

COMPANY SUCCESS BASED ON EMPLOYEES' WORK LIFE ENVIRONMENT

Team Name: Hip Hip Array! -> ['Hip', 'Hip']

Who your group is: Phuong Truong; pttruong, Tony Wu; twu18, Cynthia Vu; cvu

Where is your Blog:

Blog1: https://medium.com/@phuong_truong/project-f244587c392a

Blog2: https://medium.com/@phuong_truong/etl-extract-transform-load-database-32b18be7d40b

Blog3: https://medium.com/@phuong_truong/statistical-exploration-8e969fff3311

Blog4: https://medium.com/@phuong_truong/using-voting-classifier-to-evaluate-employee-happiness-e3504b69858e

Blog5: https://medium.com/@phuong_truong/calculating-employee-happiness-web-interface-b4affb7b1719

Blog 6: https://medium.com/@phuong_truong/interactive-application-960b3b2f497c

Blog7: https://medium.com/@phuong_truong/final-post-bef0f9584519

Where is your git repo: <https://github.com/pttruong2009/Data1030Project>

Database login: mLab.com, user: cavu17, password: KevinandM@ngos1

Vision:

- What was your "big idea"?
- Did you make measurable progress towards this idea or did you find other interesting things?

Our big question was the following: Can we predict an employee's satisfaction with their job based on their company and work environment? Some factors we were looking for include culture rating, weather, crime rate in the city, company growth etc.

When coming up with this question, we wanted to find a topic that our diverse cohort of classmates could relate to. With that thought in mind, we realized that most of the students in our program are planning to join the workforce next year. Everyone will have their own unique experience and we wanted to see if the quality of the experience could have an effect on their respective company as a whole.

Our question revolves around how work life environment factors, such as CEO approval percentage and culture rating on Glassdoor, could an employee's willingness to recommend the job to a friend. We are using this as a metric for job satisfaction because we assume that if a

person was willing to recommend the job, they are satisfied with it and vice versa. In addition, we incorporated external environmental factors such as weather data and commuting times at a company's headquarter location to see how these factors can affect the company as since rainfall, snowfall and commute times often relate to employee's satisfaction. We evaluated these aspects in relation to the headquarters under the assumption that most employees would be working at the headquarters of the company location. In general, we aimed to identify and analyze different aspects that employees encounter everyday and inherently will contribute to their experience.

We did make measurable progress towards these ideas. We were able to see which states and cities had a slight edge in changing the probability of the employee recommending the job to a friend. In addition, we were able to determine that the culture rating was the feature that impacted the willingness to recommend the job to the friend the most.

Data:

- What data did you use?
- Relative to its size was there enough information contained within it?
- Were you able to find what you wanted in the data?
- How did you collect, clean, and integrate it, store it?

Our project requires data from multiple sources. We accumulated company data which includes friend recommendation and culture ratings from Glassdoor, weather temperatures and conditions along with work commute times from eldoradocountyweather.com and project.wnyc.org respectively. As a result of continuously pulling data from those sites and adding constantly new attributes, our SQL schema changes often. In addition, we scraped the location data of the headquarters from Wikipedia and Indeed.

Due to this fluctuating structure, we decided to put our data into a MongoDB database before migrating a panda dataframe for aggregation and analyses. Currently the data is organized in a JSON structure with 7 collections within the MongoDB database named 'final_project' in mLab using the 500MB free trial. The following lists the collections and their respective attributes:

- companies (CEO name, % CEO approval, % CEO disapproval, company name, compensation rating, culture rating, overall rating, rating description, recommendation to a friend rating, senior leadership rating, work life balance rating, headquarter city, headquarter state, and growth, annual rainfall, annual snowfall, headquarter location, population, and crime rate per 1000)
- city_commute (Zip code, census display label, city, state, commute time, and margin of error)
- glass_door_all_backup (raw data scraped from API)
- state_rainfall (State, rain in inches, rain in millimeters, ranking in terms of most rain, state abbreviation)
- snowfall_data (days, state, place, inches, centimeter, state abbreviation)
- crime_data (state, city, population, crime rate per 1000 people,, state abbreviation)
- diversity_data (overall rank, city, state, total score, socioeconomic diversity ranking, culture diversity rank, economic diversity rank, household diversity rank, religious diversity rank)

NAME	DOCUMENTS	CAPPED?	SIZE
city_commute	33,120	false	8.61 MB
companies	1,069	false	1.07 MB
crime_data	9,579	false	2.47 MB
diversity_data	502	false	266.86 KB
glassdoor_all_backup	98,827	false	197.51 MB
snowfall_data	50	false	19.70 KB
state_rainfall	50	false	14.58 KB

The follow documents how we pulled data from each individual site:

Glassdoor:

Glassdoor has an API which allows us to make GET requests to retrieve the company data in JSON format with their respective attributes.

```
{
  "success": true,
  "status": "OK",
  "sessionId": "12C83CECE5BDBE76BF4E0BDA4516EA50",
  "response": {
    "attributionURL": "https://www.glassdoor.com/Reviews/company-reviews.htm",
    "currentPageNumber": 1,
    "totalNumberOfPages": 36381,
    "totalRecordCount": 363808,
    "employers": [
      {
        "id": 715,
        "name": "Walmart",
        "website": "www.walmart.com",
        "isEEP": true,
        "exactMatch": false,
        "industry": "",
        "numberOfRatings": 31380,
        "squareLogo": "https://media.glassdoor.com/sql/715/walmart-squarelogo-1408483474312.png",
        "overallRating": "3.2",
        "ratingDescription": "OK",
        "cultureAndValuesRating": "3.1",
        "seniorLeadershipRating": "2.7",
        "compensationAndBenefitsRating": "3.2",
        "careerOpportunitiesRating": "3.3",
        "workLifeBalanceRating": "2.9",
        "recommendToFriendRating": 56,
        "featuredReview": {
          "attributionURL": "https://www.glassdoor.com/Reviews/Employee-Review-Walmart-RV6773399.htm",
          "id": 6773399,
          "currentJob": false,
          "reviewDateTime": "2015-06-02 07:55:18.903",

```

By using Python's urllib3 and numpy libraries, we were able to retrieve each entry and store them within a dataframe. Glassdoor stores the company data in pages. We were able to parse through 200 pages before reaching their read limit. In order to get around this issue, we decided to throttle our API requests by only calling to 200 pages each time. Overall, we traversed through 10,000 pages. We collected a list of dictionaries of all the companies and inputted that raw data into the database as a backup collection in mLab called `glass_door_all_backup`.

To work with this data, we transformed the Glassdoor data in a panda dataframe and csv for us to map with our other data sources. This merged data source is stored in a collection called "companies," which later becomes the file in which we used to train our machine learning algorithms. The following are the attributes collected in Glassdoor: company name, culture rating, compensation rating, overall rating, rating description, recommendation to a friend rating, senior leadership rating, work life balance rating, CEO name, CEO percent of approval, and CEO percent of disapproval.

Weather:

Temperature, rainfall, and snowfall data for each state is available on www.eldoradocountyweather.com. We simply downloaded their data that goes back 30 years beginning in 2010.

Commute Time:

Average commute time based on zip code is available on project.wnyc.org. After downloading the data, we mapped the zip code to the appropriate city and state using the database available on www.unitedstateszipcodes.org.

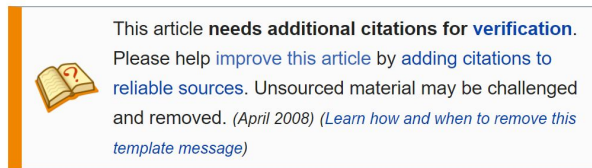
Location:

Unfortunately, the Glassdoor API did not provide us with the headquarter location during our web scraping, so we needed to look elsewhere. We ended up pulling this information from two sources: Wikipedia and Indeed

1. Wikipedia has a page titled 'List of corporate headquarters by city' which links to pages that lists companies in those respective cities. An example is included below:

List of companies in Greater Cincinnati

From Wikipedia, the free encyclopedia



This is a **list of major companies and organizations in Greater Cincinnati**, through corporate or subsidiary headquarters or through significant operational and employment presence near [Cincinnati, Ohio, USA](#).

Altogether, nine Fortune 500 companies and fifteen Fortune 1000 companies have headquarters in the Cincinnati area. With nine Fortune 500 company headquarters in the area,



We parsed through each of those pages to retrieve the names and headquarter locations of the companies.

2. The list from Wikipedia, although contained many companies, did not have enough values for us to map to a sufficient number of companies. We then decided to scrape Indeed's company profiles. Since Indeed did not have an explicit list of all the companies on the website, we had to get creative. In the search bar, we noticed that if you input only a letter such as "a", it would return you 1000 results that had an "a" as a starting letter in any part of the company name. We decided to parse through 999 pages because the first page has a different format than the rest. For example, "a" would return "Chick-fil-A," as well as "Amazon". By iterating through the alphabet, we were able to collect what we believe to be a complete list of the companies available on Indeed. After we collected a unique list of companies, we parsed through each website to grab their headquarter information that was usually available on the sidebar. If the information wasn't there, we added a null value so we could later remove it from the list to continue to map to the companies.

```

import requests
from bs4 import BeautifulSoup
r = requests.get('https://www.indeed.com/cmp?')
final = []
lst = ['a','b','c','d','e','f','g','h','i','j','k','l','m','n','o','p','q','r','s','t','u','v','w','x','y','z']
for item in lst:
    print(item)
    for page in range(2,100):
        r = 'https://www.indeed.com/cmp?q=' + item + '&p=' + str(page)
        r = requests.get(r)
        soup = BeautifulSoup(r.text, 'lxml')
        try:
            soup1 = soup.find_all('div')[4]
        except:
            break
        for i in range(0,9):
            try:
                soup2 = soup1.find_all(class_="company_result_title")[i]
                final.append(soup2.a['href'])
            except:
                break

```

Code showing how we parsed through each letter in the search bar. From here, we specifically got the latter part of the URL of each company page seen here:

```

['/cmp/A--line-Staffing-Solutions',
'/cmp/Broadspire,-A-Crawford-Company',
'/cmp/A.-Duie-Pyle,-Inc.',
'/cmp/A-Place-For-Mom',
'/cmp/A.-O.-Smith-Corporation',
'/cmp/The-Net--a--porter-Group',
'/cmp/Sixt-Rent-A-Car',
'/cmp/Pret-A-Manger',
'/cmp/Fox-Rent-A-Car',
'/cmp/Hot-Dog-On-A-Stick',
'/cmp/Save-A-Lot-Food-Stores',
'/cmp/The-Door---A-Center-of-Alternatives',
'/cmp/A-New-Leaf',
'/cmp/A--line-Staffing',
'/cmp/A-First-Name-Basis',
'/cmp/Dgi-Supply,-A-Doall-Company',
'/cmp/Make-A-Wish-Foundation-of-America',
'/cmp/Dollar-Rent-A-Car',
'/cmp/Don-S-Loch'

```

After that, we parsed through the unique company names (red box below) using the links to scrap the data found in the side box (blue box below)

The screenshot shows a web browser displaying the Indeed page for Amazon. The address bar at the top shows the URL `https://www.indeed.com/cmp/Amazon.com`, with the text `cmp/Amazon.com` highlighted by a red rectangular box. Below the address bar, the page header includes navigation links: **About**, **Reviews** (19.2K), **Salaries** (58.6K), **Photos** (43), **Jobs** (7.5K), and **Q&A** (1.5K). The Amazon logo is prominently displayed on the right, along with a blue **Follow** button. The main content area is titled **Working at Amazon** and features a video player showing a woman speaking. To the right of the video is a **Claimed Profile** section, outlined with a blue border, which contains the following information:

- Headquarters:** Seattle WA, USA
- Revenue:** more than \$10B (USD)
- Employees:** 10,000+
- Industry:**

Culture Rating:

Using the same method above, we were able to scrape the culture rating of each company from Indeed as well to diversify our ratings. We wanted to be able to confer with another source rather than use Glassdoor for everything.

Crime data:

For the data, we downloaded the excel file from the FBI's Uniform Crime Reporting program, the largest public database for crime statistics in the United States. We downloaded the excel file provided for crime in 2016, from here:

<https://ucr.fbi.gov/crime-in-the-u.s/2016/crime-in-the-u.s.-2016/tables/table-6/table-6.xls/view>.

We then added an extra column to normalize the data within each city by calculating the crime per 1000 people using the formula: $(\text{crime}/\text{population}) \times 1000$.

Diversity Data:

For the diversity data, we simply downloaded the tables from

<https://wallethub.com/edu/most-diverse-cities/12690/>. We focused our attention mainly on the total score column for our project.

Merging Data:

After acquiring our datasets, we noticed that the format of data was all over the place. In some datasets, company names were displayed with "Corporation", "Inc" and etc. at the end, and in other datasets company names without those aspects. For example, one dataset would contain "Nike Inc." whereas in others we saw "Nike". Since there were no unique column identifiers available in our datasets, in order to merge everything, we first had to create the unique identifier.

In terms of the Glassdoor and Indeed datasets, the unique identifier would be company name and we would enforce a specific name format for the datasets.

In terms of the weather and commute time datasets, we made the assumption that all companies within the same state would see the same amount of rainfall, snowfall and would have the same average commute time to work. Since we made this assumption, we were able to map the weather and commute time on state name of the company.

How we created the company name format:

We wrote a simple equation in excel to help us strip non-letters from the names of the companies. An example of stripping ".", and "," is shown below:

=IFERROR(REPLACE(H2,SEARCH(".",H2,1),1,""),H2)	
I	J
Stripping "."	Stripping ","
Agilent Technologies Inc	Agilent Technologies Inc
American Airlines Group	American Airlines Group
Advance Auto Parts	Advance Auto Parts
Apple Inc	Apple Inc
AbbVie	AbbVie
AmerisourceBergen Corp	AmerisourceBergen Corp
Abbott Laboratories	Abbott Laboratories

After, we then stripped the filler parts of a company's name, such as, "Inc","Corp","LLP", and etc. For simplicity's sake, we implemented another quick excel formula:

=IF(OR(RIGHT(J24,3)="Inc",RIGHT(J24,3)="Ltd",RIGHT(J24,3)="plc",RIGHT(J24,3)="Co."),LEFT(J24,LEN(J24)-4),J24)						
J	K	L	M	N	O	
Stripping ","	Stripping 3 letter - "Inc"	Stripping 4 letter - "Corp"	Stripping "Group"	Strip "Co"	Strip "&" at end	
Assurant Inc	Assurant	Assurant	Assurant	Assurant	Assurant	
Arthur J Gallagher & Co.	Arthur J Gallagher &	Arthur J Gallagher &	Arthur J Gallagher &	Arthur J Gallagher &	Arthur J Gallagher	
Akamai Technologies Inc	Akamai Technologies	Akamai Technologies	Akamai Technologies	Akamai Technologies	Akamai Technologies	
Albemarle Corp	Albemarle Corp	Albemarle	Albemarle	Albemarle	Albemarle	
Alaska Air Group Inc	Alaska Air Group	Alaska Air Group	Alaska Air	Alaska Air	Alaska Air	
Allstate Corp	Allstate Corp	Allstate	Allstate	Allstate	Allstate	
Allegion	Allegion	Allegion	Allegion	Allegion	Allegion	
Alexion Pharmaceuticals	Alexion Pharmaceuticals	Alexion Pharmaceuticals	Alexion Pharmaceuticals	Alexion Pharmaceuticals	Alexion Pharmaceuticals	
Applied Materials Inc	Applied Materials	Applied Materials	Applied Materials	Applied Materials	Applied Materials	
AMETEK Inc	AMETEK	AMETEK	AMETEK	AMETEK	AMETEK	
Affiliated Managers Group	Affiliated Managers Group	Affiliated Managers Group	Affiliated Managers	Affiliated Managers	Affiliated Managers	
Amgen Inc	Amgen	Amgen	Amgen	Amgen	Amgen	
Ameriprise Financial	Ameriprise Financial	Ameriprise Financial	Ameriprise Financial	Ameriprise Financial	Ameriprise Financial	

Once we established a common format for company name within each dataset, we were able to map all of our collected data together.

Post-Exploratory Statistics:

The "Recommend to a Friend" rating is on a scale of 1 to 100, with 1 being the absolute lowest score and extreme conclusion of *would not recommend their company to a friend* and 100 being the absolute highest score and extreme conclusion of *would strongly recommend their company to a friend*.

In our dataset, the lowest “Recommend to a Friend” Rating is 7, held by the companies Affinia Group and Tea Collection. The highest “Recommend to a Friend” rating is 100, held by a total of 8 companies.

We recognize that in our situation, the number of people giving reviews of the company is very significant. Surely if a company only has one review, that review only delineates that one person’s perspective, which may not accurately depict the larger population.

To combat this, for the exploratory statistics, we filtered that data and only considered the companies that had more than 500 total reviews or recommendations. See our quick filter below:

Custom AutoFilter

Show rows where:

Number of Recommenders

is greater than 500

☒ And ☐ Or

Use ? to represent any single character
Use * to represent any series of characters

OK Cancel

R	S	T	U	V	W	X	Y	Z	AA	AB
HQ_Sta	populat	crime_r	diversit	Number of Recommenders						
Texas	851849	71.38302	67.38	2559						

Running a quick pivot table of our data, we immediately came upon some relevant patterns. Firstly, it is quite apparent that the culture rating of a company from glassdoor, has a very strong positive correlation with the “recommendation to a friend” rating. Here are the companies that received “recommendation to a friend” ratings **between 7 and 20** (bottom 10%):

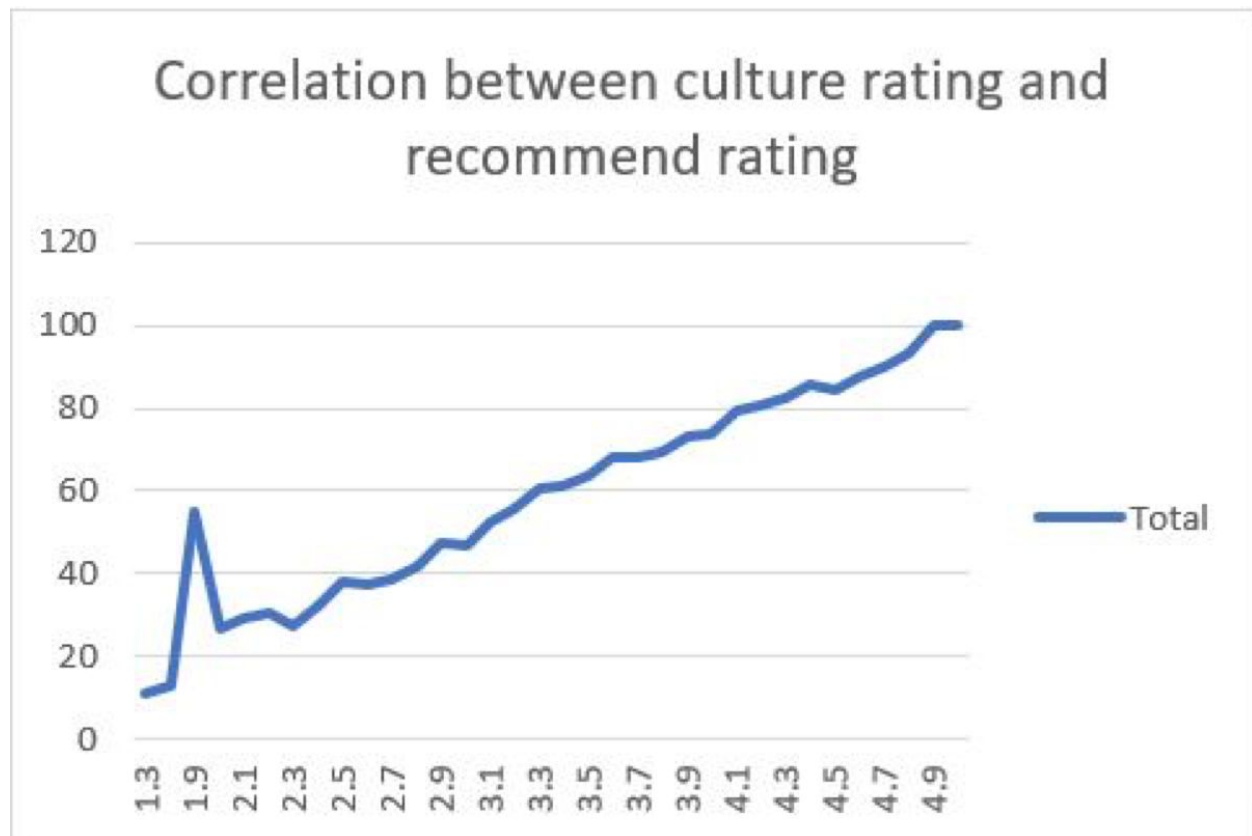
recommendToFriendRating		(Multiple Items) ▾		
company_name ▾	HQ_State ▾	HQ_City ▾	culture_rating ▾↑	
☐ Affinia Group	☐ Michigan	☐ Lansing	2.4	
☐ Bulova	☐ New York	☐ New York	2.2	
☐ Cavalier Telephone	☐ Virginia	☐ Richmond	3	
☐ CKE Restaurants	☐ Tennessee	☐ Franklin	2.1	
☐ Fitch	☐ Illinois	☐ Chicago	2.7	
☐ Iconix Brand Group	☐ New York	☐ New York	2.6	
☐ ImpreMedia	☐ New York	☐ New York	2.3	
☐ Integrated Electrical Services	☐ Texas	☐ Houston	2.3	
☐ MF Global	☐ New York	☐ New York	3	
☐ National Airlines	☐ Michigan	☐ Lansing	1.4	
☐ Prometheus Global Media	☐ New York	☐ New York	2.4	
☐ RadioShack	☐ Texas	☐ Fort Worth	2.4	
☐ Tailored Brands	☐ Texas	☐ Houston	2.4	
☐ Tea Collection	☐ California	☐ San Francisco	2.7	
☐ Technorati	☐ California	☐ San Francisco	2.3	
☐ Wenner Media	☐ New York	☐ New York	2.6	
☐ Wyman-Gordon	☐ Massachusetts	☐ Boston	1.3	
		AVERAGE	2.358823529	
		MEDIAN	2.4	
		MINIMUM	1.3	
		MAXIMUM	3	

Now let's look at some of the companies that received the highest "recommendation to a friend" (top 10%) ratings, **between 85–100**:

recommendToFriendRating	(Multiple Items)				
company_name	HQ_State	HQ_City	culture_rating		
Acumatica	Washington	Kirkland	4		
Allianz Life	Minnesota	Golden Valley	4.3		
Altair Engineering	Michigan	Lansing	3.9		
AppDynamics	California	San Francisco	4.3		
Associated Press	New York	New York	4.1		
Automattic	California	San Francisco	4.6		
B&H Photo Video	New York	New York	4.2	AVERAGE	4.329333333
Bain & Company	Massachusetts	Boston	4.7	MEDIAN	4.3
Bain Capital	Massachusetts	Boston	4	MINIMUM	3.3
Broadleaf Commerce	Texas	Dallas	5	MAXIMUM	5
CenterPoint Energy	Texas	Houston	4.1		
Ceridian	Minnesota	Bloomington	4.3		
Chronicle Books	California	San Francisco	4.7		
Clustrix	California	San Francisco	4.1		
Commonwealth Financial Net	California	San Diego	4.5		
Compare.com	Virginia	Richmond	5		
Conductor	New York	New York	4.8		
Delta Air Lines	Georgia	Atlanta	4.3		
DocuSign	California	San Francisco	4.6		
Dominion Energy	Virginia	Richmond	3.9		
dotloop	Ohio	Cincinnati	4.7		
DoubleClick	New York	New York	4.1		
Evercore Partners	New York	New York	4.4		

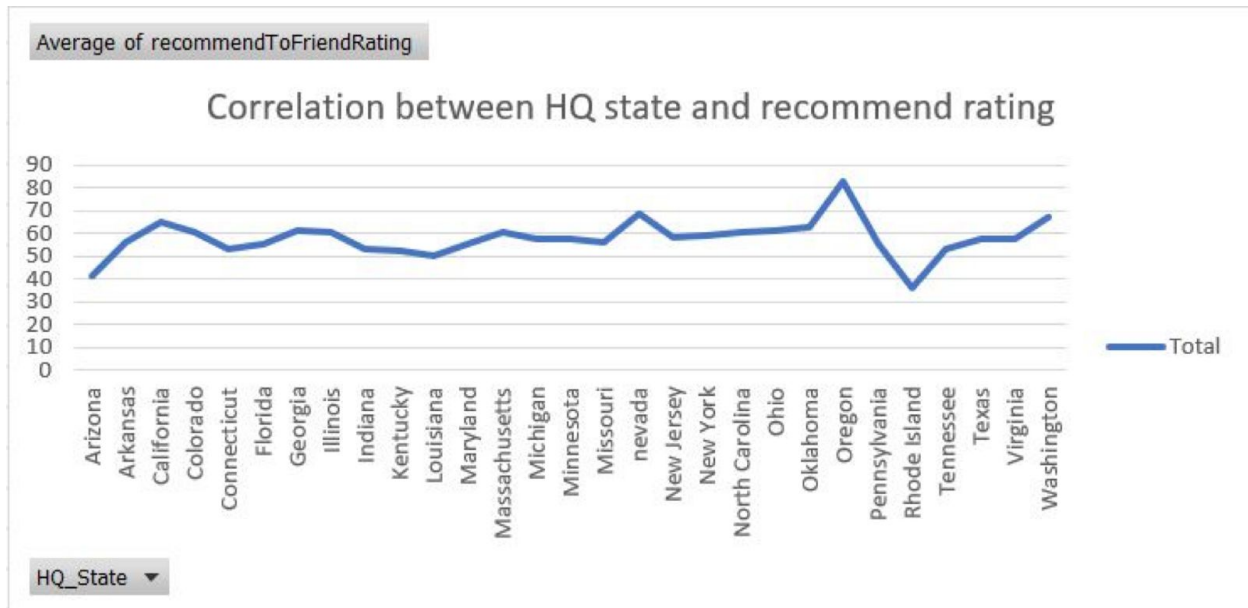
From our data, we see that the average “recommendation to a friend” rating increased drastically as the culture rating increased. This makes sense, since that if an employee is rating the culture of their company highly, they are most likely happy with where they work and therefore would highly likely also recommend their company to a friend.

Here is a graph of their relationship:



Once again, you can see the positive correlation that was previously mentioned. No cultural ratings of above 4.5 produced recommendation ratings less than 80.

Next, we made a graph to see if there were any significant patterns between the headquarter state and the recommendation rating:



We noticed that there really wasn't too much going on between state and recommendation rating. Oregon seemed to slightly edge out some of the other states with the highest average recommendation rating, while Arizona and Rhode Island seemed to have the lowest average recommendation ratings. Other than those states, the rest seemed to average roughly the same recommendation rating.

Additionally, we looked at some statistics of all the recommendation ratings in our dataset:

AVERAGE	59.5745283
MEDIAN	61
MINIMUM	7
MAXIMUM	100
STANDARD DEVIATION	17.40716947
VARIANCE	303.0095489

This gave us a sense of which points would be within standard deviation of the mean.

MongoDB Data Upload:

After scraping, we inputted the data into our MongoDB database in a couple of ways.

For our backup database from the raw Glassdoor collection, we directly transferred each item in the list to the collection glass_door_all_backup. It took a long time to run the parsing code to completion, and we knew that API access would expire within a week, and not all the pages could be parsed again. We used the following code to loop through each item and their respective attribute in our list of each company, and straight into our database. In our actual implication, we tested it out by putting the values in a local MongoDB database to make sure it worked, before placing it to the mLab cloud. The following is the code to place the data into the cloud.

```
#Calling database on mLab
client = MongoClient('mongodb://[REDACTED]:[REDACTED]@ds111476.mlab.com:11476/final_project')
db = client.get_database('final_project')

#Inputting the data into the database called "glassdoor all backup"
for i in range(len(a)):
    company = a[i]
db.glassdoor_all_backup.insert_one(company)
```

2. Our fully mapped dataset (resolved entity resolution) that combines the datasets from Glassdoor, weather dataset (snow and rain specifically), commute times, headquarter location, diversity data and crime rates are stored within a CSV.

For input into a local database:

We converted this CSV into numpy arrays for each attribute in python. We would then loop through respective values in each array for each company to input them into the database. We input these values first in a local MongoDB database to test out if it worked before directly inputting it into the mLab cloud. In this example, 'a' is the list of all the information collected from the API. Since mapping created null values as not all companies provide the same information, we allowed those null values to exist in the database.

Here is a sample of how we executed this (using only a few attributes) into the local database:


```

for i in range(len(a)):
    company = {'company_name': names[i],
              'culture_rating': culture_rating[i],
              'ceo_names': ceo_names[i],
              'ceo_pctApprove': ceo_pctApprove[i],
              'ceo_pctDisapprove': ceo_pctDisapprove[i]}
    db_gd.info.insert_one(company)

```

For input into mLab cloud:

Mlab has a tool where we could import a csv directly to a collection in the database. For the fully mapped companies data, we imported it into a collection called “companies” by executing this command in our command terminal.

```

(C:\ProgramData\Anaconda3) C:\Users\vucyn>mongoimport -h ds111476.mlab.com:11476 -d final_project -c companies -u <user>
-p <password> --file <input .csv file> --type csv --headerline

```

We in addition imported the other individual datasets, except for Glassdoor raw data the same method by using this tool.

Overall through all of these sources, we were able to gather a sufficient amount. Glassdoor provided a tremendous amount of features per company, and many other sources such as the weather and the crime data were some of the biggest public database for that feature.

Methodology:

- What did you do with your data?
- What techniques were used to pick apart the data?
- Did you use ML? Stats?
- How did you visualize your data? Report on your data?

In addition to this outline, we have more concrete requirements:

- **Data:** Please submit your cleaned data or, if it’s too large, a reference to the original data as well as the scripts you used to clean it.
- **ML/Stats:** Use at least **two** machine learning or statistical analysis techniques to analyze your data, explain what you did in the end, and talk about the inferences you uncovered.

- **Visualization:** Provide at least **two** distinct visualizations of your data or final results. This means two *different* techniques. If you use bar charts to analyze one aspect of your data, while you may use bar charts again, the second use will not count as a *distinct* visualization.
- **Additional work:** In addition to the requirements in the ML and visualization sections above, we would like to see at least one extra from either category. That means a total of five deliverables.

Preparing the Data for Machine Learning:

As stated in the previous post, our merged dataset is in a csv which consisted of more than 1,000 companies. This dataset consisted of 20 features. In the process of feature engineering, we removed features such as company name, CEO name, and location of headquarters because they are not considered important to our algorithm or there existed another feature that provided this mapping.

The 20 features are the following, as in the previous post: 6

- CEO name
- CEO approval percentage
- CEO disapproval percentage
- Company name
- Compensation rating
- Culture rating
- Overall rating
- Rating description
- Recommendation to a friend rating
- Senior leadership rating
- Work life balance rating
- Growth
- Annual rainfall in location
- Annual snowfall in location
- Average commute time
- City of Headquarter
- State of Headquarter
- Population

- Crime rate
- Diversity rating (by city)

All company related data was taken from Glassdoor except for the culture rating.

However, we ELIMINATED the following features:

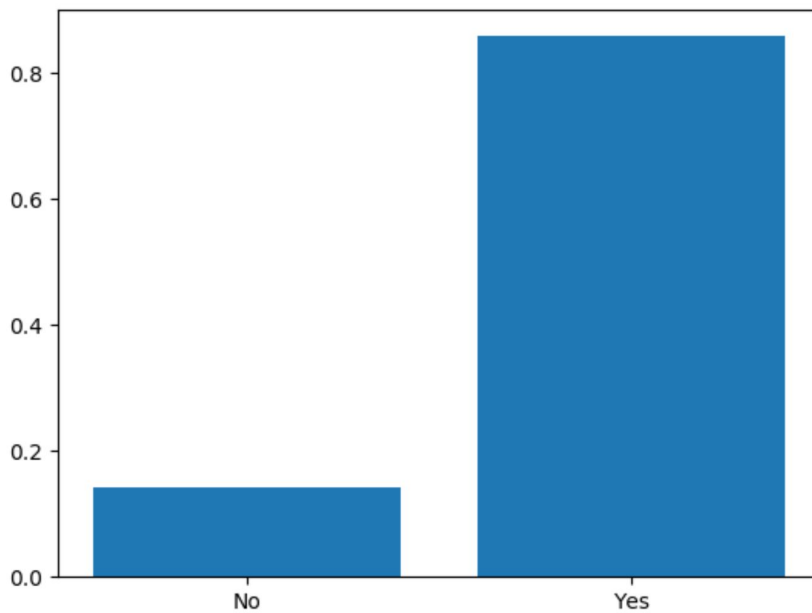
- CEO names, company name, HQ city, HQ state, database ID, population, rate description (irrelevant for calculation)
- Growth (not enough information found for all companies)
- Compensation rating, senior leadership rating, work life balance rating, CEO approval and disapproval (did not contribute significantly to the calculation)
- Overall rating (direct correlation with the classification)

As a result, we kept the following for training dating:

- Culture rating
- Annual snowfall
- Crime per 1000 people
- Annual rainfall
- Average commute

The recommendation to a friend rating, which reflects the overall rating of the company, is our dependent variable that measures the happiness level of the employees at the company. We assume that if you are happy with your company, you would be willing to recommend it to a friend and vice versa. Our threshold for a 'yes' recommendation will be a recommendation score of 60.

Here is the result of a sample probability:



In this example, the algorithm will predict that you would recommend to a friend with the following probabilities.

The final step was to remove any rows of data that contain either zero or none values. As a result, the final dataset consists of 857 companies, which is a large enough data set for these classification models. Below is a picture of the final dataset:

Out[7]:

	culture_rating	recommendToFriendRating	Annual Rainfall	Annual Snowfall	Avg_Commute_work	crime_rate_per_1000
0	2.6	0	32.8	51.1	21.3	70.070119
1	3.7	1	42.9	28.2	22.0	73.345725
2	3.3	1	42.9	28.2	23.4	72.754456
3	3.6	0	42.9	28.2	23.4	72.754456
4	3.5	1	42.9	28.2	23.4	72.754456
8	2.5	0	54.2	6.3	31.9	31.524805
9	4.5	1	22.2	0.0	21.0	44.309769
10	3.4	1	39.2	24.6	24.0	74.906594
11	3.1	0	22.2	0.0	28.2	116.072383
13	4.3	1	27.3	54.0	26.4	40.623027
14	3.8	1	22.2	0.0	28.2	116.072383
15	2.2	0	54.2	6.3	31.9	31.524805
16	2.5	0	41.8	123.8	24.8	34.985832
19	4.5	1	42.9	28.2	23.4	72.754456
20	3.8	1	47.7	43.8	19.8	50.160488
22	4.5	1	27.3	54.0	33.7	27.189152

Model Construction

The dataset was broken down into training and testing sets. We used a standard 80/20 breakdown and separated our independent and dependent variables.

Through trial and error, we eliminated features and model parameters that did not have a strong impact on the overall recommendation to a friend. The machine learning model we used to analyze our data is a Voting Classifier with soft voting for two different classifiers: Random Forest, Multinomial Logistic Regression. We used Python's RandomizedSearchCV and cross-validation to properly tune our hyperparameters (eg. max_depth, min_samples_split, weights, etc.) for each model to reach the highest accuracy for both the train and test datasets. Below is our code for constructing this model.

```
# fit the model using the entire dataset

clf2 = RandomForestClassifier()
clf5 = LogisticRegression(multi_class='multinomial', solver='newton-cg')
ecf = VotingClassifier(estimators=[('rf', clf2), ('lr', clf5)], voting='soft')

param_grid = {
    'rf_n_estimators': [2, 4, 8, 16],
    'rf_min_samples_leaf': [50, 100, 500, 128, 250, 400],
    'weights': [[7, 3], [2, 5], [3, 4], [5, 3], [4, 2], [2, 3]]
}

ecf = RandomizedSearchCV(estimator=eclf, param_distributions=param_grid, cv=10, scoring='accuracy')

ecf.fit(x_train, y_train)
```

Results:

We were able to achieve 85% for the accuracy of both the train and test datasets. As a result, the model uses culture rating, state annual snowfall, state annual rainfall, crime per 1000 people and average work commute time within the city to develop an employee happiness score for the company.

```
#training accuracy
y_predict = eclf.predict(x_train)
metrics.accuracy_score(y_train, y_predict)

Out[10]: 0.84285714285714286

In [12]: #test accuracy
y_predict = eclf.predict(x_test)
metrics.accuracy_score(y_test, y_predict)

Out[12]: 0.85987261146496818
```

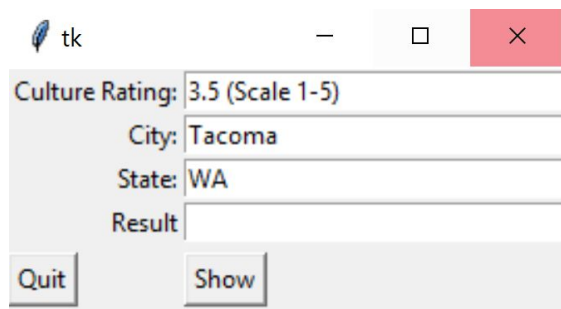
As a caveat, we previously were able to reach an accuracy score of 92% and 85% for the training and testing set. However, we went with this model because we wanted to utilize other data sources rather than relying on Glassdoor data more heavily. In the end, almost all the features were taken from other sites and the target feature to predict was the only one taken from Glassdoor that was used for training. This gives us the security that our model works well since none of the more subjectively collected data is correlated from coming from coming from the same data source as the target feature to predict.

Visualization and Reporting:

We have two ways of presenting our findings. The first way will be seen by the user in our interactive applications. The second uses the Google Maps API that plots our findings directly on the map.

1. User Interface:

Our interactive applications and our development will be explained in more detail in a later post. In short, we created a user interface application that allows one to input certain values as seen here:

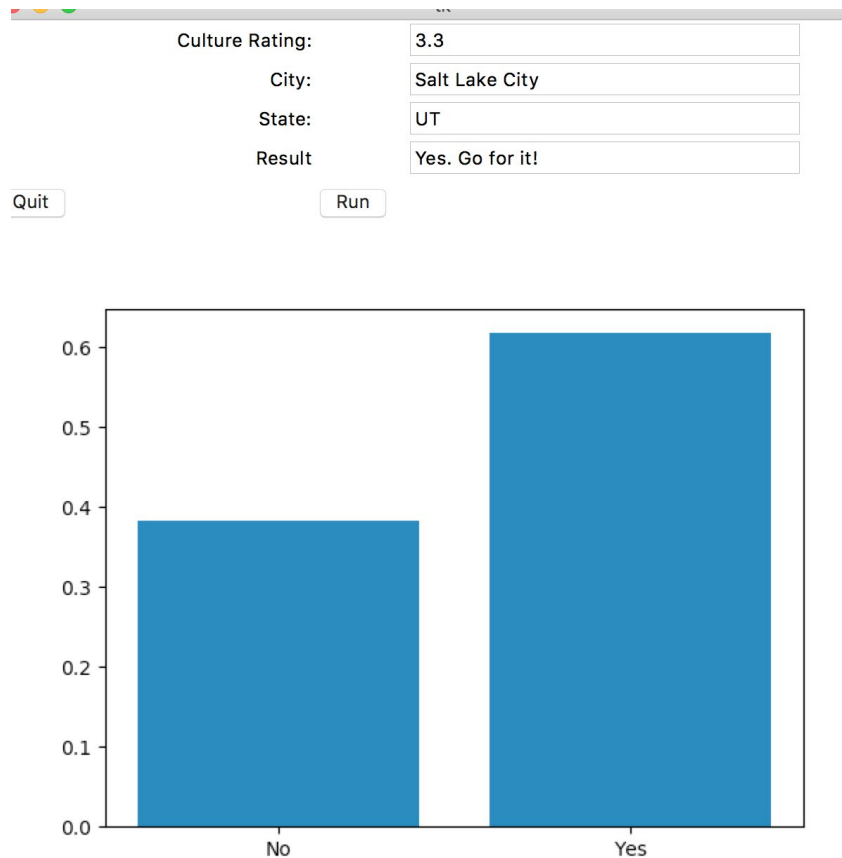


The image shows a Tkinter window titled 'tk' with a standard macOS-style title bar (red, yellow, and green buttons). The window contains a form with the following elements:

- A label 'Culture Rating:' followed by a text entry field containing '3.5 (Scale 1-5)'.
- A label 'City:' followed by a text entry field containing 'Tacoma'.
- A label 'State:' followed by a text entry field containing 'WA'.
- A label 'Result' followed by an empty text entry field.
- At the bottom left, a 'Quit' button.
- At the bottom right, a 'Show' button.

We will go more into depth about our application on the next post.

Our output will also have a bar graph displaying the probabilities of the 2 different categories, which are the possible choices for a recommendation to a friend: Yes or No. With those 2 categories on the x-axis, the y-axis will contain the likelihood percentages of each category as calculated by our algorithm. The category with the highest percentage will ultimately be the chosen choice that our website has predicted for the user. Below is an example of our visualization.



The inputted values run through our pre-trained model and the user gets the following screen back

The bar with the input 'Result' shows if you would recommend to a friend with the given values. The bar given output is the likelihood of you recommending to a friend comparatively to each other.

2. Google Earth Pro API:

Another amazingly stunning tool for data visualization is Google Earth Pro. The software uses Keyhole Markup language Zipped(KMZ) files. KMZ is a file format that will allow users to display geographic data in mapping applications. The KMZ files can be created in Python by inserting the proper values you want into the placement object.

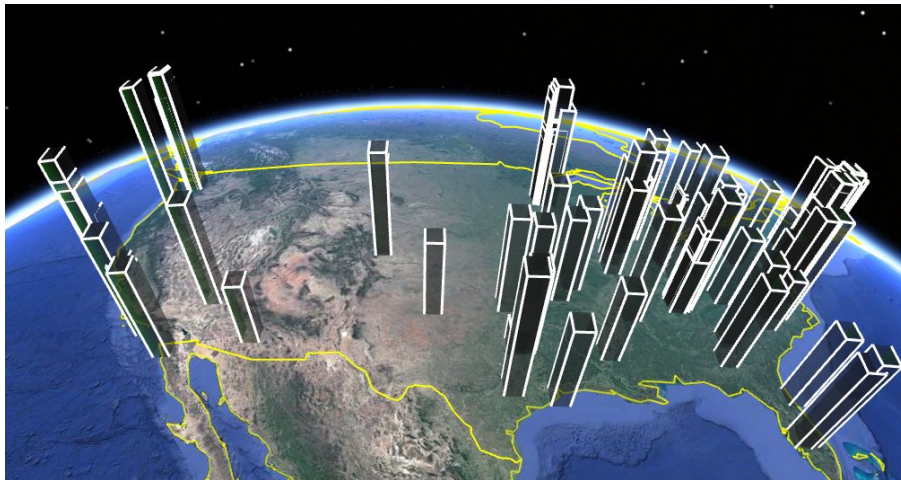
We used this tool to display a 3-D bar-graph across company headquarters in the United States showing Friend Recommendation scores. The bars are created from cuboids coded with identical square bases and heights that vary to reflect the data. Bars are placed on the map of the U.S. according to their respective latitude and longitude coordinates. An example of the code for one bar of the graph is shown below.

```

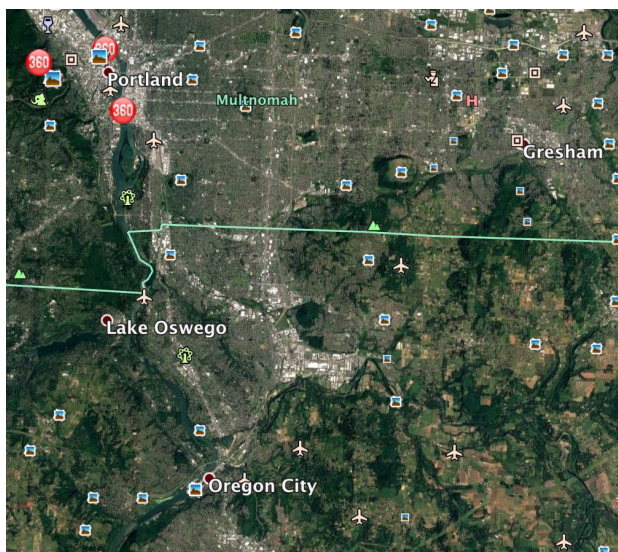
<Placemark><name>Friend_Recommend</name><styleUrl>#transGreenPoly</styleUrl><Polygon><extrude>1</extrude><altitude
Mode>relativeToGround</altitudeMode><outerBoundaryIs><LinearRing><coordinates> -71.0597732,42.3584308,605384.6154
-71.0597732,41.3584308,605384.6154 -70.0597732,41.3584308,605384.6154
-70.0597732,42.3584308,605384.6154</coordinates></LinearRing></outerBoundaryIs></Polygon></Placemark>

```

Below is an image of our visual in Google Earth Pro:



The user are also able to interact with the visual by navigating to the location of each bar. For example, the user could look at the location and earth view of the area surrounding the bar of interest:



With these two visuals, our users will be able to know the confidence of the algorithm suggestions and to explore areas with high Friend Recommendation score.

Final User API:

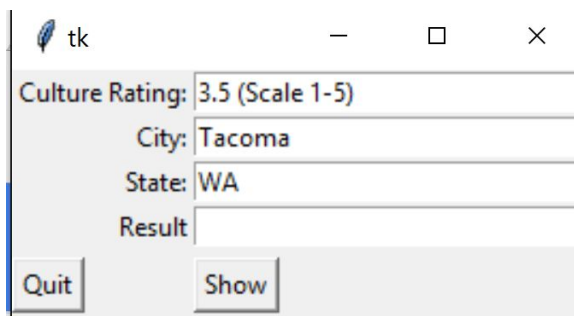
- Provide documentation on the User API to your project. Include information on how to load data, update data, run an analysis, generate reports, visualizations, etc.

For our interactive component, we have two parts. The first an interactive application where you are able to input your own values, and will see the likelihood of being recommended. The second is the same application, but applied as Slack bot application. We'll go into detail here of each one.

1. This is a python code that creates a friendly user interface of the API. In other words, someone could input their own values and the algorithm would run, as an API would, but we created a UI to simplify the process for the user. The program returns the probability charts and the prediction to see if you would recommend to a friend or not.

The interactive application asks you for the following inputs:

- The culture rating (Scale 1-5)
- HQ City
- HQ State

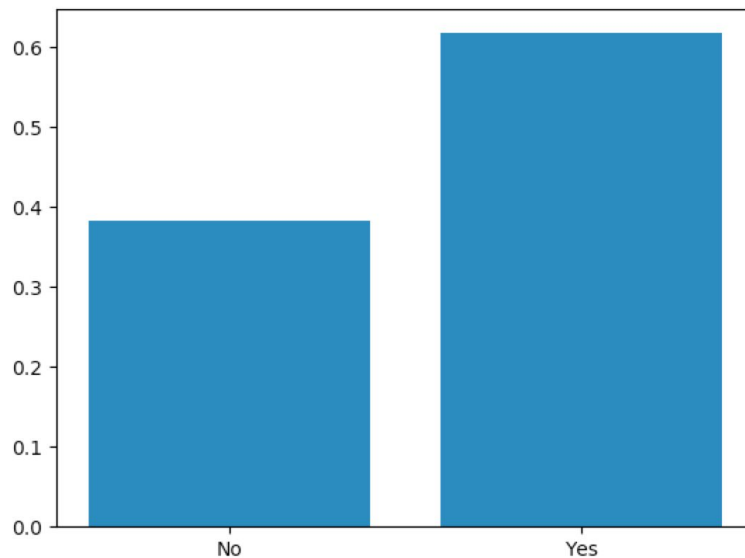


The screenshot shows a Tkinter window with a title bar that says 'tk'. Inside the window, there are four text input fields stacked vertically. The first field is labeled 'Culture Rating:' and contains the value '3.5 (Scale 1-5)'. The second field is labeled 'City:' and contains 'Tacoma'. The third field is labeled 'State:' and contains 'WA'. The fourth field is labeled 'Result' and is currently empty. Below these fields, there are two buttons: 'Quit' on the left and 'Show' on the right.

The headquarter location will correspond to the level of rainfall, level of snowfall, crime rate and the commute time. The corresponding information is directly connected to database cloud on mLab, so everytime we run this algorithm, it's directly connected to our database. With this inputted information, we will be able to predict the level of satisfaction of the employee at that particular company as seen here again:

Culture Rating:	3.3
City:	Salt Lake City
State:	UT
Result	Yes. Go for it!

Quit Run



Now how does this work? The inputted information will be transformed into a dataframe, with the level of rainfall and commute time in the input format of our machine learning algorithm. The rainfall and commute time data will be stored in MongoDB for the algorithm to query with the input values. From there, the output will be a probability array for each class, which we will translate for our user to see.

Our output will also have a bar graph displaying the probabilities of the 2 different categories, which are the possible choices for a recommendation to a friend: Yes or No, as seen in the previous post.

Behind the scenes:

Using Tkinter, we are able to construct a UI. We called on the Tkinter object and named it main. The following code allowed us to design the physical attributes of the UI shown above:


```

main = Tk()
Label(main, text = "Culture Rating:").grid(row=0, sticky='e')
Label(main, text = "City:").grid(row=1, sticky='e')
Label(main, text = "State:").grid(row=2, sticky='e')
Label(main, text = "Result").grid(row=3, sticky='e')

num1 = Entry(main, width=30)
num2 = Entry(main, width=30)
num3 = Entry(main, width=30)
blank = Entry(main, width=30)
num1.insert(END, '3.5 (Scale 1-5)')
num2.insert(END, 'Tacoma')
num3.insert(END, 'WA')

num1.grid(row=0, column=1)
num2.grid(row=1, column=1)
num3.grid(row=2, column=1)
blank.grid(row=3, column=1)

Button(main, text='Quit', command=main.destroy).grid(row=4, column=0, sticky=W, pady=4)
Button(main, text='Show', command=show_answer).grid(row=4, column=1, sticky=W, pady=4)

mainloop()

```

We have sectioned the code into 5 different parts which will allow us to explain each part clearly.

Section 1 creates the input labels of each row with a blank grid next it in the UI. The `.grid(row=0)` part identifies where the location of the grid should be while `sticky='e'` makes sure it is in line with the labels.

Section 2 establishes that the top three grids are to be entry boxes that have a width of 30. Each of the entries inside the grid are call num1, num2, and num3. The insert statement adds the following comment into the entry boxes, so the user knows the format of the values they are to put in.

Section 3 makes sure that the values in Section 2 appear. Without section 3, the objects exists, but would not be visible to the user.

Section 4 creates the buttons 'Quit' and 'Show' with their location. We will speak more about the command 'show_answer' later. That button calls on the function that allows the application to work properly.

Section 5 makes sure that the application doesn't immediately close after running. Instead, the application will continue to be there until it is exited out.

As we mentioned before, `show_answer` is the function that implements the values given in the application. We have divided up the function into 4 sections to make it easier to explain, what each part does.

```
def show_answer():
    # Ans = int(num1.get()) + int(num2.get())
    # blank.insert(0, Ans)
    #
    #
    # culture_rating = 3.1
    # Annual_Rainfall = 54.5
    # Annual_Snowfall = 0.0
    # Avg_Commute_work = 26.9
    # crime_rate_per_1000 = 77.971077
    # diversity = 67.51

    x = num2.get()
    y = num3.get()
    blank.delete(0, END)
    fig = plt.figure(1)
    plt.ion()
    plt.gcf().clear()

    try:
        commute = db.city_commute.find_one({'city': x, 'state': y}, {'commute_time_mins_est': True})['commute_time_mins_est']
    except:
        blank.delete(0, END)
        blank.insert(0, "City/State does not exist in database")

    try:
        inches_rain = db.state_rainfall.find_one({'Abbreviation': y}, {'Inches': True})['Inches']
    except:
        blank.delete(0, END)
        blank.insert(0, "City/State does not exist in database")

    try:
        inches_snow = db.snowfall_data.find_one({'Abbreviation': y}, {'inches': True})['inches']
    except:
        blank.delete(0, END)
        blank.insert(0, "City/State does not exist in database")

    try:
        diversity = db.diversity_data.find_one({'city': x, 'state': y}, {'total_score': True})['total_score']
    except:
        blank.delete(0, END)
        blank.insert(0, "City/State does not exist in database")

    try:
        crime = db.crime_data.find_one({'City': x, 'abbreviation': y}, {'crime_rate': True})['crime_rate']
    except:
        blank.delete(0, END)
        blank.insert(0, "City/State does not exist in database")
```

```

try:
    culture=float(num1.get())
    if culture >5 or culture < 1:
        blank.delete(0, END)
        blank.insert(0, "Culture rating not valid")

except:
    blank.delete(0, END)
    blank.insert(0, "Enter culture rating with correct syntax")

x_t = {'culture_rating': [culture],
       'Annual_Rainfall': [inches_rain],
       'Annual_Snowfall': [inches_snow],    3
       'Avg_Commute_work': [commute],
       'crime_rate_per_1000': [crime],
       'diversity': [diversity]}
x_t = pd.DataFrame(data=x_t)
x_t = x_t.reindex_axis(['culture_rating', 'Annual_Rainfall', 'Annual_Snowfall', 'Avg_Commute_work', 'crime_rate_per_1000', 'diversity'], axis=1)
probas = eclf.predict_proba(x_t)
probas2 = eclf.predict(x_t)

if probas2[0] == 0:
    blank.insert(0, "Would not recommend to friends")

else:
    if culture <= 5 and culture >= 1:
        blank.insert(0, "Would recommend to friends!!!")

#decision graph

x = [1, 2]
labels = ['No', 'Yes']
plt.bar(x, probas[0], align='center')
plt.xticks(x, labels)                                4

canvas = FigureCanvasTkAgg(fig, main)
plot_widget = canvas.get_tk_widget()
plot_widget.grid(row=5, columnspan=2)

```

Section 1 sets the structure of the whole function. It first grabs the values in the latter two boxes (city and state) and stores them into 2 variables. Afterwards, it runs the algorithm and calls for the plots to be produced.

Section 2 establishes all the error flags to make sure the values are placed inside in the correct format. For example, the culture rating has to be a valid value between 1-5, and that the city and state combinations are available in our dataset. The program will return appropriate errors when users do not correctly insert the values.

Section 3 uses the inputted values and runs it directly through the database to collect the necessary information to run the algorithm for the user. For example, if the input was 'Seattle' and 'WA,' the function would go into the database on mLab and grab the corresponding rainfall in inches, snowfall in inches, crime rate, average commute information. Afterwards, the values are inputted into the predetermined model to get the probabilities and the results of the recommendation.

Section 4 makes the plot of the probability of recommending to a friend or not based on the probabilities established in Section 3 with the correct labels and axes.

2. Our second application is similar to the first but it is implemented as a slack bot.

Our team used a slack bot to commute the recommendation result through slack. It essentially acts as a "friend" to whether or not it recommends the company to the user given the inputs. We also use another

bot to communicate the confidence level of the recommendation with a similar graph mentioned above in the UI.

Interactive Application: If you created one, include information on your interactive application, including screenshots, and a live link if one exists.

- **Results:** Fully explain and analyze the results from your data, i.e. the inferences or correlations you uncovered, the tools you built, or the visualizations you created.

A bot can easily be created on the Slack website within your workspace account. Once the bot is created, you would need to have your Python script connect with it. Below is the code to initiate our script's connection with the two Slack bots mentioned above.

```
# instantiate Slack clients
slack_client = SlackClient('xoxb-285243337382-trQ8Ywc88dCdCdjHedSg5IBH')
slack = Slacker("xoxb-285636270311-HqQMMkAum8wl4FCbSPVUqBs9")
```

The code below continues to run on the back end to capture the messages flow from Slack. The second set of code identifies messages that are targeting our bot, in which the user uses the @ symbol to call our first bot's name.


```
#initiate the bot connection
if __name__ == "__main__":
    READ_WEBSOCKET_DELAY = 1 # 1 second delay between reading from firehose
    if slack_client.rtm_connect():
        print("StarterBot connected and running!")
        while True:
            command, channel = parse_slack_output(slack_client.rtm_read())
            if command and channel:
                handle_command(command, channel)
            time.sleep(READ_WEBSOCKET_DELAY)
    else:
        print("Connection failed. Invalid Slack token or bot ID?")
```


```
def parse_slack_output(slack_rtm_output):
    """
    The Slack Real Time Messaging API is an events firehose.
    this parsing function returns None unless a message is
    directed at the Bot, based on its ID.
    """
    output_list = slack_rtm_output
    if output_list and len(output_list) > 0:
        for output in output_list:
            print(output)
            if output and 'text' in output and AT_BOT in output['text']:
                # return text after the @ mention, whitespace remove
                return output['text'].split(AT_BOT)[1].strip().lower(), \
                    output['channel']
    return None, None
```


Once the bot is connected and we are receiving messages targeting our first bot, we would need to have a way to parse out the command that is sent by the user via slack to the bot. Our `handle_command()` parse the message and ensure that the command meets our requirement. If so, the function will aggregate culture rating, commute, crime, rainfall, and snowfall data from the input value and run them through our model. The output response and the probability plot will then be sent back by our respective bots into the channel where the command was initiated. Below is the code for these responses:

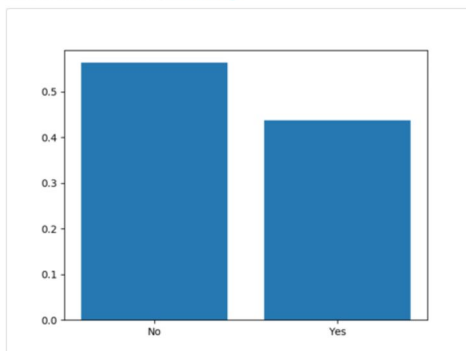
```
# Sending bot response to channel
slack_client.api_call("chat.postMessage", channel='G8E2KGF8W',
                      text=response, as_user=True)
if (response == "No, I don't recommend it!") or (response == "Yes. Go for it!"):
    slack.files.upload("botfig.png", channels="G8E2KGF8W")
```

Here is an example an input call and the responses that our bots generated:

 **ptruong** 1:42 AM
@friend-bot company 3.3 Tacoma WA

 **friend-bot** APP 1:42 AM
No, I don't recommend it!

 **friend-bot-confidence** APP 1:42 AM
uploaded this image: [botfig](#) ▾



The format for the command message is @friend-bot company culture_rating city state. In this case, culture_rating is 3.3. City and state are Tacoma and WA respectively. Friend-bot is our recommender bot and friend-bot-confidence is our confidence bot. You will notice that friend-bot recommended “No” with a probability of 60%. One could think of this number as a confidence level.

Users could either use our UI or the Slack bot to input the associated values with the company they are interested in. Both interfaces are easy to use and provide an informative out for the user. Additionally, Slack bot would provide the user with the same feeling as though they are receiving the recommendation from an actual friend!