SpaceX

Winning Space Race with Data Science

Cintia Campos
10.Dec.2021

GitHub URL:

https://github.com/sharkgurl/IBM_Data_Science/tree/master/applied_data_science_capstone

OUTLINE

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

photo source: PowerPoint Design Ideas

EXECUTIVE SUMMARY

Summary of methodologies

The SpaceX project uses exploratory and predictive data analyses with Folium and Plotly Dash data visualizations to gather insight into the Falcon 9 first stage successful landing rates.

Summary of all results

This model predicted that when Falcon 9 first stage lands successfully, we can determine the cost of a launch

INTRODUCTION

Project background and context

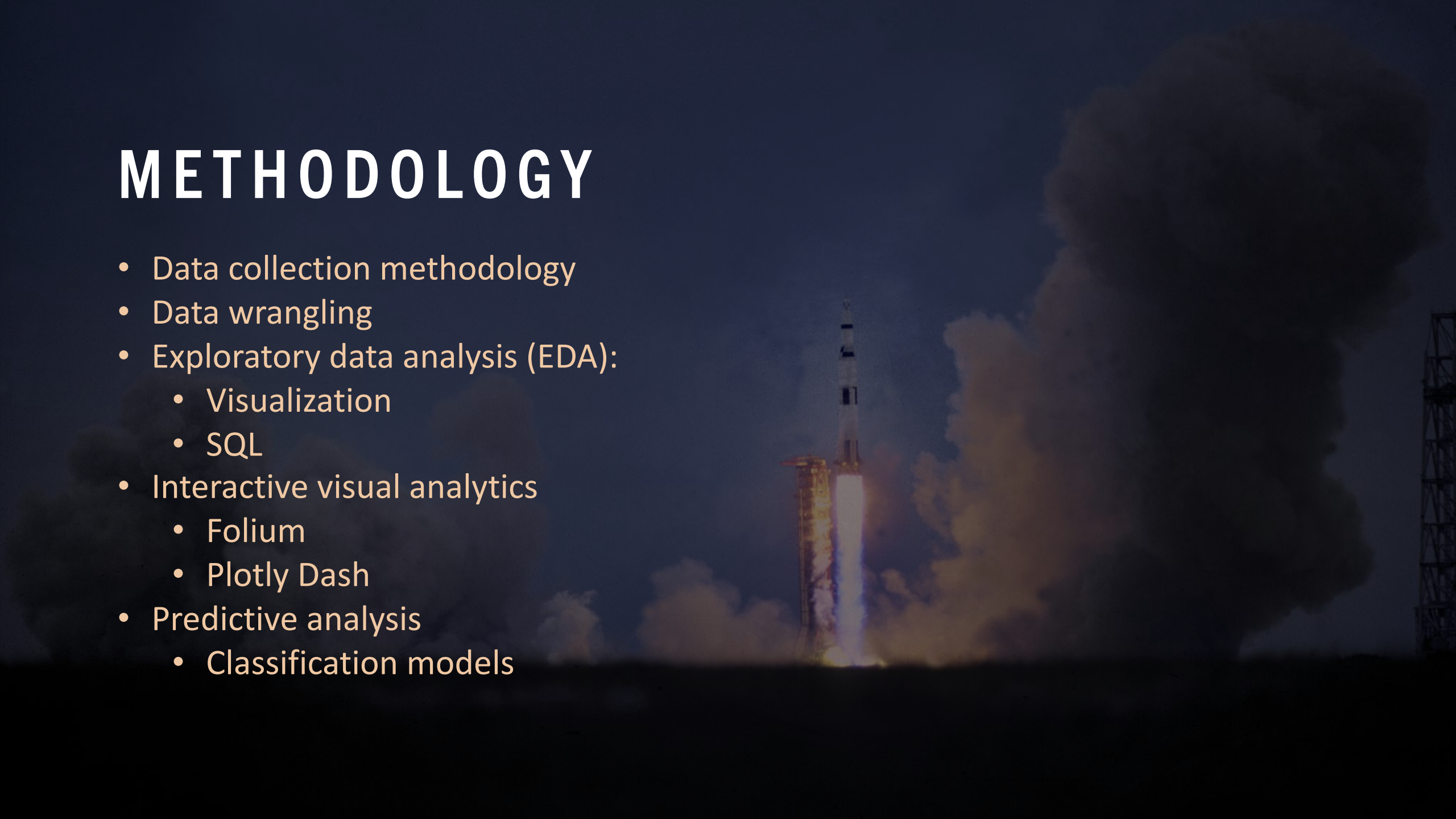
SpaceX advertises that Falcon 9 rocket launches on its website with a cost of $62M while other providers cost upward of $165M each. Savings are driven by the first stage reuse by SpaceX.

Problem Statement

By predicting whether Falcon 9 first stage lands successfully, we can determine the cost of a launch and use that information when an alternate company bids against SpaceX for a rocket launch.

METHODOLOGY

- Data collection methodology
- Data wrangling
- Exploratory data analysis (EDA):
  - Visualization
  - SQL
- Interactive visual analytics
  - Folium
  - Plotly Dash
- Predictive analysis
  - Classification models



DATA COLLECTION

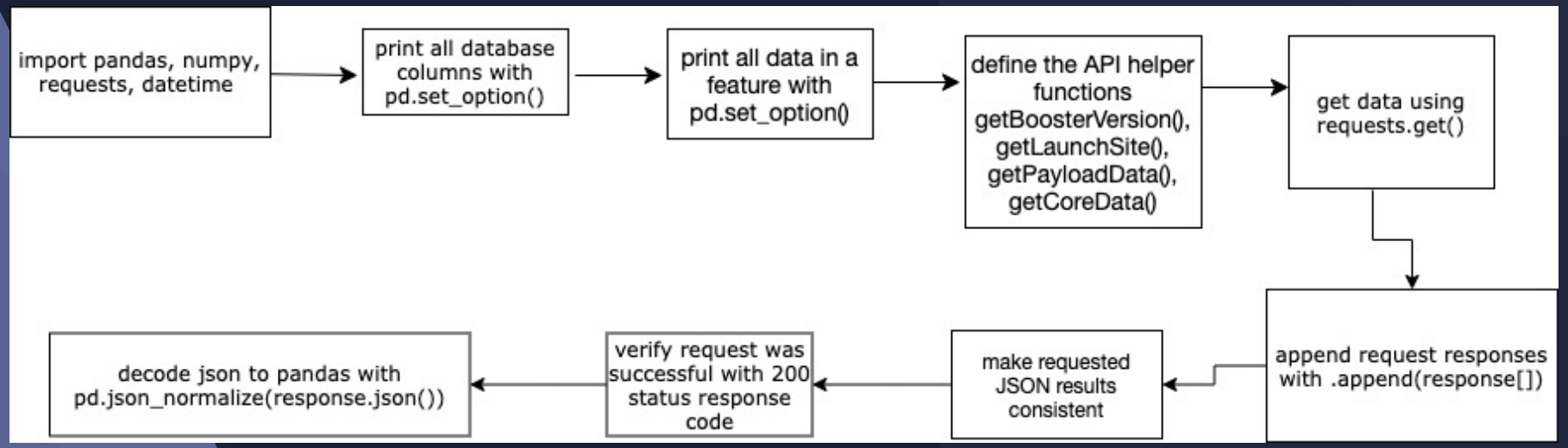
GitHub URL:

https://github.com/sharkgurl/IBM_Data_Science/blob/master/applied_data_science_capstone/SpaceX%20Falcon%209%20first%20stage%20Landing%20Prediction.ipynb

Data **Collection**

Datasets were collected via REST API requests, appended to a list, and normalized into a flat table.

- requests.get(url)
- response.json()
- pd.json_normalize(response.json())



From the `rocket` column we would like to learn the booster name.

In [2]:
```
# Takes the dataset and uses the rocket column to call the API and append the data to the list
def getBoosterVersion(data):
    for x in data['rocket']:
        response = requests.get("https://api.spacexdata.com/v4/rockets/"+str(x)).json()
        BoosterVersion.append(response['name'])
```

From the `launchpad` we would like to know the name of the launch site being used, the logitude, and the latitude.

In [3]:
```
# Takes the dataset and uses the launchpad column to call the API and append the data to the list
def getLaunchSite(data):
    for x in data['launchpad']:
        response = requests.get("https://api.spacexdata.com/v4/launchpads/"+str(x)).json()
        Longitude.append(response['longitude'])
        Latitude.append(response['latitude'])
        LaunchSite.append(response['name'])
```

From the `payload` we would like to learn the mass of the payload and the orbit that it is going to.

In [4]:
```
# Takes the dataset and uses the payloads column to call the API and append the data to the lists
def getPayloadData(data):
    for load in data['payloads']:
        response = requests.get("https://api.spacexdata.com/v4/payloads/"+load).json()
        PayloadMass.append(response['mass_kg'])
        Orbit.append(response['orbit'])
```

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, wheter the core is reused, wheter 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.

In [5]:
```
# Takes the dataset and uses the cores column to call the API and append the data to the lists
def getCoreData(data):
    for core in data['cores']:
            if core['core'] != None:
                response = requests.get("https://api.spacexdata.com/v4/cores/"+core['core']).json()
                Block.append(response['block'])
                ReusedCount.append(response['reuse_count'])
                Serial.append(response['serial'])
            else:
                Block.append(None)
                ReusedCount.append(None)
                Serial.append(None)
            Outcome.append(str(core['landing_success'])+' '+str(core['landing_type']))
            Flights.append(core['flight'])
            GridFins.append(core['gridfins'])
            Reused.append(core['reused'])
            Legs.append(core['legs'])
            LandingPad.append(core['landpad'])
```

Now let's start requesting rocket launch data from SpaceX API with the following URL:

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

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

SpaceX Rest API Calls

SpaceX
REST API

GitHub URL:

https://github.com/sharkgurl/IBM_Data_Science/blob/master/applied_data_science_capstone/SpaceX%20Falcon%209%20first%20stage%20Landing%20Prediction.ipynb

DATA COLLECTION - SCRAPING

Web scraping used Python BeautifulSoup to webscrape html tables, parse their data and convert them to a pandas dataframe.

GitHub URL:

https://github.com/sharkgurl/IBM_Data_Science/blob/master/applied_data_science_capstone/SpaceX%20Falcon%209%20first%20stage%20Landing%20Prediction.ipynb

DATA WRANGLING

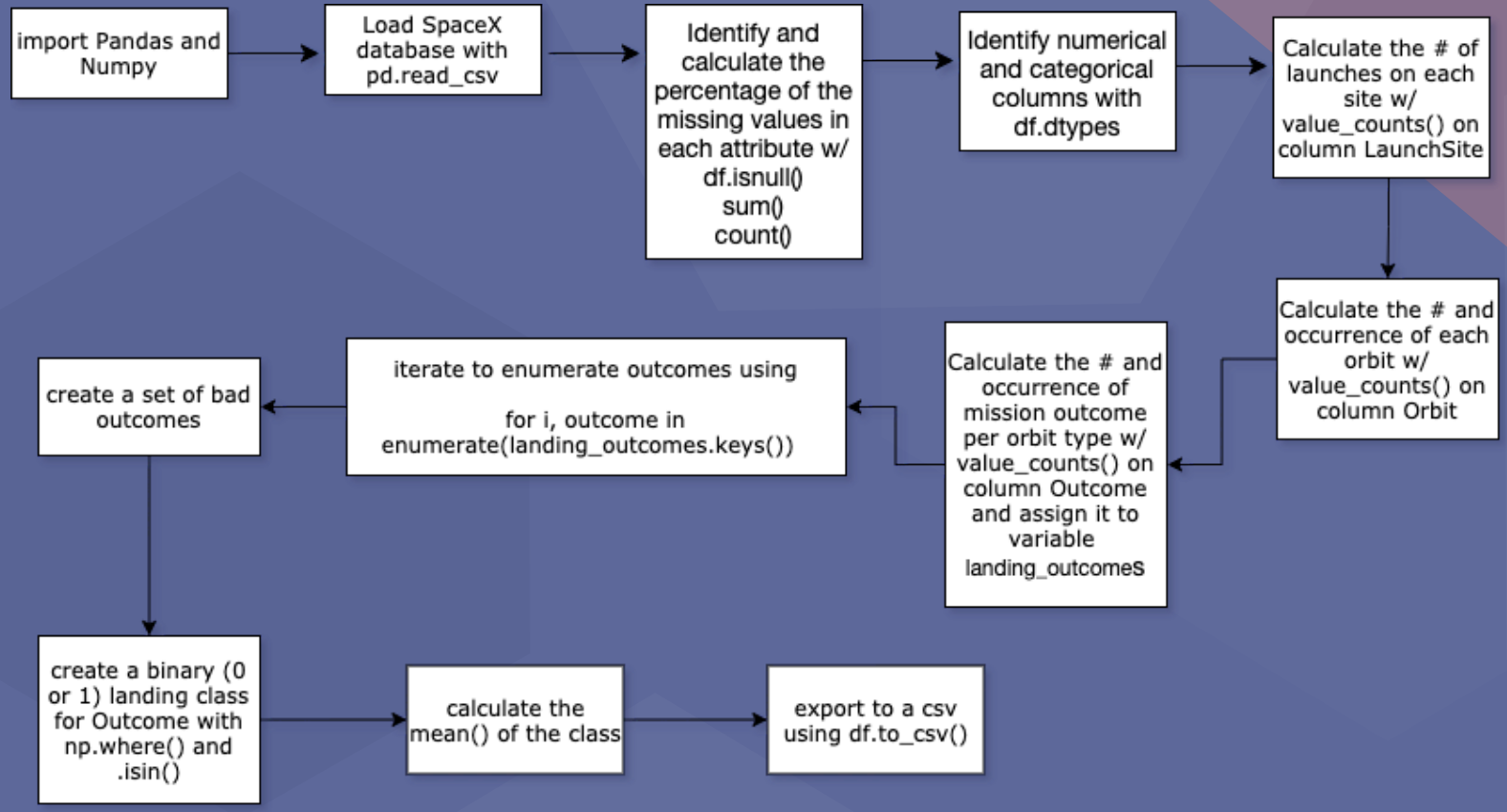
GitHub URL:

https://github.com/sharkgurl/IBM_Data_Science/blob/master/applied_data_science_capstone/SpaceX_Data_Wrangling.ipynb

How data was processed

This process converted landing outcomes (True Ocean, False Ocean, True RTLS, False RTLS, True ASDS, False ASDS) into training labels where “1” represented a successful booster landing, and “0” represented an unsuccessful landing.

flowchart



Data **Wrangling**

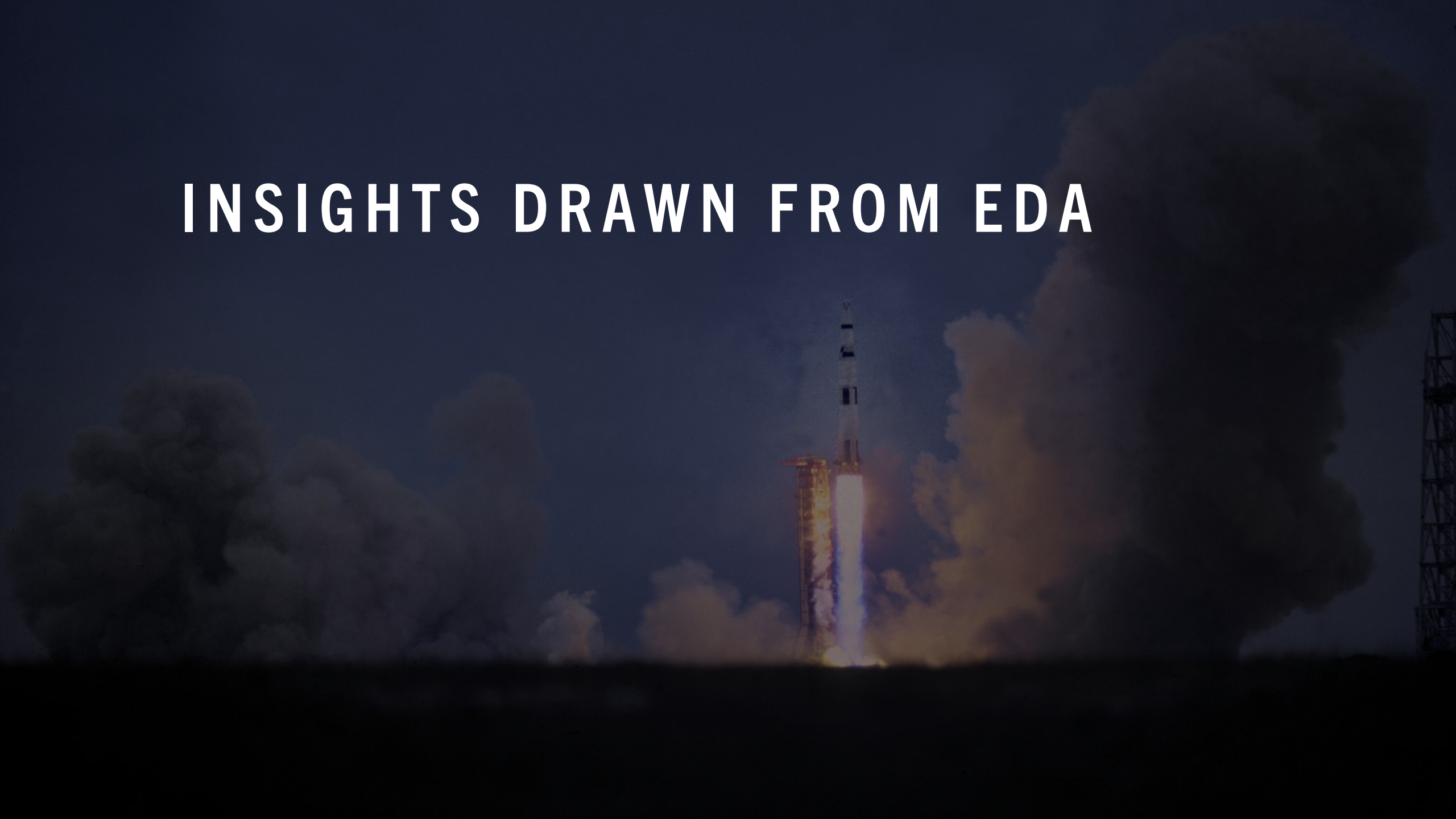
RESULTS

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

GitHub URL:

https://github.com/sharkgurl/IBM_Data_Science/blob/master/applied_data_science_capstone/SpaceX%20EDA%20with%20Visualization.ipynb

INSIGHTS DRAWN FROM EDA



EDA with Data Visualization

- The pandas, numpy, matplotlib.pyplot, and seaborn libraries were used in order to create various scatterplots, bar charts, and line charts.

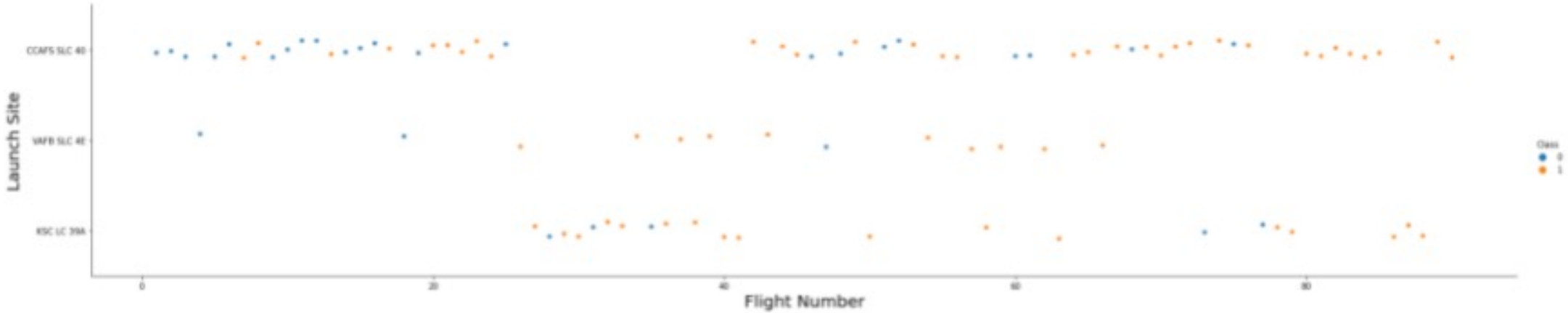
https://github.com/sharkgurl/IBM_Data_Science/blob/master/applied_data_science_capstone/SpaceX%20EDA%20with%20Visualization.ipynb

Flight Number vs. Launch Site

This scatterplot shows how FlightNumber and Launch Site affect the launch outcome.

As Flight number increases, the first stage is more lightly to land successfully.

In [10]:
```
# 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

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()
```

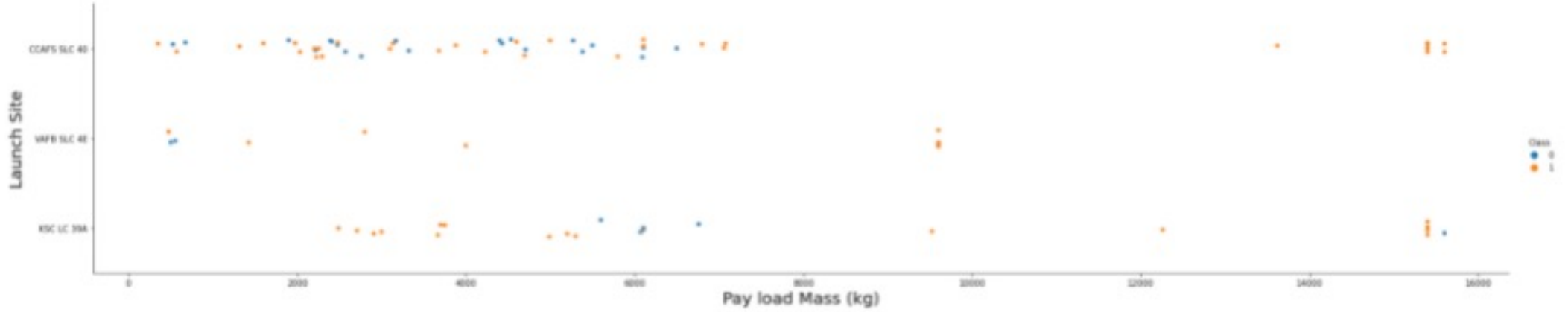


Payload vs. Launch Site

This scatterplot shows how Payload and Launch Site affect the launch outcome.

Payloads that approach max payload tended to launch from CCAFS SLC 40 & KSC LC 39A. Payloads less than 8000 kg tended to fail at a higher rate when launched from CCAFS SLC 40.

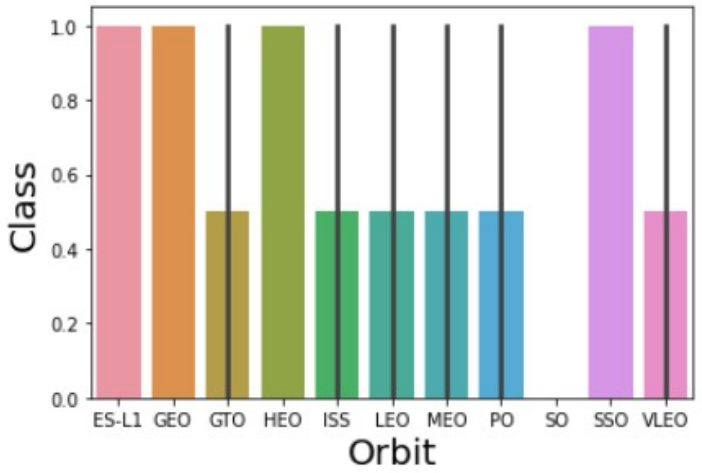
In [11]:
```
# Plot a scatter point chart with x axis to be Pay Load Mass (kg) and y axis to be the launch site, and hue to be the class value

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



Success Rate vs. Orbit Type

Orbits with high success rate are ES-L1, GEO, HEO, and SSO.

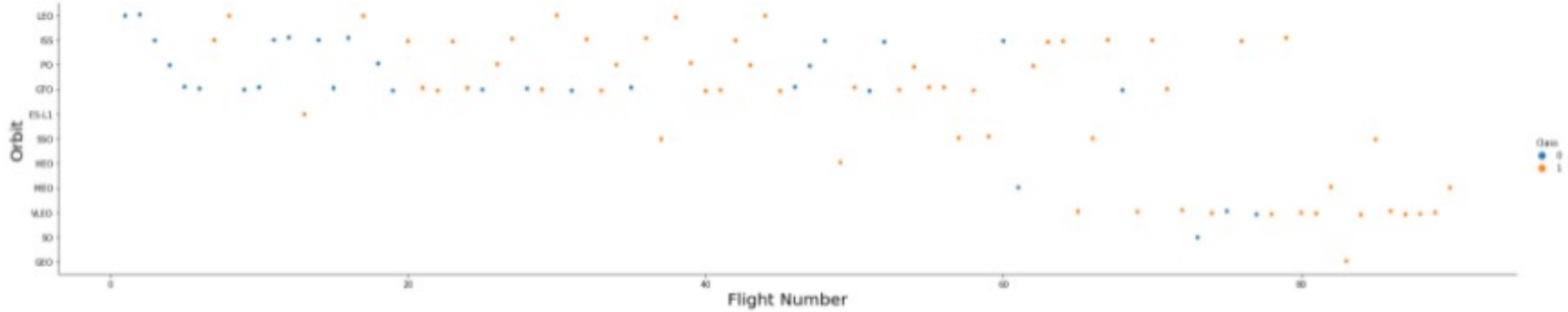
```
[13]: # HINT use groupby method on Orbit column and get the mean of Class column
t = df.groupby(['Orbit', 'Class'])['Class'].agg(['mean']).reset_index()
sns.barplot(y="Class", x="Orbit", data=t)

plt.xlabel("Orbit",fontsize=20)
plt.ylabel("Class",fontsize=20)
plt.show()
```



Flight Number vs. Orbit Type

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

[12]:
```
# 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()
```

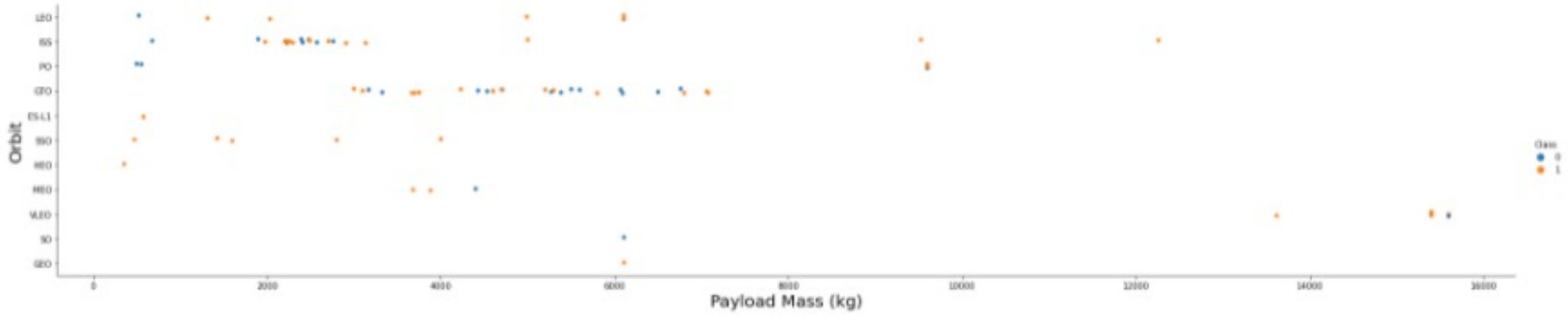


Payload vs. Orbit Type

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

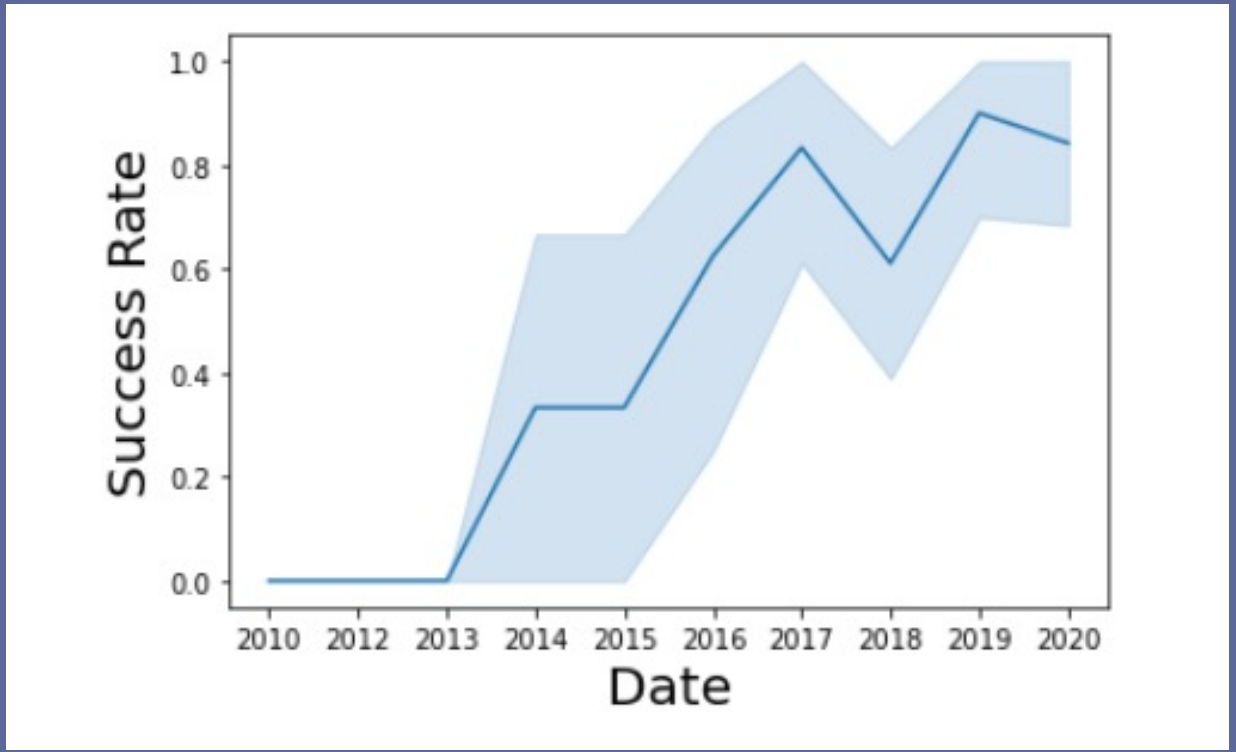
However for GTO we cannot distinguish this well as both positive landing rate and negative landing (unsuccessful mission) are both there.

```
n [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="Orbit", x="PayloadMass", hue="Class", data=df, aspect = 5)
        plt.xlabel("Payload Mass (kg)",fontsize=20)
        plt.ylabel("Orbit",fontsize=20)
        plt.show()
```



Launch Success Yearly Trend

The success rate has been increasing overall since 2013 despite the drop in 2018.



EDA with SQL

- In order to query the database and retrieve the desired data, sqlalchemy, ibm_db_sa, and ipython-sql were installed.

- Queries included:

  - SELECT
  - DISTINCT
  - WHERE
  - LIKE
  - %
  - LIMIT
  - SUM()
  - AVG()
  - MAX()
  - MIN()
  - GROUP BY
  - HAVING
  - =, <, >
  - SUBQUERIES
  - EXTRACT
  - COUNT
  - BETWEEN
  - DESC
  - ORDER BY

All Launch Site Names

The SELECT DISTINCT query provided the unique launch site names eliminating all duplicates

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

```
[4]: %sql SELECT DISTINCT(LAUNCH_SITE) FROM SPACEXDATASET;
```

```
 * ibm_db_sa://wcy90736:***@ba99a9e6-d59e-4883-8fc0-d6a8c9f7a08f.c1ogj3sd0tgtu0lqde00.databas
es.appdomain.cloud:31321/bludb
Done.
```

t[4]:

launch_site
CCAFS LC-40
CCAFS SLC-40
KSC LC-39A
VAFB SLC-4E

Launch Site Names Begin with 'CCA'

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

In [5]:
```
%sql SELECT * FROM SPACEXDATASET WHERE LAUNCH_SITE LIKE '%CCA%' LIMIT 5;
```

```
 * ibm_db_sa://wcy90736:***@ba99a9e6-d59e-4883-8fc0-d6a8c9f7a08f.clogj3sd0tgtu0lqde00.databases.appdomain.cloud:31321/bludb
Done.
```

Out[5]:

DATE	time__utc_	booster_version	launch_site	payload	payload_mass__kg_	orbit	customer	mission_outcome
2010-06-04	18:45:00	F9 v1.0 B0003	CCAFS LC-40	Dragon Spacecraft Qualification Unit	0	LEO	SpaceX	Success
2010-12-08	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
2012-05-22	07:44:00	F9 v1.0 B0005	CCAFS LC-40	Dragon demo flight C2	525	LEO (ISS)	NASA (COTS)	Success
2012-10-08	00:35:00	F9 v1.0 B0006	CCAFS LC-40	SpaceX CRS-1	500	LEO (ISS)	NASA (CRS)	Success
2013-03-01	15:10:00	F9 v1.0 B0007	CCAFS LC-40	SpaceX CRS-2	677	LEO (ISS)	NASA (CRS)	Success

The WHERE statement was used with SELECT to define the condition for the launch names which begun with ‘CCA’.

Total Payload Mass

The SUM() function was used to display the total payload mass carried by boosters and the HAVING clause filtered for ‘NASA’ customers.

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

In [6]:
```
%sql SELECT CUSTOMER, SUM(PAYLOAD_MASS__KG_) as "Total Payload Mass" FROM SPACEXDATASET GROUP
BY CUSTOMER HAVING CUSTOMER = 'NASA (CRS)'
```

```
 * ibm_db_sa://wcy90736:***@ba99a9e6-d59e-4883-8fc0-d6a8c9f7a08f.c1ogj3sd0tgtu0lqde00.databas
es.appdomain.cloud:31321/bludb
Done.
```

Out[6]:

customer	Total Payload Mass
NASA (CRS)	45596

Average Payload Mass by F9 v1.1

The AVG() function was used to display the average payload mass carried by booster version F9v1.1, GROUP BY grouped them by booster version, and the HAVING clause filtered for ‘F9v1.1’.

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

```
In [7]: %sql SELECT BOOSTER_VERSION, AVG(PAYLOAD_MASS__KG_) AS "Average Payload Mass" FROM SPACEXDATAS
        ET GROUP BY BOOSTER_VERSION HAVING BOOSTER_VERSION = 'F9 v1.1'

         * ibm_db_sa://wcy90736:***@ba99a9e6-d59e-4883-8fc0-d6a8c9f7a08f.c1ogj3sd0tgtu0lqde00.databas
        es.appdomain.cloud:31321/bludb
        Done.
```

Out[7]:

booster_version	Average Payload Mass
F9 v1.1	2928

First Successful Ground Landing Date

In order to get the date of the first successful outcome, the function MIN(DATE) … AS… was used. Results were grouped with GROUP BY and filtered with HAVING for successful landing outcome.

List the date when the first successful landing outcome in ground pad was acheived.

Hint:Use min function

In [8]:
```
%sql SELECT LANDING__OUTCOME, MIN(DATE) AS "First Successful Landing" FROM SPACEXDATASET GROUP
BY LANDING__OUTCOME HAVING LANDING__OUTCOME = 'Success (ground pad)';
```

```
 * ibm_db_sa://wcy90736:***@ba99a9e6-d59e-4883-8fc0-d6a8c9f7a08f.c1ogj3sd0tgtu0lqde00.databas
es.appdomain.cloud:31321/bludb
Done.
```

Out[8]:

landing__outcome	First Successful Landing
Success (ground pad)	2015-12-22

Successful Drone Ship Landing with Payload greater than 4000 but less than 6000

> AND < were used to filter payload mass

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

```
[19]: %sql select BOOSTER_VERSION from SPACEXDATASET where LANDING__OUTCOME = 'Success (drone ship)'
AND PAYLOAD_MASS__KG_ > 400 AND PAYLOAD_MASS__KG_ < 6000;
```

```
 * ibm_db_sa://wcy90736:***@ba99a9e6-d59e-4883-8fc0-d6a8c9f7a08f.c1ogj3sd0tgtu0lqde00.databas
es.appdomain.cloud:31321/bludb
Done.
```

ut[19]:

booster_version
F9 FT B1021.1
F9 FT B1022
F9 FT B1023.1
F9 FT B1026
F9 FT B1021.2
F9 FT B1029.2
F9 FT B1038.1
F9 FT B1031.2
F9 B4 B1042.1
F9 B5 B1046.1

Total Number of Successful and Failure Mission Outcomes

The SUM() function is used to get the totals for successful and failure outcomes; LIKE is used to match a string; % is a wildcard ; AND combines a 2nd condition.

List the total number of successful and failure mission outcomes

In [27]:
```
%sql SELECT sum(MISSION_OUTCOME LIKE '%Success%') as Success_Outcomes, sum(MISSION_OUTCOME LIK
E '%Failure%') as Failure_Outcomes FROM SPACEXDATASET;
```

```
 * ibm_db_sa://wcy90736:***@ba99a9e6-d59e-4883-8fc0-d6a8c9f7a08f.c1ogj3sd0tgtu0lqde00.databas
es.appdomain.cloud:31321/bludb
Done.
```

Out[27]:

success_outcomes	failure_outcomes
100	1

Boosters Carried Maximum Payload

The SELECT subquery nested in the SELECT statement with the MAX() function filters the booster_version and shows only the ones with the max payload mass

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

```
]: %sql SELECT DISTINCT BOOSTER_VERSION, PAYLOAD_MASS__KG_ FROM SPACEXDATASET WHERE PAYLOAD_MASS__KG_ = (SELECT MAX(PAYLOAD_MASS__KG_) FROM SPACEXDATASET);
 * ibm_db_sa://wcy90736:***@ba99a9e6-d59e-4883-8fc0-d6a8c9f7a08f.c1ogj3sd0tgtu0lqde00.databases.appdomain.cloud:31321/bludb
Done.
```

[21]:

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

2015 Launch Records

The SELECT WHERE IN uses the IN operator to find the failed landing_outcomes in drone_ship. ‘AND’ combines the 2015 condition to the query results.

List the failed landing_outcomes in drone ship, their booster versions, and launch site names for in year 2015

```
h [75]: %sql SELECT LANDING__OUTCOME, BOOSTER_VERSION, LAUNCH_SITE, DATE FROM SPACEXDATASET WHERE LAND
ING__OUTCOME IN 'Failure (drone ship)' AND EXTRACT(YEAR FROM DATE) = '2015';
```

```
 * ibm_db_sa://wcy90736:***@ba99a9e6-d59e-4883-8fc0-d6a8c9f7a08f.c1ogj3sd0tgtu0lqde00.databas
es.appdomain.cloud:31321/bludb
Done.
```

ut[75]:

landing__outcome	booster_version	launch_site	DATE
Failure (drone ship)	F9 v1.1 B1012	CCAFS LC-40	2015-01-10
Failure (drone ship)	F9 v1.1 B1015	CCAFS LC-40	2015-04-14

Rank Landing Outcomes Between 2010-06-04 and 2017-03-20

COUNT the landing outcomes for a specific date range using WHERE, BETWEEN, AND, and group results with GROUP BY. To rank them, ORDER BY, and list in descending order with DESC.

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

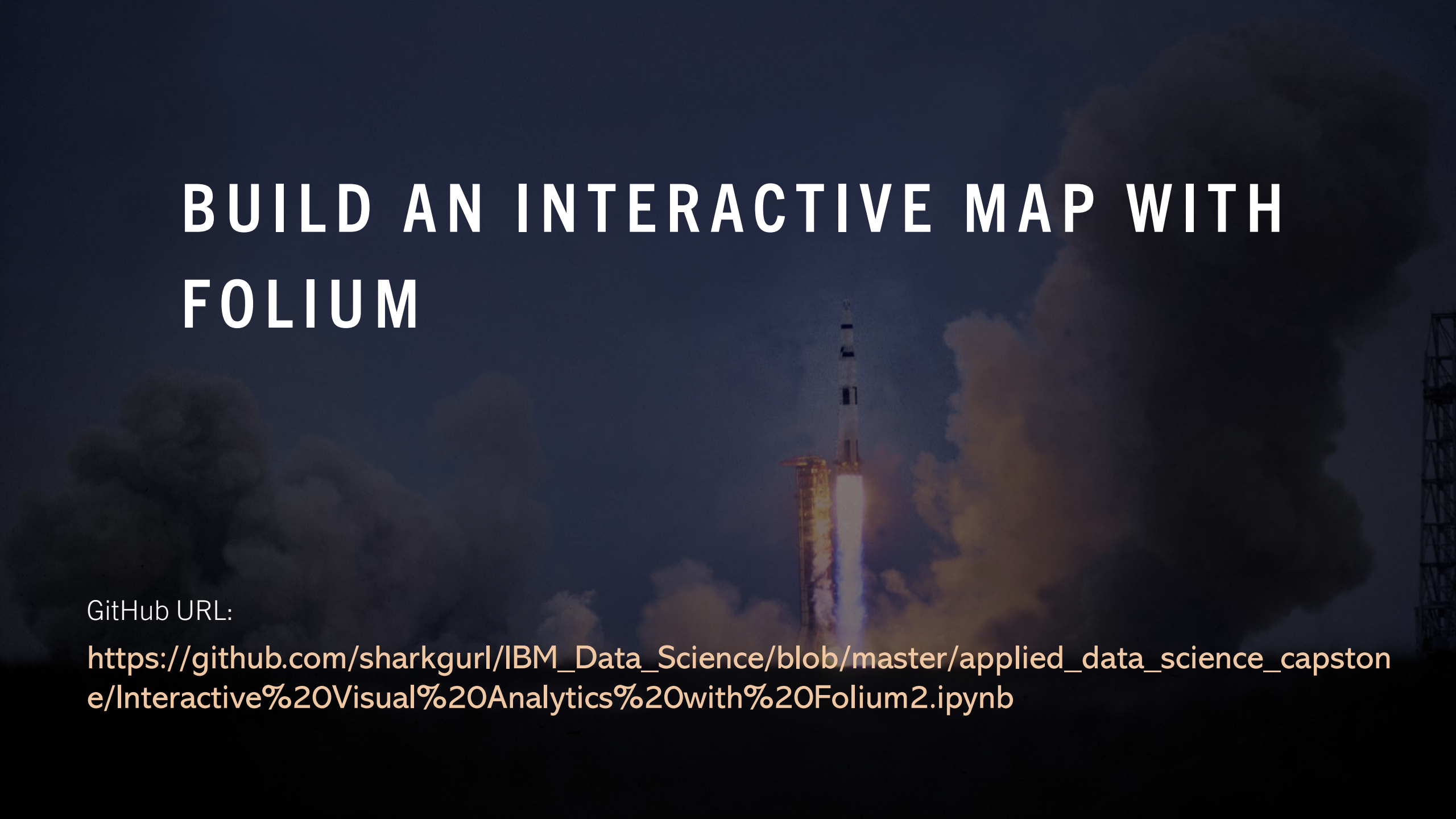
In [55]:
```
%sql SELECT LANDING__OUTCOME, COUNT(LANDING__OUTCOME) as Count FROM SPACEXDATASET WHERE DATE BETWEEN '2010-06-04' AND '2017-03-20' GROUP BY LANDING__OUTCOME ORDER BY Count DESC;
```

```
 * ibm_db_sa://wcy90736:***@ba99a9e6-d59e-4883-8fc0-d6a8c9f7a08f.c1ogj3sd0tgtu0lqde00.databases.appdomain.cloud:31321/bludb
Done.
```

Out[55]:

landing__outcome	COUNT
No attempt	10
Failure (drone ship)	5
Success (drone ship)	5
Controlled (ocean)	3
Success (ground pad)	3
Failure (parachute)	2
Uncontrolled (ocean)	2
Precluded (drone ship)	1

BUILD AN INTERACTIVE MAP WITH FOLIUM



GitHub URL:

https://github.com/sharkgurl/IBM_Data_Science/blob/master/applied_data_science_capstone/Interactive%20Visual%20Analytics%20with%20Folium2.ipynb

Map objects

A Folium Map Object was the first object to be created so that locations could be pinned on the map.

Circles (with text labels) were created and added to the map to highlight specific areas such as the different Launch Sites.

Folium Markers were added to enhance the map visualization and identify launch locations. Color coding markers helped us tell which launches were successful and which ones failed.

A MarkerCluster object was added to simplify all the launches that happened from the same location/coordinate.

PolyLine objects were added to measure distances such as how far the Launch Sites were from the Coast, or from a Highway, etc.

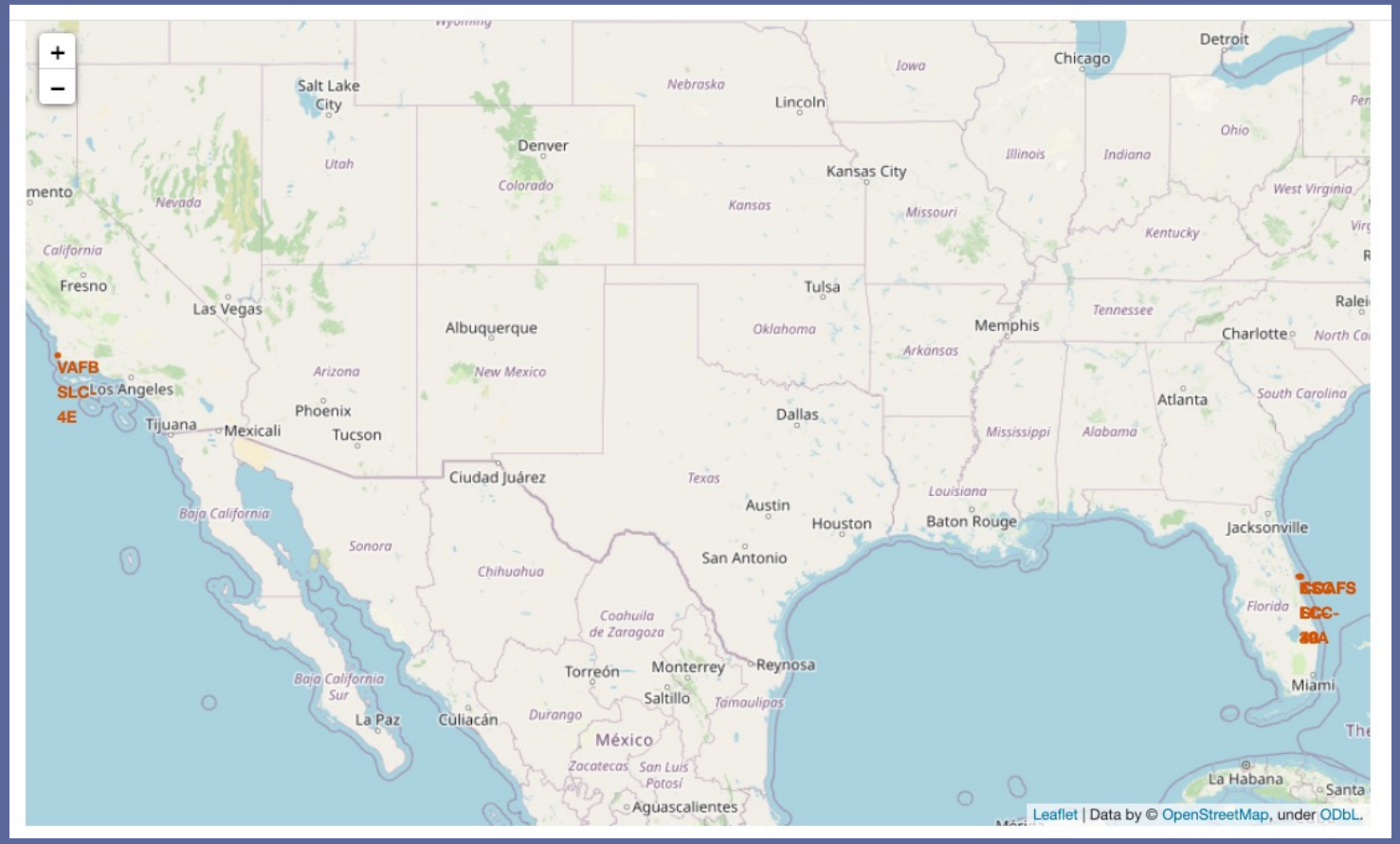
MousePosition was used to capture the coordinates of launch site proximities being measured.

A DivIcon was added to provide an anchor location, text label, popup information, etc.

Interactive Map
Folium

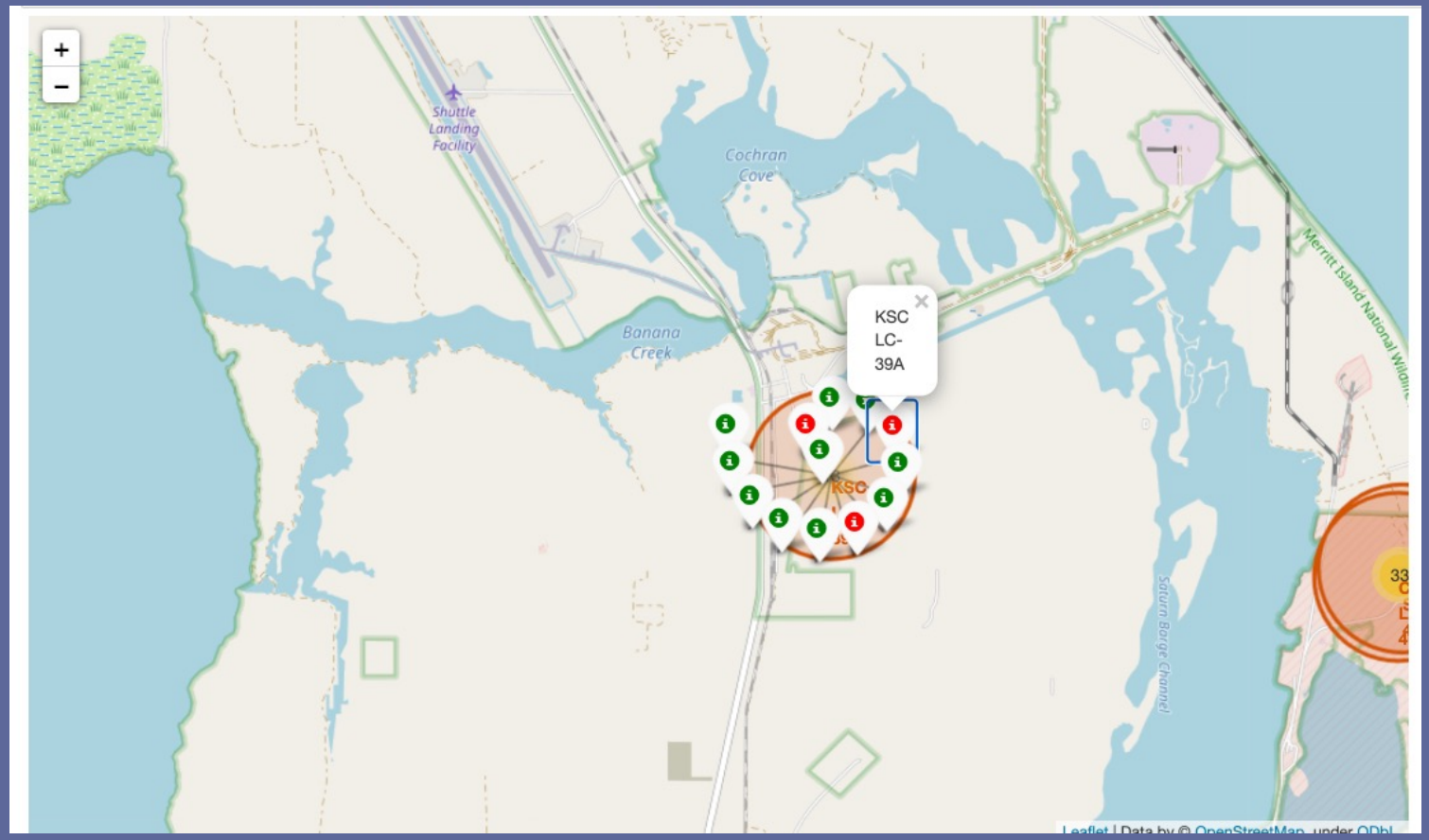
Launch Sites

- The Markers placed on the below map represent the launch sites. While one of the sites are on the west coast, 3 of the sites are in Florida and very close to each other. There are all very close to the coast.



KSC LC-39A Launch Outcomes

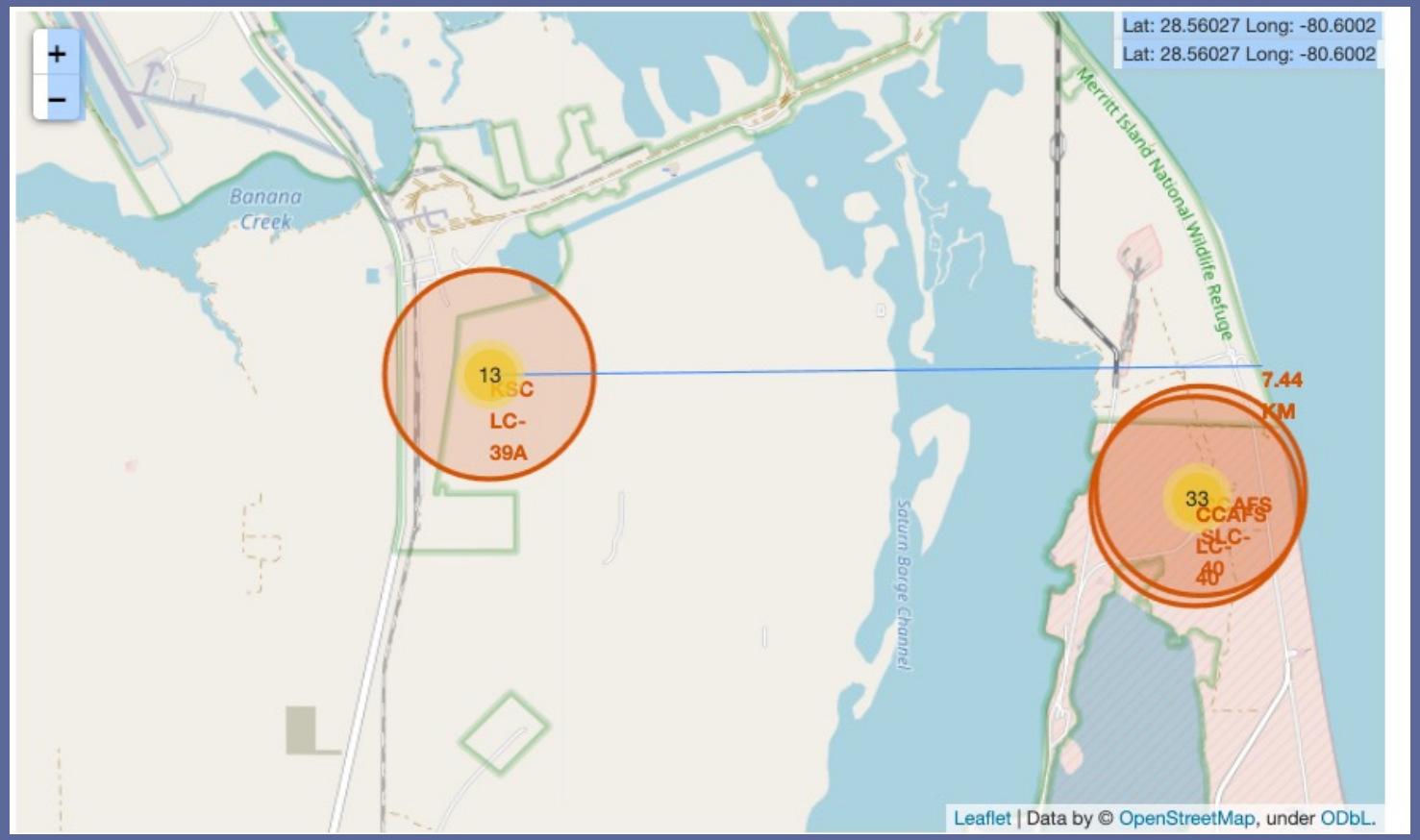
By color-coding the success/failure of the launches for each site, we can observe which sites have higher success rates than others. This is possible by adding objects such as Marker_Clusters, Circle, Color-Labeled Markers, and Popups.



LAUNCH SITES PROXIMITIES ANALYSIS

Distance between KSC LC-39A Launch Site and the Coast

All launch sites were close to the coast. Objects such as MousePosition can capture the lat, long of a point on the map which can be used to calculate the distance to a launch site as below (distance to the coast).



BUILD A DASHBOARD WITH PLOTLY DASH

GitHub URL:

https://github.com/sharkgurl/IBM_Data_Science/blob/master/applied_data_science_capstone/spacex_dash_app.py

Plotly Dash enables interactive visual analytics that can help users find patterns faster and more effective. Interactive graphs are also more appealing.

Plotly **Dash**

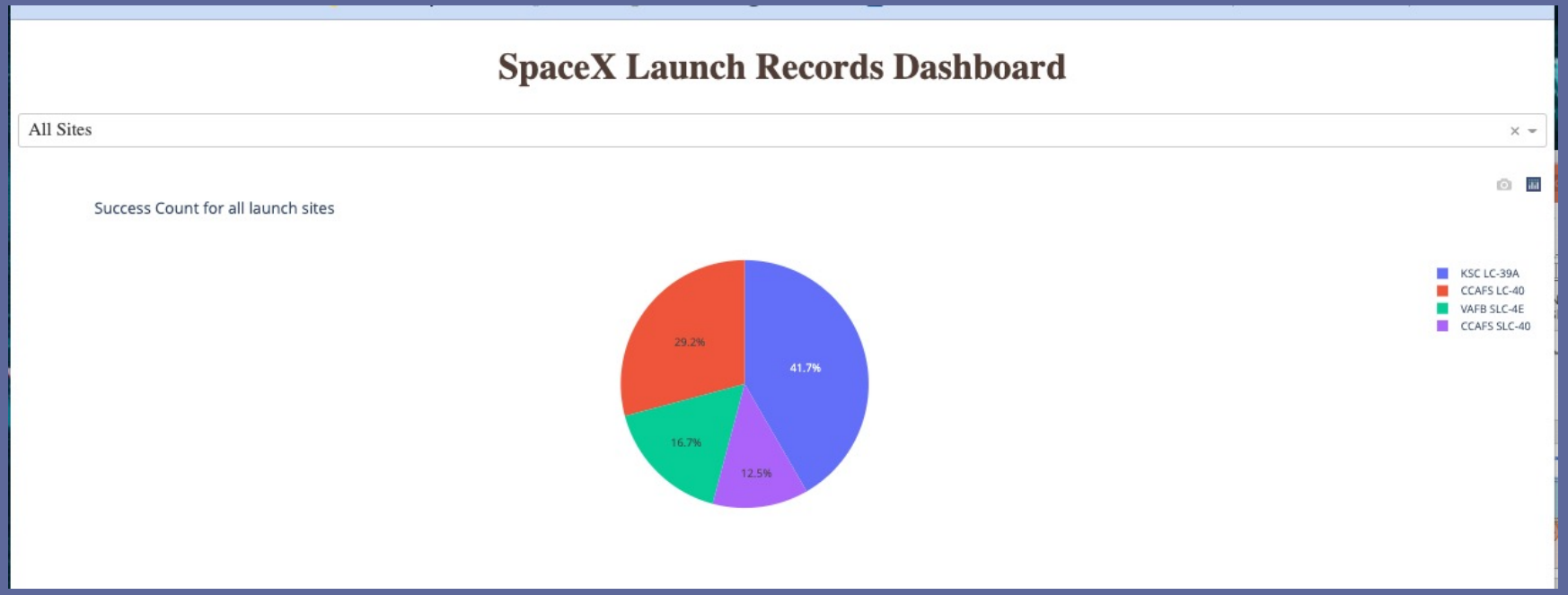
```python
# Import required libraries
import pandas as pd
import dash
import dash_html_components as html
import dash_core_components as dcc
#from dash import html
#from dash import dcc
from dash.dependencies import Input, Output
import plotly.express as px

# Read the airline data into pandas dataframe
spacex_df = pd.read_csv("spacex_launch_dash.csv")
max_payload = spacex_df['Payload Mass (kg)'].max()
min_payload = spacex_df['Payload Mass (kg)'].min()

# Create a dash application
app = dash.Dash(__name__)

# Create an app layout
app.layout = html.Div(children=[html.H1('SpaceX Launch Records Dashboard',
                                        style={'textAlign': 'center', 'color': '#503D36',
                                               'font-size': 40}),
                                # TASK 1: Add a dropdown list to enable Launch Site selection
                                dcc.Dropdown(id='site-dropdown',
                                options=[
                                    {'label':'All Sites','value':'ALL'},
                                    {'label': 'CCAFS LC-40', 'value': 'CCAFS LC-40'},
                                    {'label': 'CCAFS SLC-40', 'value': 'CCAFS SLC-40'},
                                    {'label': 'KSC LC-39A', 'value': 'KSC LC-39A'},
                                    {'label': 'VAFB SLC-4E', 'value': 'VAFB SLC-4E'},
                                ],
                                value='ALL',
                                placeholder="Select a Launch Site here",
                                searchable=True
                                ),
                                # The default select value is for ALL sites
                                # dcc.Dropdown(id='site-dropdown',...)
                                html.Br(),

                                # TASK 2: Add a pie chart to show the total successful launches count for all sites
                                # If a specific launch site was selected, show the Success vs. Failed counts for the site
                                html.Div(dcc.Graph(id='success-pie-chart')),
                                html.Br(),

                                html.P("Payload range (Kg):"),
                                # TASK 3: Add a slider to select payload range
                                #dcc.RangeSlider(id='payload-slider',...)
                                dcc.RangeSlider(id='payload-slider',
                                                min=0,
                                                max=10000,
                                                step=1000,
                                                value=[min_payload, max_payload]
                                                ),

                                # TASK 4: Add a scatter chart to show the correlation between payload and launch success
                                html.Div(dcc.Graph(id='success-payload-scatter-chart')),
                                ])
```

```python

# TASK 2:
# Add a callback function for `site-dropdown` as input, `success-pie-chart` as output

@app.callback(Output(component_id='success-pie-chart', component_property='figure'),
              Input(component_id='site-dropdown', component_property='value'))
def get_pie_chart(entered_site):
    filtered_df = spacex_df
    if entered_site == 'ALL':
        fig = px.pie(filtered_df, values='class',
        names='Launch Site',
        title='Success Count for all launch sites')
        return fig
    else:
        # return the outcomes piechart for a selected site
        filtered_df=spacex_df[spacex_df['Launch Site']== entered_site]
        filtered_df=filtered_df.groupby(['Launch Site','class']).size().reset_index(name='class count')
        fig=px.pie(filtered_df,values='class count',names='class',title=f"Total Success Launches for site {entered_site}")
        return fig


# TASK 4:
# Add a callback function for `site-dropdown` and `payload-slider` as inputs, `success-payload-scatter-chart` as output
@app.callback(Output(component_id='success-payload-scatter-chart',component_property='figure'),
              [Input(component_id='site-dropdown',component_property='value'),
               Input(component_id='payload-slider',component_property='value')])
def scatter(entered_site,payload):
    filtered_df = spacex_df[spacex_df['Payload Mass (kg)'].between(payload[0],payload[1])]
    # thought reusing filtered_df may cause issues, but tried it out of curiosity and it seems to be working fine

    if entered_site=='ALL':
        fig=px.scatter(filtered_df,x='Payload Mass (kg)',y='class',color='Booster Version Category',title='Success count on Payload mass for all sites')
        return fig
    else:
        fig=px.scatter(filtered_df[filtered_df['Launch Site']==entered_site],x='Payload Mass (kg)',y='class',color='Booster Version Category',title=f"Success c
        return fig


# Run the app
if __name__ == '__main__':
    app.run_server()
```

https://github.com/sharkgurl/IBM_Data_Science/blob/master/applied_data_science_capstone/spacex_dash_app.py

SpaceX Success Launch Counts for All Sites

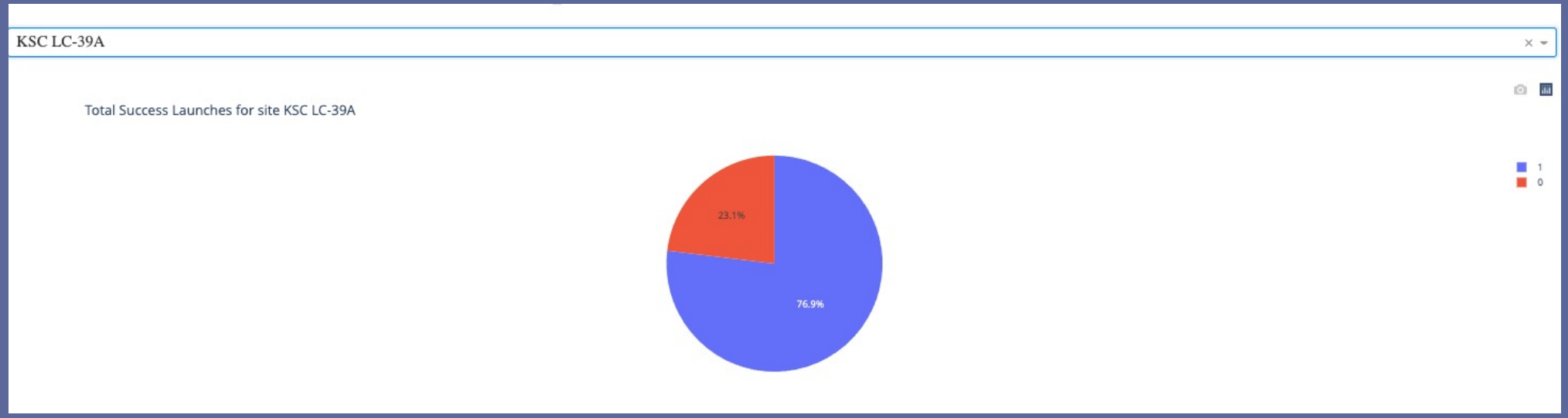
This pie chart shows the success count for all launch sites where KSC LC-39A leads with 41.7%.



KSC LC-39A Success Ratio

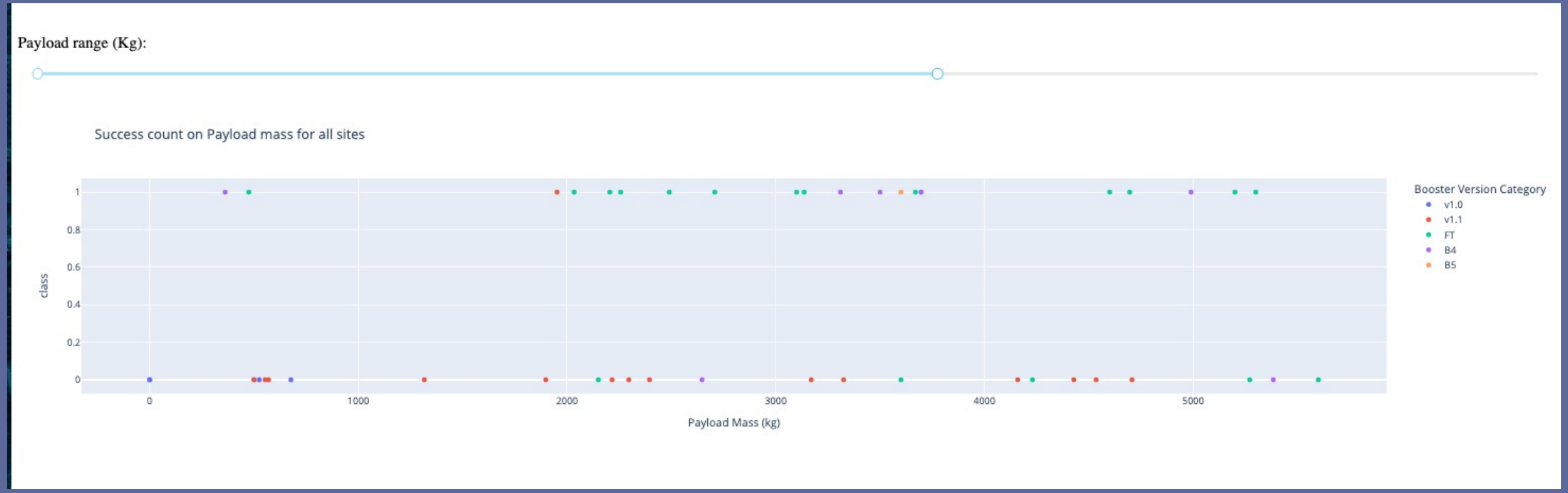
This pie chart shows the success ratio for the KSC LC-39A site alone.

76.9% of all KSC LC-39A launches were successful.

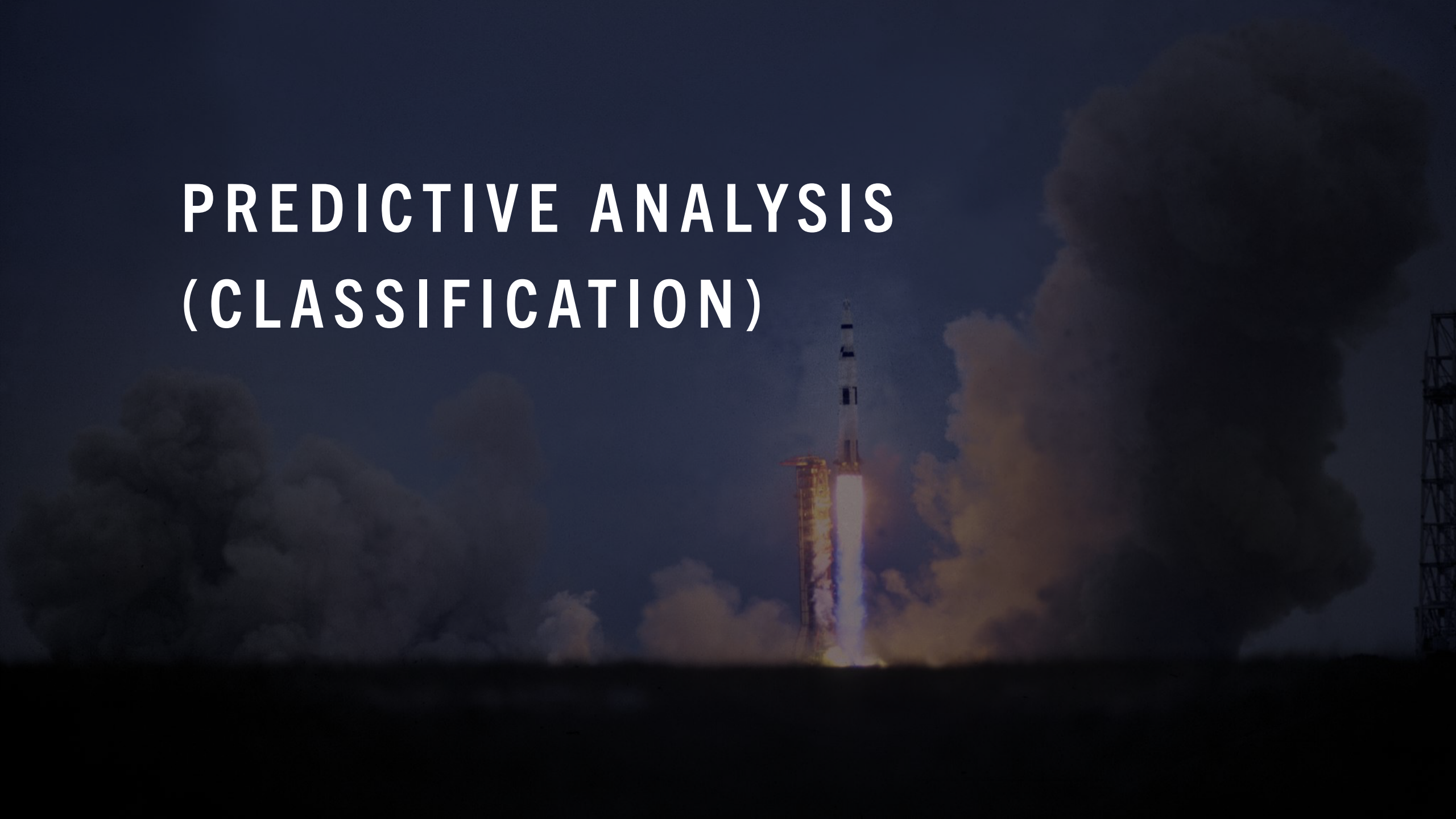


Payload vs Launch Outcome for All Sites with Slider

The FT Booster version had the largest success rate for below payload mass slicer.



PREDICTIVE ANALYSIS (CLASSIFICATION)



Predictive Analysis (Classification)

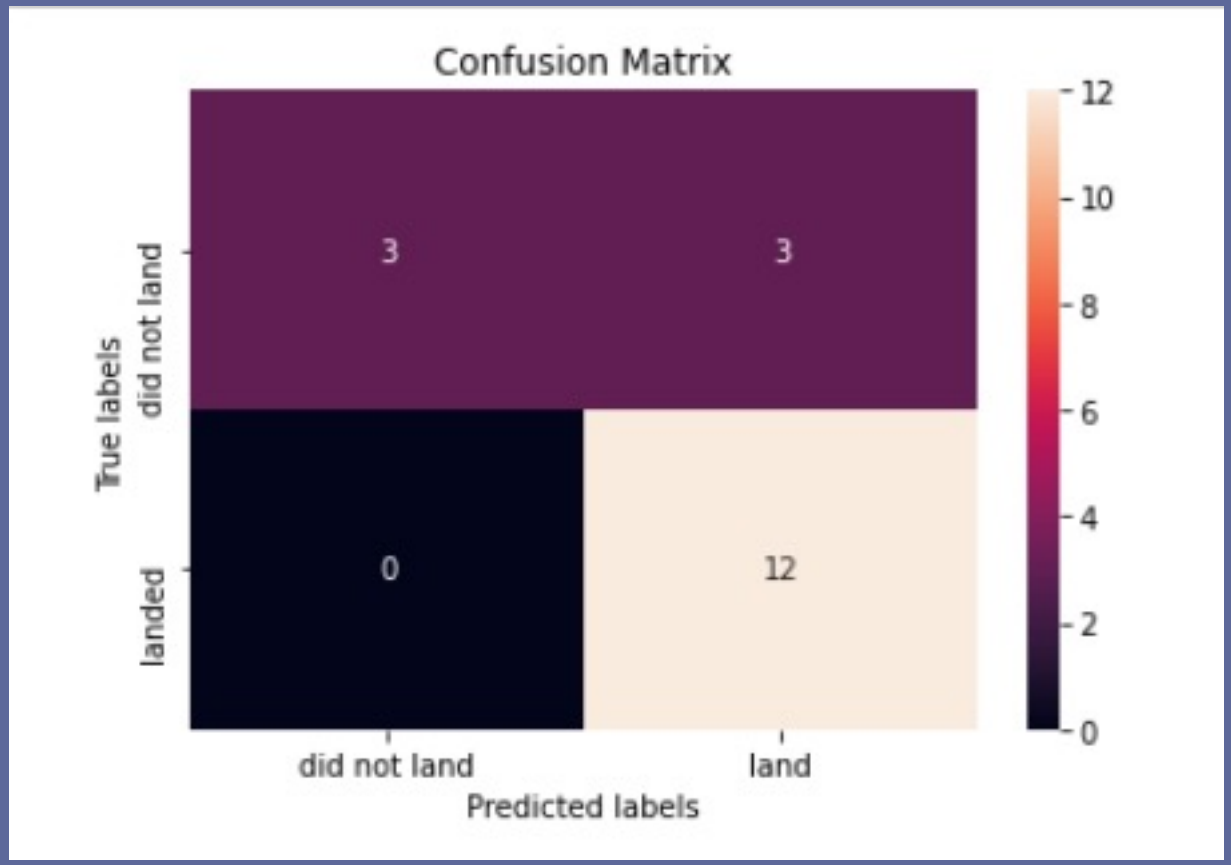
- The libraries pandas, numpy, matplotlib.pyplot, seaborn, and sklearn were used to build this analysis. A feature Class was added to the df and standardized with preprocessing.StandardScaler().
- The data was split using the function train_test_split().
- After the models were trained, hyperparameters were selected using the function GridSearchCV().
- Logistic Regression, SVM, Decision Tree, and KNN Classifiers were evaluated.
- A Confusion Matrix was used to find the best performing classification model

Classification Accuracy

All models performed practically the same, except for the decision tree which fit train data slightly better but test data worse.

Confusion Matrix

- False Positives were the major problem on all 4 confusion matrices



CONCLUSION

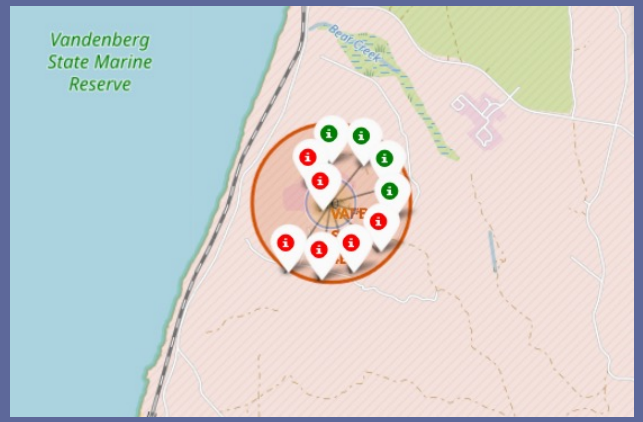
Conclusions

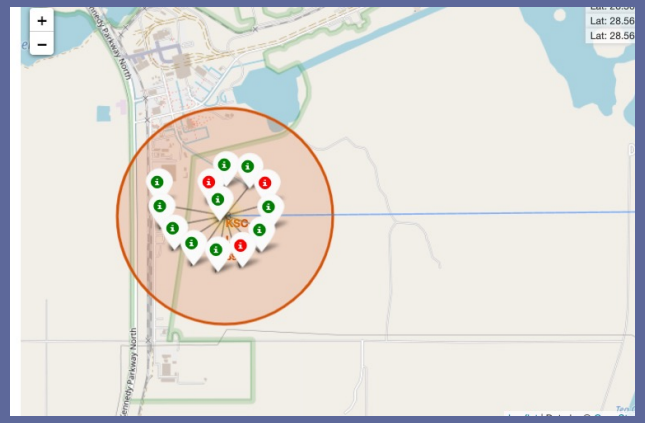
- Folium interactive visual analytics identify the KSC LC-39A as the optimal launch site .
- Plotly Dash validated that indeed the success rate of the KSC LC-39A is higher than the other sites.
- A better train accuracy does not necessarily result in better test accuracy.
- Launch Success Rate has improved considerably over the past 7 years.

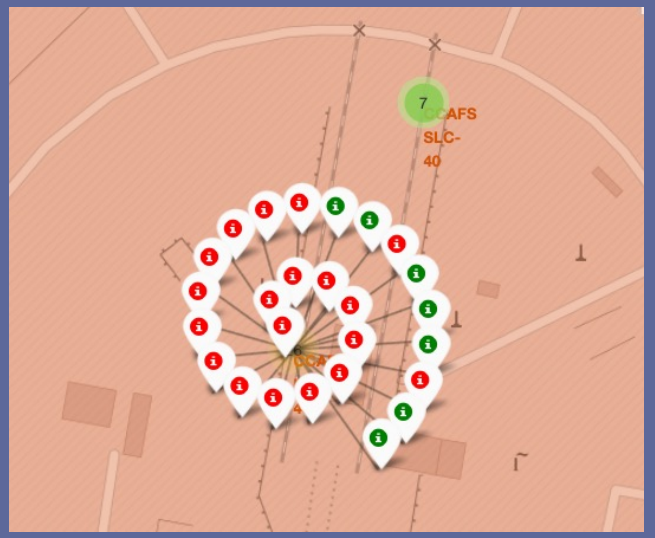
APPENDIX

Appendix

- You can see that the KSC LC-39A Launch site has a higher success rate than the other sites that can be easily identified as the optimal site with Folium visual analytics.








Thank **You**!