



Falcon 9 Rocket Launcher First Stage Landing Success Prediction Modelling

Lateef Muritala

September 2023

OUTLINE



- Executive Summary
- Introduction
- Methodology
- Conclusion

EXECUTIVE SUMMARY



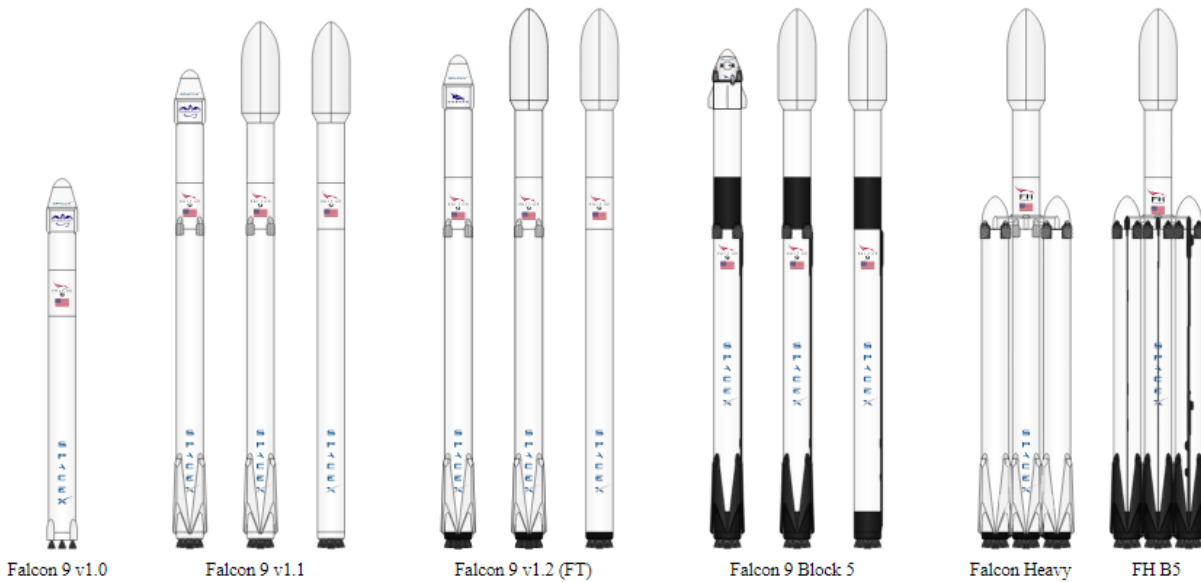
- The objective of the project is to determine if the first stage of Falcon 9 rocket launcher will land successfully for a reuse in other missions
- Successful landing of the first stage rocket launcher was found to be dependent on:
 - Orbit – farther orbits have more success rates
 - PayloadMass (Kg) – the heavier payload the more the success
 - Launch Site – more success were recorded in sites KSC LC-39A and VAFB SLC 4E. There is over 70% chance of a successful landing if the launch is conducted in sites KSC LC-39A and VAFB SLC 4E. The chance for success is further enhanced when either B5 or FT series booster version is deployed.
 - The booster version – The FT series booster version has higher success rate than other booster versions
- The predictive analysis model constructed using historical data suggested that the probability of successful landing for Falcon 9 is greater than 85%.

INTRODUCTION



- This capstone project aimed to predict if the Falcon 9 first stage will land successfully.
- The outcome is expected to guide a decision on whether or not to use SpaceX Falcon 9 rocket launches advertised to cost 62 million dollars or that of the other providers that cost upward of 165 million dollars each
- Rockets from the Falcon 9 family have been launched 264 times over 13 years, resulting in 262 full mission successes (99.2%), one partial success, and one full failure (the SpaceX CRS-7 spacecraft was lost in flight in an explosion). Additionally, one rocket and its payload AMOS-6 were destroyed before launch in preparation for an on-pad static fire test. The active version, Falcon 9 Block 5, has flown 201 missions, all full successes.
- In 2022 Falcon 9 set a new record of 60 launches (all successful) by the same launch vehicle type in a calendar year. The previous record was held by Soyuz-U, which had 47 launches (45 successful) in 1979.
- The first rocket version Falcon 9 v1.0 was launched five times from June 2010 to March 2013, its successor Falcon 9 v1.1 15 times from September 2013 to January 2016, and the Falcon 9 Full Thrust 237 times from December 2015 to present. The latest Full Thrust variant, Block 5, was introduced in May 2018.[8] While the Block 4 boosters were only flown twice and required several months of refurbishment, Block 5 versions are designed to sustain 10 flights with just some inspections.
- The Falcon Heavy derivative consists of a strengthened Falcon 9 first stage as its center core, with two additional Falcon 9 first stages attached and used as boosters, both of which are fitted with an aerodynamic nosecone instead of a usual Falcon 9 interstage.[10]
- Falcon 9 first-stage boosters landed successfully in 229 of 240 attempts (95.4%), with 199 out of 204 (97.5%) for the Falcon 9 Block 5 version. A total of 204 re-flights of first stage boosters have all successfully launched their payloads
- The historical information on the past Falcon 9 launches is available publicly on: https://en.wikipedia.org/wiki/List_of_Falcon_9_and_Falcon_Heavy_launches

METHODOLOGY





- Data collection
- Data preparation
- Data analysis
- Data visualization
 - Cross-plot visualizations
 - Folium Plots
 - Dash plots
- Predictive modelling/analysis and evaluation



Data Collection & Preparation

- The data for this study was sourced from SpaceX API and SpaceX Wiki page using a 'get' method
- The responses from the data sources were normalized/formatted into a dataframe
- Further requests were made to the API for additional information on the booster name, the target orbit, coordinates of the launching pads, the outcome of the landing success or otherwise, etc.
- Data normalization were carried out on the missing data or NaN values

Exploratory Data Analysis

- Exploratory and visual analyses of the data were conducted to gain insight into the data and identify key variables using
 - Structured Query Language (SQL)
 - Cross-plots
- The SQL enabled the data to be viewed or analyzed using a combination of query and grouping parameters 
- With the cross-plots, the relationship between launch data and parameters were explored graphically to determine their influence on a launch success 
- The outcomes from the exercises guided the selection of data for predictive analysis models

Interactive Visual Analysis

- Visual analysis of the data was also conducted on the map (Folium) and using dash plots to assess the success rates at the launch sites
- Using the coordinates of the launching pads, the launch outcomes were plotted on the map. This allows us to determine launch site success rates and the distances of the launch sites from the public infrastructures 
- The dash plots fitted with a dropdown and range slider (Plotly) were helpful in the dynamic determination of launch sites success rates without the need to regenerate the plot. The plots were automatically repainted either when the values of the dropdown changes or that of the range slider changed 

Predictive Modelling and Analysis

- The historical data were normalized and partitioned into train dataset and test dataset – 80% of the available data were randomly selected for model training purpose and the remaining 20% were used as test data
- Model training was done using four predictive algorithms - *logistic regression, support vector machine, decision tree classifier* and *k nearest neighbors*.
- The best scores obtained from the training dataset for all the models were in the range of 84.6% to 87.7%. The decision tree classifier has the highest best score.
- Three models (logistic regression, support vector machine and k nearest neighbors) falsely predicted successful landing for three cases of failed landing. The decision tree classifier on the other hand predicted more true positive and true negatives than other models but had one false positive and one false negatives on the test data.
- Just like the train dataset, the decision tree classifier had the highest score of 88.9% for the test data.

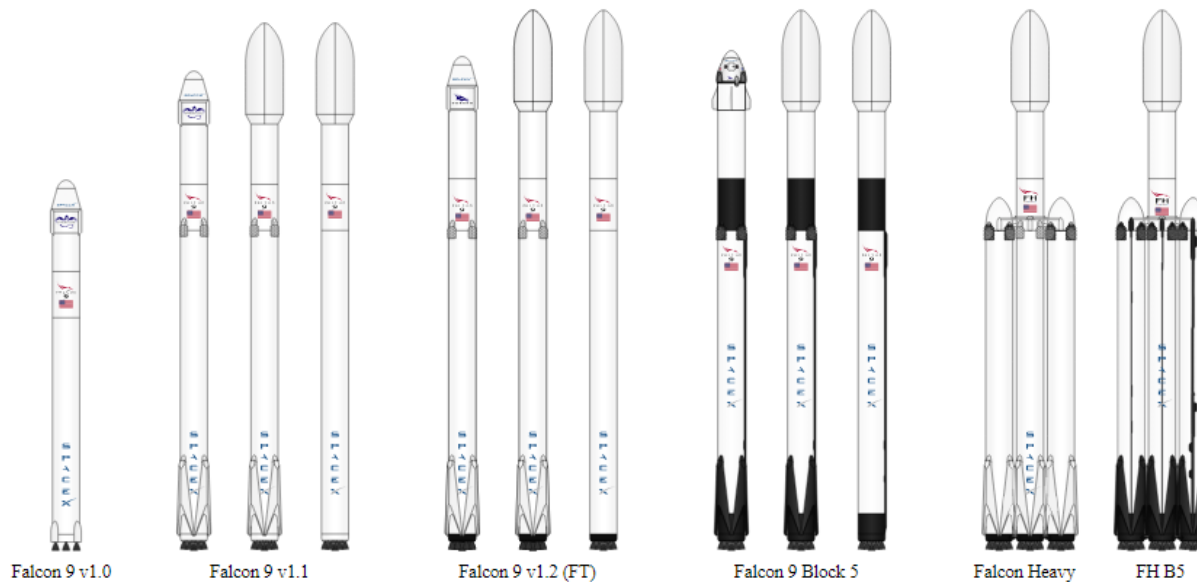
CONCLUSION



- A model to accurately predict the landing outcome of the first stage of Falcon 9 rocket launchers has been developed. The model accuracy is greater than 80% in all the four models used
- The success rate at the KSC LC-39A launch site is greater than 75% and should be considered for the next launch. Similarly, the FT and B5 booster versions are very prolific in terms of success rate
- The heavy payload rockets are more likely to succeed and the first stage launcher available for reuse many times than the lighter ones. However, the recent data showed a positive trend in success rates irrespective of the payload size
- Given the success history of the Falcon 9 rocket launcher series as well as the cost differential between it and its competitors, the chance of a successful landing is higher and it is recommended for deployment in the rocket next launch.

End

EDA with SQL outcomes



Task 1

Display the names of the unique launch sites in the space mission

```
[8]: %sql select distinct * from SPACEXTABLE
* sqlite:///my_data1.db
Done.
```

Date	Time (UTC)	Booster_Version	Launch_Site	Payload	PAYLOAD_MASS_KG_	Orbit	Customer	Mission_Outcome	Landing_Outcome
2010-04-06	18:45:00	F9 v1.0 B0003	CCAFS LC-40	Dragon Spacecraft Qualification Unit	0	LEO	SpaceX	Success	Failure (parachute)
2010-08-12	15:43:00	F9 v1.0 B0004	CCAFS LC-40	Dragon demo flight C1, two CubeSats, barrel of Brouere cheese	0	LEO (ISS)	NASA (COTS) NRO	Success	Failure (parachute)
2012-05-22	07:44:00	F9 v1.0 B0005	CCAFS LC-40	Dragon demo flight C2	525	LEO (ISS)	NASA (COTS)	Success	No attempt
2012-08-10	00:35:00	F9 v1.0 B0006	CCAFS LC-40	SpaceX CRS-1	500	LEO (ISS)	NASA (CRS)	Success	No attempt
2013-01-03	15:10:00	F9 v1.0 B0007	CCAFS LC-40	SpaceX CRS-2	677	LEO (ISS)	NASA (CRS)	Success	No attempt
2013-09-29	16:00:00	F9 v1.1 B1003	VAFB SLC-4E	CASSIOPE	500	Polar LEO	MDA	Success	Uncontrolled (ocean)
2013-03-12	22:41:00	F9 v1.1	CCAFS LC-40	SES-8	3170	GTO	SES	Success	No attempt
2014-06-01	22:06:00	F9 v1.1	CCAFS LC-40	Thaicom 6	3325	GTO	Thaicom	Success	No attempt
2014-04-18	19:25:00	F9 v1.1	CCAFS LC-40	SpaceX CRS-3	2296	LEO (ISS)	NASA (CRS)	Success	Controlled (ocean)
2014-07-14	15:15:00	F9 v1.1	CCAFS LC-40	OG2 Mission 1 6 Orbcomm-OG2 satellites	1316	LEO	Orbcomm	Success	Controlled (ocean)
2014-05-08	08:00:00	F9 v1.1	CCAFS LC-40	AsiaSat 8	4535	GTO	AsiaSat	Success	No attempt
2014-07-09	05:00:00	F9 v1.1 B1011	CCAFS LC-40	AsiaSat 6	4428	GTO	AsiaSat	Success	No attempt
2014-09-21	05:52:00	F9 v1.1 B1010	CCAFS LC-40	SpaceX CRS-4	2216	LEO (ISS)	NASA (CRS)	Success	Uncontrolled (ocean)
2015-10-01	09:47:00	F9 v1.1 B1012	CCAFS LC-40	SpaceX CRS-5	2395	LEO (ISS)	NASA (CRS)	Success	Failure (drone ship)

Task 2

Display 5 records where launch sites begin with the string 'KSC'

```
[10]: %sql select * from SPACEXTABLE where Launch_Site like "KSC%" limit 5
* sqlite:///my_data1.db
Done.
```

Date	Time (UTC)	Booster_Version	Launch_Site	Payload	PAYLOAD_MASS_KG_	Orbit	Customer	Mission_Outcome	Landing_Outcome
2017-02-19	14:39:00	F9 FT B1031.1	KSC LC-39A	SpaceX CRS-10	2490	LEO (ISS)	NASA (CRS)	Success	Success (ground pad)
2017-03-16	06:00:00	F9 FT B1030	KSC LC-39A	EchoStar 23	5600	GTO	EchoStar	Success	No attempt
2017-03-30	22:27:00	F9 FT B1021.2	KSC LC-39A	SES-10	5300	GTO	SES	Success	Success (drone ship)
2017-01-05	11:15:00	F9 FT B1032.1	KSC LC-39A	NROL-76	5300	LEO	NRO	Success	Success (ground pad)
2017-05-15	23:21:00	F9 FT B1034	KSC LC-39A	Inmarsat-5 F4	6070	GTO	Inmarsat	Success	No attempt



Task 3

Display the total payload mass carried by boosters launched by NASA (CRS)

```
[11]: %sql select Customer, sum(PAYLOAD_MASS_KG_) from SPACEXTABLE where Customer like "%NASA (CRS)%"
* sqlite:///my_data1.db
Done.
```

```
[11]: Customer sum(PAYLOAD_MASS_KG_)
-----
NASA (CRS)          48213
```

Task 4

Display average payload mass carried by booster version F9 v1.1

```
[12]: %sql select Booster_version, avg(PAYLOAD_MASS_KG_) from SPACEXTABLE where Booster_Version = "F9 v1.1"
* sqlite:///my_data1.db
Done.
```

```
[12]: Booster_Version avg(PAYLOAD_MASS_KG_)
-----
F9 v1.1          2928.4
```

Task 5

List the date where the succesful landing outcome in drone ship was acheived.

Hint: Use min function

```
[13]: %sql select min(Date) from SPACEXTABLE where Landing_Outcome = "Success (drone ship)"
* sqlite:///my_data1.db
Done.
```

```
[13]: min(Date)
-----
2016-05-27
```

Task 6

List the names of the boosters which have success in ground pad and have payload mass greater than 4000 but less than 6000

```
[14]: %sql select Booster_version from SPACEXTABLE where Landing_Outcome = "Success (ground pad)" and PAYLOAD_MASS_KG_ BETWEEN 4000 and 6000
```

```
* sqlite:///my_data1.db
```

Done.

```
[14]: Booster_Version
```

```
F9 FT B1032.1
```

```
F9 B4 B1040.1
```

```
F9 B4 B1043.1
```

Task 7

List the total number of successful and failure mission outcomes

```
[15]: %sql select Mission_Outcome, count(Mission_Outcome) from SPACEXTABLE where Mission_Outcome like "%Success%" or Mission_Outcome like "%Failure%" group by Mission_Outcome
```

```
* sqlite:///my_data1.db
```

Done.

```
[15]:
```

Mission_Outcome	count(Mission_Outcome)
Failure (in flight)	1
Success	98
Success	1
Success (payload status unclear)	1

Task 8

List the names of the booster_versions which have carried the maximum payload mass. Use a subquery

```
[18]: %sql select Booster_Version, PAYLOAD_MASS_KG_ from SPACEXTABLE where (PAYLOAD_MASS_KG_ = (select max(PAYLOAD_MASS_KG_) from SPACEXTABLE))
```

```
* sqlite:///my_data1.db
```

Done.

```
[18]:
```

Booster_Version	PAYLOAD_MASS_KG_
F9 B5 B1048.4	15600
F9 B5 B1049.4	15600
F9 B5 B1051.3	15600
F9 B5 B1056.4	15600
F9 B5 B1048.5	15600
F9 B5 B1051.4	15600
F9 B5 B1049.5	15600
F9 B5 B1060.2	15600
F9 B5 B1058.3	15600
F9 B5 B1051.6	15600
F9 B5 B1060.3	15600
F9 B5 B1049.7	15600

Task 9

List the records which will display the month names, succesful landing_outcomes in ground pad ,booster versions, launch_site for the months in year 2017

Note: SQLite does not support monthnames. So you need to use substr(Date, 4, 2) as month to get the months and substr(Date,7,4)='2017' for year.

```
[20]: %%sql select Date, substr("--JanFebMarAprMayJunJulAugSepOctNovDec", strftime('%m', Date) * 3, 3) as Month, substr(Date,1,4) as Year,
      Landing_Outcome, Booster_Version, Launch_Site from SPACEXTABLE where Landing_Outcome = "Success (ground pad)" and substr(Date,1,4) = "2017"
```

```
* sqlite:///my_data1.db
Done.
```

```
[20]:
```

Date	Month	Year	Landing_Outcome	Booster_Version	Launch_Site
2017-02-19	Feb	2017	Success (ground pad)	F9 FT B1031.1	KSC LC-39A
2017-01-05	Jan	2017	Success (ground pad)	F9 FT B1032.1	KSC LC-39A
2017-03-06	Mar	2017	Success (ground pad)	F9 FT B1035.1	KSC LC-39A
2017-08-14	Aug	2017	Success (ground pad)	F9 B4 B1039.1	KSC LC-39A
2017-07-09	Jul	2017	Success (ground pad)	F9 B4 B1040.1	KSC LC-39A
2017-12-15	Dec	2017	Success (ground pad)	F9 FT B1035.2	CCAFS SLC-40

Task 10

Rank the count of landing outcomes (such as Failure (drone ship) or Success (ground pad)) between the date 2010-06-04 and 2017-03-20, in descending order

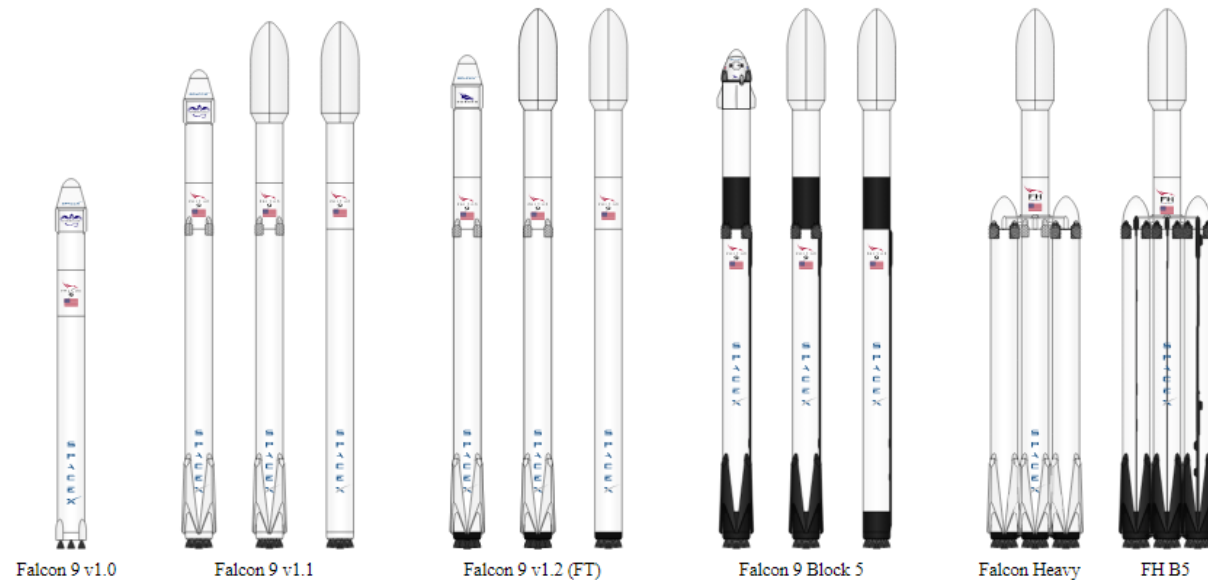
```
[21]: %%sql select Landing_Outcome, count(Landing_Outcome) from SPACEXTABLE where (Landing_Outcome = "Failure (drone ship)"
      or Landing_Outcome = "Success (ground pad)") and Date between "2010-06-04" and "2017-03-20" group by Landing_Outcome order by Date desc
```

```
* sqlite:///my_data1.db
Done.
```

```
[21]:
```

Landing_Outcome	count(Landing_Outcome)
Success (ground pad)	5
Failure (drone ship)	5

EDA with visualization slides



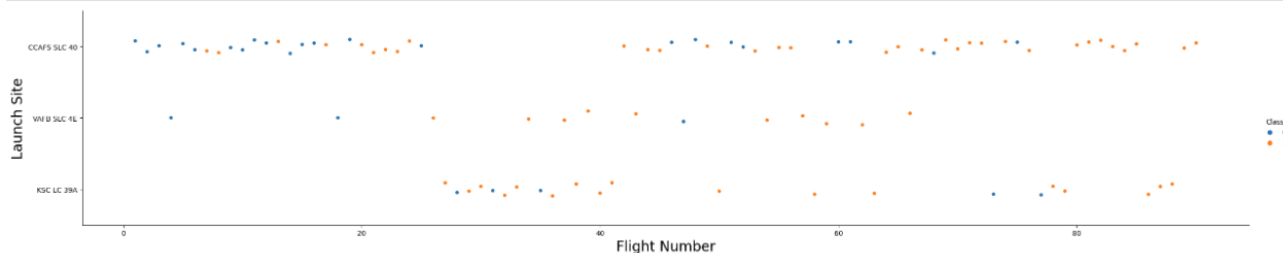
```
[6]: ### TASK 1: Visualize the relationship between Flight Number and Launch Site
```

Use the function `catplot` to plot `FlightNumber` vs `LaunchSite`, set the parameter `x` parameter to `FlightNumber`, set the `y` to `Launch Site` and set the parameter `hue` to `'class'`

```
[7]: # Plot a scatter point chart with x axis to be Flight Number and y axis to be the Launch site, and hue to be the class value
```

```
launchsite_transform = []
launchcode=0
for i, launch_site in enumerate(df["LaunchSite"]):
    if df["LaunchSite"][i].strip() == 'CCAFS SLC 40' :
        launchcode = 1
    elif df["LaunchSite"][i].strip() == 'CCAFS LC-40' :
        launchcode = 2
    elif df["LaunchSite"][i].strip() == 'KSC LC 39A' :
        launchcode = 3
    elif df["LaunchSite"][i].strip() == 'VAFB SLC 4E' :
        launchcode = 4
    launchsite_transform.append({'Flight_No' : df['FlightNumber'][i], 'Launch_Code' : launchcode, 'Launch_Site' : df["LaunchSite"][i]})

sns.catplot(y="LaunchSite", x="FlightNumber", hue="Class", data=df, aspect = 5)
plt.xlabel("Flight Number",fontsize=20)
plt.ylabel("Launch Site",fontsize=20)
plt.show()
```

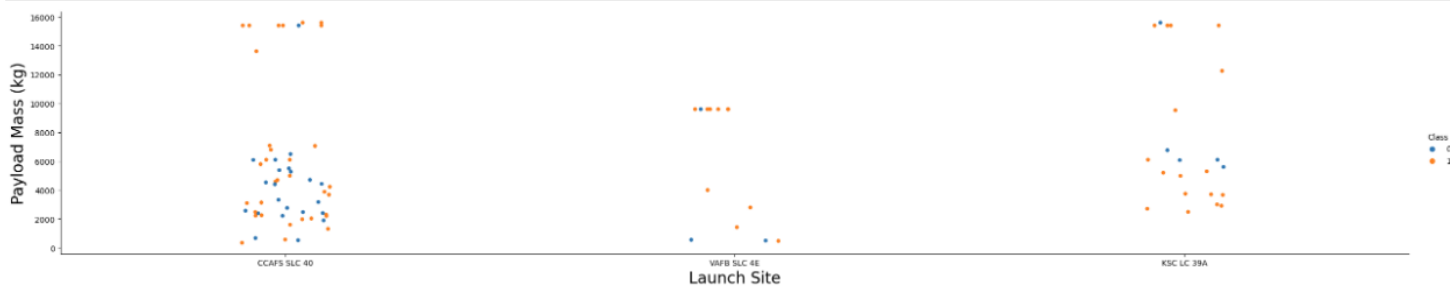


```
[8]: ### TASK 2: Visualize the relationship between Payload and Launch Site
```

We also want to observe if there is any relationship between launch sites and their payload mass.

```
[9]: # Plot a scatter point chart with x axis to be Pay Load Mass (kg) and y axis to be the Launch site, and hue to be the class value
```

```
sns.catplot(y="PayloadMass", x="LaunchSite", hue="Class", data=df, aspect = 5)
plt.xlabel("Launch Site",fontsize=20)
plt.ylabel("Payload Mass (kg)",fontsize=20)
plt.show()
```



Now if you observe Payload Vs. Launch Site scatter point chart you will find for the VAFB-SLC launchsite there are no rockets launched for heavy payload mass (greater than 10000).

```
[10]: ### TASK 3: Visualize the relationship between success rate of each orbit type
```

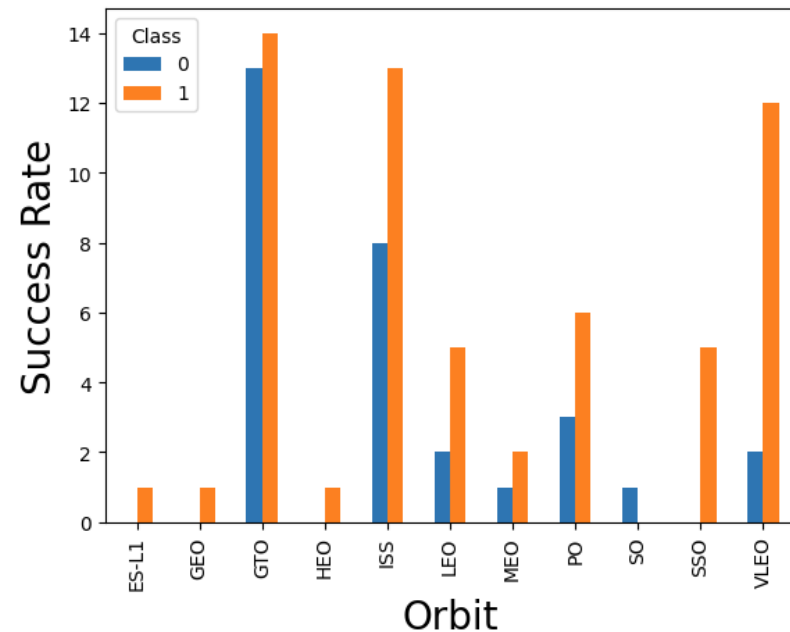
Next, we want to visually check if there are any relationship between success rate and orbit type.

Let's create a `bar chart` for the success rate of each orbit

```
[11]: # HINT use groupby method on Orbit column and get the mean of Class column
plot_data = df.groupby(['Orbit', 'Class']).size().reset_index(name='mean')
fig, ax = plt.subplots()
ax = plot_data.pivot(index='Orbit', columns='Class', values='mean').plot(kind='bar', stacked=False, ax=ax)

#sns.barplot(y="mean", x="Orbit", hue="Class", data=plot_data)
plt.xlabel("Orbit", fontsize=20)
plt.ylabel("Success Rate", fontsize=20)
```

```
[11]: Text(0, 0.5, 'Success Rate')
```



The success rate in the launch to GTO orbit is about 50% while that of SSO and VLEO is greater 90%.

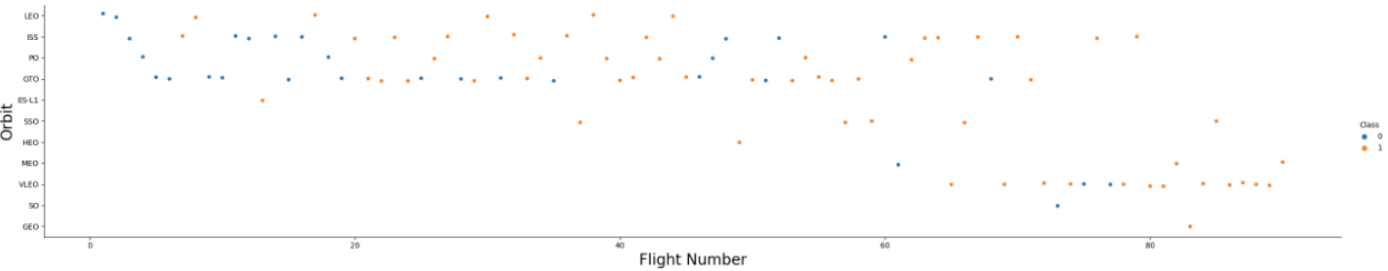
The probability of success is a very if the next launch is target at SSO, VLEO or LEO orbits.

Analyze the plotted bar chart try to find which orbits have high success rate.

```
[12]: ### TASK 4: Visualize the relationship between FlightNumber and Orbit type
```

For each orbit, we want to see if there is any relationship between FlightNumber and Orbit type.

```
[13]: # Plot a scatter point chart with x axis to be FlightNumber and y axis to be the Orbit, and hue to be the class value
sns.catplot(y="Orbit", x="FlightNumber", hue="Class", data=df, aspect = 5)
plt.xlabel("Flight Number",fontsize=20)
plt.ylabel("Orbit",fontsize=20)
plt.show()
```

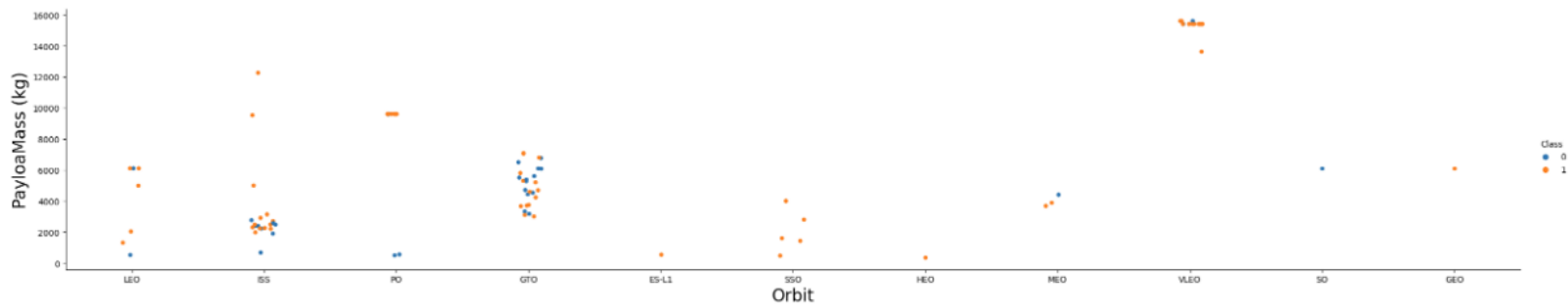


You should see that in the LEO orbit the Success appears related to the number of flights; on the other hand, there seems to be no relationship between flight number when in GTO orbit.

```
[14]: ### TASK 5: Visualize the relationship between Payload and Orbit type
```

Similarly, we can plot the Payload vs. Orbit scatter point charts to reveal the relationship between Payload and Orbit type

```
[15]: # Plot a scatter point chart with x axis to be Payload and y axis to be the Orbit, and hue to be the class value
sns.catplot(y="PayloadMass", x="Orbit", hue="Class", data=df, aspect = 5)
plt.xlabel("Orbit",fontsize=20)
plt.ylabel("PayloadMass (kg)",fontsize=20)
plt.show()
```



With heavy payloads the successful landing or positive landing rate are more for Polar,LEO and ISS.

However for GTO we cannot distinguish this well as both positive landing rate and negative landing(unsuccesful mission) are both there here.

```
[16]: ### TASK 6: Visualize the Launch success yearly trend
```

You can plot a line chart with x axis to be `Year` and y axis to be average success rate, to get the average launch success trend.

The function will help you get the year from the date:

```
[17]: # A function to Extract years from the date
```

```
year=[]  
def Extract_year():  
    for i in df["Date"]:  
        year.append(i.split("-")[0])  
    return year  
Extract_year()  
df["Date"] = year  
df.head()
```

```
[17]:
```

	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins	Reused	Legs	LandingPad	Block	ReusedCount	Serial	Longitude	Latitude	Class
0	1	2010	Falcon 9	6104.959412	LEO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B0003	-80.577366	28.561857	0
1	2	2012	Falcon 9	525.000000	LEO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B0005	-80.577366	28.561857	0
2	3	2013	Falcon 9	677.000000	ISS	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B0007	-80.577366	28.561857	0
3	4	2013	Falcon 9	500.000000	PO	VAFB SLC 4E	False Ocean	1	False	False	False	NaN	1.0	0	B1003	-120.610829	34.632093	0
4	5	2013	Falcon 9	3170.000000	GTO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B1004	-80.577366	28.561857	0

```
[18]: # Plot a Line chart with x axis to be the extracted year and y axis to be the success rate
```

```
plot_data = df.groupby(['Date', 'Class']).size().reset_index(name='mean')  
  
fig, ax = plt.subplots()  
ax = plot_data.pivot(index='Date', columns='Class', values='mean').plot(kind='line', stacked=False, ax=ax)
```



Increasing success
rate with year

IBM Developer

```
[22]: ### TASK 7: Create dummy variables to categorical columns
```

Use the function `get_dummies` and `features` dataframe to apply OneHotEncoder to the column `Orbits`, `LaunchSite`, `LandingPad`, and `Serial`. Assign the value to the variable `features_one_hot`, display the results using the method `head`. Your result dataframe must include all features including the encoded ones.

```
[23]: # HINT: Use get_dummies() function on the categorical columns  
features_one_hot = pd.get_dummies(features, prefix = ['Orbits', 'LaunchSite', 'LandingPad', 'Serial'])
```

```
[24]: ### TASK 8: Cast all numeric columns to 'float64'
```

Now that our `features_one_hot` dataframe only contains numbers cast the entire dataframe to variable type `float64`.

```
[25]: # HINT: use astype function  
features_one_hot.astype('float64')
```

```
[25]:
```

	FlightNumber	PayloadMass	Flights	GridFins	Reused	Legs	Block	ReusedCount	Orbits_E5-L1	Orbits_GEO	...	Serial_B1048	Serial_B1049	Serial_B1050	Serial_B1051	Serial_B1054	Serial_B1056	Serial_B1058	Serial_B1059	Serial_B1060	Serial_B1062
0	1.0	6104.959412	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	2.0	525.000000	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	3.0	677.000000	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	4.0	500.000000	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	5.0	3170.000000	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
...
85	86.0	15400.000000	2.0	1.0	1.0	1.0	5.0	2.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0
86	87.0	15400.000000	3.0	1.0	1.0	1.0	5.0	2.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0
87	88.0	15400.000000	6.0	1.0	1.0	1.0	5.0	5.0	0.0	0.0	...	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0
88	89.0	15400.000000	3.0	1.0	1.0	1.0	5.0	2.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0
89	90.0	3681.000000	1.0	1.0	0.0	1.0	5.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0

90 rows × 80 columns

Back

SKILLS NETWORK



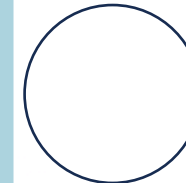
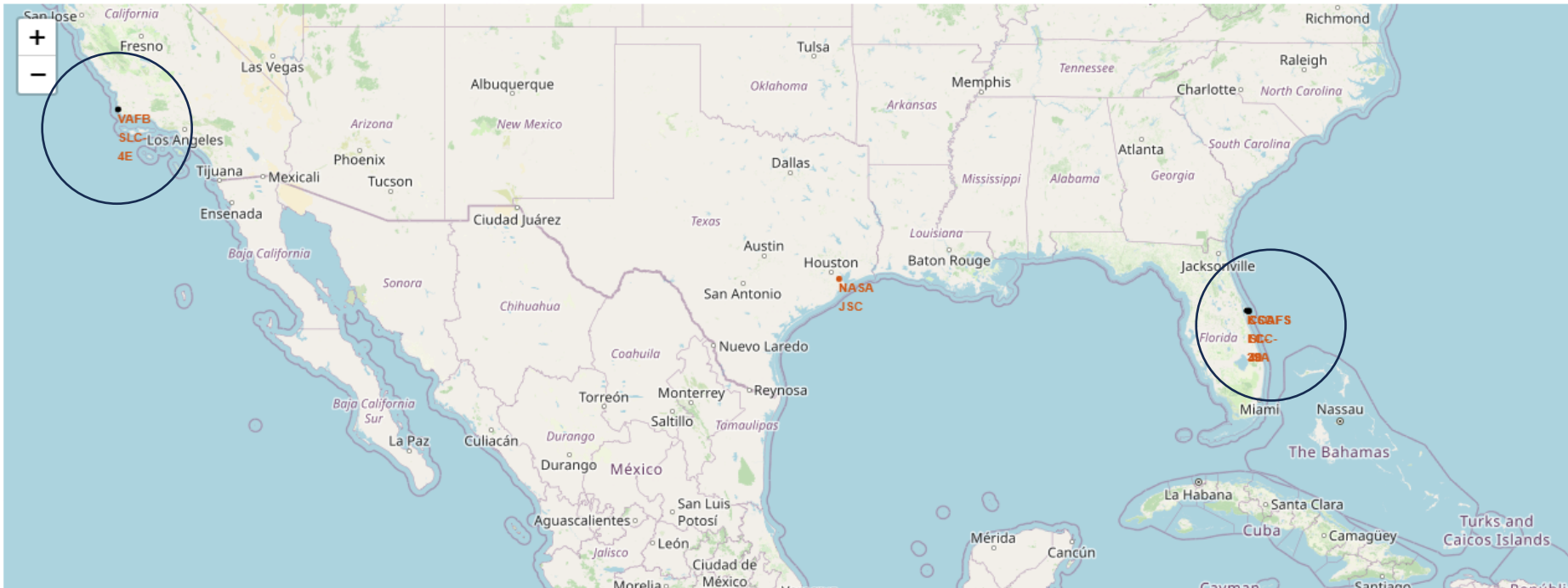
Interactive map plots



[4]: ## Task 1: Mark all launch sites on a map

```
[9]: # Initial the map
site_map = folium.Map(location=nasa_coordinate, zoom_start=10)
# For each launch site, add a Circle object based on its coordinate (Lat, Long) values. In addition, add Launch site name as a popup label
for i, location_info in launch_sites_df.iterrows():
    launch_coord = [launch_sites_df.Lat[i], launch_sites_df.Long[i]]
    #print(i, launch_coord, launch_sites_df['Launch Site'][i])
    folium.Circle(launch_coord, radius=100, color='#000000', fill=True, Popup=launch_sites_df['Launch Site'][i]).add_to(site_map)
    folium.map.Marker(
        launch_coord,
        # 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>' % launch_sites_df['Launch Site'][i],
        )
    ).add_to(site_map)
site_map.add_child(circle)
site_map.add_child(marker)
```

[9]:

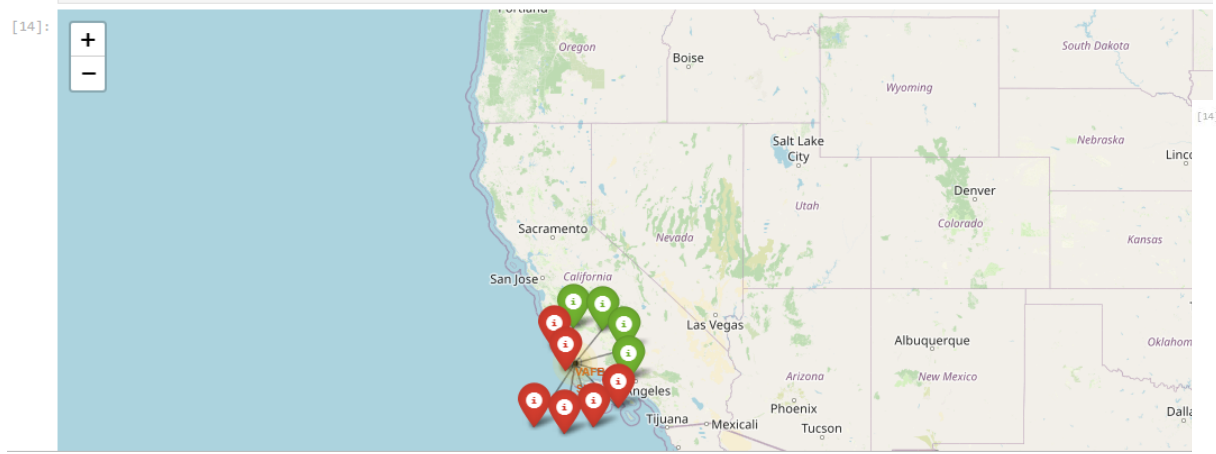


Launch site

[10]: # Task 2: Mark the success/failed launches for each site on the map

```
[14]: # Add marker_cluster to current site_map
site_map.add_child(marker_cluster)

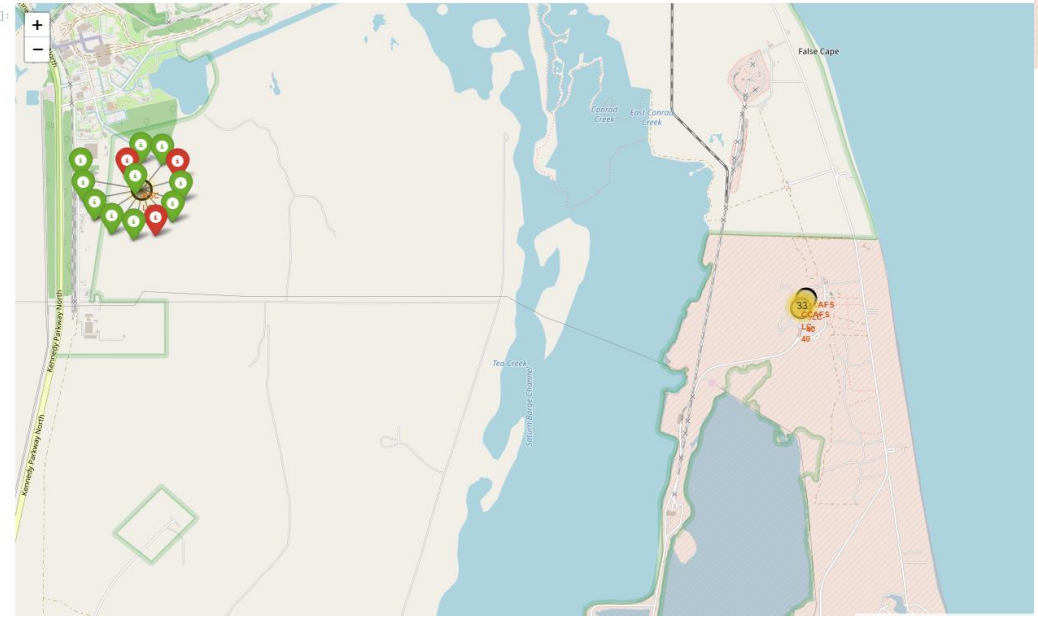
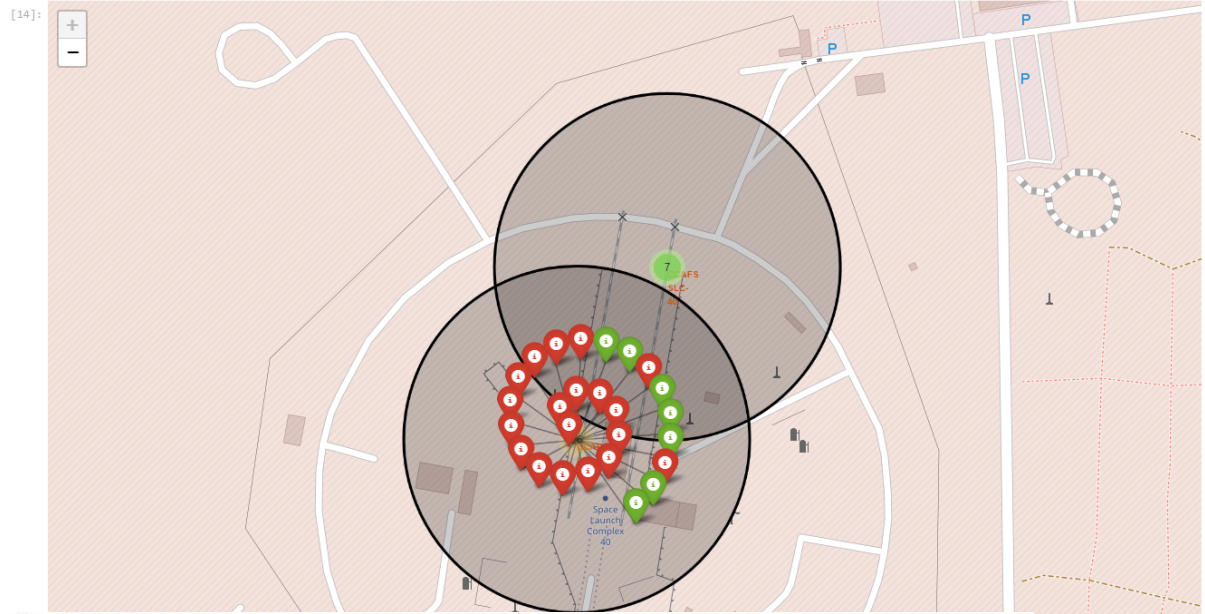
# for each row in spacex_df data frame
# create a Marker object with its coordinate
# and customize the Marker's icon property to indicate if this launch was succeeded or failed,
# e.g., icon=folium.Icon(color='white', icon_color=row['marker_color'])
for index, record in spacex_df.iterrows():
    # TODO: Create and add a Marker cluster to the site map
    # marker = folium.Marker(...)
    launch_coord = [spacex_df.Lat[index], spacex_df.Long[index]]
    test_class = spacex_df['class'][index]
    #
    # Success case
    if (test_class == 1) :
        map_icon=folium.Icon(color='green')
        marker = folium.Marker(launch_coord, icon = map_icon,)
    # Failure case
    if (test_class == 0) :
        map_icon=folium.Icon(color='red')
        marker = folium.Marker(launch_coord, icon = map_icon,)
    #
    marker_cluster.add_child(marker)
#
site_map
```



Successful landing



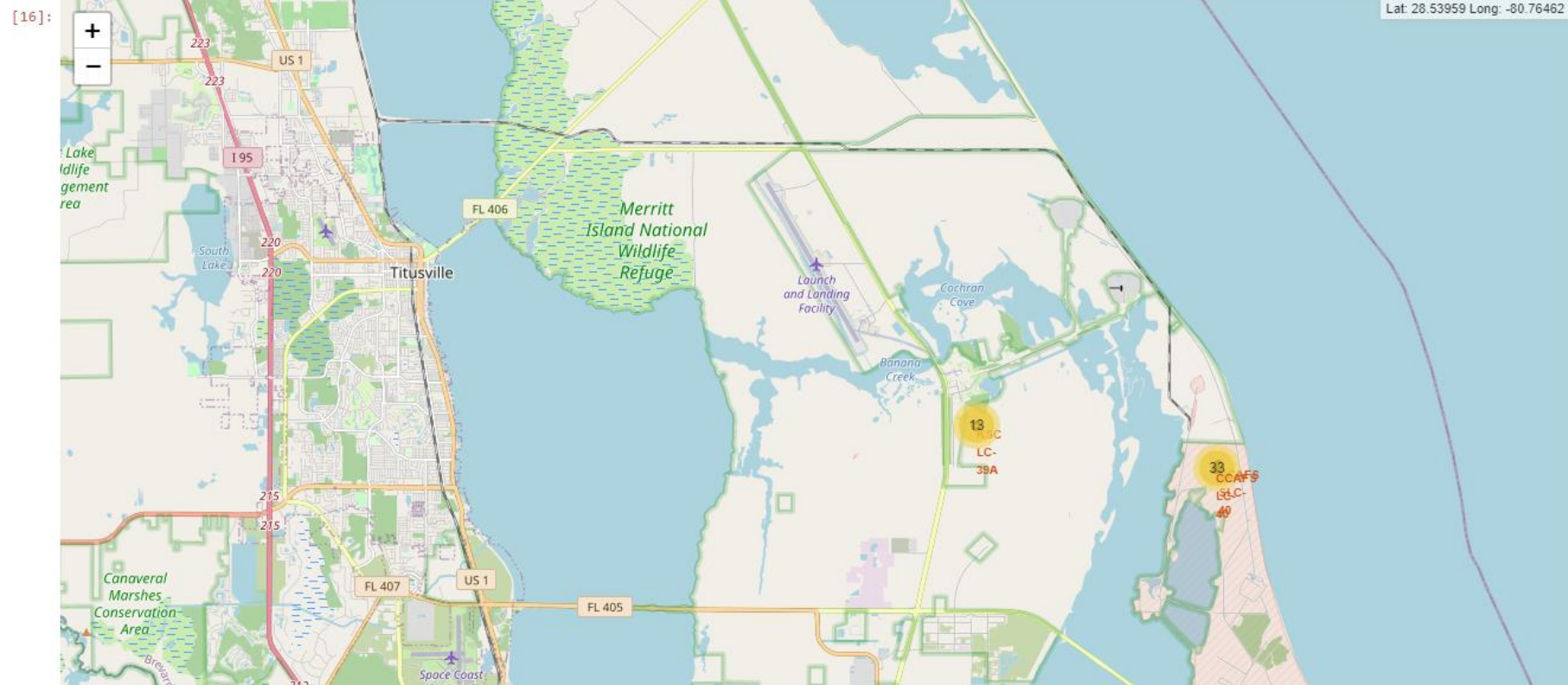
Unsuccessful landing



```
[15]: # TASK 3: Calculate the distances between a launch site to its proximities
```

```
[16]: # 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
```



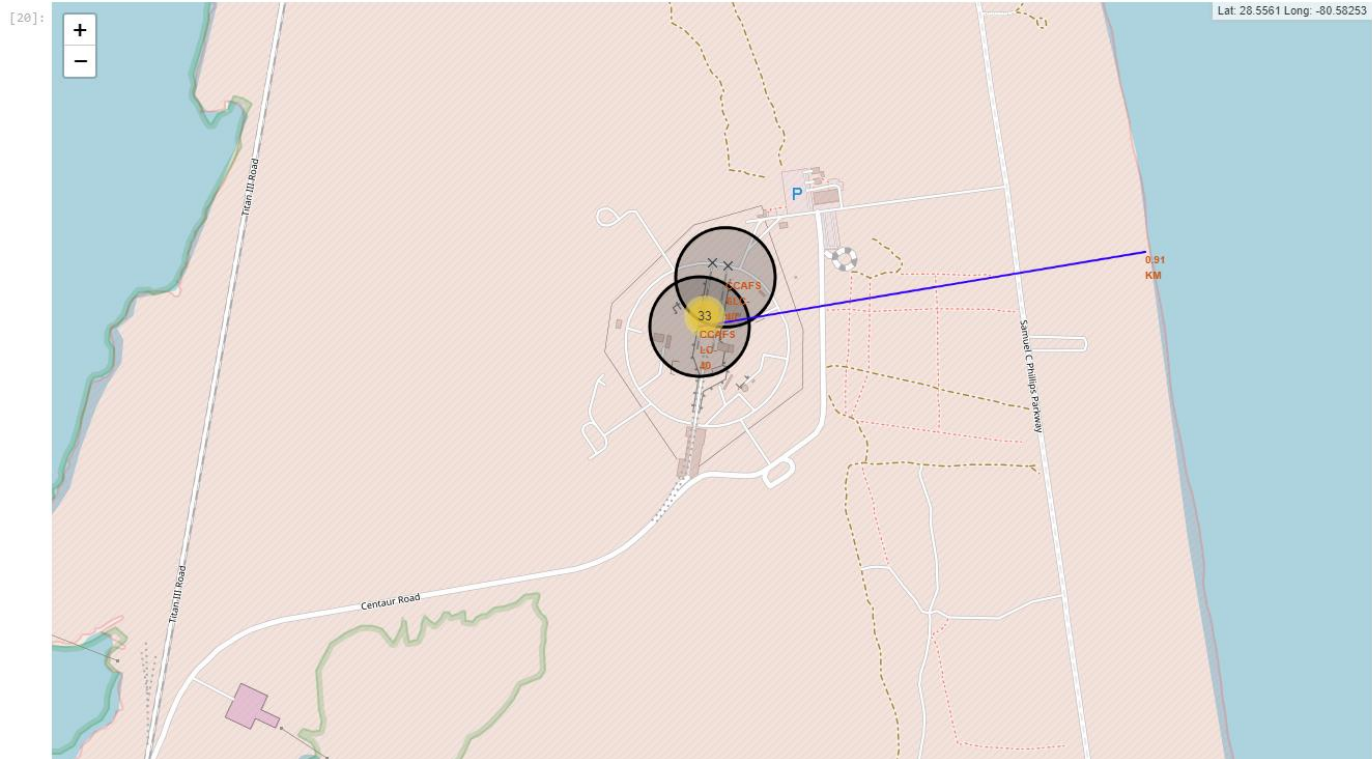
TODO: Mark down a point on the closest coastline using MousePosition and calculate the distance between the coastline point and the launch site.

```
[18]: # 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)

[19]: # Create and add a folium.Marker on your selected closest coastline point on the map
# Display the distance between coastline point and launch site using the icon property
# for example
launch_site = [spacex_df.head(1).Lat, spacex_df.head(1).Long]
coastline_coord = [28.56367, -80.56813]
distance_coastline = calculate_distance(spacex_df.head(1).Lat, spacex_df.head(1).Long, 28.56367, -80.56813)
#
distance_marker = folium.Marker(
    coastline_coord, tooltip="Coastline", popup="Coastline",
    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_coastline)
    )
)
site_map.add_child(distance_marker)
site_map
```

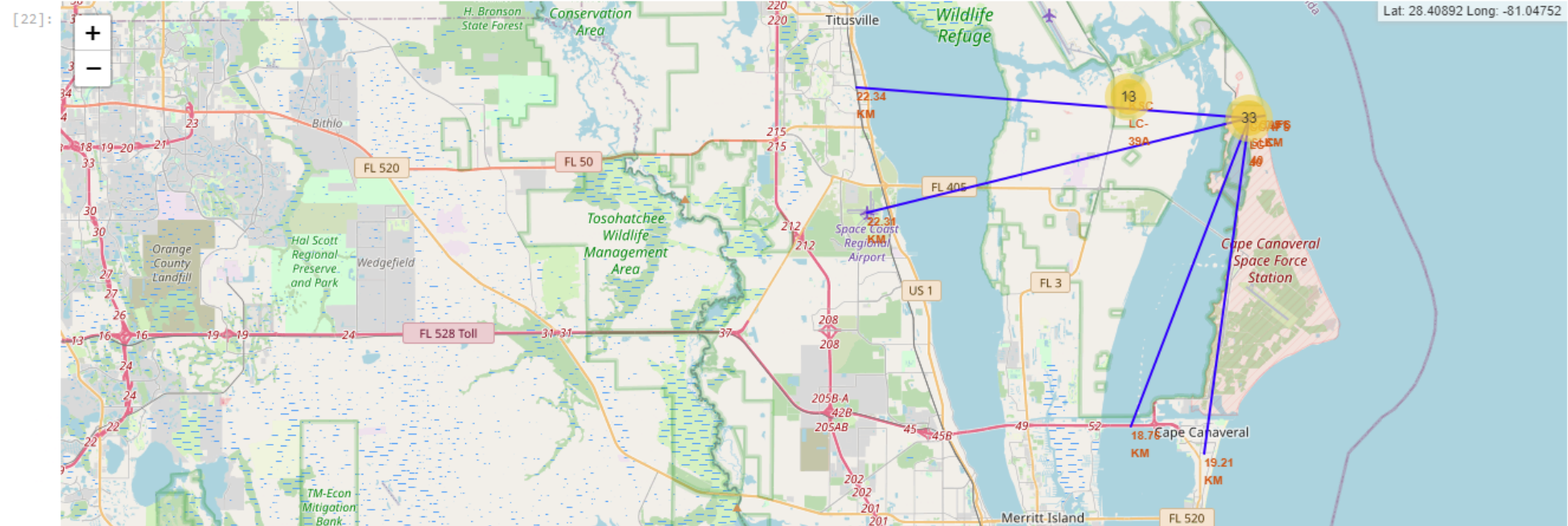
TODO: Draw a PolyLine between a launch site to the selected coastline point

```
[20]: lines=folium.PolyLine(locations=[[spacex_df['Lat'][0], spacex_df['Long'][0]], [28.56367, -80.56813]], color='blue', weight=2)
site_map.add_child(lines)
```



```
[21]: # Create a marker with distance to a closest city, railway, highway, etc.  
# Draw a line between the marker to the launch site
```

```
[22]: # create a dataframe for the locations to be checked  
dist_checker_df = pd.DataFrame({'Name' : ['City', 'Railway', 'Highway', 'Airport'], 'Lat' : [28.39106, 28.57813, 28.40484, 28.51425], 'Long' : [-80.60293, -80.8053, -80.64609, -80.  
#  
# generate the marker  
for i, record in dist_checker_df.iterrows():  
    distance = calculate_distance(spacex_df['Lat'][0], spacex_df['Long'][0], dist_checker_df['Lat'][i], dist_checker_df['Long'][i])  
    distance  
    distance_marker = folium.Marker(  
        [dist_checker_df['Lat'][i], dist_checker_df['Long'][i]], tooltip=dist_checker_df['Name'][i], popup=dist_checker_df['Name'][i],  
        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)  
        )  
    )  
    site_map.add_child(distance_marker)  
#  
# now draw the line to the markers  
lines=folium.PolyLine(locations=[[spacex_df['Lat'][0], spacex_df['Long'][0]], [dist_checker_df['Lat'][i], dist_checker_df['Long'][i]]], color='blue', weight=2)  
site_map.add_child(lines)  
site_map
```



Plotly Dash plots



TASK 1: Add a Launch Site Drop-down Input Component

Table of Contents

- Install python packages required to run the application.

Copy and paste the below command to the terminal.

```
1 python3.8 -m pip install pandas dash
```

```
theia@theiadosker-anita: /home/project x
theia@theiadosker-anita: /home/project$ python3.8 -m pip install pandas dash
Defaulting to user installation because normal site-packages is not writeable
Collecting pandas
  Downloading pandas-1.5.3-cp38-cp38-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (12.2 MB)
    11.2/12.2 MB 47.0 MB/s eta 0:00:00
Collecting dash
  Downloading dash-2.8.1-py3-none-any.whl (9.9 MB)
    9.9/9.9 MB 46.1 MB/s eta 0:00:00
Requirement already satisfied: pytz>=2020.1 in /home/theia/.local/lib/python3.8/site-packages (from pandas) (2022.1)
Requirement already satisfied: python-dateutil>=2.8.1 in /home/theia/.local/lib/python3.8/site-packages (from pandas) (2.8.2)
Collecting numpy>=1.20.3
  Downloading numpy-1.24.2-cp38-cp38-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (17.3 MB)
    17.3/17.3 MB 45.1 MB/s eta 0:00:00
Collecting dash-html-components>=2.0.0
  Downloading dash_html_components-2.0.0-py3-none-any.whl (4.1 kB)
Collecting plotly>=5.0.0
  Downloading plotly-5.13.1-py2.py3-none-any.whl (15.2 MB)
    15.2/15.2 MB 45.0 MB/s eta 0:00:00
Collecting dash-core-components>=2.0.0
  Downloading dash_core_components-2.0.0-py3-none-any.whl (3.8 kB)
Collecting dash-table>=5.0.0
  Downloading dash_table-5.0.0-py3-none-any.whl (3.9 kB)
Requirement already satisfied: flask>=1.0.4 in /home/theia/.local/lib/python3.8/site-packages (from dash) (2.2.2)
Requirement already satisfied: importlib-metadata>=1.6.0 in /home/theia/.local/lib/python3.8/site-packages (from flask>=1.0.4->dash) (4.12.0)
Requirement already satisfied: Werkzeug>=2.2.2 in /home/theia/.local/lib/python3.8/site-packages (from flask>=1.0.4->dash) (2.2.2)
```

Download a skeleton dashboard application and dataset

First, let's get the SpaceX Launch dataset for this lab:

- Run the following `wget` command line in the terminal to download dataset as `spacex_launch_dash.csv`

```
1 wget "https://cf-courses-data.s3.us.cloud-object-storage.appdomain.c"
```

- Download a skeleton Dash app to be completed in this lab:

```
1 wget "https://cf-courses-data.s3.us.cloud-object-storage.appdomain.c"
```

- Test the skeleton app by running the following command in the terminal:

```
1 python3.8 spacex_dash_app.py
```

- Observe the port number (8050) shown in the terminal.

```
theia@theiadosker-anita: /home/project x
theia@theiadosker-anita: /home/project$ python3.8 spacex_dash_app.py
spacex_dash_app.py:4: UserWarning:
The dash_html_components package is deprecated. Please replace
'import dash_html_components as html' with 'from dash import html'
  import dash_html_components as html
```

Your Application x

<https://omuritala-8050.theiadosker-3-labs-prod-theiak8s-4-tor01.proxy.cognitiveclass.ai/>

SpaceX Launch Records Dashboard

Successful Launches Per Site

Select a Launch Site:

All Sites
CCAFS LC-40
CCAFS SLC-40
VAFB SLC-4E
KSC LC-39A

Download range (Kc)

Problems theia@theiadosker-omuritala: /home/project x

127.0.0.1 - - [23/Sep/2023 09:36:15] "GET /_dash-component-suites/dash/html/dash_html_components.v2_0_14m1695476137.min.js HTTP/1.1" 200 -
127.0.0.1 - - [23/Sep/2023 09:36:18] "GET /_dash-dependencies HTTP/1.1" 200 -
127.0.0.1 - - [23/Sep/2023 09:36:18] "GET /_dash-layout HTTP/1.1" 200 -
127.0.0.1 - - [23/Sep/2023 09:36:18] "POST /_dash-update-component HTTP/1.1" 200 -
127.0.0.1 - - [23/Sep/2023 09:36:18] "GET /_dash-component-suites/dash/dcc/async-slider.js HTTP/1.1" 200 -
127.0.0.1 - - [23/Sep/2023 09:36:18] "GET /_dash-component-suites/dash/dcc/async-graph.js HTTP/1.1" 200 -
127.0.0.1 - - [23/Sep/2023 09:36:18] "GET /_dash-component-suites/dash/dcc/async-dropdown.js HTTP/1.1" 200 -
127.0.0.1 - - [23/Sep/2023 09:36:18] "GET /_dash-component-suites/plotly/package_data/plotly.min.js HTTP/1.1" 200 -
127.0.0.1 - - [23/Sep/2023 09:36:18] "POST /_dash-update-component HTTP/1.1" 200 -

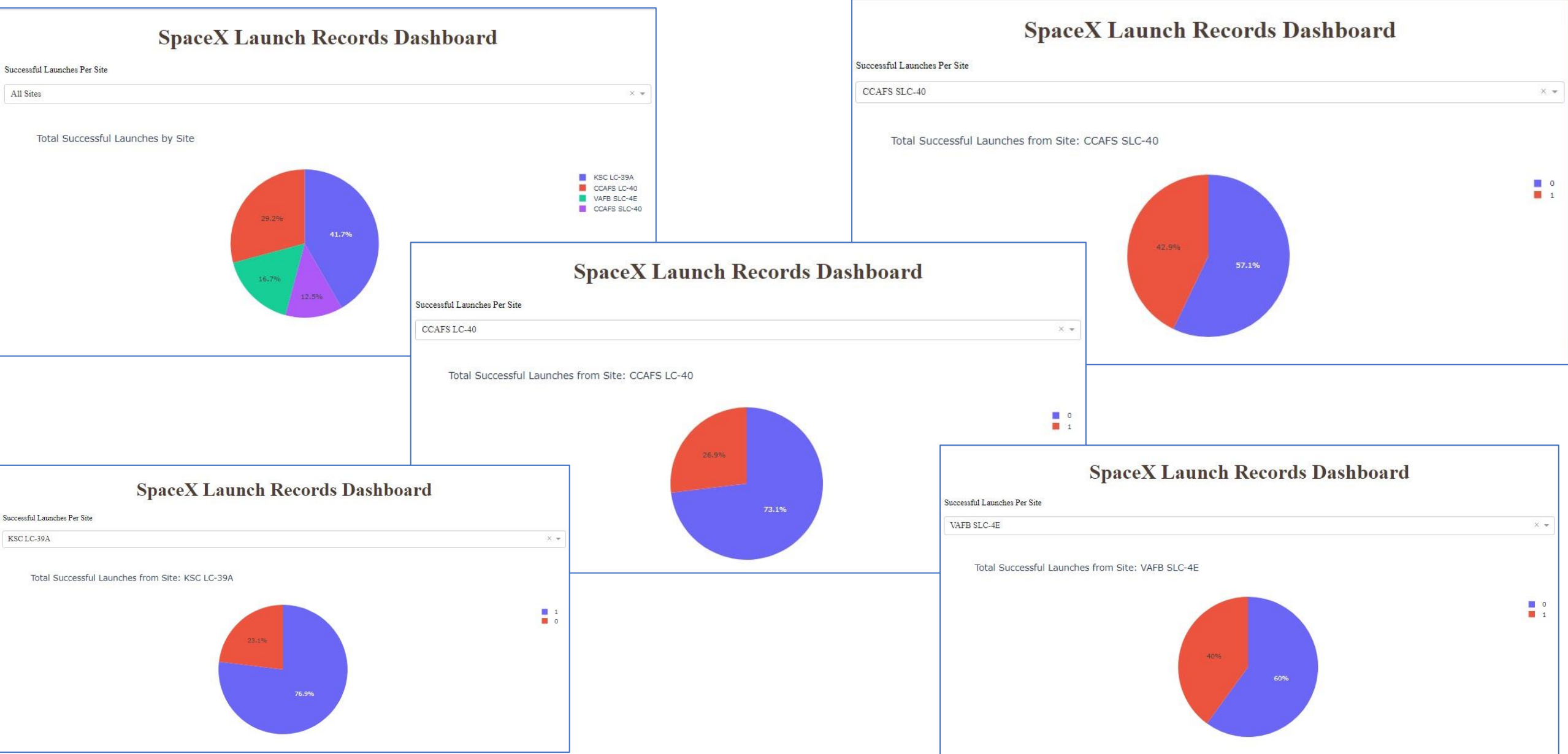
Go Live

IBM Developer

Back

SKILLS NETWORK

TASK 2: Add a callback function to render success-pie-chart based on selected site dropdown



TASK 3: Add a Range Slider to Select Payload

SpaceX Launch Records Dashboard

Successful Launches Per Site

All Sites

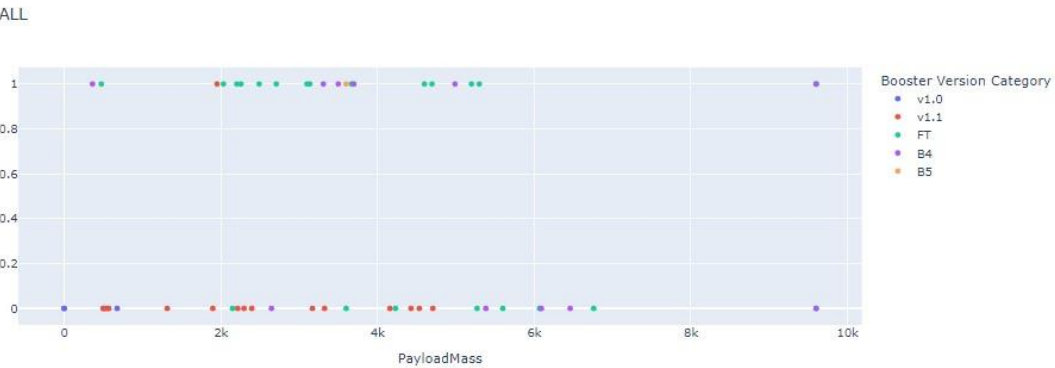
Total Successful Launches by Site



Payload range (Kg):



Correlation between payload and launch success



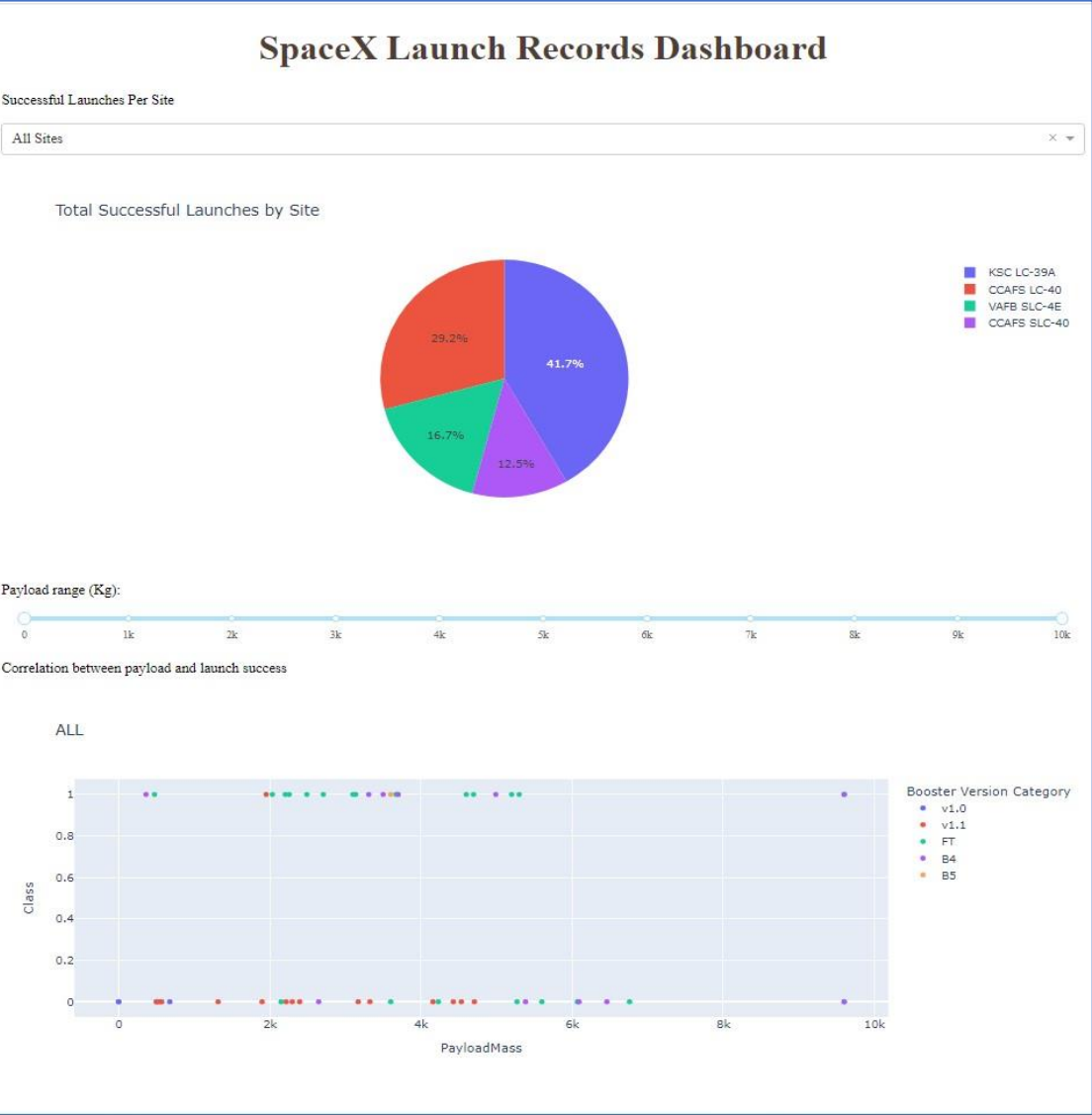
Range Slider



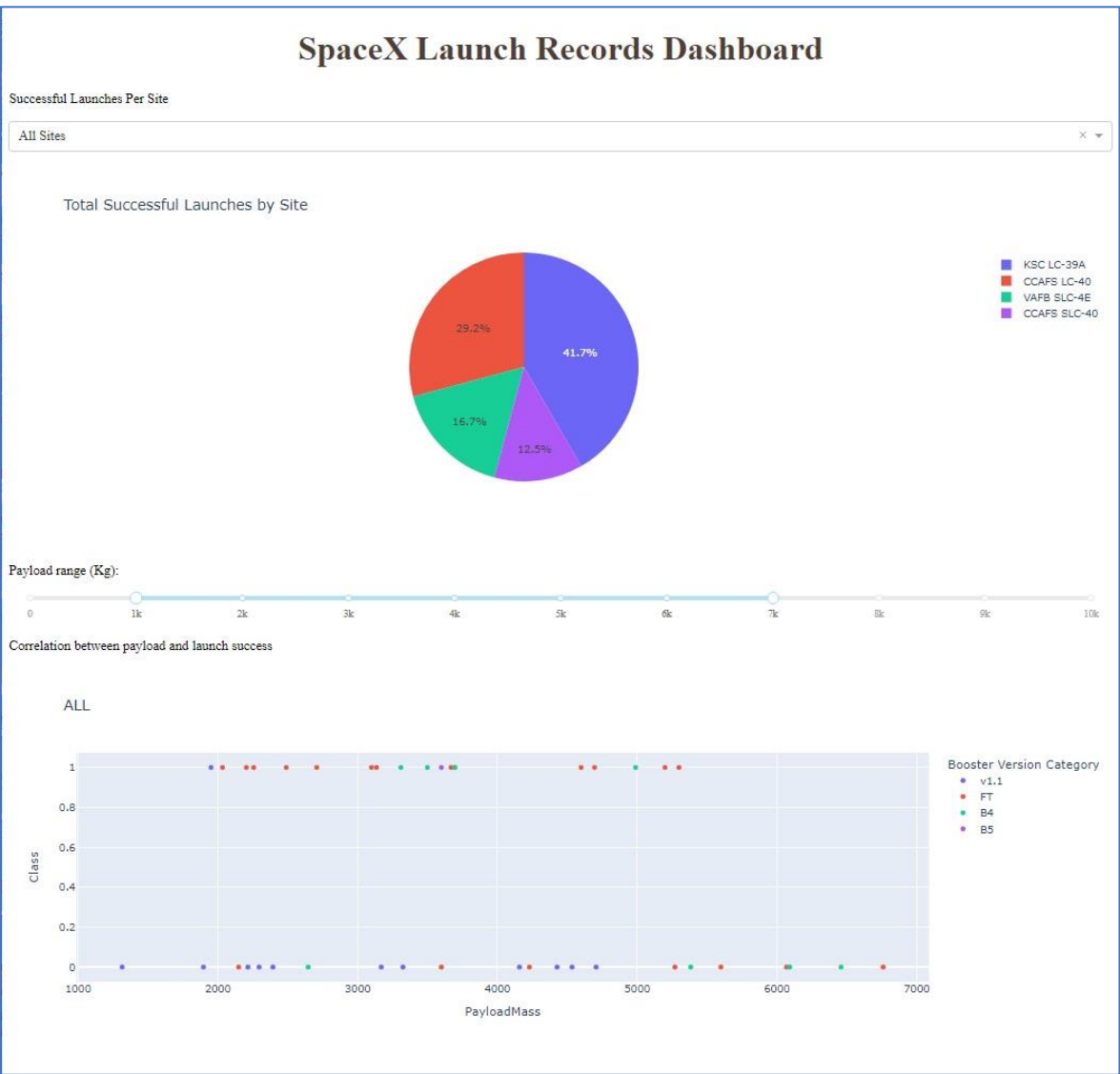
TASK 4: Add a callback function to render the

success-payload-scatter-chart

scatter plot



No data range selected



With data range selected

Predictive analysis results



Data partitioning

We split the data into training and testing data using the function `train_test_split`. The training data is divided into validation data, a second set used for training data; then the models are trained and hyperparameters are selected using the function `GridSearchCV`.

TASK 3

Use the function `train_test_split` to split the data `X` and `Y` into training and test data. Set the parameter `test_size` to 0.2 and `random_state` to 2. The training data and test data should be assigned to the following labels.

```
X_train, X_test, Y_train, Y_test
```

```
[14]: X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.20, random_state = 2, shuffle = True)
```

we can see we only have 18 test samples.



```
[15]: Y_test.shape
```

```
[15]: (18,)
```

Logistic regression model & performance

```
[20]: print("tuned hpyerparameters :(best parameters) ",logreg_cv.best_params_)  
      print("accuracy :",logreg_cv.best_score_)  
  
tuned hpyerparameters :(best parameters) {'C': 0.01, 'penalty': 'l2', 'solver': 'lbfgs'}  
accuracy : 0.8464285714285713
```

TASK 5

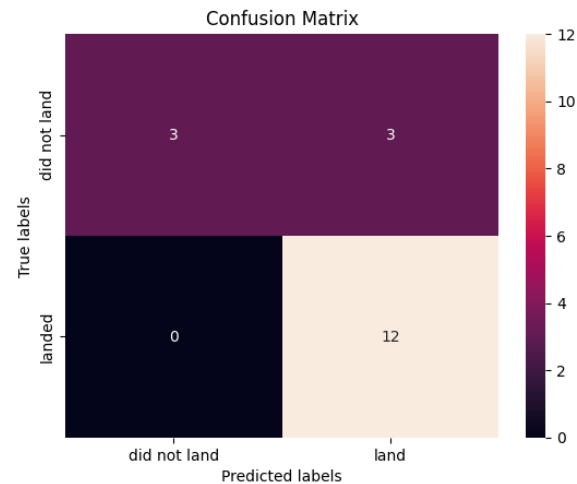
Calculate the accuracy on the test data using the method `score` :

```
[21]: score = logreg_cv.score(X_test, Y_test)  
      score
```

```
[21]: 0.8333333333333334
```

Lets look at the confusion matrix:

```
[22]: yhat=logreg_cv.predict(X_test)  
      plot_confusion_matrix(Y_test,yhat)
```



Examining the confusion matrix, we see that logistic regression can distinguish between the different classes. We see that the major problem is false positives.

Support vector machine model & performance

```
[26]: print("tuned hyperparameters :(best parameters) ",svm_cv.best_params_)
      print("accuracy :",svm_cv.best_score_)

tuned hyperparameters :(best parameters) {'C': 1.0, 'gamma': 0.03162277660168379, 'kernel': 'sigmoid'}
accuracy : 0.8482142857142856
```

TASK 7

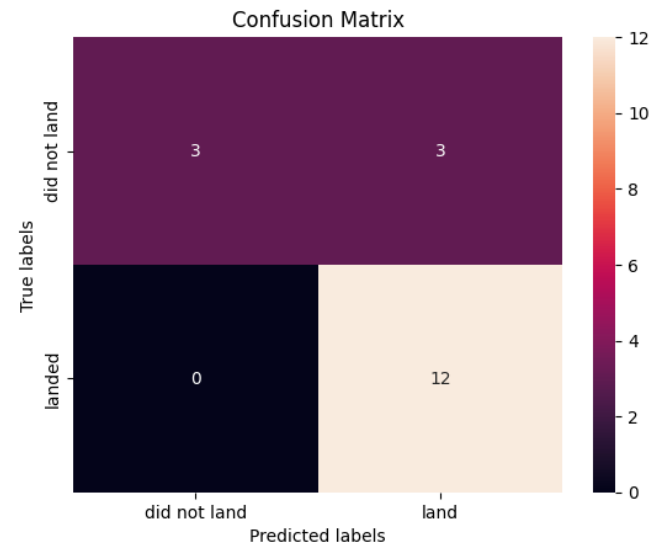
Calculate the accuracy on the test data using the method `score`:

```
[27]: score = svm_cv.score(X_test, Y_test)
      score
```

```
[27]: 0.8333333333333334
```

We can plot the confusion matrix

```
[28]: yhat=svm_cv.predict(X_test)
      plot_confusion_matrix(Y_test,yhat)
```



Decision tree classifier model & performance

```
[32]: print("tuned hyperparameters :(best parameters) ",tree_cv.best_params_)
      print("accuracy :",tree_cv.best_score_)

tuned hyperparameters :(best parameters) {'criterion': 'gini', 'max_depth': 2, 'max_features': 'sqrt', 'min_samples_leaf': 2, 'min_samples_split': 2, 'splitter': 'best'}
accuracy : 0.8767857142857143
```

TASK 9

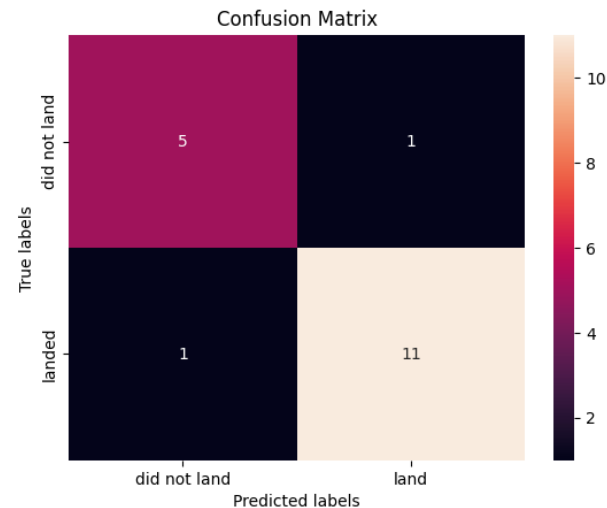
Calculate the accuracy of tree_cv on the test data using the method `score`:

```
[33]: score = tree_cv.score(X_test, Y_test)
      score
```

```
[33]: 0.8888888888888888
```

We can plot the confusion matrix

```
[34]: yhat = tree_cv.predict(X_test)
      plot_confusion_matrix(Y_test,yhat)
```



K nearest neighbour model and performance

```
[38]: print("tuned hyperparameters :(best parameters) ",knn_cv.best_params_)  
      print("accuracy :",knn_cv.best_score_)  
  
tuned hyperparameters :(best parameters) {'algorithm': 'auto', 'n_neighbors': 10, 'p': 1}  
accuracy : 0.8482142857142858
```

TASK 11

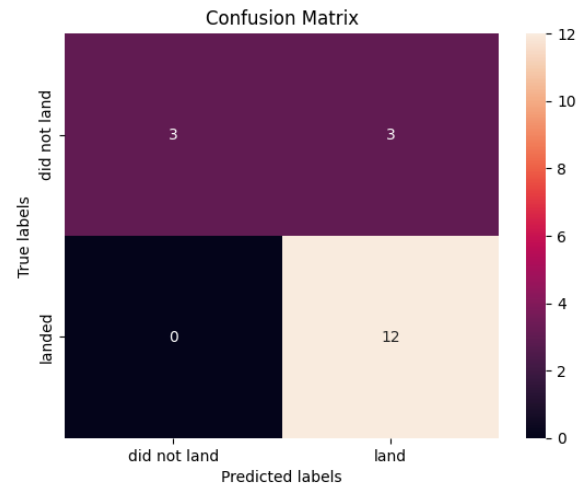
Calculate the accuracy of knn_cv on the test data using the method `score`:

```
[39]: score = knn_cv.score(X_test, Y_test)  
      score
```

```
[39]: 0.8333333333333334
```

We can plot the confusion matrix

```
[40]: yhat = knn_cv.predict(X_test)  
      plot_confusion_matrix(Y_test,yhat)
```



Model evaluation

Find the method performs best:

```
[41]: model_list = pd.DataFrame({'Model': ['Logistic regression', 'Support Vector Machine', 'Decision tree classifier', 'K nearest neighbors'],  
                             'Accuracy': [logreg_cv.best_score_, svm_cv.best_score_, tree_cv.best_score_, knn_cv.best_score_]})  
model_list
```

```
[41]:
```

	Model	Accuracy
0	Logistic regression	0.846429
1	Support Vector Machine	0.848214
2	Decision tree classifier	0.876786
3	K nearest neighbors	0.848214

```
[42]: max_accuracy = model_list['Accuracy'].max()  
best_model = model_list["Model"].where(model_list["Accuracy"] == model_list['Accuracy'].max())  
best_model.dropna(inplace=True)  
my_model = best_model.to_string()  
print(my_model + " has the highest accuracy of: %.2f" % max_accuracy)
```

```
2 Decision tree classifier has the highest accuracy of: 0.88
```