CAPSTONE PROJECT: BATTLE OF THE NEIGHBORHOODS

Venue Recommendation for United States of America Visitor's

I. PURPOSE

This document provides the details of how i reached conclusion on mostly commonly visited place and also provide recommendation on best value stay while in USA

II. INTRODUCTION

There are lot of websites that scrapes different websites to provide us a comparison on places to stay or visit.However, most of these websites provides recommendation simply based on usual tourist attractions or key residential areas that are mostly expensive or already known for travelers based on certain keywords like "Hotel", or "Backpackers" etc. The intention on this project is to collect and provide a data driven recommendation that can supplement the recommendation with statistical data. This will also be utilizing data retrieved from New York open data sources and FourSquare API venue recommendations.

The sample recommender in this notebook will provide the following use case scenario:

- A person planning to visit United States as a Tourist or an Expat and looking for a reasonable accommodation.
- The user wants to receive venue recommendation where he or she can stay or rent with close proximity to places of interest or search category option.
- The recommendation should not only present the most viable option, but also present a comparison table of all possible town venues.

For this demonstration, this notebook will make use of the following data:

- Median Rental Prices by town.
- Popular Food venues in the vicinity. (Sample category selection)

Note: While this demo makes use of Food Venue Category, Other possible categories can also be used for the same implementation such as checking categories like:

- Outdoors and Recreation
- Nightlife
- Nearby Schools, etc.

I will limit the scope of this search as FourSquare API only allows 50 free venue query limit per day when using a free user access.

III. DATA ACQUISITION

This demonstration will make use of the following data sources:

USA median residential rental prices.

Data will retrieved from open dataset from [median rent by town and flattype (https://www.quandl.com/data/ZILLOW/M1300_MPPRSF-Zillow-Home-Value-Index-Metro-Median-Price-Of-Reduction-Single-Family-Residence-Canon-City-CO)](https://www.quandl.com/data/ZILLOW/M1300_MPPRSF-Zillow-Home-Value-Index-Metro-Median-Price-Of-Reduction-Single-Family-Residence-Canon-City-CO) from [https://www.quandl.com (https://www.quandl.com)](https://www.quandl.com) website.

The original data source contains median rental prices of Singapore HDB units from 2005 up to 2nd quarter of 2018. I will retrieve rental the most recent recorded rental prices from this data source (Q2 2018) being the most relevant price available at this time. For this demonstration, I will simplify the analysis by using the average rental prices of all available flat type.

Location data retrieved using Google maps API.

Data coordinates of Town Venues will be retrieved using google API. I also make use of MRT stations coordinate as a more important center of for all towns included in venue recommendations.

Top Venue Recommendations from FourSquare API

(FourSquare website: [www.foursquare.com (http://www.foursquare.com)](http://www.foursquare.com))

I will be using the FourSquare API to explore neighborhoods in selected towns in Singapore. The Foursquare explore function will be used to get the most common venue categories in each neighborhood, and then use this feature to group the neighborhoods into clusters. The following information are retrieved on the first query:

- Venue ID
- Venue Name
- Coordinates : Latitude and Longitude
- Category Name

Another venue query will be performed to retrieve venue ratings for each location. Note that rating information is a paid service from FourSquare and we are limited to only 50 queries per day. With this constraint, we limit the category analysis with only one type for this demo. I will try to retrieve as many ratings as possible for each retrieved venue ID.

IV. METHODOLOGY

United States Cities or Towns List with median residential rental prices obtained from New York free data source

The source data contains median rental prices of United States from 2008 up to of 2019. I will retrive the most recent recorded rental prices from this data source (Q2 2018) being the most relevant price available at this time. For this demonstration, I will simplify the analysis by using the average rental prices of all available flat type.

Data Cleanup and re-grouping. The retrieved table contains some un-wanted entries and needs some cleanup.

The following tasks will be performed:

- Drop/ignore cells with missing data.
- Use most current data record.

- Fix data types.

Importing Python Libraries

This section imports required python libraries for processing data.
While this first part of python notebook is for data acquisition, we will use some of the libraries make some data visualization.

In [4]:

```
#!conda install -c conda-forge folium=0.5.0 --yes # comment/uncomment if not yet installed.
#!conda install -c conda-forge geopy --yes        # comment/uncomment if not yet installed

import numpy as np # library to handle data in a vectorized manner
import pandas as pd # library for data analsysis

# Numpy and Pandas libraries were already imported at the beginning of this notebook.
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)

import json # library to handle JSON files
from geopy.geocoders import Nominatim # convert an address into latitude and longitude valu
from pandas.io.json import json_normalize # tranform JSON file into a pandas dataframe
# Matplotlib and associated plotting modules
import matplotlib.cm as cm
import matplotlib.colors as colors
# import k-means from clustering stage
from sklearn.cluster import KMeans
import folium # map rendering library

import requests # library to handle requests
import bs4 as bs
import urllib.request

print('Libraries imported.')
```

```
Libraries imported.
```

1. Downloading towns list with and median residential rental prices

In [5]:

```
data = pd.read_csv('Sale_Prices_Msa.csv')
data.head()
#Taking only region name and # Taking the most recent report which is "2019-04"

df = pd.DataFrame(data[['RegionName','2019-04']])
#renaming RegionName to City and year to median_rent
df.rename(columns = {'RegionName':'Town','2019-04':'median_rent'}, inplace = True)

df.head()
#sgp_median_rent_by_town_data.head()
```

Out[5]:

	Town	median_rent
0	United States	NaN
1	New York, NY	NaN
2	Los Angeles-Long Beach-Anaheim, CA	632800.0
3	Chicago, IL	244400.0
4	Dallas-Fort Worth, TX	NaN

Data Cleanup and re-grouping.

The retrieved table contains some un-wanted entries and needs some cleanup. The following tasks will be performed:

- Drop/ignore cells with missing data.
- Use most current data record.
- Fix data types.

In [6]:

```
# Drop rows with rental price == 'na'.
df.dropna(subset=['median_rent'],axis = 0,inplace = True)

#drop column index as it isnt required
# Ensure that median_rent column is float64.
df['median_rent']=df['median_rent'].astype(np.float64)

df = df.reset_index(drop=True)

df.head()
```

Out[6]:

	Town	median_rent
0	Los Angeles-Long Beach-Anaheim, CA	632800.0
1	Chicago, IL	244400.0
2	San Francisco, CA	789400.0
3	Riverside, CA	353000.0
4	Phoenix, AZ	247300.0

- Note:For this demonstration, We will do a simplier analysis by using a median price for all available rental units regardless of its size.

In [7]:

```
df_avg = df.groupby(['Town'])['median_rent'].mean().reset_index()
df_avg
```

Out[7]:

	Town	median_rent
0	Adrian, MI	145800.0
1	Akron, OH	141600.0
2	Albany, OR	258600.0
3	Anchorage, AK	311700.0
4	Astoria, OR	287700.0
5	Barnstable Town, MA	396700.0
6	Bartlesville, OK	113200.0
7	Bay City, MI	82200.0
8	Beaver Dam, WI	156200.0
9	Bellingham, WA	379500.0

- Adding geographical coordinates of each town location.

In [6]:

```
# The code was removed by Watson Studio for sharing.
```

```
google_key=hidden_from_view
```

2. Retrieve town coordinates.

Google api will be used to retrive the coordinates (latitude and longitude of each town centers. The town coordinates will be used in retrieval of Foursquare API location data.

In [8]:

```
df_avg['Latitude'] = 0.0
df_avg['Longitude'] = 0.0

for idx,town in df_avg['Town'].iteritems():
    address = town + " United States" ; # I prefer to use MRT stations as more important ce
    url = 'https://maps.googleapis.com/maps/api/geocode/json?address={}&key={}'.format(addr
    print(url)
    lat = requests.get(url).json()["results"][0]["geometry"]["location"]['lat']
    lng = requests.get(url).json()["results"][0]["geometry"]["location"]['lng']
    df_avg.loc[idx,'Latitude'] = lat
    df_avg.loc[idx,'Longitude'] = lng
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-8-454c1173519d> in <module>
      4 for idx,town in df_avg['Town'].iteritems():
      5     address = town + " United States" ; # I prefer to use MRT statio
ns as more important central location of each town
----> 6     url = 'https://maps.googleapis.com/maps/api/geocode/json?address
={}&key={}'.format(address,google_key)
      7     print(url)
      8     lat = requests.get(url).json()["results"][0]["geometry"]["locati
on"]['lat']

NameError: name 'google_key' is not defined
```

In [13]:

```
#reading from saved file to avoid call to google api multiple times
df_avg = pd.read_csv('United_States_average.csv')
df_avg.head()
```

Out[13]:

	Unnamed: 0	Town	median_rent	Latitude	Longitude
0	0	Adrian, MI	145800.0	41.897547	-84.037166
1	1	Akron, OH	141600.0	41.081445	-81.519005
2	2	Albany, OR	258600.0	44.636511	-123.105928
3	3	Anchorage, AK	311700.0	61.218056	-149.900278
4	4	Astoria, OR	287700.0	46.187884	-123.831253

In [ ]:

```

```

In [14]:

```
df_avg.set_index("Town")
```

Out[14]:

Town	Unnamed: 0	median_rent	Latitude	Longitude
Adrian, MI	0	145800.0	41.897547	-84.037166
Akron, OH	1	141600.0	41.081445	-81.519005
Albany, OR	2	258600.0	44.636511	-123.105928
Anchorage, AK	3	311700.0	61.218056	-149.900278
Astoria, OR	4	287700.0	46.187884	-123.831253
Barnstable Town, MA	5	396700.0	41.700321	-70.300202
Bartlesville, OK	6	113200.0	36.747311	-95.980818
Bay City, MI	7	82200.0	43.594468	-83.888865
Beaver Dam, WI	8	156200.0	43.457769	-88.837329

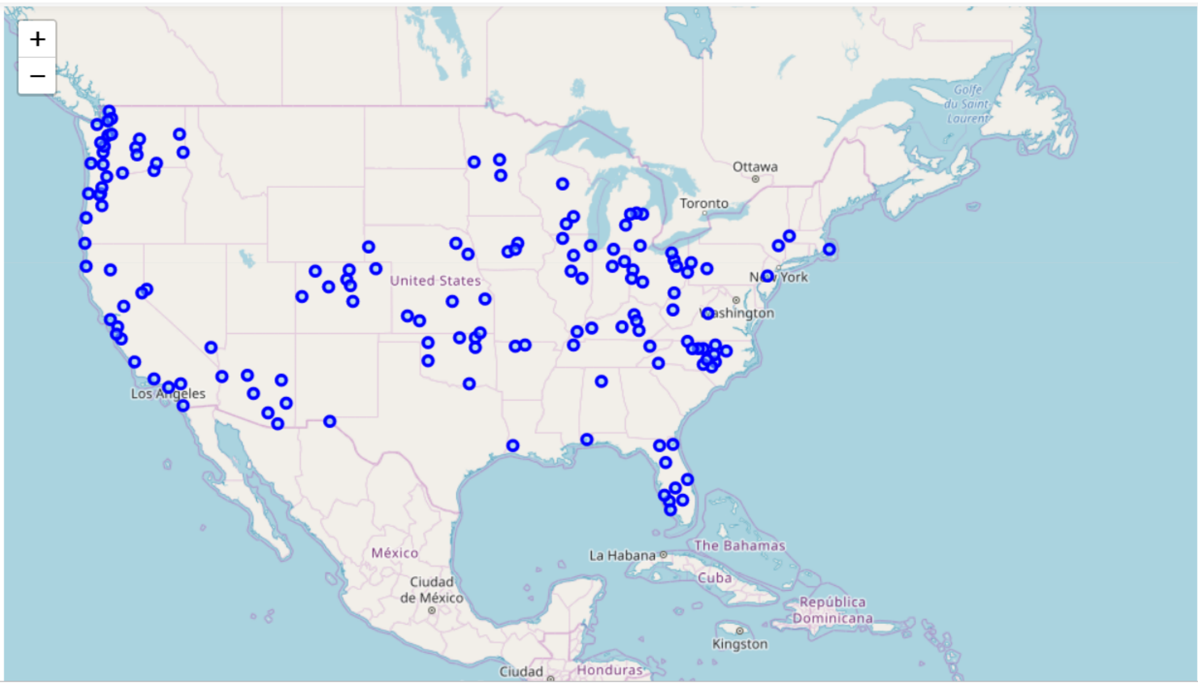
Generate Singapore basemap.

In [15]:

```
geo = Nominatim()
address = 'United States'
location = geo.geocode(address)
latitude = location.latitude
longitude = location.longitude
print('The geograpical coordinate of United States {}, {}.'.format(latitude, longitude))

# create map of USA using latitude and longitude values
map_USA = folium.Map(location=[latitude, longitude],tiles="OpenStreetMap", zoom_start=4)

# add markers to map
for lat, lng, town in zip(
    df_avg['Latitude'],
    df_avg['Longitude'],
    df_avg['Town']):
    label = town
    label = folium.Popup(label, parse_html=True)
    folium.CircleMarker(
        [lat, lng],
        radius=4,
        popup=label,
        color='blue',
        fill=True,
        fill_color='#87cefa',
        fill_opacity=0.5,
        parse_html=False).add_to(map_USA)
map_USA
```

```
c:\users\msvdp\appdata\local\programs\python\python37\lib\site-packages\ipyk
ernel_launcher.py:1: DeprecationWarning: Using Nominatim with the default "g
eopy/1.20.0" `user_agent` is strongly discouraged, as it violates Nominati
m's ToS https://operations.osmfoundation.org/policies/nominatim/ (https://op
erations.osmfoundation.org/policies/nominatim/) and may possibly cause 403 a
nd 429 HTTP errors. Please specify a custom `user_agent` with `Nominatim(use
r_agent="my-application")` or by overriding the default `user_agent`: `geop
y.geocoders.options.default_user_agent = "my-application"`. In geopy 2.0 thi
s will become an exception.
  """Entry point for launching an IPython kernel.

The geograpical coordinate of United States 39.7837304, -100.4458825.
```

Out[15]:



In [18]:

```
fileName = "United_States_average.csv"
linkName = "United_States Average Rental Prices"
create_download_link(df_avg,linkName,fileName)
```

Out[18]:

United_States Average Rental Prices
(data:text/csv;base64,LFVubmFtZWQ6IDAsVG93bixtZWRpYW5fcmVudCxMYXRpdHVkZSxMb2

In [17]:

```
from IPython.display import HTML
import base64

# Extra Helper scripts to generate download links for saved dataframes in csv format.
def create_download_link( df, title = "Download CSV file", filename = "data.csv"):
    csv = df.to_csv()
    b64 = base64.b64encode(csv.encode())
    payload = b64.decode()
    html = '<a download="{filename}" href="data:text/csv;base64,{payload}" target="_blank">
    html = html.format(payload=payload,title=title,filename=filename)
    return HTML(html)
```

V. Segmenting and Clustering Cities or Towns in USA

Retrieving FourSquare Places of interest.

Using the Foursquare API, the **explore** API function was be used to get the most common venue categories in each neighborhood, and then used this feature to group the neighborhoods into clusters. The *k*-means clustering algorithm was used for the analysis. Fnally, the Folium library is used to visualize the recommended neighborhoods and their emerging clusters.

In the ipynb notebook, the function **getNearbyVenues** extracts the following information for the dataframe it generates:

- Venue ID
- Venue Name
- Coordinates : Latitude and Longitude
- Category Name

The function **getVenuesByCategory** performs the following:

1. **category** based venue search to simulate user venue searches based on certain places of interest. This search extracts the following information:

    - Venue ID
    - Venue Name
    - Coordinates : Latitude and Longitude

   - Category Name
2. For each retrieved **venueID**, retrive the venues category rating.

In [12]:

```
# The code was removed by Watson Studio for sharing.
```

```
Hidden Foursqure API Keyset
```

In [13]:

```
# The code was removed by Watson Studio for sharing.
```

```
CLIENT_ID     = hidden
CLIENT_SECRET = hidden
VERSION       = 20190102
LIMIT         = 80
```

1. Exploring Neighbourhood in USA

Using the following foursquare api query url, search venues on all boroughs in selected USA Cities.

> `https://api.foursquare.com/v2/venues/` **search** `?`
> `client_id=` **CLIENT_ID** `&client_secret=` **CLIENT_SECRET** `&ll=` **LATITUDE** `,` **LONGITUDE** `&v=` **VI**

Retrieving data from FourSquare API is not so straight forward. It returns a json list top venues to visit to city. The scores however, is retrieved on a separate query to the FourSquare Venue API and is limited to 50 queries per day when using a free FourSquare subscription.
The following functions generates the query urls and processes the returned json data into dataframe.

The function **getNearbyVenues** extracts the following information for the dataframe it generates:

- Venue ID
- Venue Name
- Coordinates : Latitude and Longitude
- Category Name

The function **getVenuesByCategory** performs the following:

1. **category** based venue search to simulate user venue searches based on certain places of interest. This search extracts the following information:
   - Venue ID
   - Venue Name
   - Coordinates : Latitude and Longitude
   - Category Name
2. For each retrieved **venueID**, retrive the venues category rating.

The generated data frame in the second function contains the following column:

Column Name	Description
Town	Town Name
Town Latitude	Towns MRT station Latitude
Town Longitude	Town MRT station Latitude
VenueID	FourSquare Venue ID
VenueName	Venue Name
score	FourSquare Venue user rating
category	Category group name
catID	Category ID
latitude	Venue Location - latitude
longitude	Venue Location - longitude

In [19]:

```
# Foursquare Credentials
CLIENT_ID = '' # your Foursquare ID
CLIENT_SECRET = '' # your ou Secretr key
VERSION = '20180605' # Foursquare API version
LIMIT = 80

print('Your credentials:')
print('CLIENT_ID: ' + CLIENT_ID)
print('CLIENT_SECRET:' + CLIENT_SECRET)
```

```
Your credentials:
CLIENT_ID: 2FPN1CICVCJC3FTJJVN1VJG04RJ0IWGB3HGYL3HTAENMFUK1
CLIENT_SECRET:SXYV152RCW3DLF3RLTOMMSRU0LIZWJSFFRCFOQ2PXIWXK12X
```

In [20]:

```
import time
# --------------------------------------------
# The following function retrieves the venues given the names and coordinates and stores it
FOURSQUARE_EXPLORE_URL = 'https://api.foursquare.com/v2/venues/explore?'
FOURSQUARE_SEARCH_URL = 'https://api.foursquare.com/v2/venues/search?'

def getNearbyVenues(names, latitudes, longitudes, radius=500):
    global CLIENT_ID
    global CLIENT_SECRET
    global FOURSQUARE_EXPLORE_URL
    global FOURSQUARE_SEARCH_URL
    global VERSION
    global LIMIT
    venues_list=[]
    for name, lat, lng in zip(names, latitudes, longitudes):
        print('getNearbyVenues',names)
        # create the API request URL
        url = '{}&client_id={}&client_secret={}&v={}&ll={},{}&radius={}&limit={}'.format(
            FOURSQUARE_EXPLORE_URL,CLIENT_ID,CLIENT_SECRET,VERSION,
            lat,lng,radius,LIMIT)

        # make the GET request
        results = requests.get(url).json()["response"]['groups'][0]['items']

        # return only relevant information for each nearby venue
        venues_list.append([(
            name,lat,lng,
            v['venue']['id'],v['venue']['name'],
            v['venue']['location']['lat'],v['venue']['location']['lng'],
            v['venue']['categories'][0]['name']) for v in results])
        time.sleep(2)

    nearby_venues = pd.DataFrame([item for venue_list in venues_list for item in venue_list
    nearby_venues.columns = ['Town','Town Latitude','Town Longitude','Venue','Venue Latituc

    return(nearby_venues)
```

In [21]:

```
FOURSQUARE_SEARCH_URL = 'https://api.foursquare.com/v2/venues/search?'
# SEARCH VENUES BY CATEGORY

# Dataframe : venue_id_recover
# - store venue id to recover failed venues id score retrieval later if foursquare limit is
venue_id_rcols = ['VenueID']
venue_id_recover = pd.DataFrame(columns=venue_id_rcols)

def getVenuesByCategory(names, latitudes, longitudes, categoryID, radius=500):
    global CLIENT_ID
    global CLIENT_SECRET
    global FOURSQUARE_EXPLORE_URL
    global FOURSQUARE_SEARCH_URL
    global VERSION
    global LIMIT
    venue_columns = ['Town','Town Latitude','Town Longitude','VenueID','VenueName','score',
    venue_DF = pd.DataFrame(columns=venue_columns)
    print("[#Start getVenuesByCategory]")
    for name, lat, lng in zip(names, latitudes, longitudes):
        #print('getVenuesByCategory',categoryID,name) ; # DEBUG: be quiet
        # create the API request URL
        url = '{}client_id={}&client_secret={}&v={}&ll={},{}&radius={}&limit={}&categoryId=
            FOURSQUARE_SEARCH_URL,CLIENT_ID,CLIENT_SECRET,VERSION,lat,lng,radius,LIMIT,cate
        # make the GET request
        results = requests.get(url).json()
        # Populate dataframe with the category venue results
        # Extracting JSON  data values

        for jsonSub in results['response']['venues']:
            #print(jsonSub)
            # JSON Results may not be in expected format or incomplete data, in that case,
            ven_id = 0
            try:
                # If there are any issue with a restaurant, retry or ignore and continue
                # Get location details
                ven_id   = jsonSub['id']
                ven_cat  = jsonSub['categories'][0]['pluralName']
                ven_CID  = jsonSub['categories'][0]['id']
                ven_name = jsonSub['name']
                ven_lat  = jsonSub['location']['lat']
                ven_lng  = jsonSub['location']['lng']
                venue_DF = venue_DF.append({
                    'Town'          : name,
                    'Town Latitude' : lat,
                    'Town Longitude': lng,
                    'VenueID'       : ven_id,
                    'VenueName'     : ven_name,
                    'score'         : 'na',
                    'category'      : ven_cat,
                    'catID'         : ven_CID,
                    'latitude'      : ven_lat,
                    'longitude'     : ven_lng}, ignore_index=True)
            except:
                continue
    # END OF LOOP, return.
    print("\n[#Done getVenuesByCategory]")
    return(venue_DF)
```

In [34]:

```
FOURSQUARE_SEARCH_URL = 'https://api.foursquare.com/v2/venues/search?'
# SEARCH VENUES BY CATEGORY

# Dataframe : venue_id_recover
# - store venue id to recover failed venues id score retrieval later if foursquare limit is
venue_id_rcols = ['VenueID','Score']
venue_id_recover = pd.DataFrame(columns=venue_id_rcols)

def getVenuesIDScore(venueID):
    global CLIENT_ID
    global CLIENT_SECRET
    global FOURSQUARE_EXPLORE_URL
    global FOURSQUARE_SEARCH_URL
    global VERSION
    global LIMIT
    global venue_id_recover
    print("[#getVenuesIDScore]")
    venID_URL = 'https://api.foursquare.com/v2/venues/{}?client_id={}&client_secret={}&v={}
    venID_score = 0.00
    # Process results
    try:
        venID_result = requests.get(venID_URL).json()
        venID_score  = venID_result['response']['venue']['rating']
    except:
        venue_id_recover = venue_id_recover.append({'VenueID' : venueID, 'Score' : 0.0},igr
        return ["error",0.0]
    return ["success",venID_score]
```

In [25]:

```
df_avg.dtypes
```

Out[25]:

```
Town            object
median_rent    float64
Latitude       float64
Longitude      float64
dtype: object
```

In [22]:

```
venue_columns = ['Town','Town Latitude','Town Longitude','VenueID','VenueName','score','cat
df_venue = pd.DataFrame(columns=venue_columns)
```

Search Venues with recommendations on : Food Venues (Restaurants,Fastfoods, etc.)

To demonstrate user selection of places of interest, We will use this Food Venues category in our further analysis.

- This Foursquare search is expected to collect venues in the following category:
    - category
    - Food Courts
    - Coffee Shops

- Restaurants
- Cafés
- Other food venues

In [23]:

```
# Food Venues : Restaurants, Fastfoods, Etc
categoryID = "4d4b7105d754a06374d81259"
town_names = df_avg['Town']
lat_list   = df_avg['Latitude']
lng_list   = df_avg['Longitude']
df_food_venues = getVenuesByCategory(names=town_names,latitudes=lat_list,longitudes=lng_lis
df_food_venues
```

```
[#Start getVenuesByCategory]

[#Done getVenuesByCategory]
```

- Save collected USA food venues by town into csv for future use.

In [24]:

```
# Save collected USA food venues by town into csv for future use.
fileName = "food_venues.Category.csv"
linkName = "IBM Storage Link:food_venues.Category.csv"
create_download_link(df_food_venues,linkName,fileName)
```

Out[24]:

IBM Storage Link:food_venues.Category.csv
(data:text/csv;base64,LFRvd24sVG93biBMYXRpdHVkZSxUb3duIExvbmdpdHVkZSxWZW51ZUl

Search Venues with recommendations on : Outdoors and Recreation

Note:

- 2nd Test: Retrieve venues for Outdoors and Recreation.
- This section can be ran separately due to maximum limit encountered when using Foursquare free API version. I have saved simmilar results in github to run the same analyis.

In [25]:

```
# Disable for this run demo.
if (1):
    # Outdoors & Recreation,
    categoryID = "4d4b7105d754a06377d81259"
    town_names = df_avg['Town']
    lat_list   = df_avg['Latitude']
    lng_list   = df_avg['Longitude']
    df_outdoor_venues_by_town = getVenuesByCategory(names=town_names,latitudes=lat_list,lor
    fileName = "outdoorAndRecration.Category.csv"
    linkName = "IBM Storage Link:outdoorAndRecration.Category.csv"
    create_download_link(df_outdoor_venues_by_town,linkName,fileName)
```

```
[#Start getVenuesByCategory]

[#Done getVenuesByCategory]
```

In [26]:

```
    fileName = "outdoorAndRecration.Category.csv"
    linkName = "IBM Storage Link:outdoorAndRecration.Category.csv"
    create_download_link(df_outdoor_venues_by_town,linkName,fileName)
```

Out[26]:

IBM Storage Link:outdoorAndRecration.Category.csv
(data:text/csv;base64,LFRvd24sVG93biBMYXRpdHVkZSxUb3duIExvbmdpdHVkZSxWZW51ZUl

In [27]:

```
df_food_venues.head()
# The code was removed by Watson Studio for sharing.
```

Out[27]:

	Town	Town Latitude	Town Longitude	VenueID	VenueName	score	category
0	Adrian, MI	41.897547	-84.037166	4d2a426febacb1f7ff040250	Pizza Bucket	na	Pizza Places
1	Adrian, MI	41.897547	-84.037166	4d9e29ec71ac6a31ed9a4c06	The Grasshopper El Chapulin - Adrian	na	Mexican Restaurants
2	Adrian, MI	41.897547	-84.037166	5b9c2550270ee70039c5657f	Downtown Dempsey's	na	Pizza Places
3	Adrian, MI	41.897547	-84.037166	4fd257c5e4b069289209c46e	governor croswell tea room restaurant	na	Tea Rooms
4	Adrian, MI	41.897547	-84.037166	5af8842232b61d002cf554c3	Farver's	na	Bistros

In this section, We use the FourSquare API to retrieve venue scores of locations. Note that there is max

query limit of 50 in FourSquare API for free subscription. So use or query carefully.

In [37]:

```
score_is_NAN = len(df_food_venues[df_food_venues['score'].isna()].index.tolist())
print("Current score=NaN count=",score_is_NAN)
for idx in df_food_venues[df_food_venues['score'].isna()].index.tolist():
    venueID = df_food_venues.loc[idx,'VenueID']
    print(venueID)
    status,score = getVenuesIDScore(venueID)
    if status == "success":
        df_food_venues.loc[idx,'score'] = score
score_is_NAN = len(df_food_venues[df_food_venues['score'].isna()].index.tolist())
print("PostRun score=NaN count=",score_is_NAN)
print('Done',end='')
```

```
Current score=NaN count= 0
PostRun score=NaN count= 0
Done
```

In [29]:

```

fileName = "food_venues_with_Score.Category.csv"
linkName = "IBM Storage Link:food_venues_score.Category.csv"
create_download_link(df_food_venues,linkName,fileName)
```

Out[29]:

IBM Storage Link:food_venues_score.Category.csv
(data:text/csv;base64,LFRvd24sVG93biBMYXRpdHVkZSxUb3duIExvbmdpdHVkZSxWZW51ZUl

- Note: Re-run continuation, reload saved csv file. # Reloading previously saved runs to avoid re-running FourSquare API.

In [26]:

```
# The code was removed by Watson Studio for sharing.
```

- Combine venues collection into one dataframe : df_venue

In [38]:

```
# If all categories are called
if (1):
    df_venue = pd.concat([df_food_venues,df_outdoor_venues_by_town], ignore_index=True)
#else
df_venue = df_food_venues
df_venue.shape
```

Out[38]:

```
(4461, 10)
```

Data cleanup uneeded entries

- Eliminate possible venue duplicates.
- Improve the quality of our venue selection by removing venues with no ratings or 0.0

In [39]:

```
# Eliminate possible venue duplicates.
df_venue = df_venue[venue_columns]

# Drop rows with missing elements
df_venue = df_venue.dropna(axis='columns')
```

In [40]:

```
df_venue.shape
```

Out[40]:

(4461, 10)

In [41]:

```
df_venue.head()
```

Out[41]:

	Town	Town Latitude	Town Longitude	VenueID	VenueName	score	category
0	Adrian, MI	41.897547	-84.037166	4d2a426febacb1f7ff040250	Pizza Bucket	na	Pizza Places
1	Adrian, MI	41.897547	-84.037166	4d9e29ec71ac6a31ed9a4c06	The Grasshopper El Chapulin - Adrian	na	Mexican Restaurants
2	Adrian, MI	41.897547	-84.037166	5b9c2550270ee70039c5657f	Downtown Dempsey's	na	Pizza Places
3	Adrian, MI	41.897547	-84.037166	4fd257c5e4b069289209c46e	governor croswell tea room restaurant	na	Tea Rooms
4	Adrian, MI	41.897547	-84.037166	5af8842232b61d002cf554c3	Farver's	na	Bistros

In [295]:

```
# Save town venues collection.
# This list is already intersting data for display in different webpages.
fileName = "recommended.USA_town_venues.csv"
linkName = "IBM Storage Link:recommended_USA_town_venues.csv"
create_download_link(df_food_venues,linkName,fileName)
```

Out[295]:

IBM Storage Link:recommended_USA_town_venues.csv
(data:text/csv;base64,LFRvd24sVG93biBMYXRpdHVkZSxUb3duIExvbmdpdHVkZSxWZW51ZUl

Check venue count per town.

In [70]:

```
df_venue.groupby('Town').count()
```

Out[70]:

	Town Latitude	Town Longitude	VenueID	VenueName	score	category	catID	latitude	longitude
Town									
Adrian, MI	17	17	17	17	17	17	17	17	17
Akron, OH	40	40	40	40	40	40	40	40	40
Albany, OR	21	21	21	21	21	21	21	21	21
Anchorage, AK	49	49	49	49	49	49	49	49	49
Astoria, OR	45	45	45	45	45	45	45	45	45
Barnstable Town, MA	7	7	7	7	7	7	7	7	7
Bartlesville, OK	27	27	27	27	27	27	27	27	27
Bay City, MI	32	32	32	32	32	32	32	32	32

In [44]:

```
# Verify the dtypes
df_venue.dtypes
```

Out[44]:

```
Town               object
Town Latitude     float64
Town Longitude    float64
VenueID            object
VenueName          object
score              object
category           object
catID              object
latitude          float64
longitude         float64
dtype: object
```

How many unique categories can be curated from all the returned venues?

In [45]:

```
# Count number of categories that can be curated.
print('There are {} uniques categories.'.format(len(df_venue['category'].unique())))
```

```
There are 159 uniques categories.
```

What are the top 20 most common venue types?

In [46]:

```
# Check top 10 most frequently occuring venue type
df_venue.dropna(subset=['score'])
df_venue.groupby('category')['VenueName'].count().sort_values(ascending=False)[:20]
```

Out[46]:

```
category
Coffee Shops              433
American Restaurants      375
Mexican Restaurants       267
Pizza Places              256
Sandwich Places           193
Cafés                     186
Restaurants               156
Bakeries                  145
Italian Restaurants       133
Ice Cream Shops           115
Fast Food Restaurants     112
Burger Joints              89
Food                       89
Food Trucks                80
Diners                     72
Bars                       72
Seafood Restaurants        71
Breakfast Spots            70
Delis / Bodegas            70
Chinese Restaurants        69
Name: VenueName, dtype: int64
```

What are the top 20 venues given with highest score rating?

In [74]:

```
# Top 10 venues with highest given score rating

df_venue.groupby(['Town','category']).count().sort_values(by='score',ascending=False)[:20]
```

Out[74]:

Town	category	Town Latitude	Town Longitude	VenueID	VenueName	score	catID	latitude	lo
Portland, OR	Coffee Shops	16	16	16	16	16	16	16	
Seattle, WA	Coffee Shops	15	15	15	15	15	15	15	
Salinas, CA	Mexican Restaurants	10	10	10	10	10	10	10	
El Paso, TX	Mexican Restaurants	10	10	10	10	10	10	10	
Las Vegas, NV	American Restaurants	9	9	9	9	9	9	9	
Edwards, CO	American Restaurants	9	9	9	9	9	9	9	
Fort Collins, CO	Food Trucks	9	9	9	9	9	9	9	
Santa Cruz, CA	Coffee Shops	9	9	9	9	9	9	9	
Naples, FL	Italian Restaurants	9	9	9	9	9	9	9	
Redding, CA	Coffee Shops	9	9	9	9	9	9	9	
Tucson, AZ	Mexican Restaurants	9	9	9	9	9	9	9	
Dodge City, KS	Mexican Restaurants	8	8	8	8	8	8	8	
Tulsa, OK	American Restaurants	8	8	8	8	8	8	8	
Des Moines, IA	Coffee Shops	8	8	8	8	8	8	8	
Riverside, CA	Mexican Restaurants	8	8	8	8	8	8	8	
St. Cloud, MN	Pizza Places	8	8	8	8	8	8	8	
Madison, WI	Coffee Shops	8	8	8	8	8	8	8	
Akron, OH	Cafés	8	8	8	8	8	8	8	
Sacramento, CA	Coffee Shops	8	8	8	8	8	8	8	
Cleveland, OH	Coffee Shops	8	8	8	8	8	8	8	

Analyze Each USA 's nearby recommended venues

In [75]:

```
# one hot encoding
sg_onehot = pd.get_dummies(df_venue[['category']], prefix="", prefix_sep="")

# add Town column back to dataframe
sg_onehot['Town'] = df_venue['Town']

# move neighborhood column to the first column
fixed_columns = [sg_onehot.columns[-1]] + list(sg_onehot.columns[:-1])
sg_onehot = sg_onehot[fixed_columns]

# Check returned one hot encoding data:
print('One hot encoding returned "{}" rows.'.format(sg_onehot.shape[0]))

# Regroup rows by town and mean of frequency occurrence per category.
sg_grouped = sg_onehot.groupby('Town').mean().reset_index()

print('One hot encoding re-group returned "{}" rows.'.format(sg_grouped.shape[0]))
sg_grouped.head()
```

```
One hot encoding returned "4461" rows.
One hot encoding re-group returned "143" rows.
```

Out[75]:

	Town	Afghan Restaurants	African Restaurants	American Restaurants	Antique Shops	Arcades	Arepa Restaurants	Art Galleries
0	Adrian, MI	0.0	0.0	0.176471	0.0	0.0	0.0	0.0
1	Akron, OH	0.0	0.0	0.125000	0.0	0.0	0.0	0.0
2	Albany, OR	0.0	0.0	0.047619	0.0	0.0	0.0	0.0
3	Anchorage, AK	0.0	0.0	0.040816	0.0	0.0	0.0	0.0
4	Astoria, OR	0.0	0.0	0.066667	0.0	0.0	0.0	0.0

Type *Markdown* and LaTeX: α^2

Analyze USA's most visited venues

In [76]:

```
num_top_venues = 10
for town in sg_grouped['Town']:
    print("# Town=< "+town+" >")
    temp = sg_grouped[sg_grouped['Town'] == town].T.reset_index()
    temp.columns = ['venue','freq']
    temp = temp.iloc[1:]
    temp['freq'] = temp['freq'].astype(float)
    temp = temp.round({'freq': 2})
    print(temp.sort_values('freq', ascending=False).reset_index(drop=True).head(num_top_ver
    print('\n')
```

```
# Town=< Adrian, MI >
                  venue  freq
0          Pizza Places  0.24
1  American Restaurants  0.18
2   Italian Restaurants  0.12
3             Bakeries  0.06
4             Tea Rooms  0.06
5              Bistros  0.06
6   Chinese Restaurants  0.06
7         Burger Joints  0.06
8     Cuban Restaurants  0.06
9       Ice Cream Shops  0.06


# Town=< Akron, OH >
                       venue  freq
0                      Cafés  0.20
1       American Restaurants  0.12
2                Food Trucks  0.10
```

First, let's write a function to sort the venues in descending order.

In [77]:

```
def return_most_common_venues(row, num_top_venues):
    row_categories = row.iloc[1:]
    row_categories_sorted = row_categories.sort_values(ascending=False)
    return row_categories_sorted.index.values[0:num_top_venues]
```

In [78]:

```
num_top_venues = 10

indicators = ['st', 'nd', 'rd']

# create columns according to number of top venues
columns = ['Town']
for ind in np.arange(num_top_venues):
    try:
        columns.append('{}{} Most Common Venue'.format(ind+1, indicators[ind]))
    except:
        columns.append('{}th Most Common Venue'.format(ind+1))

# create a new dataframe
town_venues_sorted = pd.DataFrame(columns=columns)
town_venues_sorted['Town'] = sg_grouped['Town']

for ind in np.arange(sg_grouped.shape[0]):
    town_venues_sorted.iloc[ind, 1:] = return_most_common_venues(sg_grouped.iloc[ind, :], r

print(town_venues_sorted.shape)
town_venues_sorted.head()
```

(143, 11)

Out[78]:

	Town	1st Most Common Venue	2nd Most Common Venue	3rd Most Common Venue	4th Most Common Venue	5th Most Common Venue	6th Most Common Venue	7th Mo Comm Ven
0	Adrian, MI	Pizza Places	American Restaurants	Italian Restaurants	Burger Joints	Cuban Restaurants	Chinese Restaurants	Baker
1	Akron, OH	Cafés	American Restaurants	Coffee Shops	Food Trucks	Sandwich Places	Delis / Bodegas	Restaura
2	Albany, OR	Cafés	Sandwich Places	Thai Restaurants	Bakeries	Italian Restaurants	Mexican Restaurants	Fc
3	Anchorage, AK	Coffee Shops	Cafés	Seafood Restaurants	Pizza Places	Sandwich Places	Restaurants	Americ Restaura
4	Astoria, OR	Seafood Restaurants	Coffee Shops	Pizza Places	Food	American Restaurants	Italian Restaurants	T Restaura

Clustering Neighborhoods

Run *k*-means to cluster the Towns into 5 clusters.

In [79]:

```
# set number of clusters
kclusters = 5
sg_grouped_clustering = sg_grouped.drop('Town', 1)
# run k-means clustering
kmeans = KMeans(n_clusters=kclusters, random_state=1).fit(sg_grouped_clustering)

# check cluster labels generated for each row in the dataframe
print(kmeans.labels_[0:10])
print(len(kmeans.labels_))
```

```
[1 0 1 1 1 0 0 1 1 1]
143
```

In [206]:

```
town_venues_sorted.head()
```

Out[206]:

	Town	1st Most Common Venue	2nd Most Common Venue	3rd Most Common Venue	4th Most Common Venue	5th Most Common Venue	6th Most Common Venue	7th Mo Comm Ven
0	Adrian, MI	Pizza Places	American Restaurants	Italian Restaurants	Burger Joints	Cuban Restaurants	Chinese Restaurants	Baker
1	Akron, OH	Cafés	American Restaurants	Coffee Shops	Food Trucks	Sandwich Places	Delis / Bodegas	Restaura
2	Albany, OR	Cafés	Sandwich Places	Thai Restaurants	Bakeries	Italian Restaurants	Mexican Restaurants	Fo
3	Anchorage, AK	Coffee Shops	Cafés	Seafood Restaurants	Pizza Places	Sandwich Places	Restaurants	Americ Restaura
4	Astoria, OR	Seafood Restaurants	Coffee Shops	Pizza Places	Food	American Restaurants	Italian Restaurants	T Restaura

In [80]:

```
#town_venues_sorted = town_venues_sorted.set_index("Town")
sg_merged = df_avg.set_index("Town")
# add clustering labels
sg_merged['Cluster Labels'] = pd.Series(kmeans.labels_)
# merge sg_grouped with df_avg to add latitude/longitude for each neighborhood
sg_merged = sg_merged.join(town_venues_sorted)
sg_merged
```

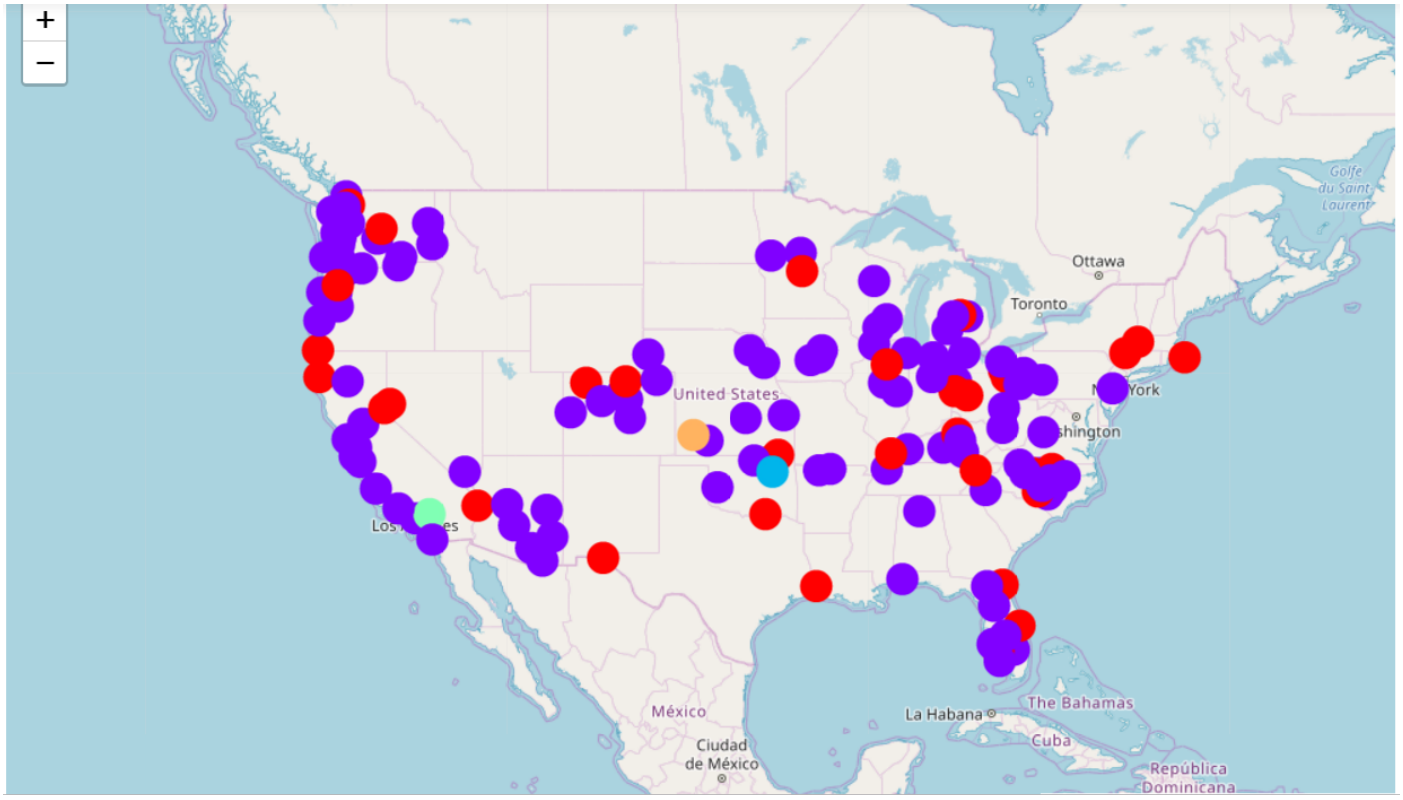
Out[80]:

	Unnamed: 0	median_rent	Latitude	Longitude	Cluster Labels	Town	1st Most Common Venue	2nd Most Common Venue	3rd Mos Commo Venu
Town									
Adrian, MI	0	145800.0	41.897547	-84.037166	NaN	NaN	NaN	NaN	Na
Akron, OH	1	141600.0	41.081445	-81.519005	NaN	NaN	NaN	NaN	Na
Albany, OR	2	258600.0	44.636511	-123.105928	NaN	NaN	NaN	NaN	Na
Anchorage, AK	3	311700.0	61.218056	-149.900278	NaN	NaN	NaN	NaN	Na
Astoria, OR	4	287700.0	46.187884	-123.831253	NaN	NaN	NaN	NaN	Na
Barnstable Town, MA	5	396700.0	41.700321	-70.300202	NaN	NaN	NaN	NaN	Na

- Save csv copy of merged data

In [81]:

```
# create map
map_clusters = folium.Map(location=[latitude, longitude], tiles="Openstreetmap", zoom_start

# set color scheme for the clusters
x = np.arange(kclusters)
ys = [i+x+(i*x)**2 for i in range(kclusters)]
colors_array = cm.rainbow(np.linspace(0, 1, len(ys)))
rainbow = [colors.rgb2hex(i) for i in colors_array]

# add markers to the map
markers_colors = []
for lat, lon, poi, cluster in zip(sg_merged['Latitude'], sg_merged['Longitude'], sg_merged.
    label = folium.Popup(str(poi) + ' Cluster ' + str(cluster), parse_html=True)
    folium.CircleMarker(
        [lat, lon],
        radius=10,
        popup=label,
        color=rainbow[cluster-1],
        fill=True,
        fill_color=rainbow[cluster-1],
        fill_opacity=1).add_to(map_clusters)

map_clusters
```



VI. Discussion and Conclusion

On this notebook, Analysis of best venue recommendations based on Food venue category has been presented. Recommendations based on other user searches like available outdoor and recreation areas are also available.The information extracted in this notebook, will be a good supplement to web based recommendations for visitors to find out nearby venues of interest and be a useful aid in deciding a place to stay or where to go during their visits.

Using Foursquare API, we have collected a good amount of venue recommnedations. Sourcing from the venue recommendations from FourSquare has its limitation, The list of venues is not exhaustive list of all the available venues is the area. Furthermore, not all the venues found in the the area has a stored ratings. For this reason, the number of analyzed venues are only about 50% of all the available venues initially collected. The results therefore may significantly change, when more information are collected on those with missing data.

The generated clusters from our results shows that there are very good and interesting places located in areas where the median rents are cheaper. This kind of results may be very interesting for travelers who are also on budget constraints. Our results also yielded some interesting findings. For instance, The initial assumption among websites providing recommendations is that the Central Area that have the highest median rent also have better food venues. Result shows that most popular food venue among residents and visitors are **Coffee Shops, American Restaurants, Mexican Restaurants**.

In [ ]:

```

```

Additional Visualizations

Supplementary doc. ¶

With the venue recommendations , it is good to have same additional visualizations of the data we have collected. Since we are dealing with categorical data, this information collected are best understood via visualization.

In [6]:

```
import numpy as np # library to handle data in a vectorized manner
import pandas as pd # library for data analsysis
import json # library to handle JSON files

# Matplotlib and associated plotting modules
import matplotlib as mpl
import matplotlib.cm as cm
import matplotlib.colors as colors
import matplotlib.pyplot as plt
# Use the inline backend to generate the plots within the browser
%matplotlib inline

import seaborn as sns
sns.set_style("darkgrid")

#If Need to re-download some files saved earlier//or scrape again.
import requests # library to handle requests
import lxml.html as lh
import bs4 as bs
import urllib.request

from IPython.display import HTML
import base64

print('Libraries imported.')
```

```
Libraries imported.
```

Restore Our Data

- Restore Singapore Food Venue Data Frame
- Restore Segmented Town Cluster Data

In [ ]:

```
```

In [5]:

```
Collecting lxml
  Downloading https://files.pythonhosted.org/packages/b8/5a/bb7f48b4e06ed600
55c8bf2ea7d8259dae40a55dc03104cd7b0782699b9a/lxml-4.3.3-cp37-cp37m-win_amd6
4.whl (https://files.pythonhosted.org/packages/b8/5a/bb7f48b4e06ed60055c8bf2
ea7d8259dae40a55dc03104cd7b0782699b9a/lxml-4.3.3-cp37-cp37m-win_amd64.whl)
 (3.6MB)
Installing collected packages: lxml
Successfully installed lxml-4.3.3
Note: you may need to restart the kernel to use updated packages.
```

In [18]:

```
# Restore dataframe collection :

df_food_venues      = pd.read_csv('food_venues_with_Score.Category.csv')

# Eliminate possible venue duplicates.
df_town_venues = df_food_venues[venue_columns]

# Drop rows with missing elements
df_town_venues = df_town_venues.dropna(axis='columns')

df_food_venues.head()
```

Out[18]:

	Town	Town Latitude	Town Longitude	VenueID	VenueName	score	category
0	Adrian, MI	41.897547	-84.037166	4d2a426febacb1f7ff040250	Pizza Bucket	na	Pizza Places
1	Adrian, MI	41.897547	-84.037166	4d9e29ec71ac6a31ed9a4c06	The Grasshopper El Chapulin - Adrian	na	Mexican Restaurants
2	Adrian, MI	41.897547	-84.037166	5b9c2550270ee70039c5657f	Downtown Dempsey's	na	Pizza Places
3	Adrian, MI	41.897547	-84.037166	4fd257c5e4b069289209c46e	governor croswell tea room restaurant	na	Tea Rooms
4	Adrian, MI	41.897547	-84.037166	5af8842232b61d002cf554c3	Farver's	na	Bistros

Check Venue Counts

In [19]:

```
venue_counts = df_town_venues.groupby('Town').count()
venue_counts
```

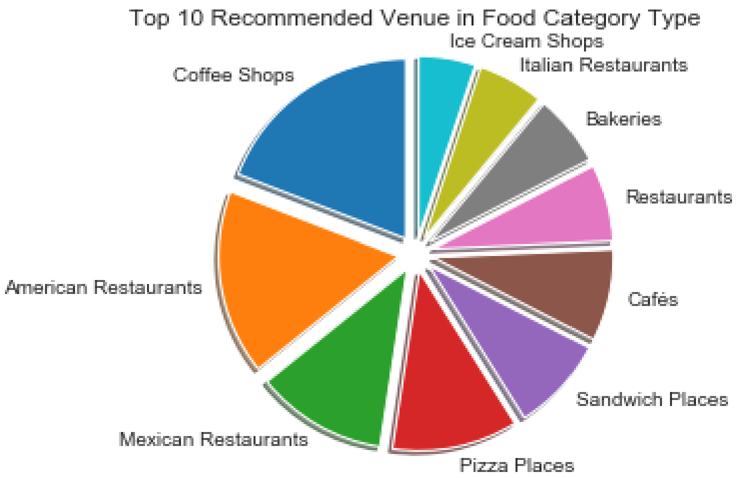
Out[19]:

	Town Latitude	Town Longitude	VenueID	VenueName	score	category	catID	latitude	lon
Town									
Adrian, MI	17	17	17	17	17	17	17	17	
Akron, OH	40	40	40	40	40	40	40	40	
Albany, OR	21	21	21	21	21	21	21	21	
Anchorage, AK	49	49	49	49	49	49	49	49	
Astoria, OR	45	45	45	45	45	45	45	45	
Barnstable Town, MA	7	7	7	7	7	7	7	7	
Bartlesville, OK	27	27	27	27	27	27	27	27	
Bay City, MI	32	32	32	32	32	32	32	32	
Beaver Dam, WI	17	17	17	17	17	17	17	17	
Bellingham, WA	50	50	50	50	50	50	50	50	
Bloomington, IL	34	34	34	34	34	34	34	34	
Boulder, CO	32	32	32	32	32	32	32	32	
Brainerd, MN	12	12	12	12	12	12	12	12	
Bremerton, WA	43	43	43	43	43	43	43	43	
Brevard, NC	36	36	36	36	36	36	36	36	
Brookings, OR	27	27	27	27	27	27	27	27	
Burlington, NC	18	18	18	18	18	18	18	18	
Campbellsville, KY	19	19	19	19	19	19	19	19	
Canton, OH	44	44	44	44	44	44	44	44	
Celina, OH	18	18	18	18	18	18	18	18	
Centralia, WA	25	25	25	25	25	25	25	25	
Champaign-Urbana, IL	48	48	48	48	48	48	48	48	
Charleston, WV	44	44	44	44	44	44	44	44	
Chicago, IL	50	50	50	50	50	50	50	50	
Cleveland, OH	50	50	50	50	50	50	50	50	
Clewiston, FL	9	9	9	9	9	9	9	9	
Coffeyville, KS	11	11	11	11	11	11	11	11	

Town	Town Latitude	Town Longitude	VenueID	VenueName	score	category	catID	latitude	lon
Colorado Springs, CO	50	50	50	50	50	50	50	50	
Coos Bay, OR	25	25	25	25	25	25	25	25	
Corvallis, OR	50	50	50	50	50	50	50	50	
...	...	...	...	...	...	...	...	...	
Safford, AZ	8	8	8	8	8	8	8	8	
Salem, OR	49	49	49	49	49	49	49	49	
Salina, KS	19	19	19	19	19	19	19	19	
Salinas, CA	41	41	41	41	41	41	41	41	
San Diego, CA	50	50	50	50	50	50	50	50	
San Francisco, CA	50	50	50	50	50	50	50	50	
San Jose, CA	50	50	50	50	50	50	50	50	
San Luis Obispo, CA	50	50	50	50	50	50	50	50	
Sanford, NC	19	19	19	19	19	19	19	19	
Santa Cruz, CA	50	50	50	50	50	50	50	50	
Scottsbluff, NE	10	10	10	10	10	10	10	10	
Seattle, WA	50	50	50	50	50	50	50	50	
Sebring, FL	20	20	20	20	20	20	20	20	
Shelton, WA	36	36	36	36	36	36	36	36	
Show Low, AZ	28	28	28	28	28	28	28	28	
Spokane, WA	50	50	50	50	50	50	50	50	
Springfield, OH	19	19	19	19	19	19	19	19	
St. Cloud, MN	46	46	46	46	46	46	46	46	
Staunton, VA	45	45	45	45	45	45	45	45	
Sterling, CO	18	18	18	18	18	18	18	18	
Trenton, NJ	38	38	38	38	38	38	38	38	
Truckee, CA	38	38	38	38	38	38	38	38	
Tucson, AZ	40	40	40	40	40	40	40	40	
Tulsa, OK	50	50	50	50	50	50	50	50	
Ventura, CA	50	50	50	50	50	50	50	50	
Weirton, OH	1	1	1	1	1	1	1	1	
Wenatchee, WA	38	38	38	38	38	38	38	38	
Winston-Salem, NC	49	49	49	49	49	49	49	49	
Woodward, OK	15	15	15	15	15	15	15	15	
Yakima, WA	43	43	43	43	43	43	43	43	

In [22]:

```
top_10_venue = df_town_venues.groupby('category')['VenueName'].count().sort_values(ascendir
labels = top_10_venue['category']
sizes  = top_10_venue['VenueName']
explode = [.1] * 10
# Plot
plt.title("Top 10 Recommended Venue in Food Category Type")
ax1 = plt.pie(sizes, explode=explode, labels=labels, shadow=True, startangle=90)
plt.axis('equal')
plt.show()
top_10_venue
```



Out[22]:

	category	VenueName
0	Coffee Shops	433
1	American Restaurants	375
2	Mexican Restaurants	267
3	Pizza Places	256
4	Sandwich Places	193
5	Cafés	186
6	Restaurants	156
7	Bakeries	145
8	Italian Restaurants	133
9	Ice Cream Shops	115

Top 20 venues given with highest score rating

In [24]:

```
# Top 20 venues with highest given score rating
top_20_scorer = df_town_venues.groupby(['Town','category'])['score'].count().sort_values(as
top_20_scorer
```

Out[24]:

	Town	category	score
0	Portland, OR	Coffee Shops	16
1	Seattle, WA	Coffee Shops	15
2	Salinas, CA	Mexican Restaurants	10
3	El Paso, TX	Mexican Restaurants	10
4	Tucson, AZ	Mexican Restaurants	9
5	Santa Cruz, CA	Coffee Shops	9
6	Redding, CA	Coffee Shops	9
7	Fort Collins, CO	Food Trucks	9
8	Las Vegas, NV	American Restaurants	9
9	Edwards, CO	American Restaurants	9
10	Naples, FL	Italian Restaurants	9
11	Canton, OH	American Restaurants	8
12	Port Angeles, WA	Coffee Shops	8
13	Chicago, IL	Coffee Shops	8
14	Cleveland, OH	Coffee Shops	8
15	St. Cloud, MN	Pizza Places	8
16	Madison, WI	Coffee Shops	8
17	Marshalltown, IA	Mexican Restaurants	8
18	Sacramento, CA	Coffee Shops	8
19	Des Moines, IA	Coffee Shops	8