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 [204…

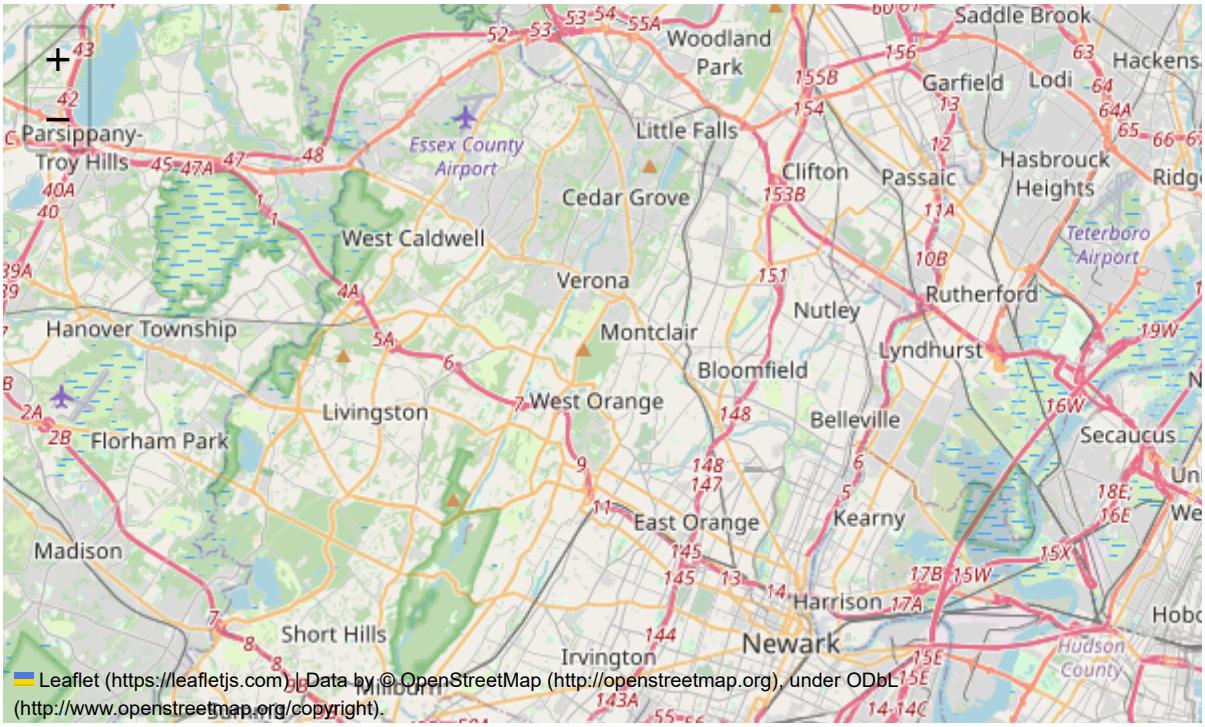
```
# Do not edit this cell
import pandas as pd
import numpy as np
import folium #install if you haven't already
import selenium #install if you haven't already
from IPython.display import Image #install if you haven't already

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 [205…
```
data = pd.read_csv('listings.csv')
```

```
C:\Users\91960\AppData\Local\Temp\ipykernel_13396\1577760308.py:1: DtypeWarning: Colu
mns (17) have mixed types. Specify dtype option on import or set low_memory=False.
  data = pd.read_csv('listings.csv')
```

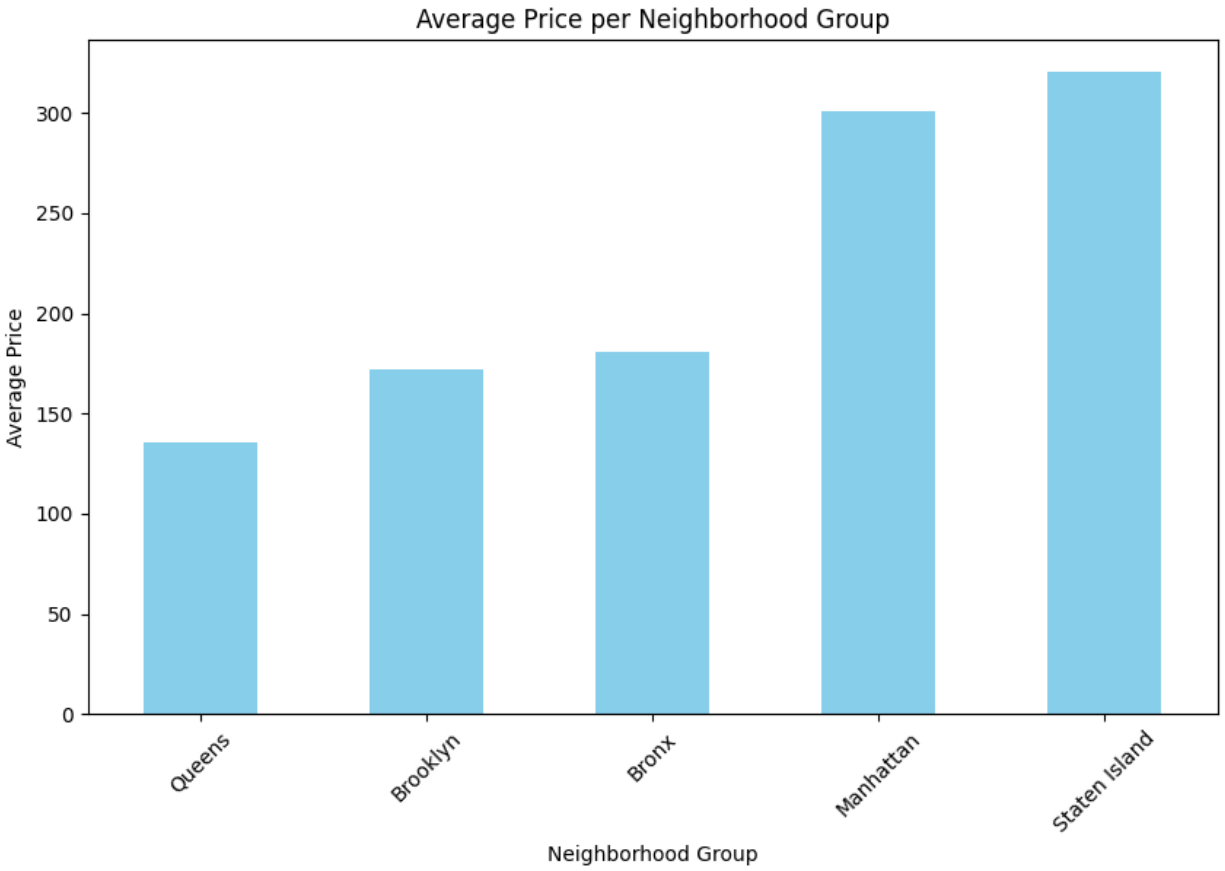
In [206…
```
from folium.plugins import MarkerCluster, FastMarkerCluster #Using either is fine but


mean_price_by_location = data.groupby(['latitude', 'longitude'])['price'].mean().reset
nyc_map = folium.Map(location=[40.693943, -73.985880], zoom_start=11)

marker_cluster = FastMarkerCluster(data=list(zip(mean_price_by_location['latitude'], m
marker_cluster.add_to(nyc_map)

convert_map_to_png(nyc_map, 'problem1a')
display(nyc_map)
```



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 [207…
```
import matplotlib.pyplot as plt

average_price_by_group = data.groupby('neighbourhood_group')['price'].mean().sort_valu


# Create a bar chart
```

```
plt.figure(figsize=(10, 6))
average_price_by_group.plot(kind='bar', color='skyblue')
plt.title('Average Price per Neighborhood Group')
plt.xlabel('Neighborhood Group')
plt.ylabel('Average Price')
plt.xticks(rotation=45)
plt.show()
```

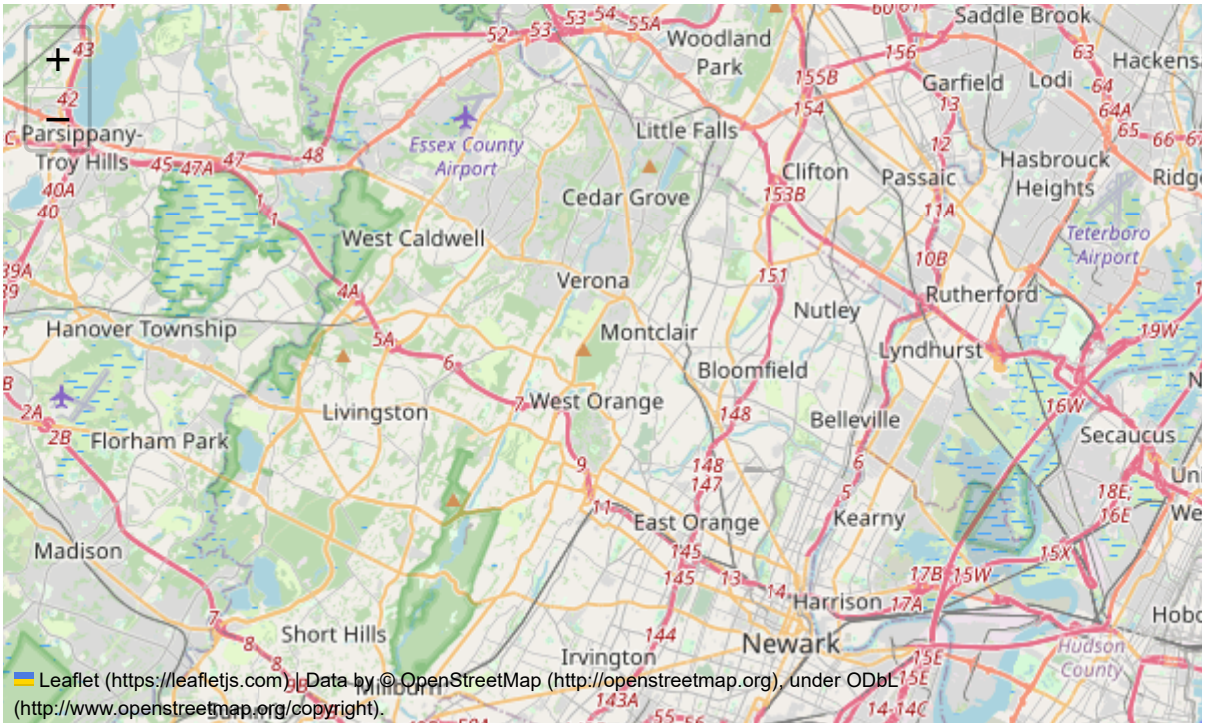


The average price of places in Staten Island and Manhattan areas is quite expensive compared to the rest. From the map I observed that they're tightly packed neighbourhoods and also have really busy ports and cruise ship terminals. And obviously the land values are high due to rich people living in these areas.

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 [208…

```
long_term_listings = data[data['minimum_nights'] >= 300]

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

marker_cluster = FastMarkerCluster(data=list(zip(long_term_listings['latitude'], long_
marker_cluster.add_to(nyc_map_2)
convert_map_to_png(nyc_map_2, 'problem1c')
display(nyc_map_2)
```

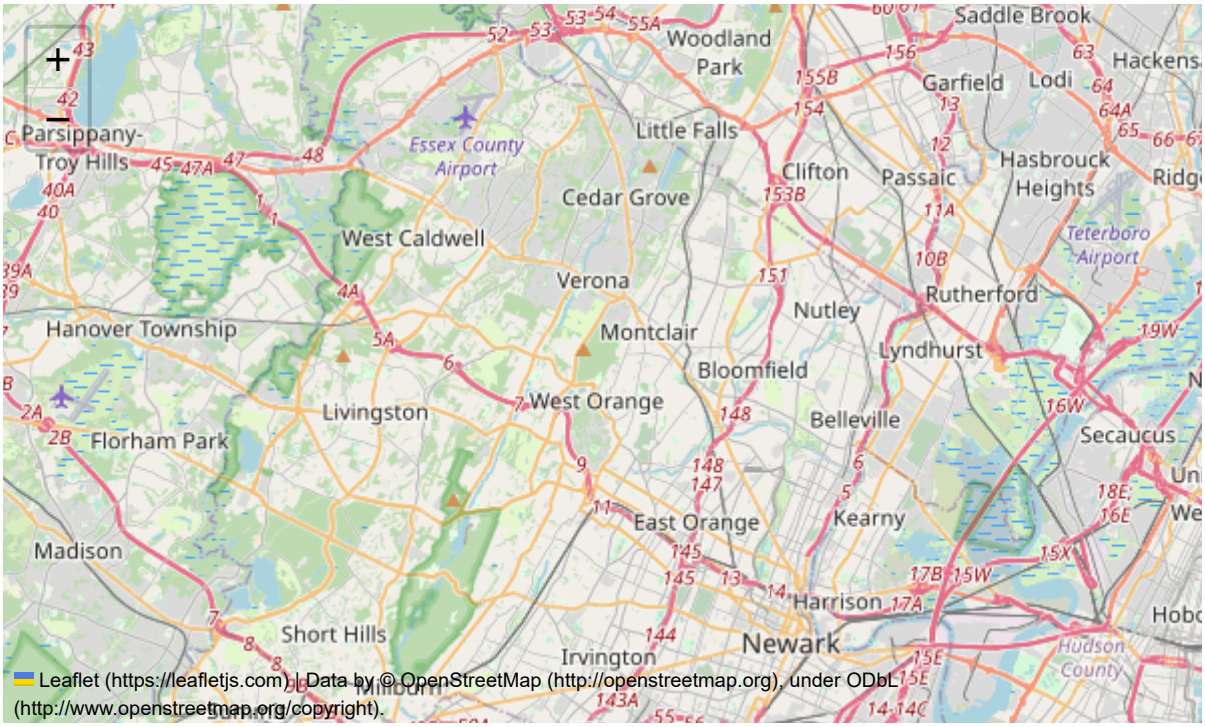


d) Using `longitude`, `latitude`, `price`, and `number_of_reviews`, use Density-based clustering to create clusters. Plot the points on the NYC map in a color corresponding to their cluster (color could be randomly assigned, but ensure each datapoint is colored to its associated cluster). For using `DBSCAN`, 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 [224…

```
# Write your code below! Leave the instantiated variables: it is for your convenience.
from sklearn.cluster import DBSCAN
from folium import CircleMarker
from folium.plugins import MarkerCluster
import random
from sklearn.preprocessing import StandardScaler

x = data[['latitude', 'longitude', 'price', 'number_of_reviews']]
scaler = StandardScaler()

X = scaler.fit_transform(x)
X = pd.DataFrame(X)
X.columns = x.columns

dbscan = DBSCAN(eps=0.3, min_samples=10)
X['cluster'] = dbscan.fit_predict(X)

colors = ['#' + ''.join(random.choice('0123456789ABCDEF') for j in range(6)) for i in
cluster_colors = {cluster: color for cluster, color in zip(X['cluster'].unique(), colo

X['latitude'] = x['latitude']
X['longitude'] = x['longitude']

nyc_map_3 = folium.Map(location=[40.693943, -73.985880], zoom_start=11)

for row in X.itertuples():
    cluster_color = cluster_colors.get(row.cluster, '#000000')  # Default to black if
```

```
    CircleMarker(
        location=[row.latitude, row.longitude],
        radius=1,
        color=cluster_color,
    ).add_to(nyc_map_3)

convert_map_to_png(nyc_map_3, 'problem1d')
display(nyc_map_3)
```



In [226…

```
print("Number of clusters")
print(X.cluster.nunique()-1)
```

```
Number of clusters
14
```

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)

When adjusting `eps` in DBSCAN:

- Increasing `eps` results in larger clusters by considering more distant points, possibly including outliers in clusters.
- Decreasing `eps` leads to smaller clusters with tightly packed points, increasing the chance of forming smaller, more distinct clusters.
- In part d, increasing to 0.9 resulted in only 2 clusters while 0.3 gave 14 clusters. I could see one predominant color almost everywhere.

When adjusting `min_samples` in DBSCAN:

- Increasing `min_samples` makes the algorithm more stringent, requiring more points to form a cluster, potentially classifying smaller clusters as noise.
- Decreasing `min_samples` relaxes the criteria for forming clusters, making it easier to create clusters, including small clusters with just a few points.
- In part d, increasing to 100 resulted in only 5 clusters while 10 gave 14 clusters. Clusters of all colors were clearly visible and aesthetically pleasing.

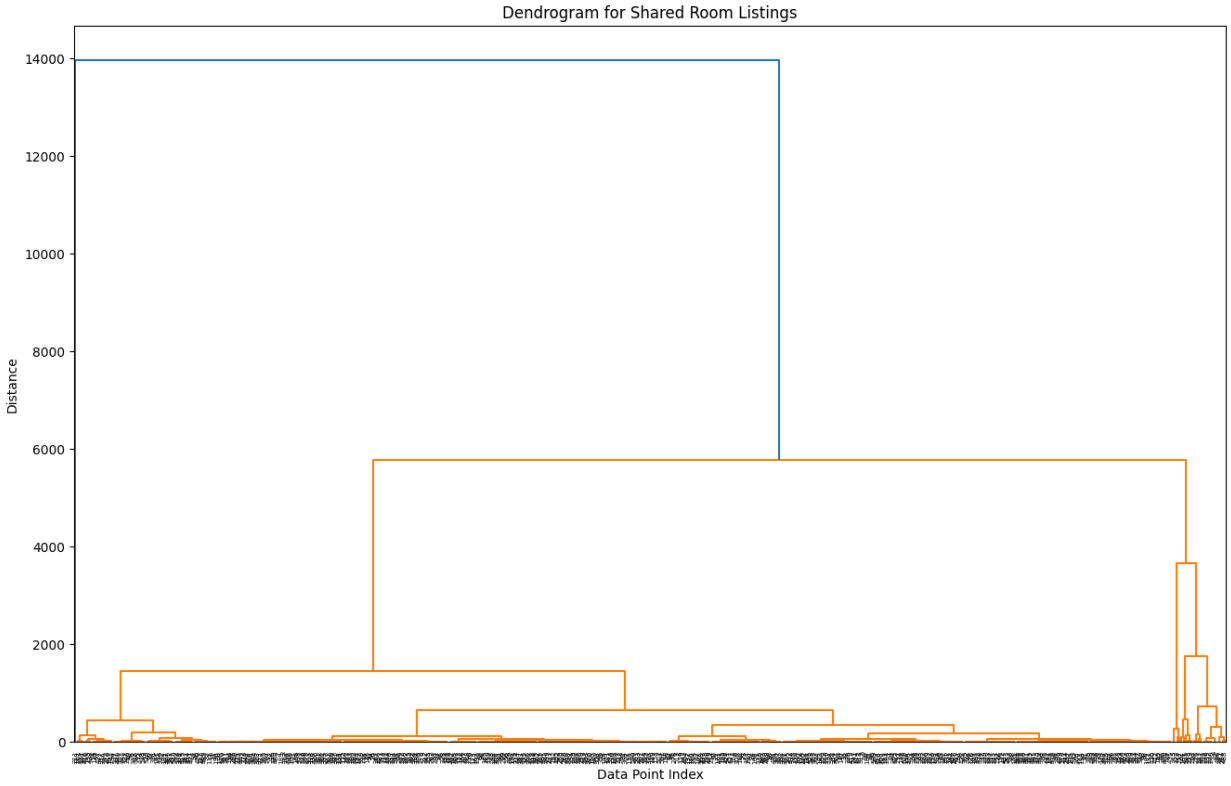
f) For part d, were the clusters seemed to be scattered or grouped together? Justify your answer. - (2.5 points)

The clusters seemed to be scattered. Upon zooming into the map, I was able to see that the green points were scattered over all regions, even some of the other clusters except three clusters which look denser.

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 [216…

```
from scipy.cluster import hierarchy


shared_room_data = data[data['room_type'] == 'Shared room']
X = shared_room_data[['longitude', 'latitude', 'price']]

linkage_matrix = hierarchy.linkage(X, method='ward')  # You can use a different linkag
plt.figure(figsize=(16,10))

dendrogram = hierarchy.dendrogram(linkage_matrix)

plt.title("Dendrogram for Shared Room Listings")
plt.xlabel("Data Point Index")
plt.ylabel("Distance")
plt.show()
```



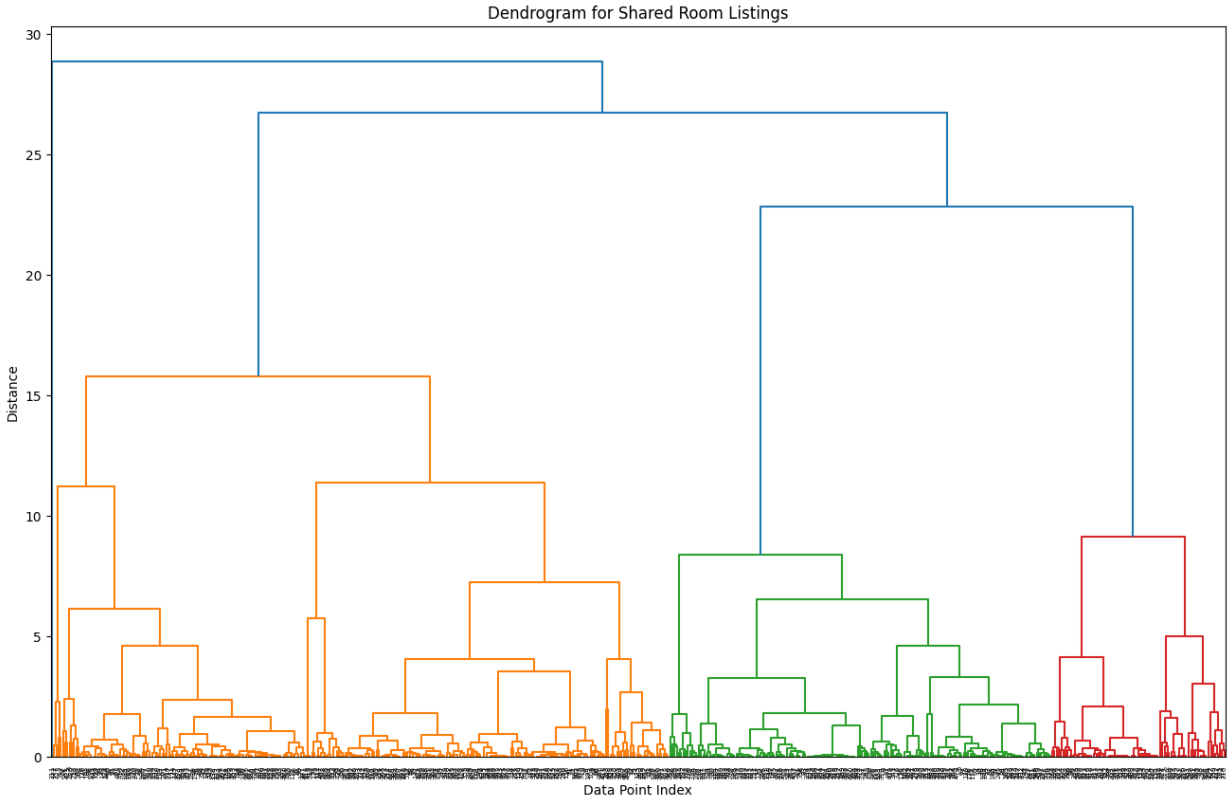
I can observe that the expensive places have smaller clusters, which is obvious. And the internal distances is also high. Whereas, the clusters that includes cheaper places have larger cluster sizes, which is also expected.

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 [215…

```
def normalize(df):
    X = df.copy()
    mean = X['longitude'].mean()
    std = X['longitude'].std()
    X['longitude'] = X['longitude'].apply(lambda x: (x-mean)/std)

    mean = X['latitude'].mean()
    std = X['latitude'].std()
    X['latitude'] = X['latitude'].apply(lambda x: (x-mean)/std)

    mean = X['price'].mean()
    std = X['price'].std()
    X['price'] = X['price'].apply(lambda x: (x-mean)/std)

    return X


shared_room_data = data[data['room_type'] == 'Shared room']
X = shared_room_data[['longitude', 'latitude', 'price']]

X = normalize(X)
linkage_matrix = hierarchy.linkage(X, method='ward')  # You can use a different linkag
plt.figure(figsize=(16,10))
dendrogram = hierarchy.dendrogram(linkage_matrix)
```

```
plt.title("Dendrogram for Shared Room Listings")
plt.xlabel("Data Point Index")
plt.ylabel("Distance")
plt.show()
```



After normalizing, I can observe that the difference between the cluster sizes has become lesser and clusters are of comparable sizes now. It is kind of what we would want to expect as a Data Scientist. Normalizing made it easier for the algorithm to differentiate various features and interpret them or treat them all equally to arrive at a more meaningful clustering.

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)

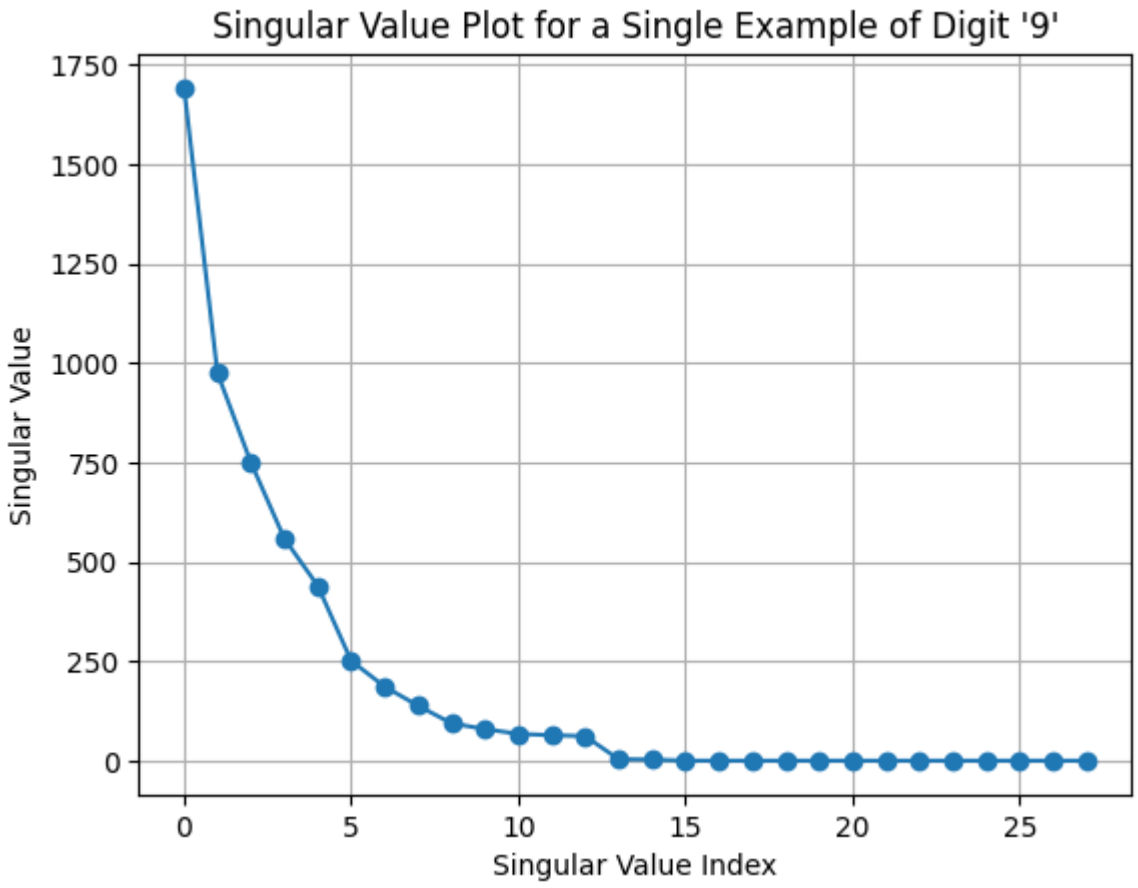
In [231…

```
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_frame=False)

mnist_data = pd.DataFrame(X, columns=[f"pixel_{i}" for i in range(X.shape[1])])
mnist_data['label'] = y
mnist_data.to_csv('mnist_data.csv', index=False)
```

```
c:\Users\91960\anaconda3\envs\tf\lib\site-packages\sklearn\datasets\_openml.py:968: F
utureWarning: The default value of `parser` will change from ``'liac-arff'`` to ``'aut
o'`` in 1.4. You can set `parser='auto'` to silence this warning. Therefore, an `Impor
tError` will be raised from 1.4 if the dataset is dense and pandas is not installed.
Note that the pandas parser may return different data types. See the Notes Section in
fetch_openml's API doc for details.
  warn(
```

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

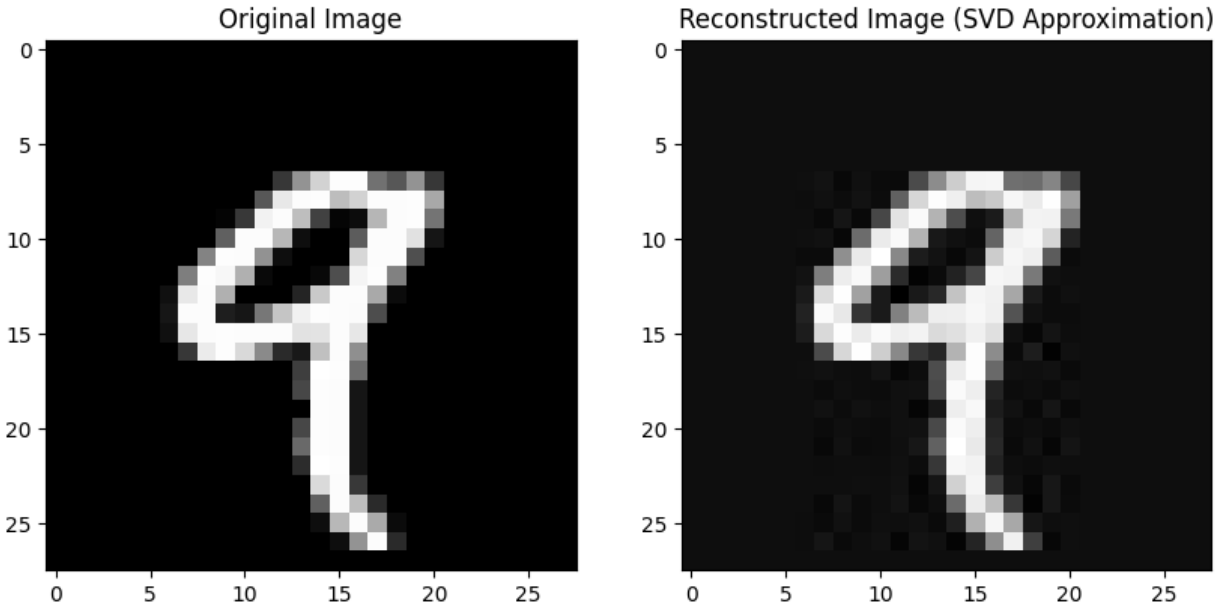
In [51]:
```
digit_9_indices = np.where(y == '9')[0]
X_digit_9 = X[digit_9_indices]

example_9 = X_digit_9[0]

U, s, Vt = np.linalg.svd(example_9.reshape(28, 28), full_matrices=False)

plt.plot(s, marker='o', linestyle='-')
plt.xlabel("Singular Value Index")
plt.ylabel("Singular Value")
plt.title("Singular Value Plot for a Single Example of Digit '9'")
plt.grid()
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 [53]:
```
U, s, Vt = np.linalg.svd(example_9.reshape(28, 28), full_matrices=False)
```

```
# Set a threshold to control the level of approximation (e.g., keep the top k singular
threshold = 10  # Adjust this value to control the approximation

# Retain only the top 'threshold' singular values and vectors, set the rest to 0
s[threshold:] = 0
approximation = np.dot(U, np.dot(np.diag(s), Vt))

# Plot the original and the reconstructed images side by side
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.imshow(example_9.reshape(28, 28), cmap='gray')
plt.title("Original Image")

plt.subplot(1, 2, 2)
plt.imshow(approximation, cmap='gray')
plt.title("Reconstructed Image (SVD Approximation)")

plt.show()
```

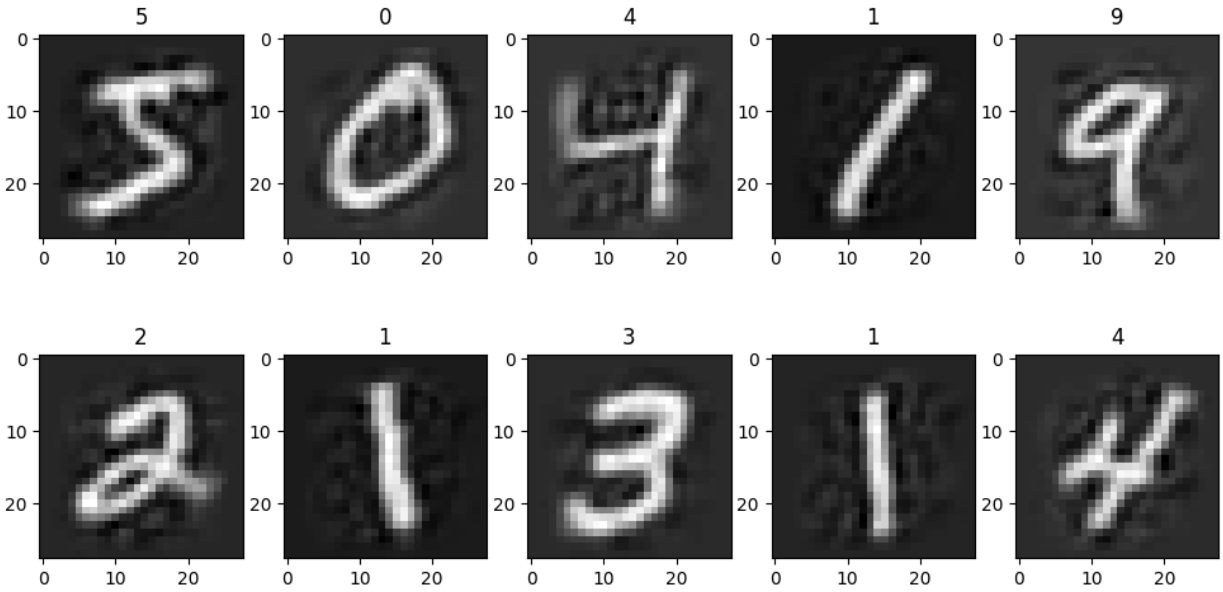


d) Consider the entire dataset as a matrix. Perform SVD and explain why / how you chose a particular rank. Note: you may not be able to run this on the entire dataset in a reasonable amount of time so you may take a small random sample for this and the following questions. (5 points)

In [237…

```
sample_x = X[:1000]
sample_y = y[:1000]
U, s, Vt = np.linalg.svd(sample_x, full_matrices=False)
threshold = 80  # Adjust this value to control the approximation
# Retain only the top 'threshold' singular values and vectors, set the rest to 0
s[threshold:] = 0
approximation = np.dot(U, np.dot(np.diag(s), Vt))
first_10_approximations = approximation[:10]
plt.figure(figsize=(12, 6))

for i in range(10):
    plt.subplot(2, 5, i + 1)
    singular_vector = first_10_approximations[i]
    singular_image = singular_vector.reshape(28, 28)
```

```
    plt.imshow(singular_image, cmap='gray')
    plt.title(sample_y[i])

plt.show()
```

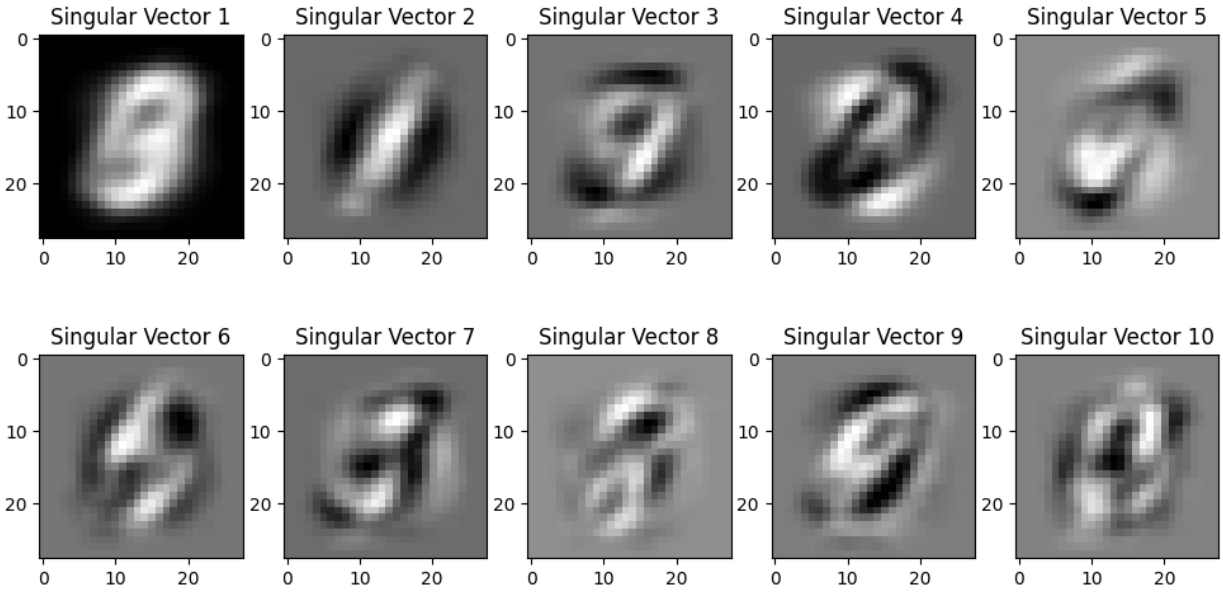


I chose the rank as 80 after trying out multiple ranks and seeing how different they're from the original values. There is still a lot of bluriness in the images but the numbers are still distinguishable and classifiable. As a human I can classify these images into digits correctly.

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 [240…

```
first_10_singular_vectors = Vt[:10]
plt.figure(figsize=(12, 6))
for i in range(10):
    plt.subplot(2, 5, i + 1)
    singular_vector = first_10_singular_vectors[i]
    singular_image = singular_vector.reshape(28, 28)
    plt.imshow(singular_image, cmap='gray')
    plt.title(f"Singular Vector {i + 1}")

plt.show()
```

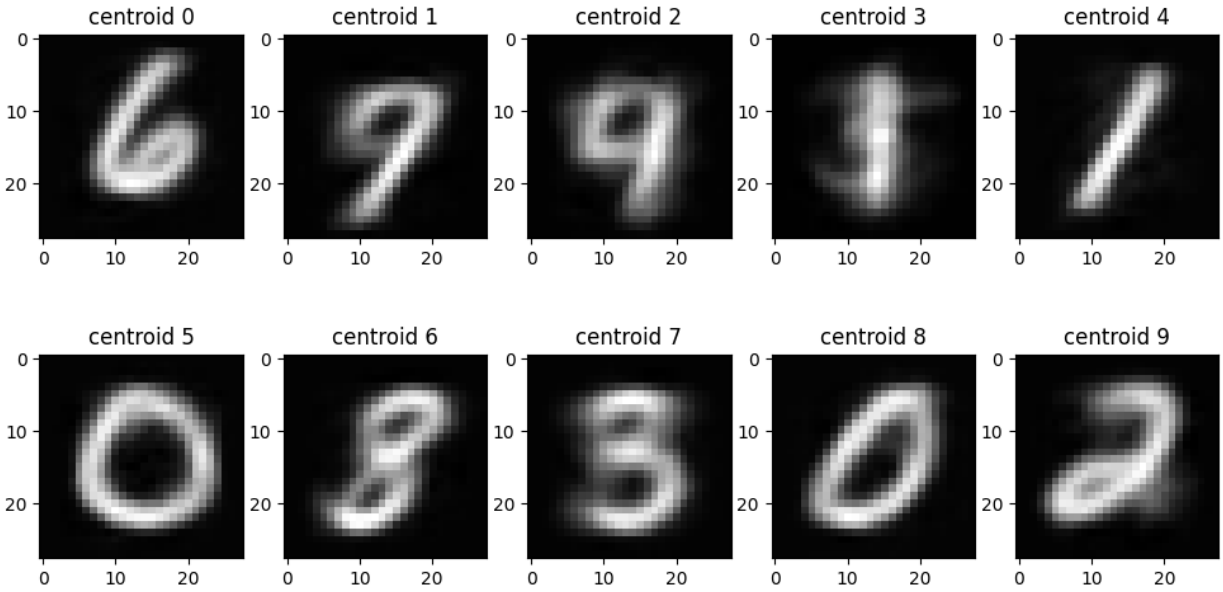


f) Using Kmeans on this new dataset, cluster the images from d) using 10 clusters and plot the centroid of each cluster. Note: the centroids should be represented as images. (10 points)

In [239…

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
# Create a K-Means model
kmeans = KMeans(n_clusters=10, random_state=0)
kmeans.fit(approximation)
plt.figure(figsize=(12, 6))

for i in range(10):
    plt.subplot(2, 5, i + 1)
    singular_vector = kmeans.cluster_centers_[i]
    singular_image = singular_vector.reshape(28, 28)
    plt.imshow(singular_image, cmap='gray')
    plt.title(f'centroid {i}')

plt.show()
```

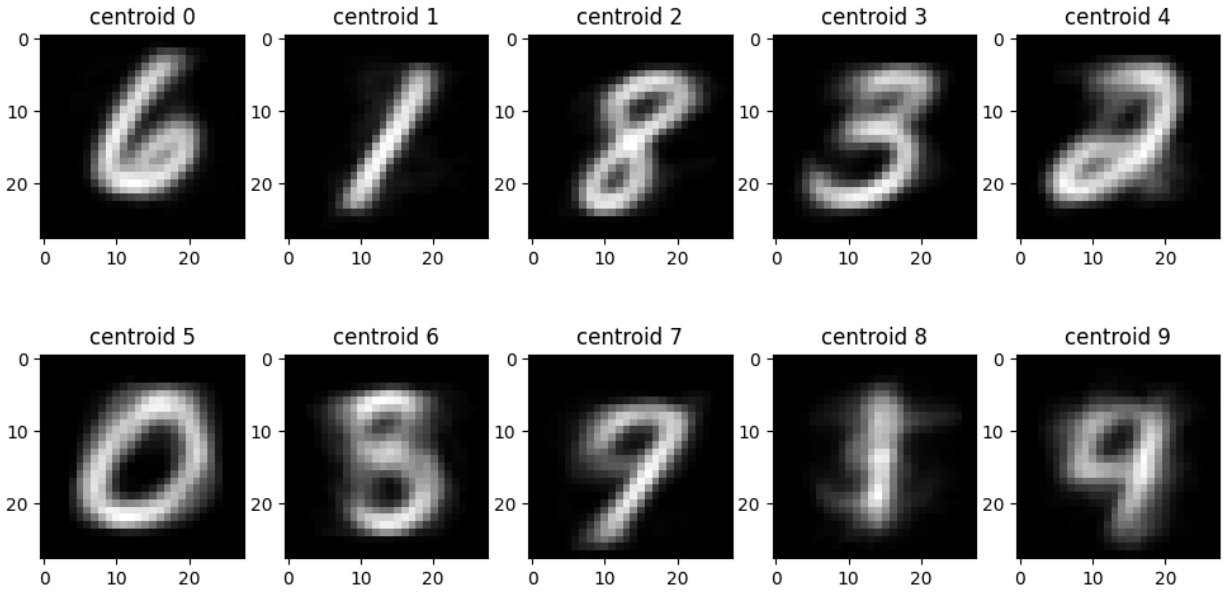
```
c:\Users\91960\anaconda3\envs\tf\lib\site-packages\sklearn\cluster\_kmeans.py:870: Fu
tureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set
the value of `n_init` explicitly to suppress the warning
  warnings.warn(
```



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 [89]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
# Create a K-Means model
kmeans = KMeans(n_clusters=10, random_state=0)
kmeans.fit(sample_x)
plt.figure(figsize=(12, 6))

for i in range(10):
    plt.subplot(2, 5, i + 1)
    singular_vector = kmeans.cluster_centers_[i]
    singular_image = singular_vector.reshape(28, 28)
    plt.imshow(singular_image, cmap='gray')
    plt.title(f'centroid {i}')

plt.show()
```

```
c:\Users\91960\anaconda3\envs\tf\lib\site-packages\sklearn\cluster\_kmeans.py:870: Fu
tureWarning: 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 warning
  warnings.warn(
```



In summary, the differences in the centroid image plots between K-means clustering of the MNIST dataset and rank-100 approximations can be summarized as follows:

- **Original Centroids** capture the complexity and diversity of features, maintaining high visual fidelity to the original images.
- **Approximated Centroids** emphasize abstraction, simplification, and data compression, resulting in reduced detail and a more uniform appearance.

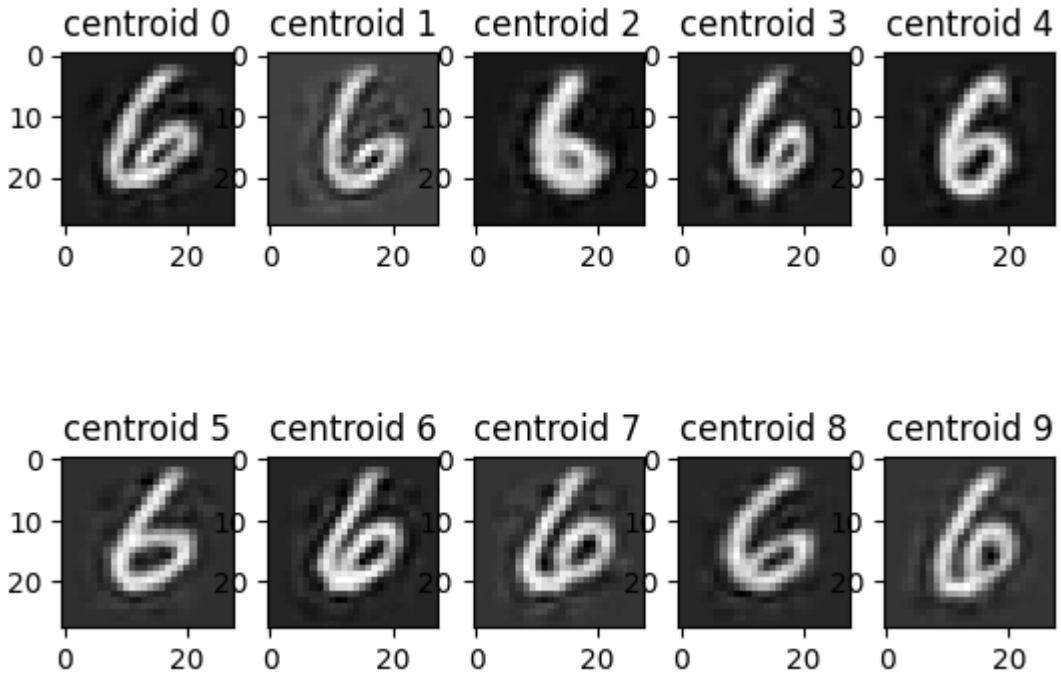
The choice between the two depends on the balance between detail and simplification required for a particular application.

Some centroids are completely different while some are approximately the same.

If you check out Centroid 1 in both, the circular arc feature of number 9 does not exist in the original centroid. The centroids need not match in both. They're clearly not comparable directly, but the approximation is actually quite good as every centroid has different features. Just to confirm, I am plotting some approximated images clustered by the kmeans algorithms of a single cluster.

In [245…

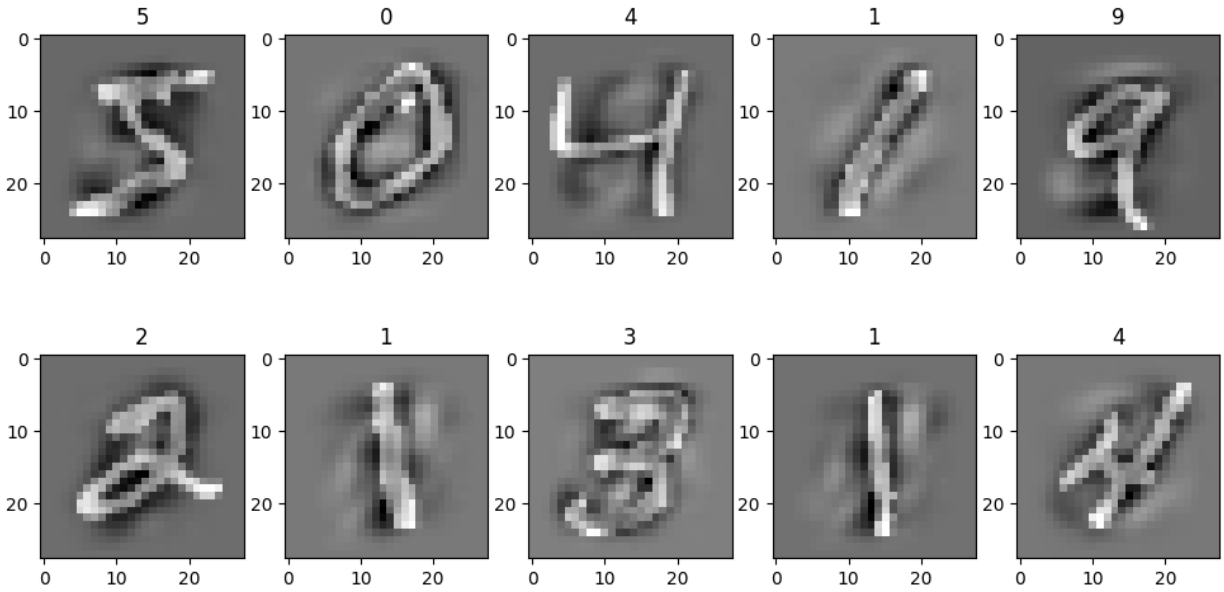
```
for i in range(10):
    plt.subplot(2, 5, i + 1)
    singular_vector = approximation[kmeans.labels_==0][i]
    singular_image = singular_vector.reshape(28, 28)
    plt.imshow(singular_image, cmap='gray')
    plt.title(f'centroid {i}')

plt.show()
```



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

In [246…

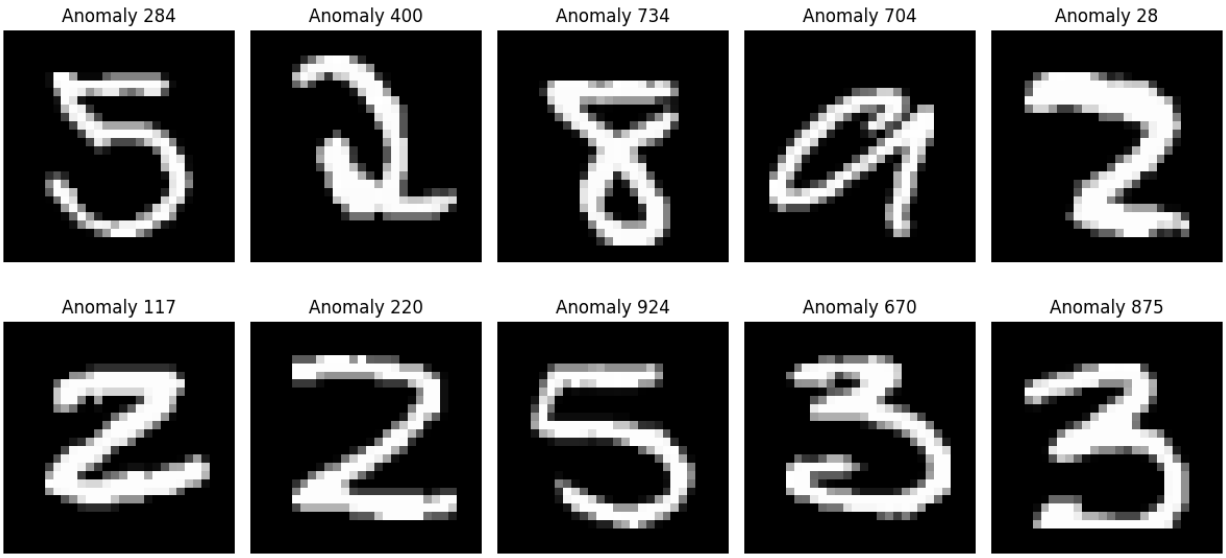
```
sample_x = X[:1000]
sample_y = y[:1000]
U, s, Vt = np.linalg.svd(sample_x, full_matrices=False)
threshold = 10  # Adjust this value to control the approximation
# Retain only the top 'threshold' singular values and vectors, set the rest to 0
s[threshold:] = 0
approximation = np.dot(U, np.dot(np.diag(s), Vt))
O = sample_x - approximation
plt.figure(figsize=(12, 6))

for i in range(10):
    plt.subplot(2, 5, i + 1)
    singular_vector = O[:10][i]
    singular_image = singular_vector.reshape(28, 28)
    plt.imshow(singular_image, cmap='gray')
    plt.title(sample_y[i])

plt.show()
```



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

In [247…

```
largest_rows_indices = np.argsort(np.linalg.norm(O, axis=1))[-10:]
fig, axes = plt.subplots(2, 5, figsize=(12, 6))

for i, ax in zip(largest_rows_indices, axes.flatten()):
    ax.imshow(X[i].reshape(28, 28), cmap='gray')
    ax.set_title(f'Anomaly {i}')
    ax.axis('off')

plt.tight_layout()
plt.show()
```



In this case, the "distance" refers to the Euclidean distance between two vectors:

Original Data vector: The original image from the MNIST dataset, represented as a vector of pixel values.

Rank-10 Approximation vector: The low-rank approximation of the original image, also represented as a vector of pixel values.

The Euclidean distance of the row vector of 'O' is nothing but the Euclidean distance between the above two vectors. The Euclidean distance between these two vectors (original and approximation) is calculated using the Euclidean distance formula, which is akin to the Pythagorean theorem in multiple dimensions. It measures how far apart these two vectors are in the feature space.

When the Euclidean distance is large, it indicates a substantial difference between the original data point and its approximation. Conversely, a smaller Euclidean distance suggests that the approximation closely resembles the original data point. So the largest rows of the matrix 'O' are considered the anamalous points.

Bonus (20pts)

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

In [126…
```
from IPython.display import Image
Image(filename="dbscan.gif", width=500, height=500)
```

Out[126]:
```
<IPython.core.display.Image object>
```

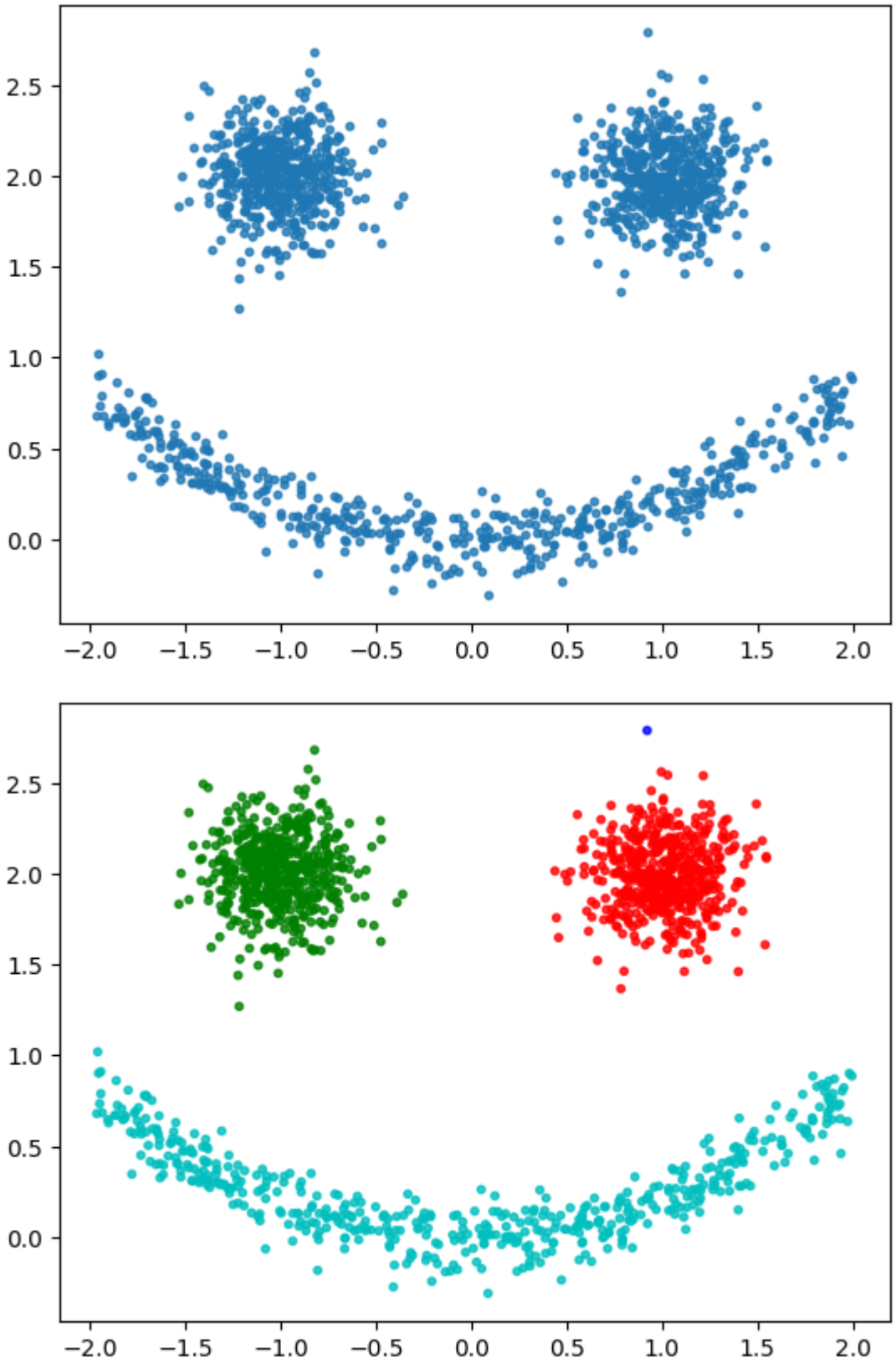
Hints:

- 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 [218…
```
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image as im
import sklearn.datasets as datasets
import matplotlib.patches as patches
```

```
# Create blobs for the eyes
eye1_x = np.random.normal(loc=-1, scale=0.2, size=500)
eye1_y = np.random.normal(loc=2, scale=0.2, size=500)

eye2_x = np.random.normal(loc=1, scale=0.2, size=500)
eye2_y = np.random.normal(loc=2, scale=0.2, size=500)

# Generate points for the smiley mouth
x_mouth = np.random.uniform(-2, 2, 500)  # Random x values in the interval [1, 5]
y_mouth = 0.2 * x_mouth**2 + np.random.normal(0, 0.1, 500)  # Generate y values

# Combine all data points
x = np.concatenate([eye1_x, eye2_x, x_mouth])
y = np.concatenate([eye1_y, eye2_y, y_mouth])
X = np.zeros((x.shape[0],2))
X[:,0] = x
X[:,1] = y
centers = [[0,0], [1,2], [-1,2]]
plt.scatter(X[:,0],X[:,1],s=10, alpha=0.8)
plt.show()
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 means not assigne
        self.snaps = []

    def snap(self,next_candidate):
        # print(5)
        TEMPFILE = "temp.png"

        fig, ax = plt.subplots()
        ax.scatter(X[:, 0], X[:, 1], c=self.assignments)
        center = (X[next_candidate,0],X[next_candidate,1])
        radius =  0.25
        circle = patches.Circle(center, radius, fill=False, edgecolor='black', linewid
        ax.add_patch(circle)
        ax.set_aspect('equal')
        fig.savefig(TEMPFILE)
        # print(TEMPFILE)
        plt.close()
        self.snaps.append(im.fromarray(np.asarray(im.open(TEMPFILE))))

    def distance(self,i,j):
        #returns 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,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: #already assigned
                #duplicates can occur when adding points in neighbourhoods that haven'
                continue
            self.assignments[next_candidate]=cluster_num
            self.snap(next_candidate)
            if self.is_core(next_candidate):
                next_neighborhood = self.get_neighborhood(next_candidate)
                neighbor_queue+=[i for i in next_neighborhood if self.assignments[i]==

        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

db = DBC(X, 3, .2)
clustering = db.dbscan()
colors = np.array([x for x in 'bgrcmykbgrcmykbgrcmykbgrcmyk'])
colors = np.hstack([colors] * 40)
plt.scatter(X[:, 0], X[:, 1], color=colors[clustering].tolist(), s=10, alpha=0.8)
plt.show()

images = db.snaps

images[0].save(
    'dbscan.gif',
    optimize=False,
    save_all=True,
    append_images=images[1:],
    loop=0,
    duration=0.01
)
```



In [219…

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

Out[219]:

```
<IPython.core.display.Image object>
```