homework2

February 22, 2023

1 Homework 2 (100 Points)

The goal of this homework is to get more practice with pandas and get practice with clustering on various datasets.

1.1 Exercise 1 - (50 points)

This exercise will be using the Airbnb dataset for NYC called `listings.csv`. You can download it directly here

a) Produce a Heatmap using the Folium package (you can install it using pip) of the mean listing price per location (lattitude and longitude) over the NYC map. (5 points)

Hints: 1. generate a base map of NYC to plot over: default_location=[40.693943, -73.985880] 2. generate an HTML file named `index.html` - open it in your browser and you'll see the heatmap

[45]:
```
import pandas as pd
import numpy as np
from folium import plugins, Map, Marker, vector_layers
from folium.plugins import HeatMap
import matplotlib.pyplot as plt

df = pd.read_csv('listings.csv', low_memory=False)
base_map = Map(location = [40.693943, -73.985880])

df['latitude'] = df['latitude'].astype(float)
df['longitude'] = df['longitude'].astype(float)
df['price'] = df['price'].astype(float)
mean = df.groupby(['latitude', 'longitude'], as_index=False)['price'].mean()
HeatMap(mean).add_to(base_map) # ,radius=8,max_zoom=13
base_map.save("index.html")
base_map
```

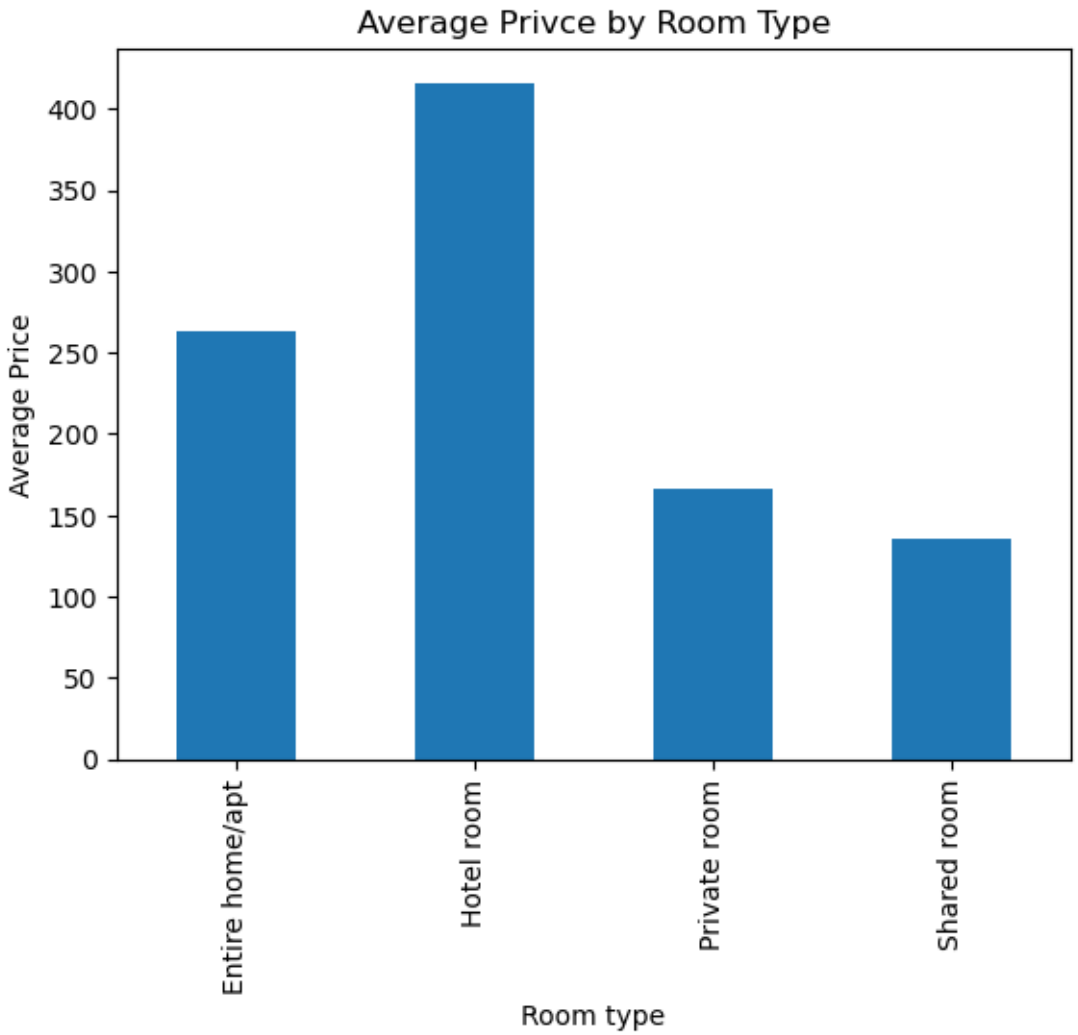
[45]:
```
<folium.folium.Map at 0x7fe6602e39d0>
```

b) Plot a bar chart of the average price per room type. Briefly comment on the relation between price and room type. - (2.5 pts)

[46]:
```
df = pd.read_csv('listings.csv', low_memory=False)
aver_price = df.groupby(['room_type'])['price'].mean()
print(aver_price)
aver_price.plot.bar(x='room_type', y='price',
                    xlabel = 'Room type', ylabel = 'Average Price',
                    title = 'Average Privce by Room Type')
```

```
room_type
Entire home/apt    263.442404
Hotel room         416.164894
Private room       166.102042
Shared room        135.400376
Name: price, dtype: float64
```

[46]:
```
<AxesSubplot:title={'center':'Average Privce by Room Type'}, xlabel='Room type',
ylabel='Average Price'>
```



Hotel room is the most expensive room type (416). Entire home/apt is the sencond most expensive room type (263). Private room (166) is $31 higher than shared room (135).

c) Plot on the NYC map the top 10 most reviewed listings (Note: some could be in the same location) - (5 pts)

[47]:
```
df = pd.read_csv('listings.csv', low_memory=False)
most_reviewed = df.nlargest(n = 10, columns = 'number_of_reviews')
top_10 = list(zip(most_reviewed['latitude'],most_reviewed['longitude']))
top_10_viewed = Map(location = [40.72, -73.98], zoom_start = 12)
for i in range(10):
    Marker(top_10[i]).add_to(top_10_viewed)
top_10_viewed.save("top_10_viewed.html")
top_10_viewed
```

[47]:
```
<folium.folium.Map at 0x7fe6d6ec5fd0>
```

d) Using `longitude`, `latitude`, `price`, and `number_of_reviews`, use Kmeans to create 5 clusters. Plot the points on the NYC map in a color corresponding to their cluster. - (15 points)

[48]:
```
from sklearn.cluster import KMeans
import folium

X = pd.read_csv('listings.csv', low_memory=False)
X = X[['longitude', 'latitude', 'price', 'number_of_reviews']]
kmeans = KMeans(n_clusters = 5).fit(X)
X['cluster'] = kmeans.labels_

colors = ["purple", "red", "orange", "green", "blue"]
Kmeans_5_cluster = Map(location=[40.693943, -73.985880])

for lat, long, cluster in zip(X['latitude'], X['longitude'], X['cluster']):
    vector_layers.CircleMarker(
        [lat, long],
        radius = 0.5,
        fill = True,
        color = colors[cluster - 1],
        fill_color = colors[cluster - 1],
        fill_opacity = 0.6,
    ).add_to(Kmeans_5_cluster)

Kmeans_5_cluster.save('Kmeans_5_cluster.html')
Kmeans_5_cluster
```

[48]:
```
<folium.folium.Map at 0x7fe62aa29c10>
```

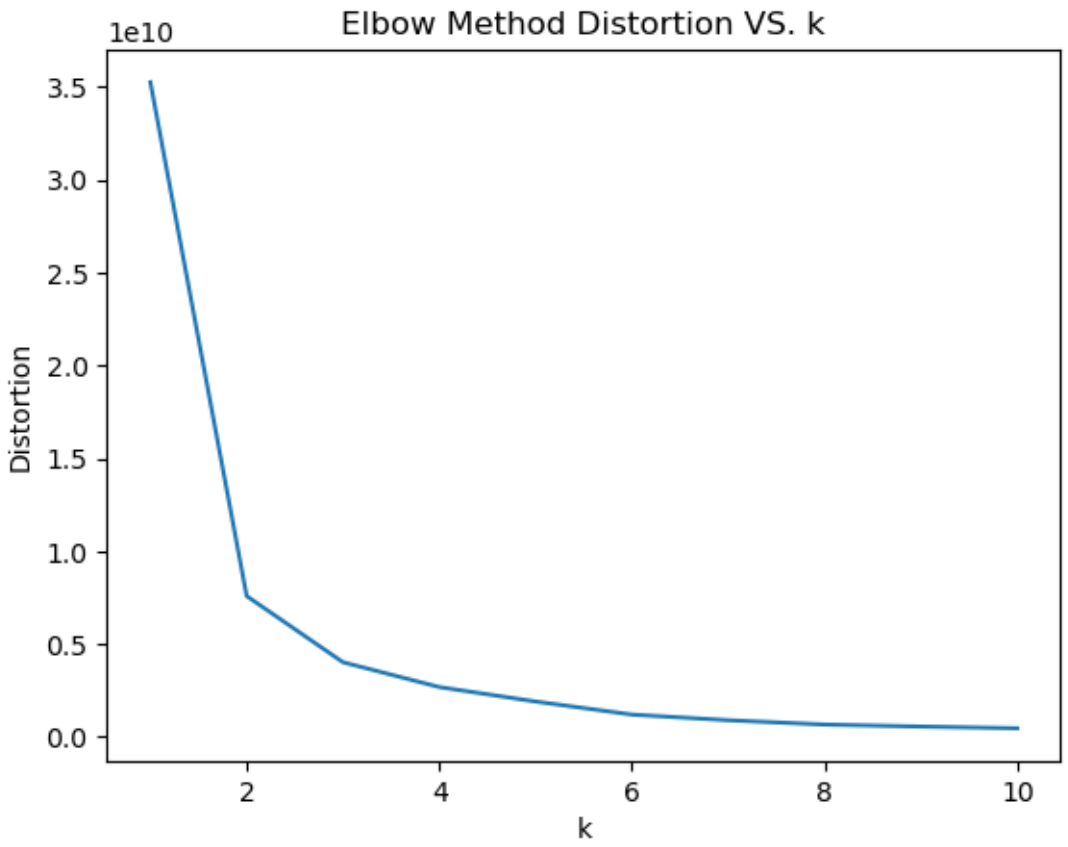
e) You should see points in the same cluster all over the map (i.e. not really clustered together...)

- briefly explain why that is. - (2.5 points)

The clusters are clustered based on not only latitude and longitude but also price and number of reviews. The reason points in the same cluster all over the map could be that these points have similar values for price and number of reviews. Making them seems a cluster although they are far away from each other in map.

f) How many clusters would you recommend using instead of 5? Display and interpret either the silhouette scores or the elbow method. - (5 points)

[49]:
```
distortion = []
K = range(1,11)
for k in K:
    kmeanModel = KMeans(n_clusters=k)
    kmeanModel.fit(X)
    distortion.append(kmeanModel.inertia_)
plt.plot(K, distortion)
plt.xlabel('k')
plt.ylabel('Distortion')
plt.title('Elbow Method Distortion VS. k')
plt.show()
```

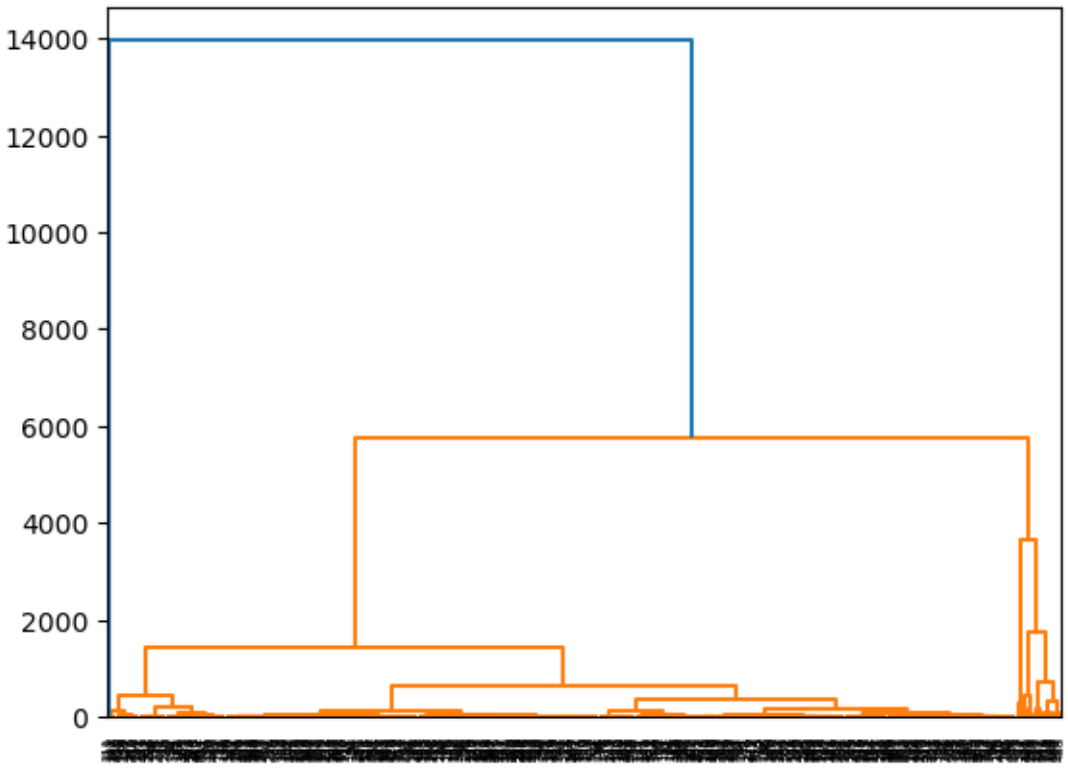


I recommend using 2 or 3 clusters instead of 5. Because after 3, the distortion value decreases almost linearly, indicating that they are not a good choice for k.

g) For all listings of type `Shared room`, plot the dendrogram of the hierarchical clustering generated from `longitude`, `latitude`, and `price`. You can use any distance function. - (10 points)

[50]:
```
from scipy.cluster import hierarchy

X = pd.read_csv('listings.csv', low_memory=False)
X = X[X['room_type'] == 'Shared room']
X = X[['latitude', 'longitude', 'price']]
l = hierarchy.linkage(X, 'ward')
hierarchy.dendrogram(l)
plt.show()
```

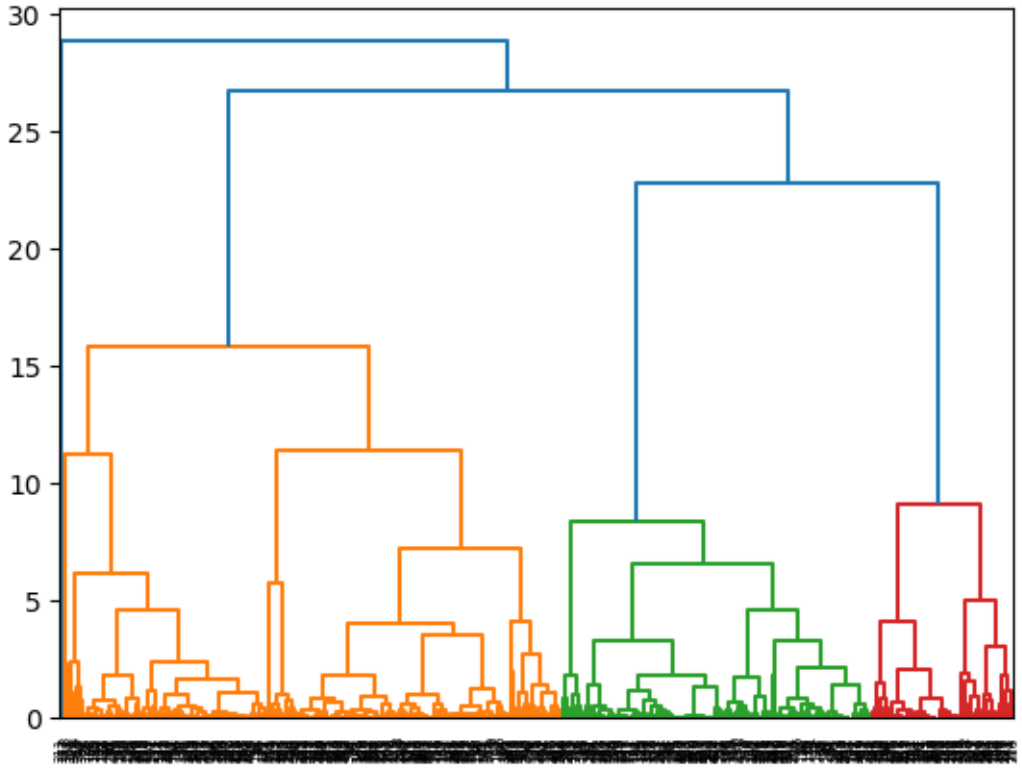


h) Normalize `longitude`, `latitude`, and `price` by subtracting by the mean (of the column) and dividing by the standard deviation (of the column). Repeat g) using the normalized data. Comment on what you observe. - (5 points)

[56]:
```
X = pd.read_csv('listings.csv', low_memory=False)
X = X[X['room_type'] == 'Shared room']
X = X[['latitude', 'longitude', 'price']]
```

```
X["norm_price"] = (X["price"]-X["price"].mean())/X["price"].std()
X = X.drop(['price'], axis=1)
X["norm_longitude"] = (X["longitude"]-X["longitude"].mean())/X["longitude"].
 ↪std()
X = X.drop(['longitude'], axis=1)
X["norm_latitude"] = (X["latitude"]-X["latitude"].mean())/X["latitude"].std()
X = X.drop(['latitude'], axis=1)

l2 = hierarchy.linkage(X, 'ward')
hierarchy.dendrogram(l2)
plt.show()
```



After normalizing longitude, latitude and price, data are displaying in more hierarchical clusters in the plot of dendrogram. There are more clusters and hierarchical clusters are more clearly than before.

1.2 Exercise 2 (50pts)

Re-using the dbscan code written in class, reproduce the following animation of the dbscan algorithm

[98]:
```
from IPython.display import Image
Image(filename="dbscan_ep.gif", width=500, height=500)
```

[98]:
```
<IPython.core.display.Image object>
```

Hints:

- First animate the dbscan algorithm for the dataset used in class (before trying to create the above dataset)
- Take a snapshot of the assignments when the point gets assigned to a cluster
- Confirm that the snapshot works by saving it to a file
- Don't forget to close the matplotlib plot after saving the figure
- Gather the snapshots in a list of images that you can then save as a gif using the code below
- Use `ax.set_aspect('equal')` so that the circles don't appear to be oval shaped
- To create the above dataset you need two blobs for the eyes. For the mouth you can use the following process to generate (x, y) pairs:
  - Pick an x at random in an interval that makes sense given where the eyes are positioned
  - For that x generate y that is 0.2 * x^2 plus a small amount of randomness
  - `zip` the x's and y's together and append them to the dataset containing the blobs

[38]:
```
import numpy as np
from PIL import Image as im
import matplotlib.pyplot as plt
import sklearn.datasets as datasets

TEMPFILE = 'temp.png'

class DBC():

    def __init__(self, dataset, min_pts, epsilon):
        self.dataset = dataset
        self.min_pts = min_pts
        self.epsilon = epsilon
        self.assignments = [0 for _ in range(len(dataset))]
        self.snaps = []


    def snapshot(self, next_candidate):
        fig, ax = plt.subplots()
        colors = np.array([x for x in 'bgrcmykbgrcmykbgrcmykbgrcmyk'])

        ax.scatter(self.dataset[:,0], self.dataset[:,1], s=10, alpha=0.8,␣
↪c=colors[self.assignments].tolist())
        center = (self.dataset[next_candidate][0], self.
↪dataset[next_candidate][1])
        cir = plt.Circle(center, radius=0.08, color = 'black', fill=False) #␣
↪create circle around the point
        ax.add_patch(cir)
```

```
        ax.set_xlim(-2.9, 2.9)
        ax.set_ylim(-1.1, 3.4)
        ax.set_aspect('equal') # necessary or else the circles appear to be␣
↪oval shaped

        fig.savefig(TEMPFILE)
        plt.close()

        return im.fromarray(np.asarray(im.open(TEMPFILE)))


    def dbscan(self):
        cluster_num = 1
        for i in range(len(self.dataset)):
            if self.assignments[i] != 0:
                # already assigned to a cluster - no need to re-evaluate
                continue
            if self.is_core(i):
                self.dfs_assign(i, cluster_num)
            cluster_num += 1
        return self.assignments

    def dfs_assign(self, i, cluster_num):
        self.assignments[i] = cluster_num
        neighbors = self.get_unlabeled_neighbors(i) # return a list of indexes ␣
↪
        while neighbors:
            next_candidate = neighbors.pop()
            if self.assignments[next_candidate] != 0:
                continue
            self.assignments[next_candidate] = cluster_num
            self.snaps.append(self.snapshot(next_candidate))
            if self.is_core(next_candidate):
                neighbors += self.get_unlabeled_neighbors(next_candidate)
        return

    def is_core(self, i):
        neighbors = []
        for j in range(len(self.dataset)):
            if i != j and np.linalg.norm(self.dataset[i] - self.dataset[j]) <=␣
↪self.epsilon:
                neighbors.append(j)
        return len(neighbors) >= self.min_pts


    def get_unlabeled_neighbors(self, i):
        neighbors = []
```

```
        for j in range (len(self.dataset)):
            if i != j and self.assignments[j] == 0 and np.linalg.norm(self.
↪dataset[i] - self.dataset[j]) <= self.epsilon:
                neighbors.append(j)
        return neighbors

centers = [(-1,2),(1,2)]
eyes, _ = datasets.make_blobs(n_samples=300,centers=centers, cluster_std=0.3)
mouth_x = [(4*np.random.random())-2 for _ in range(300)]
mouth_y = [0.2 * x * x + 0.1 * np.random.randn() - 0.05 for x in mouth_x]
mouth = list(zip(mouth_x, mouth_y))
face = np.append(eyes, mouth, axis=0)
dbc = DBC(face, 3, 0.2)
clustering = dbc.dbscan()


dbc.snaps[0].save(
    'dbscan.gif',
    optimize=False,
    save_all=True,
    append_images=dbc.snaps[1:],
    loop=0,
    duration=25
)
```

[39]:
```
Image(filename="dbscan.gif", width=500, height=500)
```

[39]:
```
<IPython.core.display.Image object>
```

[ ]:
```

```