IBM Developer
SKILLS NETWORK

Winning Space Race with Data Science

<Gözdem Çavdar>
<20.01.2023>

Outline

- Executive Summary
- Introduction
- Methodology
- Results
- Conclusion
- Appendix

Executive Summary

- Summary of methodologies
  - ❖ Collecting the Data with Application Programming Interface (API)
  - ❖ Data Wrangling
  - ❖ Explanatory Data Analysis (EDA) with Structured Query Language (SQL)
  - ❖ Explanatory Data Analysis (EDA) with Visualization
  - ❖ The Data Visualization with Folium
  - ❖ Machine Learning Prediction

- Summary of all results
  - ❖ The Results of Explanatory Data Analysis (EDA)
  - ❖ The Results of Launch Sites Proximities Analysis
  - ❖ The Results of Predictive Analysis

Introduction

We would like to make a prediction about the successful landing of Falcon9 in its first stage in this project. On its website, SpaceX promotes Falcon 9 rocket flights for 62 million dollars; other suppliers charge upwards of 165 million dollars for each launch. A large portion of the savings is due to SpaceX's ability to reuse the first stage. Therefore, if we can figure out if the first stage will land, we can figure out how much a launch will cost. If a different business want to compete with SpaceX for a rocket launch, it may use the information provided here.

Section 1

Methodology

Methodology

Executive Summary

- Data collection methodology:
  - The rocket launch data was requested from SpaceX Application Programming Interface (API)
- Perform data wrangling
  - The number of launches on each site were determined, the number and occurrence of each orbit were calculated, the number and occurence of mission outcome per orbit type were calculated, a landing outcome label from Outcome column were created.
- Perform exploratory data analysis (EDA) using visualization and SQL
- Perform interactive visual analytics using Folium
- Perform predictive analysis using classification models
  - How to build, tune, evaluate classification models

Data Collection

- The SpaceX launch data were requested from SpaceX Application Programming Interface (API) using the GET request.
- Pandas dataframe were created and filtered to only include Falcon9 launches.
- The missing values were replaced in the data with the mean of values.

Data Collection – SpaceX API

In [6]:
```
spacex_url="https://api.spacexdata.com/v4/launches/past"
```

In [7]:
```
response = requests.get(spacex_url)
```

Check the content of the response

In [8]:
```
print(response.content)
```

```
":"dLQ2tZEH6G0","article":"https://en.wikipedia.org/wiki/Ratsat","wikipedia":"https://
en.wikipedia.org/wiki/Ratsat"},"static_fire_date_utc":"2008-09-20T00:00:00.000Z","stat
ic_fire_date_unix":1221868800,"net":false,"window":0,"rocket":"5e9d0d95eda69955f709d1e
b","success":true,"failures":[],"details":"Ratsat was carried to orbit on the first su
ccessful orbital launch of any privately funded and developed, liquid-propelled carrie
r rocket, the\xc2\xa0SpaceX Falcon 1","crew":[],"ships":[],"capsules":[],"payloads":["
5eb0e4b7b6c3bb0006eeb1e5"],"launchpad":"5e9e4502f5090995de566f86","flight_number":4,"na
me":"RatSat","date_utc":"2008-09-28T23:15:00.000Z","date_unix":1222643700,"date_local
":"2008-09-28T11:15:00+12:00","date_precision":"hour","upcoming":false,"cores":[{"core
":"5e9e289ef3591855dc3b2626","flight":1,"gridfins":false,"legs":false,"reused":false,"
landing_attempt":false,"landing_success":null,"landing_type":null,"landpad":null}],"au
to_update":true,"tbd":false,"launch_library_id":null,"id":"5eb87cdbffd86e000604b32d"},
{"fairings":{"reused":false,"recovery_attempt":false,"recovered":false,"ships":[]},"li
nks":{"patch":{"small":"https://images2.imgbox.com/ab/5a/Pequxd5d_o.png","large":"http
s://images2.imgbox.com/92/e4/7Cf6MLY0_o.png"},"reddit":{"campaign":null,"launch":null,
"media":null,"recovery":null},"flickr":{"small":[],"original":[]},"presskit":"http://w
ww.spacex.com/press/2012/12/19/spacexs-falcon-1-successfully-delivers-razaksat-satelli
te-orbit","webcast":"https://www.youtube.com/watch?v=yTaIDooc80g","youtube_id":"yTaIDo
oc80g","article":"http://www.spacex.com/news/2013/02/12/falcon-1-flight-5","wikipedia"
:"https://en.wikipedia.org/wiki/RazakSAT"},"static_fire_date_utc":null,"static_fire_da
te_unix":null,"net":false,"window":0,"rocket":"5e9d0d95eda69955f709d1eb","success":tru
e,"failures":[],"details":null,"crew":[],"ships":[],"capsules":[],"payloads":["5eb0e4b
```

You should see the response contains massive information about SpaceX launches. Next, let's try to discover some more relevant information for this project.

Task 1: Request and parse the SpaceX launch data using the GET request

To make the requested JSON results more consistent, we will use the following static response object for this project:

In [9]:
```
static_json_url='https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM
```

We should see that the request was successfull with the 200 status response code

In [10]:
```
response.status_code
```

Out[10]: 200

Now we decode the response content as a Json using `.json()` and turn it into a Pandas dataframe using `.json_normalize()`

In [12]:
```
# Use json_normalize meethod to convert the json result into a dataframe
data = pd.read_json('https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud
```

Using the dataframe `data` print the first 5 rows

In [13]:
```
# Get the head of the dataframe
data.head(5)
```

In [14]:
```
# Lets take a subset of our dataframe keeping only the features we want and the flight
data = data[['rocket', 'payloads', 'launchpad', 'cores', 'flight_number', 'date_utc']]

# We will remove rows with multiple cores because those are falcon rockets with 2 extra
data = data[data['cores'].map(len)==1]
data = data[data['payloads'].map(len)==1]

# Since payloads and cores are lists of size 1 we will also extract the single value in
data['cores'] = data['cores'].map(lambda x : x[0])
data['payloads'] = data['payloads'].map(lambda x : x[0])

# We also want to convert the date_utc to a datetime datatype and then extracting the d
data['date'] = pd.to_datetime(data['date_utc']).dt.date

# Using the date we will restrict the dates of the launches
data = data[data['date'] <= datetime.date(2020, 11, 13)]
```

- From the `rocket` we would like to learn the booster name
- From the `payload` we would like to learn the mass of the payload and the orbit that it is going to
- From the `launchpad` we would like to know the name of the launch site being used, the longitude, and the latitude.
- From `cores` we would like to learn the outcome of the landing, the type of the landing, number of flights with that core, whether gridfins were used, whether the core is reused, whether legs were used, the landing pad used, the block of the core which is a number used to seperate version of cores, the number of times this specific core has been reused, and the serial of the core.

The data from these requests will be stored in lists and will be used to create a new dataframe.

In [15]:
```
#Global variables
BoosterVersion = []
PayloadMass = []
Orbit = []
LaunchSite = []
Outcome = []
Flights = []
GridFins = []
Reused = []
Legs = []
LandingPad = []
Block = []
ReusedCount = []
Serial = []
Longitude = []
Latitude = []
```

These functions will apply the outputs globally to the above variables. Let's take a looks at `BoosterVersion` variable. Before we apply `getBoosterVersion` the list is empty:

In [16]:
```
BoosterVersion
```

Out[16]: []

Now, let's apply `getBoosterVersion` function method to get the booster version

In [17]:
```
# Call getBoosterVersion
getBoosterVersion(data)
```

the list has now been update

In [18]:
```
BoosterVersion[0:5]
```

Out[18]: ['Falcon 1', 'Falcon 1', 'Falcon 1', 'Falcon 1', 'Falcon 9']

In [19]:
```
# Call getLaunchSite
getLaunchSite(data)
```

In [20]:
```
# Call getPayloadData
getPayloadData(data)
```

In [21]:
```
# Call getCoreData
getCoreData(data)
```

Finally lets construct our dataset using the data we have obtained. We we combine the columns into a dictionary.

In [22]:
```
launch_dict = {'FlightNumber': list(data['flight_number']),
'Date': list(data['date']),
'BoosterVersion':BoosterVersion,
'PayloadMass':PayloadMass,
'Orbit':Orbit,
'LaunchSite':LaunchSite,
'Outcome':Outcome,
'Flights':Flights,
'GridFins':GridFins,
'Reused':Reused,
'Legs':Legs,
'LandingPad':LandingPad,
'Block':Block,
'ReusedCount':ReusedCount,
'Serial':Serial,
'Longitude': Longitude,
'Latitude': Latitude}
```

Then, we need to create a Pandas data frame from the dictionary launch_dict.

In [23]:
```
# Create a data from launch_dict
df = pd.DataFrame(launch_dict)
```

Show the summary of the dataframe

In [24]:
```
# Show the head of the dataframe
df.head()
```

Out[24]:

	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins
0	1	2006-03-24	Falcon 1	20.0	LEO	Kwajalein Atoll	None None	1	False
1	2	2007-03-21	Falcon 1	NaN	LEO	Kwajalein Atoll	None None	1	False
2	4	2008-09-28	Falcon 1	165.0	LEO	Kwajalein Atoll	None None	1	False
3	5	2009-07-13	Falcon 1	200.0	LEO	Kwajalein Atoll	None None	1	False
4	6	2010-06-04	Falcon 9	NaN	LEO	CCSFS SLC 40	None None	1	False

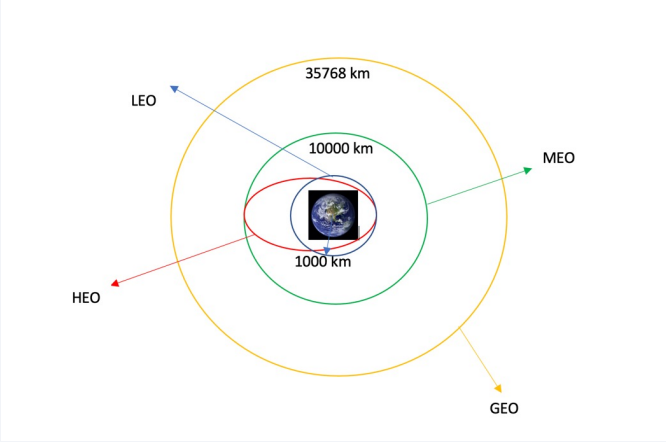
Here is the GitHub link:
https://github.com/gcavdar93/Applied-Data-Science-Capstone/blob/main/1-Data-Collection/jupyter-labs-spacex-data-collection-api.ipynb

Data Wrangling

- The number of launches on each site were determined.
- The number and occurrence of each orbit were calculated.
- The number and occurence of mission outcome per orbit type were calculated.
- A landing outcome label from Outcome column were created.

Here is the GitHub link:

https://github.com/gcavdar93/Applied-Data-Science-Capstone/blob/main/2-Data-Wrangling/labs-jupyter-spacex-Data%20wrangling.ipynb



EDA with Data Visualization

The relationship between Flight number and Launch site, Payload and Launch site, Success rate and Orbit type, Flight number and Orbit type, Payload and Orbit type, and the launch success yearly trend were visualized.

Here is the GitHub link:

https://github.com/gcavdar93/Applied-Data-Science-Capstone/blob/main/4-EDA-For-Visualization/jupyter-labs-eda-dataviz.ipynb

Build an Interactive Map with Folium

- All launch sites on the map were marked with circle and marker.
- The success/failed launches for each site on the map were marked.
- The distances between a launch site to its proximities were calculated.

Here is the GitHub link:

https://github.com/gcavdar93/Applied-Data-Science-Capstone/blob/main/5-Data-Visualization-With-Folium/lab_jupyter_launch_site_location.ipynb

Predictive Analysis (Classification)

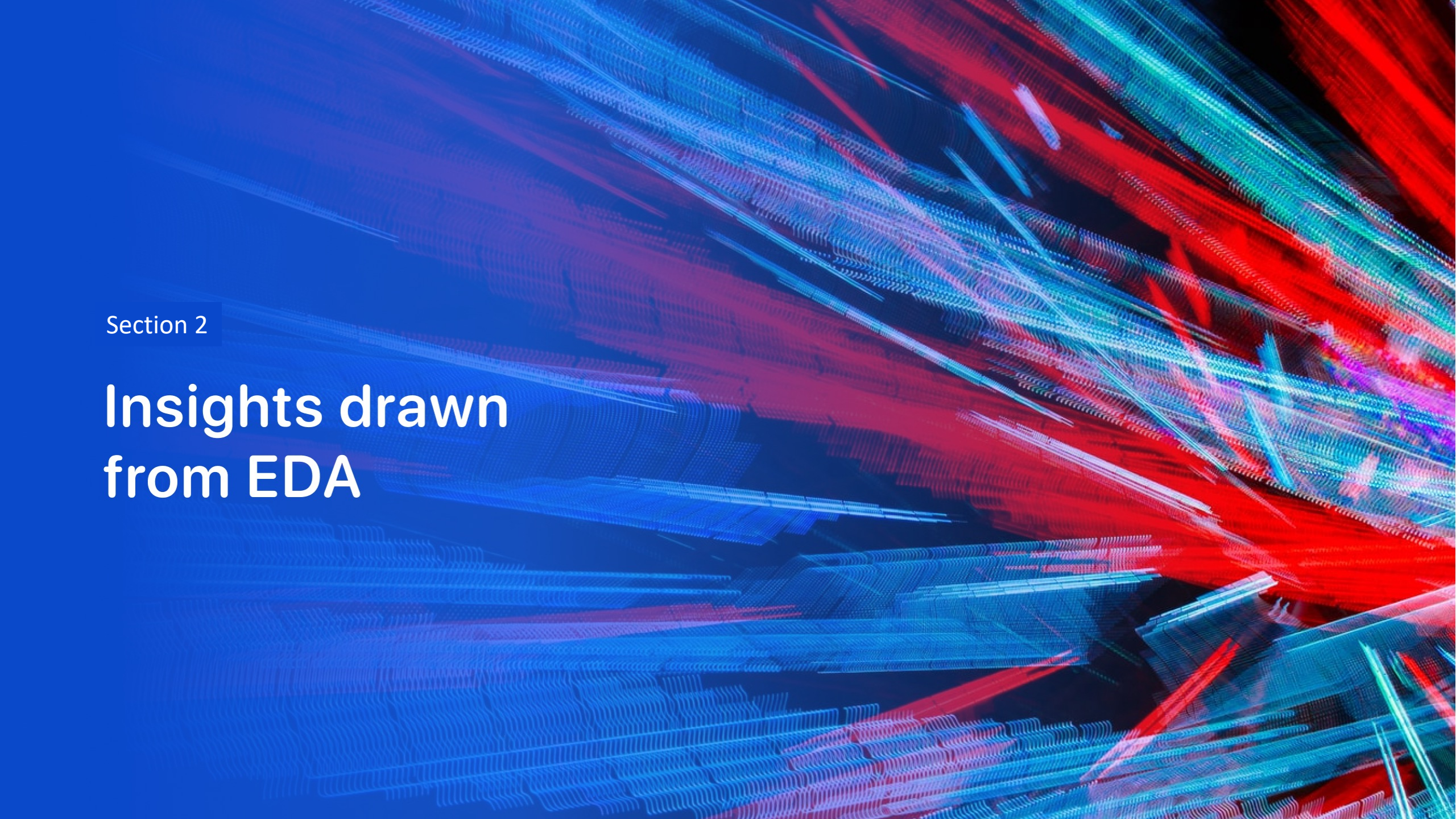
- The data was loaded and splitted into training and testing.
- Machine learning models were builded.
- The best model were selected.

Here is the GitHub link:

https://github.com/gcavdar93/Applied-Data-Science-Capstone/blob/main/6-Machine-Learning-Prediction/SpaceX_Machine%20Learning%20Prediction_Part_5.ipynb

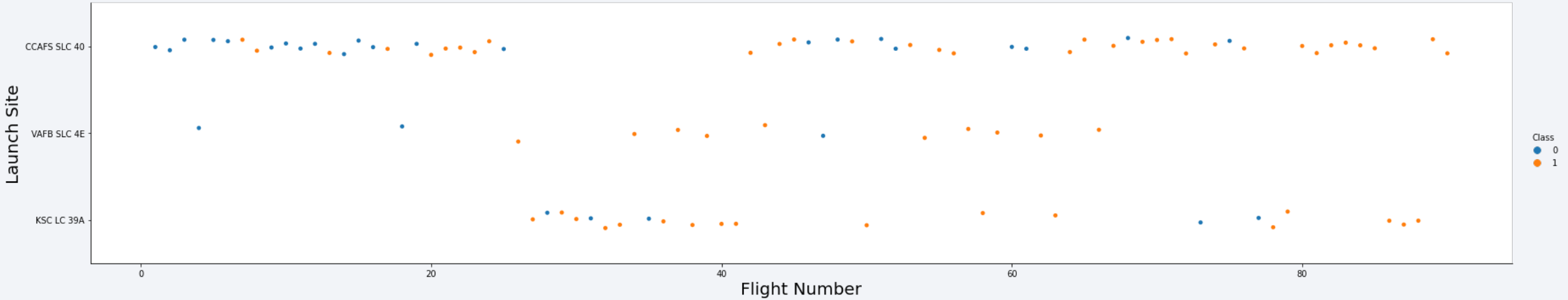
Results

- Exploratory data analysis results
- Interactive analytics demo in screenshots
- Predictive analysis results


Section 2
Insights drawn from EDA

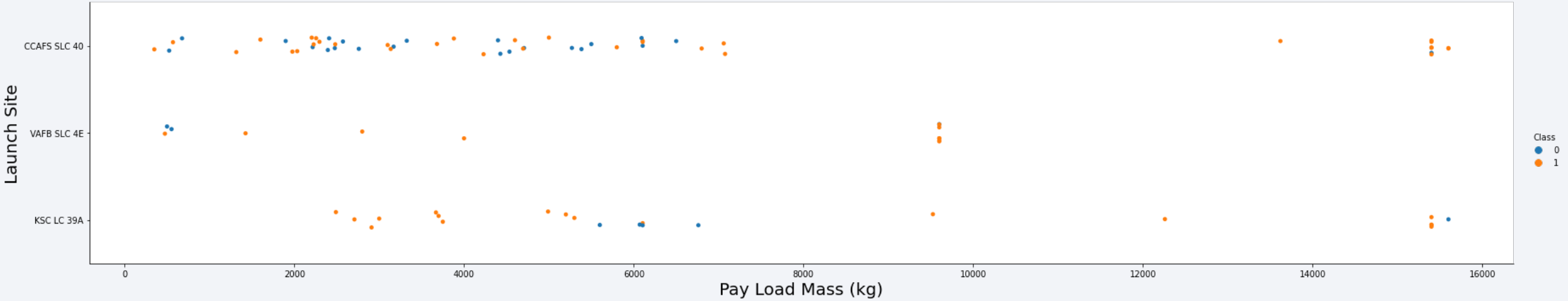
Flight Number vs. Launch Site

The larger the flight number, the greater the success rate.



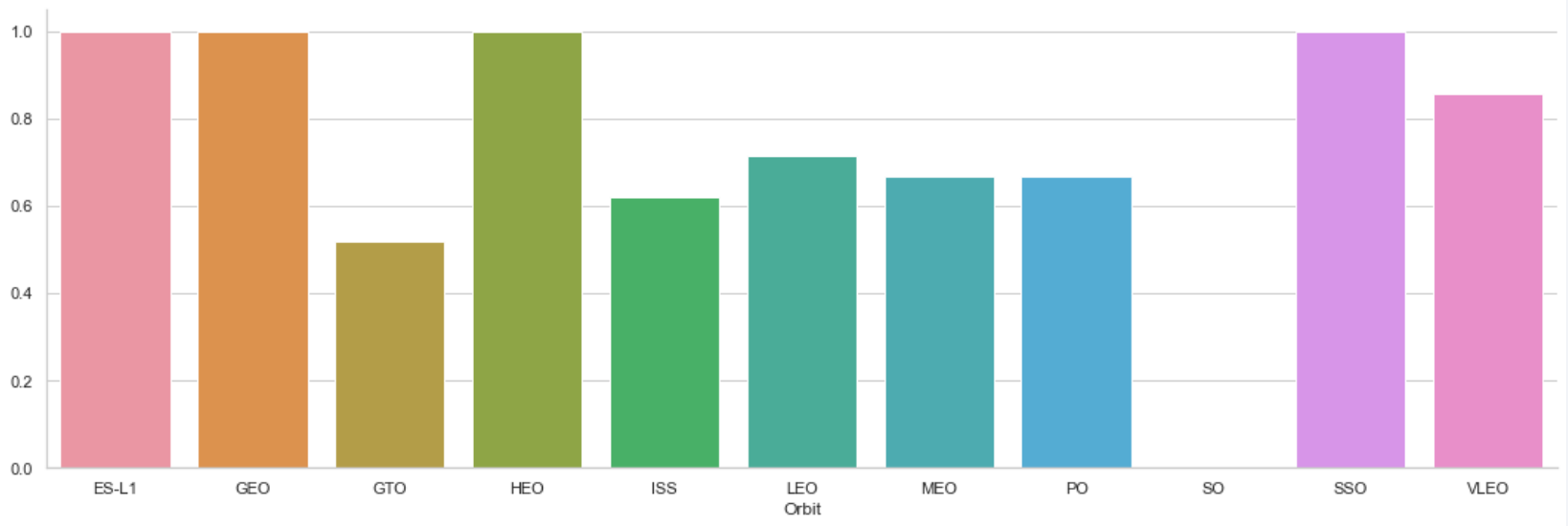
Payload vs. Launch Site

The larger payload mass, the greater the success rate.



Success Rate vs. Orbit Type

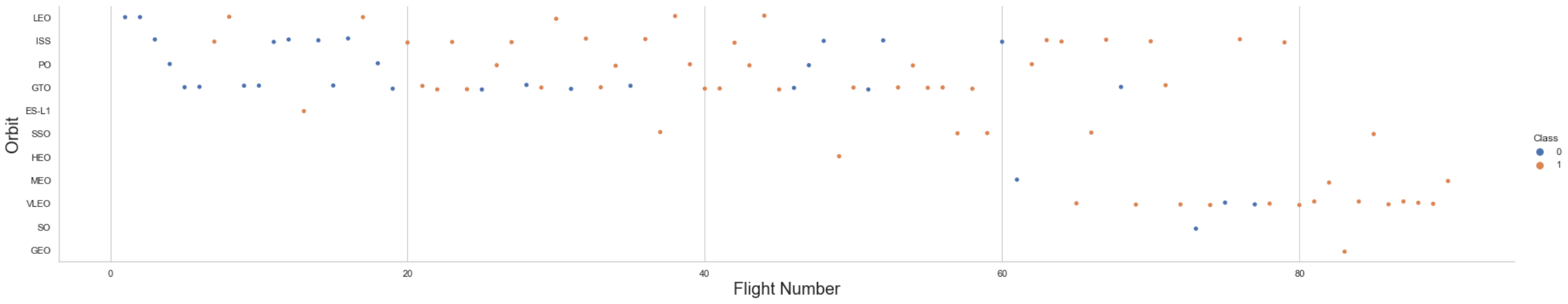
ES-L1, GEO, and HEO have the greatest success rate.



Flight Number vs. Orbit Type

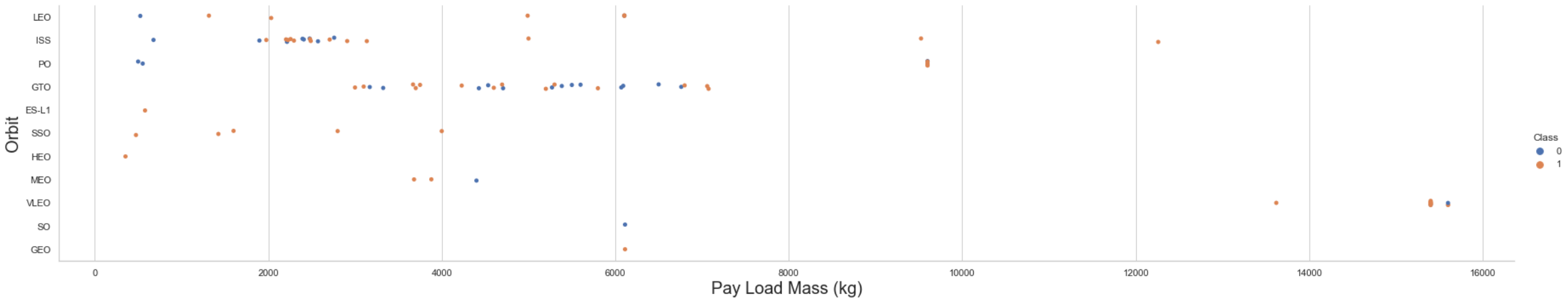
For LEO orbit, the larger the flight number, the greater the success rate.

For GTO orbit, there is no relationship between the flight number and success rate.



Payload vs. Orbit Type

The larger payload mass, the greater the success rate for PO, LEO, ISS orbits.



Launch Success Yearly Trend

From 2013 to 2020, the success rate has increased.

All Launch Site Names

In [3]:
```
# Download and read the `spacex_launch_geo.csv`
spacex_csv_file = wget.download('https://cf-courses-data.s3.us.cloud-object-storage.app
spacex_df=pd.read_csv(spacex_csv_file)
```

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

In [4]:
```
# Select relevant sub-columns: `Launch Site`, `Lat(Latitude)`, `Long(Longitude)`, `clas
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
```

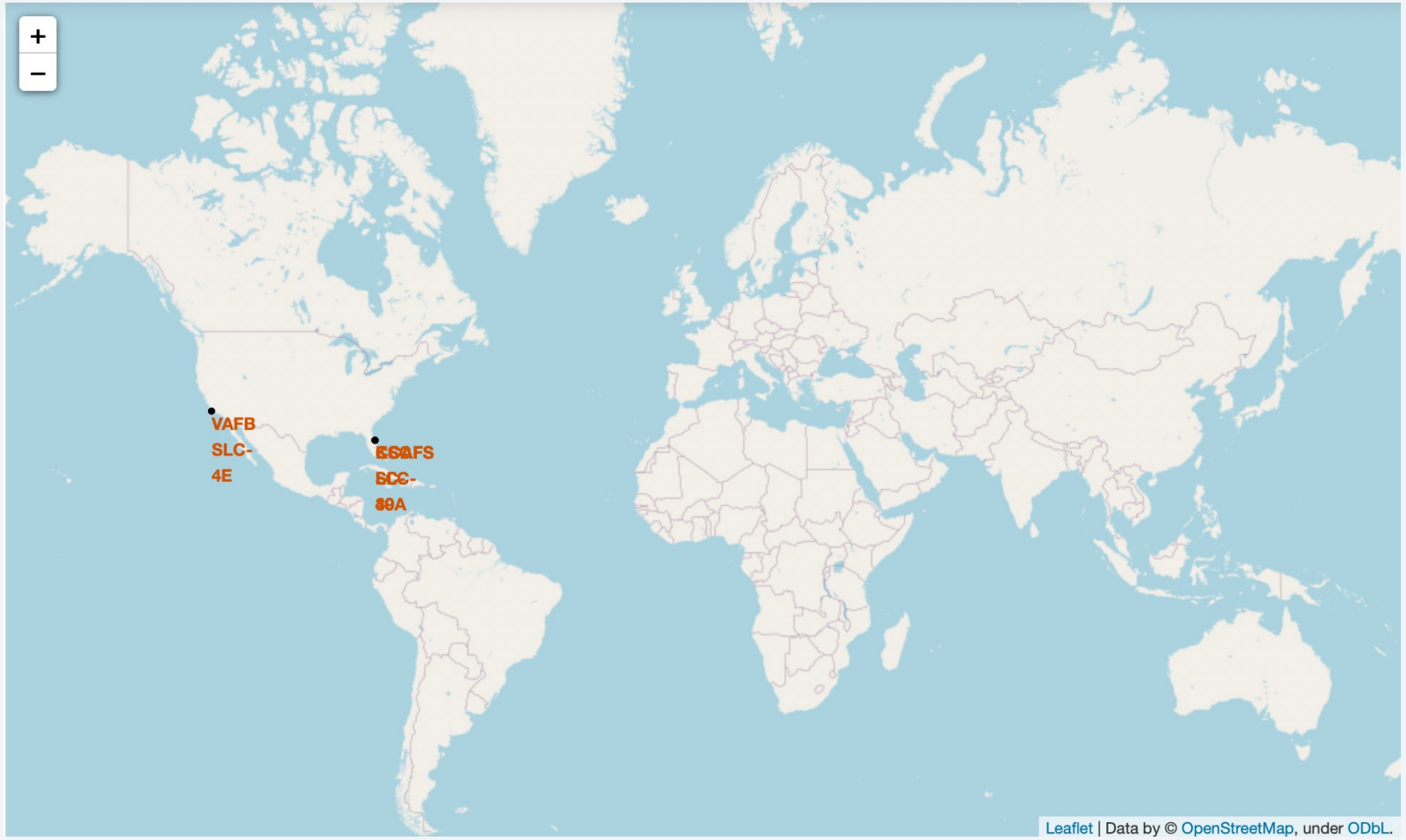
Out[4]:

	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	-120.610746


Section 3
Launch Sites
Proximities Analysis

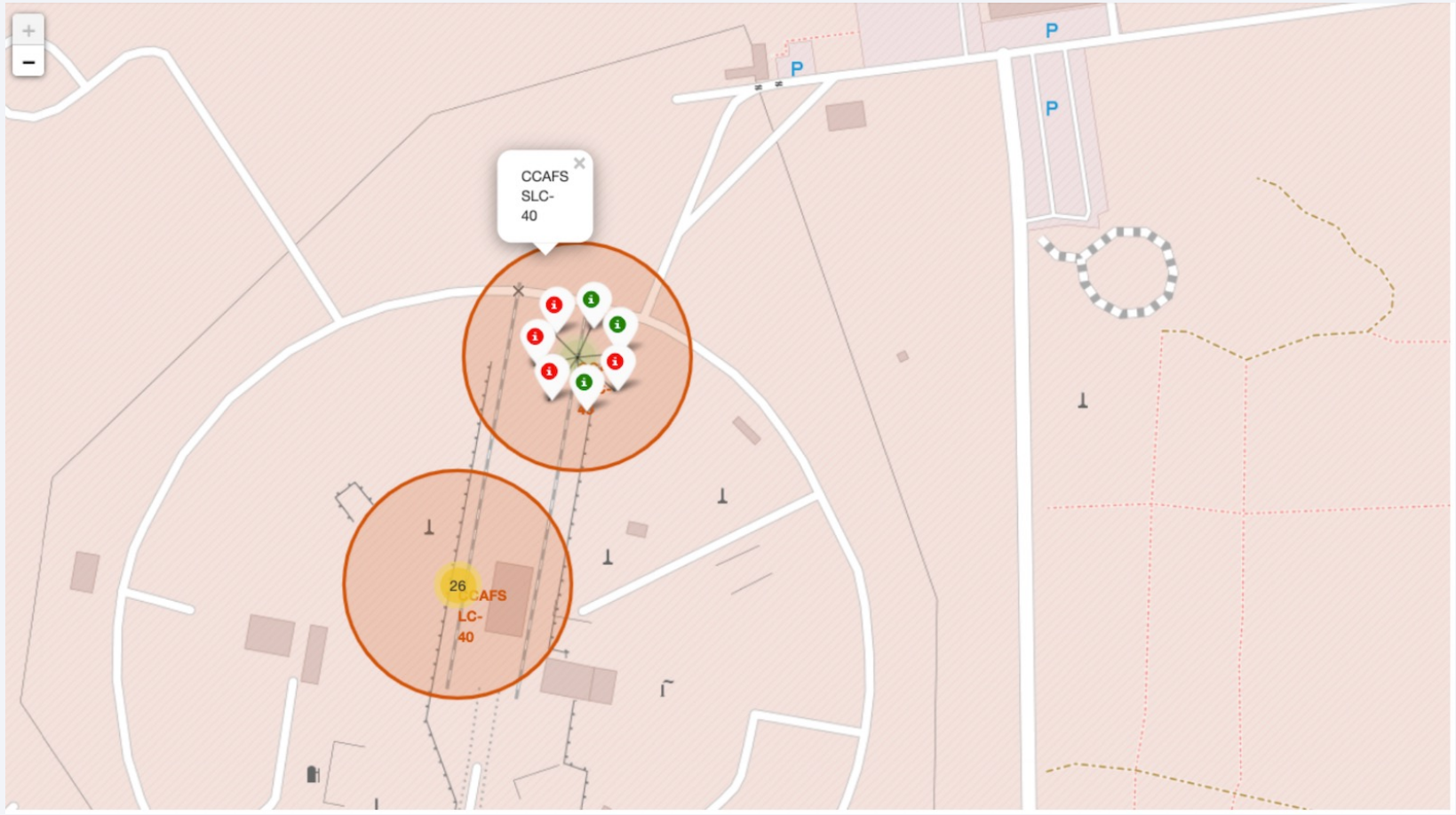
All Launch Sites on the Map

The launch sites are in the United States of America.

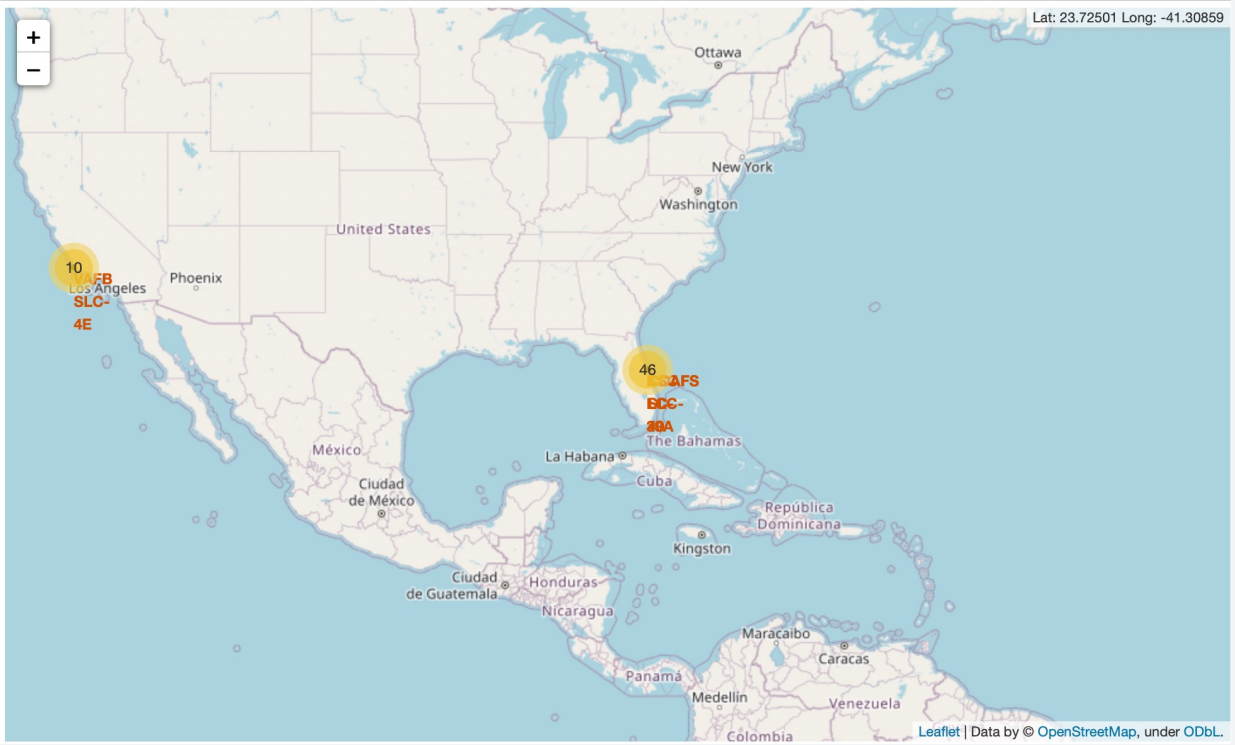


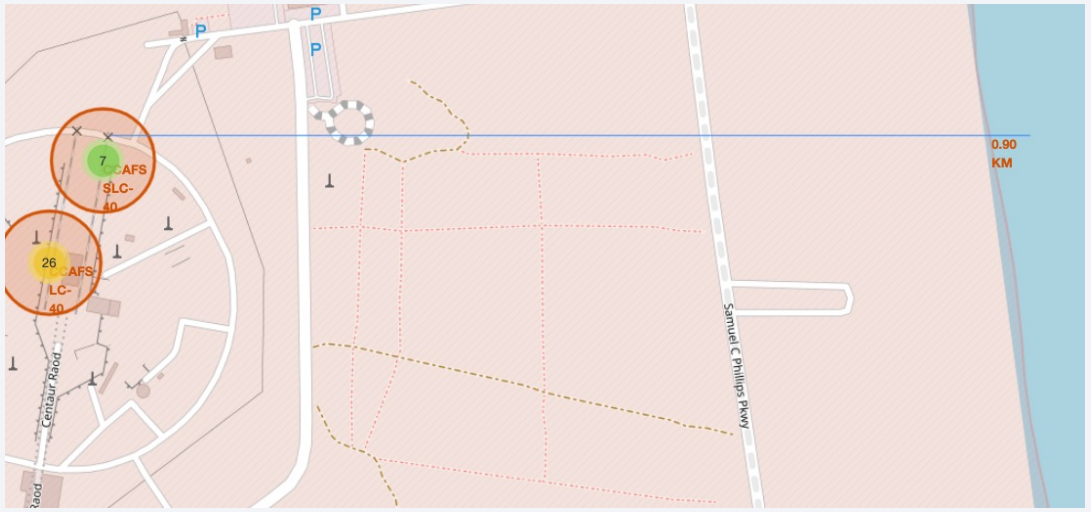
The Successful/Failed Launches for Each Site on the Map

Green markers show the successful launches while red markers show failures.



The Distances between a Launch Site to its Proximities





Section 4

Predictive Analysis (Classification)

Classification Accuracy

In [9]:
```
parameters ={'C':[0.01,0.1,1],
             'penalty':['l2'],
             'solver':['lbfgs']}

lr=LogisticRegression()
grid_search = GridSearchCV(lr, parameters, cv=10)
logreg_cv = grid_search.fit(X_train, Y_train)
```

In [10]:
```
parameters ={"C":[0.01,0.1,1],'penalty':['l2'], 'solver':['lbfgs']}# l1 lasso l2 ridge
lr=LogisticRegression()
```

We output the `GridSearchCV` object for logistic regression. We display the best parameters using the data attribute `best_params_` and the accuracy on the validation data using the data attribute `best_score_`.

In [11]:
```
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
```

In [19]:
```
parameters = {'criterion': ['gini', 'entropy'],
     'splitter': ['best', 'random'],
     'max_depth': [2*n for n in range(1,10)],
     'max_features': ['auto', 'sqrt'],
     'min_samples_leaf': [1, 2, 4],
     'min_samples_split': [2, 5, 10]}

tree = DecisionTreeClassifier()
```

In [20]:
```
grid_search = GridSearchCV(tree, parameters, cv=10)
tree_cv = grid_search.fit(X_train, Y_train)
```

In [21]:
```
print("tuned hpyerparameters :(best parameters) ",tree_cv.best_params_)
print("accuracy :",tree_cv.best_score_)
```

```
tuned hpyerparameters :(best parameters)  {'criterion': 'entropy', 'max_depth': 2, 'max_features': 'sqrt', 'min_sam
ples_leaf': 1, 'min_samples_split': 10, 'splitter': 'best'}
accuracy : 0.8892857142857145
```

In [14]:
```
parameters = {'kernel':('linear', 'rbf','poly','rbf', 'sigmoid'),
              'C': np.logspace(-3, 3, 5),
              'gamma':np.logspace(-3, 3, 5)}
svm = SVC()
```

In [15]:
```
grid_search = GridSearchCV(svm, parameters, cv=10)
svm_cv = grid_search.fit(X_train, Y_train)
```

In [16]:
```
print("tuned hpyerparameters :(best parameters) ",svm_cv.best_params_)
print("accuracy :",svm_cv.best_score_)
```

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

In [24]:
```
parameters = {'n_neighbors': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
              'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute'],
              'p': [1,2]}

KNN = KNeighborsClassifier()
```

In [25]:
```
grid_search = GridSearchCV(KNN, parameters, cv=10)
knn_cv = grid_search.fit(X_train, Y_train)
```

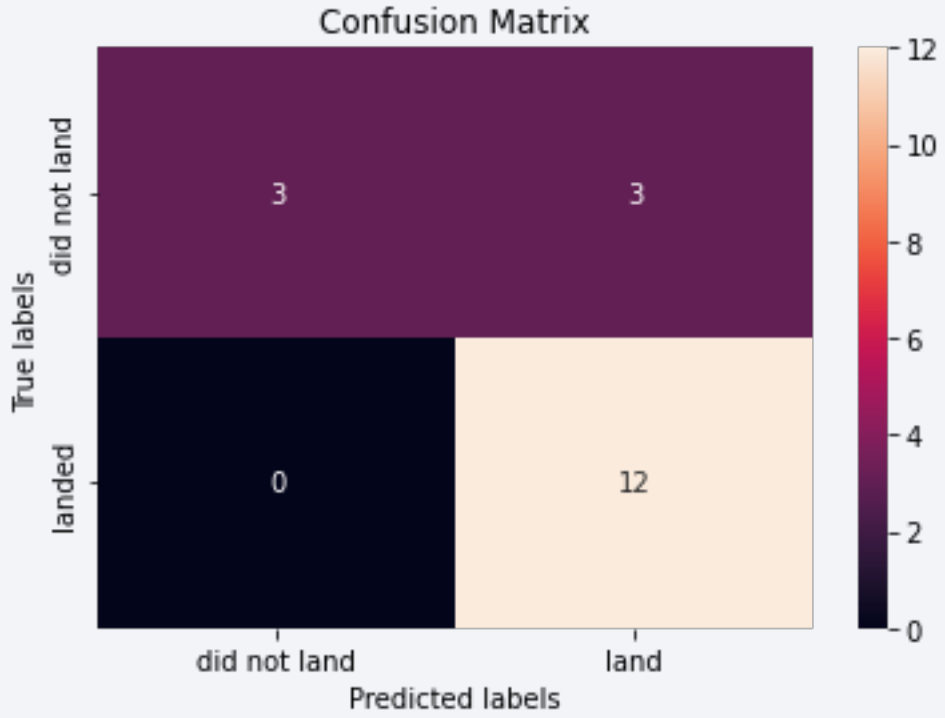
In [26]:
```
print("tuned hpyerparameters :(best parameters) ",knn_cv.best_params_)
print("accuracy :",knn_cv.best_score_)
```

```
tuned hpyerparameters :(best parameters)  {'algorithm': 'auto', 'n_neighbors': 10, 'p': 1}
accuracy : 0.8482142857142858
```

The decision tree classifier has the highest accuracy.

Confusion Matrix

The Confusion Matrix of Decision Tree classifier, which has the highest accuracy



Conclusions

- The larger the flight number, the greater the success rate.
- The larger payload mass, the greater the success rate.
- ES-L1, GEO, and HEO have the greatest success rate.
- For LEO orbit, the larger the flight number, the greater the success rate.
- For GTO orbit, there is no relationship between the flight number and success rate.
- The larger payload mass, the greater the success rate for PO, LEO, ISS orbits.
- From 2013 to 2020, the success rate has increased.
- The decision tree classifier has the highest accuracy.

Thank you!