Part 1

Algorithm

Step 1 Start

Step 2 Initialize an empty list of flight_routes to store tuples that consist of flight numbers and routes info

```
    flight_routes = []
```

An empty dictionary route_price to store flight price info

```
    route_price = {}
```

An empty dictionary route_seats to store the total seats in each flight

```
    route_seats = {}
```

An empty dictionary route_ratings to store the user rating of each flight

```
    route_ratings = {}
```

Step 3 ask user to input flight number and route using while loop

while True: take input from user

Step 4 use if statement for each user input of flight number and route

```
  flight_number = input('Enter flight number:')
  route = input('Enter flight route')
  if flight_number == 'quit' or route == 'quit':
     if len(flight_routes) < 10:
           print('Please enter at least 10 items.')
      else:
          break
```

Step 5 add a tuple of flight numbers and routes to the flight_routes list

```
  flight_routes.append((flight_number,route))
```

Step 6 using a for loop over flight_routes. For each flight

Step 7 ask user input for flight price and then store it in route_price dictionary under the flight number as the key

```
  price = input('Enter flight price:')
  route_price[flight_number] = price
```

Step 8 ask user input for the total seats and store it in route_seats dictionary under the flight number as the key

```
  seats = input('Enter flight seats:')
  route_seats[flight_number] = seats
```

Step 9 ask user input for the flight rating and store it in route_ratings under the flight number as the key

```
  rating = input('Enter flight ratings:')
  route_ratings[flight_number] = ratings
```

Step 10 initialize an empty set of unique_destinations

```
  unique_destinationations = set()
```

Step 11 split the element under the route over 'to' and then add it to unique_destinations set

```
  unique_destinations.add(route.spli(' to ')[-1])
```

Step 12 dispaly the route, price, total seats, and rating of each flight

```
  print(f'Route: {route}, Price: {price}, Seats:{seats}, Rating:
{ratings}')
```

Step 13 calculate most famous flight route that have an user rating of 3 or higher

```
  popular_routes = {flight: ratings[flight[0]] for flight in routes if
ratings[flight[0]] >= 3}
```

Step 14 calculate most expensive flight route that have price more than $500

```
  expensive_routes = {flight: price[flight[0]] for flight in routes if
price[flight[0]] > 500}
```

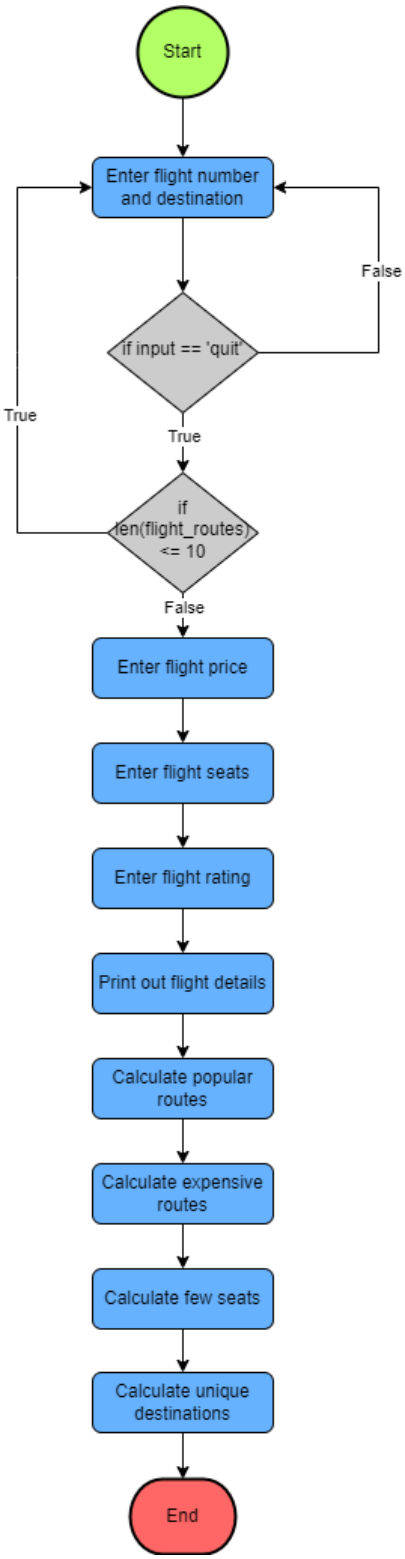
Step 15 calculate a flight route that have less seats under 10

```
  few_seats = {flight: seats[flight[0]] for flight in routes if
seats[flight[0]] < 10}
```

Step 16 display popular, expensive, few seats route, and unique destination of flight route

```
print(f'Popular Routes:{popular_routes}')
print(f'Expensive Routes:{expensive_routes}')
print(f'Few Seats Routes:{few_seats}')
print(f'Route Unique Destinations:{unique_destinations}')
```

Flow Chart



```
def enter_flight_routes():
    flight_routes = []

    while True:
        flight_number = input("Enter flight number (type 'quit' to
exit): ")
        route = input("Enter route (type 'quit' to exit): ")

        if str(flight_number).lower() == 'quit' or str(route).lower()
== 'quit':
            if len(flight_routes) < 10:
                 print(f"Current number of items:
{len(flight_routes)}. Please enter more at least 10 items.")
            else:
                break
        flight_routes.append((flight_number, route))

    # intialize route price
    route_price = {}
    # intialize route seats
    route_seats = {}
    # intialize route ratings
    route_ratings = {}
    # intialize destinations
    destinations = []
    for flight in flight_routes:
        # create route price
        price = float(input(f"Enter price for {flight[0]}: "))
        route_price[flight[0]] = price
        # create route seats
        seats = int(input(f"Enter number of seats for {flight[0]}: "))
        route_seats[flight[0]] = seats
        # create route ratings
        rating = float(input(f"Enter rating for {flight[0]} (1-5): "))
        route_ratings[flight[0]] = rating
        # separate flight number and route
        flight_number, route = flight
        # create unique destinations
        destinations.append(route.split(' to ')[-1])
        # print flights detail
        price = route_price[flight_number]
        seats = route_seats[flight_number]
        rating = route_ratings[flight_number]
        print(f"Route: {route}, Price: ${price}, Seats available:
{seats}, Rating: {rating}")

    return (flight_routes, route_ratings, route_price, route_seats,
destinations)

def filter_routes(flight_routes, route_ratings, route_price,
```

```
route_seats, destinations):
    popular_routes = {flight: route_ratings[flight[0]] for flight in
flight_routes if route_ratings[flight[0]] >= 3}
    expensive_routes = {flight: route_price[flight[0]] for flight in
flight_routes if route_price[flight[0]] > 500}
    few_seats = {flight: route_seats[flight[0]] for flight in
flight_routes if route_seats[flight[0]] < 10}
    unique_destinations_set = set(destinations)
    print("\nPopular Routes:", popular_routes)
    print("\nExpensive Routes:", expensive_routes)
    print("\nRoutes with Few Seats:", few_seats)
    print("\nUnique Destinations:", unique_destinations_set)

# Main program
flight_details = enter_flight_routes()
filter_routes(flight_routes=flight_details[0],
route_ratings=flight_details[1], route_price=flight_details[2],
route_seats=flight_details[3], destinations=flight_details[4])
```

```
Enter flight number (type 'quit' to exit): f1
Enter route (type 'quit' to exit): dubai to london
Enter flight number (type 'quit' to exit): f2
Enter route (type 'quit' to exit): london to dubai
Enter flight number (type 'quit' to exit): f3
Enter route (type 'quit' to exit): dubai to paris
Enter flight number (type 'quit' to exit): f4
Enter route (type 'quit' to exit): paris to dubai
Enter flight number (type 'quit' to exit): f5
Enter route (type 'quit' to exit): dubai to mumbai
Enter flight number (type 'quit' to exit): f6
Enter route (type 'quit' to exit): dubai to chennai
Enter flight number (type 'quit' to exit): f7
Enter route (type 'quit' to exit): dubai to dahka
Enter flight number (type 'quit' to exit): f8
Enter route (type 'quit' to exit): dubai to new york
Enter flight number (type 'quit' to exit): f9
Enter route (type 'quit' to exit): dubai to karachi
Enter flight number (type 'quit' to exit): f10
Enter route (type 'quit' to exit): dubai to kabul
Enter flight number (type 'quit' to exit): quit
Enter route (type 'quit' to exit): quit
Enter price for f1: 100
Enter number of seats for f1: 3
Enter rating for f1 (1-5): 2
Route: dubai to london, Price: $100.0, Seats available: 3, Rating: 2.0
Enter price for f2: 150
Enter number of seats for f2: 36
Enter rating for f2 (1-5): 3
Route: london to dubai, Price: $150.0, Seats available: 36, Rating:
3.0
```

```
Enter price for f3: 600
Enter number of seats for f3: 51
Enter rating for f3 (1-5): 2
Route: dubai to paris, Price: $600.0, Seats available: 51, Rating: 2.0
Enter price for f4: 300
Enter number of seats for f4: 23
Enter rating for f4 (1-5): 2
Route: paris to dubai, Price: $300.0, Seats available: 23, Rating: 2.0
Enter price for f5: 251
Enter number of seats for f5: 12
Enter rating for f5 (1-5): 3
Route: dubai to mumbai, Price: $251.0, Seats available: 12, Rating:
3.0
Enter price for f6: 500
Enter number of seats for f6: 25
Enter rating for f6 (1-5): 3
Route: dubai to chennai, Price: $500.0, Seats available: 25, Rating:
3.0
Enter price for f7: 400
Enter number of seats for f7: 43
Enter rating for f7 (1-5): 2
Route: dubai to dahka, Price: $400.0, Seats available: 43, Rating: 2.0
Enter price for f8: 700
Enter number of seats for f8: 13
Enter rating for f8 (1-5): 2
Route: dubai to new york, Price: $700.0, Seats available: 13, Rating:
2.0
Enter price for f9: 230
Enter number of seats for f9: 14
Enter rating for f9 (1-5): 2
Route: dubai to karachi, Price: $230.0, Seats available: 14, Rating:
2.0
Enter price for f10: 100
Enter number of seats for f10: 31
Enter rating for f10 (1-5): 2
Route: dubai to kabul, Price: $100.0, Seats available: 31, Rating: 2.0

Popular Routes: {('f2', 'london to dubai'): 3.0, ('f5', 'dubai to
mumbai'): 3.0, ('f6', 'dubai to chennai'): 3.0}

Expensive Routes: {('f3', 'dubai to paris'): 600.0, ('f8', 'dubai to
new york'): 700.0}

Routes with Few Seats: {('f1', 'dubai to london'): 3}

Unique Destinations: {'dahka', 'paris', 'dubai', 'mumbai', 'london',
'new york', 'kabul', 'chennai', 'karachi'}
```

Part 2

```
# Import necessary libraries
import pandas as pd
import matplotlib.pyplot as plt
```

Question 1

```
# Import the dataset into a pandas DataFrame
df = pd.read_csv('sustainable_development_report_2023.csv')

# Display a brief description of the dataset
print(df.info())
print(df.describe())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 166 entries, 0 to 165
Data columns (total 21 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   country_code   166 non-null    object 
 1   country        166 non-null    object 
 2   region         166 non-null    object 
 3   overall_score  166 non-null    float64
 4   goal_1_score   151 non-null    float64
 5   goal_2_score   166 non-null    float64
 6   goal_3_score   166 non-null    float64
 7   goal_4_score   166 non-null    float64
 8   goal_5_score   166 non-null    float64
 9   goal_6_score   166 non-null    float64
 10  goal_7_score   166 non-null    float64
 11  goal_8_score   166 non-null    float64
 12  goal_9_score   166 non-null    float64
 13  goal_10_score  149 non-null    float64
 14  goal_11_score  166 non-null    float64
 15  goal_12_score  166 non-null    float64
 16  goal_13_score  166 non-null    float64
 17  goal_14_score  126 non-null    float64
 18  goal_15_score  166 non-null    float64
 19  goal_16_score  166 non-null    float64
 20  goal_17_score  166 non-null    float64
dtypes: float64(18), object(3)
memory usage: 27.4+ KB
None
       overall_score  goal_1_score  goal_2_score  goal_3_score
goal_4_score  \
count     166.000000    151.000000    166.000000    166.000000
166.000000
mean       67.549197     75.234401     59.799100     69.694078
76.512968
std        10.295499     31.169948     10.620853     20.354575
```

```
23.181919
min        38.676086      0.000000     19.805800     12.952714
1.232250
25%        60.547488     55.779250     54.007188     51.860089
61.417938
50%        69.376528     93.300500     61.027500     75.437629
84.772875
75%        74.947511     98.950750     67.264335     85.524428
95.644063
max        86.760595    100.000000     83.401125     97.115143
99.761667

       goal_5_score  goal_6_score  goal_7_score  goal_8_score
goal_9_score  \
count    166.000000    166.000000    166.000000    166.000000
166.000000
mean      63.285420     66.710744     61.413598     71.952935
51.600648
std       16.399691     14.091641     20.364351     10.592308
26.561680
min       13.054750     32.600000      8.697000     39.535000
1.654833
25%       51.046250     55.237250     47.521312     66.426857
30.206464
50%       65.869875     67.878000     68.612750     73.157643
48.168798
75%       76.137000     76.044200     74.364000     79.626036
74.713036
max       94.021667     95.057600     99.550750     93.382750
99.128857

       goal_10_score  goal_11_score  goal_12_score  goal_13_score  \
count     149.000000     166.000000     166.000000     166.000000
mean       62.917889      72.181106      79.775904      82.119387
std        27.348955      18.215526      16.092924      21.175602
min         0.000000      13.826250      37.729429       0.000000
25%        41.608000      59.969813      68.592464      72.543000
50%        69.700500      76.851500      84.566024      90.903000
75%        84.612500      86.499437      94.084750      96.710875
max       100.000000      99.858000      98.811200      99.925333

       goal_14_score  goal_15_score  goal_16_score  goal_17_score
count     126.000000     166.000000     166.000000     166.000000
mean       65.494968      66.637486      61.546404      60.954819
std        11.475977      14.175602      15.517449      12.991856
min        36.579400      26.477500      29.438000      29.350000
25%        57.647625      56.606900      49.029477      50.854917
50%        65.412250      66.295700      60.908687      60.805000
75%        72.992375      76.585000      73.716061      71.647188
max        90.394750      97.849000      93.844909      94.026500
```

The Sustainable Development Report 2023 dataset comprises comprehensive data related to sustainability and progress towards Sustainable Development Goals (SDGs) for numerous countries. Each entry in the dataset presents information on a country's sustainability scores, regional classification, and performance on individual SDGs.

Question 2

```
# Select a random sample of 100 from the original data set
# The random_state parameter ensures reproducibility
sample_df = df.sample(n=100, random_state=1)
print(sample_df.head())
# Rerun the code to observe whether it selects the same sample
# If you don't set the random_state, it will select a different sample
each time
sample_df = df.sample(n=100, random_state=1)
print(sample_df.head())
```

```
    country_code          country               region  overall_score
\
44           KGZ  Kyrgyz Republic  E. Europe & C. Asia      74.405434

47           ISR           Israel                 OECD      73.970339

150          COG      Congo, Rep.   Sub-Saharan Africa      52.631544

66           MNE       Montenegro  E. Europe & C. Asia      71.404384

152          BFA     Burkina Faso   Sub-Saharan Africa      52.445504


     goal_1_score  goal_2_score  goal_3_score  goal_4_score
goal_5_score  \
44         92.081     60.383143     74.912929     92.578250
61.65225
47         98.626     55.391714     96.335167     99.584333
76.09800
150        24.022     53.156000     45.595643     47.320000
54.80600
66         98.945     51.761143     75.668500     88.159000
56.85700
152        26.894     58.639125     46.632286     26.679750
40.67700

     goal_6_score  ...  goal_8_score  goal_9_score  goal_10_score  \
44      66.422800  ...     69.620286     37.527000        92.6690
47      68.208200  ...     84.289333     94.729571        71.5535
150     51.992600  ...     54.185714      8.091667        19.8590
66      65.209333  ...     61.567000     61.596857        66.6515
152     44.838000  ...     70.175429     21.924143        22.1125

     goal_11_score  goal_12_score  goal_13_score  goal_14_score  \
```

```
44        89.22600      91.934571      94.814333            NaN
47        84.62600      66.557429      67.152667        37.5392
150       51.24025      95.283600      92.180667        72.0104
66        74.71575      70.367333      92.787500        52.0974
152       60.48000      95.612286      98.802000            NaN

     goal_15_score  goal_16_score  goal_17_score
44         70.8730      55.205300      71.593000
47         49.3564      69.597250      66.129750
150        83.5404      48.193556      49.124250
66         54.2580      78.451375      85.697333
152        87.9436      45.524600      57.217000

[5 rows x 21 columns]
    country_code          country               region  overall_score
\
44           KGZ  Kyrgyz Republic  E. Europe & C. Asia      74.405434

47           ISR           Israel                 OECD      73.970339

150          COG      Congo, Rep.   Sub-Saharan Africa      52.631544

66           MNE       Montenegro  E. Europe & C. Asia      71.404384

152          BFA     Burkina Faso   Sub-Saharan Africa      52.445504


     goal_1_score  goal_2_score  goal_3_score  goal_4_score
goal_5_score  \
44         92.081     60.383143     74.912929     92.578250
61.65225
47         98.626     55.391714     96.335167     99.584333
76.09800
150        24.022     53.156000     45.595643     47.320000
54.80600
66         98.945     51.761143     75.668500     88.159000
56.85700
152        26.894     58.639125     46.632286     26.679750
40.67700

     goal_6_score  ...  goal_8_score  goal_9_score  goal_10_score  \
44      66.422800  ...     69.620286     37.527000        92.6690
47      68.208200  ...     84.289333     94.729571        71.5535
150     51.992600  ...     54.185714      8.091667        19.8590
66      65.209333  ...     61.567000     61.596857        66.6515
152     44.838000  ...     70.175429     21.924143        22.1125

     goal_11_score  goal_12_score  goal_13_score  goal_14_score  \
44         89.22600      91.934571      94.814333            NaN
47         84.62600      66.557429      67.152667        37.5392
```

```
150       51.24025      95.283600      92.180667        72.0104
66        74.71575      70.367333      92.787500        52.0974
152       60.48000      95.612286      98.802000            NaN

     goal_15_score  goal_16_score  goal_17_score
44         70.8730      55.205300      71.593000
47         49.3564      69.597250      66.129750
150        83.5404      48.193556      49.124250
66         54.2580      78.451375      85.697333
152        87.9436      45.524600      57.217000

[5 rows x 21 columns]
```

After rerun of the code every times the same sample data is selected because random_state parameter is set to 1 constant value.

Question 3

```
# Display the first and last few rows of the dataset
# Also, determine the number of columns and rows in the dataset
# First few rows
print(df.head())
# Last few rows
print(df.tail())
# Number of rows and columns
print(df.shape)
```

```
  country_code  country region  overall_score  goal_1_score
goal_2_score  \
0          FIN  Finland   OECD      86.760595       99.5750
60.886750
1          SWE   Sweden   OECD      85.981397       98.8885
63.074125
2          DNK  Denmark   OECD      85.683637       99.2155
71.025250
3          DEU  Germany   OECD      83.358447       99.5105
72.366000
4          AUT  Austria   OECD      82.280189       99.4510
73.067500

   goal_3_score  goal_4_score  goal_5_score  goal_6_score  ...
goal_8_score  \
0     95.386385     97.169333      92.11125       94.3276  ...
86.789000
1     96.904000     99.761667      91.44025       95.0576  ...
84.966429
2     95.398500     99.339667      86.99800       90.7316  ...
87.562429
3     93.039357     97.162667      81.92025       88.4434  ...
86.967286
```

```
4     92.468000     97.914333      84.57925       92.1636  ...
83.274143

   goal_9_score  goal_10_score  goal_11_score  goal_12_score
goal_13_score  \
0     95.995714        98.4685      91.233750      60.059571
68.793667
1     97.586286        94.9650      90.389250      56.830571
70.031000
2     96.984857        98.1560      93.038500      44.571714
60.780667
3     95.788429        88.1470      90.096500      55.412857
64.002000
4     96.982143        94.6345      92.473667      49.623286
57.332000

   goal_14_score  goal_15_score  goal_16_score  goal_17_score
0      87.928000        85.0700      92.521091       75.60100
1      69.348667        80.1882      88.508455       85.77025
2      76.303333        92.7924      93.844909       82.14800
3      73.996000        79.2318      89.457545       84.39025
4            NaN        73.5836      87.911455       71.13025

[5 rows x 21 columns]
    country_code                   country              region
overall_score  \
161          SOM                   Somalia  Sub-Saharan Africa
48.027231
162          YEM                Yemen, Rep.                MENA
46.846980
163          TCD                      Chad  Sub-Saharan Africa
45.342321
164          CAF  Central African Republic  Sub-Saharan Africa
40.395839
165          SSD                South Sudan  Sub-Saharan Africa
38.676086

     goal_1_score  goal_2_score  goal_3_score  goal_4_score
goal_5_score  \
161       11.2740     27.306833     17.860923      55.63900
25.86100
162        4.5525     28.769714     44.467429      41.76675
13.05475
163       25.4270     38.534714     27.061071      13.30775
30.83550
164        3.1820     36.468000     12.952714      19.30575
34.20075
165        0.0000     19.805800     23.861714       1.23225
55.98875
```

```
     goal_6_score  ...  goal_8_score  goal_9_score  goal_10_score  \
161       49.3006  ...     55.669500      5.599857        73.8030
162       36.2314  ...     53.237800     14.223714        66.2925
163       42.4036  ...     64.424333      9.631571        63.0405
164       40.4204  ...     53.382333      7.064714         9.5775
165       41.0406  ...     50.917000      1.654833        26.6195

     goal_11_score  goal_12_score  goal_13_score  goal_14_score  \
161      69.417667      94.129000      99.925333      50.251200
162      52.952750      95.959143      98.667000      74.936667
163      32.822250      90.994167      99.079000            NaN
164      21.898667      94.462143      99.463333            NaN
165      13.826250      90.960000      99.408000            NaN

     goal_15_score  goal_16_score  goal_17_score
161        53.3714      40.012200      43.725667
162        48.3705      35.905714      50.094333
163        76.1944      29.438000      52.594000
164        89.7172      42.332667      36.516750
165        74.6870      38.141167      41.622000

[5 rows x 21 columns]
(166, 21)
```

The data set contains 166 rows and 21 columns. The dataset contains the different regions and its associated countries with overall score and multiple goal sets value of the sustainability of development.

Question 4

```
# Select a numerical column and arrange data in ascending and
descending order
numerical_column = 'overall_score'
asc_sorted = df.sort_values(by=numerical_column)
desc_sorted = df.sort_values(by=numerical_column, ascending=False)
print("First value in ascending order:",
asc_sorted[numerical_column].iloc[0])
print("First value in descending order:",
desc_sorted[numerical_column].iloc[0])
```

```
First value in ascending order: 38.67608607
First value in descending order: 86.76059478
```

Question 5

```
# Get the minimum and maximum values in the selected numerical column
# Compare with the results from Question 3
min_value = df[numerical_column].min()
max_value = df[numerical_column].max()
```

```
print("Minimum value:", min_value)
print("Maximum value:", max_value)
```

```
Minimum value: 38.67608607
Maximum value: 86.76059478
```

The minimum value is 38.67608607 and maximum value is 86.76059478 for the overall score of sustainability development report.

Question 6

```
# Calculate the average in two different ways
average_method1 = df[numerical_column].mean()
average_method2 = df[numerical_column].sum() /
df[numerical_column].count()
print("Average using method 1:", average_method1)
print("Average using method 2:", average_method2)
```

```
Average using method 1: 67.5491968813253
Average using method 2: 67.5491968813253
```

The first method use the builtin mean function to calculate the average. And in second method first calulate the total sum of the overall_score column, then count the total number of the records in the overall_score, and then divide the total sum with total count of records in the overall_score column. Both methods gives the same average value 67.5491968813253.

Question 7

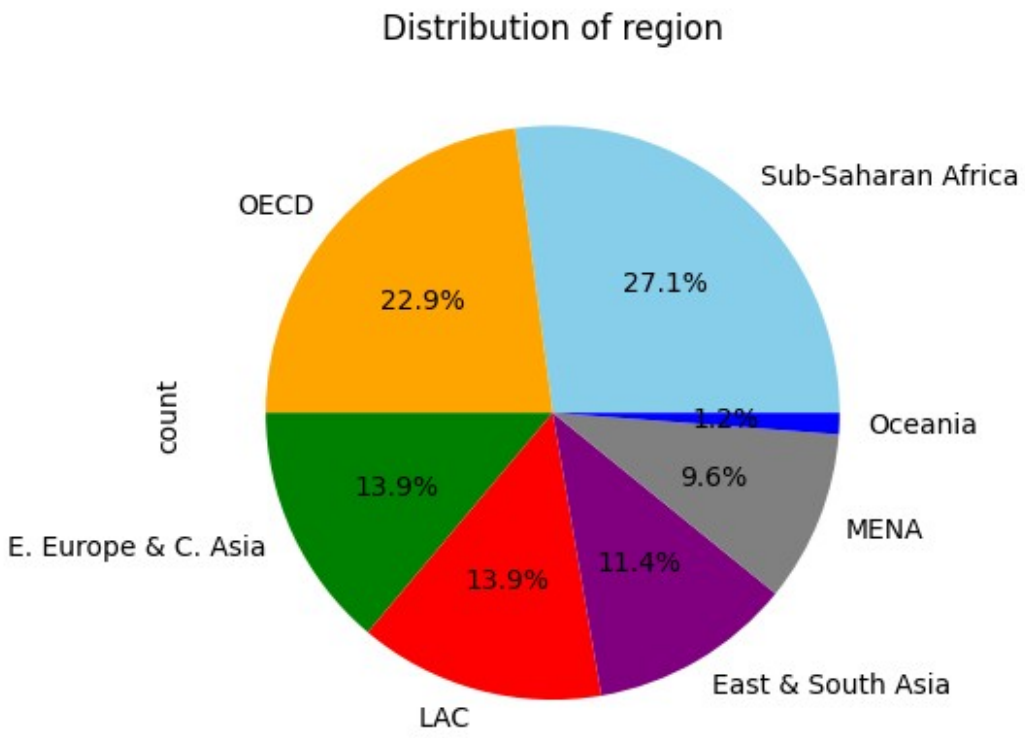
```
# Create two new columns based on a numerical column
df['new_column_1'] = df[numerical_column] + 5  # Adding 5 to the
values
df['new_column_2'] = df[numerical_column] * 2  # Doubling the values
print(df.head())
```

```
  country_code  country region  overall_score  goal_1_score
goal_2_score  \
0          FIN  Finland   OECD      86.760595       99.5750
60.886750
1          SWE   Sweden   OECD      85.981397       98.8885
63.074125
2          DNK  Denmark   OECD      85.683637       99.2155
71.025250
3          DEU  Germany   OECD      83.358447       99.5105
72.366000
4          AUT  Austria   OECD      82.280189       99.4510
73.067500

   goal_3_score  goal_4_score  goal_5_score  goal_6_score  ...
goal_10_score  \
0     95.386385     97.169333      92.11125       94.3276  ...
```

```
98.4685
1     96.904000     99.761667      91.44025       95.0576  ...
94.9650
2     95.398500     99.339667      86.99800       90.7316  ...
98.1560
3     93.039357     97.162667      81.92025       88.4434  ...
88.1470
4     92.468000     97.914333      84.57925       92.1636  ...
94.6345

   goal_11_score  goal_12_score  goal_13_score  goal_14_score
goal_15_score  \
0      91.233750      60.059571      68.793667      87.928000
85.0700
1      90.389250      56.830571      70.031000      69.348667
80.1882
2      93.038500      44.571714      60.780667      76.303333
92.7924
3      90.096500      55.412857      64.002000      73.996000
79.2318
4      92.473667      49.623286      57.332000            NaN
73.5836

   goal_16_score  goal_17_score  new_column_1  new_column_2
0      92.521091       75.60100     91.760595    173.521190
1      88.508455       85.77025     90.981397    171.962794
2      93.844909       82.14800     90.683637    171.367274
3      89.457545       84.39025     88.358447    166.716893
4      87.911455       71.13025     87.280189    164.560379

[5 rows x 23 columns]
```

Question 8

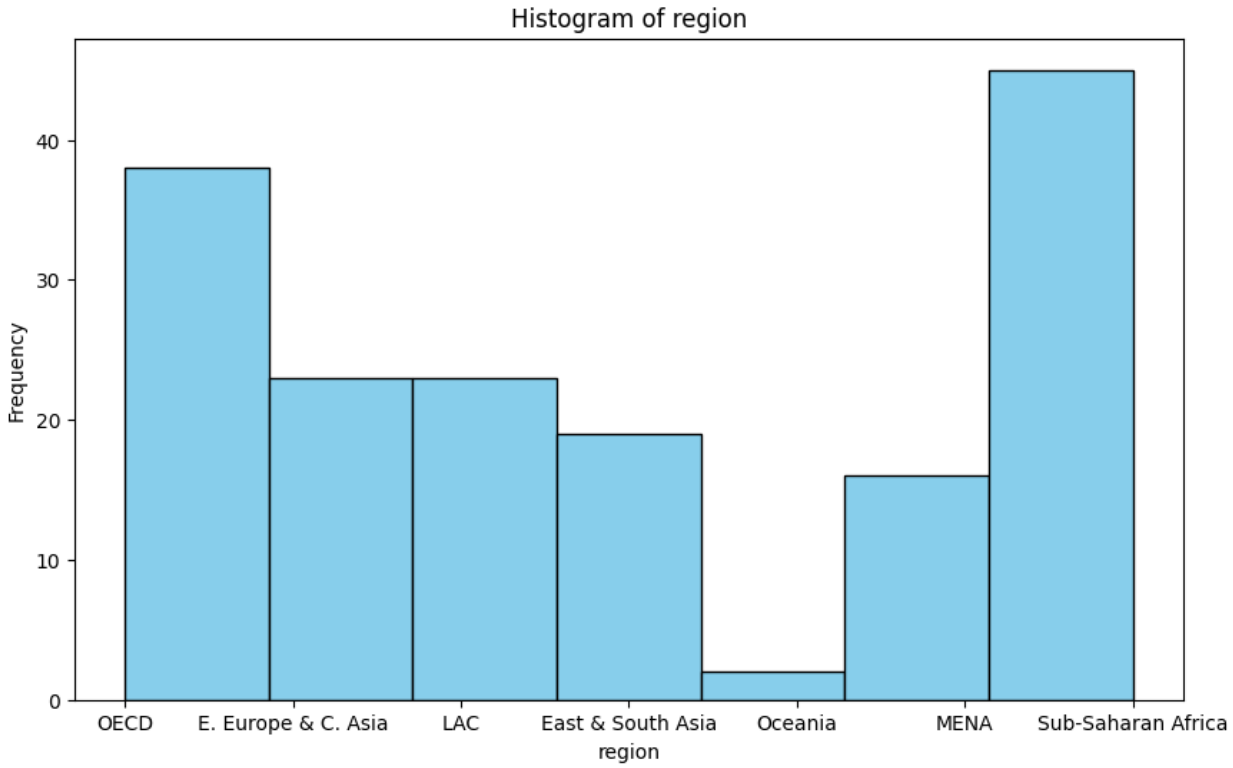
```
# Identify a column with repeated values, calculate the distribution,
and create a pie chart
repeated_column = 'region'
distribution = df[repeated_column].value_counts()
distribution.plot.pie(autopct='%1.1f%%', colors=['skyblue', 'orange',
'green', 'red', 'purple', 'grey', 'blue'], title="Distribution of " +
repeated_column)
plt.show()
```



The chart shows the distribution of the sustainability development of the regions.

Question 9

```
# Create a histogram to visualize the data
plt.figure(figsize=(10, 6))  # Adjust the figure size
plt.hist(df[repeated_column], bins=len(distribution),
color='skyblue', edgecolor='black')
plt.xlabel(repeated_column)
plt.ylabel('Frequency')
plt.title('Histogram of ' + repeated_column)
plt.show()
```



The histogram chart shows the frequency of the sustainability development of the regions.

Question 10

```
# Develop a specific question related to data filtering and analysis
# What is the average overall score for countries in the OECD region?
average_oecd_score = df[df['region'] == 'OECD']
['overall_score'].mean()
print("Average Overall Score for OECD countries:", average_oecd_score)

Average Overall Score for OECD countries: 79.15044726868422
```

The result demonstarte the average progess rate that is 79.15044726868422 for sustainability development of the OECD region countries.