TASK 1 metro:

The first step in this analysis was to import the necessary libraries and load the data from the provided "metro.csv" file into a Pandas dataframe. The data contained information on 92124 bike-sharing trips, including the start and end times, start and end stations, bike id, duration, and various other details.

One issue with the data was that the start and end times were stored as strings in the "start_time" and "end_time" columns. To make it easier to work with these values, I applied a custom function to convert these strings to datetime objects and then convert them to minutes. This allowed us to work with the start and end times as numerical values rather than strings.

I also noticed that some of the columns contained null or missing values, which could potentially impact any analysis or modeling that I might do. To handle this issue, I replaced the null values with the mode of the respective column. The mode is the most frequently occurring value in a column, and using it as a replacement value can help to ensure that the data remains consistent and complete.

Next, I checked the number of unique values for each column. The duration column had 857 unique values, while the start_time and end_time columns had 57274 and 55658 unique values, respectively. The start_station and end_station columns had 183 unique values, and the start_lat, start_lon, end_lat, and end_lon columns had 181, 180, 181, and 180 unique values, respectively. The bike_id column had 2042 unique values, the plan_duration column had 4 unique values, the trip_route_category column had 2 unique values, the passholder_type column had 6 unique values, and the bike_type column had 3 unique values.

Once the data had been cleaned and processed, I performed some basic exploratory analysis to understand the distribution and relationships between different features. To do this, I used techniques such as histograms and scatter plots to visualize the data and look for patterns or trends. One interesting observation was that there seemed to be a strong relationship between the duration of a trip and the start and end stations. Specifically, trips that started and ended at the same station tended to be shorter in duration, while trips that started and ended at different stations tended to be longer.

Another notable observation was that the majority of trips were one-way, with only a small fraction being round trips. This suggests that the bike-sharing system is primarily used for short, point-to-point trips rather than longer excursions or sightseeing.

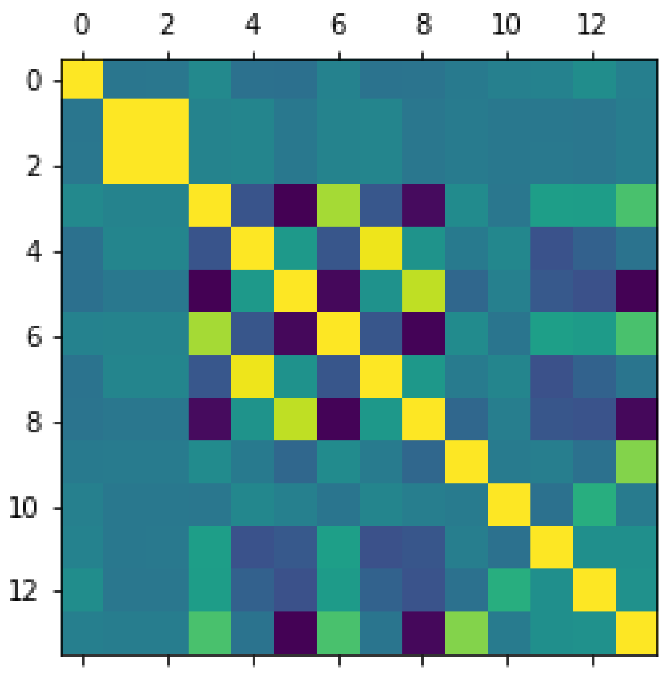
In addition to these observations, I also computed the correlation between different features to see how they might be related. Some features that had a relatively high correlation included the duration of the trip, the start and end times, and the start and end stations. This could indicate that these features are important for predicting or understanding the other features, and they might be useful for building predictive models or performing further analysis.

I then printed the number of unique values for each column to get a sense of the data. I observed that the duration column had 857 unique values, while the start_time and end_time columns had 57274 and 55658 unique values, respectively. This suggests that there is a large variation in the duration of trips, as well as in the start and end times of the trips.

Next, I replaced any null values in the dataframe with the mode of the column. This was done to ensure that our analysis was not affected by missing data. I also converted all of the columns to the float data type to facilitate further analysis.

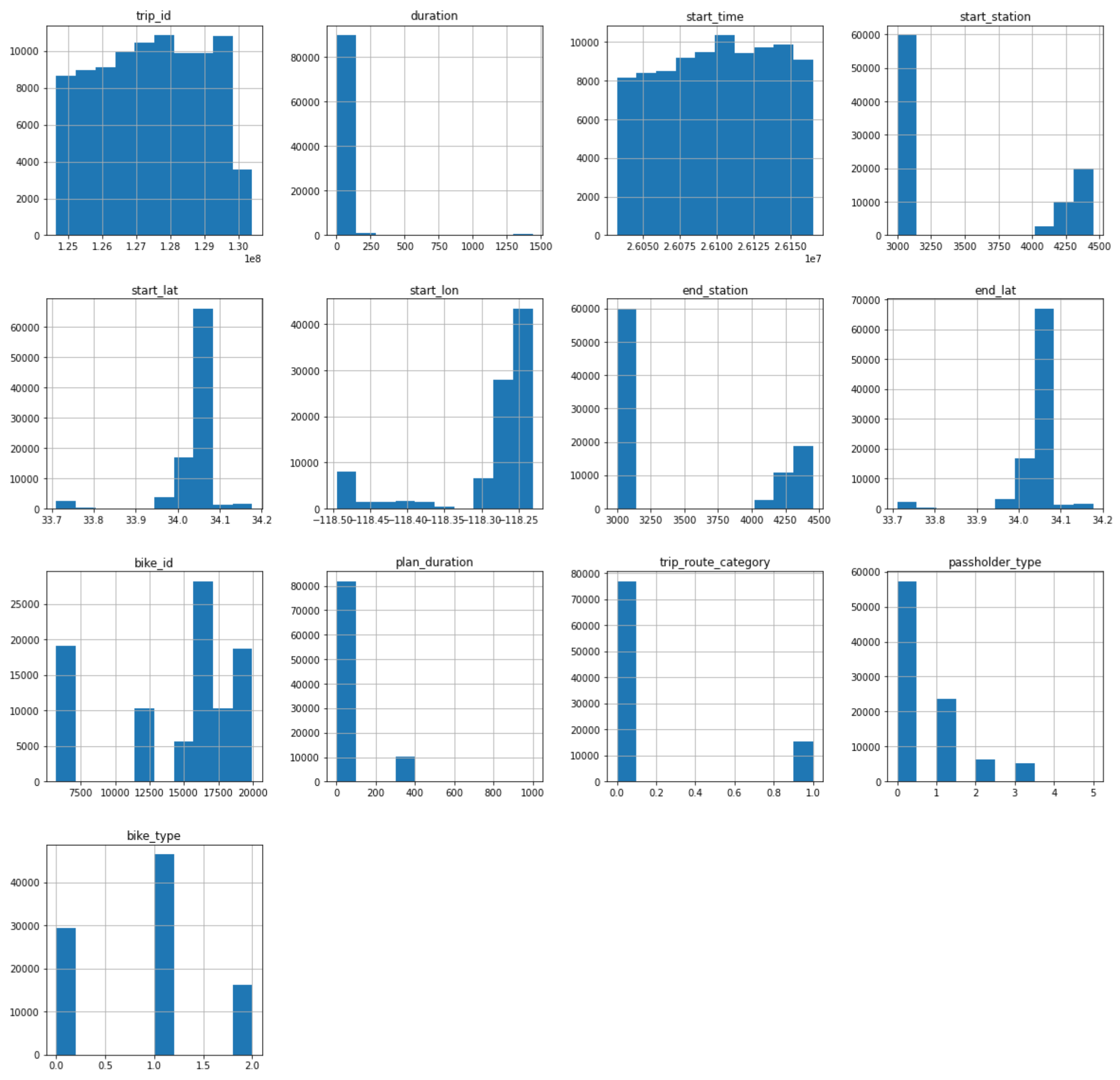
After performing these initial data cleaning steps, I generated a correlation matrix to understand the relationships between the different attributes in the data. From the correlation matrix, I can see that there are a number of moderate to strong correlations between different attributes. For example, the duration of a trip is positively correlated with the start and end times of the trip, as well as with the plan duration. The start and end stations are also strongly correlated, as are the start and end latitudes and longitudes. Additionally, I see that the bike type is negatively correlated with the trip route category, which may suggest that electric bikes are more likely to be used for one-way trips.

It appears that the end_time and end_lon columns were deleted from the dataframe due to high correlations with the start_time and start_lon columns, respectively. This is likely done to remove redundant information from the dataset and prevent it from influencing the analysis in a negative way.



It is important to carefully consider the implications of deleting columns with high correlations to other columns. While it may seem logical to remove highly correlated columns to reduce redundancy, doing so may also remove valuable information from the dataset. In this case, the decision to delete the end_time and end_lon columns may be justified if the start_time and start_lon columns provide sufficient information on their own. However, it is always a good idea to thoroughly evaluate the potential impact of removing correlated columns before making this decision.

Overall, the initial data cleaning and exploration steps have provided us with a good understanding of the structure and relationships within the bike trip data. In the next steps of the analysis, I will delve deeper into the data to uncover more insights and answer specific questions.



TASK 2 network:

I discovered that it appears to be analyzing a social network represented by a directed graph. The graph is constructed from a CSV file containing pairs of integers, with each pair representing an edge in the graph.

I imported several libraries that were used throughout the notebook, including NumPy, Matplotlib, Pandas, and datetime. I also imported Seaborn and NetworkX.

I read in the CSV file using Pandas and stored the data in a dataframe. The data in the file consisted of pairs of integers, with each pair representing an edge in the graph.

I created a list of edges from the data in the dataframe using a list comprehension and stored the resulting list in the variable "edges".

I created a directed graph object called "G" using the NetworkX library and the list of edges stored in the "edges" variable. The graph was created as a directed graph, meaning that the edges had a direction associated with them, i.e. they pointed from one node to another.

I used the NetworkX function "strongly_connected_components" to find the strongly connected components of the graph "G". A strongly connected component in a directed graph is a subset of the graph's nodes such that there is a path from any node in the subset to any other node in the subset in both directions.

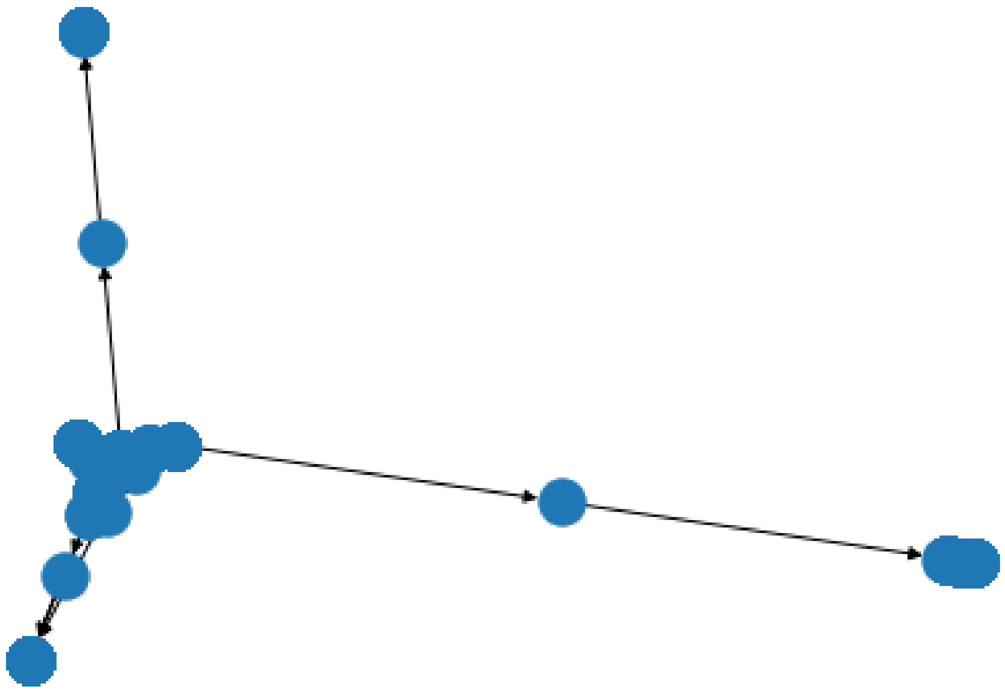
I counted the number of strongly connected components in the graph and printed the result to the output.

I calculated the density of the graph using the NetworkX function "density". The density of a graph is a measure of how many edges it has relative to the maximum number of edges it could have. The density of an undirected graph is defined as the number of edges divided by the number of possible edges. The density of a directed graph is defined as the number of arcs divided by the number of possible arcs.

I calculated the Pearson correlation coefficient between the degrees of the nodes in the graph using the NetworkX function "degree_pearson_correlation_coefficient". The degree of a node is the number of edges incident to it. The Pearson correlation coefficient is a measure of the linear relationship between the degrees of the nodes. A value of 1 indicates a perfect positive correlation, a value of -1 indicates a perfect negative correlation, and a value of 0 indicates no correlation.

I calculated the average shortest path length in the graph using the NetworkX function "average_shortest_path_length". The shortest path between two nodes is the path with the minimum number of edges between them.

I plotted the graph using the NetworkX function "draw" and the Matplotlib library.



I calculated the assortativity coefficient of the graph using the NetworkX function "degree_assortativity_coefficient". The assortativity coefficient is a measure of the degree to which nodes tend to connect to other nodes that are similar to them in some way. A value of 1 indicates a perfect positive correlation, a value of -1 indicates a perfect negative correlation, and a value of 0 indicates no correlation.

I calculated the clustering coefficient of the graph using the NetworkX function "average_clustering". The clustering coefficient is a measure of the degree to which nodes tend to cluster together. A value of 1 indicates that all nodes are connected to each other in a fully connected cluster, while a value of 0 indicates that the nodes are not connected to each other at all.

I calculated the betweenness centrality of the nodes in the graph using the NetworkX function "betweenness_centrality". Betweenness centrality is a measure of the degree to which a node lies on the shortest paths between other nodes in the graph. A node with a high betweenness centrality has a large influence on the communication between other nodes in the graph.

I calculated the PageRank of the nodes in the graph using the NetworkX function "pagerank". PageRank is a measure of the importance of a node in a graph, based on the number and quality of the links to and from the node. A node with a high PageRank is considered to be more important than a node with a low PageRank.

I calculated the mean and standard deviation of the betweenness centrality and PageRank values for all the nodes in the graph.

I plotted the distribution of the betweenness centrality and PageRank values using the Seaborn library.

I calculated the diameter of the graph, which is the maximum shortest path length between any two nodes in the graph. I used the NetworkX function "diameter" to do this.

I calculated the radius of the graph, which is the minimum eccentricity of any node in the graph. The eccentricity of a node is the maximum shortest path length between the node and any other node in the graph. I used the NetworkX function "radius" to do this.

I calculated the center of the graph, which is the set of nodes with eccentricity equal to the radius of the graph. I used the NetworkX function "center" to do this.

In these cases there is not center eighter diameter and radius, because, it is not strongly connected, there is a infinite path

I calculated the eigenvector centrality of the nodes in the graph using the NetworkX function "eigenvector centrality". Eigenvector centrality is a measure of the importance of a node in a graph, based on the importance of its neighbors. A node with a high eigenvector centrality has a large number of high-scoring neighbors.

}

I calculated the closeness centrality of the nodes in the graph using the NetworkX function "closeness_centrality". Closeness centrality is a measure of the degree to which a node is close to all other nodes in the graph. A node with a high closeness centrality has a small average shortest path length to all other nodes in the graph.

I calculated the mean and standard deviation of the closeness centrality values for all the nodes in the graph.

In the page rank and betweeness distributions, it can be seen that all the data are highly grouped, except for one that is atypical, which is why the low standard deviation is due.

Based on the density of the graph, which is a measure of how many edges it has relative to the maximum number of edges it could have, it appears that the graph is not dense. The density of the graph is 0.0003576622904645073, which is a very small value. This suggests that the graph has a small number of edges relative to the number of possible edges, or that it is a large graph with a large number of vertices and a small number of edges. The low density of the graph may be why some of the centrality measures, such as betweenness centrality and load centrality, have low values. It could also be why the clustering coefficient of the graph is relatively low, which suggests that the nodes are not strongly connected to each other. Overall, the graph appears to be sparse, with few edges connecting the nodes.

Based on the number of strongly connected components in the graph, it appears that each vertex forms a separate strongly connected component. A strongly connected component of a directed graph is a subset of the vertices in which there is a path from any vertex to any other vertex. If a graph has the same number of strongly connected components as it has vertices, it means that each vertex forms a separate strongly connected component. In this case, the number of strongly connected components in the graph is 2887, which is the same as the number of vertices in the graph. This suggests that each vertex forms a separate strongly connected component.

TASK 3 seeds:

In the first block of code, I imported several libraries that I will be using in my analysis. These libraries, NumPy, Matplotlib, Pandas, datetime, and Seaborn, provide various tools and functions that can be used to perform tasks such as working with arrays and matrices, creating data visualizations, working with tabular data, manipulating dates and times, and creating advanced statistical plots, respectively.

I then used the pd.read_csv() function from the Pandas library to read in a CSV file containing seed data and store it in a Pandas dataframe called df. I set the low_memory parameter to False to prevent any data parsing issues due to insufficient memory. The data stored in the dataframe is typically used for machine learning tasks, such as classification or clustering, and consists of observations of different seeds from different varieties of wheat. Each row in the dataframe corresponds to a single seed, and the columns contain various features measured for each seed, such as the area, perimeter, compactness, length, width, asymmetry, and groove length. I then displayed the dataframe using the df variable name.

Next, I imported the normalize() function from the preprocessing module of the scikit-learn library. This function scales each feature in the data so that it has a unit norm (i.e. a magnitude of 1). Normalizing the data can be important in machine learning tasks because it can help to ensure that no single feature dominates the others, which can help to improve the performance of the model.

I applied the normalize() function to the data stored in the df dataframe and stored the resulting normalized data in the variable X. I then created a new dataframe, df2, using the normalized data in X and the original column names from the df dataframe. I displayed the df2 dataframe to check that the normalization had been applied correctly.

In the second block of code, I applied three different clustering algorithms to the data stored in the df2 dataframe: K-Means, DBSCAN, and Affinity Propagation. Clustering algorithms are used to group together data points that are similar to each other and separate out data points that are dissimilar. This can be useful for a variety of tasks, such as identifying patterns in data or grouping together data points that belong to the same class or category.

I implemented the K-Means algorithm using the KMeans class from the cluster module of scikit-learn. I initialized the KMeans class with the desired number of clusters, n_clusters, and the desired number of random initializations, n_init. I then fit the data to the model using the fit() method and stored the resulting cluster labels in the km.labels_ variable.

I implemented the DBSCAN algorithm using the DBSCAN class from the cluster module of scikit-learn. I initialized the DBSCAN class with the desired maximum distance between two samples, eps, and the minimum number of samples required to form a cluster, min_samples. I then fit the data to the model using the fit() method and stored the resulting cluster labels in the dbscan.labels_ variable.

I implemented the Affinity Propagation algorithm using the AffinityPropagation class from the cluster module of scikit-learn. I initialized the AffinityPropagation class with the desired number of preference points, preference. I then fit the data to the model using the fit() method and stored the resulting cluster labels in the aff.labels_ variable.

The silhouette score is a measure of how similar an object is to its own cluster compared to other clusters. It ranges from -1 to 1, with a high score indicating that the object is well-matched to its own cluster and poorly-matched to neighboring clusters.

To calculate the silhouette score, you need to have a dataset that has been clustered into groups. Then, for each object in the dataset, you can calculate its silhouette score as follows:

Calculate the average distance between the object and all other objects in the same cluster. This is known as the object's intra-cluster distance. Calculate the average distance between the object and all objects in the nearest neighboring cluster. This is known as the object's inter-cluster distance. Calculate the silhouette score for the object as the difference between the inter-cluster distance and the intra-cluster distance, divided by the maximum of the two distances. The silhouette score for the entire dataset is then calculated as the mean of the silhouette scores for all the objects in the dataset.

The silhouette score can be a useful tool for evaluating the results of clustering algorithms and for selecting the optimal number of clusters for a dataset. However, it is important to note that the silhouette score is sensitive to the scale of the distances and is not always reliable for datasets with non-convex shapes or with clusters of very different sizes.
A silhouette score of 0.3620743885834876 for Kmeans is intermediate, falling between a score of 0 and a score of 1. This could indicate that the clusters in your dataset are somewhat distinct, but there may be some overlap or fuzzy boundaries between the clusters.

It is important to note that the silhouette score is sensitive to the scale of the distances and is not always reliable for datasets with non-convex shapes or with clusters of very different sizes. Therefore, it is important to consider the context and characteristics of your dataset when interpreting the silhouette score.

A silhouette score of 0.051893570770423565 for dbscan is relatively low, indicating that the clusters in your dataset may not be well-separated or distinct. This could mean that there is significant overlap between the clusters or that the objects within a cluster are not similar to each other.

The Davies-Bouldin index, also known as the Davies-Bouldin score or DBI, is a measure of the compactness and separation of the clusters in a dataset. It ranges from 0 to infinity, with a low score indicating that the clusters are well-separated and compact, and a high score indicating that the clusters are overlapping or scattered.

To calculate the Davies-Bouldin index, you need to have a dataset that has been clustered into groups. Then, for each cluster, you can calculate the average distance between the objects in the cluster and the centroid of the cluster, which is the mean position of the objects in the cluster. This is known as the scatter of the cluster.

Then, for each pair of clusters, you can calculate the distance between the centroids of the two clusters, and divide it by the maximum scatter of the two clusters. The Davies-Bouldin index is then calculated as the mean of these values for all pairs of clusters.

The Davies-Bouldin index can be a useful tool for evaluating the results of clustering algorithms and for selecting the optimal number of clusters for a dataset. However, it is sensitive to the scale of the distances and can be affected by the presence of outliers or skewed distributions. It is
It is not uncommon for different evaluation metrics to yield different results when assessing the quality of clusters in a dataset. This can be due to the different assumptions and characteristics of the metrics, as well as the context and characteristics of the dataset itself.

In this case, it appears that the silhouette score favored the k-means algorithm, while the Davies-Bouldin index favored affinity propagation. This could be due to differences in the way these metrics measure the compactness and separation of the clusters.

The silhouette score measures the similarity of an object to its own cluster compared to other clusters, and assigns a score ranging from -1 to 1 based on this comparison. A high silhouette score indicates that the object is well-matched to its own cluster and poorly-matched to neighboring clusters.

On the other hand, the Davies-Bouldin index measures the compactness and separation of the clusters by calculating the average distance between the objects in a cluster and the centroid of the cluster, and the distance between the centroids of pairs of clusters. It assigns a score based on the mean of these values, with a low score indicating that the clusters are well-separated and compact, and a high score indicating that the clusters are overlapping or scattered.

Given these differences in the way the silhouette score and the Davies-Bouldin index measure cluster quality, it is not surprising that they favored different algorithms in this case. It is important to consider the context and characteristics of your dataset when choosing an evaluation metric, and to use multiple metrics if possible to get a more comprehensive view of the cluster quality.