



Next Nike Store in New Mexico

In the beginner's mind there are many possibilities, but in the expert's mind there are few.
—Shunryu Suzuki, Zen Mind, Beginner's Mind

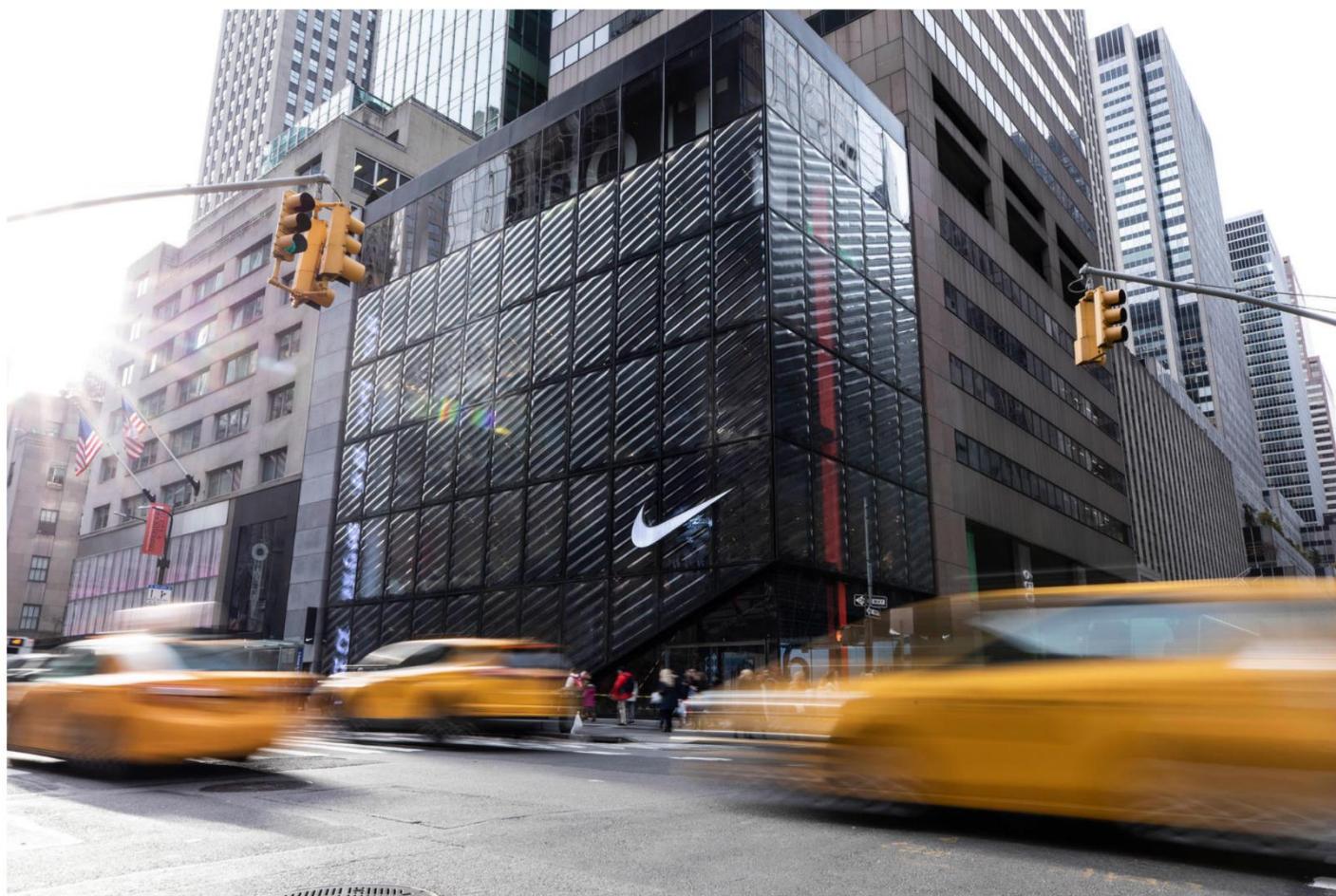
Table of contents

- [Introduction: Business Problem](#)
- [Data](#)
- [Methodology](#)
- [Finding the ideal locations](#)
- [Results](#)
- [Conclusion and Discussion](#)
- [Reference](#)

Introduction: Business Problem

Nike, Inc. is an American multinational corporation that is engaged in the design, development, manufacturing, and worldwide marketing and sales of footwear, apparel, equipment, accessories, and services. ([click the BEAUTIFUL Swoosh® above to explore more about Nike](#))

Even though online shopping is very popular nowadays, physical stores are still very important for a company. They enable consumers to feel products and corporate culture most directly, and also enhance brand awareness.



There are **273** Nike stores in **43** states in the United States, including **Nike City Flagship Shops**, **Nike Factory Stores**, **Nike Community Stores** and some other kinds of stores. Most of these shops are located in areas where commercial activities are intensive. Some states don't have any Nike store, like New Mexico, North Dakota and South Dakota.

Santa Fe is the capital of the U.S. state of New Mexico. Tourism is a major element of the Santa Fe economy, with visitors attracted year-round by the climate and related outdoor activities (such as skiing in years of adequate snowfall; hiking in other seasons) plus cultural activities of the city and the region. The city is also well known as a center for arts that reflect the multicultural character of the city. ([click here to explore Santa Fe](#))

Therefore, building a Nike store here maybe a good choice, because it can increase the diversity of Nike stores and expand brand influence. There was a Nike Factory store in Santa Fe, but it was closed, maybe because its location wasn't suitable. ([click here to explore the closed Nike Factory Store](#))

Supposing Nike decides to build a new store in Santa Fe. In this project, I will try to answer two questions:

Where should the new store be built?

What kind of store should the new store be, Nike City Flagship Shop or Nike Factory Store?

I collected the data of venues around the existing Nike store. **The data includes the number of nearby restaurants, stadiums, shops, bus stops and so on.** I used the data to build a **machine learning model(decision tree)**, and then used this model to select the appropriate locations in Santa Fe.

Data

Getting locations' latitudes and longitudes by Google map API

I collected the names of **22 Nike City Flagship Shops**, and **58 Nike Factory Stores** from Nike official website. ([click here to explore Nike stores in the USA](#))

And I also collected names of 70 random locations in different cities, there are no Nike stores around these locations.

I saved all names in a csv file. I set level='city' to Nike City Flagship Shops, level='factory' to Nike Factory Stores, and level='non' to random locations.

```
import pandas as pd
nike_store=pd.read_csv('nike_stores.csv')
nike_store.head()
```

	store	level	latitude	longitude
0	Nike Boston	city	NaN	NaN
1	Nike Bucktown	city	NaN	NaN
2	Nike Chicago	city	NaN	NaN
3	Nike Dallas North Park	city	NaN	NaN
4	Nike Eugene	city	NaN	NaN

```
nike_store.shape
```

```
(150, 4)
```

Using **Google Map API**, to get latitudes and longitudes of all 150 locations.

```
import requests

google_api_key='AIzaSyAZd9ADJwAj3PZlCn tubMXDQaxmyHasx0M'

def get_coordinates(api_key, address, verbose=False):
    try:
        url = 'https://maps.googleapis.com/maps/api/geocode/json?key={}&address={}'.format(api_key, address)
        response = requests.get(url).json()
        if verbose:
            print('Google Maps API JSON result =>', response)
        results = response['results']
        geographical_data = results[0]['geometry']['location'] # get geographical coordinates
        lat = geographical_data['lat']
        lon = geographical_data['lng']
        return lat, lon
    except:
        return None, None

for i in range(0,150):
    address = nike_store.iloc[i,0]
    lati,loni = get_coordinates(google_api_key, address)
    nike_store.iloc[i,2] = lati
    nike_store.iloc[i,3] = loni
```

Checking if there is any nan in the dataframe.

```
nike_store.isnull().any(axis=0)
```

```
store      False
level      False
latitude   False
longitude  False
dtype: bool
```

```
nike_store.head()
```

	store	level	latitude	longitude
0	Nike Boston	city	42.349985	-71.080032
1	Nike Bucktown	city	41.911654	-87.677723
2	Nike Chicago	city	41.894421	-87.623712
3	Nike Dallas North Park	city	32.867572	-96.773064
4	Nike Eugene	city	44.064194	-123.078073

Separate the locations,basing on the 'level'.

```
city_store=nike_store.loc[nike_store['level']=='city']
factory_store=nike_store.loc[nike_store['level']=='factory']
non_store=nike_store.loc[nike_store['level']=='non']
```

Now we can create a map to visualize the 150 locations, using folium.

Nike City Flagship Shops are marked by **green dots**.

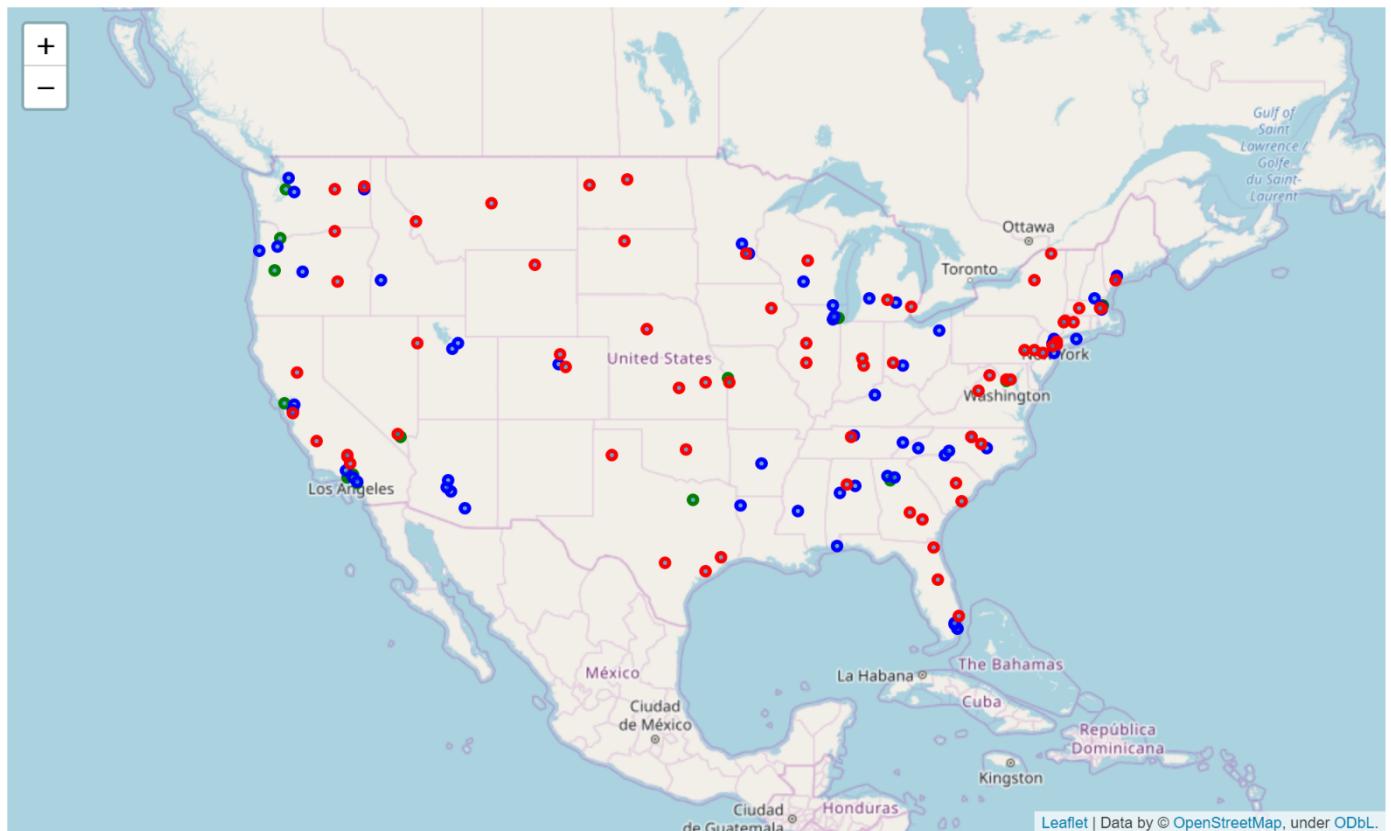
Nike Factory Stores are marked by **blue dots**.

Random locations are marked by **red dots**.

BOOM!

```
import folium # map rendering library
map_nike = folium.Map(location=[nike_store['latitude'][3], nike_store['longitude'][3]], zoom_start=4)
# add markers to map
def add_to_map(store,col):
    for lat, lng, label in zip(store['latitude'], store['longitude'], store['store']):
        label = folium.Popup(label, parse_html=True)
        folium.CircleMarker(
            [lat, lng],
            radius=3,
            popup=label,
            color=col,
            fill=True,
            fill_color='#3186cc',
            fill_opacity=0.7,
            parse_html=False).add_to(map_nike)

add_to_map(city_store, 'green')
add_to_map(factory_store, 'blue')
add_to_map(non_store, 'red')
map_nike
```



(You can **ZOOM** in to see the details. What a great module!)

Using Foursquare API to get the number of nearby venues

After having 150 targets' specific locations, I use Foursquare API to get the number of nearby venues.

An ideal location for a Nike store should be located in areas where commercial activities are intensive. That is to say, an ideal location should have many restaurants and shops nearby. Convenient public transport and parking are also required. As for a sportswear brand, being near to some stadiums and sports events may be also a good choice.

Therefore, I selected 8 categories from Foursquare Venue Categories and decided to use them as factors to distinguish the ideal locations.



8 categories are:

Stadium
Food
Athletics & Sports
Parking
Shop & Service
Clothing Store
Bus Stop
Metro Station

I also set 3 different radiiuses: **200 meters**, **500 meters**, **1000 meters** for searching nearby venues. In this way, I can combine different factors with different searching radiiuses, and have more choice when building decision tree models, then I can choose the one with the highest accuracy.

I saved the categories' IDs, radiiuses, and factor names in a csv file.

```
cat=pd.read_csv('category.csv')
cat.head()
```

	category	id	radius	name
0	Stadium	4bf58dd8d48988d184941735	200	Stadium_200
1	Stadium	4bf58dd8d48988d184941736	500	Stadium_500
2	Stadium	4bf58dd8d48988d184941737	1000	Stadium_1000
3	Food	4d4b7105d754a06374d81259	200	Food_200
4	Food	4d4b7105d754a06374d81260	500	Food_500

```
cat.shape
```

```
(24, 4)
```

Then, we got 24 venue-factors

```
from pandas.io.json import json_normalize
client_id = '5RJHC20GS14K1TPA12Z2F4W0ACZOYWINQ5UZCYEBMLCQE2IV' # Foursquare ID
client_secret = 'GDPN3YE4JTUOGRRCP1MN2ON2EYGBAYAP43AKKJCTEDHNZTB' # Foursquare Secret
version = '20190801' # Foursquare API version
limit = 500 #limit venues for each search
```

(Even setting the limit to 500, for a free Foursquare account, the number of results to return for each search is up to 50. This has some influence on choosing factors, we will talk about it later)

Define the function to get numbers of venues from each category for each location. Directly add the numbers we get to dataframe **nike_store**. And save the whole data as a csv file.

```
def getNearbyVenues(names, latitudes, longitudes, category, category_name, radius):
    venues_num=[]
    for name, lat, lng in zip(names, latitudes, longitudes):
        print(name)
        # create the API request URL
        url = 'https://api.foursquare.com/v2/venues/search?client_id={}&client_secret={}&ll={},{}&v={}&categoryIds={}&radius={}&limit={}'.format(
            client_id, client_secret, lat, lng, version, category, radius, limit)

        # make the GET request
        results = requests.get(url).json()
        venues = results['response']['venues']
        venues_list = json_normalize(venues)
        venues_number=venues_list.shape[0]
        venues_num.append(venues_number)
    category_df=pd.DataFrame({'store':nike_store['store'], category_name:venues_num})
    df_add=pd.merge(nike_store, category_df, on = 'store', how='left')
    return(df_add)
```

```
for i in range(0,24):
    nike_store = getNearbyVenues(names=nike_store['store'], latitudes=nike_store['latitude'], longitudes=nike_store['longitude'],
                                category=cat['id'][i],
                                category_name=cat['name'][i],
                                radius=cat['radius'][i])
import os
os.getcwd()
nike_store.to_csv('nike_data.csv')
```

On May 31st of 2018, Foursquare updated their API, and unlike before, now there are some limitations on how many calls you can make to the API. So when you create a free developer account, the default type is the sandbox account, with 950 regular calls per day and 50 premium calls per day. And my request for data from the above code is totally beyond the limit. So, I actually separate the for loop and used several free accounts to get the whole data. That way looks ugly :(, I won't show it here. A premium account might help(I can't afford it, maybe Nike can pay it for me?).

Finally, we get the data of locations' nearby venues.

```
nike_store=pd.read_csv('nike_data.csv')
nike_store.head()
```

		Unnamed: 0	store	level	latitude	longitude	Stadium_200	Stadium_500	Stadium_1000	Food_200	Food_500	...	Shop_Service_1000	Clothing_Store_
0	0	Nike Boston	city	42.349985	-71.080032		0	0	2	49	50	...		50
1	1	Nike Bucktown	city	41.911654	-87.677723		0	1	1	47	50	...		50
2	2	Nike Chicago	city	41.894421	-87.623712		1	2	2	50	50	...		50
3	3	Nike Dallas North Park	city	32.867572	-96.773064		0	0	0	46	50	...		50
4	4	Nike Eugene	city	44.064194	-123.078073		0	0	1	9	30	...		50

5 rows × 29 columns

Attention! We can find from the above dataframe that the max value of all factors is 50. As I mentioned before, for a free Foursquare account, the number of results to return for each search is up to 50. When we choosing factors for machine learning, we should be careful with the factors with many 50, like 'Food_1000', 'Shop_Service_1000'.

Now, let's make a **radar picture** to visualize the factors of three different levels.
Again, separate the locations into three groups, basing on the 'level'.

```
city_store=nike_store.loc[nike_store['level']=='city']
factory_store=nike_store.loc[nike_store['level']=='factory']
non_store=nike_store.loc[nike_store['level']=='non']
```

First, we need to calculate the mean values of each factor of each group.

```
import numpy as np
city_factors_mean=city_store.mean()
labels=np.array(city_factors_mean[3:].index)
city_factors_mean=np.array(city_factors_mean[3:])
city_factors_mean=np.concatenate((city_factors_mean, [city_factors_mean[0]]))

factory_factors_mean=factory_store.mean()
factory_factors_mean=np.array(factory_factors_mean[3:])
factory_factors_mean=np.concatenate((factory_factors_mean, [factory_factors_mean[0]]))

non_store_factors_mean=non_store.mean()
non_store_factors_mean=np.array(non_store_factors_mean[3:])
non_store_factors_mean=np.concatenate((non_store_factors_mean, [non_store_factors_mean[0]]))
```

Then, let's draw

```
import matplotlib.pyplot as plt
n=24
angles=np.linspace(0, 2*np.pi, n, endpoint=False)
angles=np.concatenate((angles, [angles[0]]))

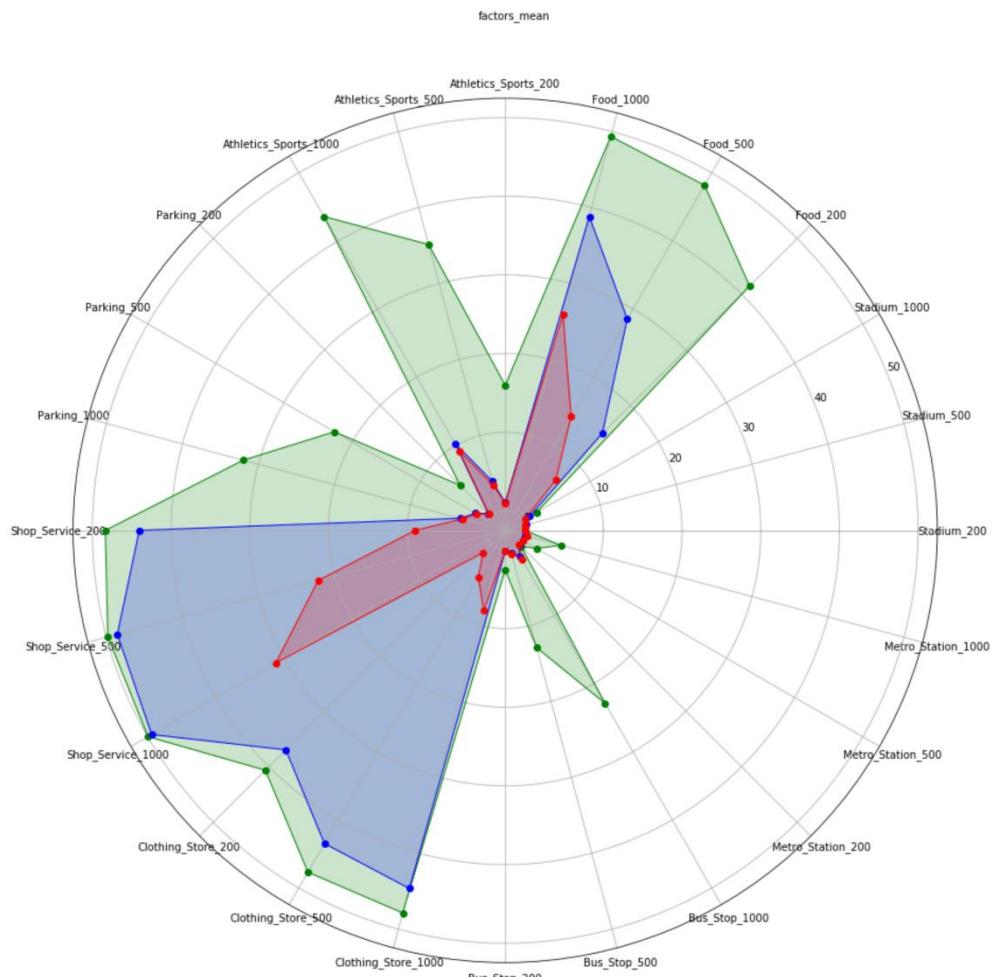
fig=plt.figure(facecolor="white")
plt.figure(figsize=(15, 15))
plt.subplot(111, polar=True)
plt.plot(angles,city_factors_mean, 'bo-', color='g', linewidth=1)
plt.fill(angles,city_factors_mean, facecolor='g', alpha=0.2)

plt.plot(angles,factory_factors_mean, 'bo-', color='blue', linewidth=1)
plt.fill(angles,factory_factors_mean, facecolor='blue', alpha=0.2)

plt.plot(angles,non_store_factors_mean, 'bo-', color='r', linewidth=1)
plt.fill(angles,non_store_factors_mean, facecolor='r', alpha=0.2)

plt.thetagrids(angles*180/np.pi, labels)
plt.figtext(0.52,0.95, 'factors_mean', ha='center')
plt.grid(True)
plt.savefig('data_radar.JPG')
plt.show()
```

<matplotlib.figure.Figure at 0x1ee7e773940>



Factors of **Nike City Flagship Shops** are marked by green dots.

Factors of **Nike Factory Stores** are marked by blue dots.

Factors of **Random locations** are marked by red dots.

Now, you know why this is called 'radar picture'.

From the picture above, we surely can say three groups of locations do have different venue-factors. Some are very significant, like Food category. Some are significant for the difference between Nike stores and random locations, like Shop &Service, Clothing Store Category. Some are significant for the difference between Nike City Flagship Shops and the rest, like Athletics & Sports, Parking, Bus Stop, Metro Station Categories. The means of Stadium Categories are very small, but for each group, the values are still different, Nike City Flagship Shops have the highest means for all three Stadium Categories.

Methodology

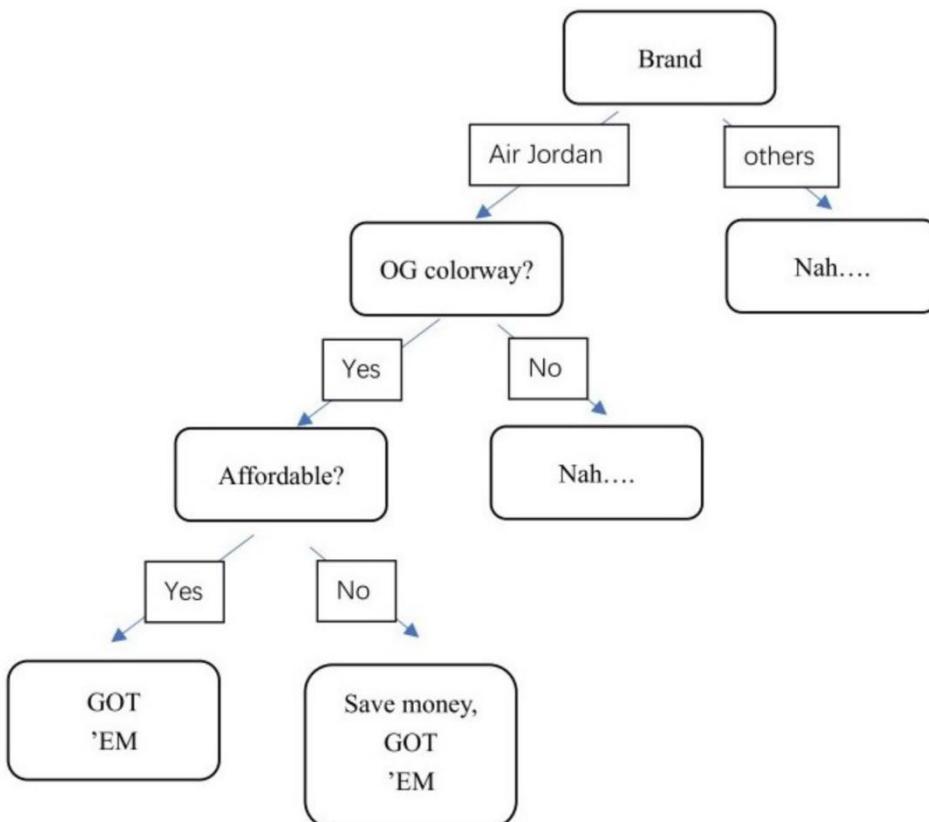
Machine Learning: decision tree model

A decision tree is a flowchart-like structure in which each internal node represents a "test" on an attribute (e.g. whether a coin flip comes up heads or tails), each branch represents the outcome of the test, and each leaf node represents a class label (decision taken after computing all attributes). The paths from root to leaf represent classification rules.

(This is the definition from Wikipedia, BORING, right? Thank you for being awake!)

Let me show you a vivid example.

This is the decision tree of an **OG Jordan sneakerhead** making a purchase decision.



Brand, OG colorway, Affordable are like venus-factors, purchase decisions are like 'levels' of locations.

Now let's use **scikit-learn** to build some more complex decision trees.

```
from sklearn.tree import DecisionTreeClassifier
```

Select 4 combinations of venue-factors.

```
X1 = nike_store[['Stadium_500', 'Food_200', 'Athletics_Sports_500', 'Parking_1000', 'Shop_Service_200', 'Clothing_Store_500', 'Bus_Stop_1000', 'Met
X2 = nike_store[['Stadium_200', 'Food_500', 'Athletics_Sports_200', 'Parking_500', 'Shop_Service_500', 'Clothing_Store_200', 'Bus_Stop_200', 'Met
X3 = nike_store[['Stadium_1000', 'Food_200', 'Athletics_Sports_1000', 'Parking_200', 'Shop_Service_200', 'Clothing_Store_200', 'Bus_Stop_1000', 'Met
X4 = nike_store[['Stadium_200', 'Food_200', 'Athletics_Sports_200', 'Parking_200', 'Shop_Service_200', 'Clothing_Store_200', 'Bus_Stop_200', 'Met
```

Get train-sets and test-sets.

```
y = nike_store['level']
from sklearn.model_selection import train_test_split
X1_trainset, X1_testset, y1_trainset, y1_testset = train_test_split(X1, y, test_size=0.3, random_state=3)
X2_trainset, X2_testset, y2_trainset, y2_testset = train_test_split(X2, y, test_size=0.3, random_state=3)
X3_trainset, X3_testset, y3_trainset, y3_testset = train_test_split(X3, y, test_size=0.3, random_state=3)
X4_trainset, X4_testset, y4_trainset, y4_testset = train_test_split(X4, y, test_size=0.3, random_state=3)
```

Also, try different max depths, select the model with the highest accuracy.

```
from sklearn import metrics
from sklearn.metrics import f1_score
from sklearn.metrics import jaccard_similarity_score

def get_accuracy(X_trainset, X_testset, y_trainset, y_testset):
    d_range = range(8, 18)
    accuracy = []
    f1 = []
    ja = []
    for d in d_range:
        Tree = DecisionTreeClassifier(criterion="entropy", max_depth = d)
        Tree.fit(X_trainset, y_trainset)
        predTree = Tree.predict(X_testset)
        accuracy.append(metrics.accuracy_score(y_testset, predTree))
        f1.append(f1_score(y_testset, predTree, average='weighted'))
        ja.append(jaccard_similarity_score(y_testset, predTree))

    result = pd.DataFrame(accuracy, index=['d=8', 'd=9', 'd=10', 'd=11', 'd=12', 'd=13', 'd=14', 'd=15', 'd=16', 'd=17'])
    result.columns = ['accuracy']
    result.insert(loc=1, column='Jacard', value=ja)
    result.insert(loc=2, column='F1-score', value=f1)
    result.columns.name = "Depth"
    return(result)
```

```
get_accuracy(X1_trainset, X1_testset, y1_trainset, y1_testset)
```

Depth	accuracy	Jacard	F1-score
d=8	0.844444	0.844444	0.849486
d=9	0.866667	0.866667	0.868315
d=10	0.888889	0.888889	0.889412
d=11	0.911111	0.911111	0.911111
d=12	0.888889	0.888889	0.890009
d=13	0.888889	0.888889	0.887749
d=14	0.866667	0.866667	0.867988
d=15	0.866667	0.866667	0.868315
d=16	0.911111	0.911111	0.911111
d=17	0.911111	0.911111	0.911111

```
get_accuracy(X2_trainset, X2_testset, y2_trainset, y2_testset)
```

Depth	accuracy	Jacard	F1-score
d=8	0.955556	0.955556	0.955347
d=9	0.888889	0.888889	0.892035
d=10	0.911111	0.911111	0.912654
d=11	0.955556	0.955556	0.955996
d=12	0.933333	0.933333	0.934910
d=13	0.955556	0.955556	0.955347
d=14	0.911111	0.911111	0.912654
d=15	0.933333	0.933333	0.933694
d=16	0.955556	0.955556	0.955347
d=17	0.955556	0.955556	0.955432

```
get_accuracy(X3_trainset,X3_testset,y3_trainset,y3_testset)
```

Depth	accuracy	Jacard	F1-score
d=8	0.866667	0.866667	0.868765
d=9	0.888889	0.888889	0.889155
d=10	0.866667	0.866667	0.868765
d=11	0.866667	0.866667	0.868765
d=12	0.888889	0.888889	0.889155
d=13	0.888889	0.888889	0.889155
d=14	0.888889	0.888889	0.889155
d=15	0.866667	0.866667	0.868765
d=16	0.888889	0.888889	0.889155
d=17	0.866667	0.866667	0.868765

```
get_accuracy(X4_trainset,X4_testset,y4_trainset,y4_testset)
```

Depth	accuracy	Jacard	F1-score
d=8	0.933333	0.933333	0.934655
d=9	0.888889	0.888889	0.891301
d=10	0.933333	0.933333	0.934655
d=11	0.888889	0.888889	0.891301
d=12	0.800000	0.800000	0.810662
d=13	0.911111	0.911111	0.914198
d=14	0.933333	0.933333	0.934655
d=15	0.800000	0.800000	0.810662
d=16	0.911111	0.911111	0.914198
d=17	0.933333	0.933333	0.934655

From the above data, we can say that from 40 trees, **X2 dataset with max depth=11** has the highest accuracy.
So, use this to train the decision tree model.

```
the_tree=DecisionTreeClassifier(criterion="entropy", max_depth = 11)
the_tree.fit(X2_trainset,y2_trainset)
```

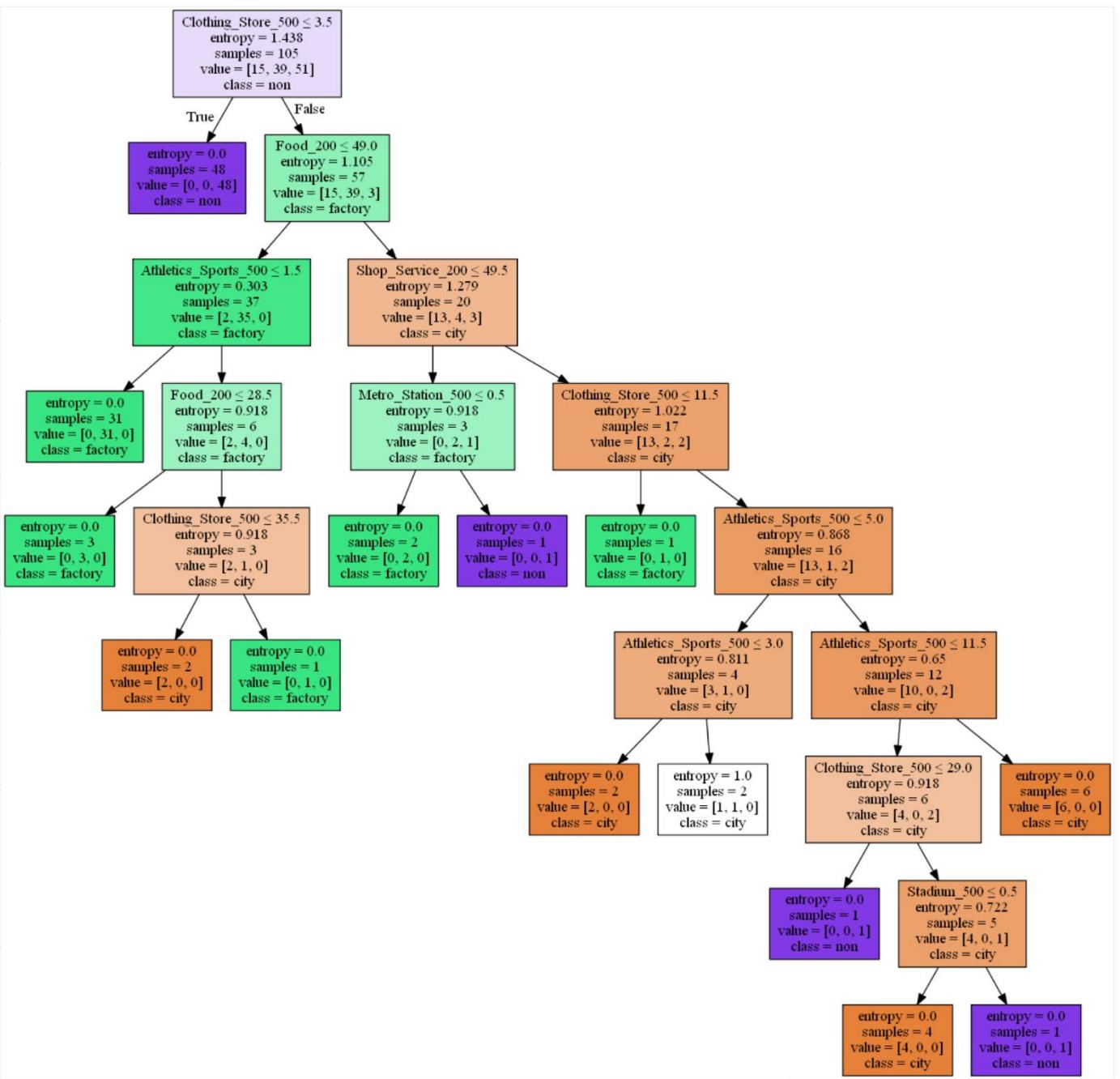
```
DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=11,
max_features=None, max_leaf_nodes=None,
min_impurity_decrease=0.0, min_impurity_split=None,
min_samples_leaf=1, min_samples_split=2,
min_weight_fraction_leaf=0.0, presort=False, random_state=None,
splitter='best')
```

Let's visualize the tree.

```
from sklearn.externals.six import StringIO
import pydotplus
import matplotlib.image as mpimg
from sklearn import tree
%matplotlib inline

dot_data = StringIO()
filename = "niketree.png"
featureNames = ['Stadium_500', 'Food_200', 'Athletics_Sports_500', 'Parking_1000', 'Shop_Service_200', 'Clothing_Store_500', 'Bus_Stop_1000', 'Met
targetNames = nike_store['level'].unique().tolist()
out=tree.export_graphviz(the_tree, feature_names=featureNames, out_file=dot_data, class_names= np.unique(y2_trainset), filled=True,  special_cha
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
graph.write_png(filename)
img = mpimg.imread(filename)
plt.figure(figsize=(300, 500))
plt.imshow(img, interpolation='nearest')
```

<matplotlib.image.AxesImage at 0x1ee09042ef0>



Now, we got the decision tree model, which will be used to predict the ideal location in Santa Fe later.

Finding the ideal locations

Creating latitude & longitude coordinates for centroids

Now, we need to create **latitude & longitude coordinates** for **centroids** of our candidate neighborhoods. We will create a grid of cells covering our area of interest which is approximately 7x7 kilometers centered around Santa Fe city center: **the Santa Fe plaza**.

Let's first use Google map API to find the latitude & longitude of Santa Fe plaza.

```

Santa_Fe=get_coordinates(google_api_key, 'Santa Fe Plaza')
Santa_Fe
(35.6874277, -105.9385121)
  
```

Now let's create a grid of area candidates, equally spaced, centered around the city center and within 3.5km from Santa Fe plaza.

Because the venue-factors chosen by me includes 200 meters and 500 meters, the neighborhoods of candidates will be defined as circular areas with a radius of 350 meters, so our neighborhood centers will be 700 meters apart.

To calculate distances we need to create our grid of locations in the Cartesian 2D coordinate system which allows us to calculate distances in meters (not in latitude/longitude degrees). Then we'll project those coordinates back to latitude/longitude degrees to be shown on Folium map. So let's create functions to convert between **WGS84 spherical coordinate system (latitude/longitude degrees)** and **UTM Cartesian coordinate system (X/Y coordinates in meters)**.

```

import pyproj
import math

def lonlat_to_xy(lon, lat):
    proj_latlon = pyproj.Proj(proj='latlong', datum='WGS84')
    proj_xy = pyproj.Proj(proj="utm", zone=33, datum='WGS84')
    xy = pyproj.transform(proj_latlon, proj_xy, lon, lat)
    return xy[0], xy[1]

def xy_to_lonlat(x, y):
    proj_latlon = pyproj.Proj(proj='latlong', datum='WGS84')
    proj_xy = pyproj.Proj(proj="utm", zone=33, datum='WGS84')
    lonlat = pyproj.transform(proj_xy, proj_latlon, x, y)
    return lonlat[0], lonlat[1]

def calc_xy_distance(x1, y1, x2, y2):
    dx = x2 - x1
    dy = y2 - y1
    return math.sqrt(dx*dx + dy*dy)

```

```

Santa_Fe_x, Santa_Fe_y = lonlat_to_xy(Santa_Fe[1], Santa_Fe[0]) # City center

k = math.sqrt(3) / 2 # Vertical offset for hexagonal grid cells
x_min = Santa_Fe_x - 5000
x_step = 1000
y_min = Santa_Fe_y - 5000 - (int(21/k)*k*500 - 10000)/2
y_step = 1000 * k

S_latitudes = []
S_longitudes = []
distances_from_center = []
S_xs = []
S_ys = []
for i in range(0, int(21/k)):
    S_y = y_min + i * y_step
    x_offset = 500 if i%2==0 else 0
    for j in range(0, 21):
        S_x = x_min + j * x_step + x_offset
        distance_from_center = calc_xy_distance(Santa_Fe_x, Santa_Fe_y, S_x, S_y)
        if (distance_from_center <= 5001):
            S_lon, S_lat = xy_to_lonlat(S_x, S_y)
            S_latitudes.append(S_lat)
            S_longitudes.append(S_lon)
            distances_from_center.append(distance_from_center)
            S_xs.append(S_x)
            S_ys.append(S_y)

print(len(S_latitudes), 'candidate neighborhood centers generated.')

```

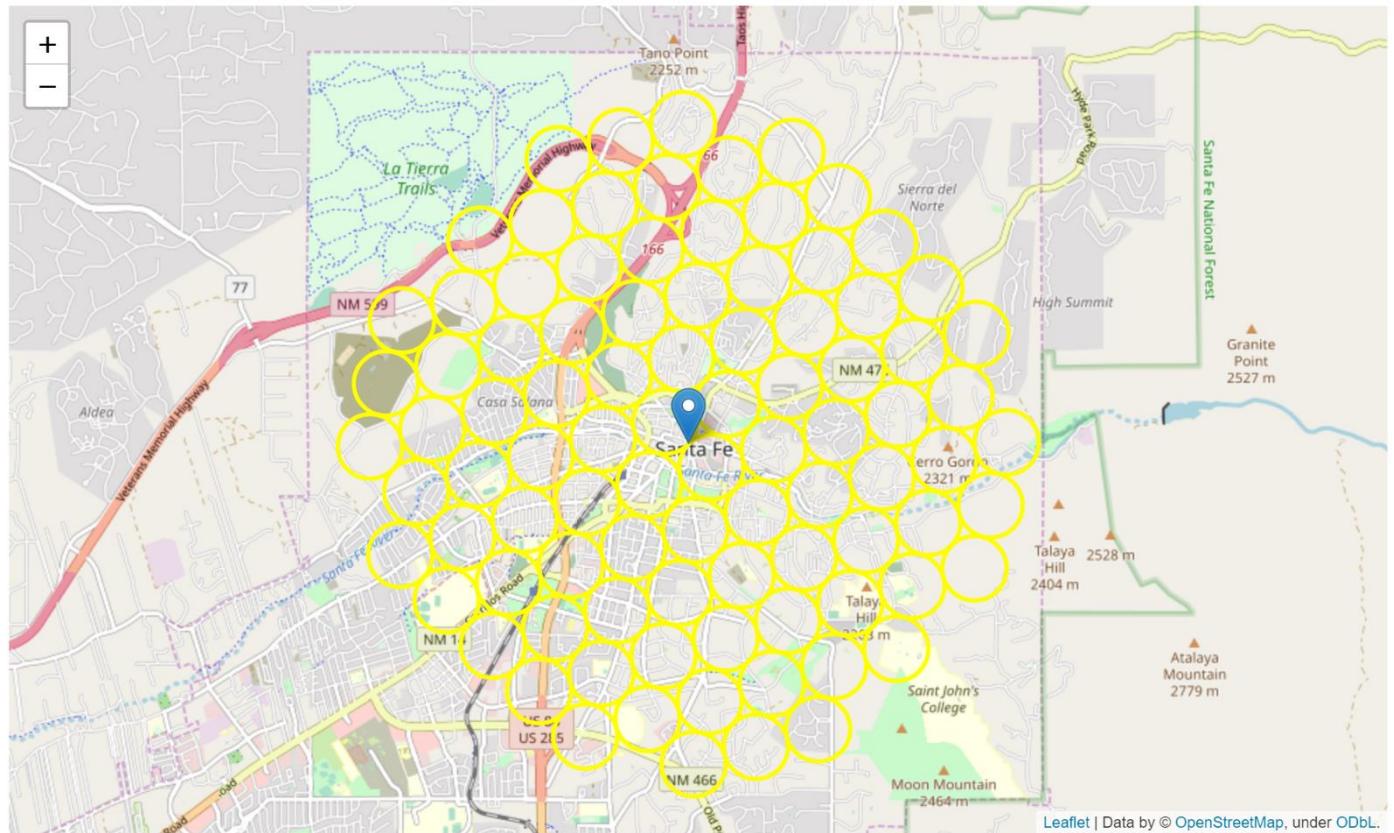
92 candidate neighborhood centers generated.

(reference)

At first, I set step = 700 and x_min = Sante_Fe_x - 3500, but didn't get the result I want. After several adjustments of parameters(steps,boundaries), I got the ideal result(neighborhood centers are 700 meters apart).

Visualizing the neighborhoods of candidates on a map

```
import folium
map_Santa_Fe = folium.Map(location=Santa_Fe, zoom_start=13)
folium.Marker(Santa_Fe, popup='Santa_Fe_Plaza').add_to(map_Santa_Fe)
for lat, lon in zip(S_latitudes, S_longitudes):
    folium.Circle([lat, lon], radius=350, color='yellow', fill=False).add_to(map_Santa_Fe)
map_Santa_Fe
```



Using Foursquare API to get the number of nearby venues of neighborhoods

Similarly, I used Foursquare API to get the number of nearby venues of neighborhoods. This time we only need 8 categories, according to the decision tree model we selected.

```
: SF_area = pd.DataFrame(S_latitudes)
SF_area.columns = ['latitude']
SF_area.insert(loc=1, column='longitude', value=S_longitudes)

: client_id = 'ZU2FD1LACBYI042025UBOPGKJ4RGCKQYKJ3TYMXF110QAA1'
client_secret = 'XKTLJPL5WFQT40IX4CYE1AXYOSJAOXIOBG0VMJURQQ5MRO'
version = '20190801' # Foursquare API version
limit = 500

: def SF_getNearbyVenues(category, category_name, radius, latitudes=SF_area['latitude'], longitudes=SF_area['longitude']):
venues_num=[]
for lat, lng in zip(latitudes, longitudes):
    # create the API request URL
    url = 'https://api.foursquare.com/v2/venues/search?client_id={}&client_secret={}&ll={,}&v={}&categoryId={}&radius={}&limit={}'.format(
        client_id, client_secret, lat, lng, version, category, radius, limit)

    # make the GET request
    results = requests.get(url).json()
    venues = results['response']['venues']
    venues_list = json_normalize(venues)
    venues_number=venues_list.shape[0]
    venues_num.append(venues_number)
SF_area[category_name]=venues_num
return(SF_area)
```

```

SF_getNearbyVenues(category=cat['id'][0], category_name=cat['name'][0], radius=cat['radius'][0])
SF_getNearbyVenues(category=cat['id'][3], category_name=cat['name'][4], radius=cat['radius'][4])
SF_getNearbyVenues(category=cat['id'][6], category_name=cat['name'][6], radius=cat['radius'][6])
SF_getNearbyVenues(category=cat['id'][9], category_name=cat['name'][10], radius=cat['radius'][10])
SF_getNearbyVenues(category=cat['id'][12], category_name=cat['name'][13], radius=cat['radius'][13])
SF_getNearbyVenues(category=cat['id'][15], category_name=cat['name'][15], radius=cat['radius'][15])
SF_getNearbyVenues(category=cat['id'][18], category_name=cat['name'][18], radius=cat['radius'][18])
SF_getNearbyVenues(category=cat['id'][21], category_name=cat['name'][21], radius=cat['radius'][21])

import os
os.getcwd()
SF_area.to_csv('SF_area.csv')

```

Because Foursquare API has limits of 950 regular calls per day and 50 premium calls per day, and one more time, my request for data from the above code is totally beyond the limit. I actually used several free accounts to get the whole data and saved it in a csv file.

```

Santa_Fe=pd.read_csv('SF_area.csv')
Santa_Fe.head()

```

	latitude	longitude	Stadium_200	Food_500	Athletics_Sports_200	Parking_500	Shop_Service_500	Clothing_Store_200	Bus_Stop_200	Metro_Station_200
0	35.698403	-105.903172	0	0	0	0	0	0	0	0
1	35.702927	-105.908838	0	0	0	0	0	0	0	0
2	35.707451	-105.914504	0	0	0	0	0	0	0	0
3	35.711976	-105.920173	0	1	0	0	0	0	0	0
4	35.716500	-105.925842	0	1	0	0	0	0	0	0

Using the selected decision tree model to predict the 'level' of each candidate

Now, we have the decision tree model and venue-factors of each candidate.

We can put the factors into decision tree model, the model will divide the candidates into three groups: **Nike City Flagship Shops**, **Nike Factory Stores** and **random locations**.

```

SF_X = Santa_Fe[['Stadium_200', 'Food_500', 'Athletics_Sports_200', 'Parking_500', 'Shop_Service_500', 'Clothing_Store_200', 'Bus_Stop_200', 'Metro_Station_200']]

```

```

prediction = the_tree.predict(SF_X)
Santa_Fe['prediction']=prediction

```

```

import os
os.getcwd()
Santa_Fe.to_csv('Santa_Fe_predict.csv')

```

```

ideal_city_store=Santa_Fe.loc[Santa_Fe['prediction']=='city']
print(len(ideal_city_store['prediction']), 'ideal city stores were found.')
ideal_factory_store=Santa_Fe.loc[Santa_Fe['prediction']=='factory']
print(len(ideal_factory_store['prediction']), 'ideal factory stores were found.')

```

0 ideal city stores were found.
2 ideal factory stores were found.

From 92 candidate neighborhoods, we found two suitable neighborhoods to build a Nike Factory store.

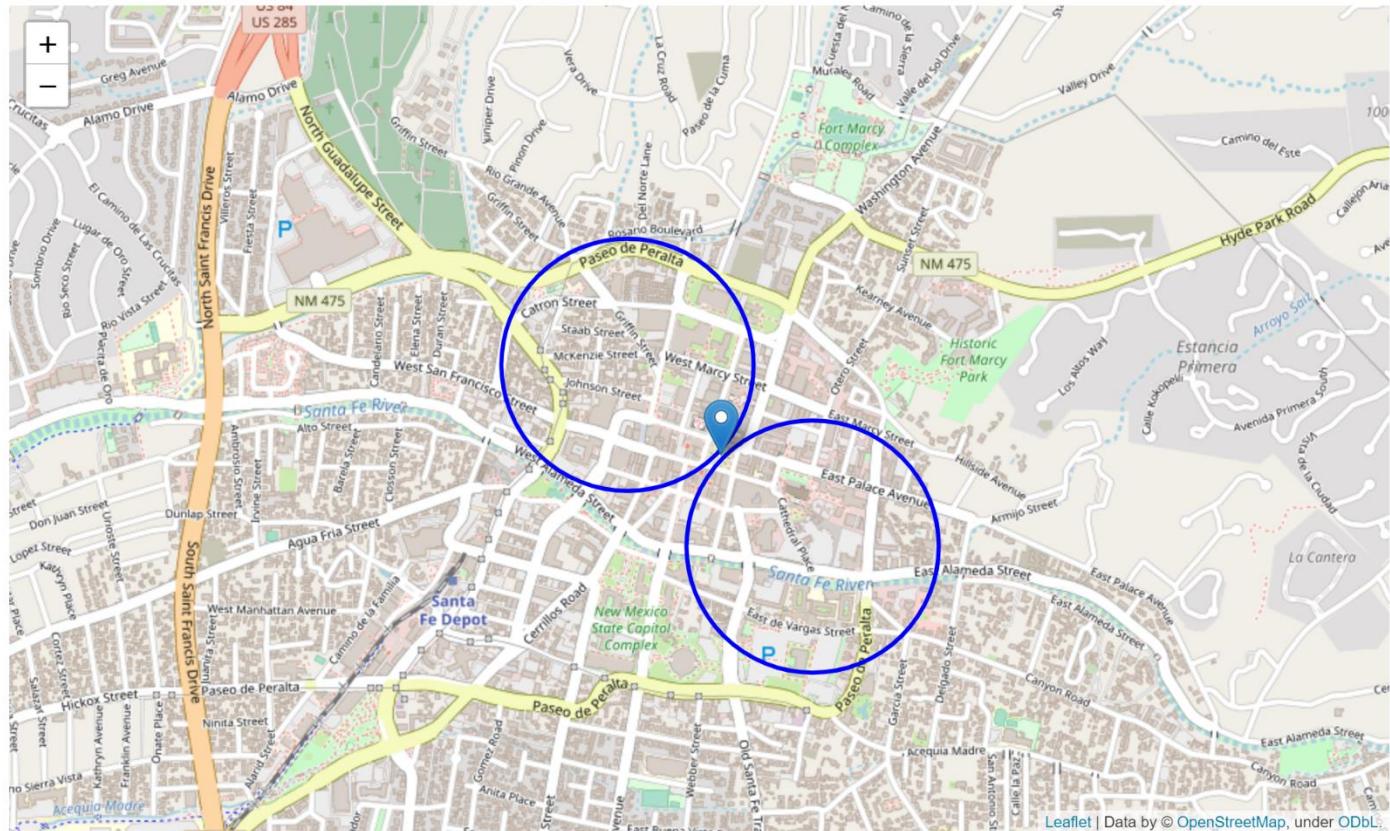
Results

Now let's visualize these two neighborhoods.

```
ideal_factory_store
```

latitude	longitude	Stadium_200	Food_500	Athletics_Sports_200	Parking_500	Shop_Service_500	Clothing_Store_200	Bus_Stop_200	Metro_Station_20
45	35.685167	-105.935677	0	50	1	2	50	6	0
46	35.689688	-105.941347	0	50	1	3	50	6	1

```
plaza=get_coordinates(google_api_key, 'Santa Fe Plaza')
map_ideal_factory_stores = folium.Map(location=plaza, zoom_start=15)
folium.Marker(plaza, popup='Santa Fe Plaza').add_to(map_ideal_factory_stores)
for lat, lon in zip(ideal_factory_store['latitude'], ideal_factory_store['longitude']):
    folium.Circle([lat, lon], radius=350, color='blue', fill=False).add_to(map_ideal_factory_stores)
map_ideal_factory_stores
```



(You can **ZOOM** in to see the details.)

We can find these two neighborhoods are very near to Sante Fe plaza, where commercial activities are intensive. If you zoom in, you can find there are lots of restaurants, shops, and parking lot in this area. Therefore, this result is reasonable.

Let's find the addresses of these two neighborhoods' centers.

```
def get_address(api_key, latitude, longitude, verbose=False):
    try:
        url = 'https://maps.googleapis.com/maps/api/geocode/json?key={}&latlng={},{}'.format(api_key, latitude, longitude)
        response = requests.get(url).json()
        if verbose:
            print('Google Maps API JSON result =>', response)
        results = response['results']
        address = results[0]['formatted_address']
        return address
    except:
        return None
```

```
address1=get_address(google_api_key, ideal_factory_store['latitude'][45], ideal_factory_store['longitude'][45], verbose=False)
address2=get_address(google_api_key, ideal_factory_store['latitude'][46], ideal_factory_store['longitude'][46], verbose=False)
print('The addresses of ideal location for Nike factory stores are:', address1, 'and', address2)
```

The addresses of ideal location for Nike factory stores are: 223 Cathedral Pl, Santa Fe, NM 87501, USA and 214 Mckenzie St, Santa Fe, NM 87501, USA



Conclusion and Discussion

In this project, I used the data of nearby venues to characterize the existing Nike store. The data includes the number of nearby restaurants, stadiums, shops, bus stops and so on. I used the data to build a machine learning model(decision tree) and then used this model to select the appropriate locations in Santa Fe.

This project may look naive in the real world, because I ignored many factors, like rental, government permission and so on.

But I believe this could be helpful for Nike to make a decision. With more improvements, this can reduce the cost of human onsite research. This project adopts a very universal logical approach, it can be extended to more types of stores and locations.

Reference

- [Nike Directory](#)
- [Wikipedia](#)
- [IBM-Applied Data Science Capstone](#)
- [IBM-Machine Learning with Python](#)
- [Foursquare-Search for Venues](#)

Thank you so much for reading!