In [1]:
```
conda install -c anaconda basemap
```

```
Collecting package metadata: done
Solving environment: done

## Package Plan ##

  environment location: /opt/conda

  added / updated specs:
    - basemap


The following packages will be downloaded:

    package                    |            build
    ---------------------------|-----------------
    basemap-1.2.0              |   py36h705c2d8_0        15.2 MB  anaconda
    ca-certificates-2019.5.15  |                0         133 KB  anaconda
    certifi-2019.3.9           |           py36_0         155 KB  anaconda
    geos-3.6.2                 |       hfc679d8_4         3.2 MB  conda-forge
    proj4-5.2.0                |       he6710b0_1         7.0 MB  anaconda
    pyproj-1.9.6               |   py36h14380d9_0          76 KB  anaconda
    pyshp-2.1.0                |             py_0          34 KB  anaconda
    ------------------------------------------------------------
                                           Total:        25.9 MB

The following NEW packages will be INSTALLED:

  basemap            anaconda/linux-64::basemap-1.2.0-py36h705c2d8_0
  geos               conda-forge/linux-64::geos-3.6.2-hfc679d8_4
  proj4              anaconda/linux-64::proj4-5.2.0-he6710b0_1
  pyproj             anaconda/linux-64::pyproj-1.9.6-py36h14380d9_0
  pyshp              anaconda/noarch::pyshp-2.1.0-py_0

The following packages will be UPDATED:

  ca-certificates    conda-forge::ca-certificates-2019.3.9~ --> anaconda::ca-
certificates-2019.5.15-0
  openssl            conda-forge::openssl-1.1.1b-h14c3975_1 --> anaconda::ope
nssl-1.1.1-h7b6447c_0

The following packages will be SUPERSEDED by a higher-priority channel:

  certifi                                     conda-forge --> anaconda



Downloading and Extracting Packages
ca-certificates-2019 | 133 KB    | ##################################### | 10
0%
certifi-2019.3.9     | 155 KB    | ##################################### | 10
0%
geos-3.6.2           | 3.2 MB    | ##################################### | 10
0%
pyproj-1.9.6         | 76 KB     | ##################################### | 10
0%
pyshp-2.1.0          | 34 KB     | ##################################### | 10
0%
```

```
basemap-1.2.0         | 15.2 MB    | ##################################### | 10
0%
proj4-5.2.0           | 7.0 MB     | ##################################### | 10
0%
Preparing transaction: done
Verifying transaction: done
Executing transaction: done

Note: you may need to restart the kernel to use updated packages.
```

In [2]:
```
%matplotlib inline
import numpy as np
import pandas as pd
import seaborn as sns
import re
import datetime as dt
import matplotlib.pyplot as plt
import sklearn.linear_model as linear_model
from sklearn.metrics import mean_squared_error as mse
import sklearn.datasets
from sklearn.model_selection import train_test_split

import plotly.offline as py
py.init_notebook_mode(connected=True)
import plotly.graph_objs as go
import plotly.figure_factory as ff
import cufflinks as cf
cf.set_config_file(offline=False, world_readable=True, theme='ggplot')

import string
from nltk.corpus import stopwords
from nltk import word_tokenize
from gensim.corpora.dictionary import Dictionary
from nltk import FreqDist

from gensim.models.tfidfmodel import TfidfModel
from gensim.similarities import MatrixSimilarity

import os
import conda

conda_file_dir = conda.__file__
conda_dir = conda_file_dir.split('lib')[0]
proj_lib = os.path.join(os.path.join(conda_dir, 'share'), 'proj')
os.environ["PROJ_LIB"] = proj_lib

from mpl_toolkits.basemap import Basemap

import io
import requests
```

Loading our Data into Jupyter Notebooks

In [3]:
```
okcupid = pd.read_csv("okcupid_profiles.csv")
okcupid.head()
```

Out[3]:

	age	body_type	diet	drinks	drugs	education	essay0	essay1	
0	22	a little extra	strictly anything	socially	never	working on college/university	about me: \n \ni would love to think...	currently working as an international agent fo...	makin abo
1	35	average	mostly other	often	sometimes	working on space camp	i am a chef: this is what that means. \n1...	dedicating everyday to being an unbelievable b...	b amor
2	38	thin	anything	socially	NaN	graduated from masters program	i'm not ashamed of much, but writing public te...	i make nerdy software for musicians, artists, ...	impr alte
3	23	thin	vegetarian	socially	NaN	working on college/university	i work in a library and go to school. . .	reading things written by old dead people	synt and o book
4	29	athletic	NaN	socially	never	graduated from college/university	hey how's it going? currently vague on the pro...	work work work work + play	i lo \nhtt

5 rows × 31 columns

We check each variable to see which columns we want to focus on and analyze.

In [4]:
```
okcupid.columns
```

Out[4]:
```
Index(['age', 'body_type', 'diet', 'drinks', 'drugs', 'education', 'essay0',
       'essay1', 'essay2', 'essay3', 'essay4', 'essay5', 'essay6', 'essay7',
       'essay8', 'essay9', 'ethnicity', 'height', 'income', 'job',
       'last_online', 'location', 'offspring', 'orientation', 'pets',
       'religion', 'sex', 'sign', 'smokes', 'speaks', 'status'],
      dtype='object')
```

Examining Location

We want to study the distribution of okcupid users around the world to learn where this dating site is most popular, where most users reside and how we could use this information to adapt our data to best serve our analysis.

Our location variable in our okcupid data is formated such that the name of the city comes before a coma followed by either the state or country of that city. We make use of regex match to get rid of all the characters before the given state/country, that way our new variable **states** contains only the state\country, thus allowing us to examine the global user population.

In [5]:
```
# We find the the best regex pattern to highlight the characters before
# each state/country.
regx1 = r"^(.+?), california"

def show_regex_match(text, regex):
# Prints the string with the regex match highlighted.

    print(re.sub(f'({regex})', r'\033[1;30;43m\1\033[m', text))

show_regex_match("san francisco, california",regx1 ) # Test
```

```
san francisco, california
```

In [6]:
```
# We use string replace with regex to only keep the state/coutry
states = okcupid['location'].str.replace( r"^(.+?), ", "")
print('Number of states/countries in okcupid:',len(states.unique()))
states.unique()
```

```
Number of states/countries in okcupid: 41
```

Out[6]:
```
array(['california', 'colorado', 'new york', 'oregon', 'arizona',
       'hawaii', 'montana', 'wisconsin', 'virginia', 'spain', 'nevada',
       'illinois', 'vietnam', 'ireland', 'louisiana', 'michigan', 'texas',
       'united kingdom', 'massachusetts', 'north carolina', 'idaho',
       'mississippi', 'new jersey', 'florida', 'minnesota', 'georgia',
       'utah', 'washington', 'west virginia', 'connecticut', 'tennessee',
       'rhode island', 'district of columbia', 'british columbia, canada',
       'missouri', 'germany', 'pennsylvania', 'netherlands',
       'switzerland', 'mexico', 'ohio'], dtype=object)
```

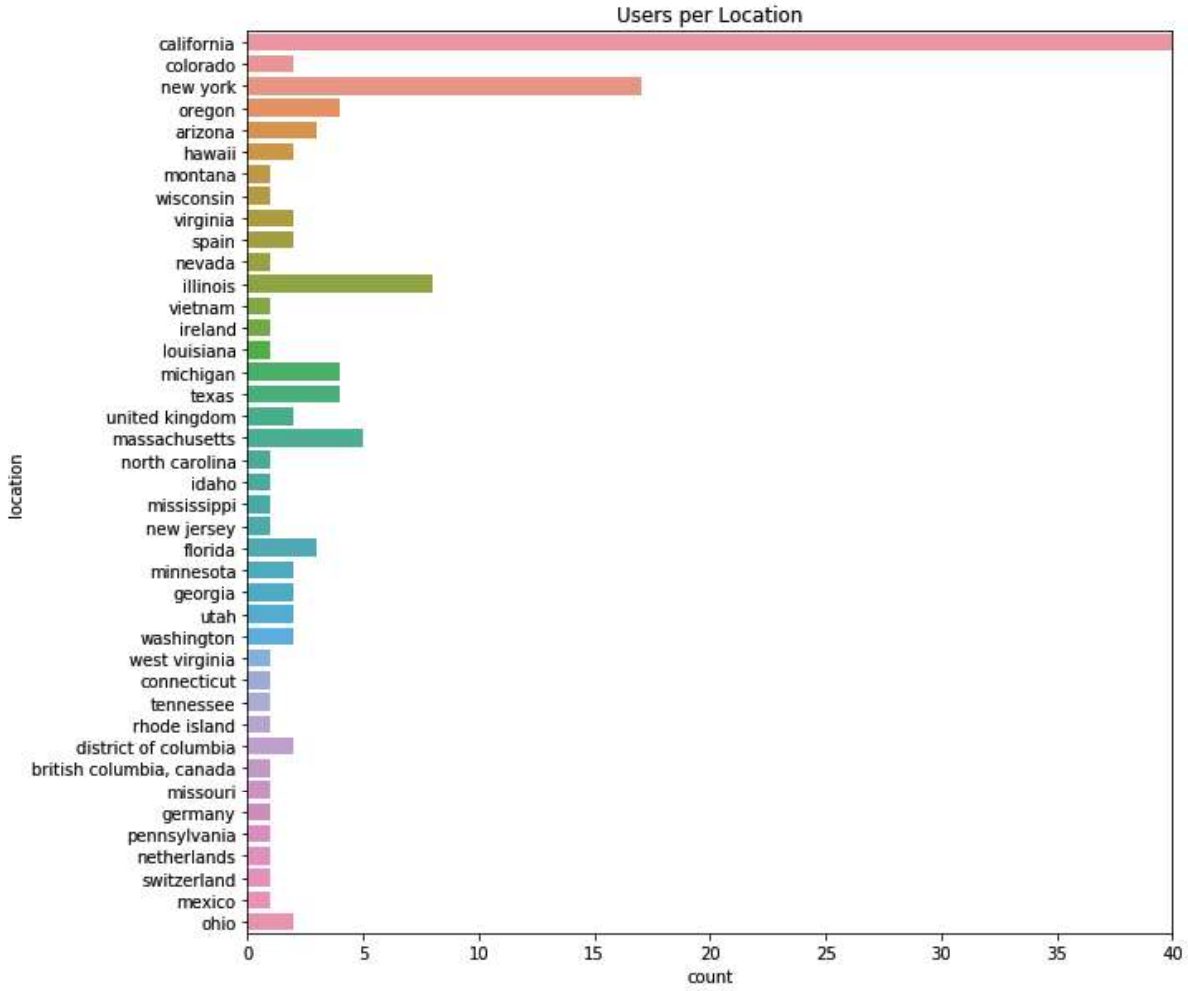
Visualization: Countplot

Now we want to visualize the count of users per city in California to decide whether this is a more even distribution of geographic location in comparison to that of the world.

In [7]:
```
locations = states.to_frame()
```

In [8]:
```
fig = plt.figure(figsize=(10,10))
fig.add_subplot(1,1,1)
ax = sns.countplot(y = "location", data = locations)
plt.xlim([0,40])
plt.title('Users per Location')
```

Out[8]: Text(0.5, 1.0, 'Users per Location')



Comparing Users in California to Users Anywhere Else

Our count plot above demonstrates an extreme difference in the number of users between California and the rest of the world. According to our visualization there is a total of 59855 okcupid users in California and only 91 users anywhere else in the world.

In [9]:
```
cali = locations[locations['location'] == 'california']
not_cali = locations[locations['location'] != 'california']
print('okcupid users worldwide:',len(okcupid['location']))
print('okcupid users in California:',len(cali))
print('okcupid users not in California',len(not_cali))
```

```
okcupid users worldwide: 59946
okcupid users in California: 59855
okcupid users not in California 91
```

The significantly small distribution of users worldwide compared to that in california allows us to make the decision to base the rest of our analysis and examination only with datasets from okcupid users that strictly reside in California.

In [10]:
```
# Once again we find the the best regex pattern to highlight all the locations
with california in them.

regx1 = r"^(.+?), california"
def show_regex_match(text, regex):

#Prints the string with the regex match highlighted.

    print(re.sub(f'({regex})', r'\033[1;30;43m\1\033[m', text))

show_regex_match("san francisco, california",regx1 )
```

```
san francisco, california
```

We create a filter that will remove all of cases that are from users outside of California.

In [11]:
```
filter = okcupid['location'].str.contains(r"^(.+?), california")
okcupid_ca = okcupid[filter]

print('lenght of new dataset:',len(okcupid_ca))
okcupid_ca.head()
```

```
lenght of new dataset: 59855

/opt/conda/lib/python3.6/site-packages/ipykernel_launcher.py:1: UserWarning:

This pattern has match groups. To actually get the groups, use str.extract.
```

Out[11]:

	age	body_type	diet	drinks	drugs	education	essay0	essay1	
0	22	a little extra	strictly anything	socially	never	working on college/university	about me: \n \ni would love to think...	currently working as an international agent fo...	makin abo
1	35	average	mostly other	often	sometimes	working on space camp	i am a chef: this is what that means. \n1...	dedicating everyday to being an unbelievable b...	amor
2	38	thin	anything	socially	NaN	graduated from masters program	i'm not ashamed of much, but writing public te...	i make nerdy software for musicians, artists, ...	impr alte
3	23	thin	vegetarian	socially	NaN	working on college/university	i work in a library and go to school. . .	reading things written by old dead people	synt and o book
4	29	athletic	NaN	socially	never	graduated from college/university	hey how's it going? currently vague on the pro...	work work work work + play	i lo \nhtt

5 rows × 31 columns

In [12]:
```
print('Number of Cities in CA:',len(okcupid_ca['location'].unique()))
okcupid_ca['location'].unique()
```

```
Number of Cities in CA: 135
```

Out[12]:
```
array(['south san francisco, california', 'oakland, california',
       'san francisco, california', 'berkeley, california',
       'belvedere tiburon, california', 'san mateo, california',
       'daly city, california', 'san leandro, california',
       'atherton, california', 'san rafael, california',
       'walnut creek, california', 'menlo park, california',
       'belmont, california', 'san jose, california',
       'palo alto, california', 'emeryville, california',
       'el granada, california', 'castro valley, california',
       'fairfax, california', 'mountain view, california',
       'burlingame, california', 'martinez, california',
       'pleasant hill, california', 'hayward, california',
       'alameda, california', 'vallejo, california',
       'benicia, california', 'el cerrito, california',
       'mill valley, california', 'richmond, california',
       'redwood city, california', 'el sobrante, california',
       'stanford, california', 'san pablo, california',
       'novato, california', 'pacifica, california',
       'lafayette, california', 'half moon bay, california',
       'fremont, california', 'orinda, california',
       'san anselmo, california', 'corte madera, california',
       'albany, california', 'san carlos, california',
       'san lorenzo, california', 'foster city, california',
       'hercules, california', 'santa cruz, california',
       'bolinas, california', 'sausalito, california',
       'millbrae, california', 'larkspur, california',
       'moraga, california', 'san bruno, california',
       'petaluma, california', 'pinole, california',
       'san geronimo, california', 'crockett, california',
       'brisbane, california', 'freedom, california',
       'montara, california', 'green brae, california',
       'woodside, california', 'ross, california',
       'east palo alto, california', 'san quentin, california',
       'rodeo, california', 'hacienda heights, california',
       'woodacre, california', 'westlake, california',
       'riverside, california', 'rohnert park, california',
       'sacramento, california', 'point richmond, california',
       'san diego, california', 'canyon country, california',
       'west oakland, california', 'kentfield, california',
       'glencove, california', 'tiburon, california',
       'santa monica, california', 'los angeles, california',
       'moss beach, california', 'hillsborough, california',
       'olema, california', 'union city, california', 'colma, california',
       'kensington, california', 'redwood shores, california',
       'brea, california', 'lagunitas, california',
       'stinson beach, california', 'santa clara, california',
       'studio city, california', 'concord, california',
       'piedmont, california', 'seaside, california',
       'forest knolls, california', 'magalia, california',
       'orange, california', 'bayshore, california',
       'los gatos, california', 'sunnyvale, california',
       'ashland, california', 'pasadena, california',
       'arcadia, california', 'milpitas, california',
       'port costa, california', 'nicasio, california',
       'livingston, california', 'granite bay, california',
       'isla vista, california', 'hilarita, california',
       'campbell, california', 'santa ana, california',
```

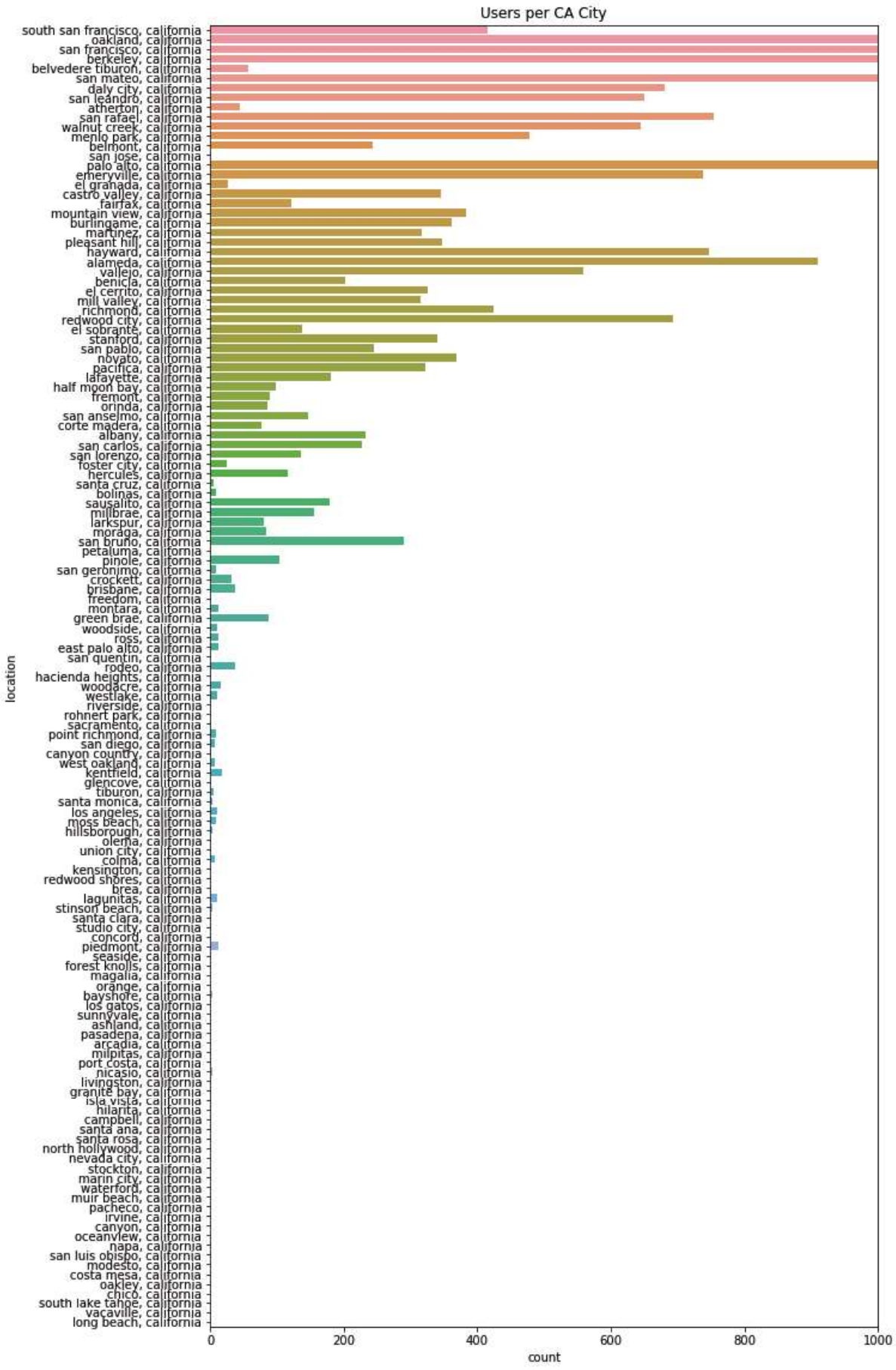
```
'santa rosa, california', 'north hollywood, california',
'nevada city, california', 'stockton, california',
'marin city, california', 'waterford, california',
'muir beach, california', 'pacheco, california',
'irvine, california', 'canyon, california',
'oceanview, california', 'napa, california',
'san luis obispo, california', 'modesto, california',
'costa mesa, california', 'oakley, california',
'chico, california', 'south lake tahoe, california',
'vacaville, california', 'long beach, california'], dtype=object)
```

Visualization: Countplot

Now we want to visualize the count of users per city in California to decide whether this is a more even distribution of geographic location in comparison to that of the world.

In [13]:
```
fig = plt.figure(figsize=(10,20))
fig.add_subplot(1,1,1)
bx = sns.countplot(y = "location", data = okcupid_ca)
plt.xlim([0,1000])
plt.title('Users per CA City')
```

Out[13]: Text(0.5, 1.0, 'Users per CA City')



Our count plot demonstrates a better distribution within the cities of california that will provide us with a better insight of the user demographics with respect to its region.

Comparing OkCupid Users to Population Distribution Across California

We wish to have a better visualization that will show the precise geographic location, distribution and magnitude of users per area within california. Hence, we download conda and import Basemap from mpl_toolkits.basemap which is a geographical visualization tool that takes in latitude and longitude coordinates to plot the population size within each region of a given area.

This visualization tool requires us to extract the latitude and longitude of each city inside of our okcupid_ca dataset and use this in our basemap projection to plot the population side within the given coordinates. To do so we will download a data frame "cities", that contain the names of each city in California with it's respctive coordinates.

In [14]:
```
url="https://raw.githubusercontent.com/jakevdp/PythonDataScienceHandbook/master/notebooks/data/california_cities.csv"
s=requests.get(url).content

cities = pd.read_csv(io.StringIO(s.decode('utf-8')))
cities.tail()
```

Out[14]:

	Unnamed: 0	city	latd	longd	elevation_m	elevation_ft	population_total
477	477	Yountville	38.403056	-122.362222	30.0	98.0	2933
478	478	Yreka	41.726667	-122.637500	787.0	2582.0	7765
479	479	YubaCity	39.134722	-121.626111	18.0	59.0	64925
480	480	Yucaipa	34.030278	-117.048611	798.0	2618.0	51367
481	481	YuccaValley	34.133333	-116.416667	1027.0	3369.0	20700

Now that we have two seperate dataframes our goal is to merge these together by location and ensure that each city in our okcupid_ca dataframe is assigned their accurate coordinates.

The variables that contain the location for each dataframe are **" city "** for dataframe **" cities "** and **" location "** for the dataframe **" okcupid_ca "**, each formated as a string.

We notice that the strings in **" city "** for **" cities "** are not seperated by any spaces, thus to make each string of California cities exactly the same for each dataframe we utilize python's string methods to make each character lowercase, get rid of the spaces, commas, and the word california.

In [15]:
```
okcupid_ca['location'] = (okcupid_ca['location']
 .str.lower()
 .str.strip()
 .str.replace('california', '')
 .str.replace(',', '')
 .str.replace(' ', '')
)
```

We wish to make a new dataframe that contains the polished variable " **location** " in " **okcupid_ca** " and a new column named " **users** " that contains the value counts of each city per location. We will name this new dataframe " **users_okcupid** "

As well, we will reset the index of " **users_okcupid** " and rename the columns " **index** " by " **city** " and " **location** " by " **users** ", so when we merge the data frames together we could call upon the column " **city** " on both frames for a smooth merge.

In [16]:
```
users = okcupid_ca['location'].value_counts()
users_okcupid = users.reset_index().rename(index = str, columns = {"index":"ci
ty","location":"users"})
print('Number of CA cities in okcupid:',len(users_okcupid['city'].unique()))
users_okcupid['city'].unique()
```

```
Number of CA cities in okcupid: 135
```

Out[16]:
```
array(['sanfrancisco', 'oakland', 'berkeley', 'sanmateo', 'paloalto',
       'alameda', 'sanrafael', 'hayward', 'emeryville', 'redwoodcity',
       'dalycity', 'sanleandro', 'walnutcreek', 'vallejo', 'menlopark',
       'richmond', 'southsanfrancisco', 'mountainview', 'novato',
       'burlingame', 'pleasanthill', 'castrovalley', 'stanford',
       'elcerrito', 'pacifica', 'martinez', 'millvalley', 'sanbruno',
       'sanpablo', 'belmont', 'albany', 'sancarlos', 'benicia',
       'lafayette', 'sausalito', 'millbrae', 'sananselmo', 'elsobrante',
       'sanlorenzo', 'fairfax', 'hercules', 'pinole', 'halfmoonbay',
       'fremont', 'greenbrae', 'orinda', 'moraga', 'larkspur',
       'cortemadera', 'belvederetiburon', 'atherton', 'brisbane', 'rodeo',
       'crockett', 'elgranada', 'fostercity', 'kentfield', 'woodacre',
       'eastpaloalto', 'piedmont', 'montara', 'ross', 'westlake',
       'woodside', 'losangeles', 'lagunitas', 'sangeronimo', 'mossbeach',
       'bolinas', 'pointrichmond', 'westoakland', 'colma', 'sandiego',
       'tiburon', 'santacruz', 'hillsborough', 'stinsonbeach', 'nicasio',
       'santamonica', 'bayshore', 'kensington', 'forestknolls',
       'redwoodshores', 'napa', 'santarosa', 'sacramento', 'sanquentin',
       'sanjose', 'petaluma', 'losgatos', 'irvine', 'waterford',
       'northhollywood', 'concord', 'hilarita', 'riverside', 'livingston',
       'pasadena', 'vacaville', 'pacheco', 'sanluisobispo', 'rohnertpark',
       'portcosta', 'costamesa', 'canyoncountry', 'granitebay',
       'oceanview', 'olema', 'magalia', 'nevadacity', 'freedom',
       'glencove', 'santaclara', 'stockton', 'santaana', 'chico',
       'milpitas', 'islavista', 'unioncity', 'sunnyvale',
       'southlaketahoe', 'modesto', 'ashland', 'marincity', 'canyon',
       'arcadia', 'seaside', 'haciendaheights', 'longbeach', 'brea',
       'studiocity', 'oakley', 'muirbeach', 'orange', 'campbell'],
      dtype=object)
```

We repeat the same string method procedure to **" city "** in **cities**.

In [17]:
```
cities['city'] = (cities['city']
 .str.lower()
 .str.strip()
 .str.replace('california', '')
 .str.replace(',', '')
 .str.replace(' ', '')
)
print(len(cities['city'].unique()))
cities['city'].unique()
```

482

Out[17]:
```
array(['adelanto', 'agourahills', 'alameda', 'albany', 'alhambra',
       'alisoviejo', 'alturas', 'amadorcity', 'americancanyon', 'anaheim',
       'anderson', 'angelscamp', 'antioch', 'applevalley', 'arcadia',
       'arcata', 'arroyogrande', 'artesia', 'arvin', 'atascadero',
       'atherton', 'atwater', 'auburn', 'avalon', 'avenal', 'azusa',
       'bakersfield', 'baldwinpark', 'banning', 'barstow', 'beaumont',
       'bell', 'bellflower', 'bellgardens', 'belmont', 'belvedere',
       'benicia', 'berkeley', 'beverlyhills', 'bigbearlake', 'biggs',
       'bishop', 'bluelake', 'blythe', 'bradbury', 'brawley', 'brea',
       'brentwood', 'brisbane', 'buellton', 'buenapark', 'burbank',
       'burlingame', 'calabasas', 'calexico', 'city', 'calimesa',
       'calipatria', 'calistoga', 'camarillo', 'campbell', 'canyonlake',
       'capitola', 'carlsbad', 'carmelbythesea', 'carpinteria', 'carson',
       'cathedralcity', 'ceres', 'cerritos', 'chico', 'chino',
       'chinohills', 'chowchilla', 'chulavista', 'citrusheights',
       'claremont', 'clayton', 'clearlake', 'cloverdale', 'clovis',
       'coachella', 'coalinga', 'colfax', 'colma', 'colton', 'colusa',
       'commerce', 'compton', 'concord', 'corcoran', 'corning', 'corona',
       'coronado', 'cortemadera', 'costamesa', 'cotati', 'covina',
       'crescentcity', 'cudahy', 'culvercity', 'cupertino', 'cypress',
       'dalycity', 'danapoint', 'danville', 'davis', 'delano', 'delmar',
       'delreyoaks', 'deserthotsprings', 'diamondbar', 'dinuba', 'dixon',
       'dorris', 'dospalos', 'downey', 'duarte', 'dublin', 'dunsmuir',
       'eastpaloalto', 'eastvale', 'elcajon', 'elcentro', 'elcerrito',
       'elkgrove', 'elmonte', 'elsegundo', 'emeryville', 'encinitas',
       'escalon', 'escondido', 'etna', 'eureka', 'exeter', 'fairfax',
       'fairfield', 'farmersville', 'ferndale', 'fillmore', 'firebaugh',
       'folsom', 'fontana', 'fortbragg', 'fortjones', 'fortuna',
       'fostercity', 'fountainvalley', 'fowler', 'fremont', 'fresno',
       'fullerton', 'galt', 'gardena', 'gardengrove', 'gilroy',
       'glendale', 'glendora', 'goleta', 'gonzales', 'grandterrace',
       'grassvalley', 'greenfield', 'gridley', 'groverbeach', 'guadalupe',
       'gustine', 'halfmoonbay', 'hanford', 'hawaiiangardens',
       'hawthorne', 'hayward', 'healdsburg', 'hemet', 'hercules',
       'hermosabeach', 'hesperia', 'hiddenhills', 'highland',
       'hillsborough', 'hollister', 'holtville', 'hughson',
       'huntingtonbeach', 'huntingtonpark', 'huron', 'imperialbeach',
       'imperial', 'indianwells', 'indio', 'industry', 'inglewood',
       'ione', 'irvine', 'irwindale', 'isleton', 'jackson',
       'jurupavalley', 'kerman', 'kingcity', 'kingsburg',
       'lacaadaflintridge', 'lafayette', 'lagunabeach', 'lagunahills',
       'lagunaniguel', 'lagunawoods', 'lahabra', 'lahabraheights',
       'lakeelsinore', 'lakeforest', 'lakeport', 'lakewood', 'lamesa',
       'lamirada', 'lancaster', 'lapalma', 'lapuente', 'laquinta',
       'larkspur', 'lathrop', 'laverne', 'lawndale', 'lemongrove',
       'lemoore', 'lincoln', 'lindsay', 'liveoaksuttercounty',
       'livermore', 'livingston', 'lodi', 'lomalinda', 'lomita', 'lompoc',
       'longbeach', 'loomis', 'losalamitos', 'losaltos', 'losaltoshills',
       'losangeles', 'losbanos', 'losgatos', 'loyalton', 'lynwood',
       'madera', 'malibu', 'mammothlakes', 'manhattanbeach', 'manteca',
       'maricopa', 'marina', 'martinez', 'marysville', 'maywood',
       'mcfarland', 'mendota', 'menifee', 'menlopark', 'merced',
       'millbrae', 'millvalley', 'milpitas', 'missionviejo', 'modesto',
       'monrovia', 'montague', 'montclair', 'montebello', 'monterey',
       'montereypark', 'montesereno', 'moorpark', 'moraga',
       'morenovalley', 'morganhill', 'morrobay', 'mountainview',
       'mountshasta', 'murrieta', 'napa', 'nationalcity', 'needles',
```

```
'nevadacity', 'newark', 'newman', 'newportbeach', 'norco',
'norwalk', 'novato', 'oakdale', 'oakland', 'oakley', 'oceanside',
'ojai', 'ontario', 'orange', 'orangecove', 'orinda', 'orland',
'oroville', 'oxnard', 'pacifica', 'pacificgrove', 'palmdale',
'palmdesert', 'palmsprings', 'paloalto', 'palosverdesestates',
'paradise', 'paramount', 'parlier', 'pasadena', 'pasorobles',
'patterson', 'perris', 'petaluma', 'picorivera', 'piedmont',
'pinole', 'pismobeach', 'pittsburg', 'placentia', 'placerville',
'pleasanthill', 'pleasanton', 'plymouth', 'pointarena', 'pomona',
'porterville', 'porthueneme', 'portola', 'portolavalley', 'poway',
'ranchocordova', 'ranchocucamonga', 'ranchomirage',
'ranchopalosverdes', 'ranchosantamargarita', 'redbluff', 'redding',
'redlands', 'redondobeach', 'redwoodcity', 'reedley', 'rialto',
'richmond', 'ridgecrest', 'riodell', 'riovista', 'ripon',
'riverbank', 'riverside', 'rocklin', 'rohnertpark', 'rollinghills',
'rollinghillsestates', 'rosemead', 'roseville', 'ross',
'sacramento', 'salinas', 'sananselmo', 'sanbernardino', 'sanbruno',
'sancarlos', 'sanclemente', 'sandcity', 'sandiego', 'sandimas',
'sanfernando', 'sanfrancisco', 'sangabriel', 'sanger',
'sanjacinto', 'sanjoaquin', 'sanjose', 'sanjuanbautista',
'sanjuancapistrano', 'sanleandro', 'sanluisobispo', 'sanmarcos',
'sanmarino', 'sanmateo', 'sanpablo', 'sanrafael', 'sanramon',
'santaana', 'santabarbara', 'santaclara', 'santaclarita',
'santacruz', 'santafesprings', 'santamaria', 'santamonica',
'santapaula', 'santarosa', 'santee', 'saratoga', 'sausalito',
'scottsvalley', 'sealbeach', 'seaside', 'sebastopol', 'selma',
'shafter', 'shastalake', 'sierramadre', 'signalhill', 'simivalley',
'solanabeach', 'soledad', 'solvang', 'sonoma', 'sonora',
'southelmonte', 'southgate', 'southlaketahoe', 'southpasadena',
'southsanfrancisco', 'stanton', 'sthelena', 'stockton',
'suisuncity', 'sunnyvale', 'susanville', 'suttercreek', 'taft',
'tehachapi', 'tehama', 'temecula', 'templecity', 'thousandoaks',
'tiburon', 'torrance', 'tracy', 'trinidad', 'truckee', 'tulare',
'tulelake', 'turlock', 'tustin', 'twentyninepalms', 'ukiah',
'unioncity', 'upland', 'vacaville', 'vallejo', 'ventura', 'vernon',
'victorville', 'villapark', 'visalia', 'vista', 'walnut',
'walnutcreek', 'wasco', 'waterford', 'watsonville', 'weed',
'westcovina', 'westhollywood', 'westlakevillage', 'westminster',
'westmorland', 'westsacramento', 'wheatland', 'whittier',
'wildomar', 'williams', 'willits', 'willows', 'windsor', 'winters',
'woodlake', 'woodland', 'woodside', 'yorbalinda', 'yountville',
'yreka', 'yubacity', 'yucaipa', 'yuccavalley'], dtype=object)
```

We merge the dataframes together by mutual column **city**

In [18]:
```
okcupid_count = pd.merge(cities, users_okcupid , on="city")
okcupid_count.head()
```

Out[18]:

	Unnamed: 0	city	latd	longd	elevation_m	elevation_ft	population_total	area_t
0	2	alameda	37.756111	-122.274444	NaN	33.0	75467	
1	3	albany	37.886944	-122.297778	NaN	43.0	18969	
2	14	arcadia	34.132778	-118.036389	147.0	482.0	56364	
3	20	atherton	37.458611	-122.200000	18.0	59.0	6914	
4	34	belmont	37.518056	-122.291667	13.0	43.0	25835	

Visualization: Geographic Data Scatter Plot (Basemap)

Now that we have a dataframe containing each CA city in okcupid with it's respective geographical coordinates we make use of Basemap toolkit to visualize the population distribution within California.

First, we assing variables for the values of latitude, longitude, total square area (Km^2), and user counts in our new dataframe **okcupid_count**.

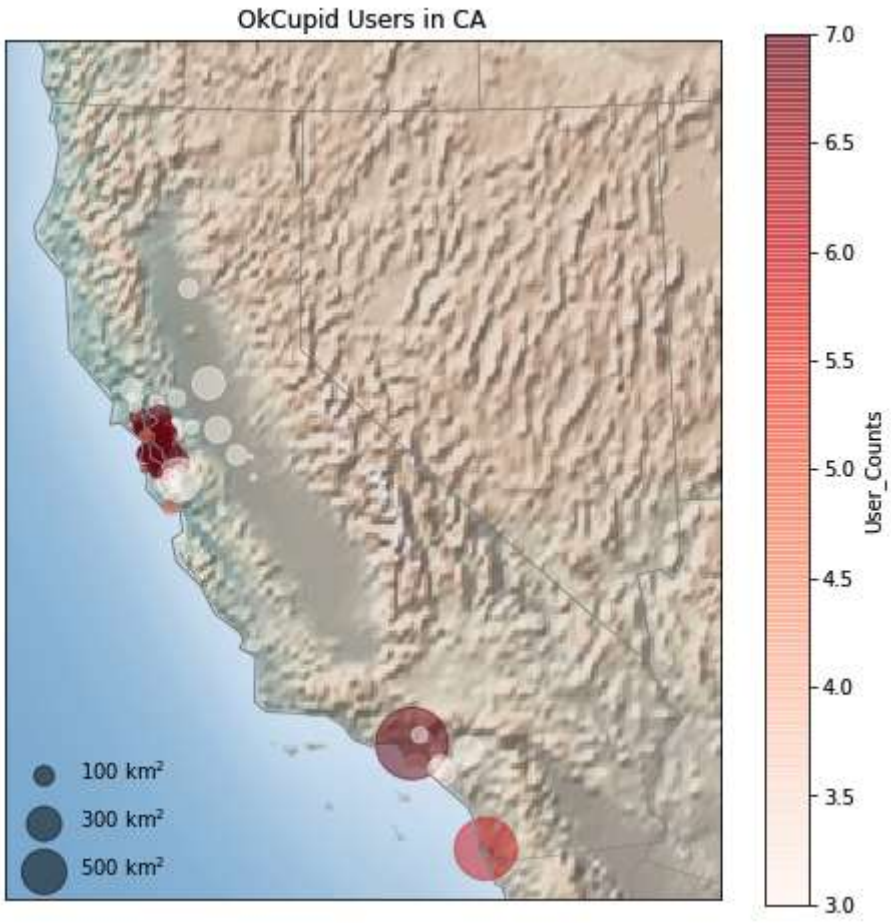
In [19]:
```
lat = okcupid_count['latd'].values
lon = okcupid_count['longd'].values
total_users = okcupid_count['users'].values
area = okcupid_count['area_total_km2'].values
```

Now, we plot our data into the basemap Geographical Scatter Plot.

In [20]:
```
# 1. Draw the map background
fig = plt.figure(figsize=(8, 8))
m = Basemap(projection='lcc', resolution='l',
            lat_0=37.5, lon_0=-119,
            width=1E6, height=1.2E6)
m.shadedrelief()
m.drawcoastlines(color='gray')
m.drawcountries(color='gray')
m.drawstates(color='gray')

# 2. Scatter city data, with color reflecting population and size reflecting a
rea
m.scatter(lon, lat, latlon=True,
          c=(total_users), s=area,
          cmap='Reds', alpha=0.5)

# 3. Create colorbar and legend
plt.colorbar(label=r'User_Counts')
plt.clim(3, 7)

# 4. Make legend with dummy points
for a in [100, 300, 500]:
    plt.scatter([], [], c='k', alpha=0.5, s=a,
                label=str(a) + ' km$^2$')
plt.legend(scatterpoints=1, frameon=False,
           labelspacing=1, loc='lower left');
plt.title('OkCupid Users in CA')
```

```
/opt/conda/lib/python3.6/site-packages/ipykernel_launcher.py:5: MatplotlibDep
recationWarning:


The dedent function was deprecated in Matplotlib 3.1 and will be removed in
3.3. Use inspect.cleandoc instead.

/opt/conda/lib/python3.6/site-packages/ipykernel_launcher.py:5: MatplotlibDep
recationWarning:


The dedent function was deprecated in Matplotlib 3.1 and will be removed in
3.3. Use inspect.cleandoc instead.

/opt/conda/lib/python3.6/site-packages/ipykernel_launcher.py:8: MatplotlibDep
recationWarning:


The dedent function was deprecated in Matplotlib 3.1 and will be removed in
3.3. Use inspect.cleandoc instead.

/opt/conda/lib/python3.6/site-packages/ipykernel_launcher.py:9: MatplotlibDep
recationWarning:


The dedent function was deprecated in Matplotlib 3.1 and will be removed in
3.3. Use inspect.cleandoc instead.
```

Out[20]: `Text(0.5, 1.0, 'OkCupid Users in CA')`



From our visualization above we learn that okcupid users tend to reside mostly in rural cities accross the coast of California such as San Fransico, Los Angeles, and San Diego, meanwhile we see a lack of activity in suburban and western areas accross the state.

Missing Values

At this point, we made the decision to remove NA values from our dataframe. Before deleting these rows, we compared the shape of the data before and after removing NA values. We decided that although we were only left with 7.4% of our data, we still had a big enough sample to proceed with.

In [21]:
```
okcupid_ca.shape
```

Out[21]: (59855, 31)

In [22]:
```
okcupid_ca = okcupid_ca.dropna()
okcupid_ca.shape
```

Out[22]: (4400, 31)

Distribution of Gender

As part of our preliminary analysis of the profiles, we examined the gender distribution between males and females on OkCupid. We found that about 56% of profiles were male, and 44% are female.

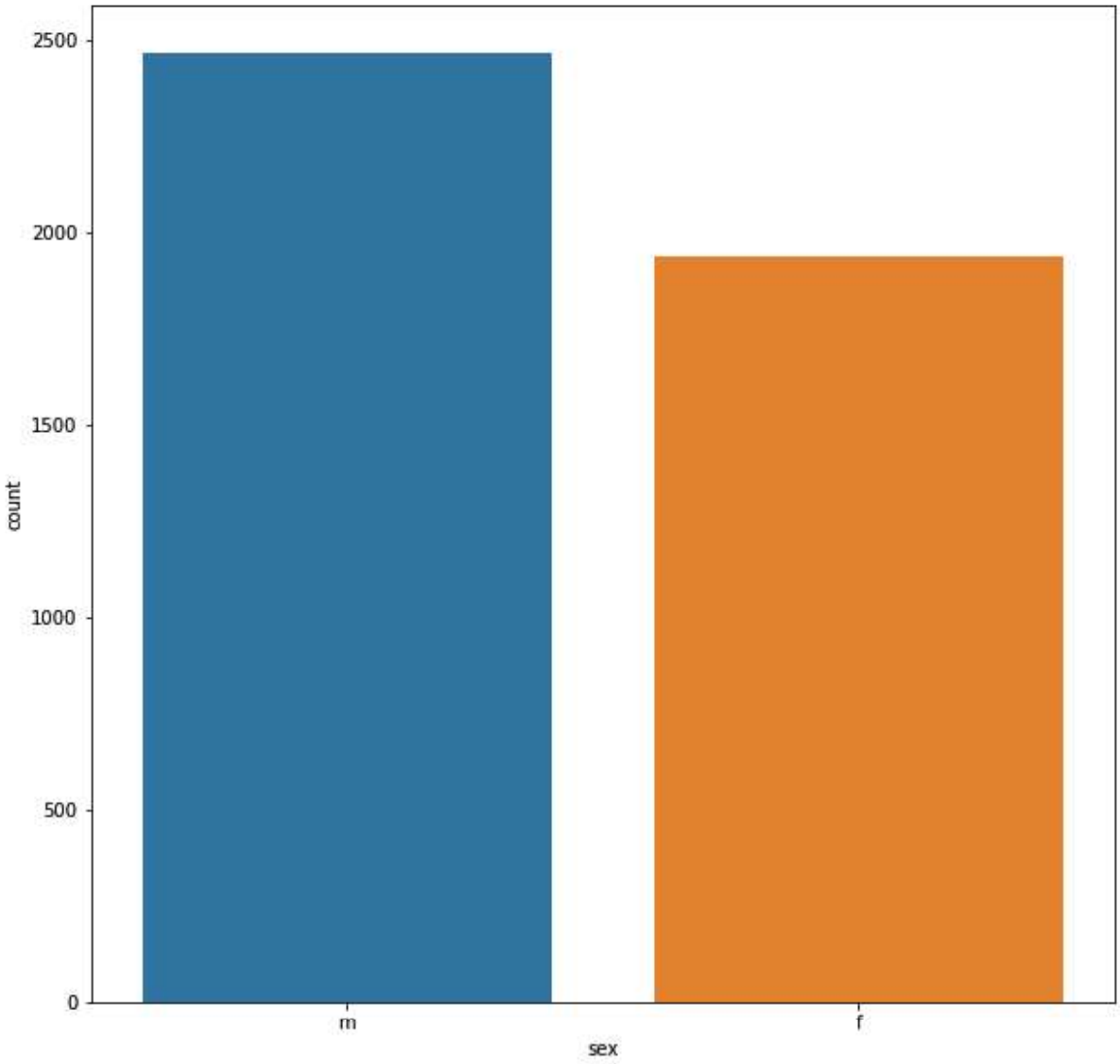
In [23]:
```
okcupid_ca["sex"].value_counts()
```

Out[23]:
```
m    2465
f    1935
Name: sex, dtype: int64
```

Visualization: Countplot

In [24]:
```
fig = plt.figure(figsize = (10, 10))
sns.countplot(x = "sex", data = okcupid_ca)
```

Out[24]: <matplotlib.axes._subplots.AxesSubplot at 0x7febe8794438>



Distribution of Age

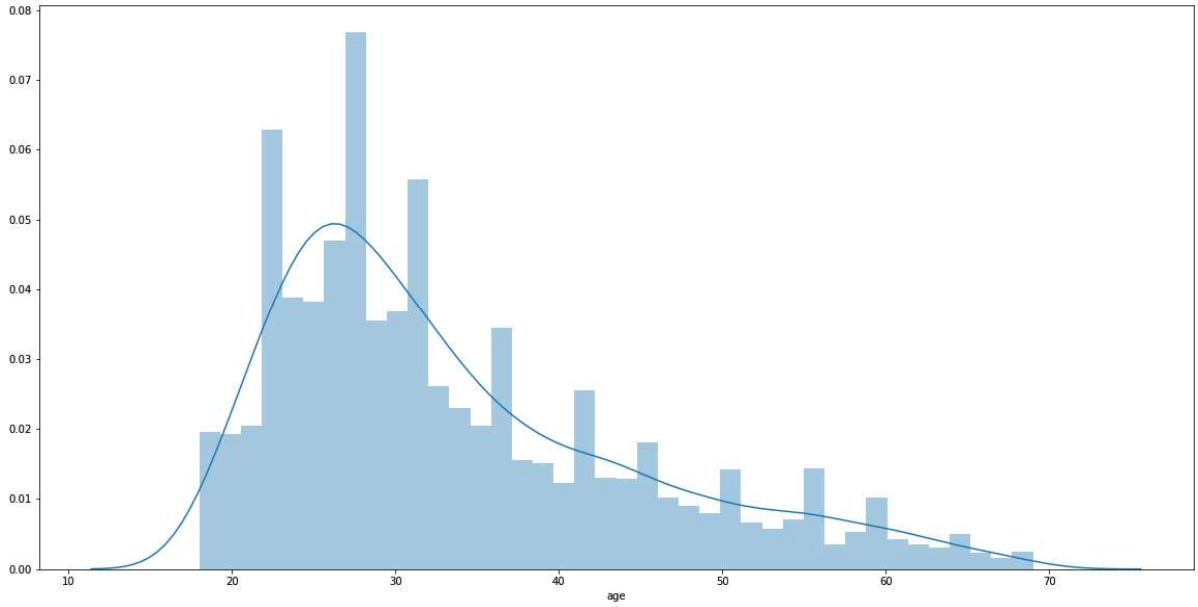
The second prelimiary analysis we looked at was the distribution of age. We can see that the majority of people on OkCupid are between the ages of 25 and 40, at an average of 33. The minimum age is 18 (you have to be 18 in order to register for an account), and the maximum is 69.

In [25]:
```
okcupid_ca["age"].describe()
```

Out[25]:
```
count    4400.000000
mean       33.659091
std        11.144330
min        18.000000
25%        25.000000
50%        30.000000
75%        40.000000
max        69.000000
Name: age, dtype: float64
```

In [26]:
```
fig = plt.figure(figsize = (20, 10))
sns.distplot(okcupid_ca["age"], bins = 40)
```

Out[26]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7febe317c668>
```



Distribution of Essay Response Polarity

Okcupid gives its users the option to describe themselves and their lives through ten distinctive essays that capture their characteristics and live styles. Each essay corresponds to a question regarding different topics of interest that others can read to make judgements on that person based on them.

We wish to analyse the essays that best describe each user and appear to have the biggest impact in the decision making for others to decide whether they will be a good match or not. Through, the analysis of the polarity in each essay we will make our decision of what essays best contain the most positive descriptions of each user and use these in order to predict their best matches.

We will create a polarity data set from Vader lexicon text that contains the measurement of sentiment per character.

Loading Polarity Data into Jupyter

In [27]:
```
sent = pd.read_csv("vader_lexicon.txt", header = None, names = ["word", "polar
ity"], sep = "\t", index_col = [0], usecols = [0, 1])
sent.head()
```

Out[27]:

word	polarity
$:	-1.5
%)	-0.4
%-)	-1.5
&-:	-0.4
&:	-0.7

Isolating the Text in Each Essay

From the 10 essays provided we decided to choose only 6 that appear to have the best descriptions and reliability of the characteristics and live styles of each person that best describes them.

We use string methods to polish each essay as to contain no other characters but those of words.

In [28]:
```
okcupid_ca["My self"] = (okcupid_ca["essay0"]
                         .str.lower()
                         .str.replace("<br />\n", ""))
punct_re = r"[^\s\w]"
okcupid_ca['no_punc0'] = okcupid_ca["My self"].str.replace(punct_re, " ")
```

In [29]:
```
okcupid_ca["My life"] = (okcupid_ca["essay1"]
                         .str.lower()
                         .str.replace("<br />\n", ""))
okcupid_ca['no_punc1'] = okcupid_ca["My life"].str.replace(punct_re, " ")
```

In [30]:
```
okcupid_ca["Noticeable traits"] = (okcupid_ca["essay3"]
                         .str.lower()
                         .str.replace("<br />\n", ""))
okcupid_ca['no_punc3'] = okcupid_ca["Noticeable traits"].str.replace(punct_re,
" ")
```

In [31]:
```
okcupid_ca["Thoughts"] = (okcupid_ca["essay6"]
                         .str.lower()
                         .str.replace("<br />\n", ""))
okcupid_ca['no_punc6'] = okcupid_ca["Thoughts"].str.replace(punct_re, " ")
```

In [32]:
```
okcupid_ca["Friday night"] = (okcupid_ca["essay7"]
                        .str.lower()
                        .str.replace("<br />\n", ""))
okcupid_ca['no_punc7'] = okcupid_ca["Friday night"].str.replace(punct_re, " ")
```

In [33]:
```
okcupid_ca["Secrets"] = (okcupid_ca["essay8"]
                        .str.lower()
                        .str.replace("<br />\n", ""))
okcupid_ca['no_punc8'] = okcupid_ca["Secrets"].str.replace(punct_re, " ")
```

Creating Tidy Formats

We will convert each essay into dataframes labeled as $\mathbf{tidy\_formats}$ that split each word in each text and assign numerical values from zero to the lenght of the string correspoding to the position of each word.

In [34]:
```
tidy_format0 = okcupid_ca["no_punc0"].str.split(expand = True).stack().reset_i
ndex(level = 1).rename(columns = {"level_1":"num", 0:"word"})
tidy_format1 = okcupid_ca["no_punc1"].str.split(expand = True).stack().reset_i
ndex(level = 1).rename(columns = {"level_1":"num", 0:"word"})
tidy_format3 = okcupid_ca["no_punc3"].str.split(expand = True).stack().reset_i
ndex(level = 1).rename(columns = {"level_1":"num", 0:"word"})
tidy_format6 = okcupid_ca["no_punc6"].str.split(expand = True).stack().reset_i
ndex(level = 1).rename(columns = {"level_1":"num", 0:"word"})
tidy_format7 = okcupid_ca["no_punc7"].str.split(expand = True).stack().reset_i
ndex(level = 1).rename(columns = {"level_1":"num", 0:"word"})
tidy_format8 = okcupid_ca["no_punc8"].str.split(expand = True).stack().reset_i
ndex(level = 1).rename(columns = {"level_1":"num", 0:"word"})
```

Looking at the Polarity of Each Essay

Tidy format allows a smooth transition to find the polarity of each essay and assing this into a new column.

In [35]:
```
okcupid_ca['polarity0'] = (
    tidy_format0
    .merge(sent, how='left', left_on='word', right_index=True)
    .reset_index()
    .fillna(0)
    .groupby("index")
    .sum()["polarity"]
)
okcupid_ca[['My self', 'polarity0']].head()
```

Out[35]:

	My self	polarity0
0	about me:i would love to think that i was some...	31.9
19	i relocated to san francisco half a year ago. ...	44.4
22	i tend to think the same way a comedian does a...	8.2
94	my names josh, and i create art for a living. ...	8.9
98	one day i will mod r/hotchickswithspreadsheets...	2.3

In [36]:
```
okcupid_ca['polarity1'] = (
    tidy_format1
    .merge(sent, how='left', left_on='word', right_index=True)
    .reset_index()
    .fillna(0)
    .groupby("index")
    .sum()["polarity"]
)
okcupid_ca[['My life', 'polarity1']].head()
```

Out[36]:

	My life	polarity1
0	currently working as an international agent fo...	5.5
19	i left my comfort zone far behind in europe, a...	10.4
22	i'm a supply and demand manager for a sustaina...	3.4
94	living it	0.0
98	presently, holyshitwhatamidoinghere and the sp...	0.0

In [37]:
```
okcupid_ca['polarity3'] = (
    tidy_format3
    .merge(sent, how='left', left_on='word', right_index=True)
    .reset_index()
    .fillna(0)
    .groupby("index")
    .sum()["polarity"]
)
okcupid_ca[['Noticeable traits', 'polarity3']].head()
```

Out[37]:

	Noticeable traits	polarity3
0	the way i look. i am a six foot half asian, ha...	-0.5
19	cheerful, open, curious, direct, active, sport...	7.4
22	that i'm smiling. pretty much at all times.	4.2
94	i honestly couldn't say....	2.0
98	i look a lot like that chick you banged your f...	1.5

In [38]:
```
okcupid_ca['polarity6'] = (
    tidy_format6
    .merge(sent, how='left', left_on='word', right_index=True)
    .reset_index()
    .fillna(0)
    .groupby("index")
    .sum()["polarity"]
)
okcupid_ca[['Thoughts', 'polarity6']].head()
```

Out[38]:

	Thoughts	polarity6
0	duality and humorous things	1.6
19	my passions and searching for new ones.	2.2
22	how to build an empire and a legacy. so far th...	4.5
94	the world	0.0
98	macros. also, puppies. i worry about the futur...	-0.6

In [39]:
```
okcupid_ca['polarity7'] = (
    tidy_format7
    .merge(sent, how='left', left_on='word', right_index=True)
    .reset_index()
    .fillna(0)
    .groupby("index")
    .sum()["polarity"]
)
okcupid_ca[['Friday night', 'polarity7']].head()
```

Out[39]:

	Friday night	polarity7
0	trying to find someone to hang out with. i am ...	0.0
19	having dinner and a night out with my close fr...	2.1
22	having beers with any number of friends and pr...	2.8
94	out.	0.0
98	home, high, and doing the dishes.	0.0

In [40]:
```
okcupid_ca['polarity8'] = (
    tidy_format8
    .merge(sent, how='left', left_on='word', right_index=True)
    .reset_index()
    .fillna(0)
    .groupby("index")
    .sum()["polarity"]
)
okcupid_ca[['Secrets', 'polarity8']].head()
```

Out[40]:

	Secrets	polarity8
0	i am new to california and looking for someone...	0.0
19	i'm pretty direct, so you can ask and i'll tel...	2.2
22	when i was a kid i thought steven segal was re...	2.6
94	no	-1.2
98	i'm terrible at small talk which makes for awk...	-2.7

Here we can see the columns in our dataframe that show the polarity for each essay.

In [48]:
```
okcupid_ca[["polarity0", "polarity1", "polarity3",  "polarity6", "polarity7",
"polarity8"]].head()
```

Out[48]:

	polarity0	polarity1	polarity3	polarity6	polarity7	polarity8
0	31.9	5.5	-0.5	1.6	0.0	0.0
19	44.4	10.4	7.4	2.2	2.1	2.2
22	8.2	3.4	4.2	4.5	2.8	2.6
94	8.9	0.0	2.0	0.0	0.0	-1.2
98	2.3	0.0	1.5	-0.6	0.0	-2.7

Fill NA Values

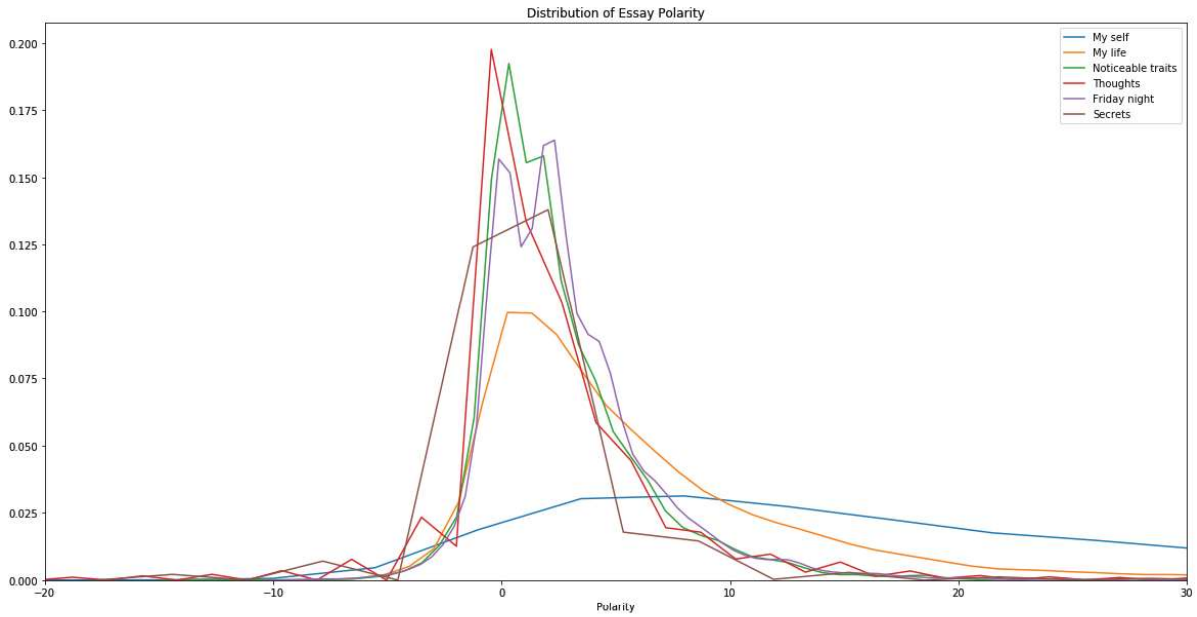
We replace each NaN value with 0 to adapt our numberical values into a distribution plot

In [42]:
```
okcupid_ca["polarity1"] = okcupid_ca["polarity1"].fillna(0)
okcupid_ca["polarity3"] = okcupid_ca["polarity3"].fillna(0)
okcupid_ca["polarity6"] = okcupid_ca["polarity6"].fillna(0)
okcupid_ca["polarity7"] = okcupid_ca["polarity7"].fillna(0)
okcupid_ca["polarity8"] = okcupid_ca["polarity8"].fillna(0)
```

Visualization: Distribution Plot

In [43]:
```
fig = plt.figure(figsize = (20, 10))
sns.distplot(okcupid_ca["polarity0"], label = 'My self', hist = False)
sns.distplot(okcupid_ca["polarity1"], label = 'My life', hist = False)
sns.distplot(okcupid_ca["polarity3"], label = 'Noticeable traits', hist = Fals
e)
sns.distplot(okcupid_ca["polarity6"], label = 'Thoughts', hist = False)
sns.distplot(okcupid_ca["polarity7"], label = 'Friday night', hist = False)
sns.distplot(okcupid_ca["polarity8"], label = 'Secrets', hist = False)
plt.xlim([-20,30])
plt.title("Distribution of Essay Polarity")
plt.xlabel("Polarity")
```

Out[43]: Text(0.5, 0, 'Polarity')



In [49]:
```
polarity_description = okcupid_ca[["polarity0", "polarity1", "polarity3", "pol
arity6", "polarity7", "polarity8"]].describe()
polarity_description
```

Out[49]:

	polarity0	polarity1	polarity3	polarity6	polarity7	polarity8
count	4400.000000	4400.000000	4400.000000	4400.000000	4400.000000	4400.000000
mean	20.160545	6.038250	2.555523	2.405341	2.880477	1.393432
std	22.807442	8.379375	3.845369	5.140514	3.696178	7.115482
min	-27.600000	-10.400000	-13.200000	-18.700000	-8.500000	-16.700000
25%	5.900000	0.900000	0.000000	0.000000	0.000000	0.000000
50%	14.200000	3.600000	1.500000	1.300000	2.100000	0.100000
75%	27.500000	8.400000	3.900000	3.700000	4.300000	2.300000
max	525.900000	119.300000	80.100000	172.200000	49.800000	399.300000

The distribution of polarity for each essay demonstrates that most texts tend to have a polarity near zero, however the texts in **"Myself"** and **"Mylife"** appear to be the essays with the most positive distribution of polarity as shown in this plot.

This analysis to find the essays with the most positive polarity contributed to our desicion of basing our predictions on essay 0 (**"Myself"**) and essay 1 (**"Mylife"**).

Looking at Most Frequently Used Words

We will now look at the most frequently used words when people spoke about themselves and their lives. We created a new column, **info**, which combined essay 0, **"MySelf"**, and essay 1, **"MyLife"**, together, stripping both from their html tags and any punctuation.

In [50]:
```
okcupid_ca["info"] = okcupid_ca["essay0"].str.replace(r"<.*>", "") + okcupid_c
a["essay1"].str.replace(r"<.*>", "")
okcupid_ca["info"].str.replace(r"\n", "").head()
```

Out[50]:
```
0     about me:i would love to think that i was some...
19    i relocated to san francisco half a year ago. ...
22    i tend to think the same way a comedian does a...
94    my names josh, and i create art for a living. ...
98    one day i will mod r/hotchickswithspreadsheets...
Name: info, dtype: object
```

Preprocessing Essay Text

In order to identify the key words in **info**, we preprocessed the text by creating a new column called **keywords** that contained all the text from **info**, then converted all words to lower case, removed numbers and punctuation, tokenized, and removed stop words.

In [51]:
```
stop_words = set(stopwords.words('english'))
punct = string.punctuation
```

In [52]:
```
okcupid_ca['key_words'] = "" # Add a new column
okcupid_ca['key_words'] = np.nan

# Convert to lowercase and remove leading/trailing whitespace
okcupid_ca['key_words'] = (okcupid_ca['info']
 .str.lower()
 .str.strip()
)

# Remove Punctuation
punct_regex = r"[^\s\w]"
okcupid_ca['key_words'] = okcupid_ca['key_words'].str.replace(punct_regex, " "
)

# Remove Numbers
num_regex = r"\d"
okcupid_ca['key_words'] = okcupid_ca['key_words'].str.replace(num_regex, " ")

# Extract the keywords (exclude stop words)
okcupid_ca['key_words'] = (okcupid_ca['key_words']
                           .apply(lambda x: ' '.join([word for word in x.split() i
f word not in stop_words])))

# Tokenize the esays using word_tokenize
okcupid_ca['key_words'] = (okcupid_ca['key_words'].apply(word_tokenize))
```

Define a Dictionary of Keywords

We now have located all the keywords in essays 0 and 1. We can use these keywords to create a dictionary and visualize the frequency distribution of the top used words.

In [53]:
```
# Processing Keywords
processed_keywords=[]
for keywords in okcupid_ca['key_words']:
    processed_keywords.append(keywords)


dictionary = Dictionary(processed_keywords)
```

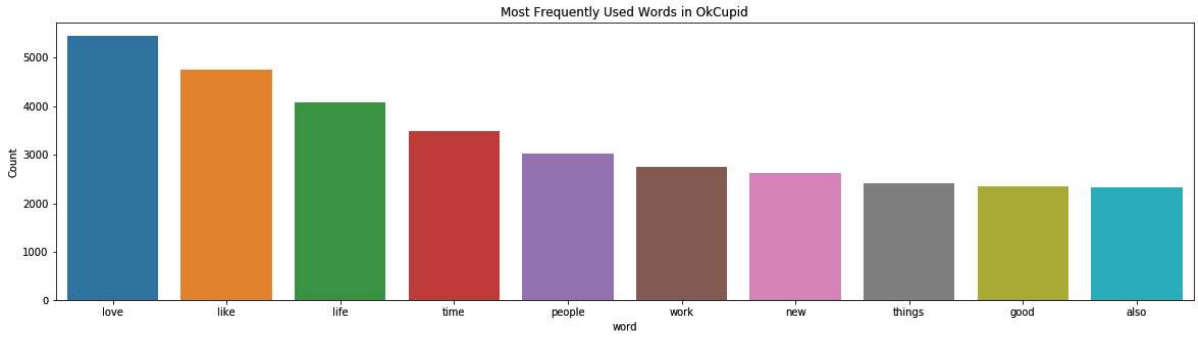
In [54]:
```
# Concatenate all keywords from all essays together
all_essay_words=[]
for word in processed_keywords:
    all_essay_words.extend(word)
```

Visualization: Bar Graph

Now we will plot the most frequently used words in these two merged essays to figure out which words okcupid users tend to use most often to describe themselves.

In [55]:
```
def plot_freq_words(words, terms = 10):

    fdist = FreqDist(words)
    words_df = pd.DataFrame({'word':list(fdist.keys()), 'count':list(fdist.val
ues())})

    # selecting top most frequent words
    d = words_df.nlargest(columns="count", n = terms)
    plt.figure(figsize=(20,5))
    ax = sns.barplot(data=d, x= "word", y = "count")
    ax.set(ylabel = 'Count')
    plt.title('Most Frequently Used Words in OkCupid')
    plt.show()
```

In [56]:
```
plot_freq_words(all_essay_words)
```



From the bar graph, we can see the most frequently used words for all users are as expected for a dating site. The most used word is "love" and "like" followed by words that represent positive sentiments. This plot gives us a great insight into what most people are looking for in this dating site such as experiencing "new things" in "life" and meeting "good people".

Predicting OkCupid Matches

Now that we have defined a dictionary of the words users use most frequently to describe themselves and their lives, we can now predict compatible OkCupid matches based on similarities in the language users chose to use in their essays.

In [57]:
```
corpus = [dictionary.doc2bow(keyword) for keyword in processed_keywords]
# Create the tf-idf model for the corpus
tfidf = TfidfModel(corpus)

# Create the similarity data structure.
# This is the most important part where we get the similarities between the es
says.
sims = MatrixSimilarity(tfidf[corpus])
```

Since we do not have any identification specific to an individual user, we will create a new column from the index labeled **user_id** to assign each user an id. In this case, we decided to identify users by their row number.

In [58]:
```
okcupid_ca = okcupid_ca.reset_index().rename(columns = {"index":"temp"})
okcupid_ca = okcupid_ca.reset_index().rename(columns = {"index":"user_id"})
okcupid_ca = okcupid_ca.drop("temp", axis = 1)
okcupid_ca.head()
```

Out[58]:

	user_id	age	body_type	diet	drinks	drugs	education	es
0	0	22	a little extra	strictly anything	socially	never	working on college/university	about me: \n<br would love to th
1	1	33	athletic	mostly anything	socially	never	graduated from masters program	i relocated to san fran half a year a
2	2	30	fit	mostly anything	socially	never	graduated from college/university	i tend to think the same v comedian doe
3	3	29	fit	mostly anything	socially	sometimes	graduated from college/university	my names josh, and i c art for a livi
4	4	31	curvy	anything	socially	sometimes	graduated from masters program	one day i wil r/hotchickswithspreadshe

5 rows × 52 columns

We have created a function that takes in any **user_id**, prints out the top words in their essay0 & essay1 along side their tf-idf score and the top 10 most similar matches for the given user.

In [60]:
```
def match_recommendation(user, dictionary, number_of_hits=10):
    top_words = 5
    # We will first start by  getting all the keywords related to the user's e
ssay
    okcupid_match = okcupid_ca.loc[okcupid_ca.user_id==user] # get the user ro
w
    keywords = okcupid_match['key_words']# Get all the keywords
    doc=[]
    for word in keywords:
        doc.extend(word)

    # Convert the doc into it's equivalent bag of words
    query_doc_bow = dictionary.doc2bow(doc)

    # convert the regular bag of words model to a tf-idf model where we have t
uples
    # of the user ID and its tf-idf value for the essay
    query_doc_tfidf = tfidf[query_doc_bow]

    # get the array of similarity values between our user and every other use
r.
    # To do this, we pass our list of tf-idf tuples to sims.
    similarity_array = sims[query_doc_tfidf]
    # the length is the number of users we have.

    similarity_series = pd.Series(similarity_array.tolist(), index=okcupid_ca.
user_id.values) #Convert to a Series
    top_hits = similarity_series.sort_values(ascending=False)[1:number_of_hits
+1]
    #get the top matching results, i.e. most similar users;
    # start from index 1 because every user is most similar to itself

    #print the words with the highest tf-idf values for the provided essay:
    sorted_tfidf_weights = sorted(tfidf[corpus[okcupid_match.index.values.toli
st()[0]]], key=lambda w: w[1], reverse=True)
    print('Top %s words associated with this user by tf-idf are: ' % top_words
)
    for term_id, weight in sorted_tfidf_weights[:top_words]:
        print(" '%s' (tf-idf score = %.3f)" %(dictionary.get(term_id), weight
))
    print("\n")

    # Print the top matches
    print("Top %s most similar matches for user %s are:" %(number_of_hits, use
r))
    top_matches=[]
    for idx, (okcupid_match,score) in enumerate(zip(top_hits.index, top_hits
)):
        print("%d %s (similarity score = %.3f)" %(idx+1, okcupid_match, score
))
        top_matches.append(okcupid_match)
    return top_matches
```

We tested our function out on user number 9. We can see user 9 has similarity score to user 1984 of 15.6%.

In [79]:
```
top_10_matches = match_recommendation(459, dictionary)
```

```
Top 5 words associated with this user by tf-idf are:
 'ilink' (tf-idf score = 0.427)
 'href' (tf-idf score = 0.417)
 'class' (tf-idf score = 0.389)
 'interests' (tf-idf score = 0.342)
 'bees' (tf-idf score = 0.130)


Top 10 most similar matches for user 459 are:
1 1774 (similarity score = 0.768)
2 3468 (similarity score = 0.754)
3 1362 (similarity score = 0.747)
4 4392 (similarity score = 0.736)
5 2962 (similarity score = 0.699)
6 527 (similarity score = 0.688)
7 176 (similarity score = 0.685)
8 562 (similarity score = 0.672)
9 769 (similarity score = 0.671)
10 314 (similarity score = 0.670)
```

Distribution of Religion Associations

Next we examined the distribution of religious associations. When setting up an OkCupid profile, users have the option of selecting the religious organization they associate with (or atheism or agnosticism) and specifying the degree of how religious they are. To start off, we wanted an overview of the various religious, stripping away to what degree of religious a person claims to be.

In [56]:
```
okcupid_ca["religion"].head()
```

Out[56]:
```
0        agnosticism and very serious about it
1     catholicism but not too serious about it
2    agnosticism and somewhat serious about it
3        agnosticism and very serious about it
4                                     atheism
Name: religion, dtype: object
```

Grouping Religions

We defined a function, firstword(), that selects the first word from the string of words in the **religion** column. This allowed us to create a new column called **religions** that grouped all responses by religious organization.

In [57]:
```
def firstword(text):
  '''
  Takes in a string of text, splits the string into a list of individual
  words, and returns the first word.
  '''
  words = text.split()
  first = words[0]
  return first

firstword("agnosticism but not too serious about it") # Testing
```

Out[57]:
```
'agnosticism'
```

In [58]:
```
okcupid_ca["religions"] = okcupid_ca["religion"].apply(firstword)
okcupid_ca["religions"].head()
```

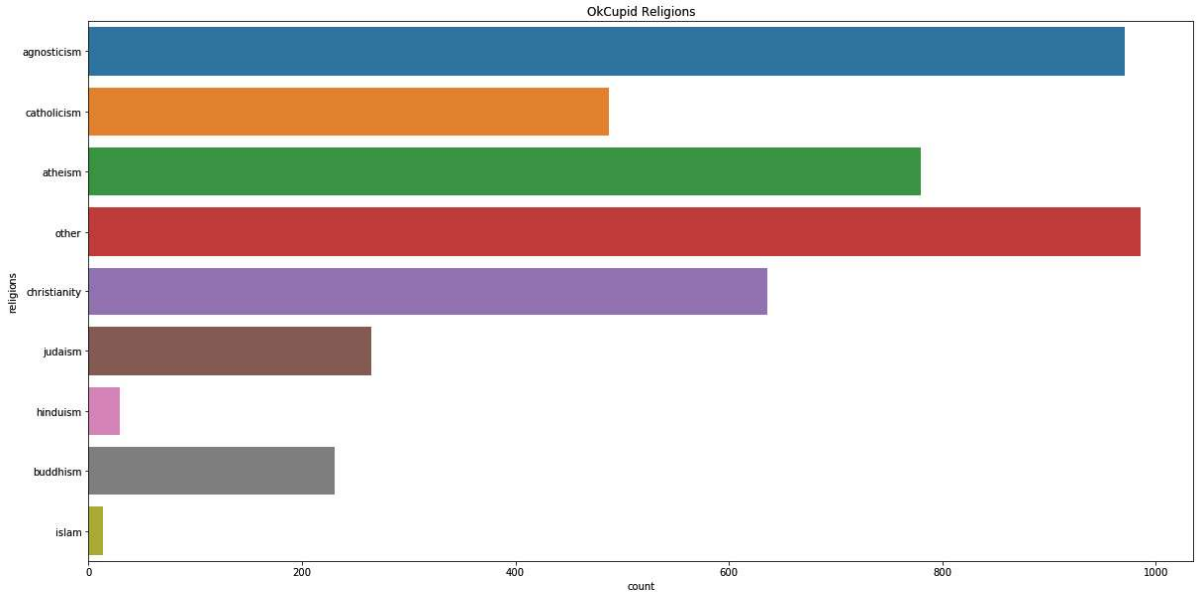
Out[58]:
```
0    agnosticism
1    catholicism
2    agnosticism
3    agnosticism
4        atheism
Name: religions, dtype: object
```

Visualization: Countplot

From the countplot below, we can see that the number one religion people identified with on their OkCupid profile was "other", followed closely by "agnosticism." This is not that surprising of a result, as all the OkCupid profiles we are looking at at this point are in California, and people tend to have increased moderate views on religion.

In [59]:
```
fig = plt.figure(figsize = (20, 10))
sns.countplot(y = "religions", data = okcupid_ca)
plt.title('OkCupid Religions')
```

Out[59]: Text(0.5, 1.0, 'OkCupid Religions')



Distribution of Drug Usage

Now we will look at people's responses when asked if they do drugs. About 78% of poeple say they never do drugs, 21% say they do drugs occassionally, and only 1.5% say they do drugs often. We noted that these responses may be biased, as people may not want to answer honestly to this sort of question on the internet.

In [60]:
```
okcupid_ca["drugs"].value_counts()
```

Out[60]:
```
never        3419
sometimes     913
often          68
Name: drugs, dtype: int64
```

Examining the Relationship Between Drug Usage and Religion

Next, we were curious to see if there was any relationship between drug usage and religion.

Quantifying Drug Usage Responses

In order to examine linear relationship, we decided to quantify the three possible responses for drug usage. We defined a new column, " **druggie** ", that includes to what level a person uses drugs. Someone who claimed to never use drugs recieved a 0, someone who claimed to sometimes use drugs received a 0.5, and someone who claimed to use drugs often recieved a 1.

In [61]:
```
okcupid_ca["druggie"] = (okcupid_ca["drugs"]
                         .str.replace('sometimes', '.5')
                         .str.replace('often', '1')
                         .str.replace('never','0'))
okcupid_ca["druggie"].to_frame()
okcupid_ca["druggie"].value_counts()
```

Out[61]:
```
0     3419
.5     913
1       68
Name: druggie, dtype: int64
```

Defining Our Predictors and Response Variables

In order to do prediction with religion, a categorical variable, we have to encode each value in the " **religions** " column as a binary vector indicating the non-numerical feature. We do this using dummy variables, which assigns a value of 0 or 1 to each column.

In [62]:
```
# Religious organization is our predictor.
x = okcupid_ca["religions"]
x = pd.get_dummies(data = x, drop_first = True)
x.head()
```

Out[62]:

	atheism	buddhism	catholicism	christianity	hinduism	islam	judaism	other
0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0
2	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0
4	1	0	0	0	0	0	0	0

In [80]:
```
# Our response variable is how frequently users do drugs.
y = okcupid_ca["druggie"]
y.head()
```

Out[80]:
```
0    0.0
1    0.0
2    0.0
3    0.5
4    0.5
Name: druggie, dtype: float64
```

Linear Regression Model

Now that we have defined our predictor and response, we can fit our linear regression model. First, we split our data into training data and testing data.

In [66]:
```
rel_tr, rel_te = train_test_split(x, test_size = .25)
print("Training Data Size: ", len(rel_tr))
print("Test Data Size: ", len(rel_te))
```

```
Training Data Size:  3300
Test Data Size:  1100
```

Using the sklearn module in the LinearRegression package, we build a least squares model and fit the data.

In [67]:
```
least_squares_model = sklearn.linear_model.LinearRegression()
least_squares_model.fit(x, y)
```

Out[67]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
         normalize=False)
```

Lastly, we calculate the root mean square error in our model. Mean square error is a measure of how close a fitted line is to our data points. The root mean square error is the distance, on average, of a data point from the fitted line, measured along a vertical line.

In [81]:
```
rmse = np.sqrt((mse(y, least_squares_model.predict(x))))
print("Root Mean Square Error:",rmse )
```

```
Root Mean Square Error: 0.22353750151119015
```

An RMSE of 0.22 is fairly high, indicating that religion is a weak predictor of drug usage.

Grouping Degree of Religious Views

Now, instead of looking at the specific religious organization users identify with, we look at the varying degrees of religious views. First, we looked at all the possible responses, and stripped away religious organization so all that was left was how religious a person claims to be.

In [69]:
```
okcupid_ca["religion"].unique()
```

Out[69]:
```
array(['agnosticism and very serious about it',
       'catholicism but not too serious about it',
       'agnosticism and somewhat serious about it', 'atheism',
       'agnosticism and laughing about it', 'other',
       'christianity and somewhat serious about it',
       'atheism and laughing about it',
       'catholicism and laughing about it',
       'judaism but not too serious about it',
       'other but not too serious about it',
       'hinduism but not too serious about it',
       'christianity but not too serious about it',
       'christianity and very serious about it',
       'other and somewhat serious about it',
       'buddhism but not too serious about it',
       'agnosticism but not too serious about it',
       'buddhism and laughing about it', 'other and laughing about it',
       'agnosticism', 'christianity',
       'catholicism and somewhat serious about it',
       'atheism but not too serious about it',
       'other and very serious about it',
       'atheism and somewhat serious about it', 'catholicism',
       'judaism and laughing about it',
       'atheism and very serious about it',
       'christianity and laughing about it', 'islam',
       'judaism and somewhat serious about it', 'buddhism',
       'hinduism and somewhat serious about it',
       'buddhism and somewhat serious about it',
       'islam and somewhat serious about it',
       'catholicism and very serious about it', 'judaism',
       'hinduism and very serious about it',
       'judaism and very serious about it',
       'buddhism and very serious about it',
       'islam but not too serious about it',
       'islam and very serious about it',
       'hinduism and laughing about it', 'hinduism',
       'islam and laughing about it'], dtype=object)
```

We defined a dictionary to rate how religious a person is on a scale from -1 to 1. A person who claims to be very atheist recieves a score of -1, a person who claims to be agnostic recieves a score of 0, and a person who claims to be very religious (for any religious organization) recieves a score of 1. People who fell between the two extremes recieved positive or negative decimal values, depending on their response.

In [70]:
```
d = {'agnosticism and very serious about it': 0,
     'catholicism but not too serious about it' : 0.25,
     'agnosticism and somewhat serious about it': 0,
     'atheism': -0.75,
     'agnosticism and laughing about it' : 0,
     'other' : 0,
     'christianity and somewhat serious about it' : 0.5,
     'atheism and laughing about it' : -.25,
     'catholicism and laughing about it' : 0.25,
     'judaism but not too serious about it' : 0.25,
     'other but not too serious about it' : 0,
     'hinduism but not too serious about it' : 0.25,
     'christianity but not too serious about it' : 0.25,
     'christianity and very serious about it' : 1,
     'other and somewhat serious about it' : 0,
     'buddhism but not too serious about it' : 0.25,
     'agnosticism but not too serious about it' : 0,
     'buddhism and laughing about it' : 0.25,
     'other and laughing about it' : 0,
     'agnosticism' : 0,
     'christianity' : 0.75,
     'catholicism and somewhat serious about it' : 0.5,
     'atheism but not too serious about it' : -.25,
     'other and very serious about it' : 0,
     'atheism and somewhat serious about it' : -.5,
     'catholicism' : 0.75,
     'judaism and laughing about it' : 0.25,
     'atheism and very serious about it' : -1,
     'christianity and laughing about it' : 0.25,
     'islam' : 0.75,
     'judaism and somewhat serious about it' : .5,
     'buddhism' : 0.75,
     'hinduism and somewhat serious about it' : 0.5,
     'buddhism and somewhat serious about it' : 0.5,
     'islam and somewhat serious about it' : 0.5,
     'catholicism and very serious about it' : 1,
     'judaism' : 0.75,
     'hinduism and very serious about it' : 1,
     'judaism and very serious about it' : 1,
     'buddhism and very serious about it': 1,
     'islam but not too serious about it' : 0.25,
     'islam and very serious about it' : 1,
     'hinduism and laughing about it' : 0.25,
     'hinduism' : 0.75,
     'islam and laughing about it' : 0.25}
```

We defined a new column called **" religious "** containing the transformed responses from **" religion "**. From computing the mean of this column, we can see that on average, OkCupid users are slightly more religious than agnostic.

In [71]:
```
okcupid_ca["religious"] = okcupid_ca["religion"].map(d)
okcupid_ca["religious"].mean()
```

Out[71]:
```
0.07431818181818182
```

Visualization: Scatterplot

In order to visualize our data, we convert columns " religious " and " druggie " into arrays.

In [73]:
```
religion_array = okcupid_ca[["religious"]].values
religion_array
```

Out[73]:
```
array([[0.  ],
       [0.25],
       [0.  ],
       ...,
       [0.  ],
       [0.  ],
       [0.  ]])
```

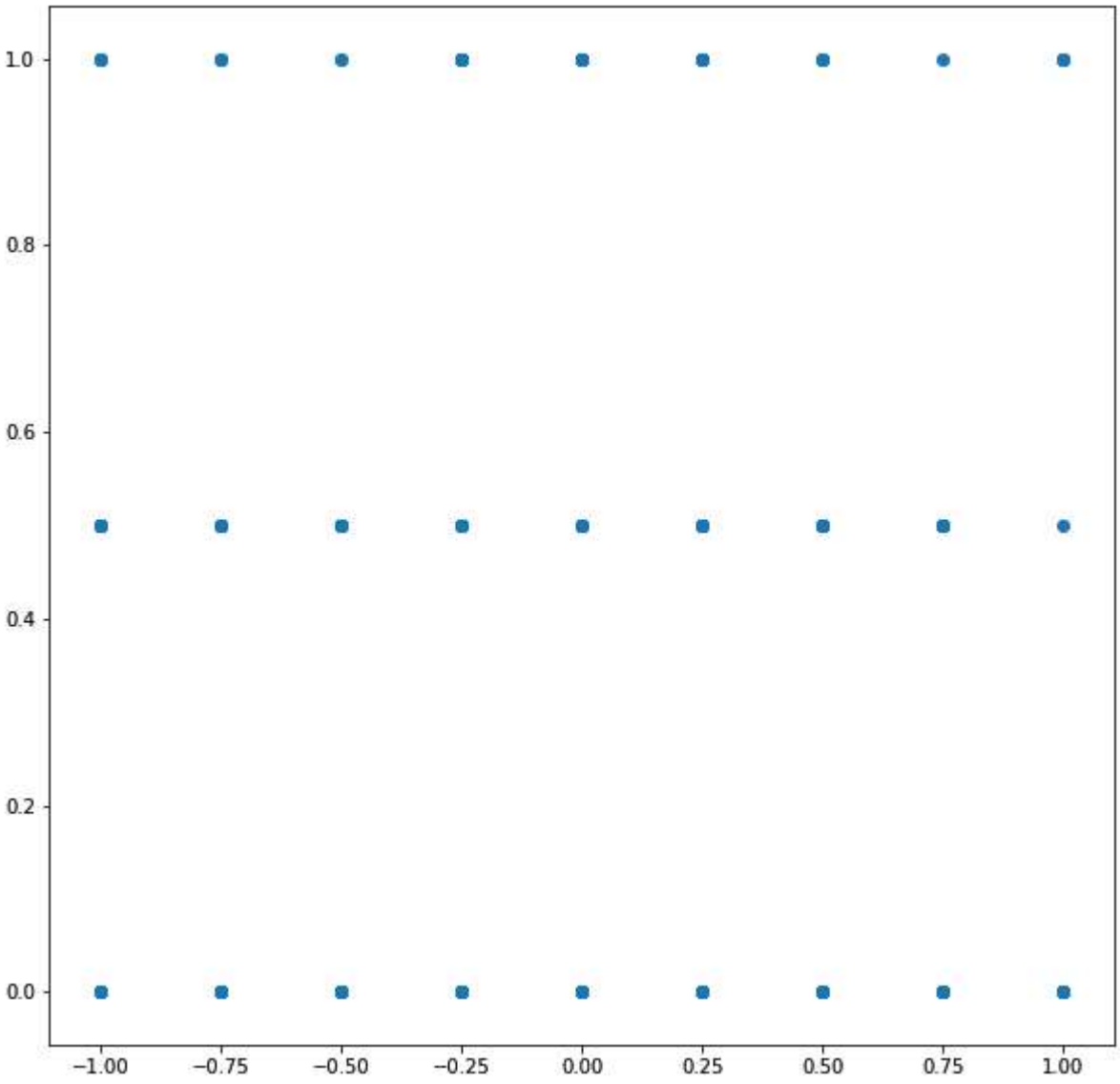
In [74]:
```
drug_array = okcupid_ca["druggie"].astype(float).values
drug_array
```

Out[74]:
```
array([0. , 0. , 0. , ..., 0. , 0.5, 1. ])
```

From our first attempt at creating a scatterplot, we noticed right away that there was too much overlapping to noticed any trends. We fix this by jittering our data.

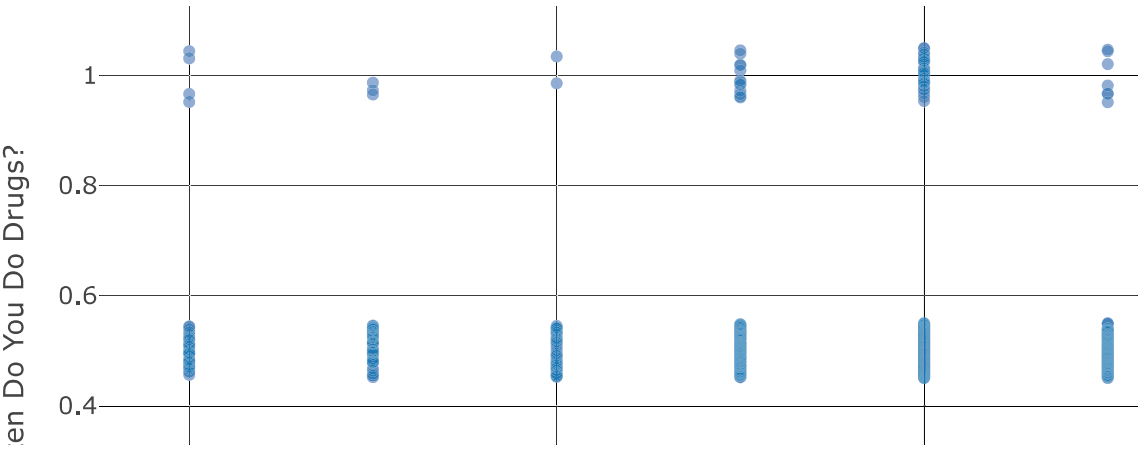
In [75]:
```
fig = plt.figure(figsize = (10, 10))
plt.scatter(religion_array, drug_array)
```

Out[75]: <matplotlib.collections.PathCollection at 0x7fb0368a2438>



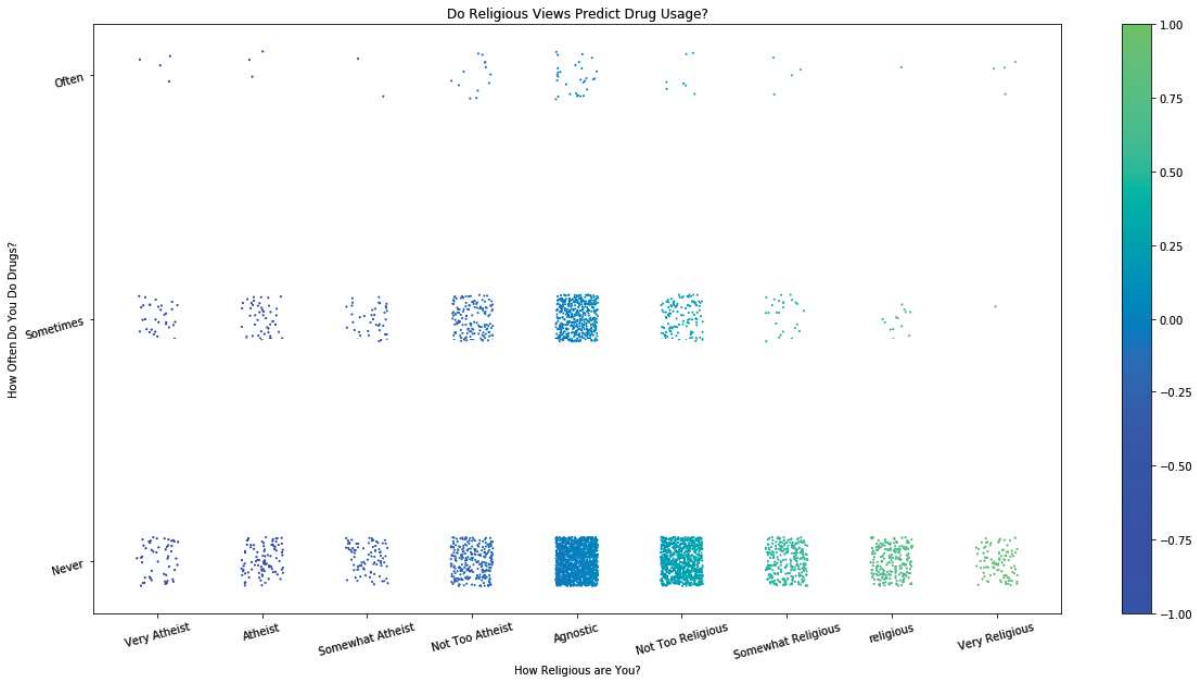
In [76]:
```
okcupid_ca["druggie"] = okcupid_ca["druggie"].astype(float)
okcupid_ca["religious"] = okcupid_ca["religious"].astype(float)
```

In [77]:
```
layout = dict(xaxis=dict(title="Do Religious Views Predict Drug Usage?"),yaxis
=dict(title="How Often Do You Do Drugs?"))
jitter_y = okcupid_ca['druggie'] + 0.1 * np.random.rand(okcupid_ca['druggie'].
size) -0.05
points = go.Scatter(x=okcupid_ca['religious'], y = jitter_y,
                    mode="markers",
                    marker=dict(opacity=0.5))
py.iplot(go.Figure(data=[points], layout=layout))
```



We started by only jittering y, but were still not satisfied, so we created one last scatterplot, jittering both x and y.

In [78]:
```
fig = plt.figure(figsize = (20, 10))
cm = plt.cm.get_cmap('winter')
jitter_y = okcupid_ca['druggie'] + 0.1 * np.random.rand(okcupid_ca['druggie'].
size) -0.05
jitter_x = okcupid_ca['religious'] + 0.1 * np.random.rand(okcupid_ca['religiou
s'].size) -0.05
points = plt.scatter(x = jitter_x, y = jitter_y, c = jitter_x, vmin = -1, vmax
= 1, s = 1, cmap = cm)
plt.colorbar(points)
plt.xticks((-1, -.75, -.5, -.25, 0, .25, .5, .75, 1), ('Very Atheist', 'Atheis
t','Somewhat Atheist', 'Not Too Atheist',  'Agnostic', 'Not Too Religious',
                    'Somewhat Religious', 'religious', 'Very Religious'))
plt.yticks((0, .5, 1), ('Never', 'Sometimes', 'Often'))
plt.xlabel("How Religious are You?")
plt.ylabel("How Often Do You Do Drugs?")
plt.yticks(rotation = 15)
plt.xticks(rotation = 15)
plt.title('Do Religious Views Predict Drug Usage?')
plt.show()
```



From our scatterplot, we can see the majority of our users never user drugs, and of those users, the majority are agnostic. Users that sometimes use drugs are mainly agnostic as well, however there are more athetiest users that sometimes use drugs than religious users. Very few users said they use drugs often; we see these responses distributed similarly to the responses for never using drugs. It is important to note that these responses may be biased, in the case that users did not want to admit to using drugs on their online dating profile.