**University Housing Finder**

Home-Hunting for Georgia Tech Students

ISyE 7406: DATA MINING AND STATISTCAL METHODS

4/25/2023

Team 9

**DISTRIBUTION OF WORK**

|  |  |
| --- | --- |
| Troy Allen | Introduction |
| Balakrishna Tirucadaiyur Suresh Kumaar | Layer of Modeling (LoM) Approach |
| Shreyas Atul Khare, Troy Allen | Literature Review |
| Chin-Heng Lin | Problem Description and Novelty |
| Balakrishna Tirucadaiyur Suresh Kumaar, Troy Allen, Aidan Mulaokar | Data Collection and Pre-processing |
| Abhishek Sanjay Mutha, Omkar Borate | Association Rule Learning |
| Chin-Heng Lin, Priyank Shukla, Troy Allen | Data Merging |
| Adian Mulaokar, Yash Patni, Balakrishna Tirucadaiyur Suresh Kumaar, Abhishek Sanjay Mutha, Shreyas Atul Khare | Data Modeling |
| Aidan Mulaokar, Balakrishna Tirucadaiyur Suresh Kumaar, Abhishek Sanjay Mutha | Property Assessment |
| Yash Patni, Priyank Shukla | Customers and Revenue Streams |
| Troy Allen, Balakrishna Tirucadaiyur Suresh Kumaar, Yash Patni, Abhishek Sanjay Mutha, Priyank Shukla, Aidan Mulaokar | Future Scope and Considerations |
| Abhishek Sanjay Mutha, Troy Allen, Omkar Borate | Conclusion |
| Shreyas Atul Khare, Aidan Mulaokar | Website Creation/PowerPoint Presentation |
| Shreyas Atul Khare, Aidan Mulaokar, Troy Allen | PowerPoint Presentation |

**TABLE OF CONTENTS**

[1. Introduction and Layer of Modeling (LoM) approach: 4](#_Toc1207886144)

[2. Literature Review 6](#_Toc1405403436)

[3. Problem Description and Novelty 7](#_Toc718062385)

[4. Data Collection and Pre-processing 7](#_Toc1634537979)

[6. Data Merging 15](#_Toc79389757)

[7. Data Modeling 17](#_Toc1018215874)

[8. Property Assessment 21](#_Toc1824541512)

[9. Customers and Revenue Streams 25](#_Toc2112092168)

[10. Future Scope and Considerations 28](#_Toc1049349242)

[11. Website Creation 31](#_Toc1681075114)

[12. Conclusion 33](#_Toc938871320)

[13. Appendix 34](#_Toc331207942)

[14. References 49](#_Toc1758703979)

## 

## **1. Introduction and Layer of Modeling (LoM) approach:**

Finding the correct home or apartment is a difficult task that many students are faced with every semester. Everyone has heard of the horror stories that many students tell regarding their housing situations in their first few years at college. Whether it be an apartment infested with bugs, a random roommate that stays up at all hours of the night, or some other unforeseen issue, housing situations for university students are often far from ideal. This is because universities are filled with students from diverse backgrounds that have a variety of personal preferences, lifestyles, communication styles, and interests. This sentiment often extends to housing preferences and many properties fail to account for the nuance of the situation. There is no dedicated resource that can help students filter and find apartments close to campus based on their choice of roommates, proximity to their department of campus, shuttle services, and other amenities. This is especially true near Georgia Tech, since Atlanta is such a crowded city, and most housing finders are aimed at the general population. Most students and their parents will be more concerned with their home’s proximity to campus and its safety levels rather than the amenities within. There are filters that exist on sites like Zillow, but most of the apartments will be recommended based on the amenities and rent alone. Adding nuance to this process in the form of specializing it towards Georgia Tech, or other university students should the scope be expanded, students should greatly increase renter satisfaction and fit people with more appropriate homes. Especially given the frequency at which students tend to change housing during their tenure at school, we aimed to create a housing tool that was tailored towards students at Georgia Institute of Technology.

The following Layer of Modeling approach was used to approach the given problem:

Students’ Problem Idea: “Housing Finder”

**LoM Chart for Summarizing Problem Background**:

1. Location
2. Accessibility
3. Department.
4. Stinger Bus Stop.
5. Library.
6. MARTA Bus Stop.
7. Subway Station.
8. Airport.
9. Region
10. Home Park.
11. Midtown.
12. Downtown.
13. West Campus.
14. Population Demographic
15. Number of Students.
16. Ethnicity (for purpose of meeting similar people).
17. Nearby Services
18. Grocery Stores.
19. Restaurants/Food Outlets.
20. Pharmacies.
21. Medical Facilities.
22. Resources
23. Budget
24. Cost of Room.
25. Utilities included/not included (Water + Wi-Fi + Trash).
26. Lease term offered.
27. Apartment
28. Type of House (Apartment/Individual).
29. No. of Rooms and Bathrooms.
30. Furnished/unfurnished.
31. Sharing allowed/not.
32. Square footage.
33. Rating.
34. Amenities offered.
35. Recreational Facilities.
36. Security.
37. Roommate
38. Want a roommate or not.
39. Want to share room or not.
40. Department of roommate.
41. Food preference.
42. Smoking and drinking preference.

Given our restrictions in terms of resources, not all these variables were able to be included, but many were.

## **2. Literature Review**

Apartment finders have been of great interest to a lot of people simply due to the fact that finding houses for a person living far away from the house location is very difficult. This is because of various factors that influence choosing a house. Factors such as safety, commute comfort are subjective to individuals and difficult to gauge from afar. For the young population, renting is also much more common. It has been noted that “millennials are less likely to own a home despite having a higher number of college graduates when compared to Gen Xers and Baby Boomers.”[3] Financial literacy and income were shown to not have that large of an impact on a person’s decision as to rent or buy a home. Instead, factors like age and college grade level were much more indicative and increasing age was positively correlated with the likelihood of buying a home instead of renting. [3] With this in mind, the value of apartment finders is even higher than in the past. Many young people, especially college students, are afraid of the commitment that comes with purchasing a home and are seeking to rent. However, the difficulty of finding an appropriate home, especially on a short-term cycle that many college students are forced into by semester cycles, remains. Apartment finders should be more focused on the younger population than they currently are because that is their main market.

Data has been a leveler for this notion. All the software that has been created works on extraction on data and uses that data to deliver the best rates or make recommendations. Data collection should be a structured and streamlined process which makes sure of efficiency and as a means of defending the findings and conclusions of the project.[1] A data collection mechanism should be ensured to have enough strength to convey to the customer suitable apartments that are appealing and personalized to the person’s preferences. This will help to eliminate some of the uncertainty that is involved in the home-hunting process, which is certainly a concept that is foreign to many young people just graduating high school.

Data runs essentially on feedback from users. It learns from whatever it gets from its direct users and makes itself improve. We try to leverage the feedback mechanism of user by providing them with the recommendations and then improving our model based off it. Data can be used for different purposes in all imaginable combinations with other data sources and without losing value. [4] This is the essence of our model which uses the existing data but uses it in a way which is unique and specified to suit students.

## **3. Problem Description and Novelty**

As mentioned above, the process of finding the perfect housing can be a difficult task, especially for students who are new to the Georgia Tech campus. To address this issue, our project aims to classify students into different clusters based on their home-hunting preferences. If we can identify the specific cluster groups, we will be able to gain deeper insights into the types of housing features that students pay more attention to when searching for an apartment to live in.

To achieve this goal, we plan to utilize various data mining and machine learning techniques, such as classification, linear regression, decision tree models, etc., to analyze and categorize the data we collected. We will analyze the factors like crime rates and amenity scoring to create a comprehensive list of suitable apartments for each student. By doing so, we can provide more personalized recommendations for the students given their individual demands and preferences.

Based on a limited sample size of the data collected, we will utilize resampling methods such as bootstrapping to ensure that our recommendations are based on more solid statistical analyses, and the outcome we get can be more accurate and reliable.

The traditional process of finding housing typically involves the consideration of features such as the renter’s budget or the number of rooms in the apartment only. However, this approach could fail to consider the unique needs of each renter. As such, we hope to create a novel housing finder that goes beyond the above generic factors, and further provides a more personalized recommendation based on a wider range of factors.

Our finder will consider factors that are more specific to Georgia Tech students, such as commute time to different department buildings. Also, other secondary yet important factors will be included in our model, such as crime rates, proximity to restaurants, grocery stores, and other amenities, in order to make sure the renters are able to choose a safe and convenient location.

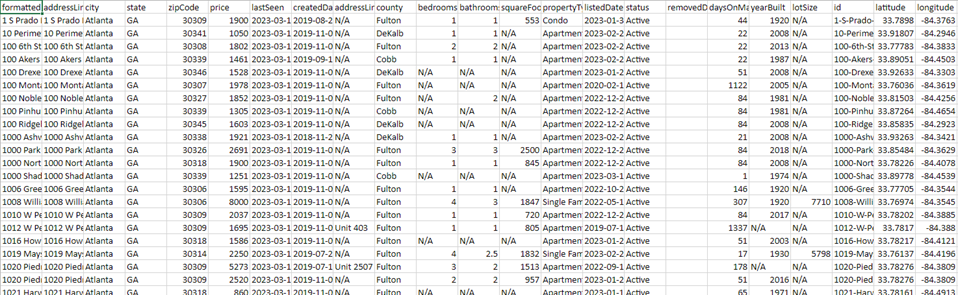
Overall, our apartment finder is very different from traditional apartment finders as we offer a more innovative way for Georgia Tech students to find their perfect housing. By providing personalized recommendations based on various unique and specific criteria, our finder will make the home-hunting process more efficient and help the students find the ideal apartment for their needs and expectations.

## **4. Data Collection and Pre-processing**

**Housing Data (Pricing, Metadata, and Geolocation):**

Pricing, location, and apartment size are factors which have a great deal of influence on selecting an apartment to rent. This is especially true for young people, Georgia Tech students in our case, where they are often living away from home and do not have large independent incomes. With that in mind, our data sources needed to provide access to these factors so they could be utilized as parameters in our approach. There are many sources of housing data online, so we decided to use API pulls in order to get specific pricing, geospatial, and metadata for Atlanta apartments and homes.

To aggregate this data, we used a RapidAPI request provided by RealityMole. Our previous plan to use Zillow had to be changed because they recently made changes to their API architecture and are now outsourcing it through a third-party site that seems to be more restrictive in its use. This ended up working in our favor because RealityMole provides geolocations for the listed apartment buildings and has a more robust REST API than what Zillow was going to provide. Below is an example of our request and results:



As can be seen in the above screenshots, we were able to easily pull apartment data that included the pricing, sizes, and geospatial data for Atlanta apartments.

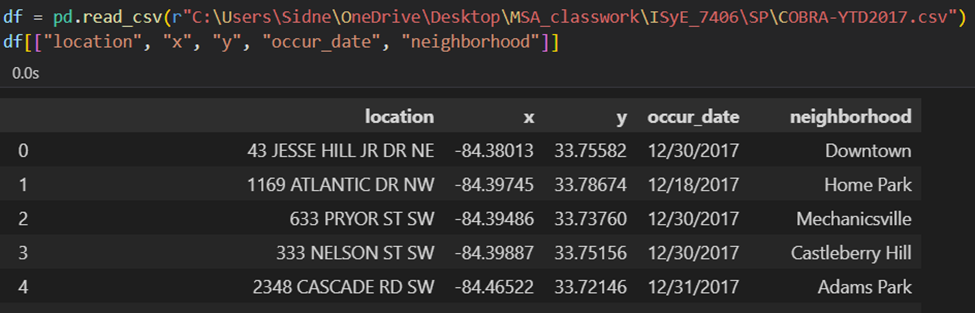
**Crime Data:**

We also wanted to consider the impact of crime and safety on a student’s apartment choice. Many parents voice their concerns about the safety of their children when they are moving away to college. This is an even more frequent notion when their child’s college is in a large city like Atlanta, which can be dangerous depending on the situation. We needed to collect data for areas surrounding Georgia Tech related to the number of crimes that occurred, the types of crimes that occurred, and how close they were in proximity to a student’s potential apartment home.

The dataset we decided on using was from Kaggle. Through it we were able to collect crimes that occurred, the coordinates of said crime, and neighborhoods in which they occurred. From this we were able to extrapolate the following:

* + How many crimes are happening within a given region (ie Home Park). We can use this to create a safety ranking and potentially a score
  + How many crimes are occurring within a certain range of a property

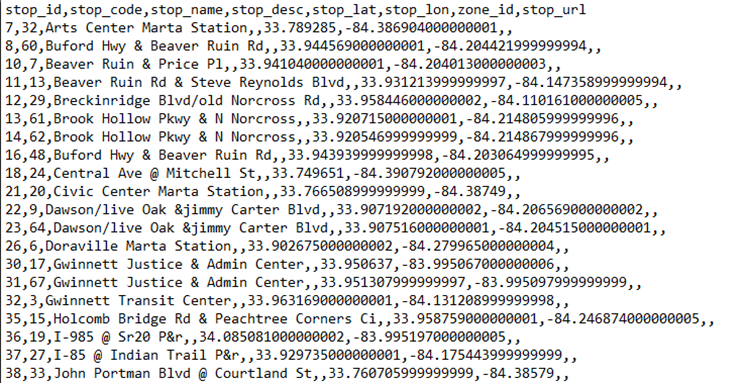
A short example of how we pulled this data is below.



Given the geospatial nature of both the housing and crime datasets, we were able to easily work with and merge this data as will be discussed later in the report. This combination and merging helped us to recommend appropriate apartments to students utilizing our tool.

**Stinger Data:**

We were also able to find an online resource for Stinger stops around Georgia Institute of Technology that provided the stops as geocoded locations. This was somewhat difficult to include in our approach given our route of analysis, but it was a decent portion of our data collection attempts.



**Student Housing and Preference Data:**

The parameters which would be most important for a student at Georgia Tech were collected with the help of floating a form to current students. The list was boiled down to primary and secondary parameters, with primary parameters including the number of roommates, proximity to home department etc. The secondary parameters included proximity to restaurants, public transportation, apartment amenities etc.

Through the data collected through the student responses, our goal was to create a database of advantages and disadvantages of apartments in which the students were currently residing in. When a query is made on the website where a student/prospective resident is hunting for an apartment, we can compare the previous responses and then provide a suitable apartment for the user. To weigh the parameters that were collected, another form was floated to understand the requirements of each potential resident.

We received around 100 responses after floating the current students housing survey. After inspecting the dataset, we observed that many residents used different names for the same department, apartment etc. For example, one student studying ‘Industrial Engineering’ here at Georgia Tech entered the course he was studying as ‘MSIE’ and some responses it was entered as ‘Industrial Engg’. To clean all the responses and to make it uniform, the find and replace tool in Excel was used to edit all these responses automatically instead of manually. The same tool was used to clean the responses obtained for the Apartment name, Rent/month etc.

Since the responses obtained from floating the form were not sufficient, bootstrapping was performed to increase the sample size. The steps performed for performing bootstrapping are listed below:

1. Collect the original Dataset: The first step for performing bootstrapping is to collect the original Dataset that we would like to resample. The dataset used for bootstrapping was obtained by asking preferences of students about their current housing.
2. Clean the dataset: The next step is to clean the dataset. Cleaning the dataset is important because it helps to ensure that the data is accurate, complete, and consistent. This would in turn result in reliable as well as valid analysis of the results. The cleaning of the dataset for this project was performed by using the ‘Find and Replace’ tool in Excel.
3. Create Bootstrap Datasets: To create bootstrap datasets, we randomly sample from the original dataset with replacement, so that the size of the new dataset is larger than the original dataset.
4. Verification and validation of new Dataset: Verification and validation of bootstrapped samples are required to ensure that the bootstrap technique has been applied correctly and that the results obtained are reliable and accurate. To ensure this, visual inspection and confidence intervals were found and compared with the original dataset.

The trends found in the dataset after bootstrapping was performed is shown in the images below:

Chart, bar chart, histogram

Description automatically generated Chart, bar chart

Description automatically generated

Chart, bar chart, histogram

Description automatically generated Chart, bar chart

Description automatically generated

Chart, bar chart, histogram

Description automatically generated Chart, bar chart

Description automatically generated

From the above figures, we can see that the majority of students prefer Apartments over Individual Houses. We can also see that the number of roommates most students have is 3, and that most likely means that the students are likely to be in a 2B2B apartment setting.

**5. Association Rule Learning:**

Apriori algorithm is a widely used algorithm in data mining for association rule learning. The Apriori algorithm can be used to mine frequent itemsets and association rules from our student data. In our case, the frequent itemsets could include combinations of department, apartment type, proximity of amenities, housing layout preference, roommate preference, and budget. The association rules generated by the Apriori algorithm could tell us which combinations of variables are most strongly associated with each other.

These insights can be useful in several ways. We can use the patterns which discover to make recommendations to incoming students about which housing options and roommates might be a good match for their preferences. It's important to note that the Apriori algorithm is just one tool for analyzing student data.

In the context of our student data, **we argue that applying the Apriori algorithm could be seen as a form of data preprocessing or exploratory data analysis**. By identifying which combinations of variables are most strongly associated with each other, we gained insights into the patterns and relationships within our data.

Here are the Steps how we implemented the Apriori Algorithm on our student data:

Step 1: Import the required libraries: pandas, numpy, seaborn, matplotlib, mlxtend, and apyori

After installing the apyori library (used in python for apriori algorithm for association rule learning. Then the above libraries need to be imported to run relevant functions and operations on data.

Step 2: Load the student data into a pandas dataframe using pd.read\_csv()

Step 3: Drop the 'Timestamp' column from the dataframe using data.drop().

Removing the time stamps from data is necessary to avoid it to be included in the apriori implementation as the time stamp is only for record purposes.

Step 4: Visualize distribution of the 'Where do you stay out of these areas?' column using a countplot and sns.countplot(). Using the above instructions, the distribution of apartments where current students are residing can be quickly understood.

Step 5: Generate a new dataframe with random samples from each categorical column using np.random.choice(), concatenate with the original data to create a larger dataset using pd.concat().

Step 6: Define a list of attributes to consider for the Apriori algorithm. Create a list of transactions where each transaction is a list of attribute values for a student. The attributes used in the algorithm to formulate association rules are: ‘In which department are you studying at Georgia Tech?', 'Housing Type', 'Housing Configuration?', 'Roommate information?', 'Rent you pay?', 'What is your Lease Term?','Food eating habits’

These attributes will help to identify the most frequent combinations in the data, the most preferred choices among the student regarding housing type and configuration, and food habits. These attributes also help in identifying the most important associations.

Step 7: Generate association rules using apriori() or apyori() with a minimum support and minimum confidence threshold of 0.1 and a minimum confidence of 0.5. It generates a set of rules based on the transactions.

In the Apriori algorithm, the "rule" gives the relationship between two sets of items, where one set is called the antecedent and the other set is called the consequent. The rule represents the conditional probability that the consequent will occur given that the antecedent has occurred.

While the support is a measure of how frequently the antecedent and consequent occur together in the dataset. It is defined as the number of transactions that contain both the antecedent and the consequent divided by the total number of transactions.

Step 8: Convert the rules iterator generated by the algorithm to a list. Then we sort the most frequent item sets and extract the top 5 item sets and similarly we identify the most interesting rules in the associations, sort them and extract the top 10 to identify the most interesting relations in the data.

**Results:**

Following results are obtained from above implementation:

As explained earlier the support value indicates how frequently the relation/association has occurred in the dataset. Thus, a higher support value implies that a rule is frequent item set/association that is observed/recorded/present in the dataset.

Also the minimum threshold for the support is set to decide which support values to be considered as significant and the respective relationships/association should be considered for further analysis or need to be closely monitored.

The following table displays the most frequent item sets along with the corresponding support values.



Similarly, the following table summarizes the most interesting associations that we found using the apriori algorithm. Association 2 implies that students from same department would prefer an apartment with a monthly rent of $800 to $1000. While association 3 tells us that a student who prefers a personal room will choose the apartment with a roommate from same department.



Association 7 indicates that a high percentage of current students have signed for a longer duration lease. Similarly, from association 6 we ca infer that students who are sharing rooms in an apartment, prefer to have vegetarian room mates as they themselves have a vegetarian diet.

Above graph indicates the top 5 frequent item sets identified in the dataset. Thus, we have used the apriori algorithm for exploratory data analysis to identify important patterns in the dataset as well as define and identify significant or interesting associations which can help us in the further analysis.

## **6. Data Merging**

As our project involved various datasets from multiple sources, we combined datasets from all relevant sources and integrated them into one single database. We used JSON data structures to create nodes that contain all relevant fields to make it easier to analyze and provide insights from the data. An example node looked like this:

{

uid: 903268798

Address: 1270 Spring St Nw

City: Atlanta

State: GA

Zip: 30309

Price: 1900

LastSeen: 2023-01-01

CreatedDate: 2019-08-02

Address2: N/A

County: Fulton

Bedrooms: 1

Bathrooms: 1

SqFt: 553

PropertyType: Condo

ListedDate: 2023-01-03

Status: Active

RemovedDate: N/A

DaysOnMarket: 44

YearBuilt: 1920

LotSize: N/A

Latitude: 33.7898

Longitude: -84.3763

NearbyCrime: [

{

Latitude: 33.7558

Longitude: -84.3801

Date: 2017-12-30

},

{

Latitude: 33.7867

Longitude: -84.3948

Date: 2017-12-30

},

]

NearbyStingerStops: []

}

As can be seen, this node contains the combined information from the apartment, crime, and stinger stop datasets for this individual listing. These types of nodes would be used to train our models based on the results from the Student Housing Preference surveys that we floated on two different occasions.

The integration of the student dataset, housing dataset, and crime dataset allowed us to create personalized recommendations that consider different factors and criteria, which lead to a more efficient and effective housing search process of our finder.

## **7. Data Modeling**

To evaluate our datasets, we decided to explore several models taught in class and techniques for improving their performance.

**Models From Class:**

1. Linear Regression: Linear Regression is a simple and widely used technique for modeling relationships between continuous variables. It is a good choice for this project as it provides a straightforward way to understand the relationship between the housing features and the students' preferences. However, the performance of the model in this case was poor, indicating that the relationship between the variables might not be linear.
2. Decision Tree: Decision Trees are a popular method for classification and regression tasks. They work well for this project because they can handle both continuous and categorical variables, and they are easy to interpret. They can capture non-linear relationships between features, which might be present in this dataset. However, they tend to overfit the data, leading to poor generalization.
3. Principal Component Analysis (PCA): PCA is a dimensionality reduction technique that can help to reduce the number of variables in the dataset while preserving the majority of the information. This method is beneficial for this project because it can simplify the analysis and reduce the risk of overfitting by removing collinearity among variables.
4. Support Vector Machine (SVM): SVM is a powerful machine learning technique for classification and regression tasks. It works by finding the optimal hyperplane that separates the data points of different classes with the largest possible margin. SVM is a good choice for this project because it can handle non-linear relationships between variables and can produce more accurate predictions, especially in high-dimensional spaces.
5. Random Forest: Random Forest is an ensemble method that combines multiple Decision Trees and averages their predictions. It can handle non-linear relationships and automatically considers feature interactions, making it a good choice for this project. Additionally, Random Forest provides a measure of feature importance, which can help to identify the most relevant variables for predicting student housing preferences.

**Methods From Class Attempted to Improve Model Performance:**

1. Cross Validation: Cross Validation is a technique to assess the performance of a model by training and testing it on different subsets of the data. It helps to reduce overfitting and provides a more reliable estimate of the model's performance. This method is beneficial for this project, as it can help to choose the best model with varying complexity for any base model.
2. Bagging: Bagging is an ensemble method that combines multiple base models to improve prediction accuracy and reduce overfitting. By aggregating the predictions of multiple models, bagging can capture more complex relationships in the data and improve the model's performance. This method is suitable for this project because it can help to create a more robust model for predicting student housing preferences.
3. Gradient Boosting: Gradient Boosting is another ensemble method that combines multiple models in a sequential manner. It focuses on minimizing errors made by previous trees, leading to a more accurate model. This technique is a good choice for this project because it can potentially capture complex relationships in the data and improve prediction accuracy compared to single models.

**Content based Filtering:**

After trying several techniques and methods to improve the performance of the model, we aim to improve the prediction capabilities of the tool. Content-based filtering is a technique used in recommender systems to recommend items (such as movies, books, or products) to users based on the content of the items themselves. In content-based filtering, features or characteristics of the item are used to recommend similar items. For example, if a user likes a particular movie, content-based filtering will recommend other movies with similar themes, actors, directors, or genres. This approach does not rely on data about other users or their ratings, but rather on the properties of the items themselves. Content-based filtering is useful when there is a limited amount of user data available, or when the recommendations need to be personalized to the individual user's interests and preferences.

In our model, content-based filtering is used to provide recommendations for housing to the user, based on the attributes entered by the user. Using attributes and preferences of the user, we design the model to provide top 5 housing recommendations to the user. Primarily, the main attributes that are taken from the user are:

1. Type of Housing: Apartment/House
2. Location (Area): Midtown (east of campus)/West campus/Home Park/Atlantic Station
3. Number of roommates user wants to share apartment with
4. Rent: Range of rent the user wants to pay for housing
5. Layout of Apartment: preference of user in terms of 2B2B/4B2B

Based on the above inputs, the procedure followed by the model is as follows:

Steps taken to implement the Content Based Filtering:

Step 1: Loading the Data

The first step is to load the data into a pandas Data Frame. In this case, the data is stored in a CSV file called 'Housing Form Edited for Content Based Filtering.csv'. This can be accomplished using the read\_csv function from pandas.

Step 2: Create the Apartment Profile

The next step is to create a profile for each apartment in the dataset. This profile will be used to calculate the similarity between apartments and the user's preferences. In this case, the profile will be a string that contains the apartment's location, rent, number of roommates, house layout, and house type.

Here, we define a function called **create\_apartment\_profile** that takes an apartment as input and returns a string containing the apartment's profile. The function uses the **join** method to concatenate the apartment's location, rent, number of roommates, house layout, and house type into a single string.

Next, we use the **apply** method of the DataFrame to apply the **create\_apartment\_profile** function to each row of the dataset. The resulting profile strings are stored in a new column called 'profile'.

Step 3: Vectorize the profiles

The next step is to vectorize the apartment profiles using the **CountVectorizer** class from scikit-learn. This will convert each profile string into a vector of token counts.

Here, we create an instance of the **CountVectorizer** class and fit it to the apartment profiles using the **fit\_transform** method. This method returns a matrix where each row corresponds to an apartment and each column corresponds to a token in the vocabulary.

Step 4: Compute Cosine Similarity

The next step is to compute the cosine similarity between the user's preferences and each apartment in the dataset. We can do this using the **cosine\_similarity** function from scikit-learn.

Here, we first use the **create\_apartment\_profile** function to create a profile string for the user's preferences. We then use the **transform** method of the vectorizer to convert this profile string into a vector of token counts.

We then use the **cosine\_similarity** function to compute the cosine similarity between the user's preferences and each apartment in the dataset. This returns an array of similarity scores, where each score corresponds to an apartment in the dataset.

Step 5: The final step is to sort the apartments by their similarity scores and recommend the top matches to the user. We can do this by sorting the DataFrame by the similarity scores in descending order and selecting the top recommendations.

To summarize the above steps:

In this case, the recommendation system is designed to recommend apartments to a user based on their preferences. The preferences of the user are stored in a dictionary called user\_profile, which contains the user's preferred location, rent, number of roommates, house layout, and house type.

The apartments in the dataset are represented as a set of attributes or features such as location, rent, number of roommates, house layout, and house type. The algorithm creates a profile for each apartment by concatenating these features into a single string. The profiles are then vectorized using the CountVectorizer class from scikit-learn. This converts each profile string into a vector of token counts.

The algorithm then computes the cosine similarity between the user's preferences and each apartment in the dataset. Cosine similarity is a metric that measures the similarity between two vectors. In this case, it measures the similarity between the user's preference vector and each apartment's profile vector. The similarity score ranges from -1 to 1, where a score of 1 indicates that the two vectors are identical, a score of -1 indicates that they are completely dissimilar, and a score of 0 indicates that they are orthogonal.

The algorithm then sorts the apartments by their similarity scores and recommends the top matches to the user. The recommended apartments are the ones with the highest similarity scores, which means that they are the most similar to the user's preferences.

One advantage of Content-Based Filtering is that it does not require user data or ratings, which can be difficult to obtain in practice. It only requires information about the attributes or features of the items being recommended. However, one limitation of Content-Based Filtering is that it may not recommend items that are outside the user's preferences or that the user has not seen before.

We suggest that Content- based Filtering is a good procedure to create user profiles and provide recommendations to students. The output with the first recommendation is closest to the user profile and matches to his/her expectations at best. The remaining 4 recommendations are equally close and can be explored by the student whether they match their expectations.

## **8. Property Assessment**

The objective of our project is to provide student housing recommendations based on factors such as roommates, rent, square footage, and distance to campus. In this section, we will assess the performance of different machine learning models applied to our dataset.

1. Linear Regression: The Linear Regression model yielded a Mean Squared Error (MSE) of 1.94 and an R squared value of -0.16. The negative R squared value indicates that the model is not an appropriate fit for the data, as it is performing worse than a horizontal line.
2. Decision Tree: The Decision Tree model produced an MSE of 1.6 and an R squared value of 0.02. Although the MSE is lower than that of the Linear Regression model, the low R squared value suggests that the model still has poor explanatory power.
3. Decision Tree with Cross Validation: The mean accuracy of this model is 0.29, which is quite low and indicates poor performance in predicting student housing preferences.
4. Decision Tree-based Bagging: The mean accuracy improved to 0.37, which, although higher than the previous model, is still not sufficient for making reliable recommendations.
5. Decision Tree-based Gradient Boosting: The mean accuracy remained unchanged at 0.37, showing no improvement over the bagging approach.
6. Principal Component Analysis (PCA): PCA was performed to reduce the dimensionality of the dataset. The explained variance ratios for the first two principal components are 0.589 and 0.251, which together account for 84% of the total variance. This suggests that a reduced-dimension dataset may still capture the majority of the information.
7. Support Vector Machine (SVM): The confusion matrix and classification report for the SVM model show an accuracy of 0.67. Although better than previous models, it may still not be reliable enough for practical use.
8. SVM-based Bagging: The accuracy of this technique showed a slight decrease in accuracy from 0.68 to 0.66.
9. SVM-based Gradient Boosting: The accuracy fell further from 0.66 to 0.62 showing that the standard SVM technique was the best for our dataset.
10. Random Forest: The Random Forest model produced a confusion matrix and classification report identical to the SVM model, with an accuracy of 0.67. This model also has limited predictive power in recommending student housing preferences.

Based on these results, the Support Vector Machine model outperforms the other models in terms of accuracy. It is important to note that the dataset is relatively small, and the performance of the models may vary when applied to larger datasets or when using different hyperparameters.

In addition, we have performed Principal Component Analysis (PCA) to reduce the dimensionality of the dataset and understand the relationships between the features. PCA revealed that the first two principal components explain around 84% of the variance in the data, with rent and distance being the most influential features.

Overall, our recommendation is to use the Support Vector Machine model for predicting housing ratings. However, we suggest further investigation into other algorithms or feature engineering techniques to improve the model's performance.

Property Assessment for Content-Based Filtering for House Recommendations:

# Property Assessment

assessment\_features = [ 'location',

    'rent',

    'number\_of\_roommates',

    'House\_Layout',

    'House\_Type']

# Select a sample apartment for assessment

apartment = apartment\_data.sample()

# Create an apartment profile for assessment

apartment\_profile = create\_apartment\_profile(apartment.iloc[0])

# Transform the apartment profile to a matrix of token counts

apartment\_vector = vectorizer.transform([apartment\_profile])

# Calculate the cosine similarities between the sample apartment and all other apartments

similarities = cosine\_similarity(apartment\_vector, apartment\_matrix)

# Get the similarity scores for all apartments except for the sample apartment

similarity\_scores = similarities[0, :].tolist()

similarity\_scores.pop(apartment.index[0])

# Get the assessment feature values for the sample apartment and convert them to numerical types

assessment\_values = apartment[assessment\_features].apply(pd.to\_numeric, errors='coerce').values.tolist()[0]

# Calculate the average feature values of the most similar apartments and convert them to numerical types

similar\_apartment\_features = apartment\_data.loc[similarity\_scores.index(max(similarity\_scores)), assessment\_features].apply(pd.to\_numeric, errors='coerce').tolist()

# Calculate the percentage difference between the sample apartment and the average features of the most similar apartments

assessment\_results = [(similar\_apartment\_features[i] - assessment\_values[i])/assessment\_values[i]\*100 for i in range(len(assessment\_features)) if not pd.isna(similar\_apartment\_features[i]) and not pd.isna(assessment\_values[i])]

# Print the assessment results

for i in range(len(assessment\_results)):

    print(f'{assessment\_features[i]}: {assessment\_values[i]}')

    print(f'Average {assessment\_features[i]} of most similar apartments: {similar\_apartment\_features[i]}')

    print(f'Percentage difference: {assessment\_results[i]:.2f}%')

The steps for the above code is as follows:

This code performs a property assessment for a randomly selected apartment by comparing its features to the features of the most similar apartments in the dataset.

The first step is to select the features that will be used for the assessment, which are stored in the assessment\_features list. Then, a random apartment is selected from the dataset using the sample() method, and its features are used to create an apartment profile using the create\_apartment\_profile() function.

Next, the apartment profile is transformed into a matrix of token counts using the same vectorizer object that was fit to the apartment descriptions in the previous code. The cosine\_similarity() function is then used to calculate the cosine similarities between the sample apartment and all other apartments in the dataset, and the similarity scores are extracted for all apartments except for the sample apartment.

The feature values for the sample apartment are then extracted and converted to numerical types using the pd.to\_numeric() method, and the average feature values of the most similar apartments are calculated and converted to numerical types as well. The percentage difference between the sample apartment and the average features of the most similar apartments is then calculated for each assessment feature, excluding any features that contain missing values.

Finally, the assessment results are printed out for each feature, along with the percentage difference between the sample apartment and the most similar apartments.

The input profile for our property assessment is:

user\_profile = {

    'location': 'Atlantic Station',

    'rent': 1000,

    'number\_of\_roommates': 3,

    'House\_Layout': '2B2B',

    'House\_Type': 'Apartment'

}

In our output below, the location feature contains a missing value because both locations are the same, so it is excluded from the assessment. The rent feature shows no difference between the sample apartment and the average rent of the most similar apartments, while the percentage difference for the number of roommates, house layout, and house type features is negative, indicating that the sample apartment has higher values for these features than the most similar apartments.

Based on these results, we can conclude that the sample apartment is generally similar to the most similar apartments in terms of rent but has more roommates and a different house layout and type than the most similar apartments. This information could be useful for making decisions about pricing, marketing, or renovations for the sample apartment.

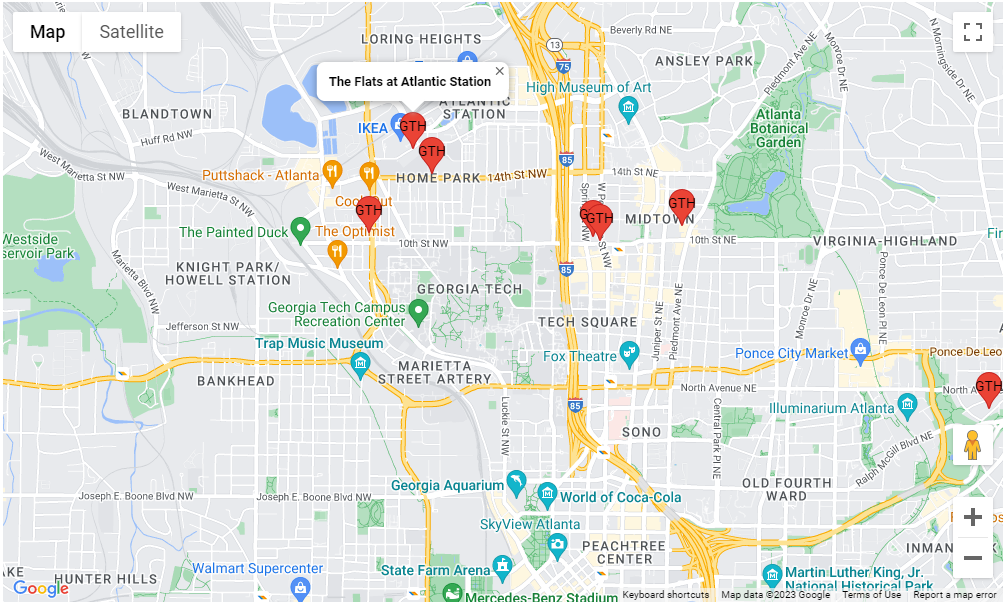
Output screenshots:

Text, letter

Description automatically generated

Hence, our recommendation is using Content based filtering by creating user profile for housing recommendations for students.

Sample Interface Output:



Unfortunately, we were not able to dedicate enough time to producing a production level front end because most of our resources were spent on developing the models since they are the backbone of our product. We were able to design a basic interface that displays the top 10 rated current apartment listings on a map for users to see, which is the minimum viable product of our project.

## **9. Customers and Revenue Streams**

As per the new addition to the Project Scope for the Spring 2023 semester, we aim to find potential customers and revenue streams for our project. The entrepreneurial aspect of the project will help us expand the scope of the project, include future scope of development/improvement, and focus on our customers' current needs.

**Potential Customers for University House Finder:**

The Project, “University House Finder” is designed to serve as an easy tool for students joining university to find housing options under one roof. The website aims to get input from potential students about the important factors that they look for while looking for housing and shows them the top matches for their preferences. Crime, proximity to campus, pricing, size of house etc are some of the important factors that every student considers while looking for housing. Especially international students, who move to a completely new country and have no knowledge of the area and pricing system in the USA, this tool will be a great insight into how the actual scenario of housing looks like around the campus.

Potential customers (people who can use the website) are as follows:

1. **Incoming students:** The website is primarily aimed at incoming students. Except for freshman students who mandatorily stay in the on-campus dorms, all other incoming students can find this website helpful to navigate through the housing hunt. Important criteria like “proximity to Department/Library/MARTA station,” “Crime Score,” “rent score” etc. can be used as filters to identify the best possible location for living around campus.
2. **Homeowners**: Individuals who have homes around the campus and make it available for leasing to students, can find this tool helpful as well. Several homeowners in Home Park for example, lease their homes to Georgia Tech students for lease duration of 1 school year. This could be helpful for students who wish to stay in fully furnished homes, and also near to the campus. With features like “list your property”, homeowners can easily put up their availabilities, for students to explore the option. Besides, some homeowners who allow students to stay as Paying Guests (PG) in their home in a room, can also find this tool useful to look for students who wish to stay alone, but in a home environment.
3. **Apartment Managers**: Several Apartments around the campus offer housing to students and some are quite popular among students as well. Such apartments can list their availability on the website, that could help students to look for the apartment easily. An “apartment score feature: can be added to the website based on the popularity of the apartment and the satisfaction of current students living in the apartment. In the google form circulated to current students, a question based on satisfaction of living at your current apartment is asked. This would help us give a score to the apartment based on the students' responses. If we do not get data for a particular apartment (due to no student living in the apartment filling out our form), we can get a satisfaction score for the nearby apartments and gauge a score for the area/street. This would be helpful to get an estimate of the relative popularity of the apartment.
4. **Shop and restaurant owners (Future Scope)**: As a part of the extension of our tool to accommodate several distinct aspects, restaurant owners like Moe’s or Tindrum can be potential customers of the website. As these restaurants are close to campus and popular among students due to taste and affordability, partnering with the restaurants to provide discount coupons to students who are registered on the website, could serve as a source of new customers and income for the restaurants, that in turn would provide some commission to the website.
5. **Current students (future scope):** Students graduating and moving out, or students moving to new cities for internships, can be customers for subleasing/transferring their lease through the website. Several apartments allow students to sublease, partnering with such apartments will help to increase the customer base and students can also find temporary housing from the website.

**Revenue Streams for Project**:

An application for housing suggestions for students at a university can have several revenue streams. and will depend on factors such as the target audience, the competition in the market, and the unique features of the application. The application serves as a marketplace for the students who are looking for housing options as well as the property managers and the individual homeowners who want to rent out their space and for other stakeholders.

The application has their target audience primarily as the students at Georgia institute of technology and the property owners / managers nearby the university. Moreover, there are several grocery shops and restaurant owners nearby who can benefit from the students living near their store / restaurants and can be seen as a potential customer for the application.

Here are some possible Revenue Streams and potential streams that can reap the application benefit in the future:

1. The first type of revenue stream model can be the Operating revenue stream which is as follows:

* **Subscription Model:** The application can offer premium features such as personalized housing suggestions, priority listings, or additional filters for a monthly or yearly subscription fee. The application also aims to provide the roommate matching algorithm which helps two individuals with a similar choice of living space to connect and plan to stay together. This along with some extra benefits like a higher number of housing suggestions per day, and possible contact with current students can be provided in return for a subscription fee.
* **Commission-Based Model**: The application can partner with real estate agents or landlords to promote their housing listings on the platform and take a commission on successful bookings or transactions.
* **Advertising Model**: The application can display targeted advertisements from local businesses such as moving companies, furniture rental services, or cleaning services. Advertisers can pay for ad placements or clicks. The application can also tie up with local restaurants and food outlets to place their ads to attract students staying on the property near to their location. This can even be a prospective tie-up between the property manager and restaurant owners.
* **Referral Model**: The application can offer referral bonuses or discounts to existing users who refer their friends or classmates to the platform.
* **Data Licensing Model**: The application can monetize the data collected from users' housing preferences and behavior by licensing it to third-party market research firms, real estate agents, or property developers.
* **Virtual Tour Model**: The application can offer virtual tours of available housing units through a partnership with real estate agents or property owners. Users can pay a fee to access these tours, which can save them time and money compared to physically visiting each property.
* **Data Analytics Model**: The application can use the data collected from users to provide insights and analytics to property developers, property owners, or government agencies. This can include information on housing demand, price trends, or student demographics.
* Premium Listing Model: The application can offer premium listing options for real estate agents or property owners, such as higher visibility on the platform or additional features such as 3D floor plans or video walkthroughs.
* **Integration Model**: The application can integrate with other related services such as moving companies or utility providers and take a commission on successful referrals or transactions. The application can tie up with Zipcar or enterprise car rental company and some movers and packer's companies to provide ease of access to the prospective users of the application and get commissions from successful availing of their services.
* **Sponsored Model**: The application can partner with other universities, student organizations, or other relevant stakeholders to promote the platform and offer sponsored housing options. This can provide additional exposure and legitimacy for the application, as well as potential revenue from sponsorships or partnerships.

1. The second type of revenue stream model can be the non-operating revenue model as follows:
2. The application owners can venture into buying some of the properties themselves and then suggest that properties on their application. This way they don’t have to hassle to partner with multiple property managers as well as they will get an additional rent revenue without spending much on advertisement.
3. The application can also earn interest/dividend revenue by investing in some of the profitable company's stocks and then earn interest on that.

## **10. Future Scope and Considerations**

The future of apartment finding tools such as the one we have created has few limitations. As modeling and machine learning techniques continue to evolve, we are witnessing a shift towards a future that is more “personalized” than ever. This same sentiment can be expanded to home-hunting, where there is clear room for further research and implementation in college cities across the country. Including more features that could contribute to the matching process is logical and would seek to improve the likelihood that a student is matched with an appropriate apartment.

Performing sentiment analysis can be used for suggesting apartments wherein we analyze the sentiments of online reviews, and feedback for apartments and perform the following steps. Performing sentiment analysis can help assign a score to each review based on the positivity or negativity associated with each review. Using sentiment analysis might thus help in ranking apartments based on the sentiment scores of the reviews.

An extension of the project can be the *Roommate Finder*- where along with finding apartments, students can also connect with people who are interested in similar types of housing as the candidate’s preferences. By implementing the roommate finder, students will not have to worry about finding roommates to share an apartment- especially the international students, that have little or no knowledge of the apartment system in USA. A limitation of this extension is the sharing of student information. To share data with prospective students who match, we need to ensure that the profiles created on the website are genuine and legitimate. This would require verification of students' profiles to match them with other students. A premium subscription that verifies the student profile and allows them enhanced access to information of other matching students can be implemented, that could benefit the students.

As explained in the Potential customers and revenue section, this tool can be extended to include restaurant owners and current students as future customers of the tool. By introducing suggestions and promotional offers to nearby restaurants, both students and owners can be benefited by availability of food and income source, respectively. Exclusive offers can be provided to users who have premium subscriptions to the website, that would enable the website owners to earn revenues.

Future Model Improvements

To achieve higher accuracy in predicting student housing preferences, several improvements can be made to the existing models and data processing techniques. Below are some suggestions for enhancing the predictive power of the models:

1. Feature Engineering: Investigate the possibility of creating new features or transforming existing features to better capture the underlying patterns in the data. For example, creating a 'rent per square foot' variable could reveal additional insights into students' preferences.
2. Feature Selection: Use techniques like Recursive Feature Elimination (RFE) or Lasso regularization to identify the most relevant features for the prediction task. By reducing the number of features, the risk of overfitting can be minimized, and model performance may improve.
3. Additional Data: Collect more data to increase the sample size, which can help to improve the model's performance by providing more information about the relationships between variables. Additionally, incorporating more diverse data points can ensure that the model generalizes well to various student housing preferences.
4. Imbalanced Data: If the dataset is imbalanced with an unequal distribution of housing preferences, consider using techniques such as oversampling the minority class or under sampling the majority class to balance the dataset. This can help the model to learn better representations of both classes, resulting in improved predictive performance.
5. Hyperparameter Tuning: Perform a systematic search for optimal hyperparameters using techniques like Grid Search or Randomized Search. This can help to fine-tune the models and achieve better performance by identifying the best combination of hyperparameters for the specific dataset.
6. Alternative Models: Explore other machine learning techniques, such as Neural Networks or XGBoost, which may be better suited for this specific prediction task. These methods may be able to capture more complex relationships in the data and provide higher accuracy.
7. Model Interpretability: Investigate the importance of features and the relationships between them by using techniques like Partial Dependence Plots (PDP) or SHapley Additive exPlanations (SHAP) values. Understanding the relationships between variables can help to guide further improvements in feature engineering and model selection.

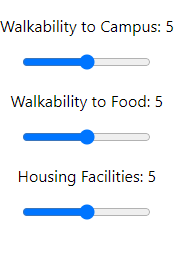
The following recommendations can serve as a future scope for upcoming batches to continue working on the same project:

They can work on a code using k-means clustering to recommend apartments to a user based on their preferences. They can create a user profile that contains their preferences for location, rent, number of roommates, house layout, and house type. Then use two or three features like rent, house type or layout to do clustering and perform k-means clustering with five clusters. Next, the code will find the cluster that is most similar to the user's input values and calculate a similarity score for each apartment based on their distance to the cluster centroid. The apartments are then filtered by the user's cluster and sorted by similarity score.

The upcoming batch can build upon this code to further enhance the apartment recommendation system. For instance, they can incorporate additional features such as the distance to public transportation, availability of parking space, and proximity to essential amenities like grocery stores and hospitals. They can also explore different clustering techniques such as hierarchical clustering and DBSCAN to identify more accurate clusters. Additionally, they can leverage machine learning models like decision trees and random forests to predict the user's preferred house layout and house type based on their historical preferences. These enhancements can significantly improve the recommendation system's performance and provide more personalized recommendations to users.

Future Interface Improvements

The core interface will remain the same as the map displaying the results from the backend model will be the center of this service. However, there needs to be an entire site built around this including things such as a navigation bar which houses interactions such as signing in, user profile and other relevant tabs. Additional functionality on the main screen could include preference input such as:

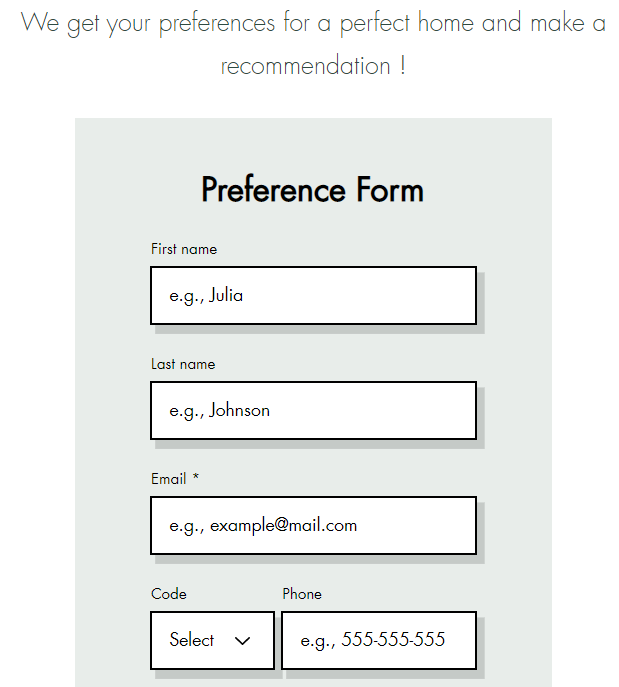


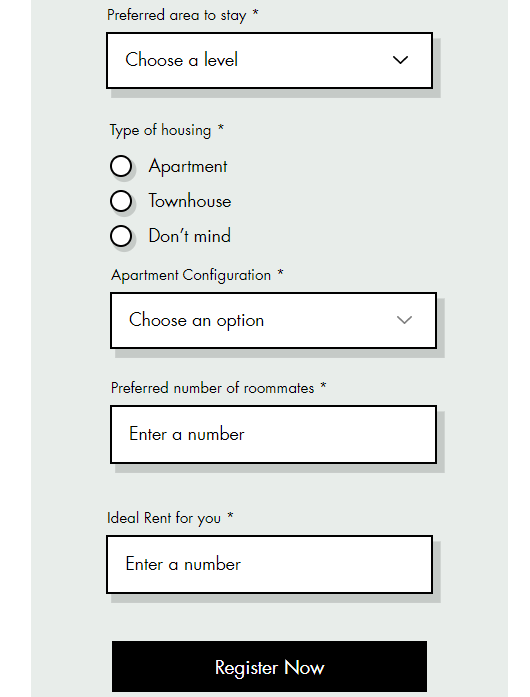
Where the user can rate their preference for various fields that the model analyzes. These preferences can be used to further adjust model weights on a per user basis to determine more personalized results. In addition, including the capability of users to create accounts on the site will help with the roommate finder and users that receive the same results as each other can be suggested as roommates.

## **11. Website Creation**

To fulfill the customer revenue streams, we created a nascent website which could potentially be used as a launcher for our application. The website asks for customer preference data and returns the top recommendations for the user as per the algorithm which can run in the backend.

The website which we present in this project could be bettered to offer more insightful data such as maps, previous testimonials but for the purposes of the project has been kept short and concise. We have added the link of the website to the report for viewing. The website was created using an online tool, which helps in making custom websites out of templates. The following are some of the screenshots of the website attached.





Website:

<https://shreyasak07.wixsite.com/unihousefinder>

## **12. Conclusion**

This semester project served as a hands-on experience in which we applied many data mining techniques that we learned throughout our studies in ISyE 7406: Data Mining and Statistical Methods. While applying said techniques we were able to create a tool to help students through a process that many of us are familiar with as both students of Georgia Tech and our previous institutions. The lack of a dedicated resource that was specifically aimed at students hunting for apartments close to their campuses was apparent and we aimed our efforts at filling this gap. This notion instilled a sense of purpose for our team that was attached to the project. From our initial iterations and ideas for the housing finder, we were able to hone our project and overcome the barriers that we encountered along the way. From having to alter the ways in which we collected the housing data, student data, and even amenity information to our selection of data mining algorithms, it was certainly not an easy task. Given our resource and temporal restrictions, we were still able to appropriately collect data and apply these data processing algorithms and techniques to train proper models, create a front-end display for output of the data mining processes, and leave room for future improvements and expansion.

## **13. Appendix**

**Output Resampling: Bootstrapping**

import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

from google.colab import files

data = pd.read\_csv('/content/Housing Form Edited.csv')

data = data.drop(columns=['Timestamp'], axis=1)

data.head(20)

data['Name Of Apartment/ Area of House?'].value\_counts()

sns.countplot(x='Name Of Apartment/ Area of House?', data = data)

plt.xticks(rotation=90, ha='center')

plt.show()

from pandas.tseries import frequencies

categorical\_columns = ['What Is Your Major?','What Type Of Housing Are You In?','Name Of Apartment/ Area of House?','How many roommates/ flatmates do you have?','How Much Is Your Individual Rent?','Rate Your Convenience for Commuting to College','Rate Your Housing Facilities (ie pool, gym etc.)','Rate Your Rent Value (Is it value for your money?)','Rate Your Apartment Safety','Rate Your Apartment Layout','Rate Proximity To Grocery Stores/ Restaurants','Does Your Apartment Allow Subleasing?','Please Rate Your Housing Overall']

num\_samples = 200 - len(data)

sampled\_data = pd.DataFrame(columns=data.columns)

for col in categorical\_columns:

  frequencies = data[col].value\_counts(normalize=True)

  samples = np.random.choice(frequencies.index, size = num\_samples, p=frequencies.values)

  new\_data = pd.DataFrame({col: samples})

  sampled\_data = pd.concat([sampled\_data, new\_data], axis=1)

extended\_data = pd.concat([data, sampled\_data], axis=1)

extended\_data = extended\_data.reset\_index(drop=True)

extended\_data.to\_csv("Output.csv", index=False)

files.download('Output.csv')

NewData = pd.read\_csv('/content/Output Form Edited.csv')

len(NewData)

for col in categorical\_columns:

  NewData[col].value\_counts()

  sns.countplot(x=NewData[col], data = NewData)

  plt.xticks(rotation=90, ha='center')

  plt.show()

**Association Rule Mining using Apriori**

!pip install apyori

# Importing the necessary libraries

import pandas as pd

from apyori import apriori

# Reading the student data into a Pandas dataframe

Output\_edited = pd.read\_csv('/content/Output\_edited.csv')

# Defining the list of attributes to consider for the Apriori algorithm

attributes = ['In which department are you studying at Georgia Tech?', 'Housing Type?', 'Housing Configuration?', 'Roommate information?', 'Rent you pay?', 'What is your Lease Term?','Food eating habits?']

# Creating a list of transactions where each transaction is a list of attribute values for a student

transactions = []

for i in range(len(Output\_edited)):

    transaction = []

    for attribute in attributes:

        transaction.append(str(Output\_edited.iloc[i][attribute]))

    transactions.append(transaction)

# Running the Apriori algorithm with a minimum support threshold of 0.1 and minimum confidence of 0.5

rules = apriori(transactions, min\_support=0.1, min\_confidence=0.5)

# Converting the rules iterator to a list

rules\_list = list(rules)

# Displaying the rules generated by the algorithm

for rule in rules\_list:

    print(rule)

for rule in rules\_list:

  itemset = list(rule.items)

  support = rule.support

  print(f"{itemset} - Support:{support}")

# Extracting the top 5 itemsets by support

top\_itemsets = sorted(rules\_list, key=lambda x: x.support, reverse=True)[:5]

for rule in top\_itemsets:

    itemset = list(rule.items)

    support = rule.support

    print(f"{itemset} - Support:{support}")

# Extracting the rules with the highest confidence

strong\_rules = sorted(rules\_list, key=lambda x: x.ordered\_statistics[0].confidence, reverse=True)[:5]

for rule in strong\_rules:

    antecedent = list(rule.ordered\_statistics[0].items\_base)

    consequent = list(rule.ordered\_statistics[0].items\_add)

    confidence = rule.ordered\_statistics[0].confidence

    print(f"{antecedent} => {consequent} - Confidence:{confidence}")

# Identifying the most frequent itemsets

frequent\_itemsets = []

for rule in rules\_list:

    itemset = list(rule.items)

    support = rule.support

    frequent\_itemsets.append((itemset, support))

# Sorting the frequent itemsets by support

frequent\_itemsets = sorted(frequent\_itemsets, key=lambda x: x[1], reverse=True)

# Displaying the 10 most frequent itemsets

print("10 Most Frequent Itemsets:")

for itemset, support in frequent\_itemsets[:10]:

    print(f"{itemset} - Support:{support}")

# Identifying the most interesting rules

interesting\_rules = []

for rule in rules\_list:

    itemset = list(rule.items)

    confidence = rule.ordered\_statistics[0].confidence

    lift = rule.ordered\_statistics[0].lift

    interesting\_rules.append((itemset, confidence, lift))

# Sorting the interesting rules by confidence

interesting\_rules = sorted(interesting\_rules, key=lambda x: x[1], reverse=True)

# Displaying the 10 most interesting rules

print("10 Most Interesting Rules:")

for itemset, confidence, lift in interesting\_rules[:10]:

    print(f"{itemset} -> Confidence:{confidence}, Lift:{lift}")

**Data Modeling**

import·math

#·Function·to·calculate·the·distance·using·the·haversine·formula

def·haversine(lat1,·lon1,·lat2,·lon2):

····lat1,·lon1,·lat2,·lon2·=·map(math.radians,·[lat1,·lon1,·lat2,·lon2])

····dlat·=·lat2·-·lat1

····dlon·=·lon2·-·lon1

····a·=·math.sin(dlat·/·2)·\*\*·2·+·math.cos(lat1)·\*·math.cos(lat2)·\*·math.sin(dlon·/·2)·\*\*·2

····c·=·2·\*·math.atan2(math.sqrt(a),·math.sqrt(1·-·a))

····earth\_radius·=·3958.8

····distance·=·earth\_radius·\*·c

····return·distance

#·Read·the·CSV·file·into·a·DataFrame

file\_path·=·'CleanOutput.csv'

df·=·pd.read\_csv(file\_path)

#·Georgia·Tech·Student·Center·coordinates

gt\_lat,·gt\_lon·=·33.774981,·-84.398066

#·Calculate·the·distance·for·each·row·and·add·it·as·a·new·column

df['distance']·=·df.apply(lambda·row:·haversine(gt\_lat,·gt\_lon,·row['latitude'],·row['longitude']),·axis=1)

#·Save·the·updated·DataFrame·to·a·new·CSV·file

output\_file\_path·=·'UpdatedOutput.csv'

df.to\_csv(output\_file\_path,·index=False)

[20]

import numpy as np  
import pandas as pd  
from sklearn.model\_selection import train\_test\_split, cross\_val\_score  
from sklearn.linear\_model import LinearRegression  
from sklearn.metrics import mean\_squared\_error, r2\_score  
  
# Sample dataset  
data = {  
 'area': [1500, 2000, 2500, 3000, 3500],  
 'bedrooms': [3, 4, 3, 5, 4],  
 'age': [10, 20, 15, 5, 2],  
 'rating': [4.0, 4.5, 4.2, 5.0, 4.8]  
}  
  
file\_path = 'HousingForm.csv'  
df = pd.read\_csv(file\_path)  
  
  
# Features (X) and target (y) extraction  
X = df[['Roommates', 'Rent', 'Footage', 'Distance']]  
y = df['Overall']  
  
# Train-test split  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)  
  
# Create and fit the model  
model = LinearRegression()  
model.fit(X\_train, y\_train)  
  
# Make predictions  
y\_pred = model.predict(X\_test)  
#print(y\_pred)  
  
print(f"Model Weights: {model.coef\_}")  
print(f"Model Bias: {model.intercept\_}")  
# Evaluate the model  
mse = mean\_squared\_error(y\_test, y\_pred)  
r2 = r2\_score(y\_test, y\_pred)  
  
print(f"Mean Squared Error: {mse:.2f}")  
print(f"R-squared: {r2:.2f}")  
  
# # Predict ratings for new users  
# new\_data = np.array([  
# [1800, 3, 8],  
# [2200, 4, 12]  
# ])  
  
df2 = pd.read\_csv('UpdatedOutput.csv')  
X\_new = df2[['Roommates', 'Rent', 'Footage', 'Distance']]  
new\_ratings = model.predict(X\_new)  
# print(f"Predicted Ratings: {new\_ratings}")  
# df2['Ratings'] = new\_ratings  
# df2.to\_csv('LR\_Ratings.csv')

Model Weights: [ 0.64680283 0.00164526 -0.00142137 0.07684151]  
Model Bias: 6.514764771449109  
Mean Squared Error: 1.94  
R-squared: -0.16

[21]

from sklearn.tree import DecisionTreeRegressor, DecisionTreeClassifier  
# Create and fit the model  
model = DecisionTreeRegressor()  
model.fit(X\_train, y\_train)  
  
# Make predictions  
y\_pred = model.predict(X\_test)  
  
# Evaluate the model  
mse = mean\_squared\_error(y\_test, y\_pred)  
r2 = r2\_score(y\_test, y\_pred)  
  
print(f"Mean Squared Error: {mse:.2f}")  
print(f"R-squared: {r2:.2f}")  
  
df2 = pd.read\_csv('UpdatedOutput.csv')  
X\_new = df2[['Roommates', 'Rent', 'Footage', 'Distance']]  
new\_ratings = model.predict(X\_new)  
print(f"Predicted Ratings: {new\_ratings}")  
# df2['Ratings'] = new\_ratings  
# df2.to\_csv('DT\_Ratings.csv')

Mean Squared Error: 1.73  
R-squared: -0.04  
Predicted Ratings: [10. 10. 8. 10. 10. 10. 8. 10. 10. 10. 8. 10. 10. 10.  
 10. 10. 10. 10. 10. 10. 10. 10. 5. 10. 8. 10. 10. 8.  
 8. 10. 10. 10. 8.5 10. 10. 8. 10. 8. 8. 10. 10. 8.  
 8. 8. 8.5 8. 8. 8. 10. 10. 8. 10. 8. 10. 10. 10.  
 10. 10. 5. 10. 8.5 10. 10. 8. 8. 8. 8. 8. 10. 8.5  
 10. 10. 7. 10. 10. 10. 10. 10. ]

[22]

# Decision Tree with cross-validation  
dt = DecisionTreeClassifier(random\_state=42)  
dt\_scores = cross\_val\_score(dt, X, y, cv=3) # 2-fold cross-validation  
dt\_mean\_score = dt\_scores.mean()  
print(f"Decision Tree mean accuracy: {dt\_mean\_score:.2f}")

Decision Tree mean accuracy: 0.29

[25]

from sklearn.ensemble import BaggingClassifier, GradientBoostingClassifier  
from sklearn.preprocessing import StandardScaler  
from sklearn.pipeline import make\_pipeline  
# Base model: Decision Tree  
base\_model = DecisionTreeClassifier(random\_state=42)  
  
# Bagging  
bagging = BaggingClassifier(estimator=base\_model, n\_estimators=100, random\_state=42)  
bagging\_scores = cross\_val\_score(bagging, X, y, cv=3) # 5-fold cross-validation  
bagging\_mean\_score = bagging\_scores.mean()  
print(f"Bagging mean accuracy: {bagging\_mean\_score:.2f}")  
  
# Boosting (using Gradient Boosting)  
boosting = GradientBoostingClassifier(n\_estimators=100, random\_state=42)  
boosting\_scores = cross\_val\_score(boosting, X, y, cv=3) # 5-fold cross-validation  
boosting\_mean\_score = boosting\_scores.mean()  
print(f"Boosting mean accuracy: {boosting\_mean\_score:.2f}")

Bagging mean accuracy: 0.37  
Boosting mean accuracy: 0.37

[15]

from sklearn.decomposition import PCA  
from sklearn.preprocessing import StandardScaler  
  
# Standardize the feature matrix  
scaler = StandardScaler()  
X\_scaled = scaler.fit\_transform(X)  
  
# Apply PCA  
pca = PCA(n\_components=2) # Reduce the dimensionality to 2  
X\_pca = pca.fit\_transform(X\_scaled)  
  
loadings = pd.DataFrame(pca.components\_, columns=X.columns, index=[f'PC{i+1}' for i in range(pca.n\_components\_)])  
print("Principal Components (Loadings):")  
print(loadings)  
  
# Display the transformed features  
# print("PCA transformed features:")  
# print(X\_pca)  
  
# Display the explained variance ratio  
print("\nExplained variance ratio:")  
print(pca.explained\_variance\_ratio\_)

Principal Components (Loadings):  
 Roommates Rent Footage Distance  
PC1 -0.616299 0.524959 -0.585267 0.045347  
PC2 -0.039752 0.089121 0.044765 -0.994220  
  
Explained variance ratio:  
[0.58941617 0.25063767]

[26]

from sklearn.svm import SVC  
from sklearn.metrics import classification\_report, confusion\_matrix  
  
  
df['Label'] = (df['Overall'] >= 8.0).astype(int)  
y = df['Label']  
# Standardize the feature matrix (both training and testing sets)  
scaler = StandardScaler()  
X\_train\_scaled = scaler.fit\_transform(X\_train)  
X\_test\_scaled = scaler.transform(X\_test)  
  
# Create and train the SVM classifier  
svm\_classifier = SVC(kernel='linear', C=1)  
svm\_classifier.fit(X\_train\_scaled, y\_train)  
  
# Make predictions on the test set  
y\_pred = svm\_classifier.predict(X\_test\_scaled)  
  
# Evaluate the classifier  
print("Confusion Matrix:")  
print(confusion\_matrix(y\_test, y\_pred))  
print("\nClassification Report:")  
print(classification\_report(y\_test, y\_pred))

Confusion Matrix:  
[[0 0 0 1 0 0]  
 [0 0 1 1 0 0]  
 [0 0 0 3 0 0]  
 [0 0 1 4 0 0]  
 [0 0 1 1 0 0]  
 [0 0 0 0 0 1]]  
  
Classification Report:  
 precision recall f1-score support  
  
 8 0.40 0.80 0.53 5  
 9 0.00 0.00 0.00 2  
 10 1.00 1.00 1.00 1  
  
 accuracy 0.36 14  
 macro avg 0.23 0.30 0.26 14  
weighted avg 0.21 0.36 0.26 14

[27]

from sklearn.pipeline import make\_pipeline  
# SVM with cross-validation  
svm = make\_pipeline(StandardScaler(), SVC(random\_state=42)) # StandardScaler is used to normalize the features  
svm\_scores = cross\_val\_score(svm, X, y, cv=3) # 5-fold cross-validation  
svm\_mean\_score = svm\_scores.mean()  
print(f"SVM mean accuracy: {svm\_mean\_score:.2f}")

SVM mean accuracy: 0.68

[28]

# Base model: Decision Tree  
base\_model = SVC(random\_state=42)  
  
# Bagging  
bagging = BaggingClassifier(estimator=base\_model, n\_estimators=100, random\_state=42)  
bagging\_scores = cross\_val\_score(bagging, X, y, cv=3) # 3-fold cross-validation  
bagging\_mean\_score = bagging\_scores.mean()  
print(f"Bagging mean accuracy: {bagging\_mean\_score:.2f}")  
  
# Boosting (using Gradient Boosting)  
boosting = GradientBoostingClassifier(n\_estimators=100, random\_state=42)  
boosting\_scores = cross\_val\_score(boosting, X, y, cv=3) # 3-fold cross-validation  
boosting\_mean\_score = boosting\_scores.mean()  
print(f"Boosting mean accuracy: {boosting\_mean\_score:.2f}")

Bagging mean accuracy: 0.66  
Boosting mean accuracy: 0.62

[17]

from sklearn.ensemble import RandomForestClassifier

# Split the dataset into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

# Create and train the Random Forest classifier

rf\_classifier = RandomForestClassifier(n\_estimators=100, random\_state=42)

rf\_classifier.fit(X\_train, y\_train)

# Make predictions on the test set

y\_pred = rf\_classifier.predict(X\_test)

# Evaluate the classifier

print("Confusion Matrix:")

print(confusion\_matrix(y\_test, y\_pred))

print("\nClassification Report:")

print(classification\_report(y\_test, y\_pred))

Confusion Matrix:  
[[ 3 6]  
 [ 1 11]]  
  
Classification Report:  
 precision recall f1-score support  
  
 0 0.75 0.33 0.46 9  
 1 0.65 0.92 0.76 12  
  
 accuracy 0.67 21  
 macro avg 0.70 0.62 0.61 21  
weighted avg 0.69 0.67 0.63 21

**Assessing Quality of Location Data obtained from API Pull**

import requests

import json

1)Define API endpoint and parameters

url = "<https://api.example.com/location>"

params = {

"lat": 37.7749,

"lon": -122.4194,

"radius": 1000

}

2)Send request to API and parse response

response = requests.get(url, params=params)

data = json.loads(response.text)

3)Verify accuracy of location data

if data["latitude"] == params["lat"] and data["longitude"] == params["lon"]:

print("Location data is accurate")

else:

print("Location data is inaccurate")

4)Check completeness of location data

if "address" in data and "city" in data["address"] and "postal\_code" in data["address"]:

print("Location data is complete")

else:

print("Location data is incomplete")

6) Check consistency of location data

# Assume we have another data source called "geo" that provides the same location information

geo = {"latitude": 37.7749, "longitude": -122.4194, "address": {"city": "San Francisco", "postal\_code": "94102"}}

if data == geo:

print("Location data is consistent with other sources")

else:

print("Location data is inconsistent with other sources")

7)Check relevance of location data

if data["business\_type"] == "restaurant":

print("Location data is relevant to restaurant analysis")

else:

print("Location data is not relevant to restaurant analysis")

8)Verify timeliness of location data

if data["last\_updated"] > "2022-01-01":

print("Location data is up-to-date")

else:

print("Location data is outdated")

9)Check ethical considerations

# Assume we have obtained consent to use the location data for our analysis

if data["consent"] == True:

print("Ethical considerations have been met")

else:

print("Ethical considerations have not been met")

**Code for Content Based Filtering:**

# Content Based Filtering

import pandas as pd

from sklearn.feature\_extraction.text import CountVectorizer

from sklearn.metrics.pairwise import cosine\_similarity

apartment\_data = pd.read\_csv('/content/Housing Form Edited for Content Based Filtering.csv')

user\_profile = {

    'location': 'Atlantic Station',

    'rent': 900,

    'number\_of\_roommates': 3,

    'House\_Layout': '2B2B',

    'House\_Type': 'Apartment'

}

def create\_apartment\_profile(apartment):

    return ' '.join([

        apartment['location'],

        str(apartment['rent']),

        str(apartment['number\_of\_roommates']),

        apartment['House\_Layout'],

        apartment['House\_Type']

    ])

apartment\_data['profile'] = apartment\_data.apply(create\_apartment\_profile, axis=1)

vectorizer = CountVectorizer().fit\_transform(apartment\_data['profile'])

similarity\_scores = cosine\_similarity(vectorizer)

# Create a CountVectorizer object

vectorizer = CountVectorizer()

# Fit the vectorizer to the apartment descriptions

vectorizer.fit(apartment\_data['profile'])

# Transform the apartment descriptions to a matrix of token counts

apartment\_matrix = vectorizer.transform(apartment\_data['profile'])

# Transform the user profile to a matrix of token counts

user\_vector = vectorizer.transform([create\_apartment\_profile(user\_profile)])

# Compute cosine similarities between the user profile and the apartment matrix

user\_scores = cosine\_similarity(user\_vector, apartment\_matrix)[0]

# Sort apartments by cosine similarity scores in descending order

recommended\_apartments = apartment\_data.iloc[user\_scores.argsort()[::-1]]

print(recommended\_apartments.head())

# Property Assessment

assessment\_features = [ 'location',

    'rent',

    'number\_of\_roommates',

    'House\_Layout',

    'House\_Type']

# Select a sample apartment for assessment

apartment = apartment\_data.sample()

# Create an apartment profile for assessment

apartment\_profile = create\_apartment\_profile(apartment.iloc[0])

# Transform the apartment profile to a matrix of token counts

apartment\_vector = vectorizer.transform([apartment\_profile])

# Calculate the cosine similarities between the sample apartment and all other apartments

similarities = cosine\_similarity(apartment\_vector, apartment\_matrix)

# Get the similarity scores for all apartments except for the sample apartment

similarity\_scores = similarities[0, :].tolist()

similarity\_scores.pop(apartment.index[0])

# Get the assessment feature values for the sample apartment and convert them to numerical types

assessment\_values = apartment[assessment\_features].apply(pd.to\_numeric, errors='coerce').values.tolist()[0]

# Calculate the average feature values of the most similar apartments and convert them to numerical types

similar\_apartment\_features = apartment\_data.loc[similarity\_scores.index(max(similarity\_scores)), assessment\_features].apply(pd.to\_numeric, errors='coerce').tolist()

# Calculate the percentage difference between the sample apartment and the average features of the most similar apartments

assessment\_results = [(similar\_apartment\_features[i] - assessment\_values[i])/assessment\_values[i]\*100 for i in range(len(assessment\_features)) if not pd.isna(similar\_apartment\_features[i]) and not pd.isna(assessment\_values[i])]

# Print the assessment results

for i in range(len(assessment\_results)):

    print(f'{assessment\_features[i]}: {assessment\_values[i]}')

    print(f'Average {assessment\_features[i]} of most similar apartments: {similar\_apartment\_features[i]}')

    print(f'Percentage difference: {assessment\_results[i]:.2f}%')

**Converting CSV to JSON data for visual display:**

import csv

import json

# Replace 'file.csv' with the path to your CSV file

csv\_file\_path = 'DT\_Ratings.csv'

# Read the CSV file and convert it into a list of dictionaries

csv\_data = []

with open(csv\_file\_path, 'r') as csvfile:

reader = csv.DictReader(csvfile)

for row in reader:

csv\_data.append(row)

# Convert the list of dictionaries into a JSON string

json\_data = json.dumps(csv\_data, indent=2)

# Print the JSON data

print(json\_data)

# Replace 'output.json' with the desired path for the output JSON file

output\_file\_path = 'results.json'

with open(output\_file\_path, 'w') as jsonfile:

json.dump(csv\_data, jsonfile, indent=2)

## **14. References**

[1] Mark Talabis, Jason Martin, Chapter 3 - Information Security Risk Assessment: Data Collection, Information Security Risk Assessment Toolkit, Syngress, 2013, Pages 63-104, ISBN 9781597497350

[2] Baum, A. (2017). PropTech 3.0: the future of real estate. Tech. rep., Saïd Business School, University of Oxford, Oxford, publisher: University of Oxford

[3] Jacobsen, C.; Monteiro, J. Real Estate: Undergraduates’ Characterization of Renting Versus Buying. *J. Bus. Manag. Admin.* **2019**, *1*, 1–5.

[4] Ciuriak, D. (2018). The Economics of Data: Implications for the Data-driven economy. Publisher: Chapter.