IBM Developer
SKILLS NETWORK

Winning Space Race with Data Science

Friederike Meyer
06.10.2022

Outline

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

Executive Summary

Summary of methodologies

- Data Collection (REST API and Web Scraping)
- Data Wrangling
- Data Visualization and SQL
- Interactive Folium Map
- Dashboard
- Classification Models

Summary of all results

- EDA Results
- Interactive Folium Maps
- Predictive Analytics

Introduction

- SpaceX offers the Falcon 9 rocket at a cost of 62 million dollar. The company is able to offer this low price because they can reuse the first stage.

- We predict whether the Falcon 9 first stage will land successfully.

Section 1

Methodology

Methodology

Executive Summary

- Data collection methodology:
    - Rest API
    - Web Scraping
- Perform data wrangling
    - Transformation of variables
- Perform exploratory data analysis (EDA) using visualization and SQL
- Perform interactive visual analytics using Folium and Plotly Dash
- Perform predictive analysis using classification models
    - Classification Models (Logistics Regression, Support Vector Machine, Decision Trees, K Nearest Neighbors)

Data Collection

Two Data Sources:

- SpaceX REST API
- Web scraping from Wikipedia

Data Collection Steps:

1) SpaceX REST API to import complete data set on SpaceX Launches
2) Web scraping with BeautifulSoup from Wikipedia to import the data contained on the table on the Wikipedia page

Data Collection – SpaceX API

- Data Collection with SpaceX REST API calls

Link to notebook:

https://jp-tok.dataplatform.cloud.ibm.com/analytics/notebooks/v2/e2f6ed1a-b996-42e5-8b8b-365f49dcc90b/view?access_token=2c8112bca28e0ebf2536623f2bbfbed4a16068e02673366715abdf14b282949f

Flowchart of Key SpaceX API calls

```
In [31]: spacex_url="https://api.spacexdata.com/v4/launches/past"

In [32]: response = requests.get(spacex_url)
```

```
In [36]: # Use json_normalize meethod to convert the json result into a dataframe
         data=pd.json_normalize(response.json())
```

```
In [41]: # Call getBoosterVersion
         getBoosterVersion(data)
```

```
In [58]: # Hint data['BoosterVersion']!='Falcon 1'

         indexnames = launch_data.loc[launch_data['BoosterVersion']!='Falcon 9'].index
         launch_data.drop(indexnames, inplace = True)
         data_falcon9 = launch_data
         data_falcon9.head()
```

Data Collection - Scraping

- Data Collection with WebScraping

Link to notebook:

https://jp-tok.dataplatform.cloud.ibm.com/analytics/notebooks/v2/5a6fffb3-fa69-4462-884c-fd1b9211f862/view?access_token=4807bc84d6fdb1cd0accda5609dd629afc682068d2333e9c61ea8339ebc17300

Flowchart of Key SpaceX API calls

```
In [31]: # use requests.get() method with the provided static_url
         # assign the response to a object

         data=requests.get(static_url)
```

Create a `BeautifulSoup` object from the HTML `response`

```
In [35]: # Use BeautifulSoup() to create a BeautifulSoup object from a response text content
         soup=BeautifulSoup(data.content, 'html5lib')
```

Print the page title to verify if the `BeautifulSoup` object was created properly

```
In [36]: # Use soup.title attribute
         soup.title

Out[36]: <title>List of Falcon 9 and Falcon Heavy launches - Wikipedia</title>
```

```
In [37]: # Use the find_all function in the BeautifulSoup object, with element type `table`
         # Assign the result to a list called `html_tables`
         html_tables=soup.find_all('table')
```

```
In [39]: column_names = []

         # Apply find_all() function with `th` element on first_launch_table
         # Iterate each th element and apply the provided extract_column_from_header() to get a column name
         # Append the Non-empty column name (`if name is not None and len(name) > 0`) into a list called column_names

         elements=first_launch_table.find_all('th')
         for element in elements:
             column_name=extract_column_from_header(element)
             if column_name!=None and len(column_name)>0:
                 column_names.append(column_name)
```

Next Step: Creating Dictionary and Appending Data to Keys

Data Wrangling

Data Wrangling Steps:

1) Calculate the number of launches on each site
2) Calculate the number and occurrence of each orbit
3) Calculate the number and occurrence of mission outcomes per orbit type
4) Create a landing outcome label from the Outcome column

Link to Notebook:

https://jp-tok.dataplatform.cloud.ibm.com/analytics/notebooks/v2/939d7293-f5a2-4b15-864d-d9d1d51c023f/view?access_token=d6cf9c2079a6430e94f5eba9c606c4d815a7693eec161021c58a1ee9935dbf34

EDA with Data Visualization

Overview of Plotted Charts:

- Scatterplot (for Continuous Variables)
- Bar Chart (for Categorical Variables)

Link to Notebook: https://jp-tok.dataplatform.cloud.ibm.com/analytics/notebooks/v2/dcd67f60-d631-475f-8f2a-ce9a8031783d/view?access_token=ae11ae1ef7163e0fbf70c46d7d246a465093d6a733bd08857140f5b3f604cec2

EDA with SQL

EDA with SQL Steps:

1) Connect to Database
2) Display the names of the unique launch sites in the space mission
3) Display 5 records where launch sites begin with the string 'CCA’
4) Display the total payload mass carried by boosters launched by NASA (CRS)
5) Display average payload mass carried by booster version F9 v1.1
6) List the date when the first successful landing outcome in ground pad was achieved
7) List the names of the boosters which have success in drone ship and have payload mass greater than 4000 but less than 6000
8) List the total number of successful and failure mission outcomes
9) List the names of the booster_versions which have carried the maximum payload mass. Use a subquery
10) List the failed landing_outcomes in drone ship, their booster versions, and launch site names for in year 2015
11) 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

Link to Notebook: https://jp-tok.dataplatform.cloud.ibm.com/analytics/notebooks/v2/0e958e7d-725f-4b75-8095-febc578ff7c6/view?access_token=80e5a8ee0a35a6778ba0615f3e7e2a7c6ac08d33c15f17ce2c33ae9ef8260ede

Build an Interactive Map with Folium

Overview of Map Items:

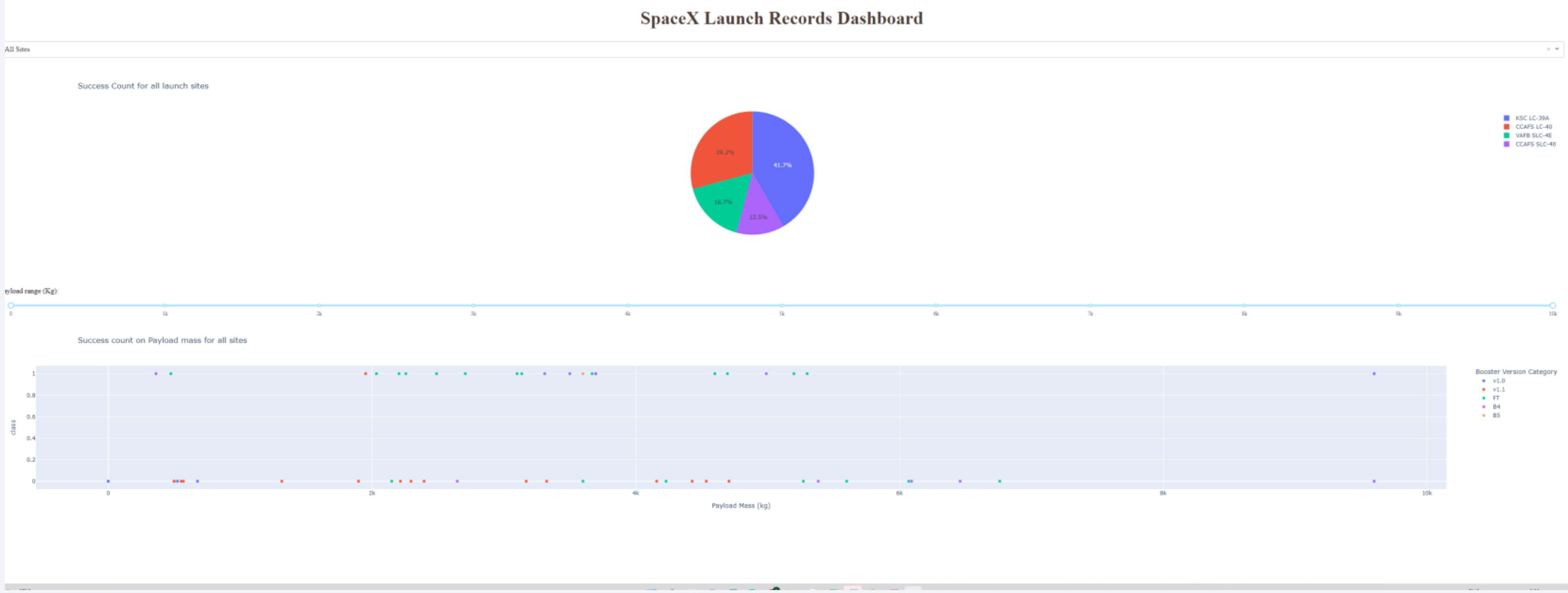
1) Launch Sites
2) Successful/failed launches
3) Proximity of Launch Site

Link to Notebook: https://jp-tok.dataplatform.cloud.ibm.com/analytics/notebooks/v2/766a1236-b2cc-4fa2-96a4-66bc4edc2317/view?access_token=265d45164af6eb93c1733fd68937fb0f0091301323027754 64ce2c5c4b469c6e

Build a Dashboard with Plotly Dash

Elements Added:

1) Dropdown Menu
2) Bar Charts
3) Scatter Plots



Link to Notebook: https://github.com/Friedie-FME/Applied-Data-Science-Capstone/blob/main/Dashboard.ipynb

Predictive Analysis (Classification)

```
In [84]: X_train, X_test, Y_train, Y_test = train_test_split(X, Y,test_size=0.2, random_state=2)
```

```
In [86]: parameters ={'C':[0.01,0.1,1],
                      'penalty':['l2'],
                      'solver':['lbfgs']}
```

```
In [87]: parameters ={"C":[0.01,0.1,1],'penalty':['l2'], 'solver':['lbfgs']}# l1 lasso l2 ridge
         lr=LogisticRegression()

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

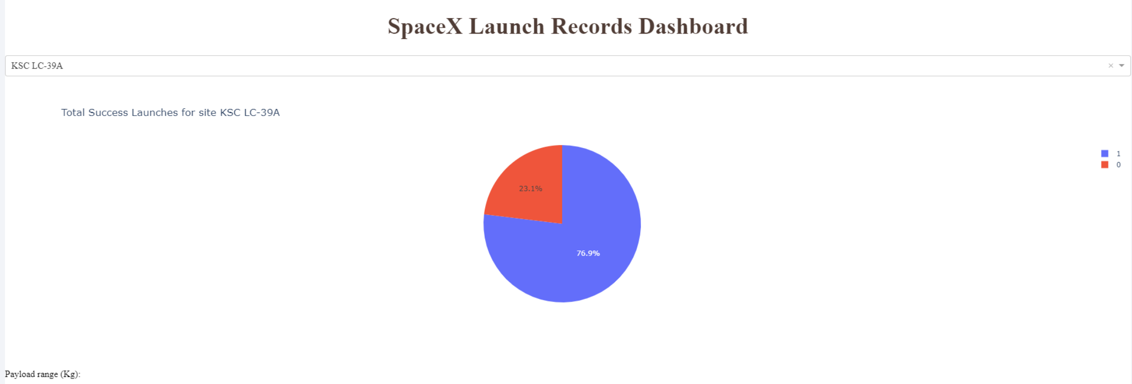
```
In [108…  print('Logistics Regression:', logreg_cv.score(X_test, Y_test))
          print( 'Support Vector Machine:', svm_cv.score(X_test, Y_test))
          print('Decision tree:', tree_cv.score(X_test, Y_test))
          print('K Nearest Neighbors:', knn_cv.score(X_test, Y_test))

          Logistics Regression: 0.8333333333333334
          Support Vector Machine: 0.8333333333333334
          Decision tree: 0.8333333333333334
          K Nearest Neighbors: 0.8333333333333334
```

Besides **logistics regression**, the following was also run: **Support Vector Machine, Decision Tree, K Nearest Neighbors**

Link to Notebook: https://jp-tok.dataplatform.cloud.ibm.com/analytics/notebooks/v2/49c66a66-013d-4372-b0c1-a45a7c18acd9/view?access_token=b7594d2676625830b347e0485fc3e7b5177b2b7241c74bfab8918938dfb99d44

Results

- The SVM, KNN and Logistics Regression models have the best prediction accuracy
- KSC LC-39A has the best success rate




```
In [108…
print('Logistics Regression:', logreg_cv.score(X_test, Y_test))
print( 'Support Vector Machine:', svm_cv.score(X_test, Y_test))
print('Decision tree:', tree_cv.score(X_test, Y_test))
print('K Nearest Neighbors:', knn_cv.score(X_test, Y_test))

Logistics Regression: 0.8333333333333334
Support Vector Machine: 0.8333333333333334
Decision tree: 0.8333333333333334
K Nearest Neighbors: 0.8333333333333334
```

Section 2

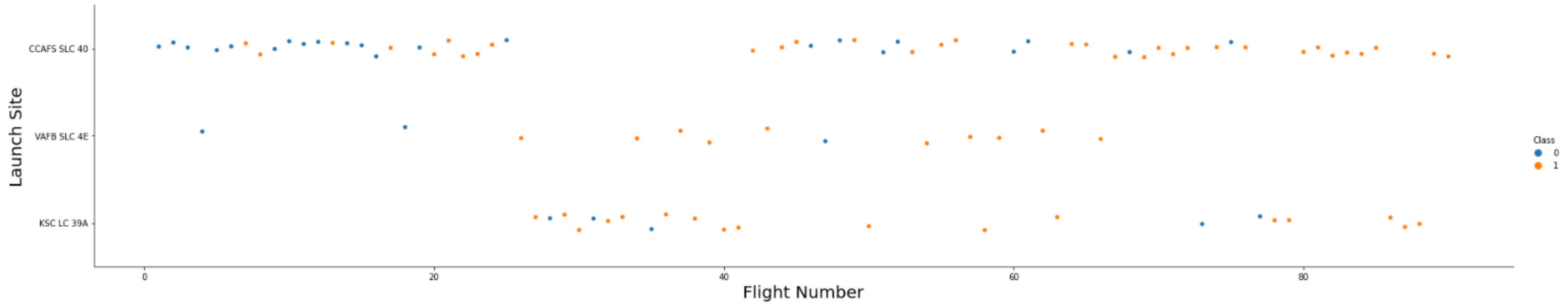
Insights drawn from EDA

Flight Number vs. Launch Site

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'`

In [4]:
```
# 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()
```

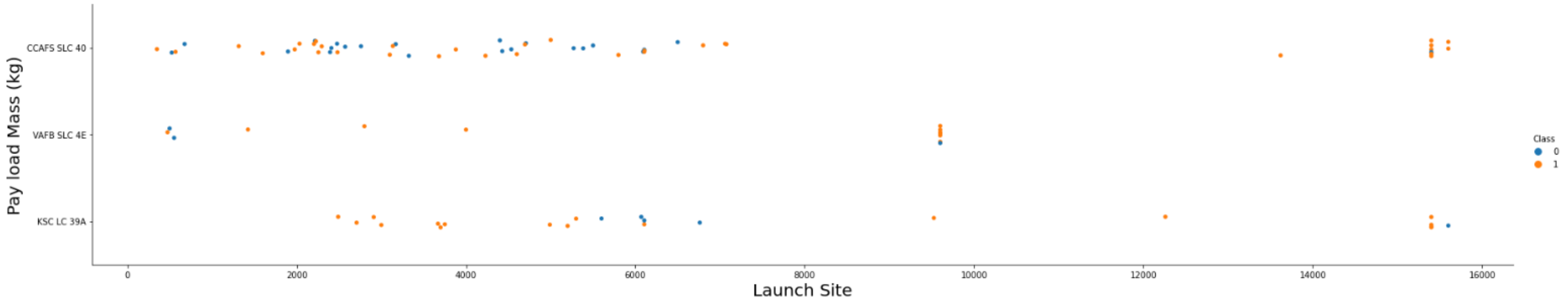


Payload vs. Launch Site

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.

In [5]:
```
# 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("Launch Site",fontsize=20)
plt.ylabel("Pay load Mass (kg)",fontsize=20)
plt.show()
```



Success Rate vs. Orbit Type

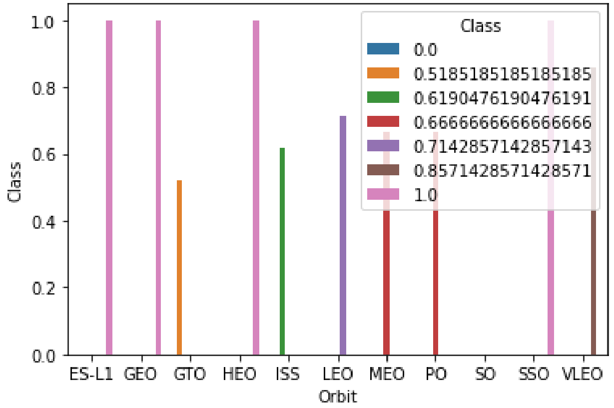
TASK 3: Visualize the relationship between success rate of each orbit type

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 sucess rate of each orbit

In [6]:
```
# HINT use groupby method on Orbit column and get the mean of Class column
success_rate = df.groupby('Orbit').mean()
success_rate.reset_index(inplace=True)
sns.barplot(x="Orbit",y="Class",data=success_rate,hue='Class')
```

Out[6]:
```
<AxesSubplot:xlabel='Orbit', ylabel='Class'>
```



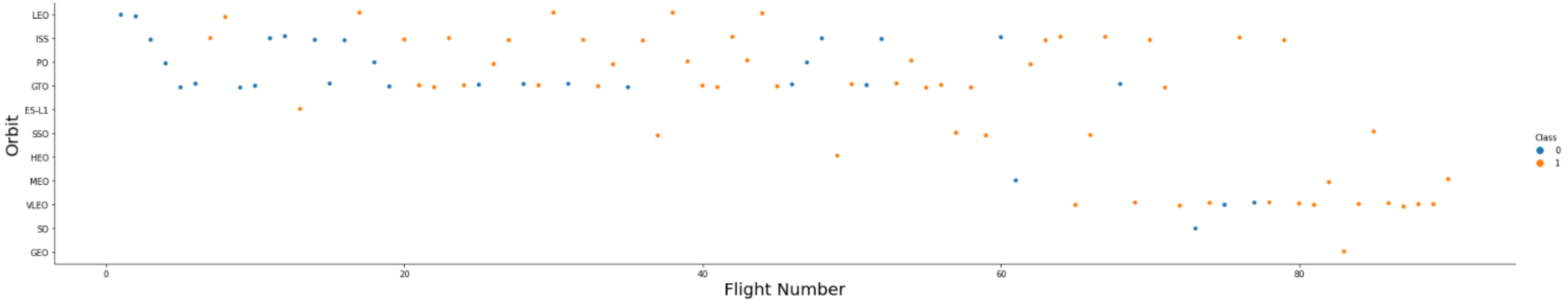
Analyze the ploted bar chart try to find which orbits have high sucess rate.

Flight Number vs. Orbit Type

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.

In [7]:
```
# 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()
```



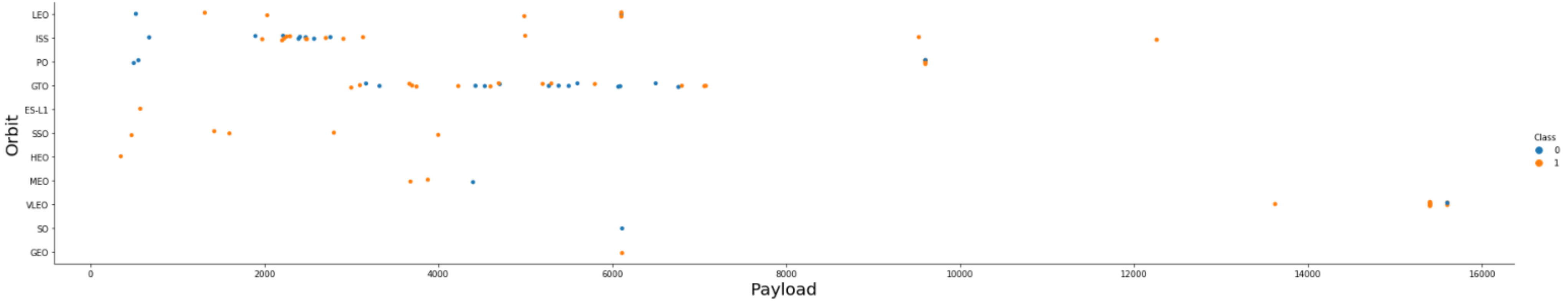
You should see that in the LEO orbit the Success appears related to the number of flights; on the other hand, there seems to be no relationship between flight number when in GTO orbit.

Payload vs. Orbit Type

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

In [10]:
```
# 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",fontsize=20)
plt.ylabel("Orbit",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(unsuccessful mission) are both there here.

Launch Success Yearly Trend

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:

In [11]:
```
# A function to Extract years from the date
year=[]
def Extract_year(date):
    for i in df["Date"]:
        year.append(i.split("-")[0])
    return year
```

In [17]:
```
# Plot a line chart with x axis to be the extracted year and y axis to be the success rate
sns.lineplot(y="successrate", x="year", hue="Class", data=df, aspect = 5)
plt.xlabel("year",fontsize=20)
plt.ylabel("success rate",fontsize=20)
plt.show()
```

All Launch Site Names

Task 1

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

```
In [19]: %sql SELECT UNIQUE(LAUNCH_SITE) FROM SPACEXDATASET

 * ibm_db_sa://tjg77476:***@b0aebb68-94fa-46ec-a1fc-1c999edb6187.c3n41cmd0nqnrk39u98g.databases.appdomain.cloud:31249/bludb
Done.
```

Out[19]:

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

Launch Site Names Begin with 'CCA'

Task 2

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

In [20]:
```
%sql select LAUNCH_SITE from SPACEXDATASET where LAUNCH_SITE LIKE 'CCA%' LIMIT 5
```

```
 * ibm_db_sa://tjg77476:***@b0aebb68-94fa-46ec-a1fc-1c999edb6187.c3n41cmd0nqnrk39u98g.databases.appdomain.cloud:31249/bludb
Done.
```

Out[20]:

launch_site
CCAFS LC-40
CCAFS LC-40
CCAFS LC-40
CCAFS LC-40
CCAFS LC-40

Total Payload Mass

Task 3

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

In [21]:
```
%sql SELECT SUM(PAYLOAD_MASS__KG_) from SPACEXDATASET WHERE PAYLOAD LIKE '%CRS%'
```

```
 * ibm_db_sa://tjg77476:***@b0aebb68-94fa-46ec-a1fc-1c999edb6187.c3n41cmd0nqnrk39u98g.databases.appdomain.cloud:31249/bludb
Done.
```

Out[21]:

1
56479

Average Payload Mass by F9 v1.1

Task 4

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

In [22]:
```
%sql SELECT AVG(PAYLOAD_MASS__KG_) from SPACEXDATASET WHERE BOOSTER_VERSION LIKE '%F9 v1.1%'
```

```
 * ibm_db_sa://tjg77476:***@b0aebb68-94fa-46ec-a1fc-1c999edb6187.c3n41cmd0nqnrk39u98g.databases.appdomain.cloud:31249/bludb
Done.
```

Out[22]:

1
3226

First Successful Ground Landing Date

Task 5

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

Hint:Use min function

In [23]:
```
%sql SELECT MIN(DATE) FROM SPACEXDATASET WHERE MISSION_OUTCOME LIKE 'Success'
```

```
 * ibm_db_sa://tjg77476:***@b0aebb68-94fa-46ec-a1fc-1c999edb6187.c3n41cmd0nqnrk39u98g.databases.appdomain.cloud:31249/bludb
Done.
```

Out[23]:

1
2010-04-06

Successful Drone Ship Landing with Payload between 4000 and 6000

Task 6

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

```
In [24]: %sql SELECT BOOSTER_VERSION FROM SPACEXDATASET WHERE LANDING__OUTCOME LIKE '%Success (drone ship)%' AND PAYLOAD_MASS__KG_ BETWEEN 4000 AND 6000

 * ibm_db_sa://tjg77476:***@b0aebb68-94fa-46ec-a1fc-1c999edb6187.c3n41cmd0nqnrk39u98g.databases.appdomain.cloud:31249/bludb
Done.
```

Out[24]:

booster_version
F9 FT B1022
F9 FT B1031.2

Total Number of Successful and Failure Mission Outcomes

Task 7

List the total number of successful and failure mission outcomes

In [25]:
```
%sql SELECT COUNT(*) FROM SPACEXDATASET GROUP BY MISSION_OUTCOME
```

```
 * ibm_db_sa://tjg77476:***@b0aebb68-94fa-46ec-a1fc-1c999edb6187.c3n41cmd0nqnrk39u98g.databases.appdomain.cloud:31249/bludb
Done.
```

Out[25]:

1
44
1

Boosters Carried Maximum Payload

Task 8

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

In [26]:
```
%sql SELECT BOOSTER_VERSION FROM SPACEXDATASET  WHERE PAYLOAD_MASS__KG_ = (SELECT MAX(PAYLOAD_MASS__KG_) FROM SPACEXDATASET)
```

```
 * ibm_db_sa://tjg77476:***@b0aebb68-94fa-46ec-a1fc-1c999edb6187.c3n41cmd0nqnrk39u98g.databases.appdomain.cloud:31249/bludb
Done.
```

Out[26]:

booster_version
F9 B5 B1048.4
F9 B5 B1049.4
F9 B5 B1049.5
F9 B5 B1060.2
F9 B5 B1058.3

2015 Launch Records

Task 9

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

In [29]:
```
%sql SELECT LANDING__OUTCOME, BOOSTER_VERSION, LAUNCH_SITE FROM SPACEXDATASET WHERE LANDING__OUTCOME LIKE 'Failure (drone ship)' AND DATE LIKE '%2015%'
```

```
 * ibm_db_sa://tjg77476:***@b0aebb68-94fa-46ec-a1fc-1c999edb6187.c3n41cmd0nqnrk39u98g.databases.appdomain.cloud:31249/bludb
Done.
```

Out[29]:

landing__outcome	booster_version	launch_site
Failure (drone ship)	F9 v1.1 B1012	CCAFS LC-40

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

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

In [30]:
```
%sql SELECT LANDING__OUTCOME, COUNT(*) AS counts FROM SPACEXDATASET WHERE date BETWEEN '2010-06-04' AND '2017-03-20' GROUP BY LANDING__OUTCOME ORDER BY counts DESC
```

```
 * ibm_db_sa://tjg77476:***@b0aebb68-94fa-46ec-a1fc-1c999edb6187.c3n41cmd0nqnrk39u98g.databases.appdomain.cloud:31249/bludb
Done.
```

Out[30]:

landing__outcome	counts
No attempt	7
Failure (drone ship)	2
Success (drone ship)	2
Success (ground pad)	2
Controlled (ocean)	1
Failure (parachute)	1

Section 3

Launch Sites Proximities Analysis

Location Map of Launch Sites

In [8]:

```
# Initial the map
site_map = folium.Map(location=nasa_coordinate, zoom_start=5)

# For each launch site, add a Circle object based on its coordinate (Lat, Long) values. In addition, add Launch site name as a popup label
for i,launch_site in launch_sites_df.iterrows():
    coord = [launch_site['Lat'], launch_site['Long']]
    site_name = launch_site['Launch Site']


circle = folium.Circle(coord, radius=1000, color='#0054d3', fill=True).add_child(folium.Popup(launch_site))
marker = folium.map.Marker(coord, icon=DivIcon(icon_size=(20,20),icon_anchor=(0,0), html='<div style="font-size: 12; color:#d35400;"><b>%s</b></div>' % launch_site['Launch Site'], ))


site_map.add_child(circle)
site_map.add_child(marker)

site_map
```

Out[8]:

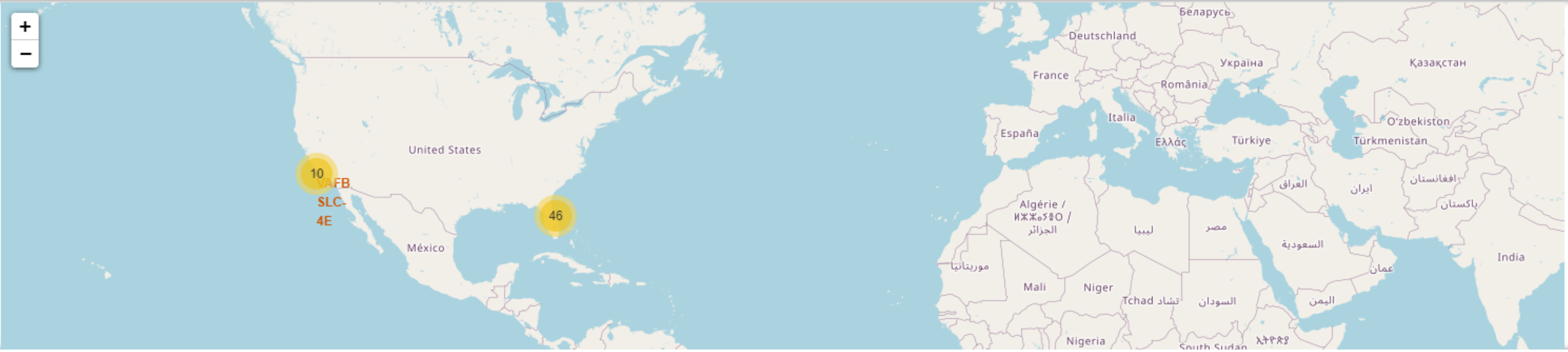
Launch Outcomes on Map

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

In [13]:

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

# for each row in spacex_df data frame
# create a Marker object with its coordinate
# and customize the Marker's icon property to indicate if this launch was successed 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([record['Lat'],record['Long']],
                           icon=folium.Icon(color='green', icon_color=record['marker_color']))
    # marker = folium.Marker(...)
    marker_cluster.add_child(marker)

site_map
```

Out[13]:



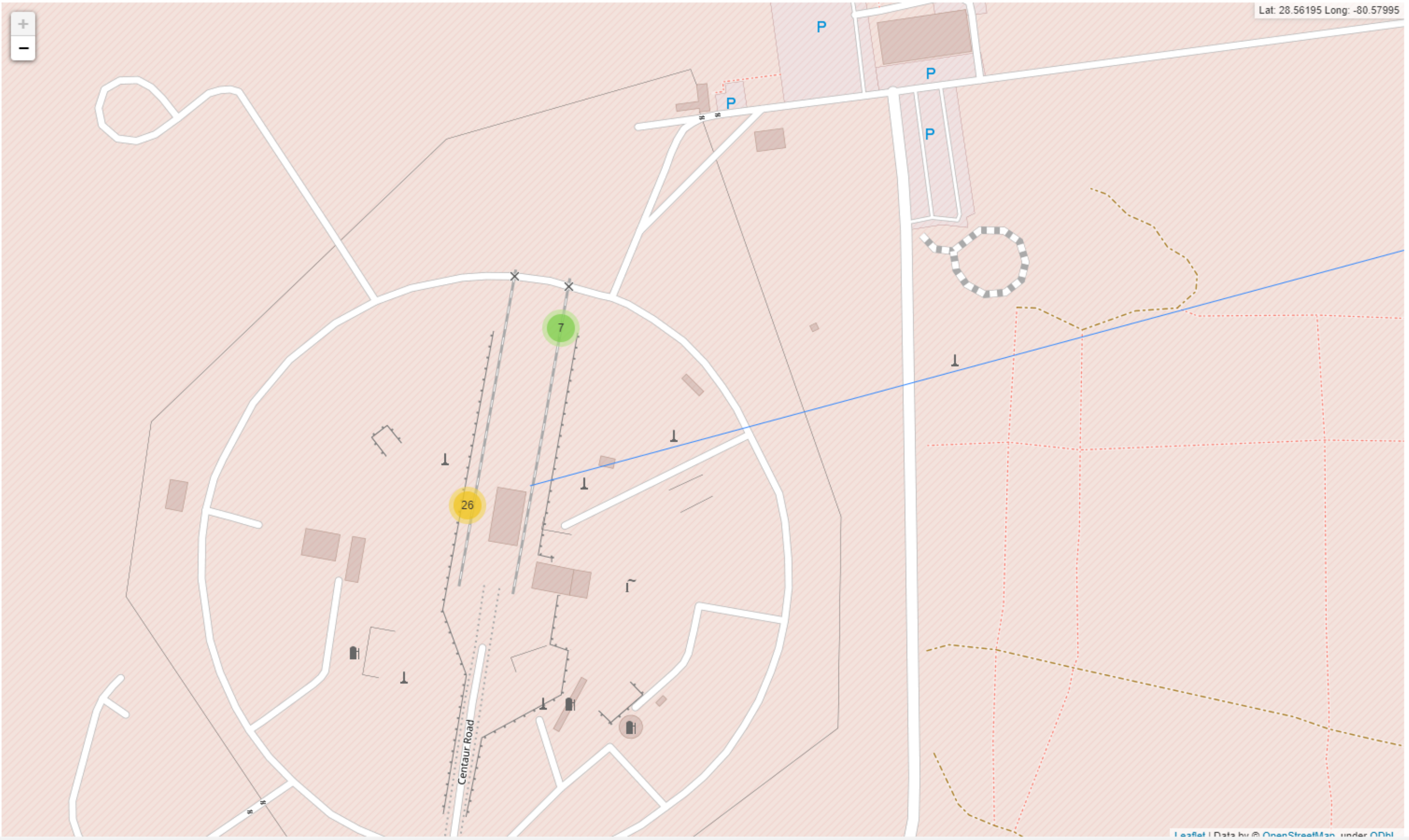
Launch Site Proximities on Map

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

In [18]:

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

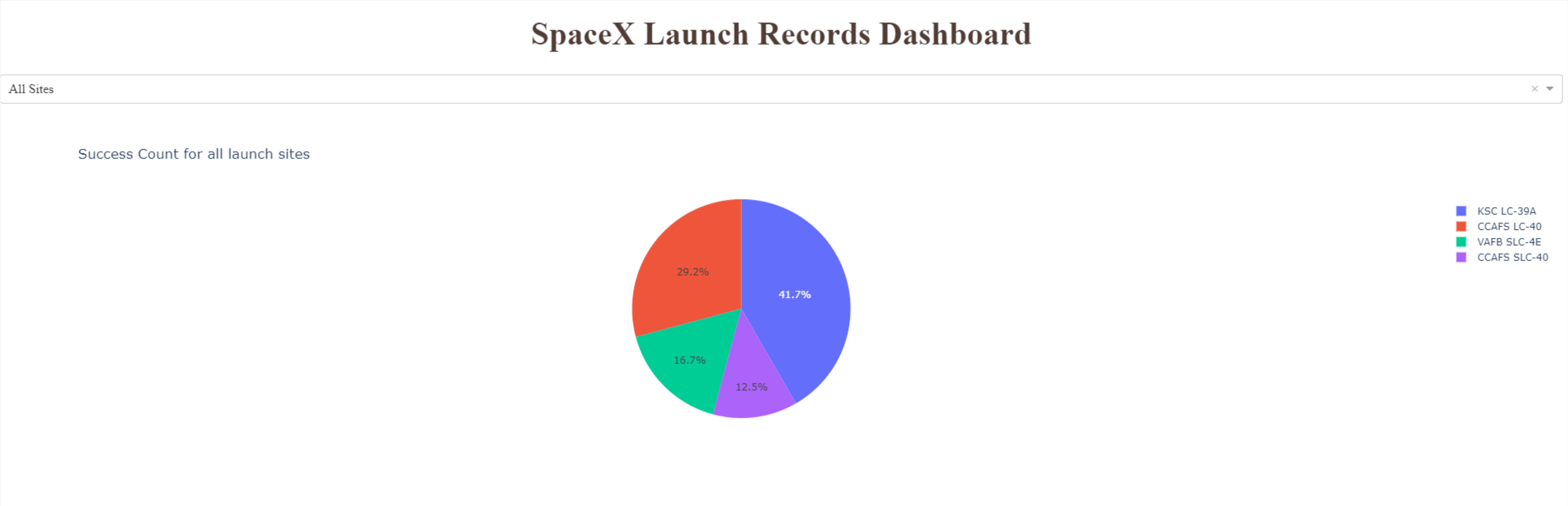
Out[18]:



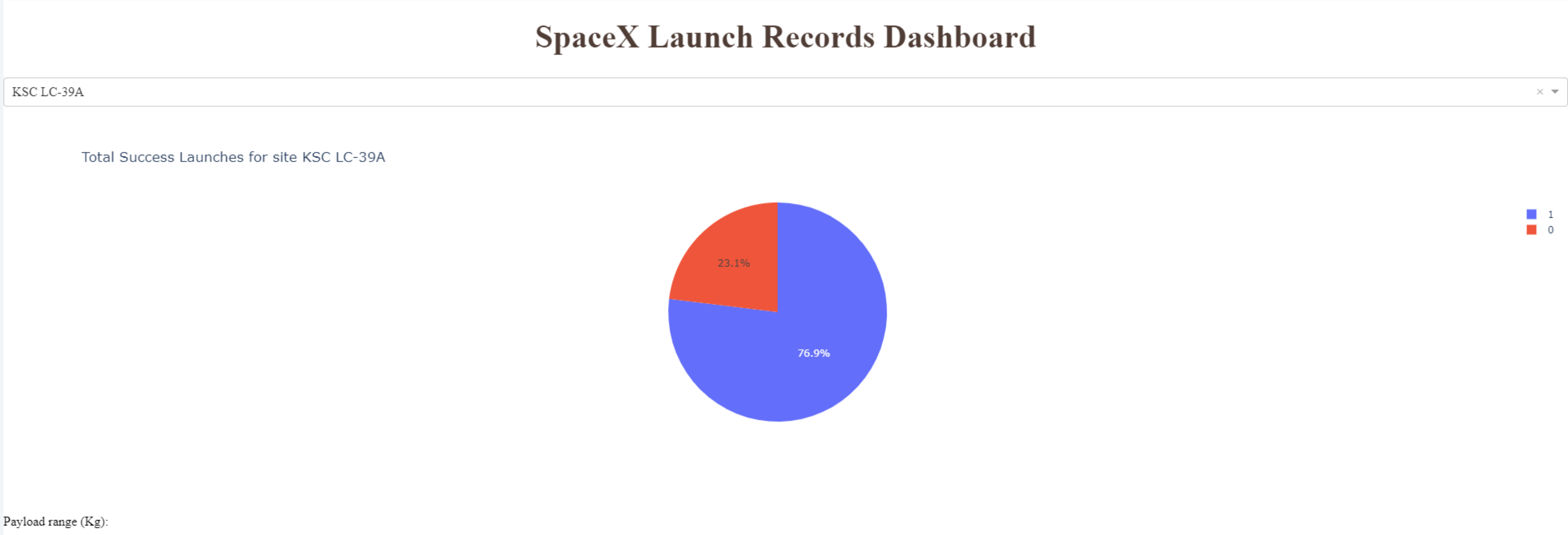
Section 4

Build a Dashboard with Plotly Dash

Launch Success for All Sites



Launch Site with Highest Launch Success



Payload vs Launch Outcome Scatter Plot

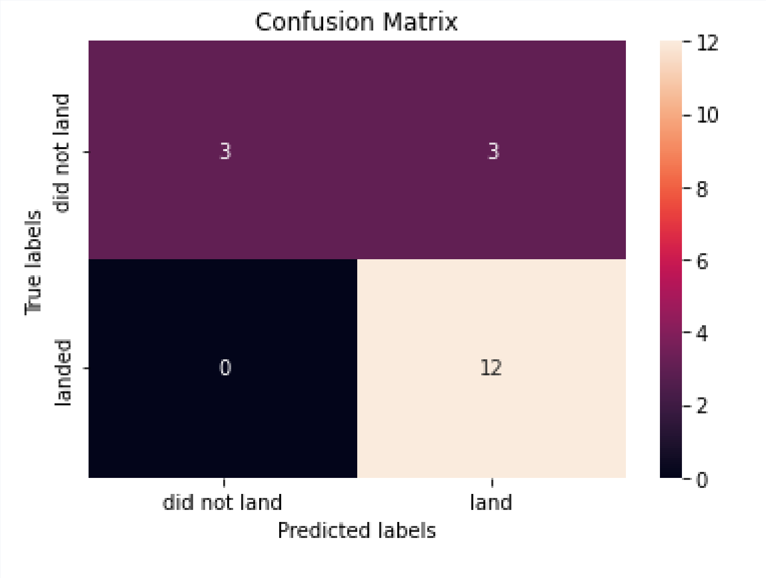
Section 5

Predictive Analysis (Classification)

Classification Accuracy

```
In [108…  print('Logistics Regression:', logreg_cv.score(X_test, Y_test))
          print( 'Support Vector Machine:', svm_cv.score(X_test, Y_test))
          print('Decision tree:', tree_cv.score(X_test, Y_test))
          print('K Nearest Neighbors:', knn_cv.score(X_test, Y_test))

          Logistics Regression: 0.8333333333333334
          Support Vector Machine: 0.8333333333333334
          Decision tree: 0.8333333333333334
          K Nearest Neighbors: 0.8333333333333334
```

Confusion Matrix



Conclusions

Most Interesting Take-Aways:

- The SVM, KNN and Logistics Regression models have the best prediction accuracy
- KSC LC-39A has the best success rate

Thank you!