Skills Network

Hands-on Lab: Interactive Visual Analytics 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 [19]:
```
!pip install folium pandas
```

```
!pip3 install folium
!pip3 install wget

#await piplite.install(['folium'])
#await piplite.install(['pandas'])
```

```
Requirement already satisfied: folium in /opt/conda/lib/python3.11/site-packages (0.18.0)
Requirement already satisfied: pandas in /opt/conda/lib/python3.11/site-packages (2.2.3)
Requirement already satisfied: branca>=0.6.0 in /opt/conda/lib/python3.11/site-packages (from folium) (0.8.
0)
Requirement already satisfied: jinja2>=2.9 in /opt/conda/lib/python3.11/site-packages (from folium) (3.1.3)
Requirement already satisfied: numpy in /opt/conda/lib/python3.11/site-packages (from folium) (2.1.3)
Requirement already satisfied: requests in /opt/conda/lib/python3.11/site-packages (from folium) (2.31.0)
Requirement already satisfied: xyzservices in /opt/conda/lib/python3.11/site-packages (from folium) (2024.
9.0)
Requirement already satisfied: python-dateutil>=2.8.2 in /opt/conda/lib/python3.11/site-packages (from pand
as) (2.9.0)
Requirement already satisfied: pytz>=2020.1 in /opt/conda/lib/python3.11/site-packages (from pandas) (2024.
1)
Requirement already satisfied: tzdata>=2022.7 in /opt/conda/lib/python3.11/site-packages (from pandas) (202
4.2)
Requirement already satisfied: MarkupSafe>=2.0 in /opt/conda/lib/python3.11/site-packages (from jinja2>=2.9
->folium) (2.1.5)
Requirement already satisfied: six>=1.5 in /opt/conda/lib/python3.11/site-packages (from python-dateutil>=
2.8.2->pandas) (1.16.0)
Requirement already satisfied: charset-normalizer<4,>=2 in /opt/conda/lib/python3.11/site-packages (from re
quests->folium) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /opt/conda/lib/python3.11/site-packages (from requests->foli
um) (3.7)
Requirement already satisfied: urllib3<3,>=1.21.1 in /opt/conda/lib/python3.11/site-packages (from requests
->folium) (2.2.1)
Requirement already satisfied: certifi>=2017.4.17 in /opt/conda/lib/python3.11/site-packages (from requests
->folium) (2024.8.30)
Requirement already satisfied: folium in /opt/conda/lib/python3.11/site-packages (0.18.0)
Requirement already satisfied: branca>=0.6.0 in /opt/conda/lib/python3.11/site-packages (from folium) (0.8.
0)
Requirement already satisfied: jinja2>=2.9 in /opt/conda/lib/python3.11/site-packages (from folium) (3.1.3)
Requirement already satisfied: numpy in /opt/conda/lib/python3.11/site-packages (from folium) (2.1.3)
Requirement already satisfied: requests in /opt/conda/lib/python3.11/site-packages (from folium) (2.31.0)
Requirement already satisfied: xyzservices in /opt/conda/lib/python3.11/site-packages (from folium) (2024.
9.0)
Requirement already satisfied: MarkupSafe>=2.0 in /opt/conda/lib/python3.11/site-packages (from jinja2>=2.9
->folium) (2.1.5)
Requirement already satisfied: charset-normalizer<4,>=2 in /opt/conda/lib/python3.11/site-packages (from re
quests->folium) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /opt/conda/lib/python3.11/site-packages (from requests->foli
um) (3.7)
Requirement already satisfied: urllib3<3,>=1.21.1 in /opt/conda/lib/python3.11/site-packages (from requests
```

```
->folium) (2.2.1)
Requirement already satisfied: certifi>=2017.4.17 in /opt/conda/lib/python3.11/site-packages (from requests
->folium) (2024.8.30)
Collecting wget
  Downloading wget-3.2.zip (10 kB)
  Preparing metadata (setup.py) ... done
Building wheels for collected packages: wget
  Building wheel for wget (setup.py) ... done
  Created wheel for wget: filename=wget-3.2-py3-none-any.whl size=9656 sha256=da0889b319bf2568af52b0d111f5c
3f95a1f295ac8f1195944393949960f9858
  Stored in directory: /home/jupyterlab/.cache/pip/wheels/40/b3/0f/a40dbd1c6861731779f62cc4babcb234387e11d6
97df70ee97
Successfully built wget
Installing collected packages: wget
Successfully installed wget-3.2
```

In [ ]:

```

```

In [20]:

```
import folium
import wget
import pandas as pd
!pip install folium==0.8.3
```

```
Collecting folium==0.8.3
  Downloading folium-0.8.3-py2.py3-none-any.whl.metadata (3.0 kB)
Requirement already satisfied: branca>=0.3.0 in /opt/conda/lib/python3.11/site-packages (from folium==0.8.3) (0.8.0)
Requirement already satisfied: jinja2 in /opt/conda/lib/python3.11/site-packages (from folium==0.8.3) (3.1.3)
Requirement already satisfied: numpy in /opt/conda/lib/python3.11/site-packages (from folium==0.8.3) (2.1.3)
Requirement already satisfied: requests in /opt/conda/lib/python3.11/site-packages (from folium==0.8.3) (2.31.0)
Requirement already satisfied: six in /opt/conda/lib/python3.11/site-packages (from folium==0.8.3) (1.16.0)
Requirement already satisfied: MarkupSafe>=2.0 in /opt/conda/lib/python3.11/site-packages (from jinja2->folium==0.8.3) (2.1.5)
Requirement already satisfied: charset-normalizer<4,>=2 in /opt/conda/lib/python3.11/site-packages (from requests->folium==0.8.3) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /opt/conda/lib/python3.11/site-packages (from requests->folium==0.8.3) (3.7)
Requirement already satisfied: urllib3<3,>=1.21.1 in /opt/conda/lib/python3.11/site-packages (from requests->folium==0.8.3) (2.2.1)
Requirement already satisfied: certifi>=2017.4.17 in /opt/conda/lib/python3.11/site-packages (from requests->folium==0.8.3) (2024.8.30)
Downloading folium-0.8.3-py2.py3-none-any.whl (87 kB)
   ---------------------------------------- 87.7/87.7 kB 10.5 MB/s eta 0:00:00
Installing collected packages: folium
  Attempting uninstall: folium
    Found existing installation: folium 0.18.0
    Uninstalling folium-0.18.0:
      Successfully uninstalled folium-0.18.0
Successfully installed folium-0.8.3
```

In [ ]:

In [21]:

```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
```

In [ ]:

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

Generating Maps with Python

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 [ ]:
```
spacex_csv_file = wget.download('https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS(
spacex_df=pd.read_csv(spacex_csv_file)
```

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

In [23]:
```
# Select relevant sub-columns: `Launch Site`, `Lat(Latitude)`, `Long(Longitude)`, `class`
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[23]:

	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

In [ ]:
```

```

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 [24]:
```
# Start location is NASA Johnson Space Center
nasa_coordinate = [29.559684888503615, -95.0830971930759]
site_map = folium.Map(location=nasa_coordinate, zoom_start=10)
```

In [ ]:
```
```

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

In [25]:
```
# Create a blue circle at NASA Johnson Space Center's coordinate with a popup label showing its name
circle = folium.Circle(nasa_coordinate, radius=1000, color='#d35400', fill=True).add_child(folium.Popup('N/
# Create a blue circle at NASA Johnson Space Center's coordinate with a icon showing its name
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>' % 'NASA JSC',
        )
    )
site_map.add_child(circle)
site_map.add_child(marker)
```

Out[25]: Make this Notebook Trusted to load map: File -> Trust Notebook

In [ ]:

```

```

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', ))
```

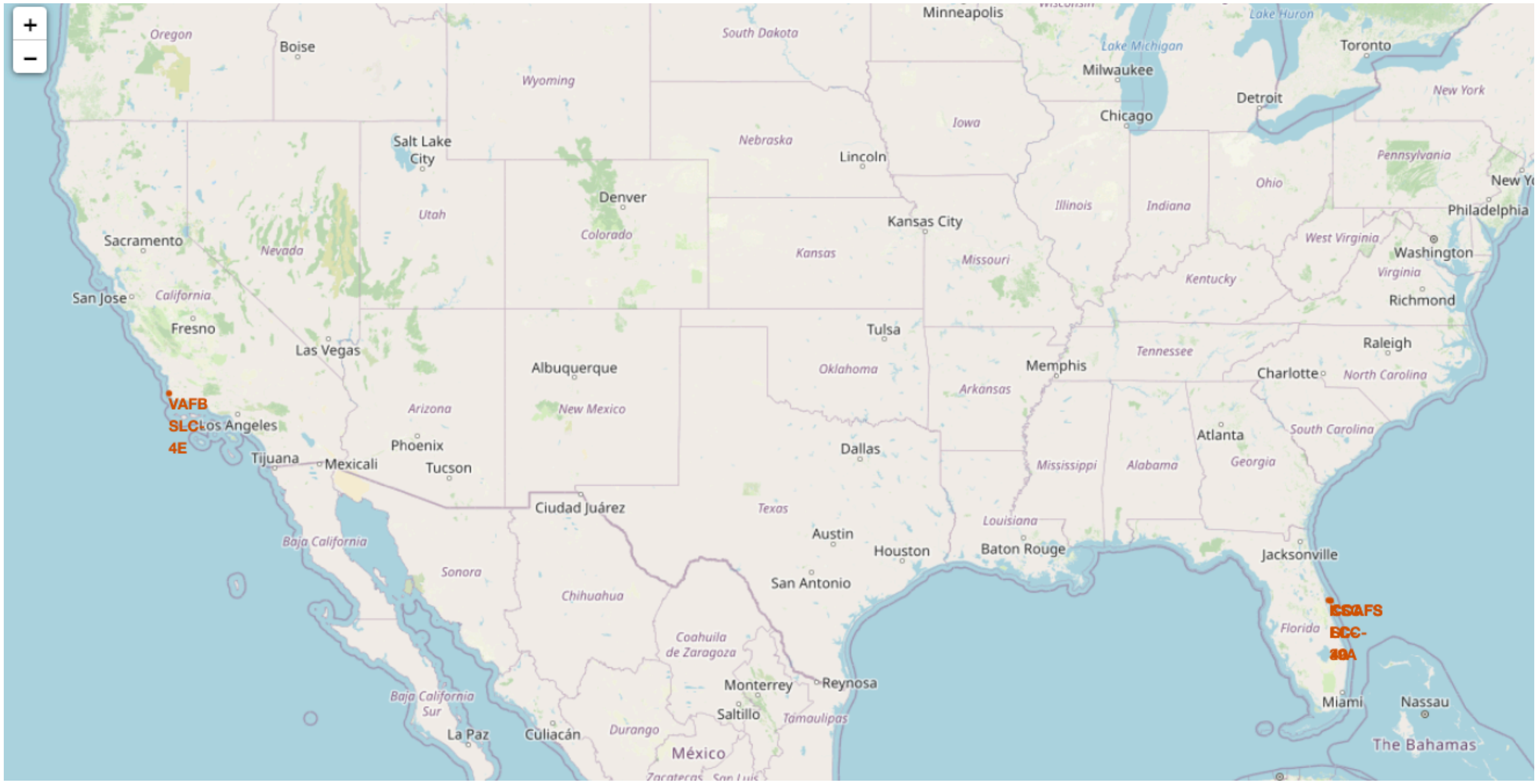
In [26]:
```
# Initialize the map
site_map = folium.Map(location=nasa_coordinate, zoom_start=4)
# For each launch site, add a Circle object based on its coordinate (Lat, Long) values. In addition, add La
for index, row in launch_sites_df.iterrows():
    coordinate = [row['Lat'], row['Long']]
    folium.Circle(coordinate, radius=1000, color='#000000', fill=True).add_child(folium.Popup(row['Launch S
    folium.map.Marker(coordinate, icon=DivIcon(icon_size=(20,20),icon_anchor=(0,0), html='<div style="font-
site_map
```

Out[26]: Make this Notebook Trusted to load map: File -> Trust Notebook

In [ ]:

```

```

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.

Yes launch sites are in proximity to the equator and the coast. This makes sense as it takes less fuel to get into space from the equator due to the physics of Earth's rotation. The launch sites in close proximity to the coast are also logical for safety reasons.

In [29]:
```
# 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 [31]:
```
spacex_df
```

Out[31]:

	Launch Site	Lat	Long	class
0	CCAFS LC-40	28.562302	-80.577356	0
1	CCAFS LC-40	28.562302	-80.577356	0
2	CCAFS LC-40	28.562302	-80.577356	0
3	CCAFS LC-40	28.562302	-80.577356	0
4	CCAFS LC-40	28.562302	-80.577356	0
5	CCAFS LC-40	28.562302	-80.577356	0
6	CCAFS LC-40	28.562302	-80.577356	0
7	CCAFS LC-40	28.562302	-80.577356	0
8	CCAFS LC-40	28.562302	-80.577356	0
9	CCAFS LC-40	28.562302	-80.577356	0
10	CCAFS LC-40	28.562302	-80.577356	0
11	CCAFS LC-40	28.562302	-80.577356	0
12	CCAFS LC-40	28.562302	-80.577356	0
13	CCAFS LC-40	28.562302	-80.577356	0
14	CCAFS LC-40	28.562302	-80.577356	0
15	CCAFS LC-40	28.562302	-80.577356	0
16	CCAFS LC-40	28.562302	-80.577356	0
17	CCAFS LC-40	28.562302	-80.577356	1
18	CCAFS LC-40	28.562302	-80.577356	1
19	CCAFS LC-40	28.562302	-80.577356	0
20	CCAFS LC-40	28.562302	-80.577356	1
21	CCAFS LC-40	28.562302	-80.577356	1
22	CCAFS LC-40	28.562302	-80.577356	1

	Launch Site	Lat	Long	class
23	CCAFS LC-40	28.562302	-80.577356	0
24	CCAFS LC-40	28.562302	-80.577356	1
25	CCAFS LC-40	28.562302	-80.577356	1
26	VAFB SLC-4E	34.632834	-120.610745	0
27	VAFB SLC-4E	34.632834	-120.610745	0
28	VAFB SLC-4E	34.632834	-120.610745	1
29	VAFB SLC-4E	34.632834	-120.610745	1
30	VAFB SLC-4E	34.632834	-120.610745	1
31	VAFB SLC-4E	34.632834	-120.610745	1
32	VAFB SLC-4E	34.632834	-120.610745	0
33	VAFB SLC-4E	34.632834	-120.610745	0
34	VAFB SLC-4E	34.632834	-120.610745	0
35	VAFB SLC-4E	34.632834	-120.610745	0
36	KSC LC-39A	28.573255	-80.646895	1
37	KSC LC-39A	28.573255	-80.646895	0
38	KSC LC-39A	28.573255	-80.646895	1
39	KSC LC-39A	28.573255	-80.646895	1
40	KSC LC-39A	28.573255	-80.646895	0
41	KSC LC-39A	28.573255	-80.646895	1
42	KSC LC-39A	28.573255	-80.646895	1
43	KSC LC-39A	28.573255	-80.646895	0
44	KSC LC-39A	28.573255	-80.646895	1
45	KSC LC-39A	28.573255	-80.646895	1

	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 [32]:
```
marker_cluster = MarkerCluster()
```

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

In [33]:
```
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[33]:

	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

In [ ]:

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

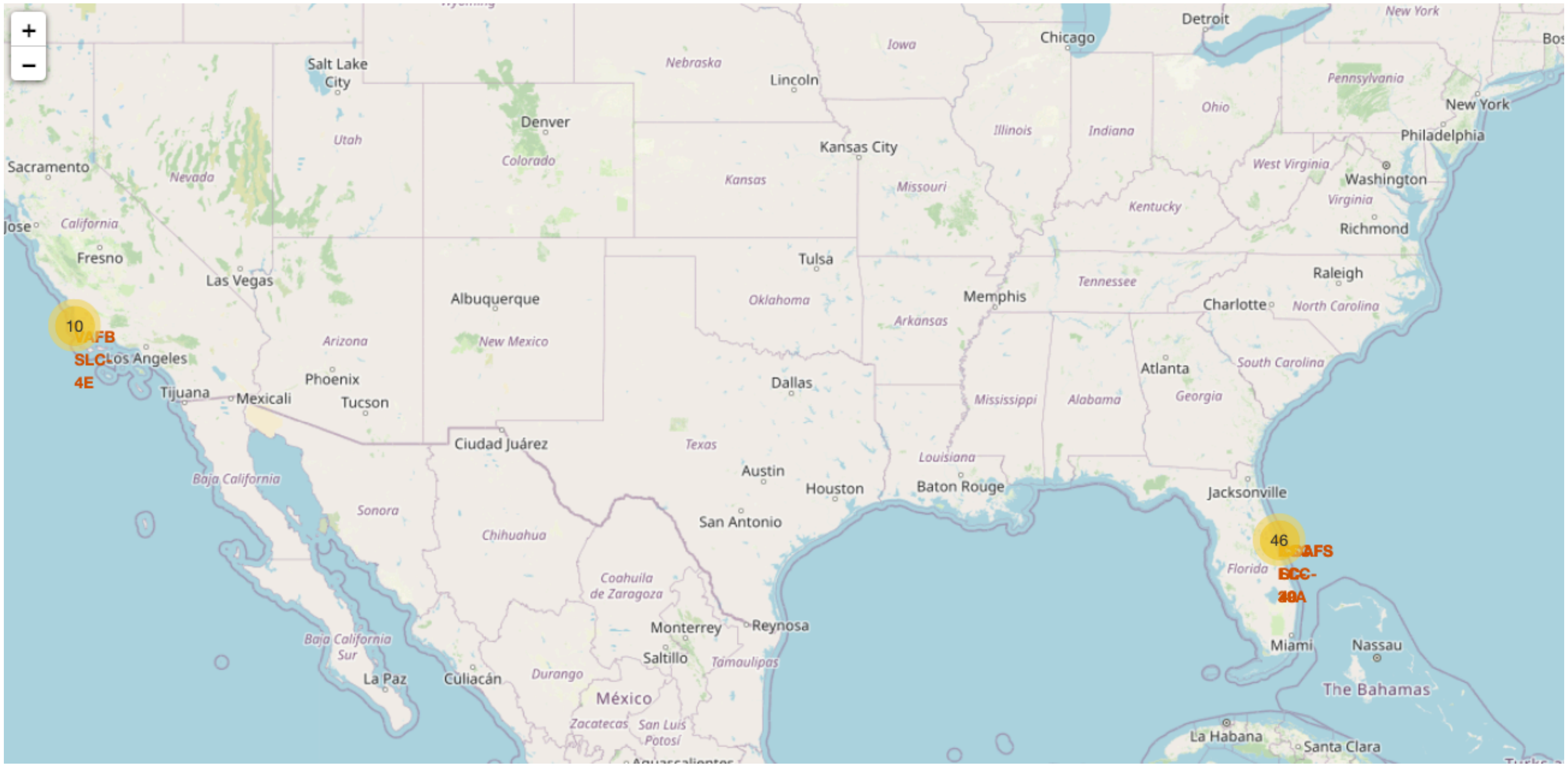
In [34]:

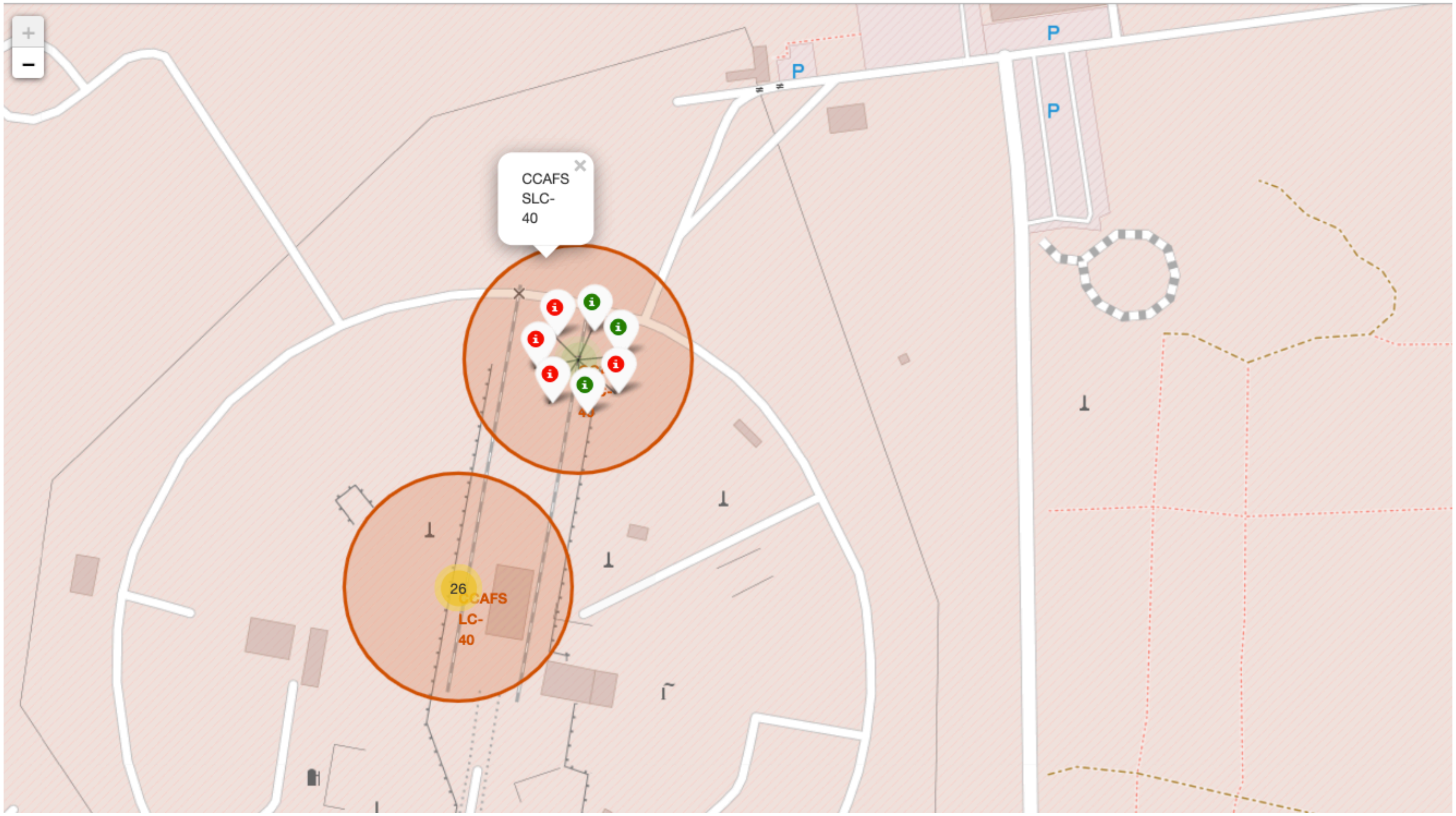
```
# Add marker_cluster to current site_map
site_map.add_child(marker_cluster)

# 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 successed or failed,
# e.g., icon=folium.Icon(color='white', icon_color=row['marker_color']]
for index, row in spacex_df.iterrows():
    # create and add a Marker cluster to the site map
    coordinate = [row['Lat'], row['Long']]
    folium.map.Marker(coordinate, icon=folium.Icon(color='white',icon_color=row['marker_color'])).add_to(ma
site_map
```

Out[34]: Make this Notebook Trusted to load map: File -> Trust Notebook

In [ ]:

Your updated map may look like the following screenshots:


10
46
Salt Lake City
Denver
Nebraska
Lincoln
Iowa
Chicago
Detroit
New York
Pennsylvania
Ohio
Illinois
Indiana
Utah
Colorado
Kansas City
Kansas
Missouri
Philadelphia
West Virginia
Washington
Virginia
Kentucky
Richmond
Sacramento
Nevada
California
Fresno
Las Vegas
Tulsa
Oklahoma
Memphis
Tennessee
Raleigh
Charlotte
North Carolina
Albuquerque
Arizona
New Mexico
Arkansas
Los Angeles
Phoenix
Tijuana
Mexicali
Tucson
Dallas
Atlanta
South Carolina
Mississippi
Alabama
Georgia
Ciudad Juárez
Texas
Louisiana
Austin
Houston
Baton Rouge
Jacksonville
Baja California
Sonora
San Antonio
Chihuahua
Florida
Coahuila de Zaragoza
Monterrey
Reynosa
Baja California Sur
Miami
Nassau
La Paz
Culiacán
Durango
Saltillo
Tamaulipas
The Bahamas
México
Zacatecas
San Luis Potosí
La Habana
Santa Clara



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

In [35]:
```
# Add Mouse Position to get the coordinate (Lat, Long) for a mouse over on the map
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[35]: Make this Notebook Trusted to load map: File -> Trust Notebook

In [ ]:

```

```

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 [36]:
```
# Add Mouse Position to get the coordinate (Lat, Long) for a mouse over on the map
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[36]: Make this Notebook Trusted to load map: File -> Trust Notebook

In [ ]:

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.

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.

In [38]:
```
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
```

In [ ]:
```
```

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

In [41]:
```
# find coordinate of the closet coastline
# e.g.,: Lat: 28.56367  Lon: -80.57163
# distance_coastline = calculate_distance(launch_site_lat, launch_site_lon, coastline_lat, coastline_lon)
launch_site_lat = 28.5631975
launch_site_lon = -80.576820
coastline_lat = 21.56328
coastline_lon = -79.13276
distance_coastline = calculate_distance(launch_site_lat, launch_site_lon, coastline_lat, coastline_lon)
print(distance_coastline,' mi')
```

```
792.0501176057842  mi
```

In [42]:
```
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>' % "{:10.2f} KM".format(distance_c
        )
    )
site_map.add_child(distance_marker)
```

Out[42]: Make this Notebook Trusted to load map: File -> Trust Notebook

In [ ]:

```

```

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

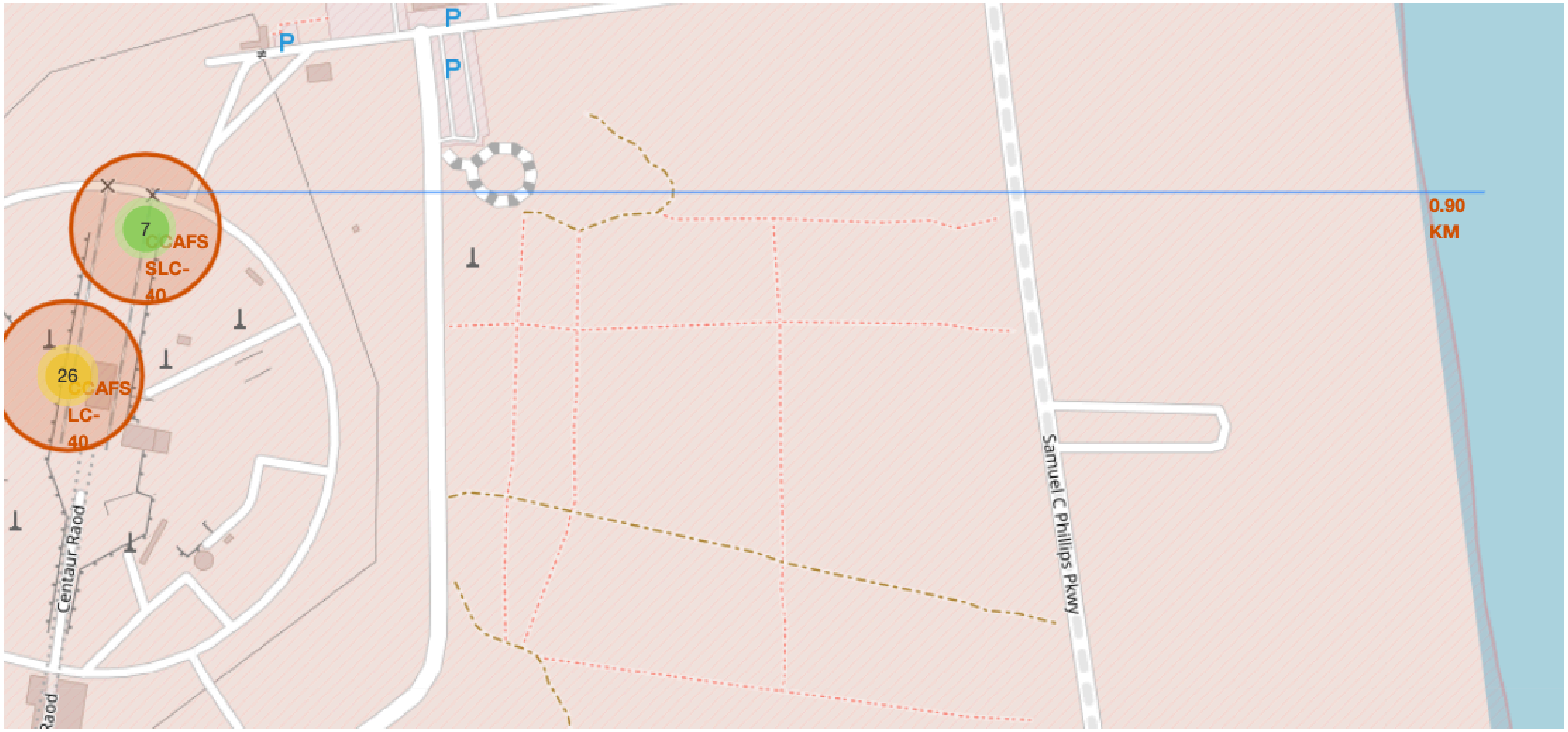
In [43]:
```
coordinates = [[launch_site_lat,launch_site_lon],[coastline_lat,coastline_lon]]
lines=folium.PolyLine(locations=coordinates, weight=1)
site_map.add_child(lines)
```

Out[43]: Make this Notebook Trusted to load map: File -> Trust Notebook

In [ ]:

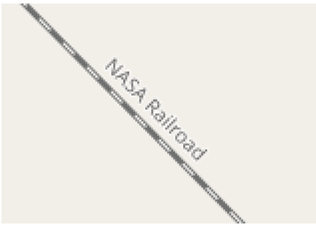
In [ ]:

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

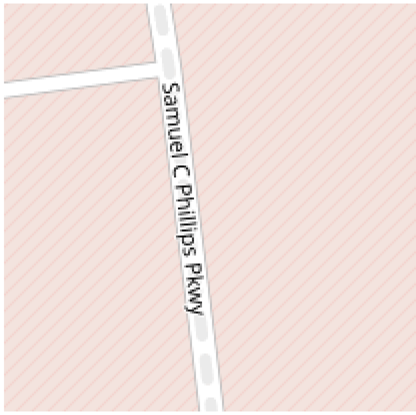


TODO: Similarly, you can draw a line betwee a launch site to its closest city, railway, highway, etc. You need to use `MousePosition` to find the their coordinates on the map first

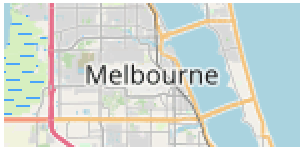
A railway map symbol may look like this:



A highway map symbol may look like this:



A city map symbol may look like this:



In [47]:
```
# Create a marker with distance to a closest city, railway, highway, etc.
# Draw a line between the marker to the launch site
closest_highway = 28.4230, -80.42905
closest_railroad = 28.4399, -80.30266
closest_city = 28.64280, -80.69651
```

In [ ]:
```

```

In [48]:
```
distance_highway = calculate_distance(launch_site_lat, launch_site_lon, closest_highway[0], closest_highwa
print('distance_highway =',distance_highway, ' mi')
distance_railroad = calculate_distance(launch_site_lat, launch_site_lon, closest_railroad[0], closest_rail
print('distance_railroad =',distance_railroad, ' mi')
distance_city = calculate_distance(launch_site_lat, launch_site_lon, closest_city[0], closest_city[1])
print('distance_city =',distance_city, ' mi')
```

```
distance_highway = 21.25679661713841  mi
distance_railroad = 30.104246776452985  mi
distance_city = 14.663342942873387  mi
```

In [ ]:

In [49]:
```
# closest highway marker
distance_marker = folium.Marker(
   closest_highway,
   icon=DivIcon(
       icon_size=(20,20),
       icon_anchor=(0,0),
       html='<div style="font-size: 12; color:#d35400;"><b>%s</b></div>' % "{:10.2f} KM".format(distance_hi
       )
   )
site_map.add_child(distance_marker)
# closest highway line
coordinates = [[launch_site_lat,launch_site_lon],closest_highway]
lines=folium.PolyLine(locations=coordinates, weight=1)
site_map.add_child(lines)

# closest railroad marker
distance_marker = folium.Marker(
   closest_railroad,
   icon=DivIcon(
       icon_size=(20,20),
       icon_anchor=(0,0),
       html='<div style="font-size: 12; color:#d35400;"><b>%s</b></div>' % "{:10.2f} KM".format(distance_ra
       )
   )
site_map.add_child(distance_marker)
# closest railroad line
coordinates = [[launch_site_lat,launch_site_lon],closest_railroad]
lines=folium.PolyLine(locations=coordinates, weight=1)
site_map.add_child(lines)

# closest city marker
distance_marker = folium.Marker(
   closest_city,
   icon=DivIcon(
       icon_size=(20,20),
       icon_anchor=(0,0),
       html='<div style="font-size: 12; color:#d35400;"><b>%s</b></div>' % "{:10.2f} KM".format(distance_ci
       )
   )
```

```
site_map.add_child(distance_marker)
# closest city line
coordinates = [[launch_site_lat,launch_site_lon],closest_city]
lines=folium.PolyLine(locations=coordinates, weight=1)
site_map.add_child(lines)
```

Out[49]: Make this Notebook Trusted to load map: File -> Trust Notebook

In [ ]:

```

```

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

- Are launch sites in close proximity to railways?

- Yes
- Are launch sites in close proximity to highways?
- Yes
- Are launch sites in close proximity to coastline?
- Yes
- Do launch sites keep certain distance away from cities?
- Yes, in my example about 15 miles

Also please try to explain your findings. As mentioned before, launch sites are in close proximity to equator to minimize fuel consumption by using Earth's ~ 30km/sec eastward spin to help spaceships get into orbit. Launch sites are in close proximity to coastline so they can fly over the ocean during launch, for at least two safety reasons-- (1) crew has option to abort launch and attempt water landing (2) minimize people and property at risk from falling debris. Launch sites are in close proximity to highways, which allows for easily transport required people and property. Launch sites are in close proximity to railways, which allows transport for heavy cargo. Launch sites are not in close proximity to cities, which minimizes danger to population dense areas.

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

Pratiksha Verma

<!--## Change Log--!>

<!--| Date (YYYY-MM-DD) | Version | Changed By | Change Description | | ----------------- | ------- | ------------- | ----------------------- | | 2022-11-09 | 1.0 | Pratiksha Verma | Converted initial version to Jupyterlite|--!>