Hotel Recommendation System. Final Report.

Jonas Cuadrado

1. *Defining the problem*

According to Tripadvisor on their yearly travel report [1], most travelers spend most of the time allocated to book a trip deciding on the flights or lodging, both for vacation or work. While the flight seems to be decided mostly on cost and, in some cases, airline preferences; lodging factors in more parameters, including other people's ratings, and location within the destination area. So far, hotel search engines do not seem to personalize the results based on the user's preferences. Or, at least, they don't encourage users to rate their stays.

In this project I will create a recommendation system for hotels where the users will be encouraged explicitly to rate their stays to improve the recommendations for their next experience. Recommendation systems are quite common in our daily life, including music, tv, or social networks, with very successful outcomes.

2. *Clients*

Anyone who travels with some frequency can benefit from this project. In reality, the client would be an existing hotel search engine (Hotels.com, Trivago, TripAdvisor, …) who would implement the proposed work on their platform.

3. *Data*

To develop a recommendation system we need a set of users who rate items, and both parts are of upmost importance: the ratings alone do not give enough information. Unfortunately, no user-rating database is readily available from any public website (most likely due to privacy reasons), so I have simulated one based on academic research.

A dataset containing hotels from different cities in the US, New York, San Francisco, Chicago, and Las Vegas, was obtained from [2]. It contains the average rating for different criteria such as location, room cleanliness, friendliness of staff, etc., as well as an overall rating. These ratings were obtained from analyzing comments on the hotels from the TripAdvisor website on 2011.

To complete this dataset, I used a pythonic scraper (selenium) to obtain the star rating and price of each hotel for a given date, defaulted by the search engine (one month after the search date for Google). Due to the age of the dataset, some hotels are permanently closed, some others do not have their price available easily, so some blanks were stored as well. The total number of hotels is slightly under 1000. To analyze the data and look at correlations reported on the next section, about 40% of the hotels without price or star rating were dropped.

According to [3], the most important attribute to decide on what hotel to book is price. [4] and [5] evidence that other parameters can play a role, but at the end the price is always the major player. From [6] and [7] we can obtain an average amount spent in hotels as a function of household income and age, and use it to generate a virtual user of some hotels. I propose a rating system for those virtual users that mostly depends on the existing mean rating.

4. Alternative datasets

The process of generating users is essentially the only (legit) option we have to generate ratings. There are other hotel lists on which to build:

- Kaggle has a large dataset containing hotel reservations, but it doesn't have ratings per se. It may contain comments and reviews from which to extract a numeric rating.
- The US fire department has a list of hotels on which federal employees can stay, with information about their safety standards. No ratings are available whatsoever.

5. Initial findings

Regarding the hotels, we observe an evident correlation between hotel price and mean user rating, as well as the number of stars the hotel has. In the case of as Vegas, the correlation is weak because hotels make more profit from gambling than the stay itself, but there is enough evidence to accept the correlations. We also observe that the dependence is not necessarily linear, but the lack of points limits how well we can assess its non-linearity.

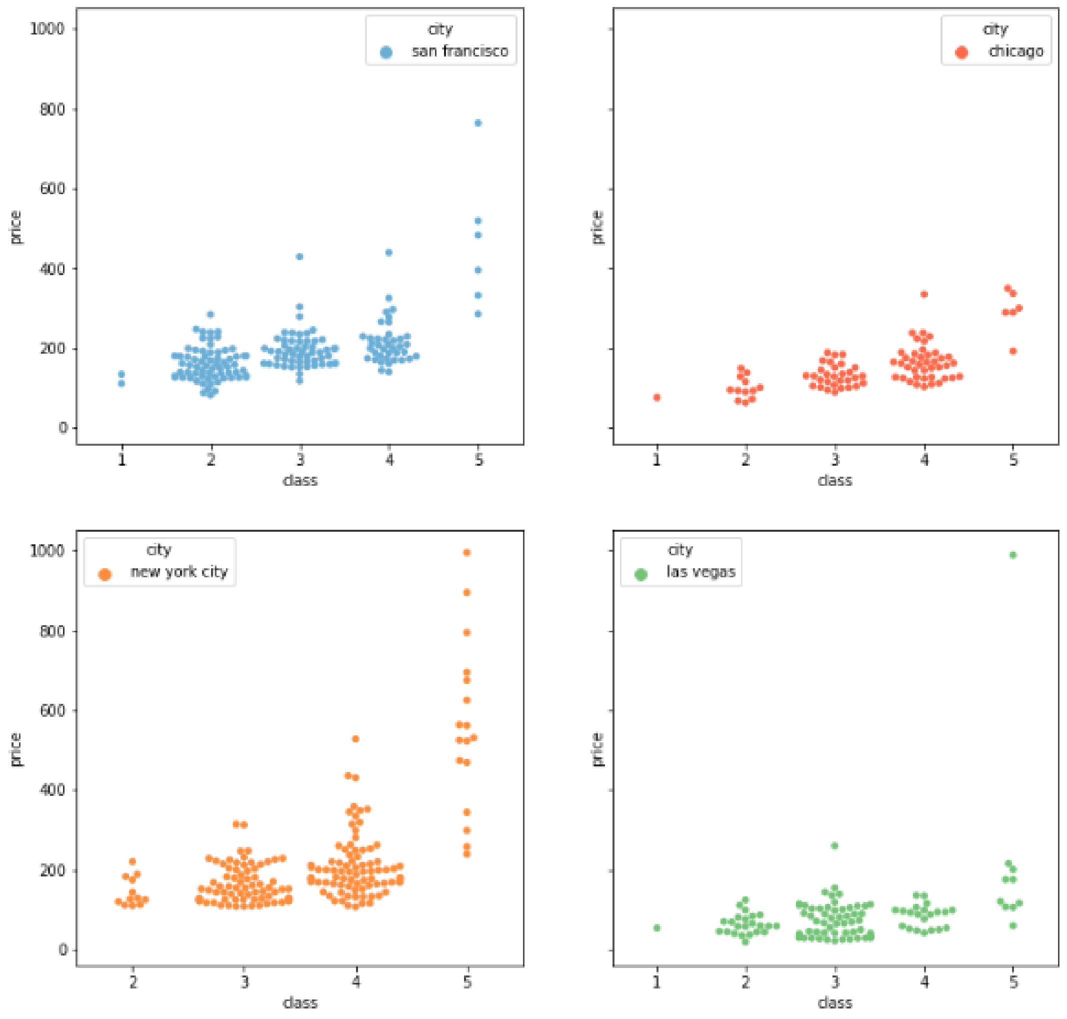


Figure 1: Price comparing different star classes.

Regarding the location, some zip codes have a large variability on the users' rating. There are hotels with low values for location in the same zip-area than well-located hotels, which essentially tells us that the area on which a hotel is located needs to be defined to a level smaller than zip-code size.

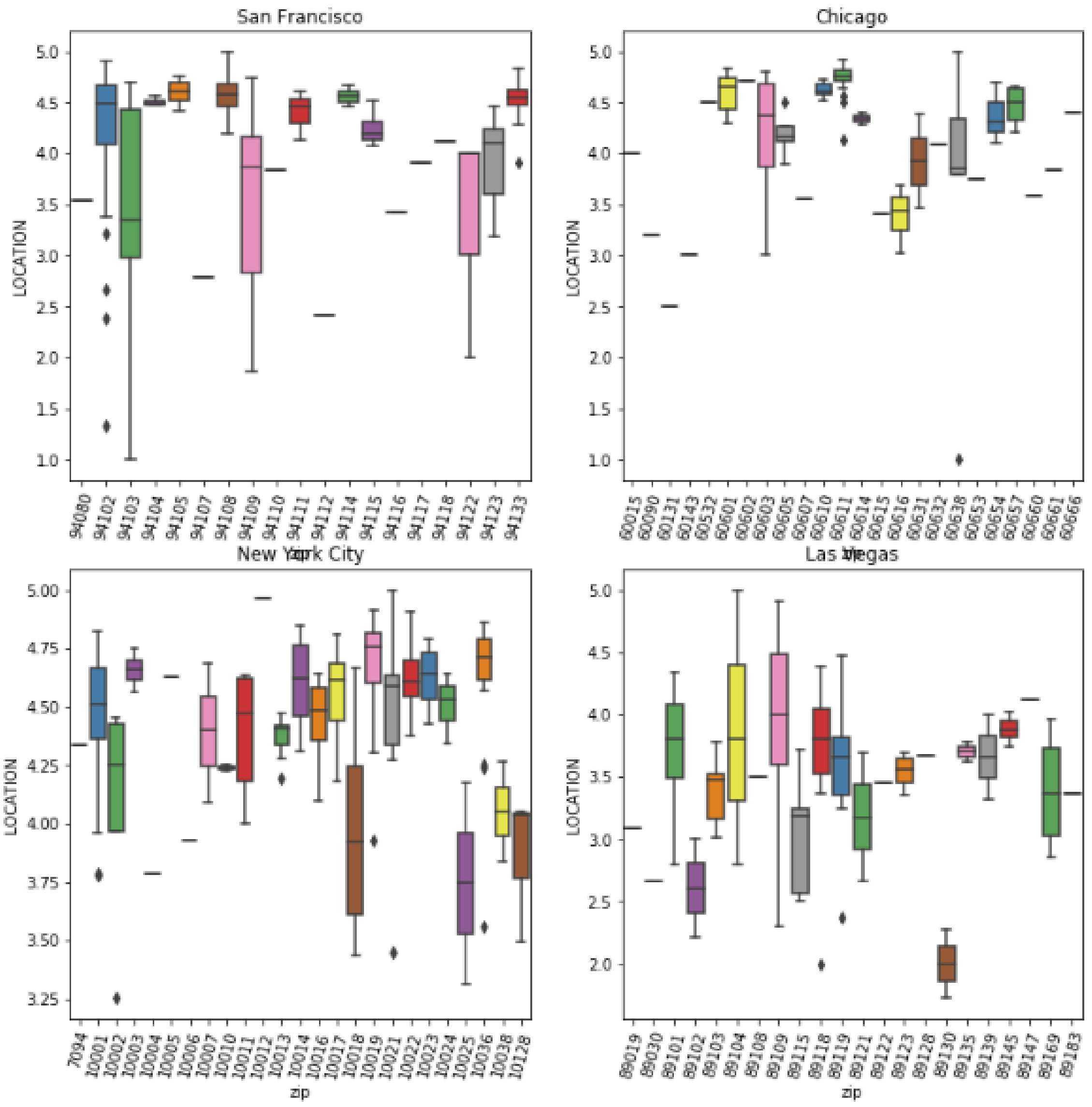


Figure 2:Location ratings for different zip codes.

Regarding the generated reviews, I have only created overall ratings, nothing regarding the other parameters existing on the dataset. To generate them, I first create a user, and assign him/her an age based on a uniform distribution and an economic power based on a gaussian distribution. Based on demographic data, these distributions capture the real trends. From there, I create an Ideal Hotel Price that users want to spend as the average between the chosen rows in the following tables:

Age	*IPH_age*
Under 25	$ 55.20
25-34	$ 118.25
35-44	$ 172.50
35-54	$ 207.00
55-64	$ 218.50
Over 65	$ 138.00

Percentile of household income	*IPH_wealth*
< 20%	$ 100.00
20% - 40%	$ 150.00
40% - 60%	$ 200.00
60% - 80%	$ 400.00
> 80%	$ 800.00

From here, a hotel is selected from a triangular probability distribution around the mean IHP. The rating assigned:

$$overall_{rating} = round(mean(rating) + f(distance\_to\_IHP) + white\ noise)$$

Assume the following:

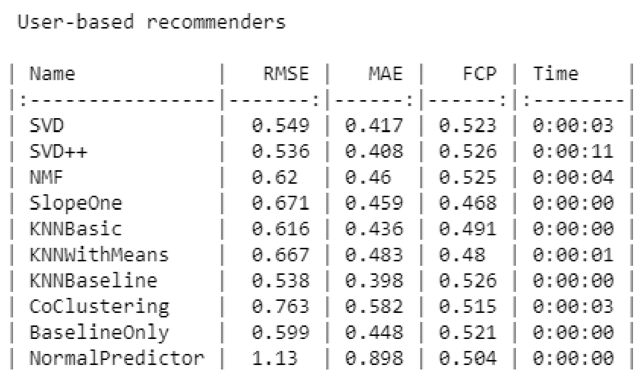
- If: $distance\_to\_IHP < 50\$, f = 0$
- If: $50\$ < distance\_to\_IHP < 100\$, f = -0.7$
- If: $100\$ < distance\_to\_IHP, f = -1.3$
- $white\ noise < 1.0$

This generates a set of ratings whose mean is very close to the one reported on the original dataset. I have generated (initially) about 9000 reviews from about 2000 users.

To recommend the hotels, different algorithms have been considered.

- Singular Value Decomposition performs poorly in time and provides poor results. The dataset is very sparse, so a dimensionality reduction is required to improve the performance.
- NMF behaves very similarly to SVD.
- kNN clustering outperforms all others in all aspects. A Cross-validation grid-search has provided the optimal values to train the model. The fraction of correct predictions is over 52% with such small dataset, most of the other approaches perform as a random recommender.

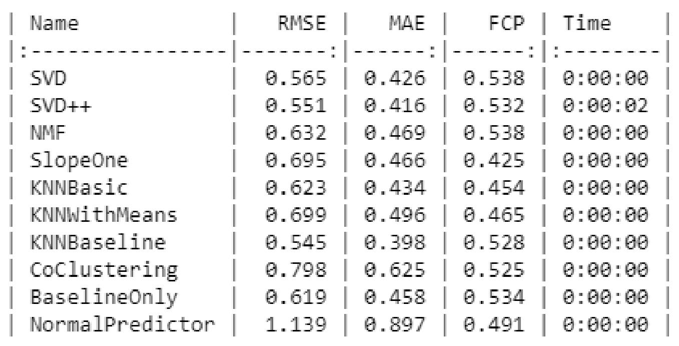
Here is a summary of the performance of each selected algorithm:



```
User-based recommenders

| Name             |   RMSE |   MAE |   FCP | Time    |
|:-----------------|-------:|------:|------:|:--------|
| SVD              |  0.549 | 0.417 | 0.523 | 0:00:03 |
| SVD++            |  0.536 | 0.408 | 0.526 | 0:00:11 |
| NMF              |  0.62  | 0.46  | 0.525 | 0:00:04 |
| SlopeOne         |  0.671 | 0.459 | 0.468 | 0:00:00 |
| KNNBasic         |  0.616 | 0.436 | 0.491 | 0:00:00 |
| KNNWithMeans     |  0.667 | 0.483 | 0.48  | 0:00:01 |
| KNNBaseline      |  0.538 | 0.398 | 0.526 | 0:00:00 |
| CoClustering     |  0.763 | 0.582 | 0.515 | 0:00:03 |
| BaselineOnly     |  0.599 | 0.448 | 0.521 | 0:00:00 |
| NormalPredictor  |  1.13  | 0.898 | 0.504 | 0:00:00 |
```

Figure 3: 5-fold cross-validation results on the whole dataset

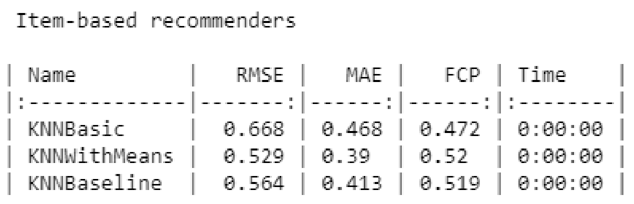
Looking at a test-training split, we observe a similar performance on unseen data. That indicates that we are not overfitting the model.



```
| Name             |   RMSE |   MAE |   FCP | Time    |
|:-----------------|-------:|------:|------:|:--------|
| SVD              |  0.565 | 0.426 | 0.538 | 0:00:00 |
| SVD++            |  0.551 | 0.416 | 0.532 | 0:00:02 |
| NMF              |  0.632 | 0.469 | 0.538 | 0:00:00 |
| SlopeOne         |  0.695 | 0.466 | 0.425 | 0:00:00 |
| KNNBasic         |  0.623 | 0.434 | 0.454 | 0:00:00 |
| KNNWithMeans     |  0.699 | 0.496 | 0.465 | 0:00:00 |
| KNNBaseline      |  0.545 | 0.398 | 0.528 | 0:00:00 |
| CoClustering     |  0.798 | 0.625 | 0.525 | 0:00:00 |
| BaselineOnly     |  0.619 | 0.458 | 0.534 | 0:00:00 |
| NormalPredictor  |  1.139 | 0.897 | 0.491 | 0:00:00 |
```

Figure 4: Performance results on a 25% of the dataset reserved for testing.

With regards to item-based recommendation, all we need is to change the axis we consider on the item-user matrix. Fortunately for us, that is simply achieved by changing an option to false. With that, only KNN is able to return results, the other algorithms do not accept those options. The results are similar to the user-based, and even though they do not show slower running time, they are known to scale worse and run slower in general.



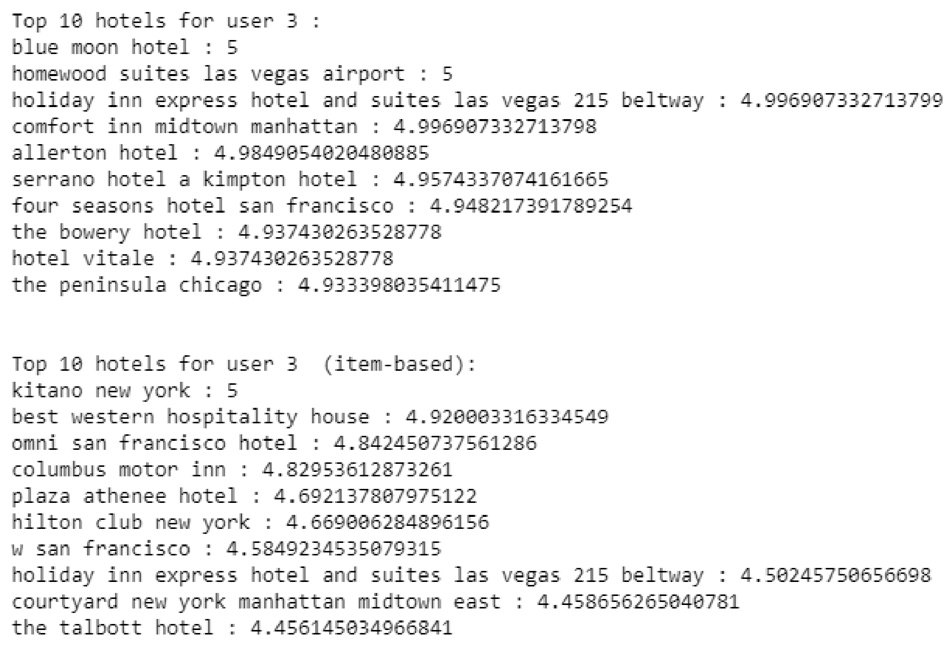
```
Item-based recommenders

| Name          |   RMSE |   MAE |   FCP | Time    |
|:--------------|-------:|------:|------:|:--------|
| KNNBasic      |  0.668 | 0.468 | 0.472 | 0:00:00 |
| KNNWithMeans  |  0.529 | 0.39  | 0.52  | 0:00:00 |
| KNNBaseline   |  0.564 | 0.413 | 0.519 | 0:00:00 |
```

Figure 5:Performance on item-based algorithms

6. Predicting capabilities

From the previous section, we can select KNN Baseline to be the default algorithm to deploy on the application. This may change when additional reviews are added. The way the surprise library works allows us to easily generate the top 10 hotels for any user. Adding a simple filter by city would be the essence of the application.

In the case that the user has not yet rated any hotel, the library provides simply the best hotels on average, without even producing an error. In such situation, user-based and item-based provide the same results.



```
Top 10 hotels for user 3 :
blue moon hotel : 5
homewood suites las vegas airport : 5
holiday inn express hotel and suites las vegas 215 beltway : 4.996907332713799
comfort inn midtown manhattan : 4.996907332713798
allerton hotel : 4.9849054020480885
serrano hotel a kimpton hotel : 4.9574337074161665
four seasons hotel san francisco : 4.948217391789254
the bowery hotel : 4.937430263528778
hotel vitale : 4.937430263528778
the peninsula chicago : 4.933398035411475


Top 10 hotels for user 3  (item-based):
kitano new york : 5
best western hospitality house : 4.920003316334549
omni san francisco hotel : 4.842450737561286
columbus motor inn : 4.82953612873261
plaza athenee hotel : 4.692137807975122
hilton club new york : 4.669006284896156
w san francisco : 4.5849234535079315
holiday inn express hotel and suites las vegas 215 beltway : 4.50245750656698
courtyard new york manhattan midtown east : 4.458656265040781
the talbott hotel : 4.456145034966841
```

Figure 6: Example output for a user that has some ratings

7. Creating the web app

Django is a python-based web app quite user-friendly and very versatile. A simple tutorial with helpful step-by-step indications can be found on [8]. The way Django works is by handling HTTP requests by forwarding them onto a pythonic template of views. It uses a set of model structures that define the data to be handled, and sends all the information to templates to generate the websites.

In this case, we will be dealing with a structure as follows:

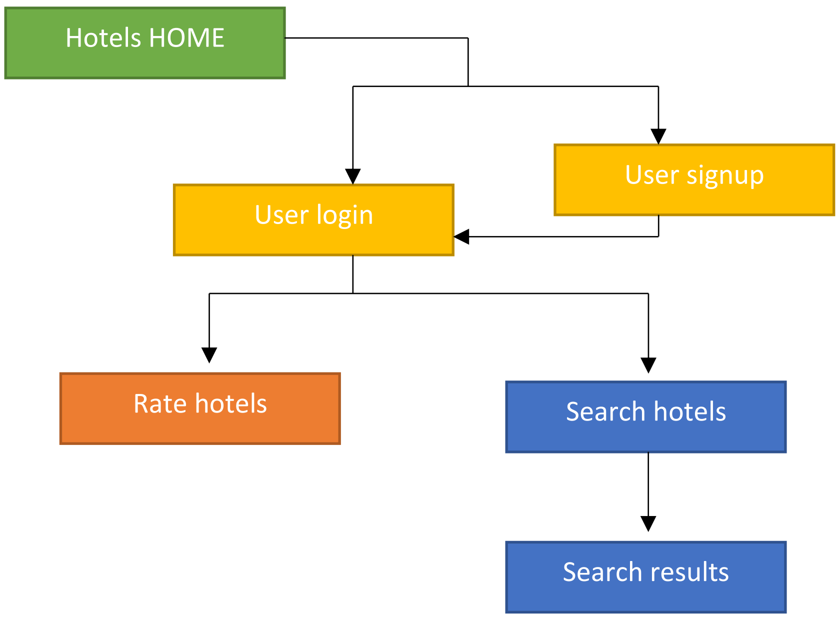
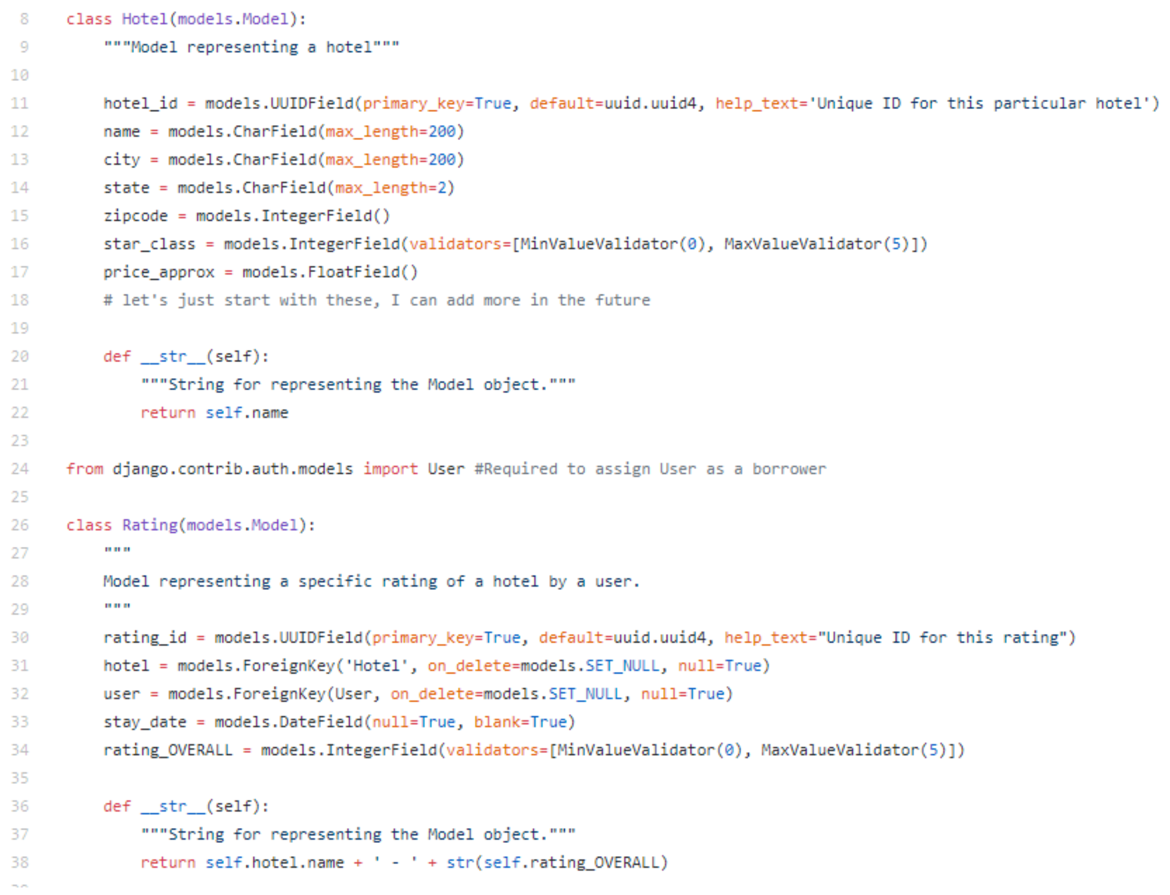


Figure 7: Web basic layout. This does not include password reset, username reminder, wrong log on and sign up, and some redirects with authentication.

The home page will be open to the public. From there, some authentication will be required. Django has an already built authentication system with a user database and user model that handles it all very easily. Due to the low level of security required here, this will be enough. All we want is to know who are you, so that we know what did you rate and we can recommend better hotels.

After the log on, there are two options: rate and search. Ratings are created following a simple rating model built upon a hotel model. The hotel model saves onto a database a hotel id, name, zip code, city and state, and star rating. The rating connects a user to a hotel, a date and a rating from 1 to 5.

The search part uses the surprise code described on the previous section, taking advantage of knowing the user, to return the lists.



```python
class Hotel(models.Model):
    """Model representing a hotel"""

    hotel_id = models.UUIDField(primary_key=True, default=uuid.uuid4, help_text='Unique ID for this particular hotel')
    name = models.CharField(max_length=200)
    city = models.CharField(max_length=200)
    state = models.CharField(max_length=2)
    zipcode = models.IntegerField()
    star_class = models.IntegerField(validators=[MinValueValidator(0), MaxValueValidator(5)])
    price_approx = models.FloatField()
    # let's just start with these, I can add more in the future

    def __str__(self):
        """String for representing the Model object."""
        return self.name

from django.contrib.auth.models import User #Required to assign User as a borrower

class Rating(models.Model):
    """
    Model representing a specific rating of a hotel by a user.
    """
    rating_id = models.UUIDField(primary_key=True, default=uuid.uuid4, help_text="Unique ID for this rating")
    hotel = models.ForeignKey('Hotel', on_delete=models.SET_NULL, null=True)
    user = models.ForeignKey(User, on_delete=models.SET_NULL, null=True)
    stay_date = models.DateField(null=True, blank=True)
    rating_OVERALL = models.IntegerField(validators=[MinValueValidator(0), MaxValueValidator(5)])

    def __str__(self):
        """String for representing the Model object."""
        return self.hotel.name + ' - ' + str(self.rating_OVERALL)
```

Figure 8: Ratings and hotels models.

To facilitate navigation, a lateral bar contains the most important links to the web locations, and the username who is logged in. To interact with the user, we need some Django forms to fill out and query the database.

8. *Deploying on cloud*

After the app is up and ready, we need to fill the database with the generated ratings and hotels (and users) and deploy it. A simple python code is able to do that by reading data from the csv files and saving them onto the generated database. With that, we can run it on a cloud server to have it open to the public. I chose Amazon Web Services because their free tier allows for a 12 months deployment with minimal cost.

According to AWS documentation, the most straightforward way to deploy an existing app is through their Elastic Beanstalk. Here, you create an ubuntu server on their end, and copy the files. You need to make sure all the required packages are installed, and all the authorizations are in order. A requirements.txt file is typically enough. The main problem I found is Apache trying to access non-existing folders that needed to be manually added.

9. *Final result*

To see the result, feel free to navigate to www.jonascuadrado.com to see by yourself. If you want to sign up it's perfect, otherwise you can log in using the credentials:

- Username: 3
- Password: password

You can replace ‘3’ by any number under 1800 to see different recommendations.

10. Conclusions and possible improvements

In this project I have successfully created a recommendation system comparing different algorithms, performing qualitative analysis on raw data, generating consumer models and deploying it on cloud. There are many possible improvements, which include:

- Using a multicriteria recommender to generate the most suitable hotels.
- Changing the algorithms as data grows, eventually deleting the virtual ratings.
- Adding style and formatting to the website.

11. References

[1] TripBarometer Travel Trends, (2016). https://www.tripadvisor.com/TripAdvisorInsights/wp-content/uploads/2018/01/Global-Report-US-Travel-Trends-TripBarometer-2016.pdf

[2] Kavita Ganesan and ChengXiang Zhai, (2011) “Opinion-Based Entity Ranking”, Information Retrieval.

[3] Dolnicar, Sara, Otter, T., (2003) “Which Hotel attributes Matter? A review of previous and a framework for future research”. http://ro.uow.edu.au/commpapers/268

[4] Radesh Rao Palakurthi, Sara J. Parks, (2000) “The effect of selected socio-demographic factors on lodging demand in the USA”, International Journal of Contemporary Hospitality Management, Vol. 12 Issue: 2, pp.135-142.

[5] Hokey Min, Hyesung Min, Ahmed Emam, (2002) “A data mining approach to developing the profiles of hotel customers”, International Journal of Contemporary Hospitality Management, Vol. 14 Issue: 6, pp.274-285.

[6] Bureau of Labor Statistics, (2010) “Travel: how much people spend?” https://www.bls.gov/spotlight/2010/travel/

[7] Minnaert, Lynn, (2017) “NYU US Family Travel Survey” https://www.scps.nyu.edu/content/dam/scps/pdf/200/200-4/200-4-16/P1718-0036-2017_Family_Travel_Survey.pdf

[8] Django tutorial by Mozilla Developer. https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Tutorial_local_library_website