In [1]:
```
from IPython import display
display.Image(url='https://images.unsplash.com/photo-1470506926202-05d3fca84c9a?ixlib=
              embed=True,
              alt="Photo by N Ellladee on Unsplash",
              width=600,
              height=400
             )
```

Out[1]:

Introduction

Now it so happens that we often want to calculate the shortest distance between two points. Shortest distance calculation is the simplest form of route analysis. Route analysis, on the other hand, is part of a large umbrella of GIS analytical works known as network analysis. Within route analysis, one can engage to find out the shortest, fastest or even the most scenic route. It all depends on the impedance you are trying to overcome.

To demonstrate shortest route analysis, we shall work with two points. These two points are Vasco da Gama Pillar and Gedi Ruins along the Kenyan coast. We have come across them in a previous exercise before, and today, they shall also be our guests, again.

Get the coordinates

As always, give a geographer a point on the earth's surface and they shall make their way around. Here are the coordinates for our beloved two points.

| Name | Longitude | Latitude | | ---- | --------- | -------- | | Vasco da Gama Pillar | 40.1276701 | -3.2236304 | | Gedi Ruins | 39.962392 | -3.334307 |

In order to conduct geospatial experiments with the above data, we have to convert it to a geodataframe. As always, the necessary tools of the trade have to be loaded first.

In [2]:
```
import pandas as pd
import geopandas as gpd
```

In [3]:
```
# Create dataframe for our places of interest
df = pd.DataFrame({
    'Name': ['Vasco da Gama', 'Gedi Ruins'],
    'Longitudes': [40.1276701, 39.962392], # These here are longitudes
    'Latitudes': [-3.2236304, -3.334307]} # These here are latitudes
)
```

In [4]:
```
# Convert the above dataframe to geodataframe
coast_gdf = gpd.GeoDataFrame(df,
                             geometry=gpd.points_from_xy(df.Longitudes, df.Latitudes)
                            )
```

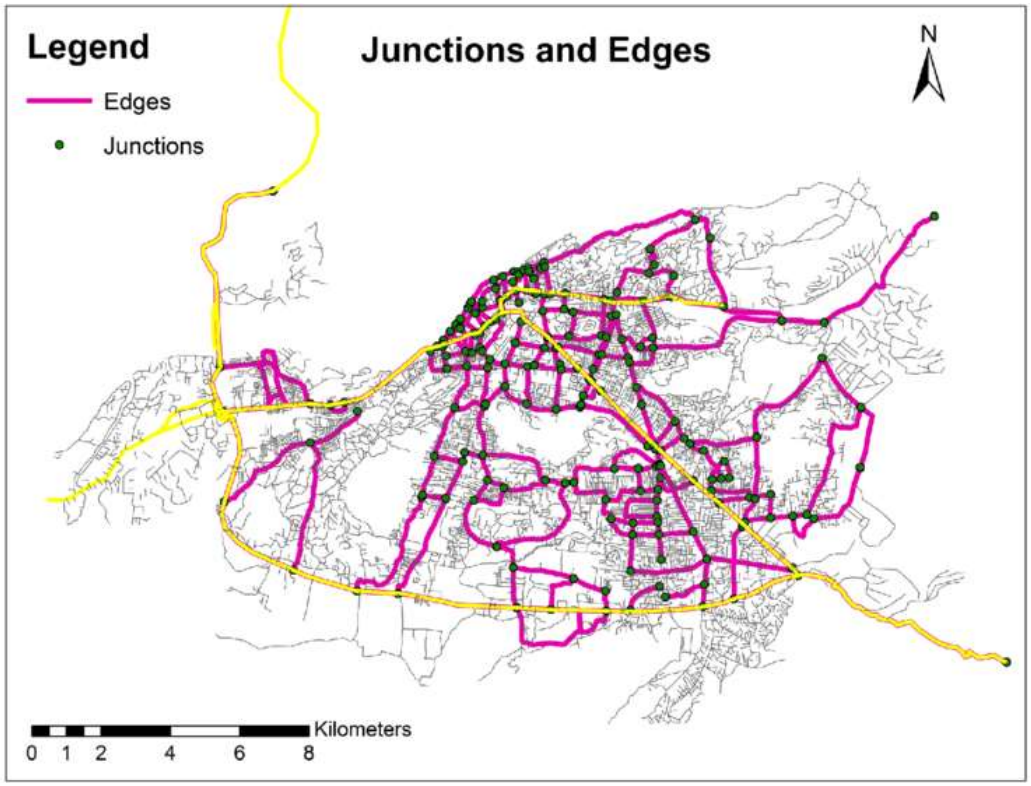
In [5]:
```
# Read the geodataframe table
coast_gdf.head()
```

Out[5]:

	Name	Longitudes	Latitudes	geometry
0	Vasco da Gama	40.127670	-3.223630	POINT (40.12767 -3.22363)
1	Gedi Ruins	39.962392	-3.334307	POINT (39.96239 -3.33431)

There we go!

Now, we would like to calculate the shortest route distance from grandiouse *Pillar* to ancient *Ruins* and extract the route. Our source of the route, the tarmac road between the two places, will be sourced from an Open Streetmap (OSM) network. The osmnx package, created by Geoff Boeing, is a tool which enables us to extract typical network components, such as edges, nodes and polygons from an OSM basemap. Now if you are new to GIS, the term edges and nodes may confuse you. But don't let it scare you either. You can think of edges as any line that connects between two points. For example, a road connecting from point A to B is an *edge*. Nodes are simply the points that mark the beginning or ending points of a line, refered to as *edges* in this article. A picture is worth a thousand words, and in here junctions and roads are examples of nodes and edges respectively.



Now, therefore, lets load the osmnx package.

In [6]:
```
# Load the osmnx package into Jupyter
import osmnx as ox
```

We want to extract the OSM data falling within the ground area limits of our two points. You can do so by specifying the upper and lower bounds (upper north, lower south, easternmost east and westernmost west) coordinate values. However, an easier method is to use a python tool that automatically calculates these values for us. The Geopandas method `total_bounds` does this for us automatically.

In [7]:
```
# First Calculate the bounds of our geodataframe
coast_gdf.total_bounds
```

Out[7]:
```
array([39.962392 , -3.334307 , 40.1276701, -3.2236304])
```

The tool `graph_from_bbox` which we shall use to extract the OSM data requires inserting the above coordinates in a particular order--that is, north, south, east, west. Therefore, we shall proceed to do so as prescribed.

In [8]:
```
# Derive osm graph from bounding box using coordinates
area_osm_graph = ox.graph_from_bbox(-3.2100000, -3.3400000, 40.1300000, 39.9500000,
                                    network_type="drive",
                                    simplify=True,
                                    )
```

Now, after we have defined the area in which we shall extract our OSM data, we need to plot it. The method to plot our OSM network is `plot_graph` .

In [9]:
```
# Project the osm graph
ox.plot_graph(area_osm_graph)
```

Out[9]: (<Figure size 800x800 with 1 Axes>, <AxesSubplot: >)

Experienced programmers would ask why proceed to typing in the coordinates manually into the `graph_from_bbox` function when Python has an innumerable number of tools that can do both sequentially: calculating the total bounds and creating a polygon from them, such as in here. However, upon using that method as explained in the stack overflow response, the `graph_from_bbox` did not recognise our polygon. After this disappointment, it's best to stick to what we are sure of.

Now, in order to calculate the shortest distance, we have to plot not only the edges, in our case the roads, but also the nodes as well. Why the nodes? It's because the nodes will define the starting point and ending point of our route. Therefore, to plot these edges and nodes, they have to be extracted from the above graph by type.

Let's first start with edges.

In [10]:
```
# Retrieve edges
edges = ox.graph_to_gdfs(area_osm_graph, nodes=False, edges=True)
edges.head()
```

Out[10]:

u	v	key	osmid	ref	name	highway	oneway	reversed	length	geo
308935402	**6360256831**	**0**	51728703	B8	Malindi Road	primary	False	False	201.422	LINES (40. -3.2 40. -3
	3759325238	**0**	51728703	B8	Malindi Road	primary	False	True	282.354	LINES (40. -3.2 40. -3
	2286758026	**0**	219527389	NaN	NaN	residential	False	False	109.363	LINES (40. -3.2 40. -3
308935403	**6360256828**	**0**	51728703	B8	Malindi Road	primary	False	False	10.657	LINES (40. -3.2 40. -3.2
	6360256831	**0**	51728703	B8	Malindi Road	primary	False	True	7.015	LINES (40. -3.2 40. -3.2

Now, let's retrieve the nodes as well.

In [11]:
```
# Retrieve nodes
nodes = ox.graph_to_gdfs(area_osm_graph, nodes=True, edges=False)
nodes.head()
```

Out[11]:

osmid	y	x	street_count	geometry
308935402	-3.224505	40.109595	3	POINT (40.10960 -3.22450)
308935403	-3.223338	40.111065	3	POINT (40.11107 -3.22334)
308935409	-3.219258	40.116571	3	POINT (40.11657 -3.21926)
308938531	-3.219440	40.116660	3	POINT (40.11666 -3.21944)
308938533	-3.219472	40.116969	3	POINT (40.11697 -3.21947)

We are now getting closer to conducting our route analysis. How we wish some of these things were as easy as pushing a button. Clearly, they are not! Two crucial nodes determine the eligibility of a route analysis: 1) the starting point, and 2) the destination. Each of these two nodes, just like every other node, has a unique id. For shortest route distance to work, the

computer will calculate which node ids between the start and end point cover the least distance. Let's get the node id for our starting point, which we have christened as `origin`.

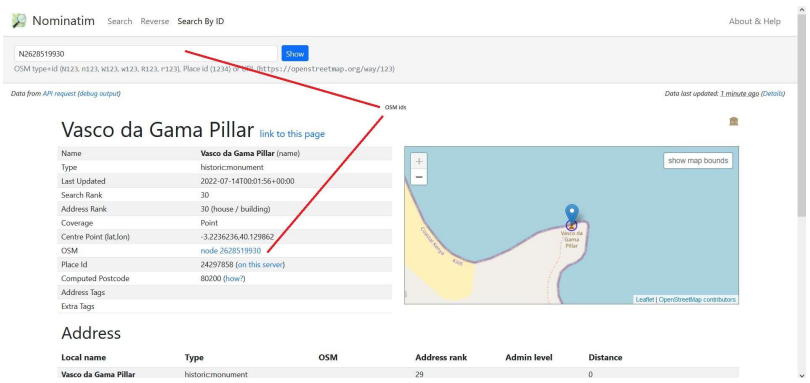
In [12]:
```
# The starting point, Vasco da Gama Pillar
origin = (
    ox.geocode_to_gdf('N2628519930', by_osmid=True)  # fetch geolocation of Vasco da G
    .to_crs(edges.crs)  # transform to global CRS
    .at[0, "geometry"]  # pick geometry of first row
    .centroid  # use the centre point
)
```

In [13]:
```
# the end point, Gedi Ruins
destination = (
    ox.geocode_to_gdf("N656556828", by_osmid=True) # fetch geolocation of Gedi ruins
    .to_crs(edges.crs)
    .at[0, "geometry"]
    .centroid
)
```

Let us go through the above two codes a bit. The intimidating function here is `geocode_to_gdf`. Apart from the very assistive explanation in the help content, why did we use alpha-numeric values rather than our simple place-names of "Vasco da Gama Pillar" and "Malindi International Airport"? The issue was that somehow `geocode_to_gdf` did not understand these place names. Numerous tries only translated to numerous errors. Luckily, the OpenStreetMap Nominatim website website has an avenue where one can also obtain the OSM id of a place after keying in the place-name. That's what we did, and those are the place-codes we got. After inserting these OSM ids in place of their string names into the `geocode_to_gdf` function, the function worked without a fuss.

In [14]:
```
from IPython import display
display.Image("E:/documents/gis800_articles/jupyter/route_analysis/nominatim.jpg", alt
```

Out[14]:



There is still another problem though. What if our nodes, the places of interest, are not along the edges? Yes, it is very much possible! If a node is not a long the edge, osmnx will be unable to locate it and thus will not carry out the route analysis. So what do we do? The `nearest_nodes` function finds the nearest nodes to our points, those along the edges of our graph network-- `area_osm_graph` -- and because they *exist*, they will be interlinkable to other nodes along the line.

In [15]:
```
# find the nearest node to our start point
origin_node_id = ox.nearest_nodes(area_osm_graph, origin.x, origin.y)
origin_node_id
```

Out[15]: 4561904751

In [16]:
```
# find the nearest node to our end point
destination_node_id = ox.nearest_nodes(area_osm_graph, destination.x, destination.y)
destination_node_id
```

Out[16]: 4311298754

Let's find out if our derived origin id exists along the line.

In [17]:
```
# find if the origin osm id exists.
index = nodes.index  # Get's the index

for i in index:
    if i == 4561904751:
        print("True, that osm id exists")
        print(nodes.loc[[i]])
        break
```

```
True, that osm id exists
                   y          x  street_count                   geometry
osmid
4561904751 -3.224063  40.127968             1  POINT (40.12797 -3.22406)
```

Ok. For sure the osm id for `origin_node_id` exists. But first, what wizardry did we do? Functions can be hard to wrap our head around sometimes, so let's break it down to help fellow programming beginners who'd wish they were pros before they even started. If wishes were horses, beggars should ride on them. Don't you think?

Let's go line by line.

```
# find if the origin osm id exists.
index = nodes.index
```

This should be pretty straightforward. The method `index` assigns the indexes of the nodes to the created `index` object.

```
for i in index:
```

Very simple. The `for` loop iterates over every index, given the pseudonym `i` here --within the object - `index`.

```
if i == 4561904751:
        print("True, that osm id exists")
        print(nodes.loc[[i]])
```

Now let's go the above suite line by line. The `if` statement says that for any index `i` within the `index` list that equals to the value `4561904751`, then do this: print `"True, that osm id exists"` and then show the particular row using:

```
print(nodes.loc[[i]])
```

The `loc` function is actually used in accessing row and column values. In here it has been used to access the index rows using the following format-- `.loc['index']`. But when it is used together with `[[]]`, we are commanding `loc` to show that specific index *only* which will subsequently print out the row attached to that index.

Finally, `break` does what it simply says, it *breaks* out from the loop.

How about the osm id for our destination, the `destination_node_id` object? does it exist?

In [18]:
```
# Find if the osm id for destination exists -- 3759325223
for i in index:
    if i == 4311298754:
        print("True, that osm id exists")
        print(nodes.loc[[i]])
        break
```

```
True, that osm id exists
                   y          x  street_count                   geometry
osmid
4311298754 -3.305467  40.017559             3  POINT (40.01756 -3.30547)
```

Now time for the big thing, calculating the shortest route between our origin and destination nodes.

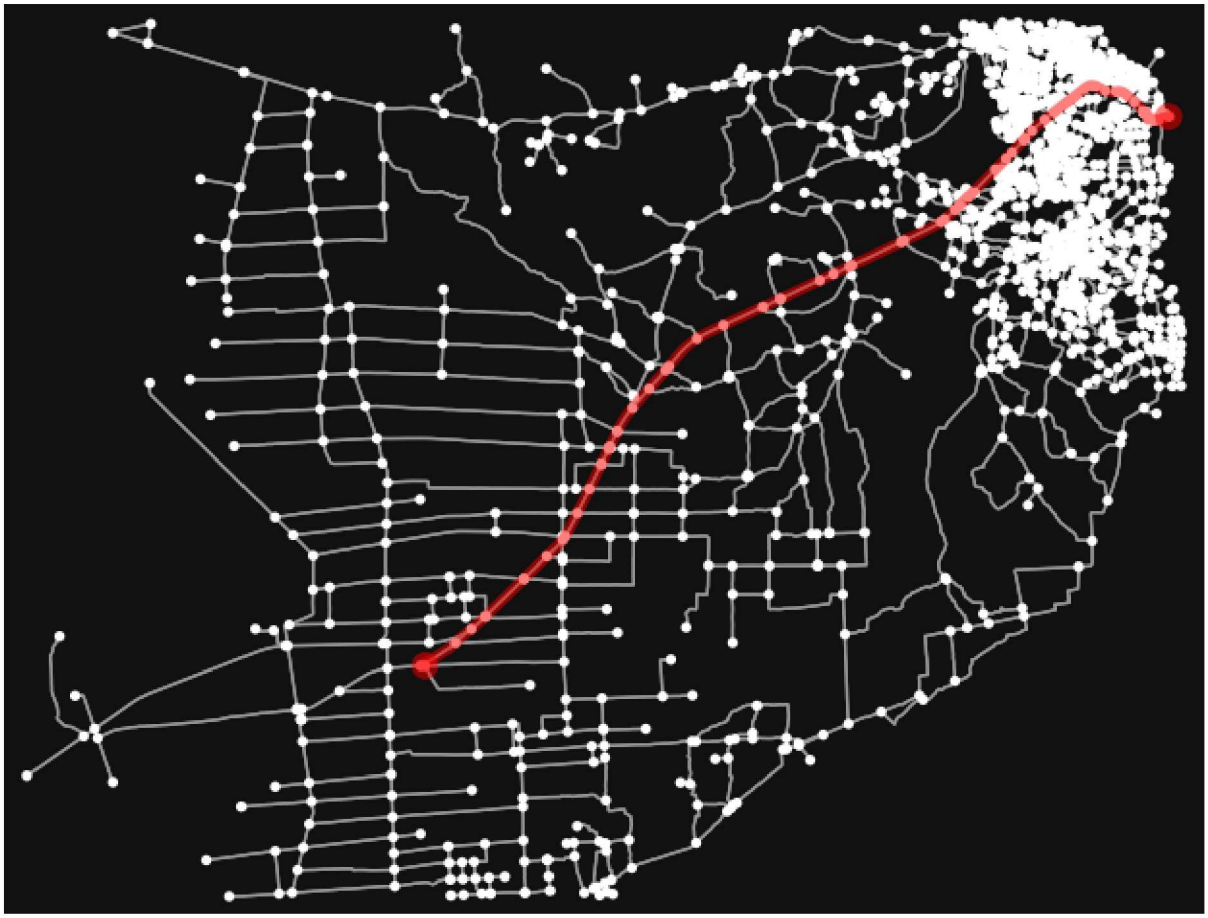
In [19]:
```
# Find the shortest path between origin and destination
route = ox.shortest_path(area_osm_graph, origin_node_id, destination_node_id) # The fu
route
```

Out[19]:
```
[4561904751,
 1329651956,
 4561904755,
 4561904744,
 4561904752,
 4561904740,
 308938819,
 6354836742,
 6354836752,
 308938818,
 6354836751,
 6354419322,
 308938812,
 9956464530,
 308938811,
 9956464477,
 6352656127,
 6352656139,
 2471753125,
 6352656137,
 308938804,
 5040196387,
 308938533,
 308938531,
 6372042772,
 4561904768,
 4561904758,
 4561905031,
 6360484057,
 3687005204,
 308938549,
 6360256813,
 6360256809,
 6360256828,
 308935403,
 6360256831,
 308935402,
 3759325238,
 2286758097,
 3759325251,
 3759325253,
 4561905052,
 6354804213,
 5736493084,
 5736493085,
 6344314013,
 6344514591,
 5736508050,
 6344454617,
 6322408262,
 6344454498,
 6322408316,
 6322408334,
 5398197506,
 6322408404,
 6322408284,
 6539960867,
```

```
 6539960846,
 6545223756,
 6322467847,
 6532206463,
 6545420665,
 6532206427,
 6545518751,
 6544691243,
 308940212,
 6544793789,
 6518765212,
 6521602455,
 6518794678,
 6521602457,
 6545223712,
 308940210,
 6521376566,
 6521716578,
 308940208,
 6521602498,
 6545109615,
 6521602491,
 6521788132,
 6558177011,
 6560665470,
 6548008111,
 6561215851,
 4311280462,
 4311298754]
```

Time to plot our route overlaying the entire road network of our area of interest. Even though the `plot_graph` function was competent in plotting our OSM network layer, the best tool for drawing our shortest route layer is in fact the `plot_graph_route` function. Here it is at work in all its glory.

In [20]:
```
# Plot the shortest path
fig, ax = ox.plot_graph_route(area_osm_graph, route)
```



The osmnx package does a good job of linking one node to another sequentially until we get a route. However, our route layer lacks many other attributes that will be helpful in assigning it to a location in space, such as `geometry` and lat lon coordinates. Remember the `route` object is a list of nodes, which `plot_graph route` has systematically ordered to form a path. How do we solve this problem? One way would be to use a `join` function where only those rows matching in both datasets (`nodes` and `route`) are retained, but the `for` loop if smartly used can achieve this objective as well.

In [21]:
```
# Get the nodes along the shortest path that also exist in the route geodataframe
route_nodes_ids = []
for i in nodes.index:
    if i in route:
        route_nodes_ids.append(nodes.loc[i])
print(route_nodes_ids)
print(len(route_nodes_ids))
```

```
[y                                  -3.224505
x                                  40.109595
street_count                               3
geometry        POINT (40.1095954 -3.2245045)
Name: 308935402, dtype: object, y                                  -3.223338
x                                  40.111065
street_count                               3
geometry        POINT (40.1110652 -3.2233382)
Name: 308935403, dtype: object, y                                   -3.21944
x                                   40.11666
street_count                               3
geometry        POINT (40.1166596 -3.2194401)
Name: 308938531, dtype: object, y                                  -3.219472
x                                  40.116969
street_count                               3
geometry        POINT (40.1169694 -3.2194715)
Name: 308938533, dtype: object, y                                 -3.222092
x                                 40.112633
street_count                              3
geometry        POINT (40.112633 -3.2220916)
Name: 308938549, dtype: object, y                                   -3.22019
x                                  40.118542
street_count                               3
geometry        POINT (40.1185425 -3.2201903)
Name: 308938804, dtype: object, y                                  -3.22045
x                                 40.121172
street_count                              3
geometry        POINT (40.1211718 -3.22045)
Name: 308938811, dtype: object, y                                   -3.22073
x                                  40.121648
street_count                               3
geometry        POINT (40.1216479 -3.2207299)
Name: 308938812, dtype: object, y                                  -3.223344
x                                  40.124216
street_count                               3
geometry        POINT (40.1242161 -3.2233436)
Name: 308938818, dtype: object, y                                  -3.223319
x                                  40.124545
street_count                               3
geometry        POINT (40.1245448 -3.2233187)
Name: 308938819, dtype: object, y                                  -3.286291
x                                   40.03832
street_count                               4
geometry        POINT (40.0383197 -3.2862912)
Name: 308940208, dtype: object, y                                  -3.275611
x                                  40.043698
street_count                               3
geometry        POINT (40.0436975 -3.2756107)
Name: 308940210, dtype: object, y                                  -3.267248
x                                  40.048383
street_count                               3
geometry        POINT (40.0483833 -3.2672479)
Name: 308940212, dtype: object, y                                 -3.224536
x                                 40.126552
street_count                              4
geometry        POINT (40.126552 -3.2245362)
Name: 1329651956, dtype: object, y                                  -3.227304
```

```
x                                    40.106656
street_count                                 3
geometry        POINT (40.1066556 -3.2273036)
Name: 2286758097, dtype: object, y                                    -3.220303
x                                    40.120356
street_count                                 3
geometry        POINT (40.1203564 -3.2203027)
Name: 2471753125, dtype: object, y                                    -3.221982
x                                    40.112772
street_count                                 3
geometry        POINT (40.1127724 -3.2219818)
Name: 3687005204, dtype: object, y                                   -3.226188
x                                   40.107697
street_count                                3
geometry        POINT (40.1076969 -3.226188)
Name: 3759325238, dtype: object, y                                    -3.227352
x                                    40.106625
street_count                                 3
geometry        POINT (40.1066248 -3.2273525)
Name: 3759325251, dtype: object, y                                    -3.227541
x                                    40.106442
street_count                                 3
geometry        POINT (40.1064424 -3.2275413)
Name: 3759325253, dtype: object, y                                    -3.305464
x                                    40.016924
street_count                                 3
geometry        POINT (40.0169244 -3.3054641)
Name: 4311280462, dtype: object, y                                    -3.305467
x                                    40.017559
street_count                                 3
geometry        POINT (40.0175595 -3.3054665)
Name: 4311298754, dtype: object, y                                    -3.223304
x                                    40.124589
street_count                                 3
geometry        POINT (40.1245893 -3.2233044)
Name: 4561904740, dtype: object, y                                    -3.224332
x                                    40.125752
street_count                                 3
geometry        POINT (40.1257516 -3.2243317)
Name: 4561904744, dtype: object, y                                    -3.224063
x                                    40.127968
street_count                                 1
geometry        POINT (40.1279677 -3.2240629)
Name: 4561904751, dtype: object, y                                    -3.223959
x                                    40.124761
street_count                                 3
geometry        POINT (40.1247614 -3.2239587)
Name: 4561904752, dtype: object, y                                    -3.224674
x                                    40.125825
street_count                                 3
geometry        POINT (40.1258247 -3.2246745)
Name: 4561904755, dtype: object, y                                    -3.219989
x                                    40.115313
street_count                                 3
geometry        POINT (40.1153125 -3.2199894)
Name: 4561904758, dtype: object, y                                    -3.219424
x                                    40.116237
```

```
street_count                                3
geometry        POINT (40.1162371 -3.2194244)
Name: 4561904768, dtype: object, y                                  -3.22083
x                                  40.114248
street_count                                3
geometry        POINT (40.1142477 -3.2208303)
Name: 4561905031, dtype: object, y                                 -3.229291
x                                  40.104717
street_count                                3
geometry        POINT (40.1047171 -3.2292906)
Name: 4561905052, dtype: object, y                                  -3.21952
x                                   40.11721
street_count                                3
geometry        POINT (40.1172102 -3.2195205)
Name: 5040196387, dtype: object, y                                 -3.246044
x                                  40.081011
street_count                                3
geometry        POINT (40.0810111 -3.2460444)
Name: 5398197506, dtype: object, y                                 -3.231847
x                                  40.102202
street_count                                3
geometry        POINT (40.1022018 -3.2318474)
Name: 5736493084, dtype: object, y                                 -3.235339
x                                  40.098793
street_count                                3
geometry        POINT (40.0987928 -3.2353394)
Name: 5736493085, dtype: object, y                                 -3.237273
x                                  40.096902
street_count                                3
geometry        POINT (40.0969021 -3.2372729)
Name: 5736508050, dtype: object, y                                 -3.239279
x                                  40.094712
street_count                                3
geometry        POINT (40.0947117 -3.2392787)
Name: 6322408262, dtype: object, y                                 -3.247381
x                                  40.078204
street_count                                3
geometry        POINT (40.0782041 -3.2473807)
Name: 6322408284, dtype: object, y                                -3.242456
x                                 40.088685
street_count                                3
geometry        POINT (40.0886849 -3.242456)
Name: 6322408316, dtype: object, y                                 -3.242692
x                                  40.088218
street_count                                3
geometry        POINT (40.0882181 -3.2426921)
Name: 6322408334, dtype: object, y                                 -3.246232
x                                  40.080609
street_count                                3
geometry        POINT (40.0806087 -3.2462319)
Name: 6322408404, dtype: object, y                                  -3.25227
x                                  40.067725
street_count                                3
geometry        POINT (40.0677247 -3.2522699)
Name: 6322467847, dtype: object, y                                 -3.235949
x                                 40.098204
street_count                                3
```

```
geometry        POINT (40.098204 -3.2359487)
Name: 6344314013, dtype: object, y                                  -3.239684
x                                 40.094182
street_count                              3
geometry        POINT (40.0941821 -3.239684)
Name: 6344454498, dtype: object, y                                  -3.237895
x                                 40.096283
street_count                              3
geometry        POINT (40.0962832 -3.2378952)
Name: 6344454617, dtype: object, y                                  -3.237119
x                                  40.09706
street_count                              3
geometry        POINT (40.0970602 -3.2371191)
Name: 6344514591, dtype: object, y                                 -3.220287
x                                 40.12052
street_count                              3
geometry        POINT (40.12052 -3.2202873)
Name: 6352656127, dtype: object, y                                  -3.220334
x                                 40.120177
street_count                              3
geometry        POINT (40.1201767 -3.2203335)
Name: 6352656137, dtype: object, y                                  -3.220291
x                                 40.120415
street_count                              3
geometry        POINT (40.1204154 -3.2202907)
Name: 6352656139, dtype: object, y                                  -3.221843
x                                  40.12288
street_count                              3
geometry        POINT (40.1228799 -3.2218431)
Name: 6354419322, dtype: object, y                                  -3.230218
x                                 40.103805
street_count                              3
geometry        POINT (40.1038045 -3.2302185)
Name: 6354804213, dtype: object, y                                  -3.223348
x                                  40.12448
street_count                              3
geometry        POINT (40.1244796 -3.2233483)
Name: 6354836742, dtype: object, y                                  -3.223228
x                                 40.124158
street_count                              3
geometry        POINT (40.124158 -3.2232283)
Name: 6354836751, dtype: object, y                                  -3.223425
x                                  40.12425
street_count                              3
geometry        POINT (40.1242496 -3.2234252)
Name: 6354836752, dtype: object, y                                  -3.222914
x                                 40.111615
street_count                              3
geometry        POINT (40.1116154 -3.2229137)
Name: 6360256809, dtype: object, y                                   -3.22287
x                                 40.111668
street_count                              3
geometry        POINT (40.1116683 -3.2228705)
Name: 6360256813, dtype: object, y                                  -3.223277
x                                 40.111139
street_count                              3
geometry        POINT (40.1111394 -3.2232774)
```

```
Name: 6360256828, dtype: object, y                                  -3.223377
x                                    40.111015
street_count                                 3
geometry        POINT (40.1110153 -3.2233769)
Name: 6360256831, dtype: object, y                                  -3.221421
x                                    40.113496
street_count                                 3
geometry        POINT (40.1134961 -3.2214207)
Name: 6360484057, dtype: object, y                                  -3.219398
x                                    40.116354
street_count                                 3
geometry        POINT (40.1163545 -3.2193982)
Name: 6372042772, dtype: object, y                                   -3.27079
x                                   40.046076
street_count                                3
geometry        POINT (40.046076 -3.27079)
Name: 6518765212, dtype: object, y                                  -3.273219
x                                     40.04489
street_count                                 3
geometry        POINT (40.0448897 -3.2732194)
Name: 6518794678, dtype: object, y                                  -3.279216
x                                    40.041947
street_count                                 4
geometry        POINT (40.0419467 -3.2792164)
Name: 6521376566, dtype: object, y                                  -3.272941
x                                    40.045021
street_count                                 3
geometry        POINT (40.0450212 -3.2729409)
Name: 6521602455, dtype: object, y                                  -3.273312
x                                    40.044839
street_count                                 3
geometry        POINT (40.0448388 -3.2733118)
Name: 6521602457, dtype: object, y                                  -3.292531
x                                    40.032293
street_count                                 3
geometry        POINT (40.0322934 -3.2925311)
Name: 6521602491, dtype: object, y                                  -3.286813
x                                    40.037943
street_count                                 4
geometry        POINT (40.0379434 -3.2868126)
Name: 6521602498, dtype: object, y                                  -3.282834
x                                    40.040149
street_count                                 4
geometry        POINT (40.0401489 -3.2828335)
Name: 6521716578, dtype: object, y                                   -3.292591
x                                   40.032232
street_count                                3
geometry        POINT (40.032232 -3.2925906)
Name: 6521788132, dtype: object, y                                    -3.26101
x                                    40.053742
street_count                                 3
geometry        POINT (40.0537424 -3.2610105)
Name: 6532206427, dtype: object, y                                  -3.254973
x                                    40.061887
street_count                                 3
geometry        POINT (40.0618869 -3.2549725)
Name: 6532206463, dtype: object, y                                  -3.251029
```

```
x                                    40.070369
street_count                                 3
geometry        POINT (40.0703687 -3.2510287)
Name: 6539960846, dtype: object, y                                    -3.248305
x                                    40.076172
street_count                                 3
geometry        POINT (40.0761723 -3.2483046)
Name: 6539960867, dtype: object, y                                    -3.264377
x                                     40.05086
street_count                                 3
geometry        POINT (40.0508597 -3.2643772)
Name: 6544691243, dtype: object, y                                   -3.270667
x                                   40.046136
street_count                                 3
geometry        POINT (40.046136 -3.2706669)
Name: 6544793789, dtype: object, y                                    -3.289252
x                                    40.035657
street_count                                 3
geometry        POINT (40.0356575 -3.2892515)
Name: 6545109615, dtype: object, y                                    -3.275532
x                                    40.043737
street_count                                 3
geometry        POINT (40.0437374 -3.2755316)
Name: 6545223712, dtype: object, y                                    -3.251095
x                                    40.070218
street_count                                 3
geometry        POINT (40.0702178 -3.2510949)
Name: 6545223756, dtype: object, y                                    -3.257006
x                                    40.057905
street_count                                 3
geometry        POINT (40.0579052 -3.2570064)
Name: 6545420665, dtype: object, y                                    -3.261624
x                                    40.053223
street_count                                 3
geometry        POINT (40.0532226 -3.2616239)
Name: 6545518751, dtype: object, y                                    -3.300016
x                                    40.024394
street_count                                 3
geometry        POINT (40.0243936 -3.3000162)
Name: 6548008111, dtype: object, y                                    -3.298079
x                                     40.02658
street_count                                 3
geometry        POINT (40.0265802 -3.2980794)
Name: 6558177011, dtype: object, y                                    -3.298215
x                                    40.026442
street_count                                 4
geometry        POINT (40.0264421 -3.2982148)
Name: 6560665470, dtype: object, y                                   -3.302102
x                                   40.022036
street_count                                 3
geometry        POINT (40.022036 -3.3021021)
Name: 6561215851, dtype: object, y                                    -3.220408
x                                    40.121068
street_count                                 3
geometry        POINT (40.1210685 -3.2204081)
Name: 9956464477, dtype: object, y                                   -3.220679
x                                   40.121574
```

```
street_count                                3
geometry        POINT (40.121574 -3.2206794)
Name: 9956464530, dtype: object]
86
```

In [22]:
```
# Convert to Dataframe
geod1 = pd.DataFrame(route_nodes_ids)
geod1
```

Out[22]:

	y	x	street_count	geometry
308935402	-3.224505	40.109595	3	POINT (40.1095954 -3.2245045)
308935403	-3.223338	40.111065	3	POINT (40.1110652 -3.2233382)
308938531	-3.219440	40.116660	3	POINT (40.1166596 -3.2194401)
308938533	-3.219472	40.116969	3	POINT (40.1169694 -3.2194715)
308938549	-3.222092	40.112633	3	POINT (40.112633 -3.2220916)
...	...	...	...	...
6558177011	-3.298079	40.026580	3	POINT (40.0265802 -3.2980794)
6560665470	-3.298215	40.026442	4	POINT (40.0264421 -3.2982148)
6561215851	-3.302102	40.022036	3	POINT (40.022036 -3.3021021)
9956464477	-3.220408	40.121068	3	POINT (40.1210685 -3.2204081)
9956464530	-3.220679	40.121574	3	POINT (40.121574 -3.2206794)

86 rows × 4 columns

Let's do a quick check to see if our origin and destination ids are part of the `geod1` geodataframe we have created. Because a geodataframe has geometry values, it is possible to convert it to a LineString. The `elif` statement here simply stands for 'else if this, then do this'.

In [23]:
```
# Check if origin and destination exists in new dataframe
for i in geod1.index:
    if i == 4561904751:
        print(f"True, the origin osm id {i} exists")
        print(geod1.loc[[i]])
    elif i == 4311298754:
        print(f"\nTrue, the destination osm id {i} exists")
        print(geod1.loc[[i]])
 # The destination id has a lower value than the origin, if you insert `pass` after el
# higher value than the destination id
```

```
True, the destination osm id 4311298754 exists
                   y          x  street_count                        geometry
4311298754 -3.305467  40.017559             3  POINT (40.0175595 -3.3054665)
True, the origin osm id 4561904751 exists
                   y          x  street_count                        geometry
4561904751 -3.224063  40.127968             1  POINT (40.1279677 -3.2240629)
```

Here is a much more concise way.

In [24]:
```
# Check if origin id and destination id exist and print out confirmation
for i in geod1.index:
    if i == 4561904751 or i == 4311298754:
        print(f"\nOSM id {i} exists.")
```

```
OSM id 4311298754 exists.

OSM id 4561904751 exists.
```

Now it's time to create a linestring from the nodes in our shortest path --the `geod1` object. We shall do so using the `shapely.geometry` module. It's about time we used it. However, a dataframe containing the `geometry` column does not make it a geodataframe.

In [25]:
```
# Check class type of geod1
type(geod1)
```

Out[25]:
```
pandas.core.frame.DataFrame
```

In [26]:
```
# transform dataframe to geodataframe
geod1 = gpd.GeoDataFrame(geod1, geometry='geometry')

# Check if dataframe has been converted to geodataframe
print(type(geod1))
print("\n")

# Check length of geodataframe, it should be shorter than nodes because this is the sh
print(len(geod1))
print("\n")

# Print the crs of the geodataframe
print(geod1.crs)
print("\n")

# Print out the geodataframe
geod1.head()
```

```
<class 'geopandas.geodataframe.GeoDataFrame'>


86


None
```

Out[26]:

	y	x	street_count	geometry
308935402	-3.224505	40.109595	3	POINT (40.10960 -3.22450)
308935403	-3.223338	40.111065	3	POINT (40.11107 -3.22334)
308938531	-3.219440	40.116660	3	POINT (40.11666 -3.21944)
308938533	-3.219472	40.116969	3	POINT (40.11697 -3.21947)
308938549	-3.222092	40.112633	3	POINT (40.11263 -3.22209)

Ok. Our `geoid` object does not have a Coordinate Reference System (CRS) assigned yet. This is a problem so long as it remains a dataframe, but to conduct spatial operations on it, it has to be converted into a geodataframe. Converting a dataframe to geodataframe is easy, assigning a CRS to one is similarly easy. However, assigning a local CRS to our `geoid` object now causes erroneous reading during distance calculation. As such, we will save the assigning of CRS to the very latter stages. There is another albeit bigger problem. Our indexes are not ordered correctly, and if you create a LineString out of `geoid1` in its current format, the resulting path will look like a journey of a man who flew from Nairobi to Timbuktu on a plane that kept offsetting the location of the two locations by hundreds of meters. In short, the index will not link to each other correctly. To ameliorate this, the index of `geoid1` will be reconfigured to that of original shortest route layer, the `route` object.

In [27]:
```
# To reorder our route nodes with those of our initial route, otherwise the route will
geod1 = geod1.reindex(route)
geod1
```

Out[27]:

	y	x	street_count	geometry
4561904751	-3.224063	40.127968	1	POINT (40.12797 -3.22406)
1329651956	-3.224536	40.126552	4	POINT (40.12655 -3.22454)
4561904755	-3.224674	40.125825	3	POINT (40.12582 -3.22467)
4561904744	-3.224332	40.125752	3	POINT (40.12575 -3.22433)
4561904752	-3.223959	40.124761	3	POINT (40.12476 -3.22396)
...	...	...	...	...
6560665470	-3.298215	40.026442	4	POINT (40.02644 -3.29821)
6548008111	-3.300016	40.024394	3	POINT (40.02439 -3.30002)
6561215851	-3.302102	40.022036	3	POINT (40.02204 -3.30210)
4311280462	-3.305464	40.016924	3	POINT (40.01692 -3.30546)
4311298754	-3.305467	40.017559	3	POINT (40.01756 -3.30547)

86 rows × 4 columns

Now let's draw the route.

In [28]:
```
# Import the geometry module from the shapely package
from shapely import geometry as gm
```

In [29]:
```
# Create a geometry for the shortest path
route_line = gm.LineString(
    list(geod1.geometry.values)
)
route_line
```

Out[29]:

b' '

A sequential line. We can bet to the bank that if we didn't reindex, the route would have resembled a bird's nest.

In [30]:
```
# Check out the values in the route_nodes geometry
geod1.geometry
```

Out[30]:
```
4561904751    POINT (40.12797 -3.22406)
1329651956    POINT (40.12655 -3.22454)
4561904755    POINT (40.12582 -3.22467)
4561904744    POINT (40.12575 -3.22433)
4561904752    POINT (40.12476 -3.22396)
                        ...
6560665470    POINT (40.02644 -3.29821)
6548008111    POINT (40.02439 -3.30002)
6561215851    POINT (40.02204 -3.30210)
4311280462    POINT (40.01692 -3.30546)
4311298754    POINT (40.01756 -3.30547)
Name: geometry, Length: 86, dtype: geometry
```

Finally, let's create a table with a list of all the coordinates and OSM ids from which we shall calculate the length of the route.

In [31]:
```
# Extract useful data from the geodataframe of route
route_geom = gpd.GeoDataFrame(
    {
        "geometry": [route_line],
        "osm_nodes": [route],
    },
    crs=edges.crs  # The CRS of the edges object
)

route_geom
```

Out[31]:

	geometry	osm_nodes
0	LINESTRING (40.12797 -3.22406, 40.12655 -3.224...	[4561904751, 1329651956, 4561904755, 456190474...

To calculate the length of the route, it is best if our data is in a local Coordinate Reference System (CRS). Values of local CRSs are normally in metres or humanly interpretable units compared to degree radians of global CRSs. Since the function that follows this is of route calculation, we can now change the CRS of `route_geom`. It is bad practice to calculate length or area of a spatial feature using the global CRS, such as EPSG:4326. This is because the resulting values will be in degree radians, which are hard to interpret.

In [32]:
```
# Check the original CRS
route_geom.crs
```

Out[32]:
```
<Geographic 2D CRS: EPSG:4326>
Name: WGS 84
Axis Info [ellipsoidal]:
- Lat[north]: Geodetic latitude (degree)
- Lon[east]: Geodetic longitude (degree)
Area of Use:
- name: World.
- bounds: (-180.0, -90.0, 180.0, 90.0)
Datum: World Geodetic System 1984 ensemble
- Ellipsoid: WGS 84
- Prime Meridian: Greenwich
```

Now convert it to a local CRS.

In [33]:
```
# Convert the route_geom crs to EPSG:32737
route_geom = route_geom.to_crs("EPSG:32737")
route_geom.crs
```

Out[33]:
```
<Derived Projected CRS: EPSG:32737>
Name: WGS 84 / UTM zone 37S
Axis Info [cartesian]:
- E[east]: Easting (metre)
- N[north]: Northing (metre)
Area of Use:
- name: Between 36°E and 42°E, southern hemisphere between 80°S and equator, onshore
and offshore. Kenya. Mozambique. Tanzania.
- bounds: (36.0, -80.0, 42.0, 0.0)
Coordinate Operation:
- name: UTM zone 37S
- method: Transverse Mercator
Datum: World Geodetic System 1984 ensemble
- Ellipsoid: WGS 84
- Prime Meridian: Greenwich
```

To bring this to a close, (we promise!) we shall use the `length` method to calculate the length of our spatial feature, which will be in metres.

In [34]:
```
# Calculate the route length
route_geom["length_m"] = route_geom.length

route_geom.head()
```

Out[34]:

	geometry	osm_nodes	length_m
0	LINESTRING (625325.329 9643571.018, 625167.956...	[4561904751, 1329651956, 4561904755, 456190474...	16564.502509

Creating a function that converts metres to kilometres will not hurt, does it?

In [35]:
```
# Create function to convert column values in metres to km
def convert_m_km(geodataframe, column_name):
    return geodataframe.assign(length_km= lambda x: x.length_m / 1000)
```

The purpose of the above function is as follows. The `convert_m_km` function requires two inputs: a geodataframe (`geodataframe`) and one specific column (`column_name`). The function calculates the units in `column_name` into metres using the formula:

`km = row_column / 1000`

We have just minimized what would have been two lines into one using the lambda function.

In [36]:
```
# Implement the function with your route attributes
convert_m_km(route_geom, route_geom['length_m'])
```

Out[36]:

	geometry	osm_nodes	length_m	length_km
0	LINESTRING (625325.329 9643571.018, 625167.956...	[4561904751, 1329651956, 4561904755, 456190474...	16564.502509	16.564503

Finally, let's view our shortest route from Vasco da Gama Pillar to Gedi Ruins overlain on top of a pretty basemap.

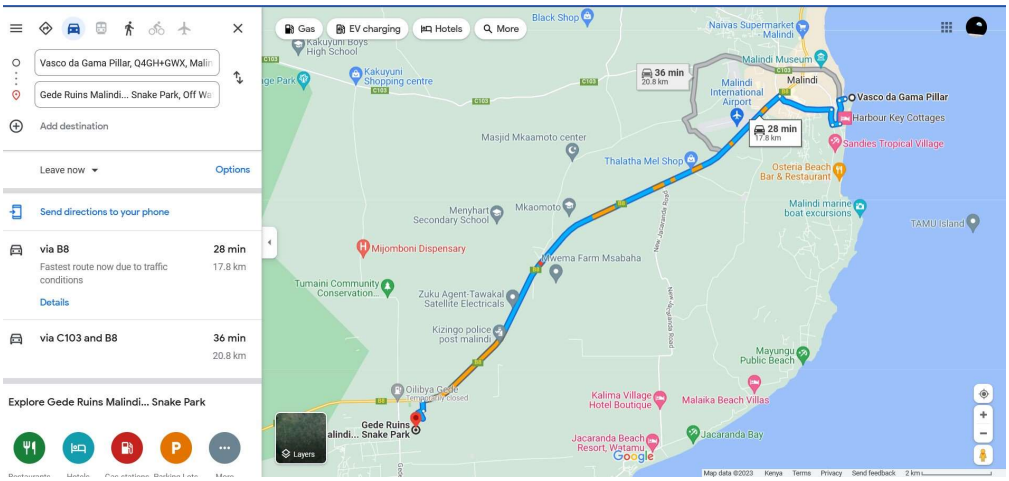
In [37]:
```
import contextily as cx
```

In [38]:
```
# Draw route on top of basemap
import contextily as cx

ax1 = route_geom.plot(linewidth=2,
                      color='red'
                     )

cx.add_basemap(ax=ax1,
               crs=route_geom.crs.to_string(),
               source=cx.providers.OpenStreetMap.France
              )
```

In google maps, the distance between the two points is 17km if using the shortest route, and 20km if using the long route. In fact the shortest distance, as shown below, is similar to our above map. Not so bad an error considering that we had to use special function to get nodes falling within the network graph, rather than our original coordinates which could have been further from the network, and thus add up to 17km if the distance between the two places is taken into account.

In [39]:
```
from IPython import display
display.Image("E:/documents/gis800_articles/jupyter/route_analysis/google_maps.jpg",
              alt="Shortest distance according to Google Maps",
              width=500,
              height=300)
```

Out[39]:



Conclusion

We have seen how to conduct a route analysis in python. The `osmnx` package is useful in deriving path networks, from which route analyses can be done. We have seen the various components that make up an Open StreetMap network, key among them edges and nodes which are vital in creating a LineString from which a systematic shortest route path can be drawn, and the distance calculated.

In [ ]: