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 (http://insideairbnb.com/get-the-data.html) 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 (https://deparkes.co.uk/2016/06/24/folium-marker-clusters/) 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)

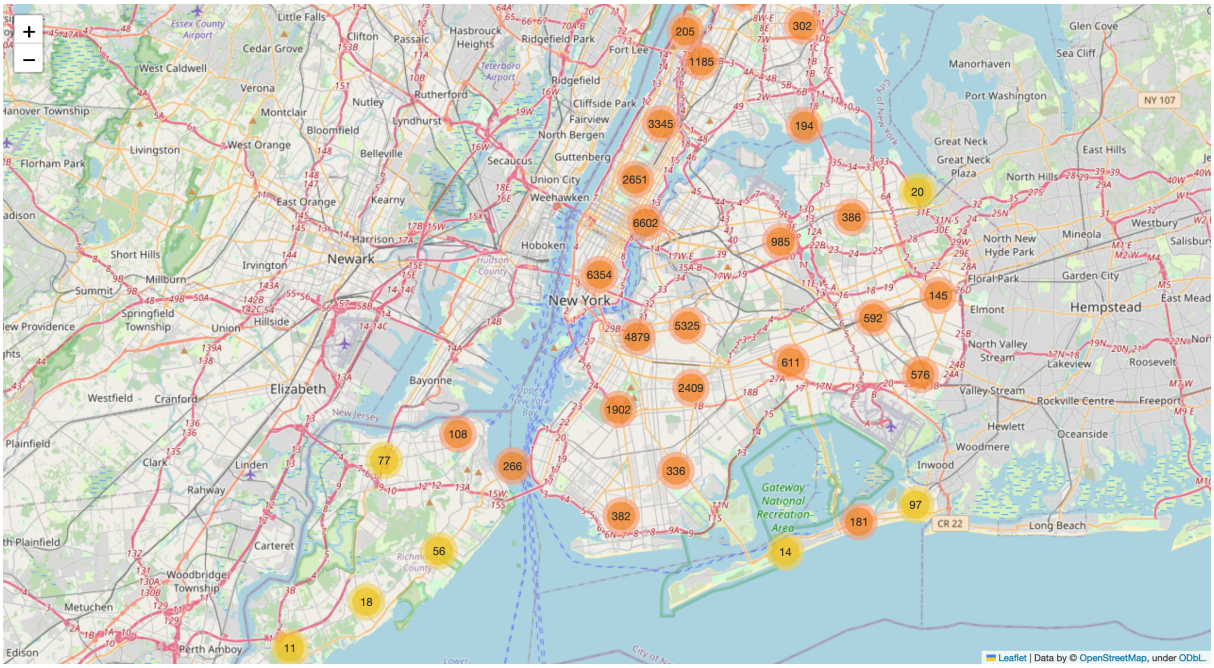
To start, generate a base map of New York City to plot over: (**`location=[40.693943, -73.985880], zoom_start = 11`**). Then, generate and save a `PNG` file named `problem1a.png`. Display it in the cell below as well using the `IPython.display` package.

In [2]:
```
# 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 [3]:
```
from folium.plugins import MarkerCluster, FastMarkerCluster #Using eithe

nyc_map = folium.Map(location=[40.693943, -73.985880], zoom_start=11)

data = pd.read_csv('listings.csv')
mean_price_data = data.groupby(['latitude', 'longitude'])['price'].mean(

marker_cluster = MarkerCluster().add_to(nyc_map)

for _, row in mean_price_data.iterrows():
    popup = row['price']
    folium.Marker([row['latitude'], row['longitude']], popup=popup).add_

convert_map_to_png(nyc_map, 'problem1a')
```

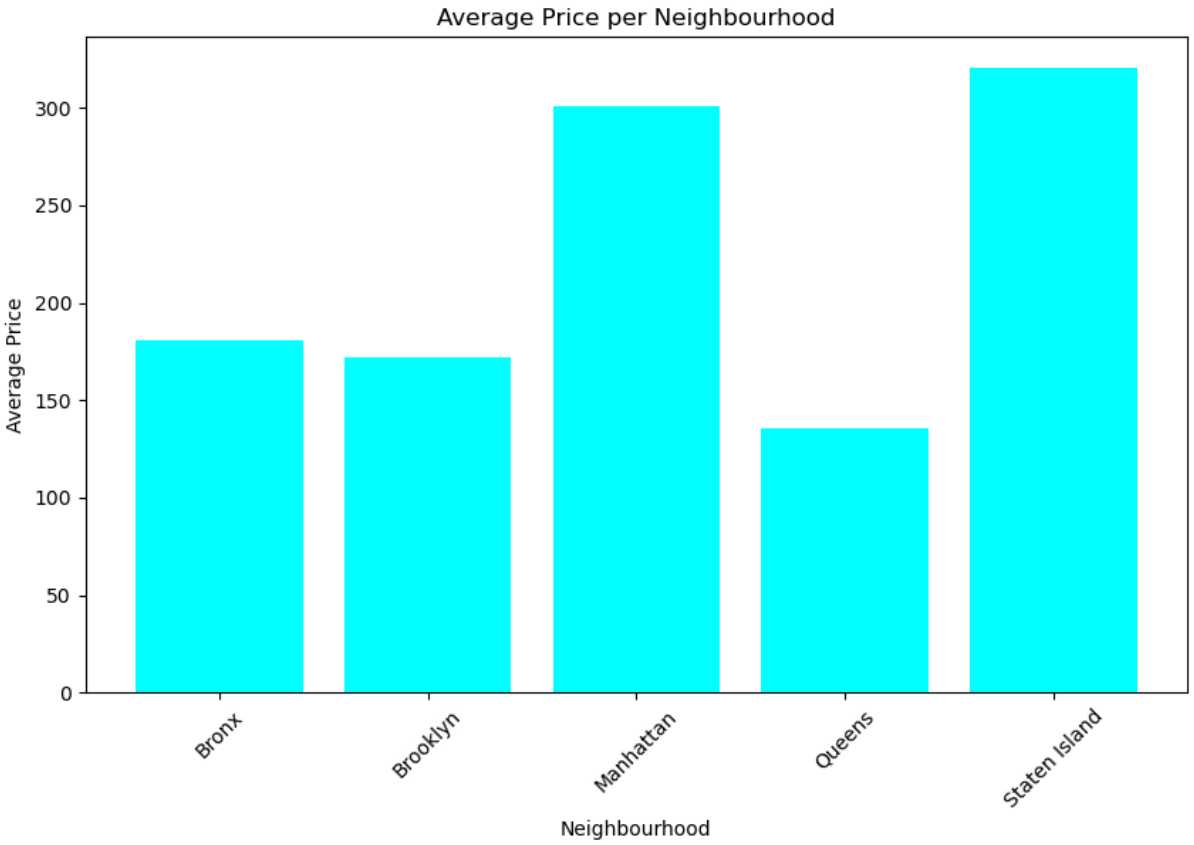
```
/var/folders/bj/hq4hjqs54_95ynh8j4m1yq3w0000gn/T/ipykernel_77279/240658
7223.py:5: DtypeWarning: Columns (17) have mixed types. Specify dtype o
ption on import or set low_memory=False.
  data = pd.read_csv('listings.csv')
```

Out[3]:



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 [4]:
```
import matplotlib.pyplot as plt

average_price_neighborhood = data.groupby('neighbourhood_group')['price']

plt.figure(figsize=(10,6))
plt.bar(average_price_neighborhood['neighbourhood_group'], average_price_
plt.title('Average Price per Neighbourhood')
plt.xlabel('Neighbourhood')
plt.ylabel('Average Price')
plt.xticks(rotation=45)
plt.show()
```



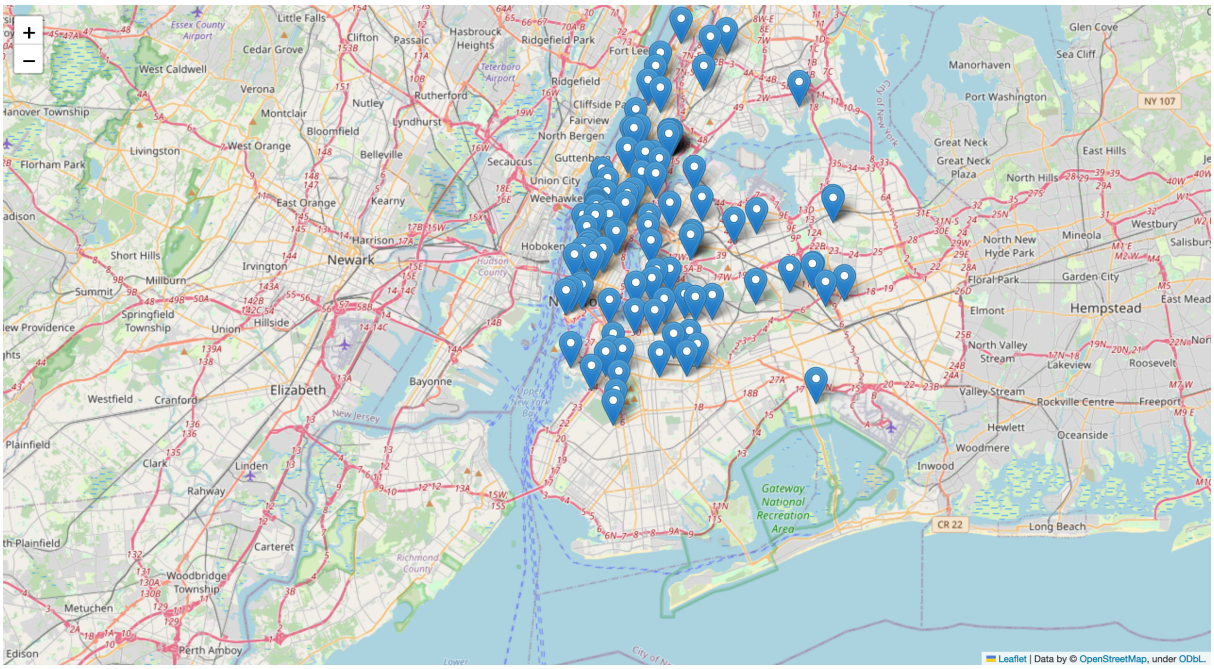
Based on the provided data, I observed that:

1. **Manhattan**: It has the second highest average price among all the neighborhood groups. This suggests that Manhattan tends to have more expensive listings compared to the other boroughs except Staten Island.
2. **Staten Island**: It has the highest average price, indicating that it is generally more expensive than the other boroughs.
3. **Bronx, Brooklyn, and Queens**: These three boroughs have lower average prices compared to Manhattan and Staten Island. Among them, Brooklyn has the highest average price, followed by the Bronx and Queens.
4. **Queens**: It has the lowest average price among the neighborhood groups, indicating that it tends to have more affordable listings on aood group.

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 [5]:
```
nyc_map_2 = folium.Map(location=[40.693943, -73.985880], zoom_start=11)

long_term_stays = data[data['minimum_nights'] >= 300]

for _, row in long_term_stays.iterrows():
    popup = row['name']
    folium.Marker([row['latitude'], row['longitude']], popup=popup).add_

convert_map_to_png(nyc_map_2, 'problem1c')
```

Out[5]:

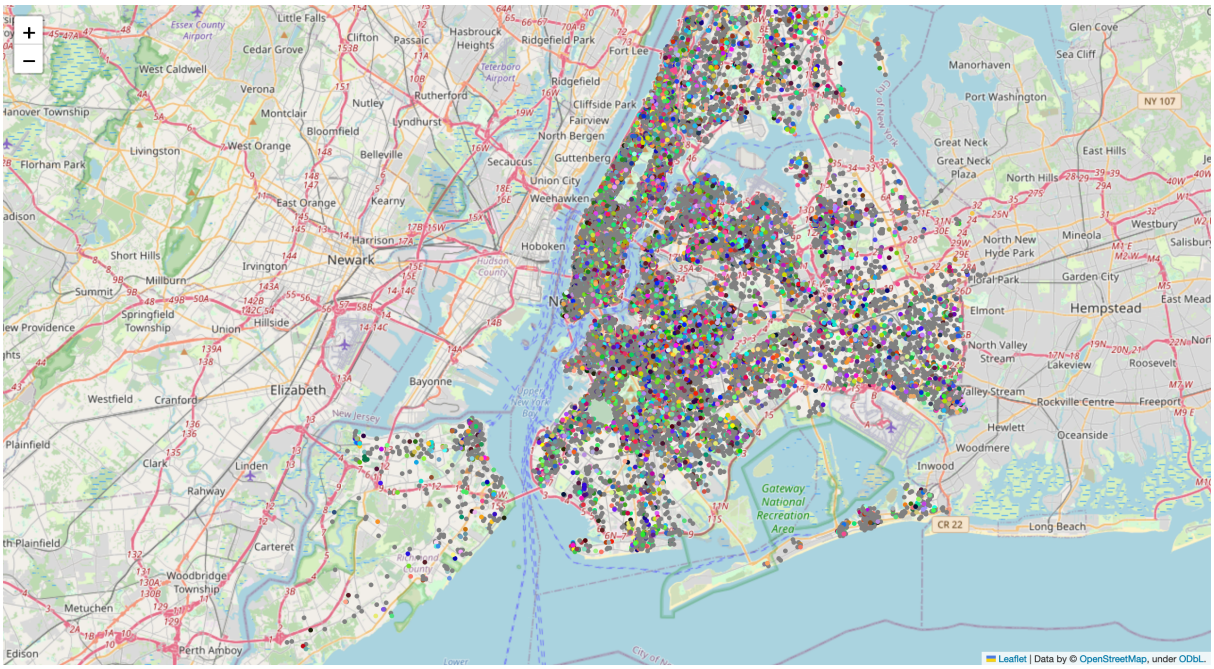


d) Using `longitude`, `latitude`, `price`, and `number_of_reviews`, use Density-based clustering to create clusters. Plot the points on the NYC map in a color corresponding to their cluster (color could be randomly assigned, but ensure each datapoint is colored to its associated cluster). For using `DBSCAN`, 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 [6]:
```
from sklearn.cluster import DBSCAN
import random

# Helper functions to generate random colors
def generate_random_color():
    color = '#{:06x}'.format(random.randint(0, 0xFFFFFF))
    return color

def generate_random_colors(n):
    colors = set()
    while len(colors) < n:
        color = generate_random_color()
        if color != '#808080':  # Exclude the color grey
            colors.add(color)
    return list(colors)

data = pd.read_csv('listings.csv')

relevant_data = data[['longitude', 'latitude', 'price', 'number_of_reviev

# Perform DBSCAN clustering
dbscan = DBSCAN(eps=0.3, min_samples=10)
clusters = dbscan.fit(relevant_data)
data['cluster'] = clusters.labels_

# find number of clusters
if -1 in dbscan.labels_:
    num_clusters = len(np.unique(dbscan.labels_)) - 1
else:
    num_clusters = len(np.unique(dbscan.labels_))

# generate x number of random colors, where x is the number of clusters
random_colors = generate_random_colors(num_clusters)
nyc_map3 = folium.Map(location=[40.693943, -73.985880], zoom_start=11)


# Plot the clusters on the map
for _, row in data.iterrows():
    cluster_label = row['cluster']
    if cluster_label == -1:
        color = '#808080'  # Assign grey color to noise points
    else:
        color = random_colors[cluster_label] # assign a colors for each p
    folium.CircleMarker([row['latitude'], row['longitude']], radius=1, co

# Print the number of clusters
print(f"Number of clusters formed: {num_clusters}")

# Save the map as a PNG file
convert_map_to_png(nyc_map3, 'problem1d')
```

```
/Users/nurassylmedeuov/opt/anaconda3/lib/python3.9/site-packages/scipy/
__init__.py:155: UserWarning: A NumPy version >=1.18.5 and <1.25.0 is r
equired for this version of SciPy (detected version 1.26.0
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversio
n}"
/var/folders/bj/hq4hjqs54_95ynh8j4m1yq3w0000gn/T/ipykernel_77279/261198
8679.py:17: DtypeWarning: Columns (17) have mixed types. Specify dtype
option on import or set low_memory=False.
  data = pd.read_csv('listings.csv')
/Users/nurassylmedeuov/opt/anaconda3/lib/python3.9/site-packages/sklear
n/utils/validation.py:623: FutureWarning: is_sparse is deprecated and w
ill be removed in a future version. Check `isinstance(dtype, pd.SparseD
type)` instead.
  if not hasattr(array, "sparse") and array.dtypes.apply(is_sparse).any
():

Number of clusters formed: 627
```

Out[6]:



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)

1. Increasing `eps`:

- Increasing `eps` expands the neighborhood size for core points, resulting in larger clusters and more points being included in a cluster.
- For instance, when testing part d with `eps=0.9` (larger than the original `eps=0.3`), we might find that more points are included in clusters, and clusters may merge if they are within this expanded neighborhood size. And we got 627 clusters for that.

2. Decreasing `eps`:

- Decreasing `eps` reduces the neighborhood size for core points, potentially leading to smaller, more tightly packed clusters.
- If we test part d with `eps=0.1` (smaller than the original `eps=0.3`), clusters might become more compact. And we got 527 clusters

3. Increasing `min_samples`:

- Increasing `min_samples` requires more number of points within a neighborhood for a point to be considered a core point. This can lead to more conservative clustering, with fewer points being included in each cluster.
- When testing part d with `min_samples=20` (larger than the original `min_samples=10`), we might find that clusters become more tightly defined. It can result in smaller clusters and more noise points. And we got 250 clusters

4. Decreasing `min_samples`:

- Decreasing `min_samples` allows points to be considered core points even with fewer neighbors, which can lead to larger and more loosely defined clusters.
- When testing part d with `min_samples=1` (smaller than the original `min_samples=10`), clusters can become larger and less densely populated, potentially including more points in

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

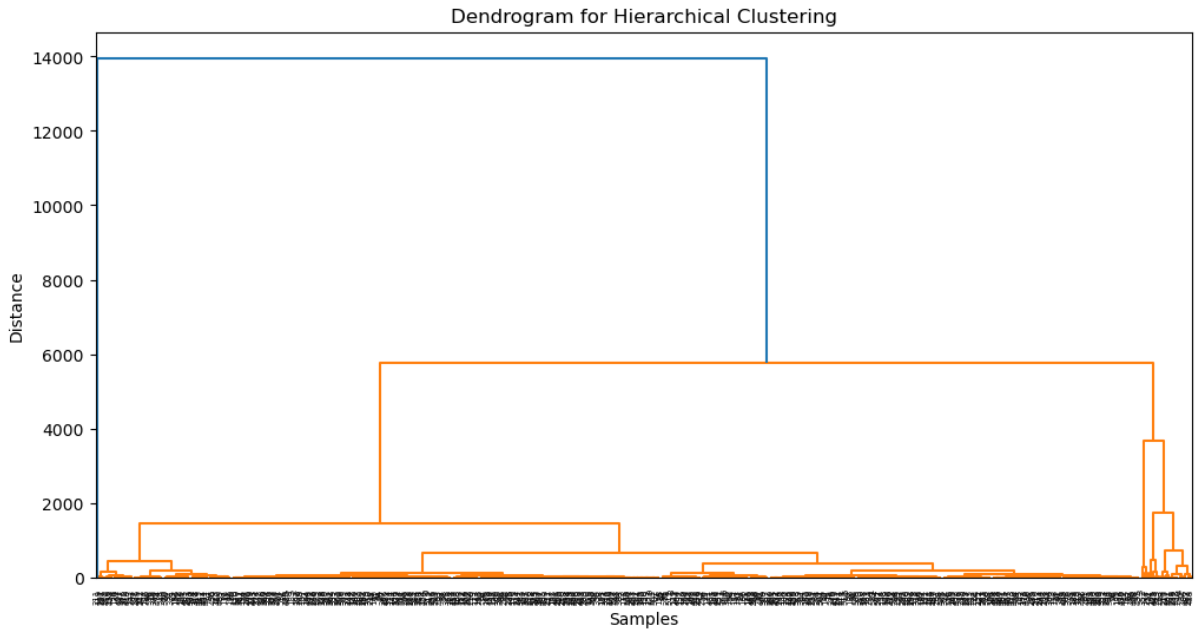
From my observations the clusters seem to be all mixed up together, while the points in the clusters were close together.

- With `eps = 0.3, min_samples=10` on the above map:
- About the overall arrangement of clusters on the map, it seemed that clusters are grouped together on the map.
- However, about the internal structure of individual cluster, it seemed that points within a cluster are scattered around the cluster.

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 [7]:
```
from scipy.cluster import hierarchy

data = pd.read_csv('listings.csv')

shared_room_data = data[data['room_type'] == 'Shared room']


relevant_data = shared_room_data[['longitude', 'latitude', 'price']]

Z = hierarchy.linkage(relevant_data, method='ward', metric='euclidean')

# Plot the dendrogram
plt.figure(figsize=(12, 6))
dn = hierarchy.dendrogram(Z)
plt.title('Dendrogram for Hierarchical Clustering')
plt.xlabel('Samples')
plt.ylabel('Distance')
plt.show()
```

```
/var/folders/bj/hq4hjqs54_95ynh8j4m1yq3w0000gn/T/ipykernel_77279/427137
9522.py:3: DtypeWarning: Columns (17) have mixed types. Specify dtype o
ption on import or set low_memory=False.
  data = pd.read_csv('listings.csv')
```

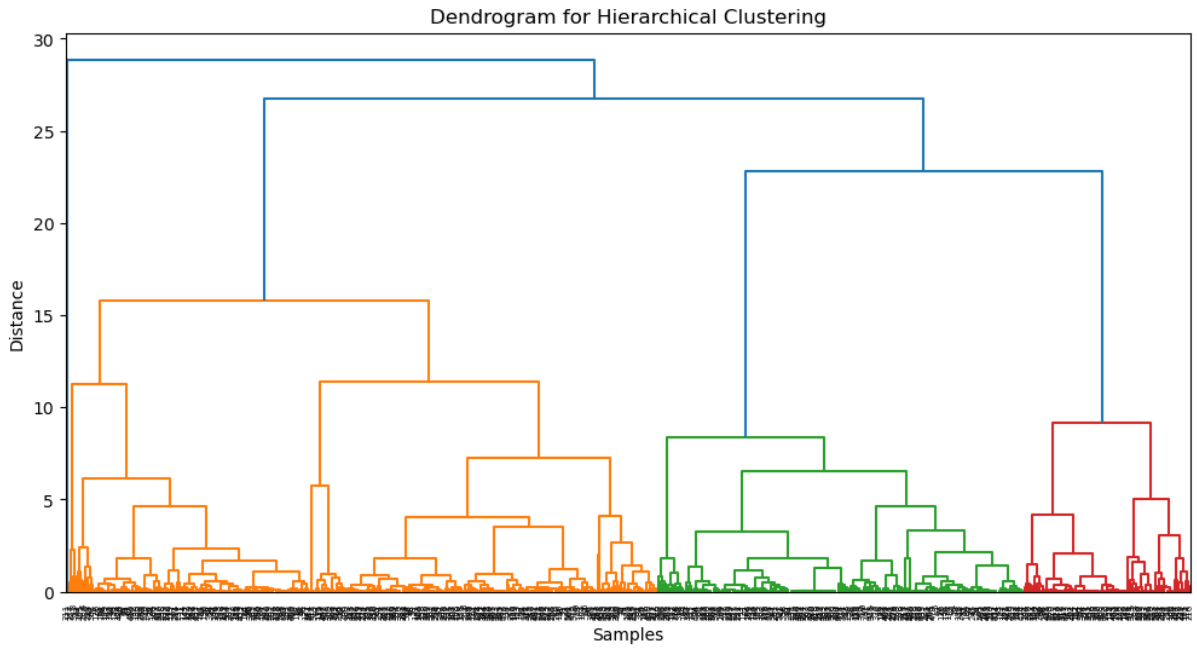


- Based on the dendrogram, we can see that many clusters are joined together with height lower than 2000.
- In the above dendrogram, we can also see that if we were to draw a horizontal line at height 5000, we can get 3 big cluster: 1 blue on the left, 1 orange in the middle (shorter and more spread) and one orange cluster on the right (taller and more compact).
- When we perform a horizontal cut, clusters that are in a segment are more similar than each other, compared to the other segments that are separated by the cut.
- Some of the notable heights which indicates the degree of the difference between branches are: 6000, 4000, and 1800.

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 [8]:
```
from scipy.cluster import hierarchy

data = pd.read_csv('listings.csv')

shared_room_data = data[data['room_type'] == 'Shared room']


relevant_data = shared_room_data[['longitude', 'latitude', 'price']]

#normalize

relevant_data = (relevant_data - relevant_data.mean()) / relevant_data.s

Z = hierarchy.linkage(relevant_data, method='ward', metric='euclidean')

# Plot the dendrogram
plt.figure(figsize=(12, 6))
dn = hierarchy.dendrogram(Z)
plt.title('Dendrogram for Hierarchical Clustering')
plt.xlabel('Samples')
plt.ylabel('Distance')
plt.show()
```

```
/var/folders/bj/hq4hjqs54_95ynh8j4m1yq3w0000gn/T/ipykernel_77279/222671
7182.py:3: DtypeWarning: Columns (17) have mixed types. Specify dtype o
ption on import or set low_memory=False.
  data = pd.read_csv('listings.csv')
```



- After being normalized, we can observe the merging distance between clusters better using the normalized distance.

- We can see that many clusters merge together with distance difference <= 15.
- We observe that, from distance > 16, there is a jump to distance 23 and distance 28 to the final merge.

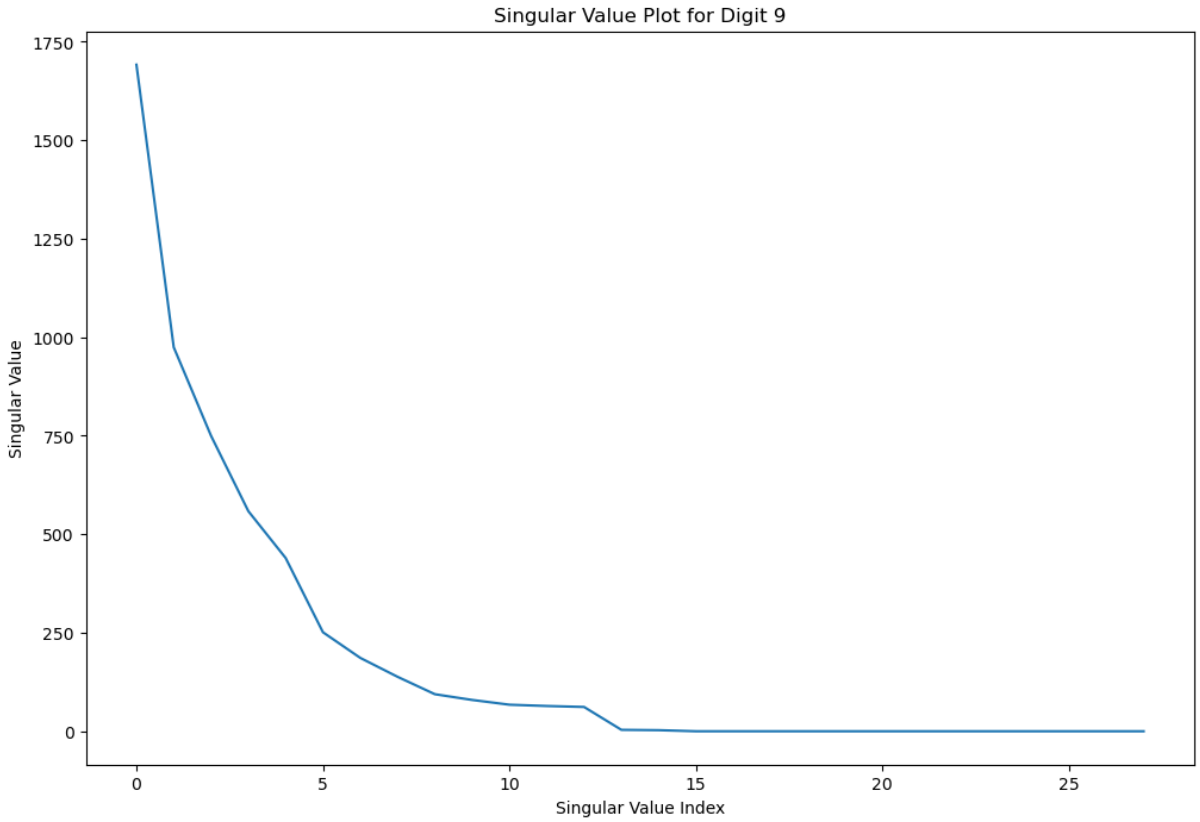
Exercise 2 (50 points)

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 [9]:
```
import matplotlib.pyplot as plt

from sklearn.datasets import fetch_openml

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

df = pd.DataFrame(X, columns=[f"pixel{i}" for i in range(X.shape[1])])
df['target'] = y

df.head(50)
# Save the DataFrame as a CSV file
df.to_csv('mnist.csv', index=False)
```

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

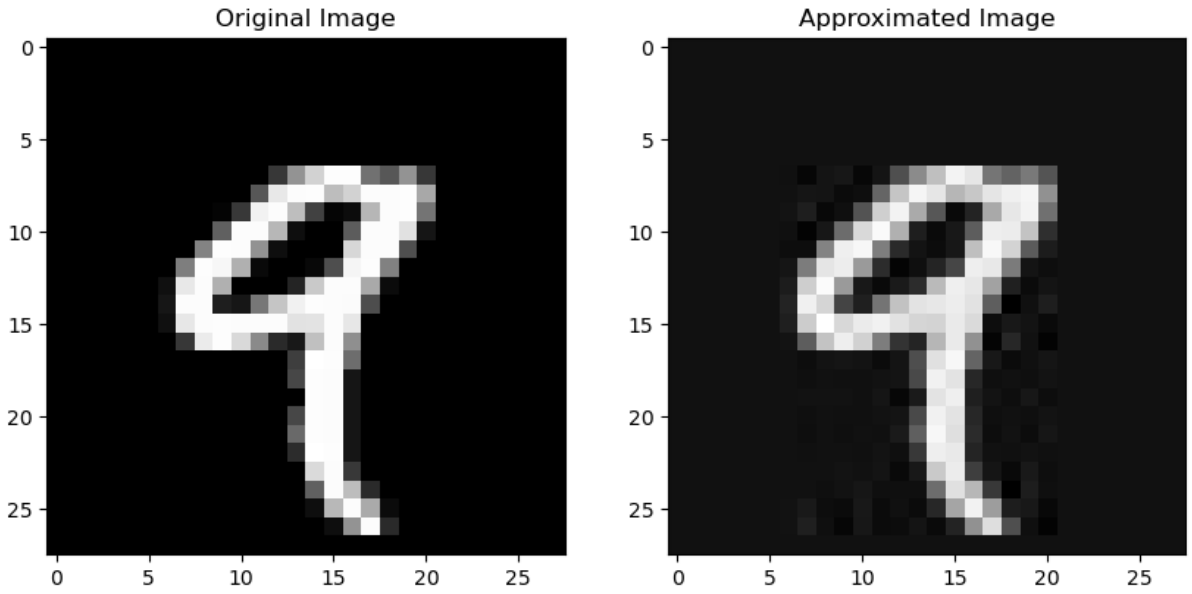
In [10]:
```
mnist_df = pd.read_csv("mnist.csv")

digit_9 = mnist_df.loc[mnist_df['target'] == 9].head(1).values[0, : -1]

U, S, Vt = np.linalg.svd(digit_9.reshape(28, 28))

plt.figure(figsize=(12,8))
plt.plot(S)
plt.title('Singular Value Plot for Digit 9')
plt.xlabel('Singular Value Index')
plt.ylabel('Singular Value')

plt.show()
```



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)

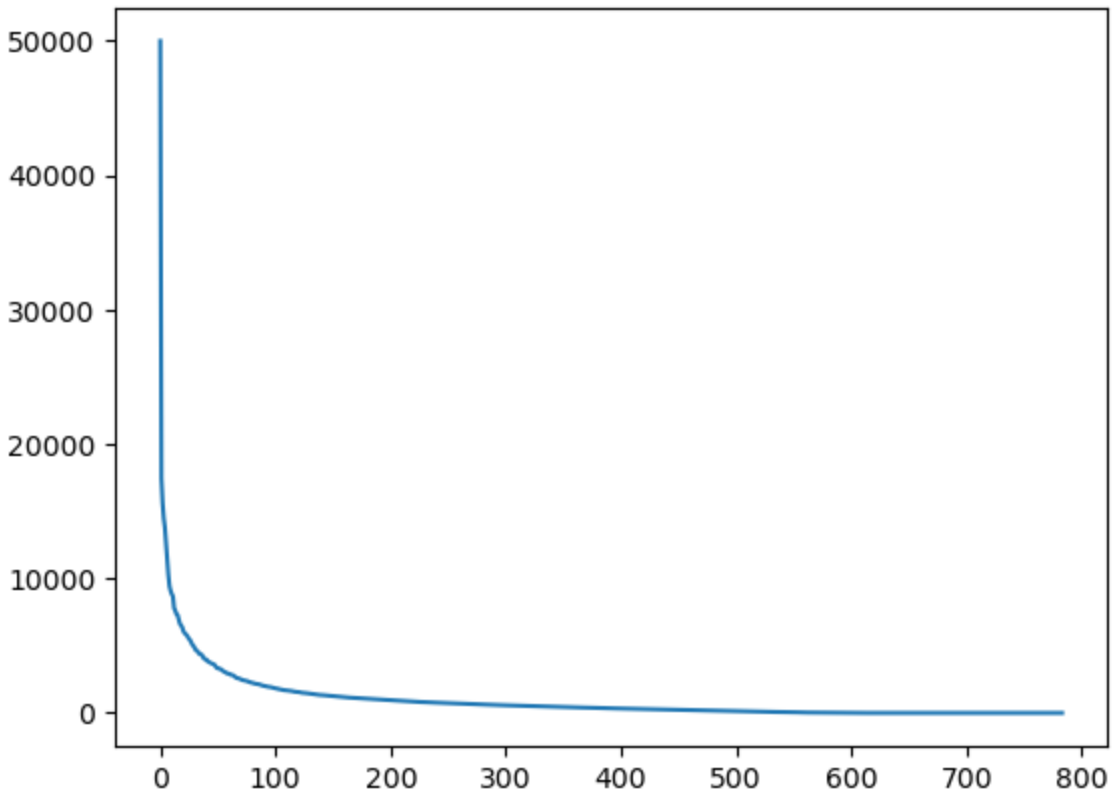
In [11]:

```
mnist_df = pd.read_csv("mnist.csv")

digit_9 = mnist_df[mnist_df['target'] == 9]

image_values = digit_9.drop('target', axis=1).values[0]

image_9_original = image_values.reshape(28, 28)

U, S, Vt = np.linalg.svd(image_9_original)
S[8:] = 0
approx_image_9 = U @ np.diag(S) @ Vt

# Plot the original and the approximated image of the digit 9
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.imshow(image_9_original, cmap='gray')
plt.title('Original Image')
plt.subplot(1, 2, 2)
plt.imshow(approx_image_9, cmap='gray')
plt.title('Approximated Image')
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)

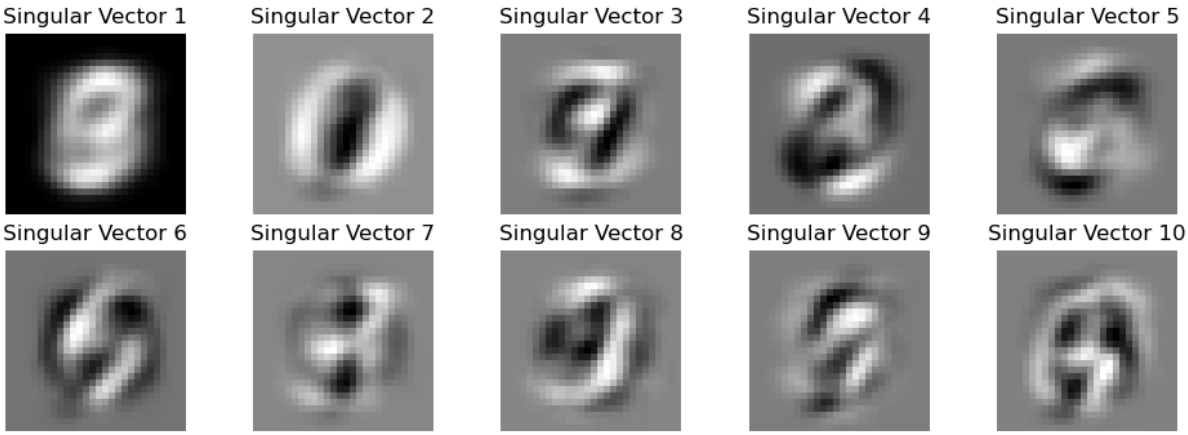
In [12]:
```
mnist_df = pd.read_csv("mnist.csv")

sample_size = 1000
mnist_sample = mnist_df.drop('target', axis=1).sample(n=sample_size, ran

# Perform SVD
U, S, Vt = np.linalg.svd(mnist_sample, full_matrices=False)

plt.plot(S)
plt.show()
```



After performing SVD and plotting the Singular values, we see that after x being approximately 50, the significance of the singular values goes down drasitaclly, which means rank 50 would be a reasonable choice.

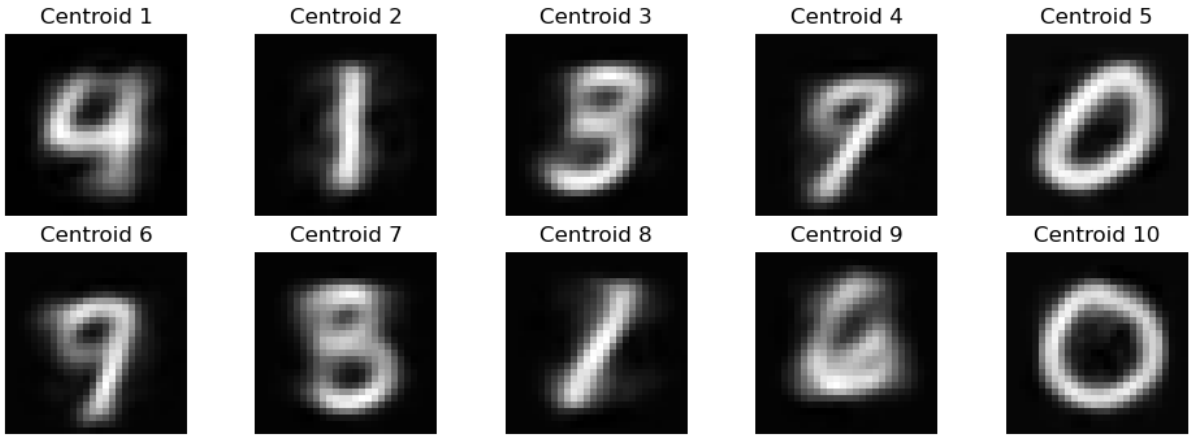
e) Plot the first 10 singular vectors. Notice that each singular vector's length will be 784 so you can plot them as a 28x28 image. (5points)

In [13]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# Read the CSV file into a DataFrame
mnist_df = pd.read_csv("mnist.csv")

sample_size = 1000
mnist_sample = mnist_df.sample(n=sample_size, random_state=20)

# Perform SVD
U, S, Vt = np.linalg.svd(mnist_sample.drop('target', axis=1), full_matri

# Plot the first 10 singular vectors as images
plt.figure(figsize=(12, 4))
for i in range(10):
    singular_vector = Vt[i].reshape(28, 28)
    plt.subplot(2, 5, i + 1)
    plt.imshow(singular_vector, cmap='gray')
    plt.title(f"Singular Vector {i+1}")
    plt.axis('off')
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)

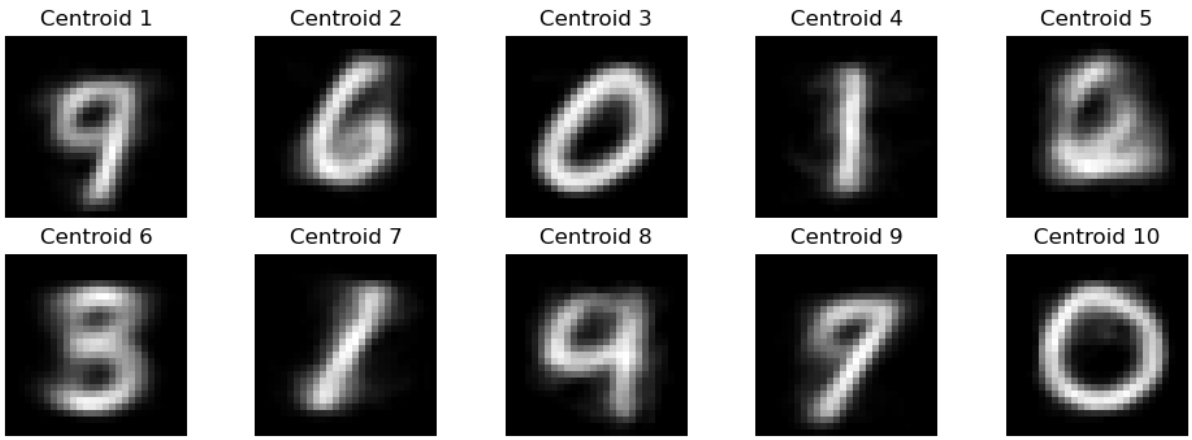
In [14]:
```
from sklearn.cluster import KMeans

U, S, Vt = np.linalg.svd(mnist_sample.drop('target', axis = 1), full_mat
S[50:] = 0
new_mnist_sample = U @ np.diag(S) @ Vt

kmeans = KMeans(n_clusters=10).fit(new_mnist_sample)

centroids = kmeans.cluster_centers_

# Plot the centroids as images
plt.figure(figsize=(12, 4))
for i in range(10):
    centroid_image = centroids[i].reshape(28, 28)
    plt.subplot(2, 5, i + 1)
    plt.imshow(centroid_image, cmap='gray')
    plt.title(f"Centroid {i+1}", fontsize=12)
    plt.axis('off')
plt.show()
```



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 [15]:
```
from sklearn.cluster import KMeans
#  Perform K-means clustering with 10 clusters
kmeans = KMeans(n_clusters=10)
cluster_labels = kmeans.fit_predict(mnist_sample.drop('target', axis=1))

# Get the centroids
centroids = kmeans.cluster_centers_

# Plot the centroids as images
plt.figure(figsize=(12, 4))
for i in range(10):
    centroid_image = centroids[i].reshape(28, 28)
    plt.subplot(2, 5, i + 1)
    plt.imshow(centroid_image, cmap='gray')
    plt.title(f"Centroid {i+1}", fontsize=12)
    plt.axis('off')
plt.show()
```

```
/Users/nurassylmedeuov/opt/anaconda3/lib/python3.9/site-packages/sklear
n/utils/validation.py:623: FutureWarning: is_sparse is deprecated and w
ill be removed in a future version. Check `isinstance(dtype, pd.SparseD
type)` instead.
  if not hasattr(array, "sparse") and array.dtypes.apply(is_sparse).any
():
```



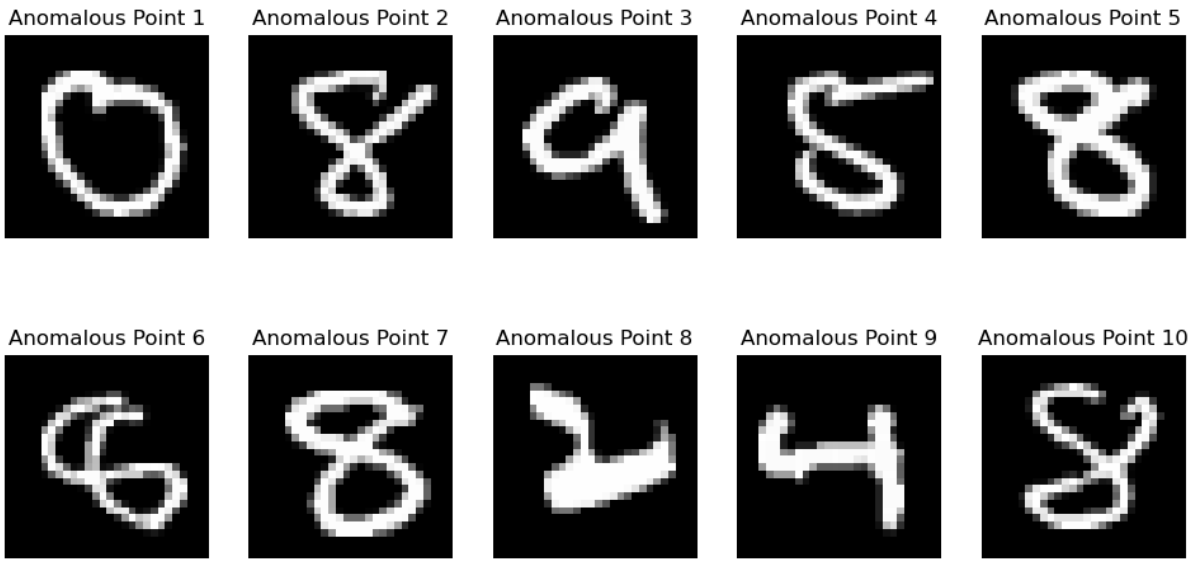
The centroids of the original matrix, and the lower dimension one look exactly the same to me!

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 [38]:
```
# Read the CSV file into a DataFrame
mnist_df = pd.read_csv("mnist.csv")

sample_size = 1000
original_dataset = mnist_df.sample(n=sample_size, random_state=20)
original_dataset = original_dataset.drop('target', axis = 1)


U, S, Vt = np.linalg.svd(original_dataset, full_matrices=False)
S[10:] = 0
rank_10_dataset = U @ np.diag(S) @ Vt

matrix_0 = original_dataset.to_numpy() - rank_10_dataset
```

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 [39]:
```

distances_from_origin = np.linalg.norm(matrix_0, axis=1)
largest_indices = np.argsort(distances_from_origin)[-10:]

plt.figure(figsize=(12, 6))
for i, index in enumerate(largest_indices):
    image_data = original_dataset.iloc[index, :].values
    image = image_data.reshape(28, 28)
    plt.subplot(2, 5, i + 1)
    plt.imshow(image, cmap='gray')
    plt.title(f"Anomalous Point {i + 1}")
    plt.axis('off')
plt.show()
```



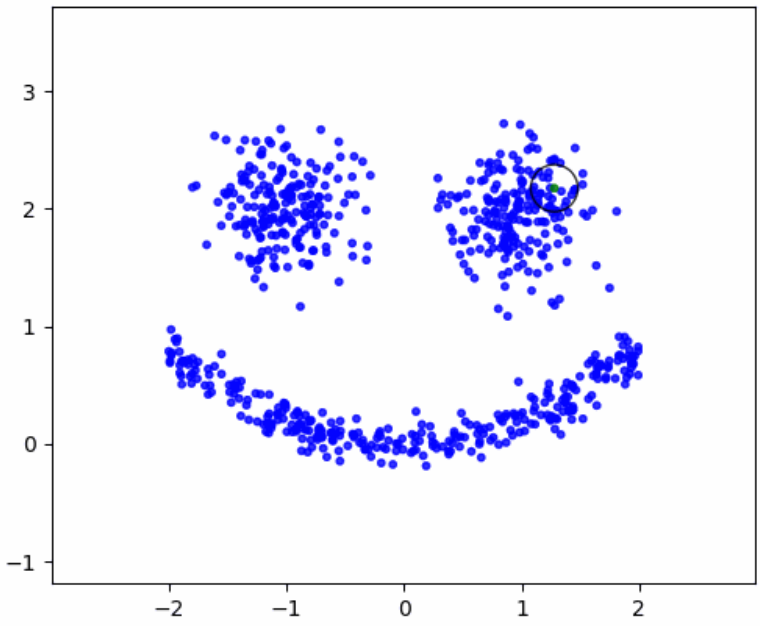
The largest rows in the matrix O represent data points that are very different from what was expected, therefore can be considered anomalous.

Bonus (20pts)

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

In [18]:
```
from IPython.display import Image
Image(filename="dbscan.gif", width=500, height=500)
```

Out[18]:

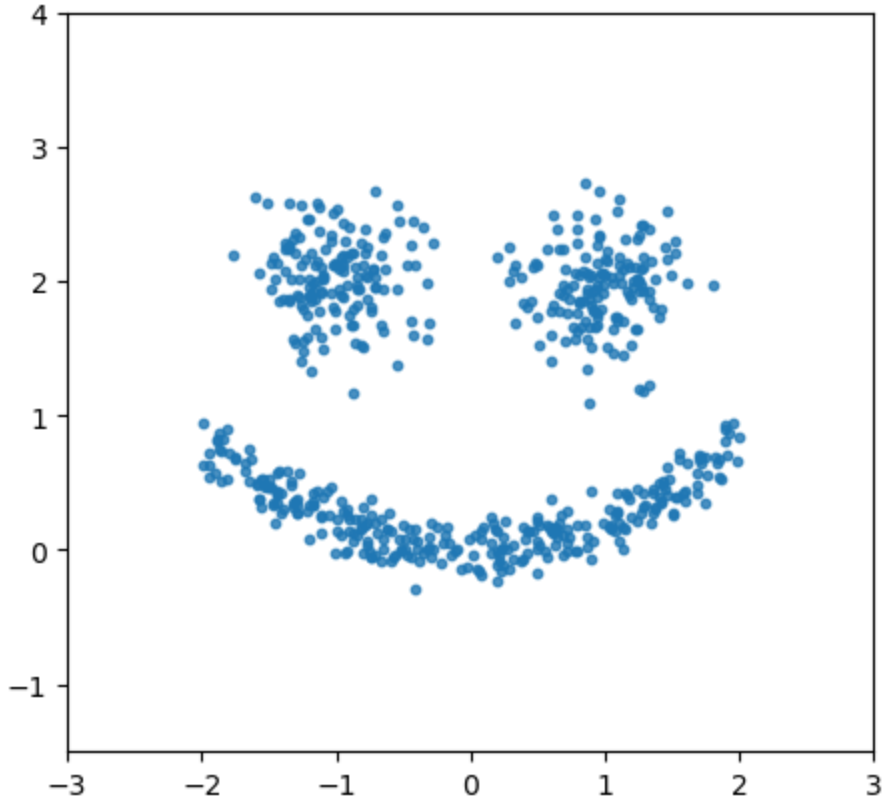


Hints:

- 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 [19]:
```
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(self.dataset))]  # 0 mea
        self.snaps = []

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

    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 assign(self, i, clusterNum ):
        self.assignments[i] = clusterNum
        neighborQueue = self.get_neighborhood(i)

        while neighborQueue:
            nextCandidate = neighborQueue.pop()

            if self.assignments[nextCandidate] != 0:
                    #already assigned
                    #duplicates can occur when adding point in neighborh
                    continue
            self.assignments[nextCandidate] = clusterNum
            point = (self.dataset[nextCandidate][0], self.dataset[nextCa
            snapshottaken = self.snapshot(point)
            self.snaps.append(snapshottaken)
            if self.is_core(nextCandidate):
                nextNeighborhood = self.get_neighborhood(nextCandidate)
                neighborQueue += ([ i for i in nextNeighborhood if self.a
        return

    def snapshot(self, point):
        fig, ax = plt.subplots()
        colors = np.array([x for x in 'bgrcmykbgrcmykbgrcmykbgrcmyk'])
        colors = np.hstack([colors] * 20)
```

```
        ax.scatter(self.dataset[:, 0], self.dataset[:, 1], color=colors[
        cir = plt.Circle(xy = (point), radius = 0.4, fill = False) # crea
        ax.add_patch(cir)
        ax.set_xticks([-2,-1,0,1,2])
        ax.set_yticks([-1,0,1,2,3])
        ax.set_xlim([-3,3])
        ax.set_ylim([-1,4])
        ax.set_aspect('equal') # necessary or else the circles appear to

        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.is_core(i) and self.assignments[i] == 0:
                self.assign(i, cluster_num)
                # self.snaps.append(self.snapshot())
                cluster_num += 1
        return self.assignments

# Set the center of eyes
centers = [[-1, 2], [1, 2]]
eyes, _ = datasets.make_blobs(n_samples=300, centers=centers, cluster_st
                              random_state=0)

# set the width of mouth to [-2,2]
mouth_x = np.random.uniform(-2, 2, 300)  # Random x-coordinates within a
mouth_y = 0.2 * mouth_x ** 2 + 0.12 * np.random.randn(300)  # Calculate y

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

fig, ax = plt.subplots()
ax.scatter(face[:, 0], face[:, 1], s=10, alpha=0.8)
# Shape the eyes to be circle
ax.set_aspect('equal')

# Set plot limit
ax.set_ylim(-1.5, 4)
ax.set_xlim(-3, 3)
dbc = DBC(face, 3, 0.25)
clustering = dbc.dbscan()

dbc.snaps[0].save(
    'dbscan_1.gif',
    optimize=False,
    save_all=True,
    append_images=dbc.snaps[1:],
    loop=0,
    duration=100  # Adjust duration as needed
)
```


4
3
2
1
0
−1
−3
−2
−1
0
1
2
3

In [20]:
```
from IPython.display import display, Image

# Specify the path to the GIF file
gif_path = './dbscan_1.gif'

# Use the `display` function to show the GIF
display(Image(filename=gif_path))
```