Homework 2 (100 Points)

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

Exercise 1 - (50 points)

This exercise will be using the AirBnB dataset for New York City called `listings.csv`. You should find this data in your downloaded repository. If not, it is a resource under Piazza.

a) Produce a Marker Cluster using the Folium and Selenium package (you can install them using pip) of the mean listing price per location (lattitude and longitude) over the New York City map. (5 points)

To start, generate a base map of New York City to plot over: (**`location=[40.693943, -73.985880], zoom_start = 11`**). Then, generate and save a `PNG` file named `problem1a.png`. Display it in the cell below as well using the `IPython.display` package.

In [ ]:
```
# Do not edit this cell


import pandas as pd
import numpy as np
import folium #install if you haven't already
import selenium #install if you haven't already
from IPython.display import Image #install if you haven't already

def convert_map_to_png(map, filename):
    """
    Method to convert a folium map to a png file by
    saving the map as an html file and then taking a
    screenshot of the html file on the browser.

    map : folium map object
        The map to be converted to a png file
    filename : str, does not include file type
    """
    import os
    import time
    from selenium import webdriver

    html_filename=f'{filename}.html'
    map.save(html_filename)

    tmpurl=f'file://{os.getcwd()}/{html_filename}'
```

```
    try:
        try:
            browser = webdriver.Firefox()
        except:
            browser = webdriver.Chrome()
    except:
        browser = webdriver.Safari()

    browser.get(tmpurl)
    time.sleep(5)
    browser.save_screenshot(f'{filename}.png')
    browser.quit()
    os.remove(html_filename)

    return Image(f'{filename}.png')
```

In [ ]:
```
from folium.plugins import MarkerCluster, FastMarkerCluster #Using either is
import pandas as pd
from IPython.display import Image
# Write your code below! Leave the instantiated variables: it is for your co
nyc_map = folium.Map(location=[40.693943, -73.985880], zoom_start=11)
airbnbdata = pd.read_csv('listings.csv')
mean_prices = airbnbdata.groupby(['latitude', 'longitude'])['price'].mean().
for x in mean_prices.itertuples():
    folium.Marker(location=[x.latitude, x.longitude], popup=f'Mean Price: ${
marker_cluster = MarkerCluster().add_to(nyc_map)
convert_map_to_png(nyc_map, 'problem1a')
display(Image('problem1a.png'))
```

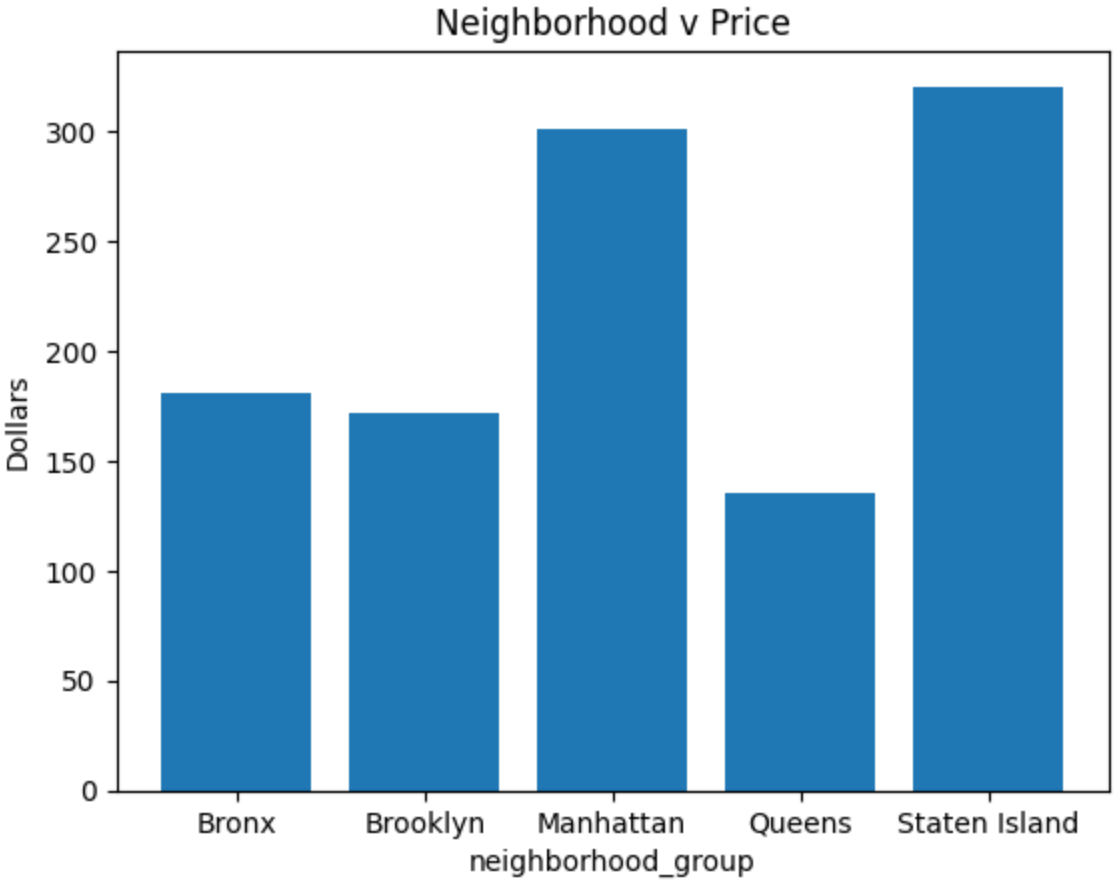
```
/var/folders/rv/hd876kss68148qrgkftj79l80000gn/T/ipykernel_24503/2586671195.
py:6: DtypeWarning: Columns (17) have mixed types. Specify dtype option on i
mport or set low_memory=False.
  airbnbdata = pd.read_csv('listings.csv')
```

b) Plot a bar chart of the average price per neighbourhood group. Briefly comment on the relation between the price and neighbourhood group (use your map to analyze it). - (2.5 pts)

In [ ]:
```
import matplotlib.pyplot as plt
airbnb_data = pd.read_csv('listings.csv')
avg_price = airbnb_data.groupby('neighbourhood_group')['price'].mean().reset
plt.bar(avg_price['neighbourhood_group'], avg_price['price'])
plt.title('Neighborhood v Price')
plt.xlabel('neighborhood_group')
plt.ylabel('Dollars')
```

```
/var/folders/rv/hd876kss68148qrgkftj79l80000gn/T/ipykernel_24503/4072783404.
py:2: DtypeWarning: Columns (17) have mixed types. Specify dtype option on i
mport or set low_memory=False.
  airbnb_data = pd.read_csv('listings.csv')
```

Out[ ]:
```
Text(0, 0.5, 'Dollars')
```



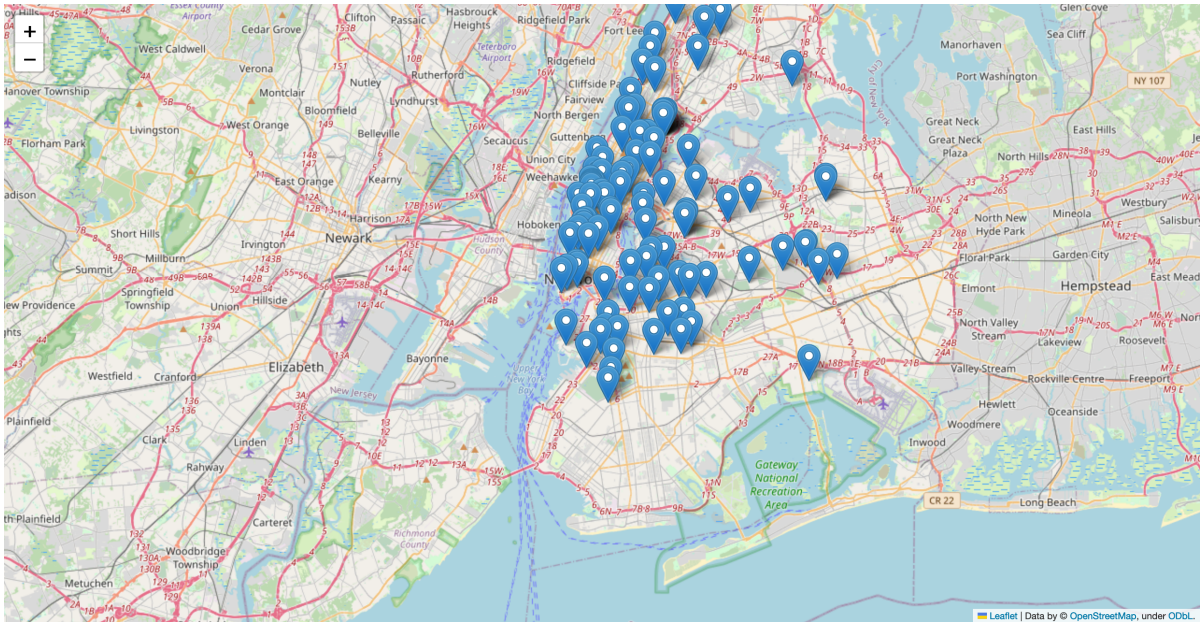
-> Manhattan and Staten Island are the most expensive in terms of average price. Queens is the least expensive in terms of average price. The Bronx and Brooklyn are similar in average price.

c) You're going to be living in New York City long term so you'd like to find places you can stay that are at minimum 300 days (inclusive). Plot a map that displays all the locations of these places. (Note: some could be in the same location) - (5 pts)

In [ ]:
```
# Write your code below! Leave the instantiated variables: it is for your co
air_bnb_df = pd.read_csv('listings.csv')
minimum300 = air_bnb_df[air_bnb_df['minimum_nights'] >= 300]
nyc_map_2 = folium.Map(location=[40.693943, -73.985880], zoom_start=11)
for y,x in minimum300.iterrows():
    folium.Marker(location=[x['latitude'], x['longitude']], popup=f'Listing



convert_map_to_png(nyc_map_2, 'problem1c')
```

```
/var/folders/rv/hd876kss68148qrgkftj79l80000gn/T/ipykernel_24503/3696803702.
py:2: DtypeWarning: Columns (17) have mixed types. Specify dtype option on i
mport or set low_memory=False.
  air_bnb_df = pd.read_csv('listings.csv')
```

Out[ ]:



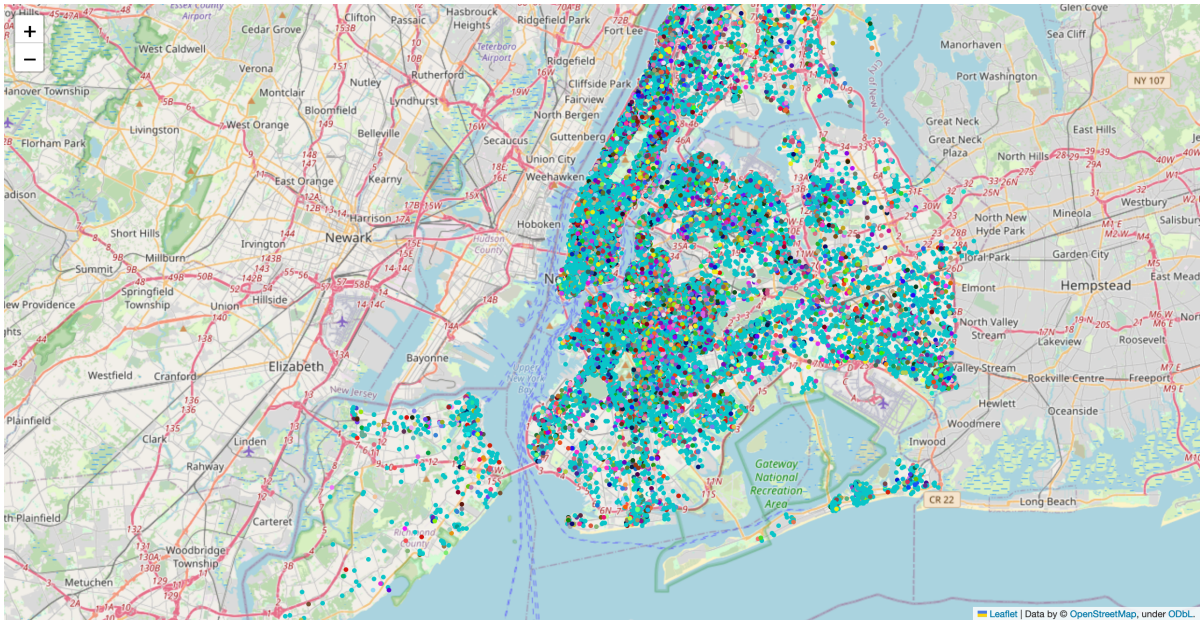
d) Using `longitude`, `latitude`, `price`, and `number_of_reviews`, use Density-based clustering to create clusters. Plot the points on the NYC map in a color corresponding to their cluster (color could be randomly assigned, but ensure each datapoint is colored to its associated cluster). For using `DBSCAN`, have the settings **`eps=0.3, min_samples=10`**. Use a `CircleMarker` with `radius=1`. Plot the clusters on the map and print the number of clusters made. - (15 pts)

In [ ]:

```
from sklearn.cluster import DBSCAN
import random
import folium

df = pd.read_csv('listings.csv')
new_df = df[['latitude', 'longitude', 'price', 'number_of_reviews']]
new_df = new_df.apply(pd.to_numeric)
new_df = new_df.dropna()
min = 10
epsilon = 0.3
dbscan = DBSCAN(eps=epsilon, min_samples=min)
clusters = dbscan.fit_predict(new_df)
nyc_map_3 = folium.Map(location=[40.693943, -73.985880], zoom_start=11)
random_colors = ['#' + ''.join(random.choice('0123456789ABCDEF') for j in ra
for latitude, longitude, cluster in zip(new_df['latitude'], new_df['longituc
    folium.CircleMarker(
        location=[latitude, longitude],
        radius=1,
        color=random_colors[cluster],
    ).add_to(nyc_map_3)

convert_map_to_png(nyc_map_3, 'problem1d')
```

```
/var/folders/rv/hd876kss68148qrgkftj79l80000gn/T/ipykernel_26686/2344116520.
py:5: DtypeWarning: Columns (17) have mixed types. Specify dtype option on i
mport or set low_memory=False.
  df = pd.read_csv('listings.csv')
```

Out[ ]:



e) What would happen if you were to increase/decrease `eps`, and what would happen if you were to increase/decrease `min_samples`? Give some examples when running part d (you don't have to give the map image, just say something such as "When testing part d with ... ") - (5 points)

If epsilon were to increase, then points farther away from the center point is considered part of the cluster, and vice versa for if epsilon were to decrease. If min_samples were to increase, then more samples are required within the radius of epsilon to create a center point while a smaller min_samples means that less points are required within the radius of epsilon to form a center point.

f) For part d, were the clusters seemed to be scattered or grouped together? Justify your answer. - (2.5 points)

The clusters seem to be scattered. As shown in the graph, there seems to be a lot of colors scattered everywhere rather than in groups.

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. Describe your findings. - (10 points)

In [ ]:

```
import matplotlib.pyplot as plt
import pandas as pd
from scipy.cluster import hierarchy
dfair = pd.read_csv('listings.csv')
shared_room_listings = dfair[dfair['room_type'] == 'Shared room']
criteria = shared_room_listings[['longitude', 'latitude', 'price']]

hierarchy.dendrogram(hierarchy.linkage(criteria, method='ward'))
```

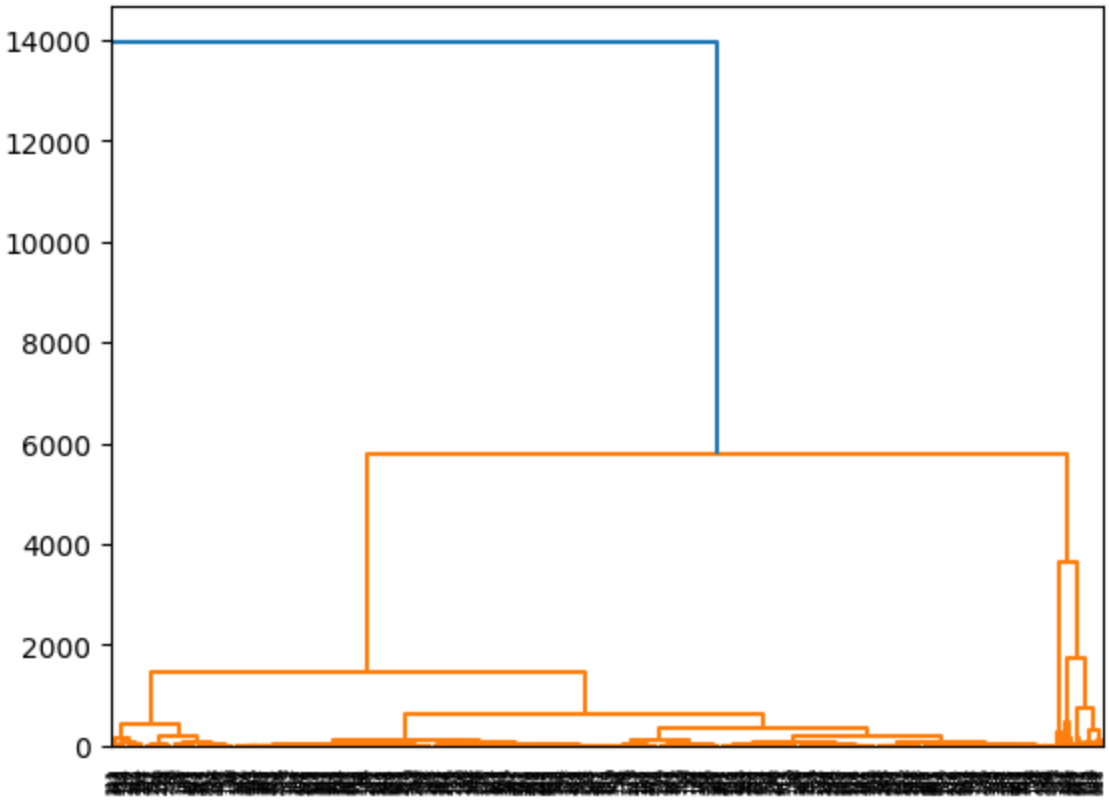
```
/var/folders/rv/hd876kss68148qrgkftj79l80000gn/T/ipykernel_26686/2196097151.
py:4: DtypeWarning: Columns (17) have mixed types. Specify dtype option on i
mport or set low_memory=False.
  dfair = pd.read_csv('listings.csv')
```

Out[ ]:

```
{'icoord': [[35.0, 35.0, 45.0, 45.0],
  [25.0, 25.0, 40.0, 40.0],
  [15.0, 15.0, 32.5, 32.5],
  [65.0, 65.0, 75.0, 75.0],
  [55.0, 55.0, 70.0, 70.0],
  [85.0, 85.0, 95.0, 95.0],
  [145.0, 145.0, 155.0, 155.0],
  [135.0, 135.0, 150.0, 150.0],
  [125.0, 125.0, 142.5, 142.5],
  [115.0, 115.0, 133.75, 133.75],
  [105.0, 105.0, 124.375, 124.375],
  [90.0, 90.0, 114.6875, 114.6875],
  [165.0, 165.0, 175.0, 175.0],
  [102.34375, 102.34375, 170.0, 170.0],
  [62.5, 62.5, 136.171875, 136.171875],
  [23.75, 23.75, 99.3359375, 99.3359375],
  [185.0, 185.0, 195.0, 195.0],
  [235.0, 235.0, 245.0, 245.0],
  [255.0, 255.0, 265.0, 265.0],
  [240.0, 240.0, 260.0, 260.0],
  [225.0, 225.0, 250.0, 250.0],
  [215.0, 215.0, 237.5, 237.5],
  [205.0, 205.0, 226.25, 226.25],
  [275.0, 275.0, 285.0, 285.0],
  [215.625, 215.625, 280.0, 280.0],
  [190.0, 190.0, 247.8125, 247.8125],
  [295.0, 295.0, 305.0, 305.0],
  [315.0, 315.0, 325.0, 325.0],
  [300.0, 300.0, 320.0, 320.0],
  [218.90625, 218.90625, 310.0, 310.0],
  [355.0, 355.0, 365.0, 365.0],
  [405.0, 405.0, 415.0, 415.0],
  [395.0, 395.0, 410.0, 410.0],
  [385.0, 385.0, 402.5, 402.5],
  [375.0, 375.0, 393.75, 393.75],
  [360.0, 360.0, 384.375, 384.375],
  [345.0, 345.0, 372.1875, 372.1875],
  [335.0, 335.0, 358.59375, 358.59375],
  [425.0, 425.0, 435.0, 435.0],
  [445.0, 445.0, 455.0, 455.0],
  [430.0, 430.0, 450.0, 450.0],
  [346.796875, 346.796875, 440.0, 440.0],
  [475.0, 475.0, 485.0, 485.0],
  [465.0, 465.0, 480.0, 480.0],
  [515.0, 515.0, 525.0, 525.0],
  [505.0, 505.0, 520.0, 520.0],
  [495.0, 495.0, 512.5, 512.5],
  [535.0, 535.0, 545.0, 545.0],
  [503.75, 503.75, 540.0, 540.0],
  [472.5, 472.5, 521.875, 521.875],
  [565.0, 565.0, 575.0, 575.0],
  [555.0, 555.0, 570.0, 570.0],
  [595.0, 595.0, 605.0, 605.0],
  [585.0, 585.0, 600.0, 600.0],
  [615.0, 615.0, 625.0, 625.0],
  [645.0, 645.0, 655.0, 655.0],
```

```
'C1',
'C1',
'C1',
'C1',
'C1',
'C1',
'C1',
'C1',
'C1',
'C1',
'C1',
'C1',
'C1',
'C1',
'C1',
'C1',
'C1',
'C1',
'C1',
'C1',
'C1',
'C1',
'C1',
'C1',
'C1',
'C1',
'C1',
'C1',
'C1',
'C1',
'C1',
'C1',
'C1',
'C1',
'C1',
'C1',
'C1',
'C1',
'C1',
'C1',
'C1',
'C1',
'C1',
'C1',
'C1',
'C1',
'C1',
'C1',
'C1',
'C1',
'C1',
'C1',
'C1',
'C1',
'C1',
'C1',
```

```
 'C1',
 'C1',
 'C1',
 'C1',
 'C1',
 'C1',
 'C1',
 'C1',
 'C1',
 'C1',
 'C1',
 'C1',
 'C1',
 'C1',
 'C1',
 'C1',
 'C1',
 'C1',
 'C1',
 'C1',
 'C1',
 'C1',
 'C1',
 'C1',
 'C1',
 'C1',
 'C1',
 'C1',
 'C1',
 'C1']}
```



The graph shows hows the data is grouped into clusters. Clusters that are closer to the root contain more points. The clusters that are farther from the root contain less points, and are more similar to each other. The height of the branches is the dissimilarity between the different clusters. Every leaf at the bottom of the graph is a single shared room data point.

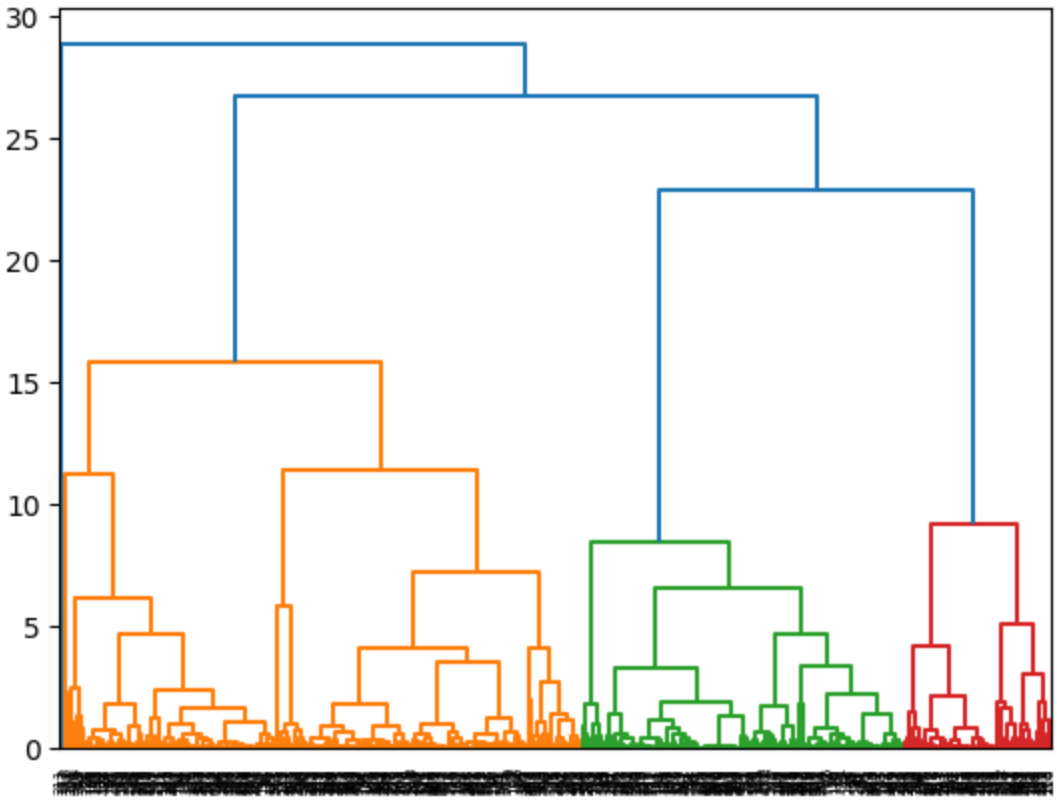
h) Normalize `longitude`, `latitude`, and `price` by subtracting by the mean (of the column) and dividing by the standard deviation (of the column). Repeat g) using the normalized data. Comment on what you observe. - (5 points)

In [ ]:
```
import matplotlib.pyplot as plt
import pandas as pd
from scipy.cluster import hierarchy
dfair = pd.read_csv('listings.csv')
shared_room_listings = dfair[dfair['room_type'] == 'Shared room']
criteria = shared_room_listings[['longitude', 'latitude', 'price']]
normalized = (criteria - criteria.mean()) / criteria.std()
hierarchy.dendrogram(hierarchy.linkage(normalized, method='ward'))
```

```
/var/folders/rv/hd876kss68148qrgkftj79l80000gn/T/ipykernel_26686/1688190215.
py:4: DtypeWarning: Columns (17) have mixed types. Specify dtype option on i
mport or set low_memory=False.
  dfair = pd.read_csv('listings.csv')
```

```
'C2',
'C2',
'C2',
'C2',
'C3',
'C3',
'C3',
'C3',
'C3',
'C3',
'C3',
'C3',
'C3',
'C3',
'C3',
'C3',
'C3',
'C3',
'C3',
'C3',
'C3',
'C3',
'C3',
'C3',
'C3',
'C3',
'C3',
'C3',
'C3',
'C3',
'C3',
'C3',
'C3',
'C3',
'C3',
'C3',
'C3',
'C3',
'C3',
'C3',
'C3',
'C3',
'C3',
'C3',
'C3',
'C3',
'C3',
'C3',
'C3',
'C3',
'C3',
'C3',
'C3',
'C3',
'C3',
'C3',
```

```
 'C3',
 'C3',
 'C3',
 'C3',
 'C3',
 'C3',
 'C3',
 'C3',
 'C3',
 'C3',
 'C3',
 'C3',
 'C3',
 'C3',
 'C3',
 'C3',
 'C3',
 'C3',
 'C3',
 'C3',
 'C3',
 'C3',
 'C3',
 'C3',
 'C3',
 'C3']}
```



The normalization of the data creates more children from the roots. This results in better and more clusters.

Exercise 2 (50 points)

a) Fetch the "mnist_784" data and store is as a `.csv` (that way you don't have to fetch it every time - which takes about 30s). (2.5 points)

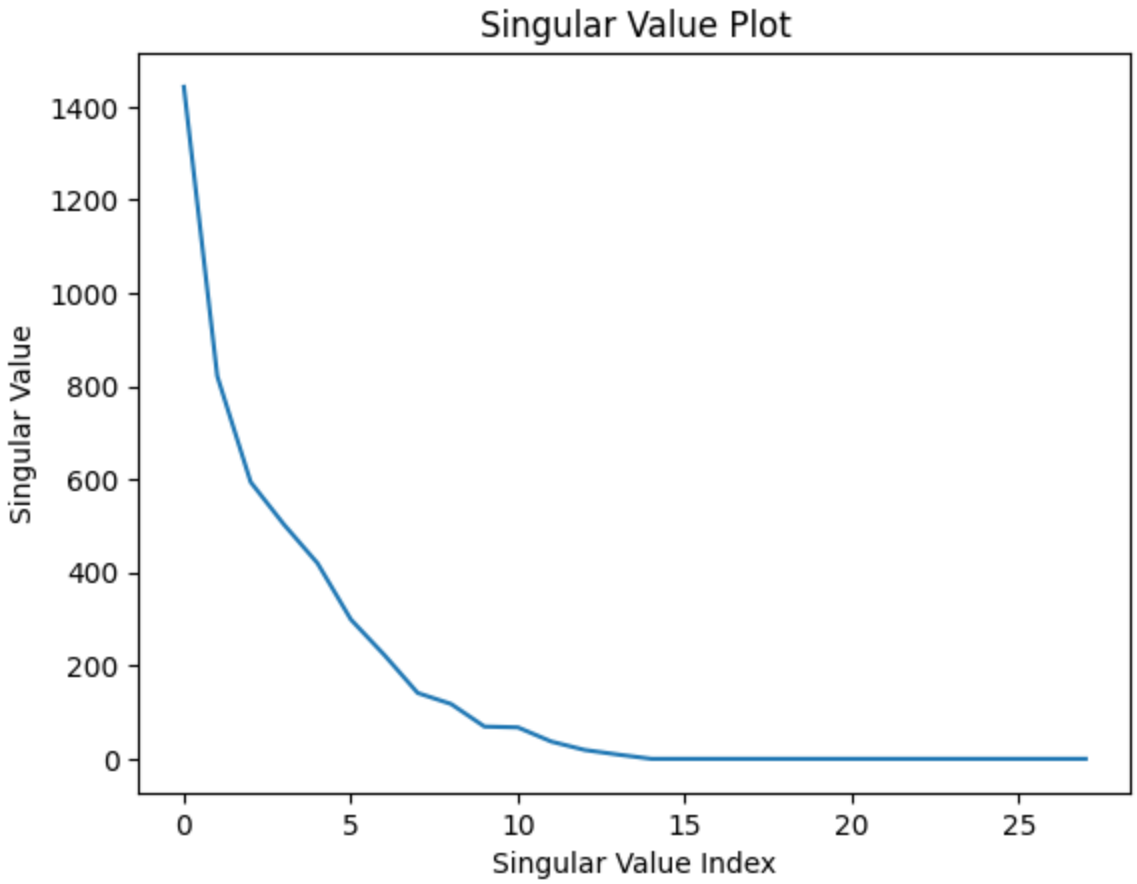
In [ ]:
```
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.datasets import fetch_openml

X, y = fetch_openml(name="mnist_784", version=1, return_X_y=True, as_frame=F

# your code here
mnist_784_df = pd.DataFrame(X, columns=[f"pixel{j}" for j in range(X.shape[1
mnist_784_df["label"] = y
mnist_784_df.to_csv("mnist_784_data.csv")
```

```
/Users/kelvin/Library/Python/3.9/lib/python/site-packages/sklearn/datasets/_
openml.py:1022: FutureWarning: The default value of `parser` will change fro
m `'liac-arff'` to `'auto'` in 1.4. You can set `parser='auto'` to silence t
his warning. Therefore, an `ImportError` will be raised from 1.4 if the data
set is dense and pandas is not installed. Note that the pandas parser may re
turn different data types. See the Notes Section in fetch_openml's API doc f
or details.
  warn(
```

b) Plot the singular value plot for a single example of the 9 digit (2.5 points)

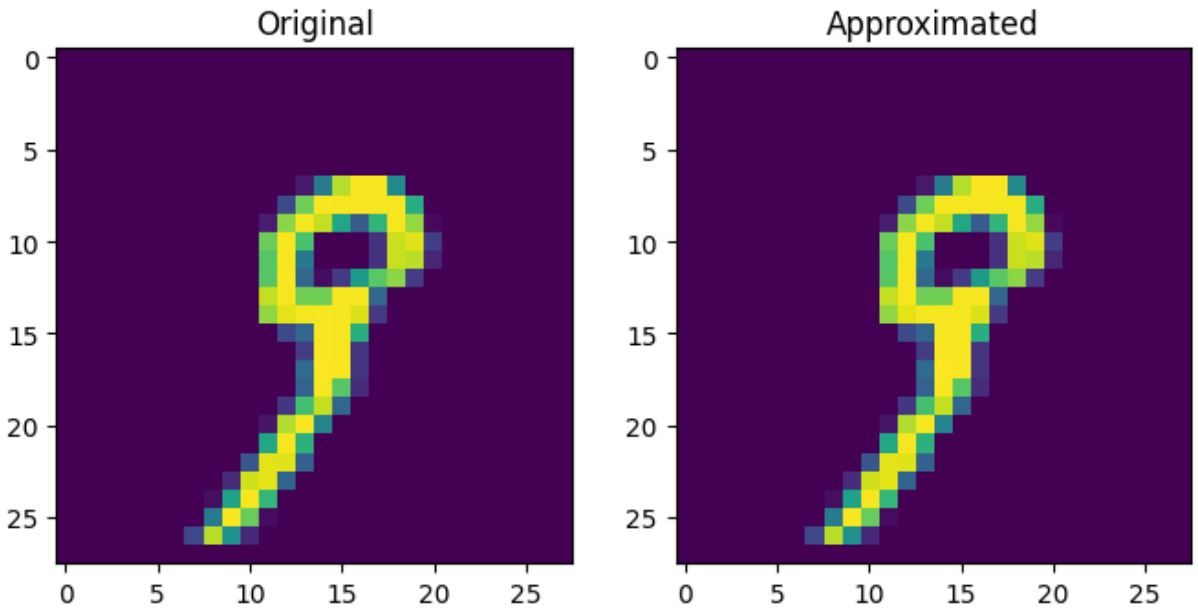
In [ ]:
```
import numpy as np
digit_9 = X[y == '9']
ex = digit_9[1]
y, l, z = np.linalg.svd(ex.reshape(28, 28))
plt.plot(l)
plt.title("Singular Value Plot")
plt.xlabel("Singular Value Index")
plt.ylabel("Singular Value")
```

Out[ ]:
```
Text(0, 0.5, 'Singular Value')
```



c) Just like we did in class with the image of the boat: By setting some singular values to 0, plot the approximation of an image of a 9 digit next to the original image. (5 points)

In [ ]:
```
import numpy as np
num_zero = 100
digit_9 = X[y == '9']
ex = digit_9[1]
y, l, z = np.linalg.svd(ex.reshape(28, 28))
copy = l.copy()
copy[num_zero:] = 0
new = y.dot(np.diag(copy)).dot(z)

plt.figure(figsize=(8, 4))
plt.subplot(1, 2, 1)
plt.imshow(ex.reshape(28, 28))
plt.title("Original")

plt.subplot(1, 2, 2)
plt.imshow(new)
plt.title("Approximated")
plt.show()
```



d) Consider the entire dataset as a matrix. Perform SVD and explain why / how you chose a particular rank. Note: you may not be able to run this on the entire dataset in a reasonable amount of time so you may take a small random sample for this and the following questions. (5 points)

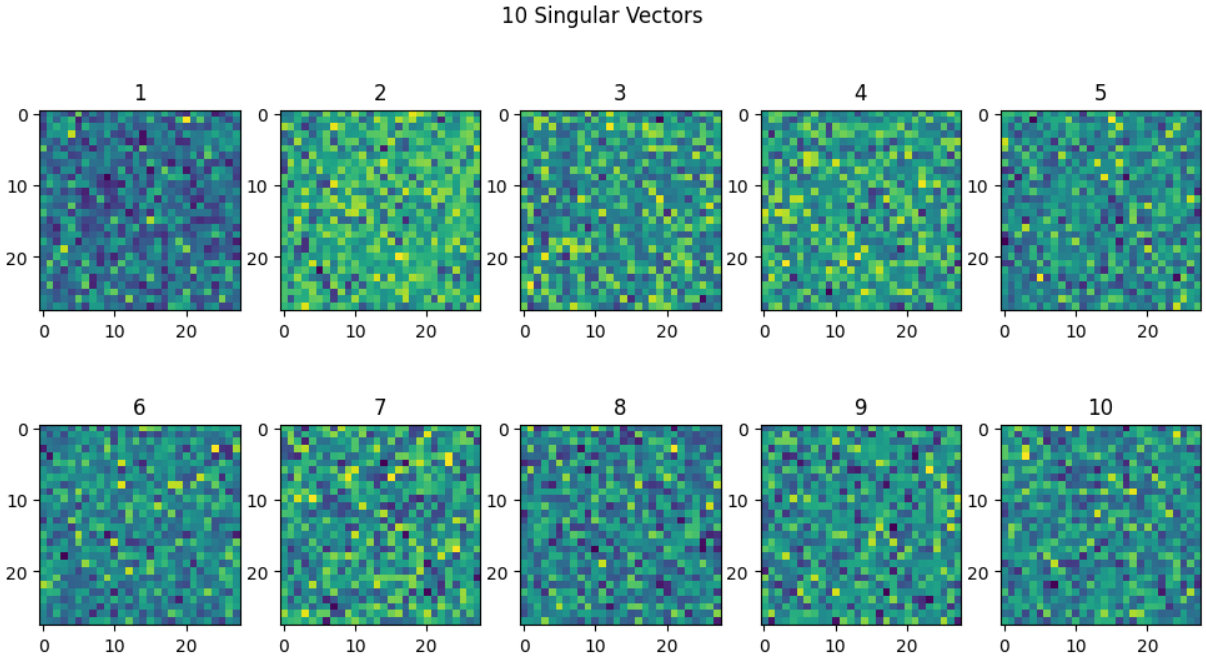
In [ ]:
```
from sklearn.datasets import fetch_openml
import numpy as np

X, y = fetch_openml(name="mnist_784", version=1, return_X_y=True, as_frame=F
sample_size = 784
random_indices = np.random.choice(X.shape[0], sample_size, replace=False)
sampled_data = X[random_indices, :]
U, s, VT = np.linalg.svd(sampled_data, full_matrices=False)
explained_variance = np.cumsum(s ** 2) / np.sum(s ** 2)
desired_explained_variance = 0.95
rank = np.argmax(explained_variance >= desired_explained_variance) + 1
```

```
/Users/kelvin/Library/Python/3.9/lib/python/site-packages/sklearn/datasets/_
openml.py:1022: FutureWarning: The default value of `parser` will change fro
m `'liac-arff'` to `'auto'` in 1.4. You can set `parser='auto'` to silence t
his warning. Therefore, an `ImportError` will be raised from 1.4 if the data
set is dense and pandas is not installed. Note that the pandas parser may re
turn different data types. See the Notes Section in fetch_openml's API doc f
or details.
  warn(
```

e) Plot the first 10 singular vectors. Notice that each singular vector's length will be 784 so you can plot them as a 28x28 image. (5points)

In [ ]:
```
import matplotlib.pyplot as plt

fig, axes = plt.subplots(2, 5, figsize=(12, 6))
fig.suptitle("10 Singular Vectors")
```

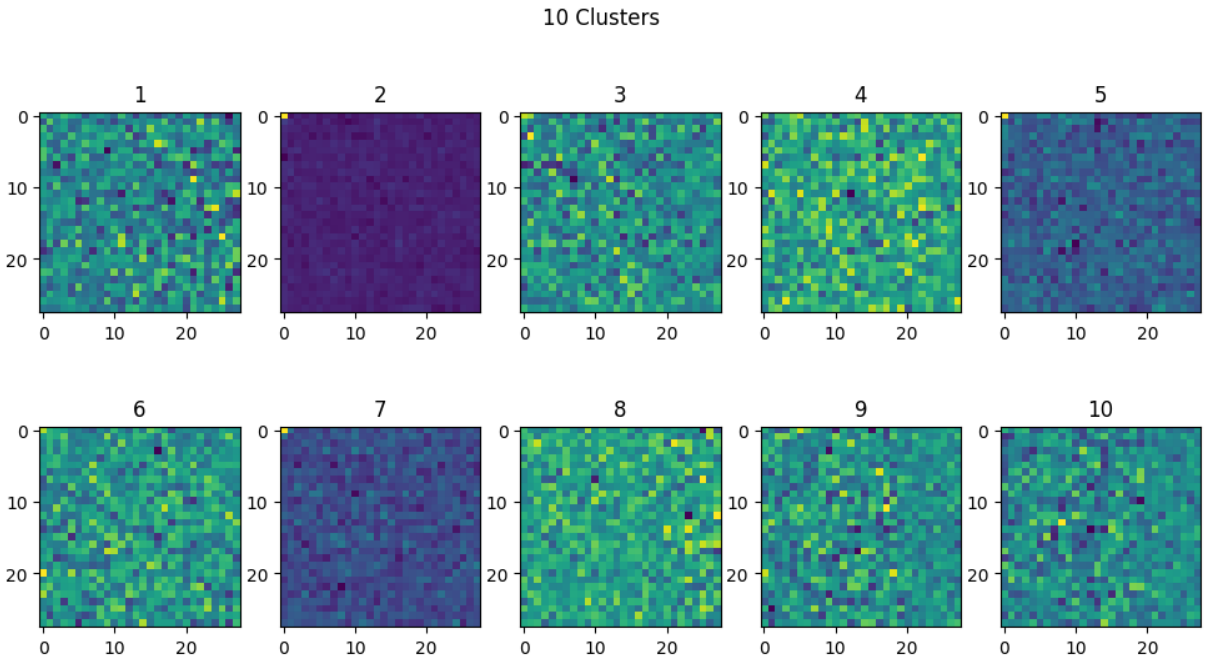
```
for x in range(10):
    single_vector = U[:, x]
    single_vector_image = single_vector.reshape(28, 28).T
    axis = axes[x // 5, x % 5]
    axis.imshow(single_vector_image)
    axis.set_title (x + 1)

plt.show()
```



f) Using Kmeans on this new dataset, cluster the images from d) using 10 clusters and plot the centroid of each cluster. Note: the centroids should be represented as images. (10 points)

In [ ]:
```
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

kmeans = KMeans(n_clusters=10)
labels = kmeans.fit_predict(U[:,:784])
centroid = kmeans.cluster_centers_
fig, axes = plt.subplots(2, 5, figsize=(12, 6))
fig.suptitle("10 Clusters")
for x in range(10):
    vector = centroid[x,:]
    image = vector.reshape(28,28).T
    axis = axes[x // 5, x % 5]
    axis.imshow(image)
    axis.set_title (x + 1)
plt.show()
```

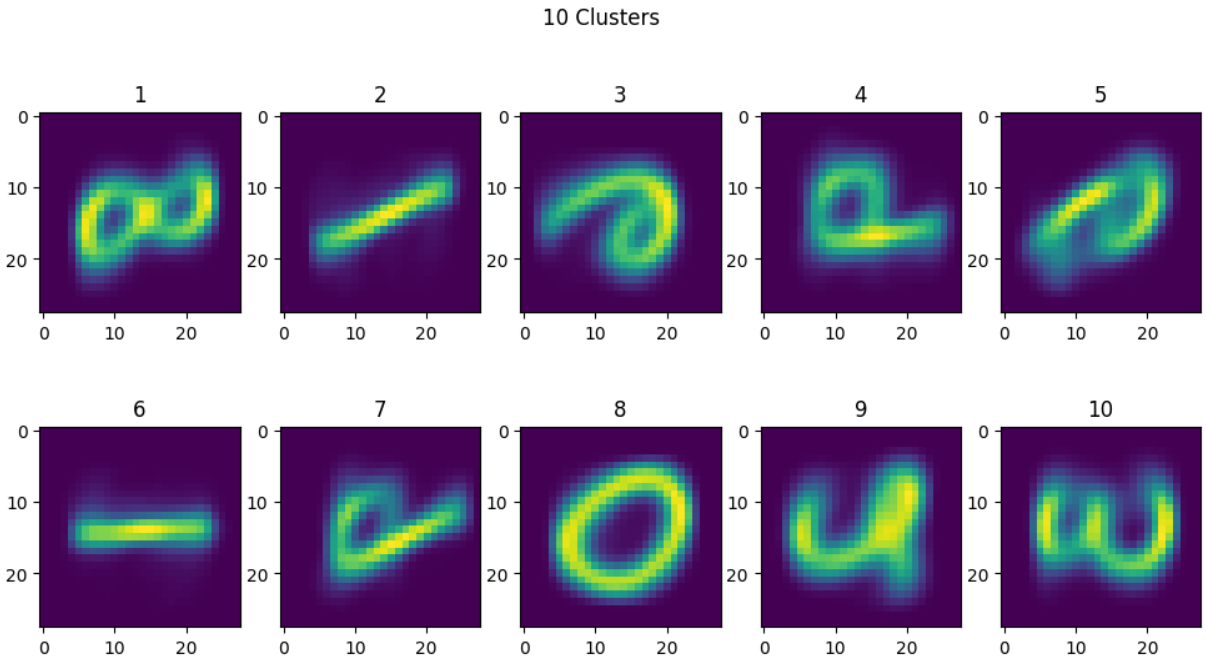
```
/Users/kelvin/Library/Python/3.9/lib/python/site-packages/sklearn/cluster/_k
means.py:1416: FutureWarning: The default value of `n_init` will change from
10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the wa
rning
  super()._check_params_vs_input(X, default_n_init=10)
```



g) Repeat f) on the original dataset (if you used a subset of the dataset, keep using that same subset). Comment on any differences (or lack thereof) you observe between the centroids. (5 points)

In [ ]:
```
from sklearn.datasets import fetch_openml
X, y = fetch_openml(name="mnist_784", version=1, return_X_y=True, as_frame=F
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt


kmeans = KMeans(n_clusters=10)
labels = kmeans.fit_predict(X[:,:784])
centroid = kmeans.cluster_centers_
fig, axes = plt.subplots(2, 5, figsize=(12, 6))
fig.suptitle("10 Clusters")
for x in range(10):
    vector = centroid[x,:]
    image = vector.reshape(28,28).T
    axis = axes[x // 5, x % 5]
    axis.imshow(image)
    axis.set_title (x + 1)
plt.show()
```

```
/Users/kelvin/Library/Python/3.9/lib/python/site-packages/sklearn/datasets/_
openml.py:1022: FutureWarning: The default value of `parser` will change fro
m `'liac-arff'` to `'auto'` in 1.4. You can set `parser='auto'` to silence t
his warning. Therefore, an `ImportError` will be raised from 1.4 if the data
set is dense and pandas is not installed. Note that the pandas parser may re
turn different data types. See the Notes Section in fetch_openml's API doc f
or details.
  warn(
/Users/kelvin/Library/Python/3.9/lib/python/site-packages/sklearn/cluster/_k
means.py:1416: FutureWarning: The default value of `n_init` will change from
10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the wa
rning
  super()._check_params_vs_input(X, default_n_init=10)
```



The difference between the adjusted dataset and the original image. The adjusted images are much more incomprehensible compared to the images produced from the original.

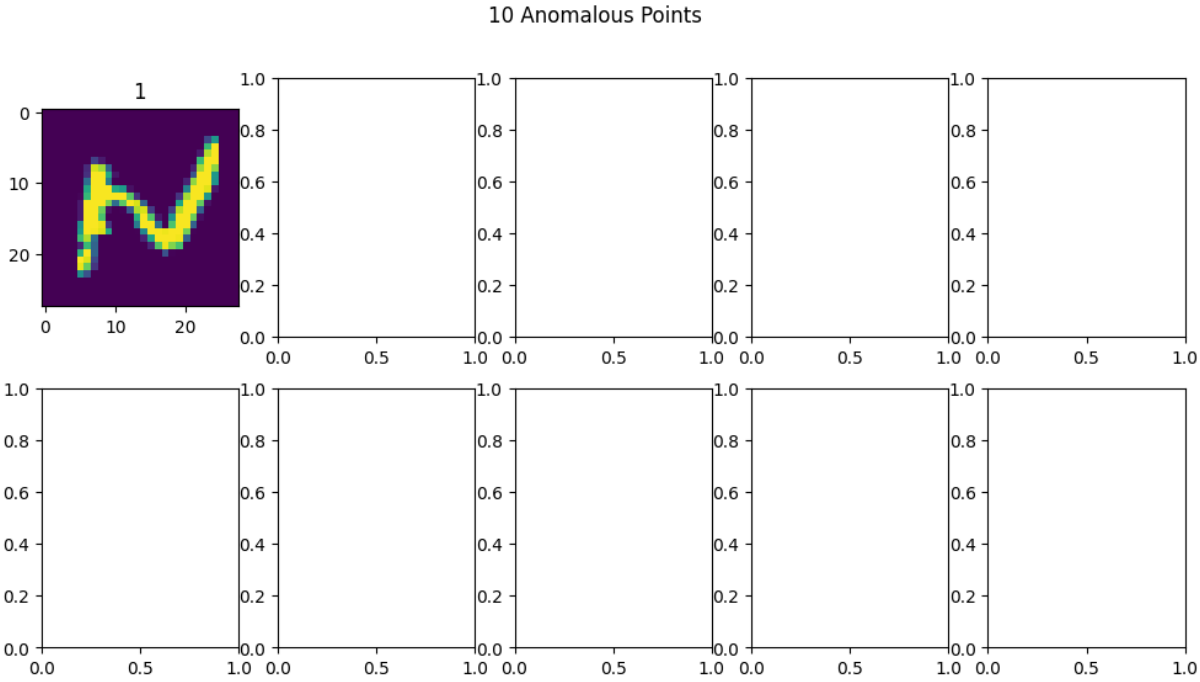
h) Create a matrix (let's call it `0`) that is the difference between the original dataset and the rank-10 approximation of the dataset. (5 points)

In [ ]:
```
from sklearn.datasets import fetch_openml
from sklearn.decomposition import TruncatedSVD

X, y = fetch_openml(name="mnist_784", version=1, return_X_y=True, as_frame=F
rank = 10  # Replace with your desired rank
svd = TruncatedSVD(n_components=rank)
X_reduced = svd.fit_transform(X)
X_approx = svd.inverse_transform(X_reduced)

# Calculate the residual (difference between original and low-rank approxima
0 = X - X_approx
```

```
/Users/kelvin/Library/Python/3.9/lib/python/site-packages/sklearn/datasets/_
openml.py:1022: FutureWarning: The default value of `parser` will change fro
m `'liac-arff'` to `'auto'` in 1.4. You can set `parser='auto'` to silence t
his warning. Therefore, an `ImportError` will be raised from 1.4 if the data
set is dense and pandas is not installed. Note that the pandas parser may re
turn different data types. See the Notes Section in fetch_openml's API doc f
or details.
  warn(
```

i) The largest (using euclidean distance from the origin) rows of the matrix `0` could be considered anomalous data points. Briefly explain why. Plot the 10 images responsible for the 10 largest rows of that matrix `0`. (10 points)

In [ ]:
```
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
import numpy as np

euclidean_distance = np.linalg.norm(0)
farthest_points = np.argsort(euclidean_distance)[-10:]
fig, axes = plt.subplots(2, 5, figsize=(12, 6))
fig.suptitle("10 Anomalous Points")
for x, y in enumerate(farthest_points):
    anomalous_image = X[y, :784]
    anomalous_image =  anomalous_image.reshape(28,28).T
    axis = axes[x // 5, x % 5]
    axis.imshow(anomalous_image)
    axis.set_title (x + 1)
plt.show()
```



Bonus (20pts)

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

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

Out[ ]:
```
<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

In [ ]:
```
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.snaps = []
        self.assignments = [0 for _ in range(len(self.dataset))] #0 means nc

    def snapshot(self):
        fig, ax = plt.subplots()
        colors = np.array([x for x in 'bgrcmykbgrcmykbgrcmykbgrcmyk'])
        colors = np.hstack([colors] * 40)

        ax.scatter(self.dataset[:, 0], self.dataset[:, 1], color=colors[self
```

```
        #ax.set_xlim(...)
        #ax.set_ylim(...)
        ax.set_aspect('equal') # necessary or else the circles appear to be

        fig.savefig(TEMPFILE)
        plt.close()

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

    def get_neighborhood(self,i):
        neighborhood = []
        for j in range(len(self.dataset)):
            if self.distance(i,j) <= self.epsilon and i != j:
                neighborhood.append(j)
        return neighborhood

    def is_core(self, i):
        return len(self.get_neighborhood(i)) >= self.min_pts

    def distance(self,i,j):
        return np.linalg.norm(self.dataset[i] - self.dataset[j])

    def assign(self, i, cluster_num):
        self.assignments[i] = cluster_num
        neighbor_queue = self.get_neighborhood(i)
        while neighbor_queue:
            next_candidate = neighbor_queue.pop()
            if self.assignments[next_candidate] != 0:
                continue
            self.assignments[next_candidate] = cluster_num

            if self.is_core(next_candidate):
                snap = self.snapshot()
                self.snaps.append(snap)
                next_neighborhood = self.get_neighborhood(next_candidate)
                neighbor_queue += [i for i in next_neighborhood if self.assi

        return

    def dbscan(self):
        cluster_num = 1
        for i in range(len(self.dataset)):
            if self.is_core(i) and self.assignments[i] == 0:
                self.assign(i, cluster_num)

                cluster_num += 1
        return self.assignments



centers = [[2,3], [4,3], [3, 1],[2,1], [4,1], [5,1.5], [1,1.5]]
eyes, _ = datasets.make_blobs(n_samples=500, centers=centers, cluster_std=0.
                              random_state=0)

mouth_x =   np.random.uniform(.25,.25,100)
mouth_y = .2 * mouth_x**2
```

```
face = np.append(eyes, np.column_stack((mouth_x, mouth_y)), axis=0)

dbc = DBC(face, min_pts=5, epsilon=0.4)
clustering = dbc.dbscan()

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


from IPython.display import Image
Image(filename="dbscan_face.gif", width=500, height=500)
```

Out[ ]: <IPython.core.display.Image object>