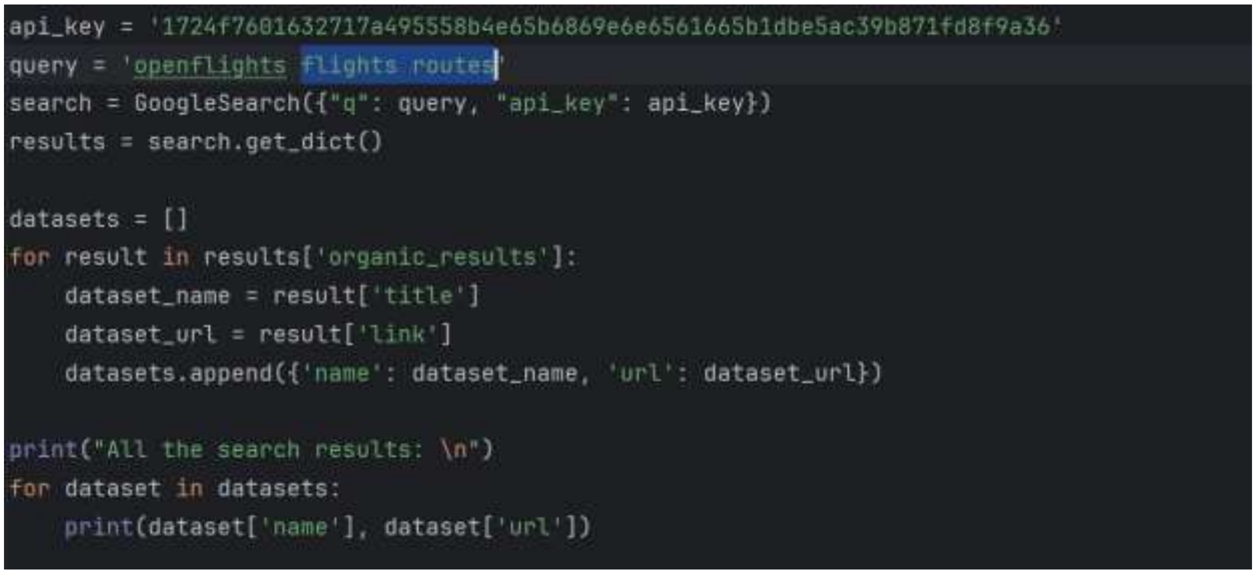
Air flights scraper

Summary of the idea:

Air flights scraper is a project to scrap the datasets of flights around the airports of the world, accessing on its source and destination airport. It establishes a connection between the source and destination airport per single flight to detect the airplanes routes, in order to determine at the end which airport is the most important according to number of landings. The software looks for datasets first, selects one to web scrape, and then arranges the data for preview. Following that, the data is linked together, and a network frame of nodes and edges is drawn. Next, it analyses this network to calculate each node's degree and betweenness centrality in order to identify the most important airport. Next, using the KDE concept to plot a heatmap based on the locations of each airport in the world map. Lastly, it plots the 2D relationship network in 3D to make the nodes and edges seem like a 3D model. The Tkinter library is used to include all of these characteristics into a graphical user interface, which gives the program a preview.

First: Web scraping:

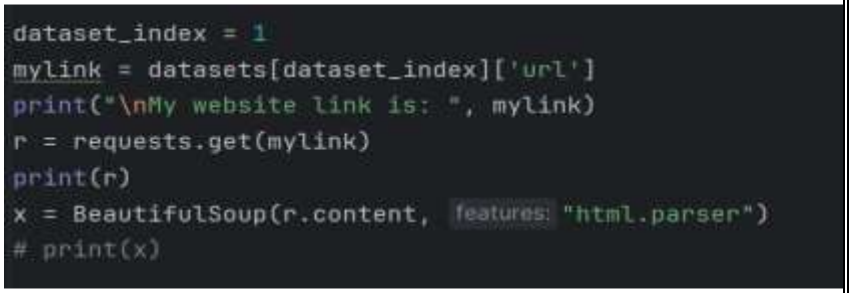
I used the bs4 libraries, requests, and SerpAPI to scrape the data. Before I could find the dataset I needed, I first searched for a website whose HTML inspect code could be used for successful web scraping. Next, I added a Google search function to my website that searches for "flights routes datasets." I looped over the data's organic results section and dictionary-stored the dataset name and URL for use later. After that, I printed this dictionary to print all the websites that the search found as illustrated in figure (1).



```
api_key = '1724f7601632717a495558b4e65b6869e6e6561665b1dbe5ac39b871fd8f9a36'
query = 'openflights flights routes'
search = GoogleSearch({"q": query, "api_key": api_key})
results = search.get_dict()

datasets = []
for result in results['organic_results']:
    dataset_name = result['title']
    dataset_url = result['link']
    datasets.append({'name': dataset_name, 'url': dataset_url})

print("All the search results: \n")
for dataset in datasets:
    print(dataset['name'], dataset['url'])
```

Figure (1): The figure shows the dataset search in the web scraping stage

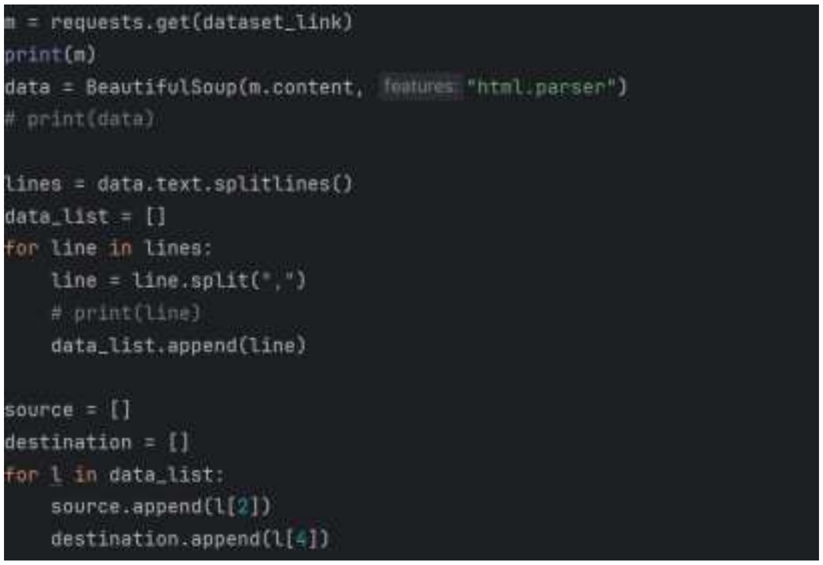
From these links, I chose my own website link by indexing to be my source website of web scraping. I printed the link of this website and began the process of web scraping as shown in figure (2).



```
dataset_index = 1
mylink = datasets[dataset_index]['url']
print("\nMy website link is: ", mylink)
r = requests.get(mylink)
print(r)
x = BeautifulSoup(r.content,  features: "html.parser")
# print(x)
```

Figure (2): The figure shows the selection of the website and parsing its content in the web scraping stage

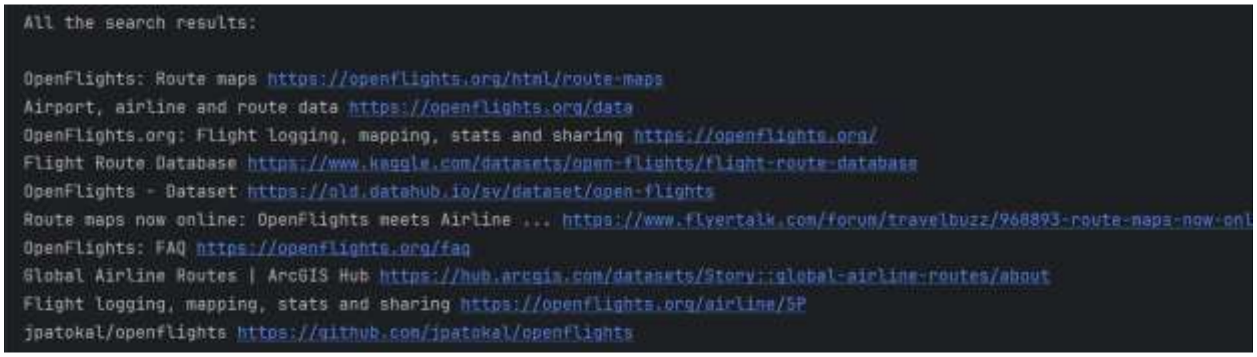
Before beginning, I ensured about the ability to scrap the website by applying the function of (request.get) and it gave me <Response 200> which is available. I applied the BeautifulSoup function to parse the text of the website and after that search with what I want inside it. After checking the HTML code, I found the dat file that I want inside a div named ("a") with an attribute name of "href", so I looped on the dat files and printed them. Then, I chose my own dat file which the dataset I will scrap from, so I printed it. And again I applied the BeautifulSoup function to parse and split the lines of it. Finally, I looped on the lines of my dat file and appended the columns I want in separated lists to make me able use them later as shown in figure (3).



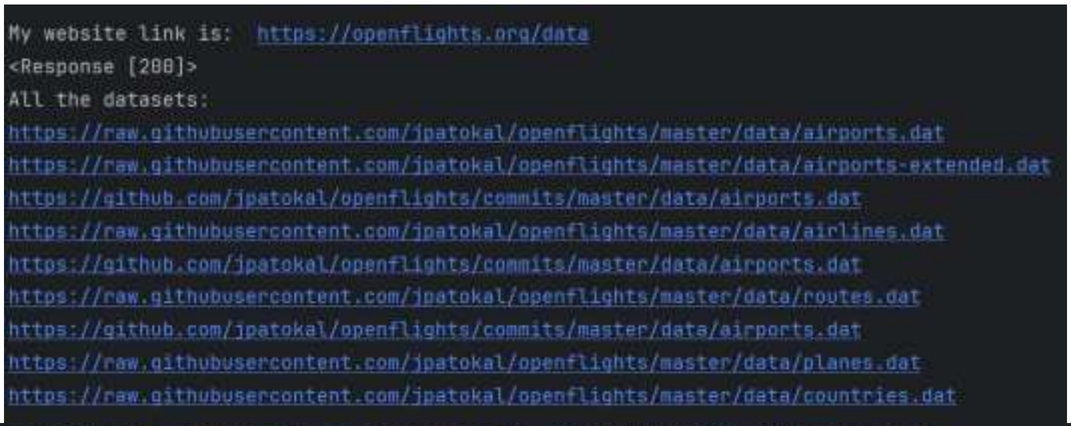
```
m = requests.get(dataset_link)
print(m)
data = BeautifulSoup(m.content,  features: "html.parser")
# print(data)

lines = data.text.splitlines()
data_list = []
for line in lines:
    line = line.split(",")
    # print(line)
    data_list.append(line)

source = []
destination = []
for l in data_list:
    source.append(l[2])
    destination.append(l[4])
```

Figure (3): The figure shows the scraping process, splitting the lines and organizing of data in lists in the web scraping stage

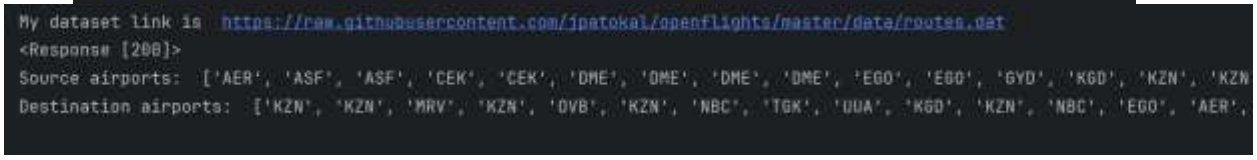
And the output is:



```
All the search results:

OpenFlights: Route maps https://openflights.org/html/route-maps
Airport, airline and route data https://openflights.org/data
OpenFlights.org: Flight logging, mapping, stats and sharing https://openflights.org/
Flight Route Database https://www.kaggle.com/datasets/open-flights/flight-route-database
OpenFlights - Dataset https://old.datahub.io/sv/dataset/open-flights
Route maps now online: OpenFlights meets Airline ... https://www.flyertalk.com/forum/travelbuzz/968893-route-maps-now-onl
OpenFlights: FAQ https://openflights.org/faq
Global Airline Routes | ArcGIS Hub https://hub.arcgis.com/datasets/Story::global-airline-routes/about
Flight logging, mapping, stats and sharing https://openflights.org/airline/5P
jpatokal/openflights https://github.com/jpatokal/openflights
```



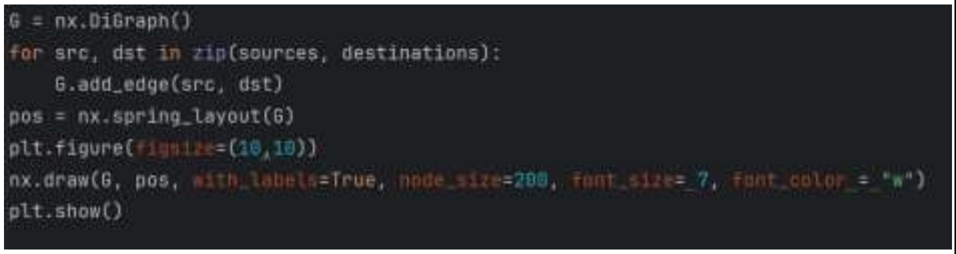
```
My website link is:  https://openflights.org/data
<Response [200]>
All the datasets:
https://raw.githubusercontent.com/jpatokal/openflights/master/data/airports.dat
https://raw.githubusercontent.com/jpatokal/openflights/master/data/airports-extended.dat
https://github.com/jpatokal/openflights/commits/master/data/airports.dat
https://raw.githubusercontent.com/jpatokal/openflights/master/data/airlines.dat
https://github.com/jpatokal/openflights/commits/master/data/airports.dat
https://raw.githubusercontent.com/jpatokal/openflights/master/data/routes.dat
https://github.com/jpatokal/openflights/commits/master/data/airports.dat
https://raw.githubusercontent.com/jpatokal/openflights/master/data/planes.dat
https://raw.githubusercontent.com/jpatokal/openflights/master/data/countries.dat
```



```
My dataset link is  https://raw.githubusercontent.com/jpatokal/openflights/master/data/routes.dat
<Response [200]>
Source airports:  ['AER', 'ASF', 'ASF', 'CEK', 'CEK', 'DME', 'DME', 'DME', 'DME', 'EGO', 'EGO', 'GYD', 'KGD', 'KZN', 'KZN
Destination airports:  ['KZN', 'KZN', 'MRV', 'KZN', 'OVB', 'KZN', 'NBC', 'TGK', 'UUA', 'KGD', 'KZN', 'NBC', 'EGO', 'AER',
```

Second: Network construction:

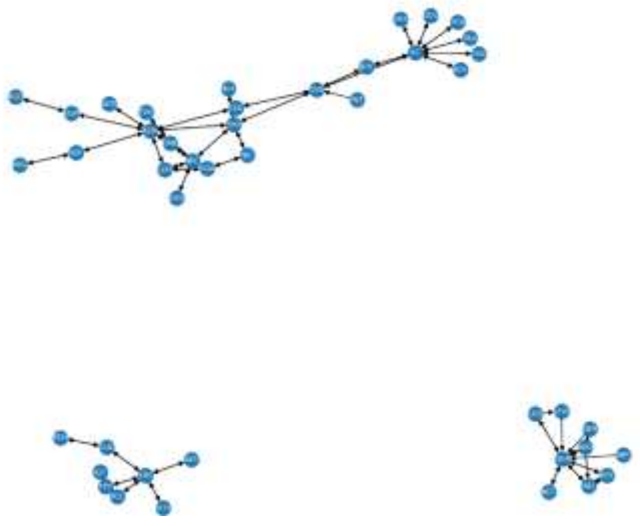
For the network implementation, I made a directed graph to show the arrows and directions between the source and the destination airports. So, I looped in the lists of the sources and destination and added edge between them. After that, I used a spring layout for previewing the network with a good look. And finally, I drew it after initializing the figure size and some parameters like the node size, font size, and font color as shown in figure (4).



```
G = nx.DiGraph()
for src, dst in zip(sources, destinations):
    G.add_edge(src, dst)
pos = nx.spring_layout(G)
plt.figure(figsize=(10,10))
nx.draw(G, pos, with_labels=True, node_size=200, font_size=_7, font_color_=_"w")
plt.show()
```

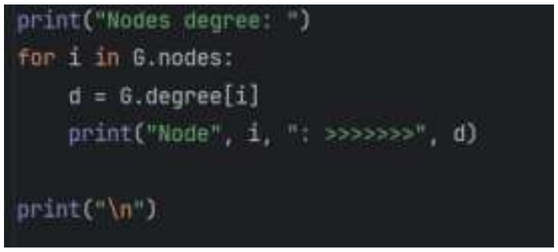
Figure (4): The figure shows the adding of edges in the directed graph in the network implementation stage

And the output is:



Third: Network analysis:

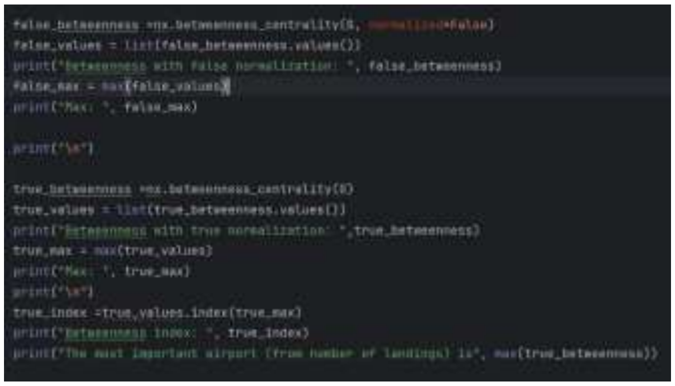
During the network analysis phase, I began by looping over each node to get its degree. Following that, I used the degree function to calculate each node's degree centrality, which can be seen in figure (5).



```
print("Nodes degree: ")
for i in G.nodes:
    d = G.degree[i]
    print("Node", i, ": >>>>>>>", d)

print("\n")
```

Figure (5): The figure shows the degree centrality in the network analysis stage

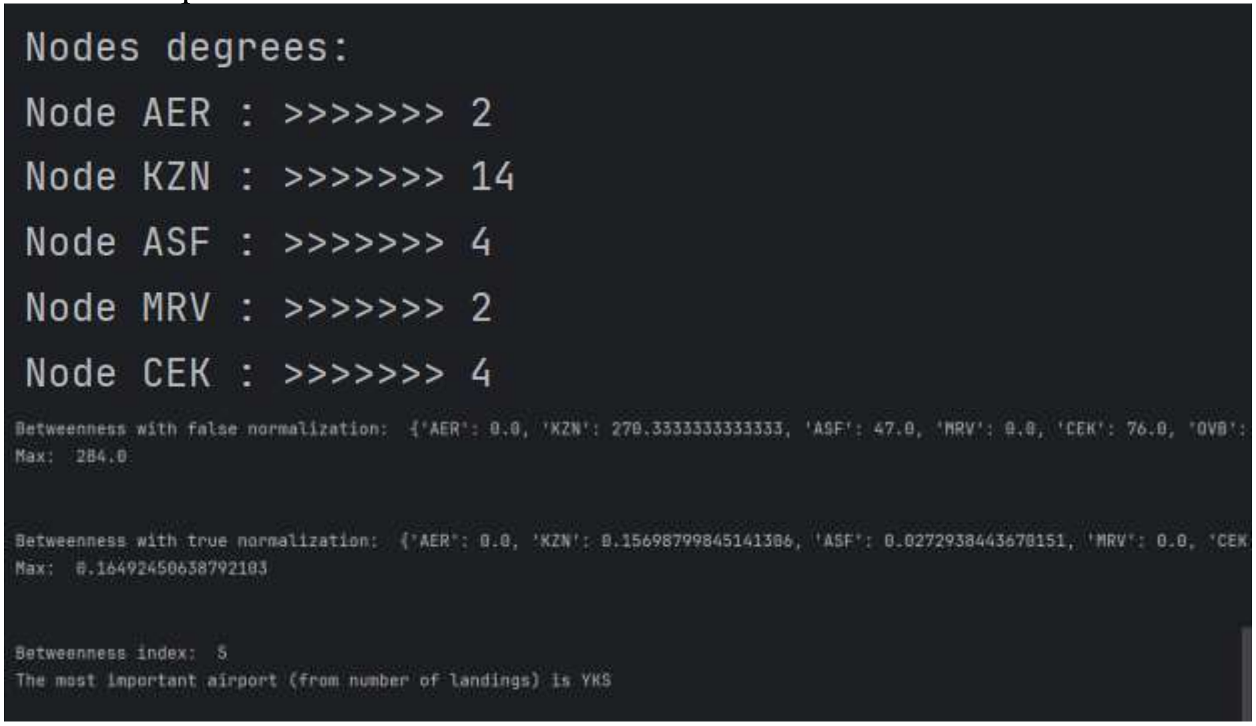
Second, using two different sets of parameters, the false normalization and the true normalization, I calculated each node's betweenness. The values appear as ratios less than 1 after true normalization. I then computed and printed the maximum of these betweenness values. As seen in figure (6), I displayed the name and index of the node with the highest betweenness value to display the most important node between them which represents the most important airport in the dataset.



```
false_betweenness =nx.betweenness_centrality(G, normalized=False)
false_values = list(false_betweenness.values())
print("Betweenness with false normalization: ", false_betweenness)
false_max = max(false_values)
print("Max: ", false_max)

print("\n")

true_betweenness =nx.betweenness_centrality(G)
true_values = list(true_betweenness.values())
print("Betweenness with true normalization: ",true_betweenness)
true_max = max(true_values)
print("Max: ", true_max)
print("\n")
true_index =true_values.index(true_max)
print("Betweenness index: ", true_index)
print("The most important airport (from number of landings) is", max(true_betweenness))
```

Figure (6): The figure shows the betweenness centrality in the network analysis stage

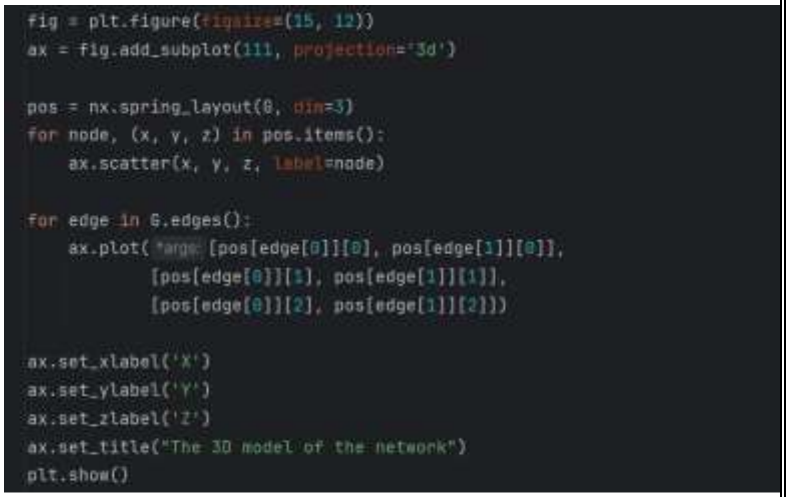
And the output is:



```
Nodes degrees:
Node AER : >>>>>>> 2
Node KZN : >>>>>>> 14
Node ASF : >>>>>>> 4
Node MRV : >>>>>>> 2
Node CEK : >>>>>>> 4
Betweenness with false normalization:  {'AER': 0.0, 'KZN': 270.3333333333333, 'ASF': 47.0, 'MRV': 0.0, 'CEK': 76.0, 'OVB':
Max:  284.0


Betweenness with true normalization:  {'AER': 0.0, 'KZN': 0.15698799845141306, 'ASF': 0.0272938443670151, 'MRV': 0.0, 'CEK
Max:  0.16492450638792103


Betweenness index:  5
The most important airport (from number of landings) is YKS
```

Fourth: 3D modelling:

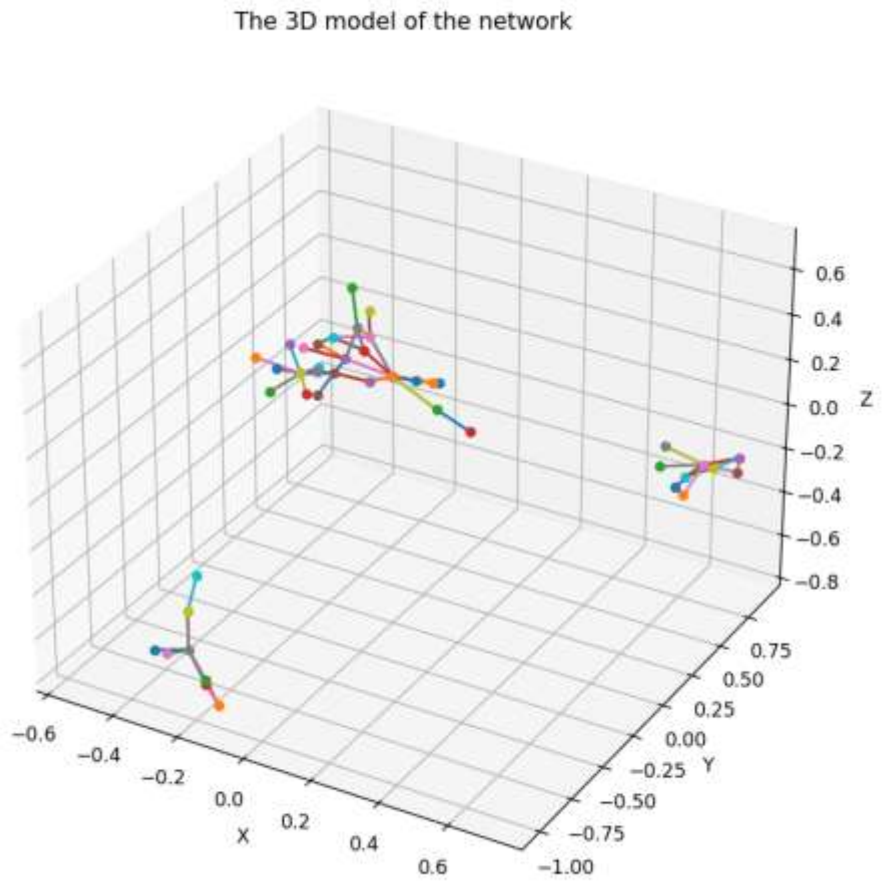
Firstly, I constructed the network in 2D dimensions, and then I used the same nodes and edges to build it in 3D. I then used 3D to plot a figure after changing its dimensions and projection. In order to show the network clearly in the 3D model, I ensured that the layout is a spring_layout. I looped over the nodes from the layout's dictionary and scattered them, changing their labels according to the names of the airports, in order to plot the nodes. As seen in figure (7), I also plotted lines in the x, y, and z coordinates between each pair of points to represent the edges.



```
fig = plt.figure(figsize=(15, 12))
ax = fig.add_subplot(111, projection='3d')

pos = nx.spring_layout(G, dim=3)
for node, (x, y, z) in pos.items():
    ax.scatter(x, y, z, label=node)

for edge in G.edges():
    ax.plot( *args: [pos[edge[0]][0], pos[edge[1]][0]],
            [pos[edge[0]][1], pos[edge[1]][1]],
            [pos[edge[0]][2], pos[edge[1]][2]])

ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Z')
ax.set_title("The 3D model of the network")
plt.show()
```

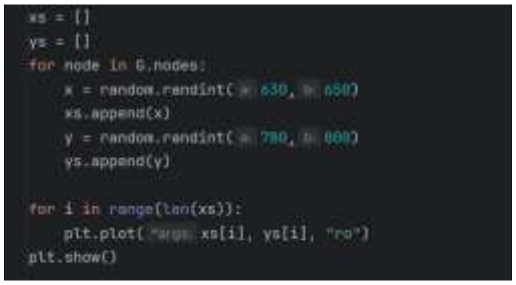
Figure (8): The figure shows the plotting of nodes and edges in the 3D modelling stage

And the output is:



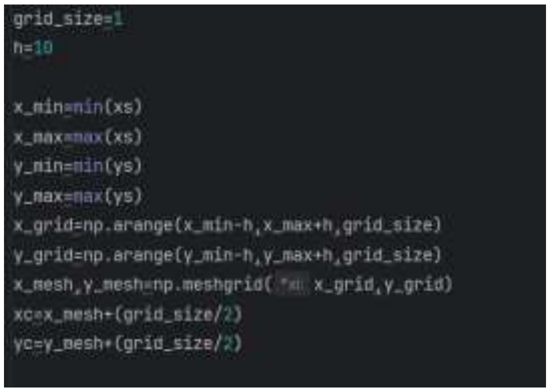
Fifth: Heatmap:

In the end, I applied the heatmap to the data, displaying each airport's location on a map of the world map. I started by giving the airport's locations range of x and y coordinates to plot, then I looped over them to create the figure (8).
Next, as seen in figure (9), I drew the x and y mesh to compare and illustrate the impact of the betweenness and frequency on the points and how their color will change as a result on the map.
The KDE function that applies the equation comes next. And in order to put it into practice, I iterated over each row in the image to check if it is inside the radius. If it is, the function is run again, and the pixel is colored as seen in figure (10).
Finally, after applying KDE, I displayed the colors on the mesh grid and used the addWeighted function to blend the colored mesh grid picture with the image of the world map, allowing the positions of the airports on the world map to be seen as illustrated in figure (11).
This is the output:



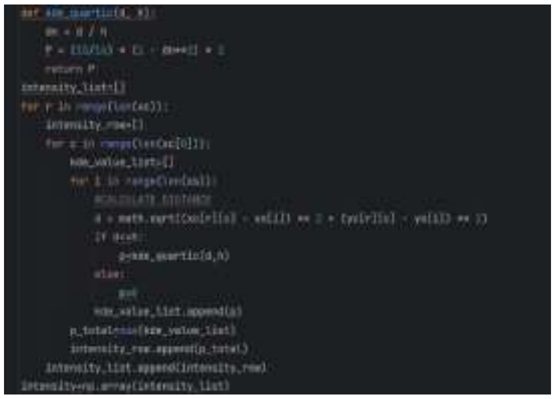
```
xs = []
ys = []
for node in G.nodes:
    x = random.randint(630, 650)
    xs.append(x)
    y = random.randint(780, 800)
    ys.append(y)

for i in range(len(xs)):
    plt.plot(xs[i], ys[i], "ro")
plt.show()
```

Figure (8): The figure shows the plotting of points in the heatmap stage



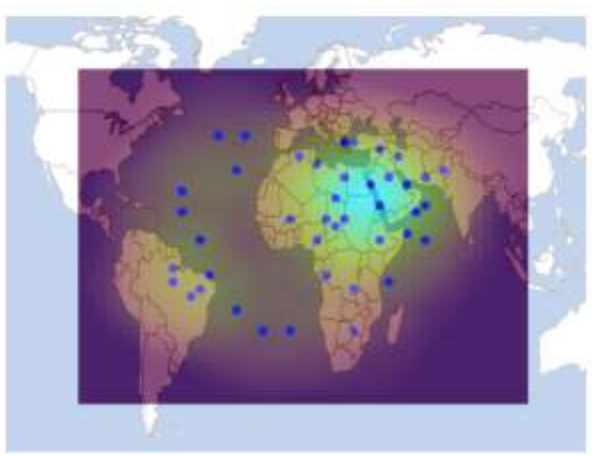
```
grid_size=1
h=10

x_min=min(xs)
x_max=max(xs)
y_min=min(ys)
y_max=max(ys)
x_grid=np.arange(x_min-h,x_max+h,grid_size)
y_grid=np.arange(y_min-h,y_max+h,grid_size)
x_mesh,y_mesh=np.meshgrid(x_grid,y_grid)
xc=x_mesh+(grid_size/2)
yc=y_mesh+(grid_size/2)
```

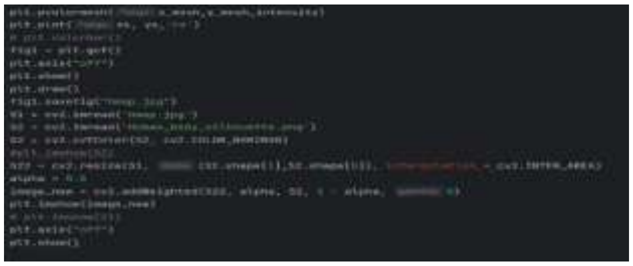
Figure (9): The figure shows drawing the mesh in the heatmap stage



```
def kde_quartic(d, h):
    dn = d / h
    P = (15/16) * (1 - dn**2) ** 2
    return P
intensity_list=[]
for r in range(len(xc)):
    intensity_row=[]
    for c in range(len(xc[0])):
        kde_value_list=[]
        for i in range(len(xs)):
            #CALCULATE DISTANCE
            d = math.sqrt((xc[r][c] - xs[i]) ** 2 + (yc[r][c] - ys[i]) ** 2)
            if d<=h:
                p=kde_quartic(d,h)
            else:
                p=0
            kde_value_list.append(p)
        p_total=sum(kde_value_list)
        intensity_row.append(p_total)
    intensity_list.append(intensity_row)
intensity=np.array(intensity_list)
```

Figure (10): The figure shows applying the KDE in the heatmap stage



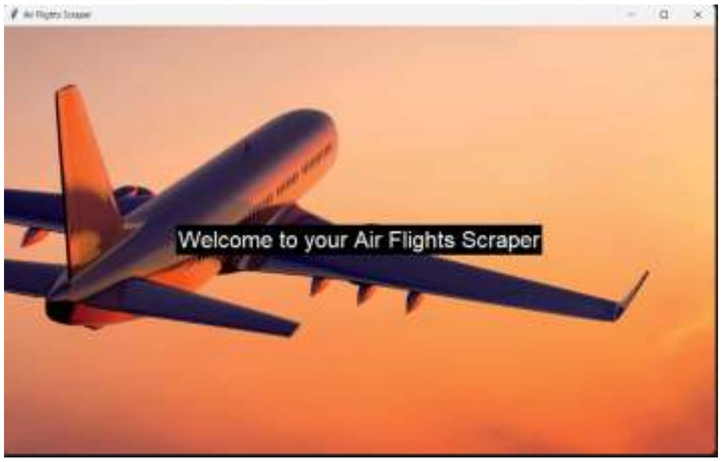


```
plt.pcolormesh(x_mesh,y_mesh,intensity)
plt.plot(xs, ys, 'ro')
# plt.colorbar()
fig1 = plt.gcf()
plt.axis("off")
plt.show()
plt.draw()
fig1.savefig('heap.jpg')
S1 = cv2.imread('heap.jpg')
S2 = cv2.imread('[illegible].png')
S2 = cv2.cvtColor(S2, cv2.COLOR_BGR2RGB)
#plt.imshow(S2)
S22 = cv2.resize(S1, (S2.shape[1],S2.shape[0]), interpolation = cv2.INTER_AREA)
alpha = 0.5
image_new = cv2.addWeighted(S22, alpha, S2, 1 - alpha, 0)
plt.imshow(image_new)
# plt.imshow(S1)
plt.axis("off")
plt.show()
```

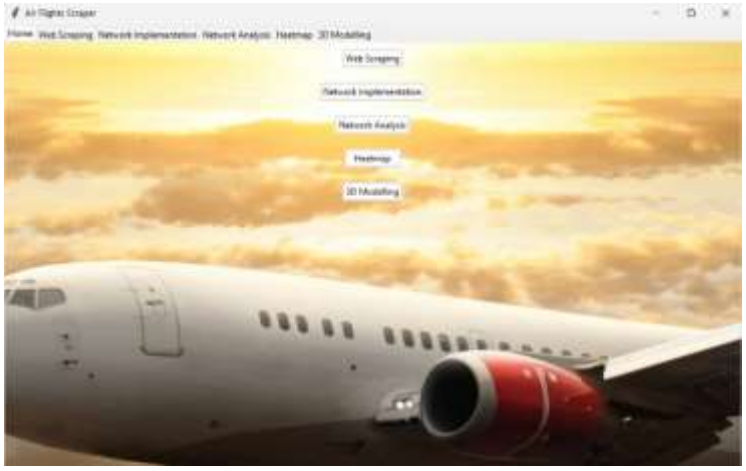
Figure (11): The figure shows merging the two photos and plotting them in the heatmap stage

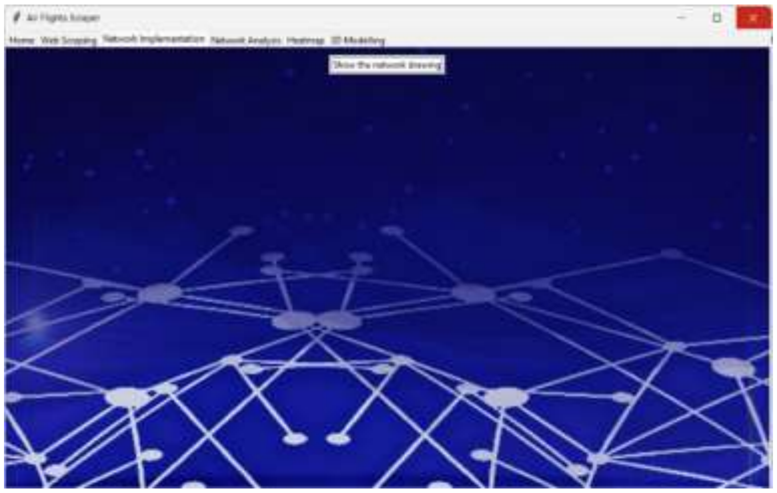
Sixth: GUI:

After implementing the whole project, I made a GUI with Tkinter to add some visualization to it. This is the first startup splash screen which is an image of a plane that appears for 5 seconds.



After that, I made a home tab and tab for each stage, with buttons in it to show the data and outputs when clicked.


Air Flights Scraper
Home
Web Scraping
Network Implementation
Network Analysis
Heatmap
3D Modelling
Web Scraping
Network Implementation
Network Analysis
Heatmap
3D Modelling


Air Flights Scraper
Home
Web Scraping
Network Implementation
Network Analysis
Heatmap
3D Modelling
Show the network drawing