response of each API call from the JSON format to a Dataframe format with the specified columns, another function (which incorporates the first function) for making a series of API calls and returning the information in the dataframe format. I then proceeded to store the raw data in MySQL.

My data collection with the course list request proceeded in two phases. I first collected as many courses as possible without any specifications about the courses, then I collected courses by going through the list of primary categories that Udemy uses to label the courses. In total, 39,512 courses were collected.

With regard to the other two types of API requests, an important technical constraint was imposed by the fact that, unlike the course list request, which allows for obtaining information of a large quantity of courses with each request, the curriculum and course reviews requests only return information about one specific course. Given the time constraint and the necessity of proper API handling, it was therefore not possible to obtain reviews and curricula for all the courses retrieved with the course list request.

In the end I collected reviews and curricula for two specific groups of courses to enable comparative analyses later on. The first group is comprised of the 100 courses with the highest numbers of subscribers (ranging from about 180,000 to 1,800,000) in the data already collected. The second group is comprised of the first 100 courses with more than 10,000 subscribers. The idea is to compare the most popular courses with courses which are still popular and more moderately so (based on the distribution shown below of the numbers of subscribers across all courses, above 10,000 is still much above the 75 percentile). The two groups are stored together in the reviews and curricula tables.

count 3.951200e+04

mean 5.577677e+03

std 2.490657e+04

min 0.000000e+00

25% 7.800000e+01

50% 6.070000e+02

75% 3.179000e+03

max 1.837007e+06

Name: num\_subscribers, dtype: float64

In the Course Reviews table, the number of reviews for the first group (100 000) is much larger than the number of reviews for the second group (9 190). This is due to the fact that the first group has many more subscribers and hence many more reviews.

Similar to the collection process with the course list request, I created two functions for using the reviews and the curriculum requests. I then stored the raw data in MySQL.

**Data Cleaning and Data Preparation**

The three tables are cleaned and prepared in turn:

**Course list**

* Dropped duplicated rows
* Dropped rows with null values
* Dropped the “discount” and “discount\_price” columns which are not useful
* Changed the data type of the column “published\_time” to datetime
* For the “price” column, removed currency sign, changed the value “free” to “0”, and changed the datatype to float
* Changed the data type of “is\_practice\_test\_course” to Boolean.
* Removed HTML tags from the “description column
* Changed blank values in the “level” column to “All Levels”
* Added “months\_since\_created” column (number of months since the course was published)
* Created “num\_subscriberes\_monthly” column (“num\_subscribers” divided by “months\_since\_created”)

**Course reviews**

* Changed the datatype of “created\_time” column to datetime

**Curricula**

* No changes needed.

I also created a separate instructors table with a unique ID assigned to each instructor. I then

merged this table with the Course List table and dropped the “instructor” and “instructor\_job\_title” columns from the Course List table. In the end, the Course List table contains only the “instructor\_id” for information related to instructors. When it comes to database design, it is important to strike a balance between normalization and denormalization. While the data from the course list API request comes in a denormalized form, I believe that partly normalizing the data would increase the overall usability and flexibility of the entire dataset, given the fact that the same instructor can teach multiple classes and that more information about instructors can be easily added later on.

**Data Storage and Creation of the Entity Relationship Diagram (ERD)**

The four tables are then sent to MySQL for storage. I then created primary and foreign key constraints with the commands shown below, followed by the ERD:

alter table course\_list add primary key (course\_id);

alter table instructors add primary key (instructor\_id);

alter table course\_list add foreign key (instructor\_id) references instructors(instructor\_id);

alter table curricula add foreign key (course\_id) references course\_list(course\_id);

alter table reviews add primary key (review\_id);

alter table reviews add foreign key (course\_id) references course\_list(course\_id);

A computer screen shot of a diagram

Description automatically generated

**The Dataset**

The final dataset contains the following tables and columns:

Course List

(39512 rows)

1. course\_id Unique ID of the course
2. title Title of the course
3. published\_time Time when the course was published
4. num\_subscribers Number of subscribers of the course
5. price Price of the course
6. avg\_rating Average rating of the course
7. avg\_recent\_rating Average recent rating of the course (period unknown)
8. num\_reviews Number of reviews given for the course
9. num\_lectures Number of lectures in the course
10. num\_quizzes Number of quizzes in the course
11. is\_practice\_test\_course Whether the course is a test preparation course
12. language Language of the course
13. content\_length Length of course content
14. primary\_category The primary category to which the course belongs
15. primary\_subcategory The primary subcategory to which the course belongs
16. level Level of the students targeted by the course
17. headline Headline of the course
18. description Brief description of the course
19. url URL of the course webpage
20. months\_since\_created Number of months since the course was created
21. num\_subscribers\_monthly Number of subscribers per month
22. instructor\_id Unique ID assigned to each course instructor

Reviews

(109190 rows)

1. review\_id Unique ID of the review
2. rating Rating given by the review
3. comment Comment in the review
4. created\_time Time when the review was made
5. user The user who gave the review
6. course\_id ID of the course reviewed

Curricula

(200 rows)

1. type\_session\_1 Session 1 type (a chapter or a lecture belonging to a

chapter)

1. title\_session\_1 Session 1 title
2. type\_session\_2 Session 2 type
3. title\_session\_2 Session 2 title
4. type\_session\_3 Session 3 type
5. title\_session\_3 Session 3 title
6. type\_session\_4 Session 4 type
7. title\_session\_4 Session 4 title
8. course\_id ID of the course in question

Instructors

(19588 rows)

1. instructor\_id Unique ID assigned to each course instructor
2. instructor Name of the instructor
3. instructor\_job\_title Job title of the instructor

**Database Type**

The main reason for choosing MySQL rather than a NoSQL database for this project is the fact that the data collected is relational and needs to be stored in a tabular framework to facilitate the kind of analysis needed for this project. The moderate size of my data also means that it can be securely stored in MySQL.

**Exploratory Data Analysis and Data Visualization**

Since the metric that my project focuses on is the number of subscribers for a course, I began by plotting a histogram of the metric across all the courses. I plotted the chart on the logarithmic scale due to the highly uneven distribution. The result is a right skewed histogram, which is expected given the statistical summary chart on page 2 showing that the median is less than the mean.

A graph of a number of subscribers

Description automatically generated

**Number of Subscribers by Course Subject**

One of the most important considerations, whether from the point of view of a MOOC platform like Udemy or someone thinking about starting an online course, concerns the themes of courses. Udemy divides its course into 12 primary categories each of which is further divided into subcategories. I first created a table showing the primary categories ranked by the average number of subscribers, with the following SQL query:

p\_categories = pd.read\_sql("select primary\_category ,avg(num\_subscribers) as average\_num\_subscribers, avg(avg\_rating) as average\_rating from udemy\_final.course\_list group by primary\_category order by avg(num\_subscribers) desc", engine)

A table of numbers and text

Description automatically generated

Then I sorted the table according to average rating with the following command:

p\_categories\_1 = p\_categories.sort\_values(by='average\_rating', ascending=False)

A screenshot of a table

Description automatically generated

I then plotted the two tables to create two charts placed side by side (note: the scale of the x-axis of the chart on the right starts at 3.00):

A graph showing a number of people

Description automatically generated with medium confidence

It is interesting to note, from the visualizations above, that no category has yet attained an average rating of more than 4.25. Moreover, IT & Software, which is in the top half among the categories in terms of the number of subscribers, is almost at the bottom of the list when it comes to average rating. This observation suggests that a possible strategy for a course in IT & Software to differentiate itself from its potential competitors is to try to achieve a rating of more than 4.

I then move onto the primary subcategories. Since there are more than 100 subcategories in total, I selected only the top 15 in terms of the number of subscribers which I then sort by as the number of subscribers as well as by average rating:

p\_subcategories = pd.read\_sql("select primary\_subcategory ,round(avg(num\_subscribers)) as average\_num\_subscribers, avg(avg\_rating) as average\_rating from udemy\_final.course\_list group by primary\_subcategory order by avg(num\_subscribers) desc limit 15", engine)

A table with numbers and words

Description automatically generated

Then:

p\_subcategories\_1 = p\_subcategories.sort\_values(by='average\_rating', ascending=False)

A table of numbers and words

Description automatically generated with medium confidence

A graph showing the amount of information

Description automatically generated with medium confidence

Four subcategories (Influence, Paid Advertising, Photography Tools, Search Engine Optimization) have ratings above 4.25. Growth Hacking, and especially Programming Languages, differ sharply across the two metrics (better than average popularity with lower than average rating). Again, this could inform decision making about possible ways for a course in these two subcategories to differentiate itself.

**Identification of Courses with High Potential**

An important question faced by any actor in the digital education sector concerns emerging trends, such as what courses are likely to be successful in the future. Judgments concerning what courses have high potential are crucial for the making of decisions such as what courses to start or which courses to target for marketing efforts. I took the following steps to identity courses with high potential:

I first selected the top 15 courses in terms of the number of subscribers:

total\_subscribers\_rank = pd.read\_sql("select course\_id, title ,num\_subscribers, num\_subscribers\_monthly, level, avg\_rating from udemy\_final.course\_list order by num\_subscribers desc limit 15", engine)



Then I selected, from a pool of courses which are at least 3 months old (to rule out possible ephemerality), the top 15 courses in terms of the average number of subscribers by month (num\_subscribers divided by months\_since\_created):

monthly\_subscribers\_rank = pd.read\_sql("select course\_id, title ,num\_subscribers\_monthly, months\_since\_created, num\_subscribers, level, avg\_rating, price from udemy\_final.course\_list where months\_since\_created >= 3 order by num\_subscribers\_monthly desc limit 15", engine)

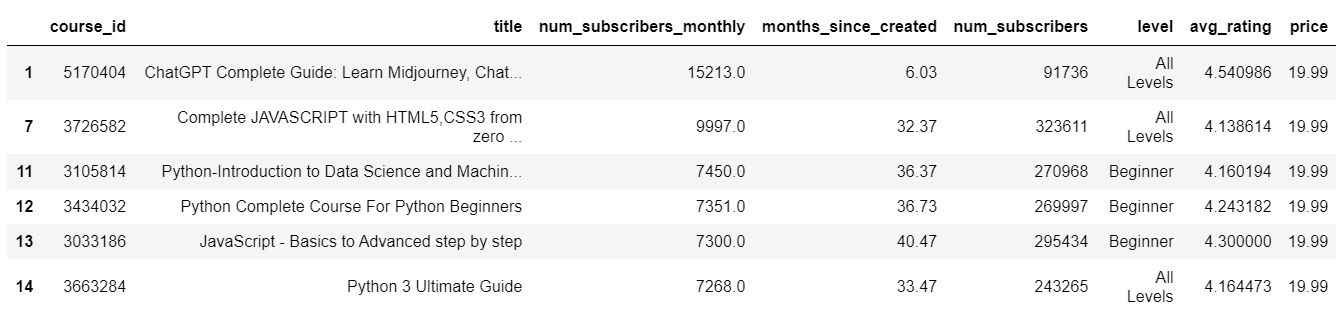
A screenshot of a computer

Description automatically generated

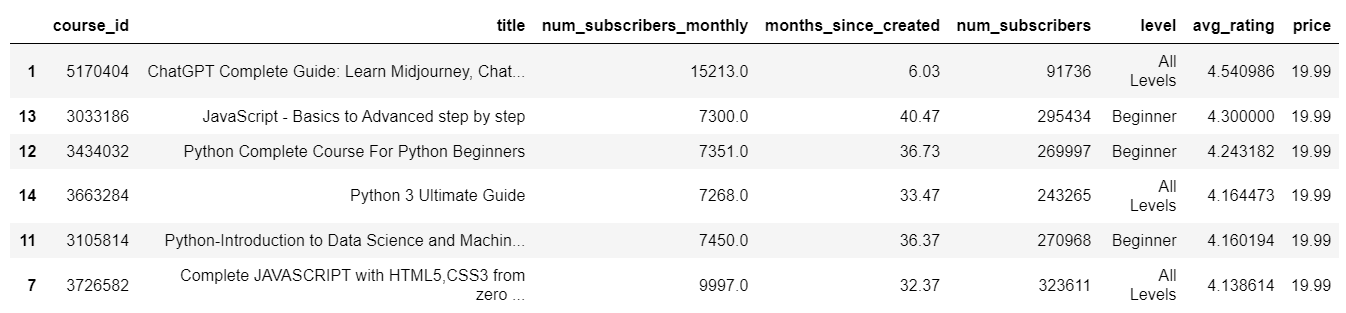
Then I selected those courses which are in the second table but which are *not* in the first table, namely courses which have a very high number of subscribers per month and which will likely reach the top ranks in terms of total number of subscribers given more time:

ids\_potential = set(monthly\_subscribers\_rank['course\_id'])-(set(monthly\_subscribers\_rank['course\_id'])&set(total\_subscribers\_rank['course\_id']))

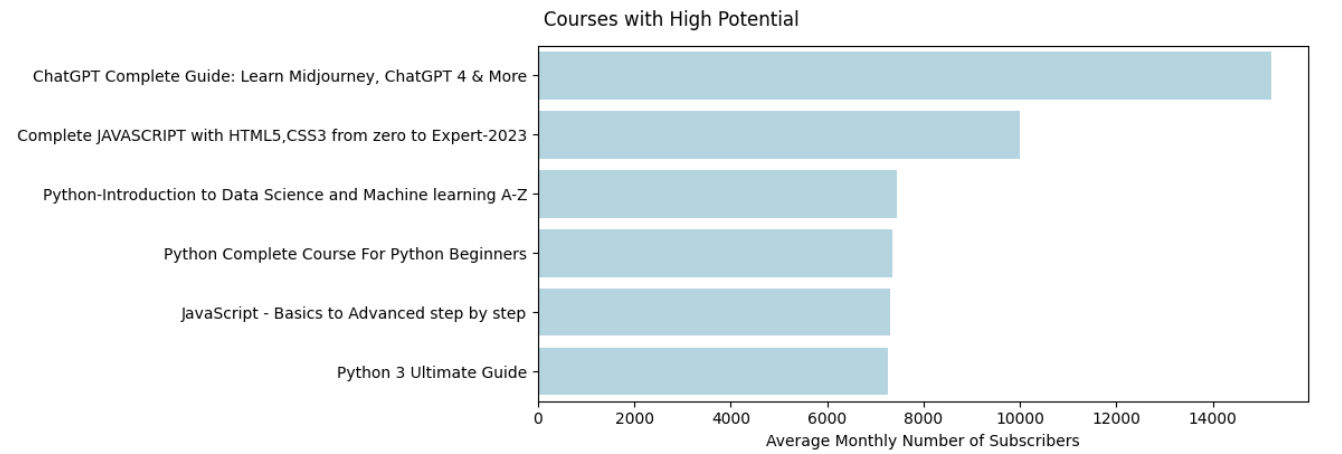
courses\_with\_potential = monthly\_subscribers\_rank[monthly\_subscribers\_rank['course\_id'].isin(ids\_potential)]

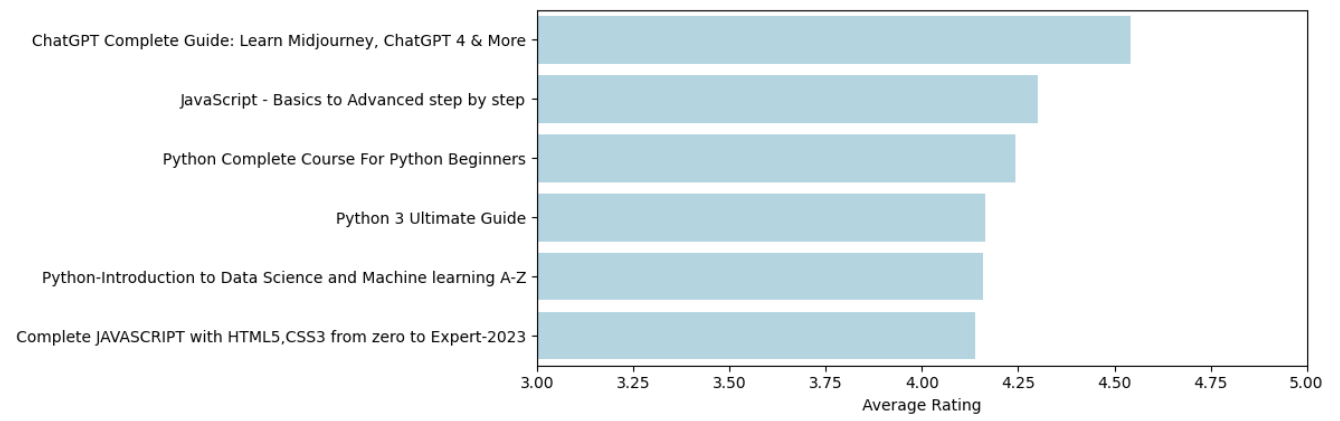


And ranked by average rating:



I then plotted both tables:



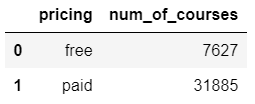


It is interesting to note that all the popular courses, whether by total number of subscribers or by the numbers of subscribers per month, have low entry requirements since they target students of either the beginner level or of all levels. Moreover, none of the courses with potential I identified are free though they are on the lowest price level (19.99 euros) in the dataset.

**Pricing Strategy**

Another important consideration is pricing. To find out the distribution of paid and free courses:

paid\_or\_free = pd.read\_sql("select case when price > 0 then 'paid' else 'free' end as pricing, count(1) as num\_of\_courses from udemy\_final.course\_list group by pricing", engine)

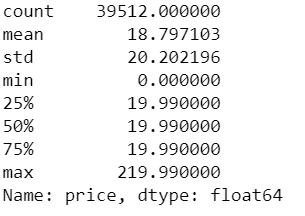


Close to 20% of the courses are free. I then made a table of the top 20 paid courses in terms of the number of subscribers and their prices:

top\_paid\_courses = pd.read\_sql("select course\_id, title ,num\_subscribers, price from udemy\_final.course\_list where price > 0 order by num\_subscribers desc limit 20", engine)



Here is the statistical summary of prices across all courses:



As mentioned earlier, the lowest price among paid courses in the dataset is 19.99 euros. The fact that the average price is below that suggests that price tolerance is likely to be low among Udemy users. This can be seen from the fact that more than half of the most popular paid courses on the table on the previous page are in the low price range (less than 20). The potentially high price elasticity should make one hesitate before choosing to raise the price of a course. That being said, the table also shows that it is possible to charge a moderately higher price (such as above 25) and have still have a high number of subscribers.

**Comparative Analysis of Reviews**

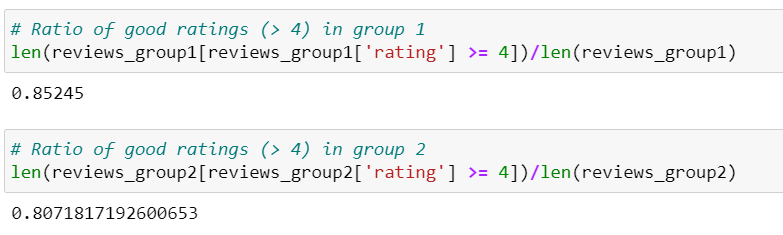
Aside from the information about ratings in the Course List table, I also have rating information the Course Reviews table with courses belonging to two groups: the most popular courses and the first 100 courses with more than 10 000 subscribers. I would like to compare the proportions of good ratings of the two groups. I first obtain from the Course Reviews table in SQL the reviews for both groups before dividing them into two separate dataframes. Then I calculate the proportion of ratings above 4 in each group:

reviews\_both = pd.read\_sql('''select r.course\_id, cl.title ,cl.num\_subscribers, r.rating from udemy\_final.reviews r

inner join udemy\_final.course\_list cl

on r.course\_id = cl.course\_id

order by num\_subscribers desc''', engine)



Both groups have a high percentage of good ratings in the most recent reviews on the Udemy platform. The first group has a higher ratio of good recent reviews, which is interesting given the fact the first group also has a much higher number of reviews (100 000 for the first group and 9190 for second).

1. https://www.udemy.com/developers/affiliate/ [↑](#footnote-ref-1)