Dataset Correction

```
import pandas as pd
import numpy as np
import math
import csv

columns = ['Flight No.', 'Timestamp', 'Altitude', 'Latitude', 'Longitude']
# Step 1: Read and preprocess the dataset
dataset = pd.read_csv("NA_11_Jun_29_2018_UTC11.CSV", sep = " ", names = columns)

dataset.to_csv("NA_11_Jun_29_2018_UTC11_Output.CSV", index = False)
# Extract the necessary columns: Flight No., Timestamp, Altitude, Latitude, and Longitude
print(dataset.head(10))
```

Creating dataset for transmission link

```
data_trans = {"Mode k":[1, 2, 3, 4, 5, 6, 7],
      "Mode Color":["Red", "Orange", "Yellow", "Green", "Blue", "Pink", "Purple"],
      "Switching Threshold(km)":[500, 400, 300, 190, 90, 35, 5.56],
      "transmission Rate": [31.895, 43.505, 52.857, 63.970, 77.071, 93.854, 119.130]}
df_trans = pd.DataFrame(data_trans)
df_trans.to_csv("Transmission Link.csv", index = False)
print(df_trans)
```

Conversion to 3D Cartesian Coordinate and finding the Distance Between them

```
def calculate_distance(lat1, lon1, alt1, lat2, lon2, alt2):
    earth_radius = 6371  # Radius of the Earth in kilometers
    lat1_rad = math.radians(lat1)
    lon1_rad = math.radians(lon1)
    lat2_rad = math.radians(lat2)
    lon2_rad = math.radians(lon2)

    equatorial_radius = 6378.137  # kilometers
    polar_radius = 6356.752  # kilometers
    eccentricity = math.sqrt(1 - (polar_radius ** 2) / (equatorial_radius ** 2))

    x1 = (equatorial_radius + alt1) * math.cos(lat1_rad) * math.cos(lon1_rad)
    y1 = (equatorial_radius + alt1) * math.cos(lat1_rad) * math.sin(lon1_rad)
    z1 = ((equatorial_radius * (1 - eccentricity ** 2)) + alt1) * math.sin(lat1_rad)

    x2 = (equatorial_radius + alt2) * math.cos(lat2_rad) * math.cos(lon2_rad)
    y2 = (equatorial_radius + alt2) * math.cos(lat2_rad) * math.sin(lon2_rad)
    z2 = ((equatorial_radius * (1 - eccentricity ** 2)) + alt2) * math.sin(lat2_rad)
```

```
    distance = math.sqrt((x2 - x1) ** 2 + (y2 - y1) ** 2 + (z2 - z1) ** 2)
    return distance
```

Conversion to 3D Cartesian Coordinates for single value

```
def Convert_3D(latitude, longitude, altitude):

    earth_radius = 6371  # Radius of the Earth in kilometers
    latitude_rad = math.radians(latitude)
    longitude_rad = math.radians(longitude)

    equatorial_radius = 6378.137  # kilometers
    polar_radius = 6356.752  # kilometers
    eccentricity = math.sqrt(1 - (polar_radius ** 2) / (equatorial_radius ** 2))

    # Heathrow Airport coordinates to x, y, z coordinates
    x = (equatorial_radius + altitude) * math.cos(latitude_rad) * math.cos(longitude_rad)
    y = (equatorial_radius + altitude) * math.cos(latitude_rad) * math.sin(longitude_rad)
    z = ((equatorial_radius * (1 - eccentricity ** 2)) + altitude) * math.sin(latitude_rad)

    return x, y, z
```

To find transmission rate

Function to calculate the data transmission rate based on the distance between airplanes

```
def calculate_transmission_rate(distance):
    if 500 <= distance:
        return 31.895
    elif 400 <= distance < 500:
        return 43.505
    elif 300 <= distance < 400:
        return 52.857
    elif 190 <= distance < 300:
        return 63.970
    elif 90 <= distance < 190:
        return 77.071
    elif 35 <= distance < 90:
        return 93.854
    elif 5.56 <= distance < 35:
        return 119.130
    else:
        return 0
```

Single Objective Optimization

```
# Load and preprocess the dataset
airplanes = []
ground_stations = []

flight_name=input("Enter starting string code of flight: ")
with open('NA_11_Jun_29_2018_UTC11_Output.csv', 'r') as file:
    reader = csv.reader(file)
    next(reader)  # Skip the header row
    for row in reader:
        flight_no = row[0]
        altitude = float(row[2])
        latitude = float(row[3])
        longitude = float(row[4])

        # Convert latitude, longitude, and altitude to 3D Cartesian coordinates
        coordinates = Convert_3D(longitude, latitude, altitude)
        x = coordinates[0]
        y = coordinates[1]
        z = coordinates[2]

        if flight_no.startswith(flight_name):
            airplanes.append((flight_no, x, y, z))
        else:
            ground_stations.append((flight_no, x, y, z))

        # if flight_no.startswith('AA'):
        #     airplanes.append((flight_no, x, y, z))
        # else:
        #     ground_stations.append((flight_no, x, y, z))

        # if flight_no.startswith('BA'):
        #     airplanes.append((flight_no, x, y, z))
        # else:
        #     ground_stations.append((flight_no, x, y, z))

        # if flight_no.startswith('DA'):
        #     airplanes.append((flight_no, x, y, z))
        # else:
        #     ground_stations.append((flight_no, x, y, z))

        # if flight_no.startswith('LH'):
        #     airplanes.append((flight_no, x, y, z))
        # else:
        #     ground_stations.append((flight_no, x, y, z))
```

```
        # if flight_no.startswith('UA'):
        #     airplanes.append((flight_no, x, y, z))
        # else:
        #     ground_stations.append((flight_no, x, y, z))
def find_max_data_rate_routing_paths(airplanes, ground_stations):
    routing_paths = []

    for airplane in airplanes:
        airplane_id, x_airplane, y_airplane, z_airplane = airplane
        max_data_rate = 0.0
        max_data_rate_path = []

        for i in range(len(ground_stations)):
            for j in range(i + 1, len(ground_stations)):
                ground_station_id, x_gs, y_gs, z_gs = ground_stations[i]
                next_ground_station_id, next_x_gs, next_y_gs, next_z_gs = ground_stations[j]

                # Calculate the distance between the current ground station and the next ground station
                distance = calculate_distance(x_gs, y_gs, z_gs, next_x_gs, next_y_gs, next_z_gs)
                # print(distance)
                # Calculate the data transmission rate for the link
                transmission_rate = calculate_transmission_rate(distance)

                if transmission_rate > max_data_rate:
                    max_data_rate = transmission_rate
                    max_data_rate_path = [(ground_station_id, transmission_rate), (next_ground_station_id,
transmission_rate)]
                elif transmission_rate == max_data_rate:
                    max_data_rate_path.append((ground_station_id, transmission_rate))
                    max_data_rate_path.append((next_ground_station_id, transmission_rate))
                else:
                    max_data_rate_path = [(ground_station_id, transmission_rate), (next_ground_station_id,
transmission_rate)]
                    max_data_rate = transmission_rate

        routing_paths.append({'Airplane': airplane_id, 'Routing Path': max_data_rate_path, 'End-to-End
Data Rate': max_data_rate})

    return routing_paths


# Convert the ground_stations list to a set to remove duplicates
ground_stations = list(set(ground_stations))

# Call the function to find the routing paths with maximum data transmission rate
routing_paths = find_max_data_rate_routing_paths(airplanes, ground_stations)
```

```
# Print and store the routing paths in a text file
with open('routing_paths_relay.txt', 'w') as file:
    for path in routing_paths:
        file.write(str(path) + '\n\n')
        # Print the routing path and its respective data transmission rates
        # file.write(f"An example journey is given below (Here is just an example, not a real optimized routing path):\n")
        file.write(f"{path['Airplane']}: is the source airplane\n")
        for i in range(len(path['Routing Path'])):
            node, rate = path['Routing Path'][i]
            if i == 0:
                file.write(f"({node}, {rate}): The next relay node is {node}, the data transmission rate between {path['Airplane']} and {node} is {rate} Mbps.\n")
            else:
                prev_node, _ = path['Routing Path'][i-1]
                file.write(f"({node}, {rate}): The next relay node is {node}, the data transmission rate between {prev_node} and {node} is {rate} Mbps.\n")
        file.write(f"End-to-end data rate: '{path['End-to-End Data Rate']}': the final end-to-end data rate is {path['End-to-End Data Rate']} Mbps.\n\n")

# Print the routing paths and their respective data transmission rates
for path in routing_paths:
    print(f"Airplane: {path['Airplane']}")
    for i in range(len(path['Routing Path'])):
        node, rate = path['Routing Path'][i]
        if i == 0:
            print(f"({node}, {rate}): The next relay node is {node}, the data transmission rate between {path['Airplane']} and {node} is {rate} Mbps.")
        else:
            prev_node, _ = path['Routing Path'][i-1]
            print(f"({node}, {rate}): The next relay node is {node}, the data transmission rate between {prev_node} and {node} is {rate} Mbps.")
    print(f"End-to-end data rate: '{path['End-to-End Data Rate']}': the final end-to-end data rate is {path['End-to-End Data Rate']} Mbps.")
    print()
```

```
def find_max_data_rate_routing_paths(airplanes, ground_stations):
    routing_paths = []

    for airplane in airplanes:
        airplane_id, x_airplane, y_airplane, z_airplane = airplane
        max_data_rate = 0.0
        max_data_rate_path = []
        visited_ground_stations = set()

        for i in range(len(ground_stations)):
            for j in range(i + 1, len(ground_stations)):
                ground_station_id, x_gs, y_gs, z_gs = ground_stations[i]
                next_ground_station_id, next_x_gs, next_y_gs, next_z_gs = ground_stations[j]

                # Calculate the distance between the current ground station and the next ground station
                distance = calculate_distance(x_gs, y_gs, z_gs, next_x_gs, next_y_gs, next_z_gs)

                # Calculate the data transmission rate for the link
                transmission_rate = calculate_transmission_rate(distance)

                if transmission_rate > max_data_rate:
                    max_data_rate = transmission_rate
                    max_data_rate_path = [(ground_station_id, transmission_rate), (next_ground_station_id,
transmission_rate)]
                elif transmission_rate == max_data_rate:
                    max_data_rate_path.append((ground_station_id, transmission_rate))
                    max_data_rate_path.append((next_ground_station_id, transmission_rate))
                else:
                    max_data_rate_path = [(ground_station_id, transmission_rate), (next_ground_station_id,
transmission_rate)]
                    max_data_rate = transmission_rate

                visited_ground_stations.add(ground_station_id)
                visited_ground_stations.add(next_ground_station_id)

        # Add remaining ground stations that were not part of the optimal path
        remaining_ground_stations = [(ground_station_id, transmission_rate) for ground_station_id, x_gs,
y_gs, z_gs in ground_stations if ground_station_id not in visited_ground_stations]
        max_data_rate_path.extend(remaining_ground_stations)

        routing_paths.append({'Airplane': airplane_id, 'Routing Path': max_data_rate_path, 'End-to-End
Data Rate': max_data_rate})

    return routing_paths


# Convert the ground_stations list to a set to remove duplicates
```

```
ground_stations = list(set(ground_stations))

# Call the function to find the routing paths with maximum data transmission rate
routing_paths = find_max_data_rate_routing_paths(airplanes, ground_stations)

# Print and store the routing paths in a text file
with open('routing_paths.txt', 'w') as file:
    for path in routing_paths:
        file.write(str(path) + '\n\n')

# Print the routing paths and their respective data transmission rates
for path in routing_paths:
    print(path)
```

Multiple objective optimisation

```
def find_optimal_routing_paths(airplanes, ground_stations):
    routing_paths = []

    for airplane in airplanes:
        airplane_id, x_airplane, y_airplane, z_airplane = airplane
        optimal_path = []
        max_data_rate = 0.0
        min_latency = float('inf')

        for ground_station in ground_stations:
            ground_station_id, x_gs, y_gs, z_gs = ground_station

            # Calculate the distance between the airplane and ground station
            distance = calculate_distance(x_airplane, y_airplane, z_airplane, x_gs, y_gs, z_gs)

            # Calculate the data transmission rate for the link
            transmission_rate = calculate_transmission_rate(distance)

            if transmission_rate == 0:
                latency = float('inf')
            else:
                # Calculate the latency for the link
                latency = distance / transmission_rate

            if transmission_rate > max_data_rate:
                max_data_rate = transmission_rate
                min_latency = latency
                optimal_path = [(ground_station_id, max_data_rate, min_latency)]
            elif transmission_rate == max_data_rate and latency < min_latency:
                min_latency = latency
                optimal_path = [(ground_station_id, max_data_rate, min_latency)]
```

```
        routing_paths.append({'Airplane': airplane_id, 'Optimal Path': optimal_path, 'End-to-End Data Rate': max_data_rate, 'End-to-End Latency': min_latency})

    return routing_paths

# Call the function to find the optimal routing paths with maximum data transmission rate and minimum latency
optimal_routing_paths = find_optimal_routing_paths(airplanes, ground_stations)

# Print and store the optimal routing paths in a text file
with open('optimal_routing_paths_length.txt', 'w') as file:
    for path in optimal_routing_paths:
        file.write(str(path) + '\n')
        # print(path)

for path in optimal_routing_paths:
    print(path)
```