Launch Sites Locations Analysis with Folium

Estimated time needed: **40** minutes

The launch success rate may depend on many factors such as payload mass, orbit type, and so on. It may also depend on the location and proximities of a launch site, i.e., the initial position of rocket trajectories. Finding an optimal location for building a launch site certainly involves many factors and hopefully we could discover some of the factors by analyzing the existing launch site locations.

In the previous exploratory data analysis labs, you have visualized the SpaceX launch dataset using `matplotlib` and `seaborn` and discovered some preliminary correlations between the launch site and success rates. In this lab, you will be performing more interactive visual analytics using `Folium`.

Objectives

This lab contains the following tasks:

- **TASK 1:** Mark all launch sites on a map
- **TASK 2:** Mark the success/failed launches for each site on the map
- **TASK 3:** Calculate the distances between a launch site to its proximities

After completed the above tasks, you should be able to find some geographical patterns about launch sites.

Let's first import required Python packages for this lab:

In [1]:
```
!pip3 install folium
!pip3 install wget
```

```
Collecting folium
  Downloading folium-0.17.0-py2.py3-none-any.whl.metadata (3.8 kB)
Collecting branca>=0.6.0 (from folium)
  Using cached branca-0.7.2-py3-none-any.whl.metadata (1.5 kB)
Requirement already satisfied: jinja2>=2.9 in c:\users\91939\anaconda3\nani\lib\s
ite-packages (from folium) (3.1.4)
Requirement already satisfied: numpy in c:\users\91939\anaconda3\nani\lib\site-pa
ckages (from folium) (1.26.4)
Requirement already satisfied: requests in c:\users\91939\anaconda3\nani\lib\site
-packages (from folium) (2.32.2)
Requirement already satisfied: xyzservices in c:\users\91939\anaconda3\nani\lib\s
ite-packages (from folium) (2022.9.0)
Requirement already satisfied: MarkupSafe>=2.0 in c:\users\91939\anaconda3\nani\l
ib\site-packages (from jinja2>=2.9->folium) (2.1.3)
Requirement already satisfied: charset-normalizer<4,>=2 in c:\users\91939\anacond
a3\nani\lib\site-packages (from requests->folium) (2.0.4)
Requirement already satisfied: idna<4,>=2.5 in c:\users\91939\anaconda3\nani\lib
\site-packages (from requests->folium) (3.7)
Requirement already satisfied: urllib3<3,>=1.21.1 in c:\users\91939\anaconda3\nan
i\lib\site-packages (from requests->folium) (2.2.2)
Requirement already satisfied: certifi>=2017.4.17 in c:\users\91939\anaconda3\nan
i\lib\site-packages (from requests->folium) (2024.6.2)
Downloading folium-0.17.0-py2.py3-none-any.whl (108 kB)
   ---------------------------------------- 0.0/108.4 kB ? eta -:--:--
   --- ------------------------------------ 10.2/108.4 kB ? eta -:--:--
   ------------------------------------- -- 102.4/108.4 kB 1.5 MB/s eta 0:00:01
   ---------------------------------------- 108.4/108.4 kB 1.3 MB/s eta 0:00:00
Using cached branca-0.7.2-py3-none-any.whl (25 kB)
Installing collected packages: branca, folium
Successfully installed branca-0.7.2 folium-0.17.0
Collecting wget
  Downloading wget-3.2.zip (10 kB)
  Preparing metadata (setup.py): started
  Preparing metadata (setup.py): finished with status 'done'
Building wheels for collected packages: wget
  Building wheel for wget (setup.py): started
  Building wheel for wget (setup.py): finished with status 'done'
  Created wheel for wget: filename=wget-3.2-py3-none-any.whl size=9680 sha256=51a
76c0cbad58efc9192669bc8f541942f1ba7d6dc7608ec43f8a1b81a7d3dd9
  Stored in directory: c:\users\91939\appdata\local\pip\cache\wheels\01\46\3b\e29
ffbe4ebe614ff224bad40fc6a5773a67a163251585a13a9
Successfully built wget
Installing collected packages: wget
Successfully installed wget-3.2
```

In [2]:
```python
import folium
import wget
import pandas as pd
```

In [3]:
```python
# Import folium MarkerCluster plugin
from folium.plugins import MarkerCluster
# Import folium MousePosition plugin
from folium.plugins import MousePosition
# Import folium DivIcon plugin
from folium.features import DivIcon
```

If you need to refresh your memory about folium, you may download and refer to this previous folium lab:

Generating Maps with Python

Task 1: Mark all launch sites on a map

First, let's try to add each site's location on a map using site's latitude and longitude coordinates

The following dataset with the name `spacex_launch_geo.csv` is an augmented dataset with latitude and longitude added for each site.

In [4]:
```
# Download and read the `spacex_launch_geo.csv`
spacex_csv_file = wget.download('https://cf-courses-data.s3.us.cloud-object-stor
spacex_df=pd.read_csv(spacex_csv_file)
```

Now, you can take a look at what are the coordinates for each site.

In [5]:
```
# Select relevant sub-columns: `Launch Site`, `Lat(Latitude)`, `Long(Longitude)`
spacex_df = spacex_df[['Launch Site', 'Lat', 'Long', 'class']]
launch_sites_df = spacex_df.groupby(['Launch Site'], as_index=False).first()
launch_sites_df = launch_sites_df[['Launch Site', 'Lat', 'Long']]
launch_sites_df
```

Out[5]:

	Launch Site	Lat	Long
0	CCAFS LC-40	28.562302	-80.577356
1	CCAFS SLC-40	28.563197	-80.576820
2	KSC LC-39A	28.573255	-80.646895
3	VAFB SLC-4E	34.632834	-120.610745

Above coordinates are just plain numbers that can not give you any intuitive insights about where are those launch sites. If you are very good at geography, you can interpret those numbers directly in your mind. If not, that's fine too. Let's visualize those locations by pinning them on a map.

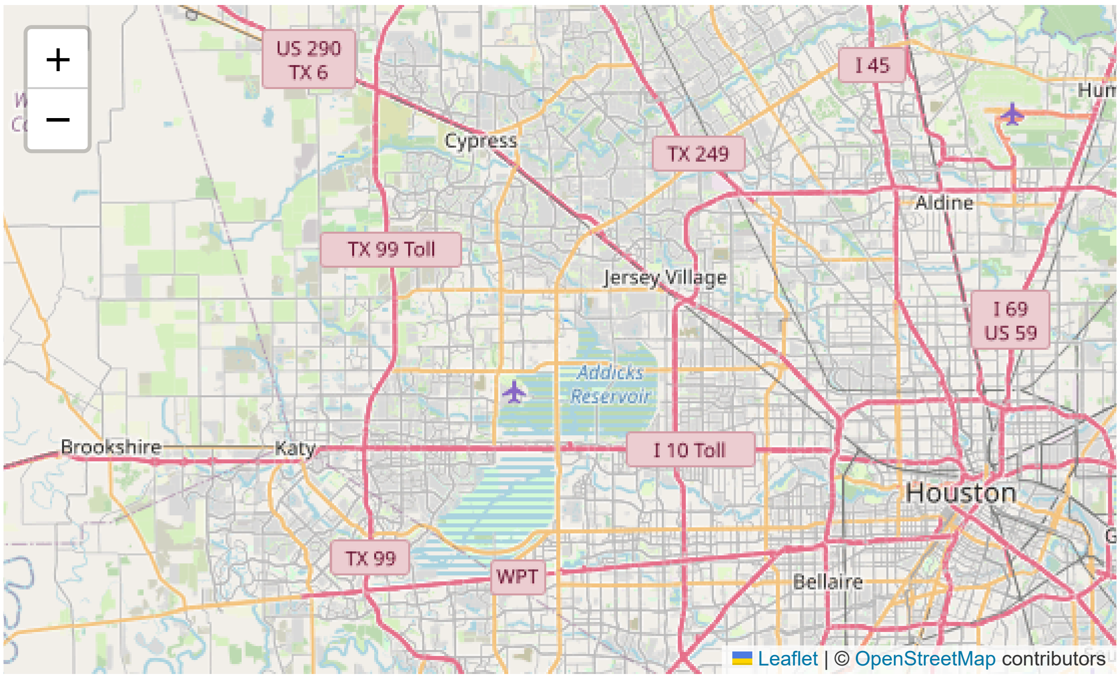
We first need to create a folium `Map` object, with an initial center location to be NASA Johnson Space Center at Houston, Texas.

In [6]:
```
# Start location is NASA Johnson Space Center
nasa_coordinate = [29.559684888503615, -95.0830971930759]
site_map = folium.Map(location=nasa_coordinate, zoom_start=10)
```

We could use `folium.Circle` to add a highlighted circle area with a text label on a specific coordinate. For example,

In [7]:
```
# Create a blue circle at NASA Johnson Space Center's coordinate with a popup la
circle = folium.Circle(nasa_coordinate, radius=1000, color='#d35400', fill=True)
# Create a blue circle at NASA Johnson Space Center's coordinate with a icon sho
marker = folium.map.Marker(
    nasa_coordinate,
    # Create an icon as a text label
    icon=DivIcon(
        icon_size=(20,20),
        icon_anchor=(0,0),
        html='<div style="font-size: 12; color:#d35400;"><b>%s</b></div>' % 'NAS
        )
    )
site_map.add_child(circle)
site_map.add_child(marker)
```

Out[7]:



and you should find a small yellow circle near the city of Houston and you can zoom-in to see a larger circle.

Now, let's add a circle for each launch site in data frame `launch_sites`

TODO: Create and add `folium.Circle` and `folium.Marker` for each launch site on the site map

An example of folium.Circle:

`folium.Circle(coordinate, radius=1000, color='#000000', fill=True).add_child(folium.Popup(...))`

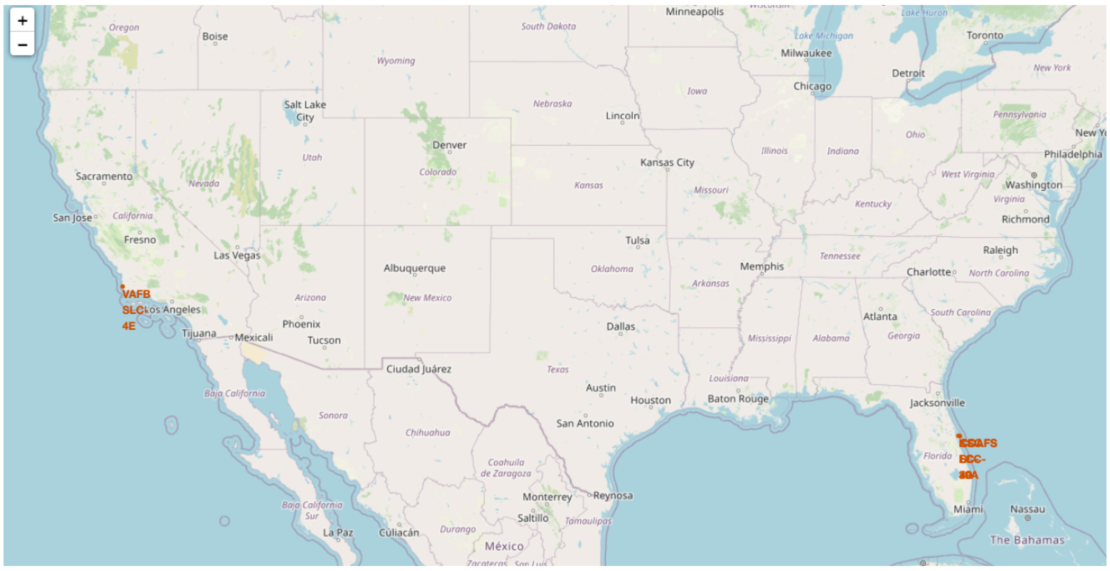
An example of folium.Marker:

`folium.map.Marker(coordinate, icon=DivIcon(icon_size=(20,20),icon_anchor=(0,0), html='<div style="font-size: 12; color:#d35400;"><b>%s</b></div>' % 'label', ))`

In [8]:

```
# Initial the map
site_map = folium.Map(location=nasa_coordinate, zoom_start=5)
# For each launch site, add a Circle object based on its coordinate (Lat, Long)
```

The generated map with marked launch sites should look similar to the following:



Now, you can explore the map by zoom-in/out the marked areas , and try to answer the following questions:

- Are all launch sites in proximity to the Equator line?
- Are all launch sites in very close proximity to the coast?

Also please try to explain your findings.

Task 2: Mark the success/failed launches for each site on the map

Next, let's try to enhance the map by adding the launch outcomes for each site, and see which sites have high success rates. Recall that data frame spacex_df has detailed launch records, and the `class` column indicates if this launch was successful or not

In [9]:
```
spacex_df.tail(10)
```

Out[9]:

	Launch Site	Lat	Long	class
46	KSC LC-39A	28.573255	-80.646895	1
47	KSC LC-39A	28.573255	-80.646895	1
48	KSC LC-39A	28.573255	-80.646895	1
49	CCAFS SLC-40	28.563197	-80.576820	1
50	CCAFS SLC-40	28.563197	-80.576820	1
51	CCAFS SLC-40	28.563197	-80.576820	0
52	CCAFS SLC-40	28.563197	-80.576820	0
53	CCAFS SLC-40	28.563197	-80.576820	0
54	CCAFS SLC-40	28.563197	-80.576820	1
55	CCAFS SLC-40	28.563197	-80.576820	0

Next, let's create markers for all launch records. If a launch was successful `(class=1)`, then we use a green marker and if a launch was failed, we use a red marker `(class=0)`

Note that a launch only happens in one of the four launch sites, which means many launch records will have the exact same coordinate. Marker clusters can be a good way to simplify a map containing many markers having the same coordinate.

Let's first create a `MarkerCluster` object

In [10]:
```
marker_cluster = MarkerCluster()
```

TODO: Create a new column in `launch_sites` dataframe called `marker_color` to store the marker colors based on the `class` value

In [11]:
```
# Apply a function to check the value of `class` column
# If class=1, marker_color value will be green
# If class=0, marker_color value will be red
spacex_df['marker_color'] = list(map(lambda x: 'green' if x==1 else 'red', space
spacex_df.tail(10)

# Function to assign color to launch outcome
def assign_marker_color(launch_outcome):
    if launch_outcome == 1:
        return 'green'
    else:
        return 'red'

spacex_df['marker_color'] = spacex_df['class'].apply(assign_marker_color)
```

In [12]:
```
# Function to assign color to launch outcome
def assign_marker_color(launch_outcome):
    if launch_outcome == 1:
        return 'green'
    else:
        return 'red'
```

```
spacex_df['marker_color'] = spacex_df['class'].apply(assign_marker_color)
spacex_df.tail(10)
```

Out[12]:

	Launch Site	Lat	Long	class	marker_color
46	KSC LC-39A	28.573255	-80.646895	1	green
47	KSC LC-39A	28.573255	-80.646895	1	green
48	KSC LC-39A	28.573255	-80.646895	1	green
49	CCAFS SLC-40	28.563197	-80.576820	1	green
50	CCAFS SLC-40	28.563197	-80.576820	1	green
51	CCAFS SLC-40	28.563197	-80.576820	0	red
52	CCAFS SLC-40	28.563197	-80.576820	0	red
53	CCAFS SLC-40	28.563197	-80.576820	0	red
54	CCAFS SLC-40	28.563197	-80.576820	1	green
55	CCAFS SLC-40	28.563197	-80.576820	0	red

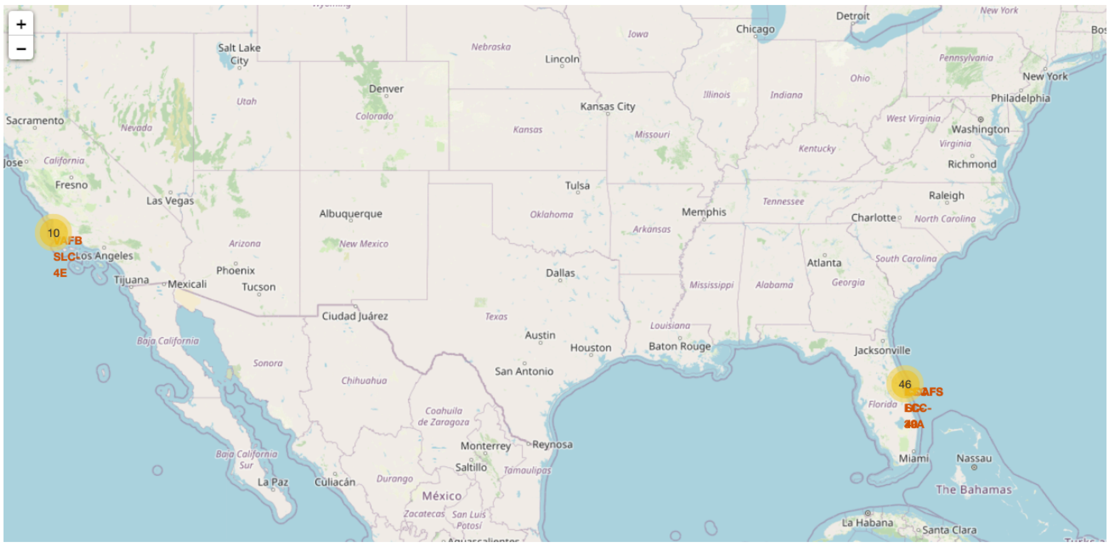
TODO: For each launch result in `spacex_df` data frame, add a `folium.Marker` to `marker_cluster`

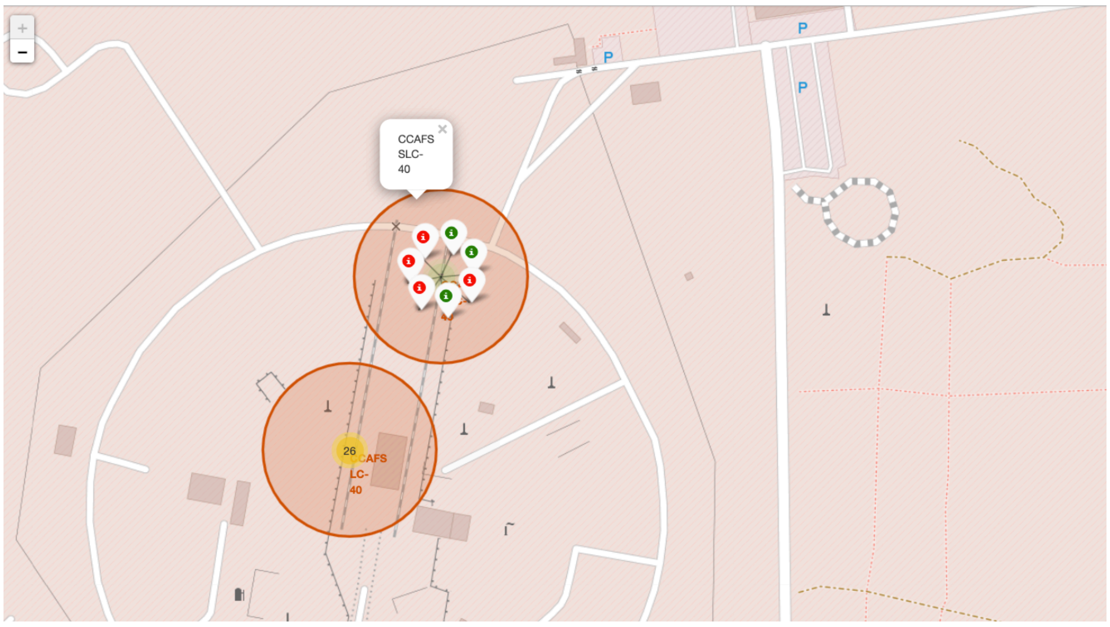
In [13]:

```
# Add marker_cluster to current site_map
site_map.add_child(marker_cluster)

# for each row in spacex_df data frame
# create a Marker object with its coordinate
# and customize the Marker's icon property to indicate if this launch was succes
# e.g., icon=folium.Icon(color='white', icon_color=row['marker_color']
for index, record in spacex_df.iterrows():
    # TODO: Create and add a Marker cluster to the site map
    # marker = folium.Marker(...)
    marker_cluster.add_child(marker)

site_map
```

Out[13]:

Your updated map may look like the following screenshots:





From the color-labeled markers in marker clusters, you should be able to easily identify which launch sites have relatively high success rates.

TASK 3: Calculate the distances between a launch site to its proximities

Next, we need to explore and analyze the proximities of launch sites.

Let's first add a `MousePosition` on the map to get coordinate for a mouse over a point on the map. As such, while you are exploring the map, you can easily find the coordinates of any points of interests (such as railway)

In [14]:
```
# Add Mouse Position to get the coordinate (Lat, Long) for a mouse over on the m
formatter = "function(num) {return L.Util.formatNum(num, 5);};"
mouse_position = MousePosition(
    position='topright',
    separator=' Long: ',
    empty_string='NaN',
    lng_first=False,
    num_digits=20,
    prefix='Lat:',
    lat_formatter=formatter,
    lng_formatter=formatter,
)

site_map.add_child(mouse_position)
site_map
```

Out[14]:

Now zoom in to a launch site and explore its proximity to see if you can easily find any railway, highway, coastline, etc. Move your mouse to these points and mark down their coordinates (shown on the top-left) in order to the distance to the launch site.

You can calculate the distance between two points on the map based on their `Lat` and `Long` values using the following method:

```
In [15]: from math import sin, cos, sqrt, atan2, radians

         def calculate_distance(lat1, lon1, lat2, lon2):
             # approximate radius of earth in km
             R = 6373.0

             lat1 = radians(lat1)
             lon1 = radians(lon1)
             lat2 = radians(lat2)
             lon2 = radians(lon2)

             dlon = lon2 - lon1
             dlat = lat2 - lat1

             a = sin(dlat / 2)**2 + cos(lat1) * cos(lat2) * sin(dlon / 2)**2
             c = 2 * atan2(sqrt(a), sqrt(1 - a))

             distance = R * c
             return distance
```

TODO: Mark down a point on the closest coastline using MousePosition and calculate the distance between the coastline point and the launch site.

```
In [16]: # find coordinate of the closet coastline
         # e.g.,: Lat: 28.56367  Lon: -80.57163
         # distance_coastline = calculate_distance(launch_site_lat, launch_site_lon, coas
         launch_site_lat = 28.563197
         launch_site_lon = -80.576820
         coastline_lat = 28.56334
         coastline_lon = -80.56799
         distance_coastline = calculate_distance(launch_site_lat, launch_site_lon, coastl
         print(distance_coastline,' km')
```

```
0.8627671182499878  km
```

TODO: After obtained its coordinate, create a `folium.Marker` to show the distance

```
In [17]: # Create and add a folium.Marker on your selected closest coastline point on the
         # Display the distance between coastline point and launch site using the icon pr
         # for example
         # distance_marker = folium.Marker(
         #    coordinate,
         #    icon=DivIcon(
         #        icon_size=(20,20),
         #        icon_anchor=(0,0),
         #        html='<div style="font-size: 12; color:#d35400;"><b>%s</b></div>' % "{:
         #        )
         #    )
         distance_marker = folium.Marker(
            [coastline_lat, coastline_lon],
            icon=DivIcon(
                icon_size=(20,20),
                icon_anchor=(0,0),
                html='<div style="font-size: 12; color:#d35400;"><b>%s</b></div>' % "{:10
                )
```

```
    )
site_map.add_child(distance_marker)
```

Out[17]:

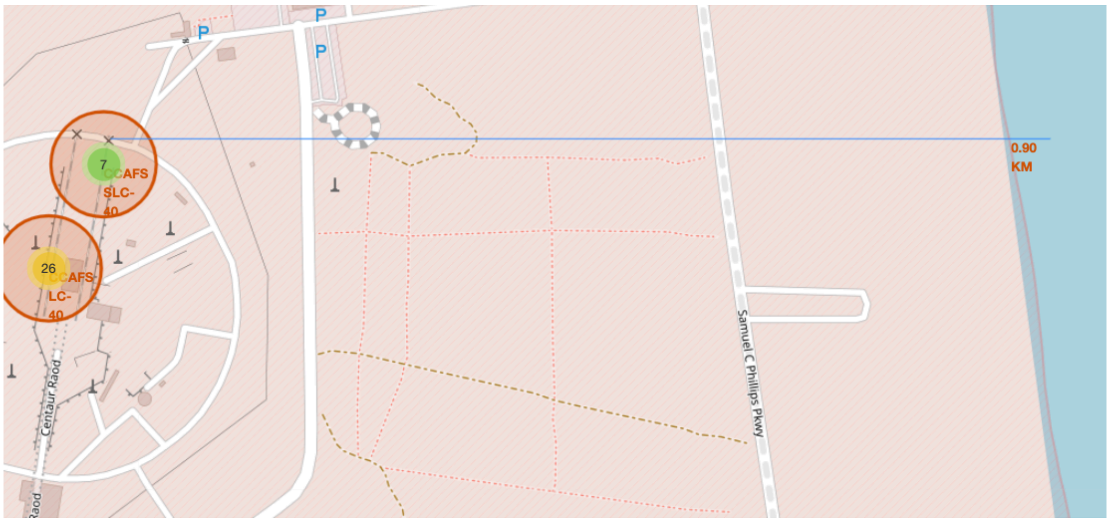
TODO: Draw a `PolyLine` between a launch site to the selected coastline point

In [18]:

```
# Create a `folium.PolyLine` object using the coastline coordinates and launch s
# lines=folium.PolyLine(locations=coordinates, weight=1)
#site_map.add_child(lines)
coordinates = [[launch_site_lat,launch_site_lon],[coastline_lat,coastline_lon]]
lines=folium.PolyLine(locations=coordinates, weight=1)
site_map.add_child(lines)
```

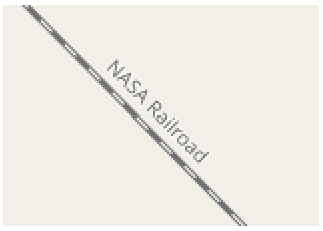
Out[18]:

Your updated map with distance line should look like the following screenshot:



TODO: Similarly, you can draw a line betwee a launch site to its closest city, railway, highway, etc. You need to use `MousePosition` to find the their coordinates on the map first

A railway map symbol may look like this:



A highway map symbol may look like this:

A city map symbol may look like this:

In [19]:
```
# Create a marker with distance to a closest city, railway, highway, etc.
# Draw a line between the marker to the launch site
launch_coordinate = [28.562302, -80.57682]
```

```
city    = [28.61200, -80.80788]
railway = [28.55752, -80.80155]
highway    = [28.5402, -80.85079]

city_distance = calculate_distance(city[0], city[1], launch_coordinate[0], launc
railway_distance = calculate_distance(railway[0], railway[1], launch_coordinate[
highway_distance = calculate_distance(highway[0], highway[1], launch_coordinate[

colors = ['red','orange','green']
html_colors = ['#dc3545','#fd7e14','#198754']

for coordinate ,distance, color, html_color in zip([city, railway, highway], [ci
    marker = folium.map.Marker(
            coordinate,
            # Create an icon as a text label
            icon=DivIcon(
                icon_size=(20,20),
                icon_anchor=(0,0),
                html='+html_color+' '>%s' % str(round(distance, 2)) + 'km',
            )
            )
    marker.add_to(site_map)
    folium.PolyLine([coordinate, launch_coordinate], color=color).add_to(site_ma

site_map.add_child(lines)
```

Out[19]:

In [ ]:

In [ ]:

After you plot distance lines to the proximities, you can answer the following questions easily:

- Are launch sites in close proximity to railways?
- Are launch sites in close proximity to highways?
- Are launch sites in close proximity to coastline?
- Do launch sites keep certain distance away from cities?

Also please try to explain your findings.

In [20]:
```python
print("City Distance", city_distance)

print("Railway Distance", railway_distance)

print("Highway Distance", highway_distance)

print("Coastline Distance", distance_coastline)
```

```
City Distance 23.234752126023245
Railway Distance 21.961465676043673
Highway Distance 26.88038569681492
Coastline Distance 0.8627671182499878
```

Proximity to the Equator:

The closer the launch site to the equator, the easier it is to launch to equatorial orbit, and the more help you get from Earth's rotation for a prograde orbit. Rockets launched from sites near the equator get an additional natural boost - due to the rotational speed of earth - that helps save the cost of putting in extra fuel and boosters.

Safety:

Coasts help ensure that spent stages dropped along the launch path or failed launches don't fall on people or property.

Launch Site Safety/Security:

Needs to be an exclusion zone around the launch site to keep unauthorized people away and keep people safe.

Proximity to Transportation/Infrastructure and Cities:

Launch sites need to be away from anything a failed launch can damage, but still close enough to roads/rails/docks to be able to bring people and material to or from it in support of launch activities.

Next Steps:

Now you have discovered many interesting insights related to the launch sites' location using folium, in a very interactive way. Next, you will need to build a dashboard using Ploty Dash on detailed launch records.

Authors

Yan Luo

Other Contributors

Joseph Santarcangelo

Change Log

Date (YYYY-MM-DD)	Version	Changed By	Change Description
2021-05-26	1.0	Yan	Created the initial version