Establishing Baseline Climate Partitions via Clustering

Simranjeet Bilkhu (400611361)

Introduction

"Climate change effects will be most acutely felt by future generations" (Boon, 2015). This fact may be easily ignored by certain governmental bodies, making it ever more important for the generations to come, to detect and visualize climate effects for policy makers. By establishing a baseline climate partition of a region, we may be able to visualize the effects of climate change by observing how these partitions evolve over time. In this paper we perform a cluster analysis on North American climate data averaged over the years 1970-2000. This will aid future research in determining how clusters have evolved over time with future data, specially by establishing a new baseline to compare to.

The data for this analysis is available here - a site that houses high spatial resolution global climate data.

The spatial resolution of the data is 10 arcminutes (roughly $320km^2$), and the data was truncated to only contain observations from North America. There were many missing observations which were removed altogther since most of missing observations were located in very large bodies of water like the Pacific and Atlantic Oceans. We also standardized the data before fitting any model. In total there are roughly 130000 observations left in the dataset. Here is a summary of the features included in the dataset:

- Measures of temperature including: mean annual temperature, standard deviation, Range, and average temperatures for the wettest, driest, warmest and coldest quarters.
- Measures of Precipitation including: Annual precipitation, least and most precipitation during a month, and average temperatures for the wettest, driest, coldest, and warmest quarters.
- Location (longitude and latitude)

Temperature measures are in Celsius, and precipitation measures are in millimeters. Numerical summaries of the features are available in the supplementary material.

Clustering Applications

KMeans Clustering

We first applied K-Means clustering. To find the ideal value of K we performed a silhouette analysis, deciding the number of clusters K based on the following criteria:

- Least amount of negative values for the silhouette coefficient
- Highest Sillouhette Score
- Similarly sized clusters

For both criteria, K=5 was the best choice for the number of clusters. Here is a visual representation of what the clusters look like in the spatial domain:

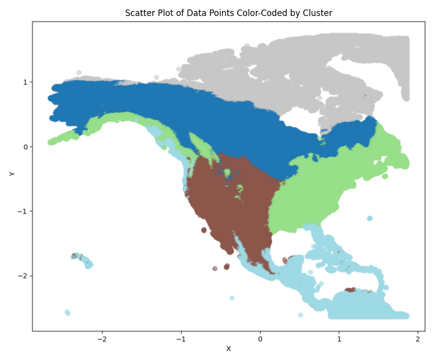


Figure 1: Cluster Coded Map of North America using KMeans

More details on the silhouette analysis are available in the supplementary material. We believe that stripe pattern seen in the plot above is due to spatial dependency in the features. We hope to have more reasonable looking clusters when we apply a spatial constraints in the next section.

Agglomerative Clustering With a Contiguity Constraint

In fitting the constrained agglomerative clustering model , we first chose the spatial constrain. We decided that points 170 kilometers from each other should belong to the same cluster. The reason being is that we might assume that areas that are within roughly 170 kilometers away from each other should receive similar weather. For reference Toronto and London Ontario are roughly 190 kilometers apart. Details on how this constraint was applied can be found in the supplementary material. We chose number of clusters in a similar fashion to the method used for KMeans, this time K=6 was the best choice. We plot the map of the clusters below:

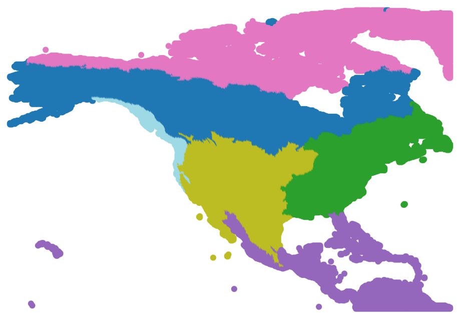


Figure 2: Cluster coded map of North America using Constrained Agglomerative Clustering

While the above plot appears to be an improvement on the results from KMeans, we do observe large clusters and small clusters together in the same plot which may be undesirable for weather related studies.

For the sake computational efficiency, a ward lineage was used to fit the model. More details on the silhouette plots are available in the supplementary material.

Principal Component Analysis with Constrained Hierarchical Clustering

We hope to improve upon the model above by using principal component analysis on the features and then using constrained hierarchical clustering. We first decide on the number of principal components to use by plotting the explained variance against the number of principal components.

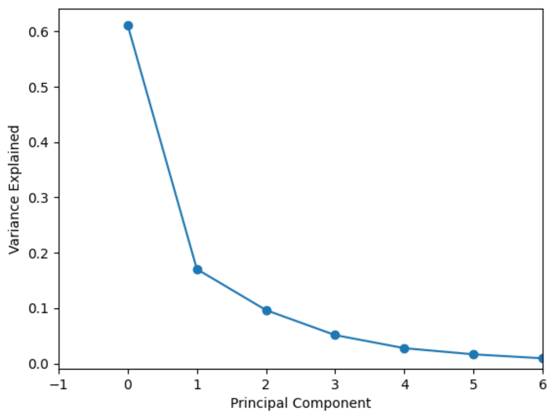


Figure 3: Number of principle components by Variance explained

The plot appears to flatten out after 2 principal components, thus we choose to fit a constrained agglomerative clustering model on those 2 principal components. We used the same contiguity constraint as the model above, and we chose the number of clusters in a similar manner - the best option this time was K=2. The figure below is a map of the clusters:

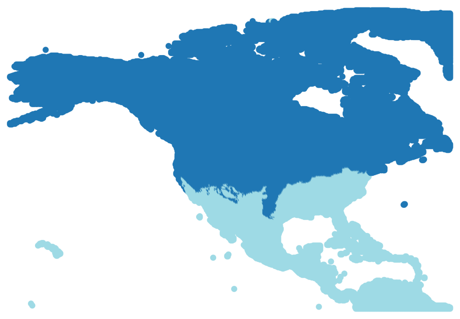


Figure 4: Cluster coded map of North America using Constrained Agglomerative Clustering after PCA

Model Comparison

We use the rand index to assess model similarity:

Model	Rand Score
KMeans vs Agglomerateive	0.88
Agglomerative vs Agglomerative (after PCA)	0.57
KMeans va Agglomerative (after PCA)	0.59

We see that KMeans and constrained agglomerative clustering (without PCA) agree well. The final model (with PCA) only somewhat agrees with the other two models.

References

"Agglomerative vs Divisive Hierarchical Clustering Explained." Accessed November 9, 2024. https://eyer.ai/blog/agglomerative-vs-divisive-hierarchical-clustering-explained/.

"Bioclimatic Variables — WorldClim 1 Documentation." Accessed November 8, 2024. https://www.worldclim.org/data/bioclim.html.

Boon, Helen J. "Climate Change Ignorance: An Unacceptable Legacy." The Australian Educational Researcher 42, no. 4 (September 1, 2015): 405–27. https://doi.org/10.1007/s13384-014-0156-x.

Chen, Daniel. "Chendaniely/Pyprojroot." Python, November 4, 2024. https://github.com/chendaniely/pyprojroot.

Demoitre, Ulderique. "Reference for Agglomerative Clustering Poor Performance." Forum post. Cross Validated, January 5, 2018. https://stats.stackexchange.com/q/321711.

Frees, Daniel. "Scaling Agglomerative Clustering for Big Data." Medium, August 30, 2023. https://towardsdatascience.com/scaling-agglomerative-clustering-for-big-data-an-introduction-to-rac-fb26a6b326ad.

Harris, Charles R., K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, et al. "Array Programming with NumPy." Nature 585, no. 7825 (September 2020): 357–62. https://doi.org/10.1038/s41586-020-2649-2.

Frees, Daniel. "Array Programming with NumPy." Nature 585, no. 7825 (September 2020): 357–62. https://doi.org/10.1038/s41586-020-2649-2.

Hunter, J. D. "Matplotlib: A 2D Graphics Environment." Computing in Science & Engineering 9, no. 3 (2007): 90–95. https://doi.org/10.1109/MCSE.2007.55.

Ibrahim.H. "Answer to 'How to Get Agglomerative Clustering "Centroid" in Python Scikit-Learn.' " Stack Overflow, March 12, 2020. https://stackoverflow.com/a/60653270.

"Nearest Neighbor Graph - an Overview | ScienceDirect Topics." Accessed November 18, 2024. https://www.sciencedirect.com/topics/computer-science/nearest-neighbor-graph.

Pandalove. "How to Get Agglomerative Clustering 'Centroid' in Python Scikit-Learn." Forum post. Stack Overflow, June 5, 2019. https://stackoverflow.com/q/56456572.

Pedregosa, F., G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, et al. "Scikit-Learn: Machine Learning in Python." Journal of Machine Learning Research 12 (2011): 2825–30.

Pedregosa, F., G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, et al. "Scikit-Learn: Machine Learning in Python." Journal of Machine Learning Research 12 (2011): 2825–30.

Pérez, Fernando, and Brian E. Granger. "IPython: A System for Interactive Scientific Computing." Computing in Science and Engineering 9, no. 3 (May 2007): 21–29. https://doi.org/10.1109/MCSE.2007.53.

Pedregosa, F., G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, et al. "IPython: A System for Interactive Scientific Computing." Computing in Science and Engineering 9, no. 3 (May 2007): 21–29. https://doi.org/10.1109/MCSE.2007.53.

Python documentation. "Glob — Unix Style Pathname Pattern Expansion." Accessed November 18, 2024. https://docs.python.org/3/library/glob.html.

r0f1. "Answer to 'Python Equivalent of R "Here" Package.' " Stack Overflow, July 13, 2019. https://stackoverflow.com/a/57019222.

r0f1. "Python Equivalent of R 'Here' Package." Forum post. Stack Overflow, February 15, 2019. https://stackoverflow.com/q/54706031.

Sumengen, Baris, Anand Rajagopalan, Gui Citovsky, David Simcha, Olivier Bachem, Pradipta Mitra, Sam Blasiak, Mason Liang, and Sanjiv Kumar. "Scaling Hierarchical Agglomerative Clustering to Billion-Sized Datasets." arXiv, May 25, 2021. https://doi.org/10.48550/arXiv.2105.11653.

The pandas development team. “Pandas-Dev/Pandas: Pandas.” Zenodo, September 2024. https://doi.org/10.5281/zenodo.13819579.

Virtanen, Pauli, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, et al. “SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python.” Nature Methods 17 (2020): 261–72. https://doi.org/10.1038/s41592-019-0686-2.

Waskom, M. L., (2021). seaborn: statistical data visualization. Journal of Open Source Software, 6(60), 3021, https://doi.org/10.21105/joss.03021.

Kelsey Jordahl, Joris Van den Bossche, Martin Fleischmann, Jacob Wasserman, James McBride, Jeffrey Gerard, … François Leblanc. (2020, July 15). geopandas/geopandas: v0.8.1 (Version v0.8.1). Zenodo. http://doi.org/10.5281/zenodo.3946761

Supplemetary Material

The code below was created with the help of GitHub Copilot. GitHub Copilot is an AI pair programmer that helps you write code faster and with less effort. For more information about GitHub Copilot, visit: https://github.com/features/copilot

Data Cleaning

```python
import glob
import rasterio

# Work on importing data
tif_files = glob.glob('**/*.tif', recursive=True)
rasters = []
transforms = []

for tif_file in tif_files:
    with rasterio.open(tif_file) as src:
        rasters.append(src.read(1))  # Read the first band of each tif file
        transforms.append(src.transform)  # Store the transform of each tif file

# Access the transform attribute of the second raster
transform = transforms[1]
print(transform)

# Dictionary to store dataframes for each band
dataframes = {}
summaries = {}

for i in range(0, 19):  # Bands 3 to 13
    raster = rasters[i]
    flattened_raster = raster.flatten()

    height, width = raster.shape
    x_coords, y_coords = np.meshgrid(np.arange(width), np.arange(height))
    x_coords_flat = x_coords.flatten()
    y_coords_flat = y_coords.flatten()

    lon, lat = rasterio.transform.xy(transform, y_coords_flat, x_coords_flat)

    # Create DataFrame for the current band
    df_raster = pd.DataFrame({
        f'Band_{i}': flattened_raster,
        'x': lon,
        'y': lat
    })

    # Filter and store in the dictionary
    df_filtered = df_raster[df_raster[f'Band_{i}'] >= -1000]
    dataframes[f'Band_{i}'] = df_filtered

    # Generate and store summary statistics
    summaries[f'Band_{i}'] = df_filtered.describe()
```

```
# Example of how to access the dataframes and summaries:
print(dataframes['Band_3'].head())  # Access dataframe for Band 3
print(summaries['Band_3'])         # Access summary statistics for Band 3


data_complete = {
    'x': dataframes['Band_0'].x, 'y': dataframes['Band_0'].y, 'Band_1':
     ↪ dataframes['Band_0'].Band_0, 'Band_2': dataframes['Band_1'].Band_1, 'Band_3':
     ↪ dataframes['Band_2'].Band_2,'Band_4': dataframes['Band_3'].Band_3, 'Band_5':
     ↪ dataframes['Band_4'].Band_4, 'Band_6': dataframes['Band_5'].Band_5, 'Band_7':
     ↪ dataframes['Band_6'].Band_6, 'Band_8': dataframes['Band_7'].Band_7, 'Band_9':
     ↪ dataframes['Band_8'].Band_8, 'Band_10': dataframes['Band_9'].Band_9, 'Band_11':
     ↪ dataframes['Band_10'].Band_10, 'Band_12': dataframes['Band_11'].Band_11,
     ↪ 'Band_13': dataframes['Band_12'].Band_12, 'Band_14':
     ↪ dataframes['Band_13'].Band_13, 'Band_15': dataframes['Band_14'].Band_14,
     ↪ 'Band_16': dataframes['Band_15'].Band_15, 'Band_17':
     ↪ dataframes['Band_16'].Band_16, 'Band_18': dataframes['Band_17'].Band_17,
     ↪ 'Band_19': dataframes['Band_18'].Band_18
}
data_completed = pd.DataFrame(data_complete)
data_completed.to_csv('data_cleaned')
data = pd.read_csv('data_cleaned')
# Define the bounds for North America
north_america_bounds = {
    'x_min': -168.0,  # Westernmost longitude
    'x_max': -52.6,   # Easternmost longitude
    'y_min': 5.0,     # Southernmost latitude
    'y_max': 83.1     # Northernmost latitude
}
# Filter the data to include only observations within the bounds of North America
north_america_data = data[(data['x'] >= north_america_bounds['x_min']) &
                          (data['x'] <= north_america_bounds['x_max']) &
                          (data['y'] >= north_america_bounds['y_min']) &
                          (data['y'] <= north_america_bounds['y_max'])]
# Display the first few rows of the truncated data



north_america_data.describe()
data = north_america_data
data_cleaned = data.rename(columns={
    'Band_1': 'Annual_Mean_T', 'Band_2': 'Mean_Diurnal_Range', 'Band_3': 'Isothermality',
     ↪ 'Band_4': 'Temp_Seasonality', 'Band_5': 'Max_Temperature_Month', 'Band_6':
     ↪ 'Min_Temperature_Month', 'Band_7': 'Temp_Annual_Range', 'Band_8':
     ↪ 'Mean_Temp_Wet_Quarter', 'Band_9': 'Mean_Temp_Dry_Quarter', 'Band_10':
     ↪ 'Mean_Temp_Warm_Quarter', 'Band_11': 'Mean_Temp_Cold_Quarter', 'Band_12':
     ↪ 'Annual_Precip', 'Band_13': 'Precip_Wet_Month', 'Band_14': 'Precip_Dry_Month',
     ↪ 'Band_15': 'Precip_Seasonality', 'Band_16': 'Precip_Wet_Quarter', 'Band_17':
     ↪ 'Precip_Dry_Quarter', 'Band_18': 'Precip_Warm_Quarter', 'Band_19':
     ↪ 'Precip_Cold_Quarter'
})

data_not_std = data_cleaned
```

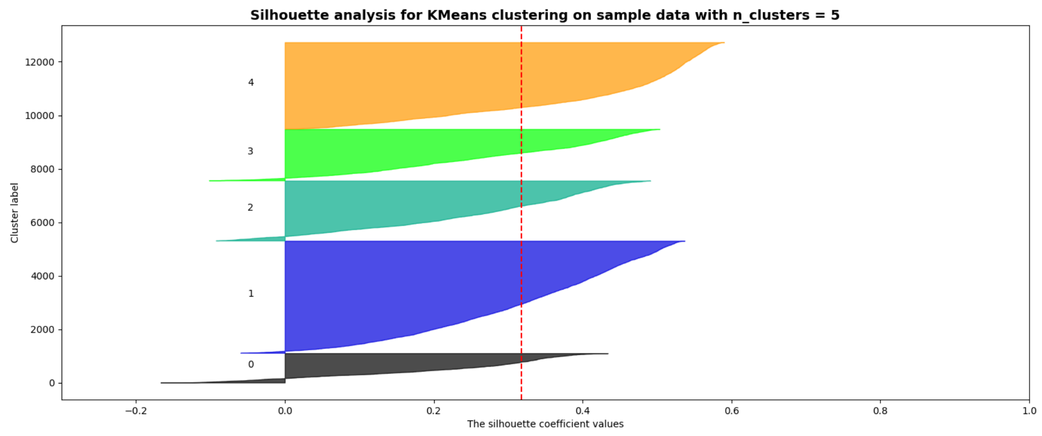
```
data_not_std.to_csv('non_standardized_data.csv')

data_cleaned.head()
from sklearn.preprocessing import StandardScaler
# Select columns to standardize
columns_to_standardize = data_cleaned.columns.difference(['x', 'y'])
# Initialize the scaler
scaler = StandardScaler()
# Fit and transform the data
data_cleaned[columns_to_standardize] =
 ↪  scaler.fit_transform(data_cleaned[columns_to_standardize])
data_cleaned['x_standardized'] = scaler.fit_transform(data_cleaned[['x']])
data_cleaned['y_standardized'] = scaler.fit_transform(data_cleaned[['y']])
# Print the standardized data
print(data_cleaned.std())
data_cleaned.to_csv('cleaned_data.csv')
```

KMeans (fitting and silhouette analysis)

```
df = pd.read_csv('cleaned_data.csv')
df = df.drop(df.columns[0], axis=1)
df = df.drop(columns=['x', 'y'])
# K = 2
range_n_clusters = [2, 3, 4, 5, 6, 7, 8, 9, 10]
for n_clusters in range_n_clusters:
    km = KMeans(n_clusters=n_clusters, n_init=20, random_state=0)
    km.fit(df)
    cluster_labels_km = km.predict(subsamp_df)
    # average silhouette score
    silhouette_avg_km = silhouette_score(subsamp_df, cluster_labels_km)
    # compute the silhouette scores for each sample
    sample_silhouette_values = silhouette_samples(subsamp_df, cluster_labels_km)
    fig, ax1 = plt.subplots(1, 1)
    fig.set_size_inches(18, 7)
    ax1.set_xlim([-0.3, 1])# change this based on the silhouette range
    y_lower = 10
    for i in range(n_clusters):
        # Aggregate the silhouette scores for samples belonging to
        # cluster i, and sort them
        ith_cluster_silhouette_values = sample_silhouette_values[cluster_labels_km == i]
        ith_cluster_silhouette_values.sort()
        size_cluster_i = ith_cluster_silhouette_values.shape[0]
        y_upper = y_lower + size_cluster_i
        color = cm.nipy_spectral(float(i) / n_clusters)
        ax1.fill_betweenx(
            y=np.arange(y_lower, y_upper),
            x1=0,
            x2=ith_cluster_silhouette_values,
            facecolor=color,
            edgecolor=color,
            alpha=0.7,
        )
        # label the silhouette plots with their cluster numbers at the middle
        ax1.text(-0.05, y_lower + 0.5 * size_cluster_i, str(i))
        # Compute the new y_lower for next cluster silhouette scores
        y_lower = y_upper + 10
    ax1.set_title("The silhouette plot for various cluster")
    ax1.set_xlabel("The silhouette coefficient values")
    ax1.set_ylabel("Cluster label")
    # vertical line for average silhouette score of all the values
    ax1.axvline(x=silhouette_avg_km, color="red", linestyle="--")
    plt.title(
        "Silhouette analysis for KMeans clustering on sample data with n_clusters = %d"
        % n_clusters,
        fontsize=14,
        fontweight="bold",
    )
plt.show()
```

The resulting silhouette plots show that K = 5 was the best choice (it was the only choice with no negative silhouette coefficient values)



Note on sillouhette analysis: To ease the computational burden, we computed sillouhette scores on a subsample of the data.

Fitting the model and spatial plot (KMeans)

```
km6 = KMeans(n_clusters=6, n_init=20, random_state=0)
km6.fit(df)
cluster_labels_km6 = km6.predict(subsamp_df)
# Assuming df has 'x_standardized', 'y_standardized', and 'cluster' columns
test_labels5 = km5.fit_predict(df)
df['cluster'] = test_labels5  # Ensure that cluster labels are in the DataFrame
# Create the plot
fig, ax = plt.subplots(figsize=(10, 8))
# Scatter plot, color-coded by cluster
scatter = ax.scatter(df_spare['x_standardized'], df_spare['y_standardized'],
 ↪  c=df['cluster'], cmap='tab20', alpha=0.6)
# Set axis labels
ax.set_xlabel('X')
ax.set_ylabel('Y')
# Optionally, set title
ax.set_title('Scatter Plot of Data Points Color-Coded by Cluster')
# Show the plot
plt.savefig('kmeans_scatter.png')
plt.show()
```

Agglomerative Clustering (fitting and silhouette analysis)

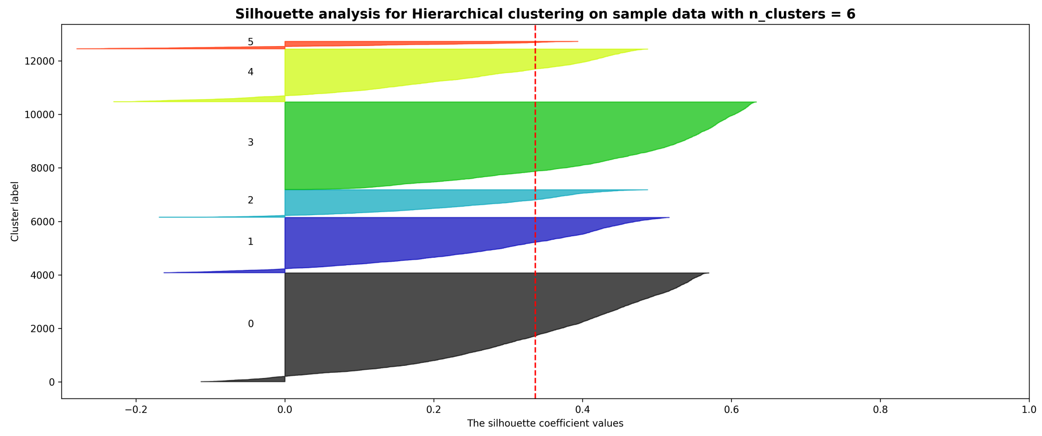
```
from sklearn.metrics.cluster import rand_score
from sklearn.metrics import pairwise_distances_argmin_min
df = pd.read_csv('cleaned_data.csv')
df.columns
df = df.drop(columns = ['x_standardized', 'y_standardized'])
# Computind the distance matrics
geometry = [Point(xy) for xy in zip(df['x'], df['y'])]
sf_data = gpd.GeoDataFrame(df, geometry=geometry)
sf_data = sf_data.drop(columns = ['x', 'y'])
knn = 10
X_coord = np.array(list(sf_data.geometry.astype(object).apply(lambda geom: (geom.x,
↪  geom.y))))
knn_graph = kneighbors_graph(X_coord, n_neighbors=knn, include_self=False)
sf_data.shape
# Drop the geometry column to create the features dataframe
features_df = sf_data.drop(columns=['geometry'])
def agglomerative_clustering_predict(train_data, train_labels, test_data):
    """
    Predict clusters for new data points using NearestCentroid.

    Parameters:
    - train_data (pd.DataFrame or np.array): Training dataset used for clustering.
    - test_data (pd.DataFrame or np.array): New data points to predict clusters for.
    - n_clusters (int): Number of clusters to form.
    - linkage (str): Linkage criterion to use for the clustering algorithm (e.g., 'ward',
↪   'complete').
    - connectivity (array-like, optional): Connectivity matrix for structured clustering.

    Returns:
    - test_labels (np.array): Predicted cluster labels for the test data.
    """
    # Fit NearestCentroid on the training data to find centroids of clusters
    centroid_model = NearestCentroid()
    centroid_model.fit(train_data, train_labels)

    # Predict cluster labels for test_data based on the nearest centroid
    test_labels = centroid_model.predict(test_data)

    return test_labels
range_n_clusters = [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]
for n_clusters in range_n_clusters:
    ag = AgglomerativeClustering(linkage="ward", connectivity= knn_graph,n_clusters =
↪   n_clusters)
    cluster_labels_ag = ag.fit_predict(features_df)
    test_labels_ag = agglomerative_clustering_predict(features_df,  cluster_labels_ag,
 ↪  subsamp_df)
    # average silhouette score
    silhouette_avg_ag = silhouette_score(subsamp_df, test_labels_ag)
    # compute the silhouette scores for each sample
    sample_silhouette_values = silhouette_samples(subsamp_df, test_labels_ag)
    fig, ax1 = plt.subplots(1, 1)
    fig.set_size_inches(18, 7)
    ax1.set_xlim([-0.3, 1])# change this based on the silhouette range
    y_lower = 10
    for i in range(n_clusters):
        # Aggregate the silhouette scores for samples belonging to
        # cluster i, and sort them
        ith_cluster_silhouette_values = sample_silhouette_values[test_labels_ag == i]
        ith_cluster_silhouette_values.sort()
        size_cluster_i = ith_cluster_silhouette_values.shape[0]
        y_upper = y_lower + size_cluster_i
        color = cm.nipy_spectral(float(i) / n_clusters)
        ax1.fill_betweenx(
            y=np.arange(y_lower, y_upper),
            x1=0,
            x2=ith_cluster_silhouette_values,
            facecolor=color,
            edgecolor=color,
            alpha=0.7,
        )
        # label the silhouette plots with their cluster numbers at the middle
        ax1.text(-0.05, y_lower + 0.5 * size_cluster_i, str(i))
        # Compute the new y_lower for next cluster silhouette scores
        y_lower = y_upper + 10
    ax1.set_title("The silhouette plot for various cluster")
    ax1.set_xlabel("The silhouette coefficient values")
    ax1.set_ylabel("Cluster label")
    # vertical line for average silhouette score of all the values
    ax1.axvline(x=silhouette_avg_ag, color="red", linestyle="--")
    plt.title(
        "Silhouette analysis for Hierarchical clustering on sample data with n_clusters =
         ↪  %d"
        % n_clusters,
        fontsize=14,
        fontweight="bold",
    )
plt.savefig('Agglom_sil.png', dpi = 300)
plt.show()
```



Fitting the model and spatial plot

```
sf_data['cluster'] = cluster_labels_ag6
fig, ax = plt.subplots(figsize=(10, 8))
sf_data.plot(
    ax=ax,
    column='cluster',
    cmap='tab20',
    legend=False
)
ax.set_axis_off()
plt.savefig('agg_space.png', dpi = 300)
plt.show()
```

Model with PCA

```
from sklearn.preprocessing import scale
from sklearn.decomposition import PCA, TruncatedSVD
df = pd.read_csv('cleaned_data.csv')
subsamp = df.sample(frac=0.1, random_state=0)
df = df.drop(df.columns[0], axis=1)
df = df.drop(columns = ['x_standardized', 'y_standardized'])
df_fet = df.drop(columns = ['x', 'y'])
pca_comp = PCA()
plot3 = pd.DataFrame(pca_comp.fit_transform(df_fet))
fig, ax1 = plt.subplots()
ax1.plot(pca_comp.explained_variance_ratio_, '-o')
ax1.set_ylabel('Variance Explained')
ax1.set_ylim(ymin=-0.01)
for ax in fig.axes:
    ax.set_xlabel('Principal Component')
    ax.set_xlim(-1, 6)
fig.savefig('pca_plot.png')
subsamp_df = data_pca_df.sample(frac=0.05, random_state=0)
# Computind the distance matrics
geometry = [Point(xy) for xy in zip(df['x'], df['y'])]
```

```
sf_data = gpd.GeoDataFrame(data_pca_df, geometry=geometry)
knn = 10
X_coord = np.array(list(sf_data.geometry.astype(object).apply(lambda geom: (geom.x,
 ↪  geom.y))))
knn_graph = kneighbors_graph(X_coord, n_neighbors=knn, include_self=False)
sf_data.shape
# Drop the geometry column to create the features dataframe
features_df = sf_data.drop(columns=['geometry'])
def agglomerative_clustering_predict(train_data, train_labels, test_data):
    """
    Predict clusters for new data points using NearestCentroid.

    Parameters:
    - train_data (pd.DataFrame or np.array): Training dataset used for clustering.
    - test_data (pd.DataFrame or np.array): New data points to predict clusters for.
    - n_clusters (int): Number of clusters to form.
    - linkage (str): Linkage criterion to use for the clustering algorithm (e.g., 'ward',
 ↪  'complete').
    - connectivity (array-like, optional): Connectivity matrix for structured clustering.
    Returns:
    - test_labels (np.array): Predicted cluster labels for the test data.
    """
    # Fit NearestCentroid on the training data to find centroids of clusters
    centroid_model = NearestCentroid()
    centroid_model.fit(train_data, train_labels)
    # Predict cluster labels for test_data based on the nearest centroid
    test_labels = centroid_model.predict(test_data)
    return test_labels
range_n_clusters = [2, 3, 4, 5, 6, 7, 8, 9]
for n_clusters in range_n_clusters:
    ag = AgglomerativeClustering(linkage="ward", connectivity= knn_graph,n_clusters =
 ↪  n_clusters)
    cluster_labels_ag = ag.fit_predict(features_df)
    test_labels_ag = agglomerative_clustering_predict(features_df,  cluster_labels_ag,
 ↪  subsamp_df)
    # average silhouette score
    silhouette_avg_ag = silhouette_score(subsamp_df, test_labels_ag)
    # compute the silhouette scores for each sample
    sample_silhouette_values = silhouette_samples(subsamp_df, test_labels_ag)
    fig, ax1 = plt.subplots(1, 1)
    fig.set_size_inches(18, 7)
    ax1.set_xlim([-0.3, 1])# change this based on the silhouette range
    y_lower = 10
    for i in range(n_clusters):
        # Aggregate the silhouette scores for samples belonging to
        # cluster i, and sort them
            ith_cluster_silhouette_values = sample_silhouette_values[test_labels_ag == i]
 ↪  ith_cluster_silhouette_values.sort()
        size_cluster_i = ith_cluster_silhouette_values.shape[0]
        y_upper = y_lower + size_cluster_i
        color = cm.nipy_spectral(float(i) / n_clusters)
        ax1.fill_betweenx(
            y=np.arange(y_lower, y_upper),
            x1=0,
            x2=ith_cluster_silhouette_values,
            facecolor=color,
            edgecolor=color,
            alpha=0.7,
        )
        # label the silhouette plots with their cluster numbers at the middle
        ax1.text(-0.05, y_lower + 0.5 * size_cluster_i, str(i))
       # Compute the new y_lower for next cluster silhouette scores
        y_lower = y_upper + 10
    ax1.set_title("The silhouette plot for various cluster")
    ax1.set_xlabel("The silhouette coefficient values")
    ax1.set_ylabel("Cluster label")
    # vertical line for average silhouette score of all the values
    ax1.axvline(x=silhouette_avg_ag, color="red", linestyle="--")
    plt.title(
        "Silhouette analysis for Hierarchical clustering on sample data with n_clusters =
         ↪  %d"
        % n_clusters,
        fontsize=14,
        fontweight="bold",
    )
plt.savefig('pca_sil.png', dpi = 300)
plt.show()
# plotting
sf_data['cluster'] = cluster_labels_ag2
fig, ax = plt.subplots(figsize=(10, 8))
sf_data.plot(
    ax=ax,
    column='cluster',
    cmap='tab20',
    legend=False
)
ax.set_axis_off()
plt.savefig('pca_space.png', dpi = 300)
plt.show()
cluster_labels_ag2 = fit_agglomerative_clustering(2)
# plotting
sf_data['cluster'] = cluster_labels_ag2
fig, ax = plt.subplots(figsize=(10, 8))
sf_data.plot(
    ax=ax,
    column='cluster',
    cmap='tab20',
    legend=False
)
ax.set_axis_off()
plt.savefig('pca_space.png', dpi = 300)
plt.show()
```

PCA Loadings

```python
pca_df_fet = PCA(n_components=2)

data_pca = pca_df_fet.fit_transform(df_fet)
data_pca_df = pd.DataFrame(data_pca, columns=['PC1', 'PC2'])
loadings = pca_df_fet.components_
loadings_df = pd.DataFrame(loadings, columns=df_fet.columns, index=['PC1', 'PC2'])
loadings_df = loadings_df.round(2)
```

Model Comparison

```python
df = pd.read_csv('cleaned_data.csv')
df = df.drop(df.columns[0], axis=1)
df_agg = df.drop(columns = ['x_standardized', 'y_standardized'])
df_km = df.drop(columns = ['x', 'y'])
# Computind the distance matrics
geometry = [Point(xy) for xy in zip(df_agg['x'], df_agg['y'])]
sf_data = gpd.GeoDataFrame(df_agg, geometry=geometry)
sf_data = sf_data.drop(columns = ['x', 'y'])
knn = 10
X_coord = np.array(list(sf_data.geometry.astype(object).apply(lambda geom: (geom.x,
 ↪  geom.y))))
knn_graph = kneighbors_graph(X_coord, n_neighbors=knn, include_self=False)
sf_data.shape
# Drop the geometry column to create the features dataframe
features_df = sf_data.drop(columns=['geometry'])
pca_df_fet = PCA(n_components=1)
data_pca = pca_df_fet.fit_transform(df_km)
data_pca_df = pd.DataFrame(data_pca, columns=['PC1'])
ag6 =AgglomerativeClustering(linkage="ward", connectivity= knn_graph,n_clusters = 6)
ag2.fit(features_df)
# This is the model in part b)
km5 = KMeans(n_clusters = 5, n_init = 20, random_state = 0)
km2.fit(df_km)
pca_ag2 = AgglomerativeClustering(linkage='ward', n_clusters=2)
# This is the model in part c)
pca_agg_model = AgglomerativeClustering(linkage = "ward", n_clusters=2, connectivity=
 ↪  knn_graph)
pca_ag2_labs = pca_agg_model.fit_predict(data_pca)
# Getting the rand index
comparison1 = rand_score(ag2.labels_, km2.labels_)
comparison2 = rand_score(ag2.labels_, pca_ag2_labs)
comparison3 = rand_score(km2.labels_, pca_ag2_labs)
print(f"Agglomerative VS KMeans: {comparison1}")
print(f"Agglomerative VS Agglomerative with PCA: {comparison2}")
print(f"KMeans VS Agglomerative with PCA: {comparison3}")
```

Numerical Summaries

```python
import pandas as pd
numerical_summeries_data = pd.read_csv('non_standardized_data.csv')
numerical_summeries_data = numerical_summeries_data.drop(['Unnamed: 0.1', 'Unnamed: 0'],
 ↪  axis=1)
summary = numerical_summeries_data.describe().transpose()
summary.style.format("{:.2f}").background_gradient(cmap='viridis')
```

Table 2

	count	mean	std	min	25%	50%	75%	max
x	126760.00	-100.22	25.47	-167.92	-115.75	-98.75	-80.92	-52.75
y	126760.00	52.30	17.89	5.08	41.08	54.58	65.92	83.08
Annual_Mean_T	126760.00	0.71	12.71	-26.76	-8.98	-0.59	9.19	29.39
Mean_Diurnal_Range	126760.00	10.19	3.37	1.00	7.28	10.23	12.29	21.15
Isothermality	126760.00	29.67	16.98	9.13	16.90	24.67	36.38	100.00
Temp_Seasonality	126760.00	1069.73	421.16	0.00	817.99	1171.06	1401.02	1739.38
Max_Temperature_Month	126760.00	20.91	9.48	-6.25	15.03	21.16	29.00	44.69
Min_Temperature_Month	126760.00	-18.32	16.34	-46.76	-31.22	-22.88	-7.64	24.70
Temp_Annual_Range	126760.00	39.24	10.67	1.00	34.57	41.53	47.09	56.29
Mean_Temp_Wet_Quarter	126760.00	11.04	8.93	-24.21	4.99	10.85	16.84	33.51
Mean_Temp_Dry_Quarter	126760.00	-7.67	18.21	-41.33	-22.60	-9.91	5.60	29.84
Mean_Temp_Warm_Quarter	126760.00	13.90	8.50	-9.82	8.49	13.60	20.30	35.14
Mean_Temp_Cold_Quarter	126760.00	-12.12	16.77	-42.46	-25.63	-15.80	-1.20	28.16
Annual_Precip	126760.00	635.20	565.30	11.00	252.00	451.00	863.00	11191.00
Precip_Wet_Month	126760.00	95.38	82.03	2.00	44.00	76.00	110.00	2328.00
Precip_Dry_Month	126760.00	23.24	25.55	0.00	6.00	14.00	30.00	484.00
Precip_Seasonality	126760.00	51.64	20.58	5.52	38.28	53.19	64.27	138.00
Precip_Wet_Quarter	126760.00	255.04	220.73	6.00	113.00	202.00	305.00	5282.00
Precip_Dry_Quarter	126760.00	81.15	85.99	0.00	24.00	50.00	106.00	1507.00
Precip_Warm_Quarter	126760.00	195.13	126.45	2.00	97.00	177.00	265.00	5282.00
Precip_Cold_Quarter	126760.00	124.20	176.91	0.00	31.00	60.00	146.00	1980.00