IBM Developer
SKILLS NETWORK

Winning Space Race with Data Science

<Todoran>
<02.02.2024>

Table of contents

- Abstract
- Introduction
- Methodology
- Results
- Conclusion
- Appendix

Abstract

- Data was gathered to answer the question of what determines the success and the costs of a launch?
- Through API Requests and webscsraping we obtained a dataframe of requested information
- Data was cleaned and organised through wrangling
- EDA and SQL Analysis showed that success appears to be related to the number of flights, with a negative correlation between Payload Mass and outcome and appears to be higher for specific orbits
- The ML Models used for classifying and predicting outcome were Logistic Regression, Support Vector Machine, Decision Tree and K Nearest Neighbors and had the same accuracy, namely 0.8333333333333334
- A Dashboard with 2 charts with user-interaction for dinamic visualisation, to show the total successful launches count for all sites or for a specific launch site and the correlation between payload and launch success .This way the viewer has the possibility to explore in detail the relationships between launch site, payload and outcome
- An interactive Folim map shows Launch sites, with launch number and outcomes, also showing that all Launch Sites are near to the coastline, with good connection possibilities (railway, highway, airport), but not too near to inhabited areas
- **Point most important:** using predictive analysis one can estimate costs and find ways to improve outcome

Introduction

- Project background: what determines the success and the costs of a launch? Has anyone given an useful answer?
- SpaceX It is the only private company ever to return a spacecraft from low-earth orbit, which it first accomplished in December 2010. SpaceX advertises Falcon 9 rocket launches on its website with a cost of 62 million dollars wheras other providers cost upward of 165 million dollars each, much of the savings is because Space X can reuse the first stage. So they might have the answer....
- Spacex Data shows that if we can determine if the first stage will land, we can determine the cost of a launch.
- Are there other predictors of landing succes and/or of costs of a launch?
- Are launch sites in close proximity to railways, highways, coastline? yes
- Do launch sites keep certain distance away from cities? yes

Section 1

Methodology

Methodology

Executive Summary

- Data collection methodology:

  API-Requests and Webscraping

- Data wrangling:

  id and dealing with missing/null data

  extract data about launches, create outcome label and determine succes rate

- Perform exploratory data analysis (EDA) using visualization and SQL
- Perform interactive visual analytics using Folium and Plotly Dash
- Perform predictive analysis using classification models LR, SVM, Tree and KNN

Data Collection

- There were 2 modalities:
- API Request -> Parse Data -> Filter Data -> data Wrangling - dealing with missing data
- Webscraping: Request the page from its URL -> extract HTML table -> parse Table
- Dataframes resulted:

```
# Show the head of the dataframe
data2.head()
```

	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

```
df= pd.DataFrame({ key:pd.Series(value) for key, value in launch_dict.items() })
df.head()
```

	Flight No.	Launch site	Payload	Payload mass	Orbit	Customer	Launch outcome	Version Booster	Booster landing	Date
0	1	CCAFS	CCAFS	0	LEO	SpaceX	Success\n	F9 v1.0B0003.1	Failure	4 June 2010
1	2	CCAFS	CCAFS	0	LEO	NASA	Success	F9 v1.0B0004.1	Failure	8 December 2010
2	3	CCAFS	CCAFS	525 kg	LEO	NASA	Success	F9 v1.0B0005.1	Not attempted\n	22 May 2012
3	4	CCAFS	CCAFS	4,700 kg	LEO	NASA	Success\n	F9 v1.0B0006.1	No attempt	8 October 2012

Data Collection – SpaceX API

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

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

Check the content of the response

```
[17]: print(response.content)
```

b'[{"fairings":{"reused":false,"recovery_attempt":false,"recovered":false,"ships":[]},"links":{"patch":{"small":"https://images2.imgbox.com/94/f2/NN6Ph45r_o.png","large":"https://images2.imgbox.com/5b/02/QcxHUb5V_o.png"},"reddit":{"campaign":null,"launch":null,"media":null,"recovery":null},"flickr":{"small":[],"original":[]},"presskit":null,"webcast":"https://www.youtube.com/watch?v=0a_00nJ_Y88","youtube_id":"0a_00nJ_Y88","article":"https://www.space.com/2196

```
static_json_url='https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS032
```

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

```
response.status_code
```

200

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

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

```
# Lets take a subset of our dataframe keeping only the featu
data = data[['rocket', 'payloads', 'launchpad', 'cores', 'fl

# We will remove rows with multiple cores because those are
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
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 datatyp
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)]
```

```
#Global variables
BoosterVersion = []
PayloadMass = []
Orbit = []
LaunchSite = []
Outcome = []
Flights = []
GridFins = []
Reused = []
Legs = []
LandingPad = []
Block = []
```

```
# Call getLaunchSite
getLaunchSite(data)

# Call getPayloadData
getPayloadData(data)

# Call getCoreData
getCoreData(data)
```

```
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 laun

```
# Create a data from launch_dict
data2 = pd.DataFrame(launch_dict)
```

```
data_falcon9 = data2[data2['BoosterVersion']== 'Falcon 9']
```

```
mean = data_falcon9["PayloadMass"].astype("float").mean()
# Replace the np.nan values with its mean value
data_falcon9["PayloadMass"].replace (np.nan, mean, inplace = True)
data_falcon9.isnull().sum()
```

```
data_falcon9.to_csv('dataset_part_1.csv', index=False)
```

- Get API response-> transform it into Json file and then Dataframe -> use the API again to get information about the launches -> get variables -> apply functions to clean data -> construct dataset -> filter Dataframe -> replace missing values -> export into new csv
- further reference: https://github.com/ioana-t07/Applied-Data-Science-Capstone/blob/main/jupyter-labs-spacex-data-collection-api.ipynb

Data Collection – Scraping

```
response = requests.get(static_url).text
```

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

```
# Use BeautifulSoup() to create a BeautifulSoup objec
soup =BeautifulSoup(response)
```

```
html_tables = soup.find_all('table')
```

Starting from the third table is our target table contains the actual launch records.

```
# Let's print the third table and check its content
first_launch_table = html_tables[2]
```

```
# 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')
```

Starting from the third table is our target table contains the actual launch records.

```
# Let's print the third table and check its content
first_launch_table = html_tables[2]
```

```
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

search = first_launch_table.find_all("th")
for th in search:
    name = extract_column_from_header(th)
    if name is not None and len(name) > 0:
        column_names.append(name)
```

```
launch_dict= dict.fromkeys(column_names)

# Remove an irrelvant column
del launch_dict['Date and time ( )']

# Let's initial the launch_dict with each value to be an empty list
launch_dict['Flight No.'] = []
launch_dict['Launch site'] = []
launch_dict['Payload'] = []
launch_dict['Payload mass'] = []
launch_dict['Orbit'] = []
launch_dict['Customer'] = []
launch_dict['Launch outcome'] = []
# Added some new columns
launch_dict['Version Booster']=[]
launch_dict['Booster landing']=[]
launch_dict['Date']=[]
launch_dict['Time']=[]
```

```
extracted_row = 0
#Extract each table
for table_number,table in enumerate(soup.find_all('table',"wikitable plainrowheaders collapsible")):
   # get table row
    for rows in table.find_all("tr"):
        #check to see if first table heading is as number corresponding to launch a number
        if rows.th:
            if rows.th.string:
                flight_number=rows.th.string.strip()
                flag=flight_number.isdigit()
        else:
            flag=False
        #get table element
        row=rows.find_all('td')
        #if it is number save cells in a dictonary
        if flag:
            extracted_row += 1
            # Flight Number value
            # TODO: Append the flight_number into launch_dict with key `Flight No.`
            datatimelist=date_time(row[0])
            launch_dict['Flight No.'].append(flight_number)
            #print(flight_number)


            # Date value
            # TODO: Append the date into launch_dict with key `Date`
            date = datatimelist[0].strip(',')
            launch_dict['Date'].append(date)
            #print(date)

            # Time value
            # TODO: Append the time into launch_dict with key `Time`
            time = datatimelist[1]
            launch_dict['Time'].append(time)
            #print(time)

            # Booster version
            # TODO: Append the bv into launch_dict with key `Version Booster`
            bv=booster_version(row[1])
            if not(bv):
                bv=row[1].a.string
            print(bv)
            launch_dict['Version Booster'].append(bv)

            # Launch Site
            # TODO: Append the bv into launch_dict with key `Launch Site`
            launch_site = row[2].a.string
            #print(launch_site)
            launch_dict['Launch site'].append(launch_site)

            # Payload
            # TODO: Append the payload into launch_dict with key `Payload`
            payload = row[3].a.string
            #print(payload)
            launch_dict['Payload'].append(launch_site)
```

- Request the Falcon9 Launch page from its URL ->

  Extract the right table ->Extract all column/variable names from the HTML table header-> Create a data frame by parsing the launch HTML tables

- further reference: https://github.com/ioana-t07/Applied-Data-Science-Capstone/blob/main/jupyter-labs-webscraping%20(1).ipynb

Data Wrangling

```
df.isnull().sum()/len(df)*100

FlightNumber        0.000000
Date                0.000000
BoosterVersion      0.000000
PayloadMass         0.000000
```

```
df.dtypes

FlightNumber        int64
Date               object
BoosterVersion     object
PayloadMass       float64
```

```
df['LaunchSite'].value_counts()

CCAFS SLC 40    55
KSC LC 39A      22
VAFB SLC 4E     13
Name: LaunchSite, dtype: int64
```

```
df['Orbit'].value_counts()

GTO      27
ISS      21
VLEO     14
PO        9
LEO       7
SSO       5
MEO       3
ES-L1     1
HEO       1
SO        1
GEO       1
Name: Orbit, dtype: int64
```

```
# landing_outcomes = values on Outcome column
landing_outcomes = df['Outcome'].value_counts()
landing_outcomes

True ASDS      41
None None      19
True RTLS      14
False ASDS      6
True Ocean      5
False Ocean     2
None ASDS       2
False RTLS      1
Name: Outcome, dtype: int64
```

We create a set of outcomes where the second stage did not land successfully:

```
bad_outcomes=set(landing_outcomes.keys()[[1,3,5,6,7]])
bad_outcomes

{'False ASDS', 'False Ocean', 'False RTLS', 'None ASDS', 'None None'}

bad_outcomes1=landing_outcomes.keys()[[1,3,5,6,7]]
```

```
landing_class = []

for element in df['Outcome']:
    if element in bad_outcomes:
        landing_class.append(0)
    else:
        landing_class.append(1)

landing_class
```

We can use the following line of code to determine the success rate:

```
df["Class"].mean()

0.6666666666666666
```

```
df.to_csv("dataset_part_2.csv", index=False)
```

- *Data processing - EDA: id null and type data -> Calculate the number of launches on each site -> Calculate the number and occurrence of each orbit -> Calculate the number and occurence of mission outcome of the orbits -> Create a landing outcome label from Outcome column -> determine the success rate -> export into new csv*
- *further reference*: https://github.com/ioana-t07/Applied-Data-Science-Capstone/blob/main/labs-jupyter-spacex-Data%20wrangling%20(1).ipynb

EDA with Data Visualization

- Plotted charts to explore different variables relationships:
- A Scatterplot to see how the FlightNumber (indicating the continuous launch attempts) and Payload variables would affect the launch outcome
- A catplot to see the relationship between Flight Number and Launch Site
- A catplot to see the relationship between Payload and Launch Site
- A Barplot to visualize the relationship between success rate and each orbit type
- A catplot to see the relationship between FlightNumber and Orbit type
- A catplot to see the relationship between Payload and Orbit type
- A lineplot to see the launch success yearly trend
- further reference: https://github.com/ioana-t07/Applied-Data-Science-Capstone/blob/main/jupyter-labs-eda-dataviz.ipynb.jupyterlite%20(1).ipynb

EDA with SQL - Queries

- Display the names of the unique launch sites in the space mission
- Display 5 records where launch sites begin with the string 'CCA'
- Display the total payload mass carried by boosters launched by NASA (CRS)
- Display average payload mass carried by booster version F9 v1.1
- List the date when the first succesful landing outcome in ground pad was acheived
- List the names of the boosters which have success in drone ship and have payload mass greater than 4000 but less than 6000
- List the total number of successful and failure mission outcomes
- List the names of the booster_versions which have carried the maximum payload mass
- List the records which will display the month names, failure landing_outcomes in drone ship ,booster versions, launch_site for the months in year 2015
- 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
- further reference: https://github.com/ioana-t07/Applied-Data-Science-Capstone/blob/main/jupyter-labs-eda-sql-coursera_sqllite%20(1).ipynb

Build an Interactive Map with Folium

```python
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
```

	Launch Site	Lat	Long
0	CCAFS LC-40	28.562302	-80.577356
1	CCAFS SLC-40	28.563197	-80.576820
2	KSC LC-39A	28.573255	-80.646895
3	VAFB SLC-4E	34.632834	-120.610745

```python
nasa_coordinate_CCAFS_LC = [28.562302, -80.577356]
nasa_coordinate_CCAFS_SLC = [28.563197, -80.576820]
nasa_coordinate_KSC_LC = [28.573255, -80.646895]
nasa_coordinate_VAFB_SLC = [34.632834, -120.610745]
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
circleCCAFS_LC = folium.Circle(nasa_coordinate_CCAFS_LC, radius=1000, color='#d35400', fill=True).add_child(folium.Popup('
markerCCAFS_LC = folium.map.Marker(nasa_coordinate_CCAFS_LC,
     icon=DivIcon(icon_size=(20,20), icon_anchor=(0,0),
        html='<div style="font-size: 12; color:#d35400;"><b>%s</b></div>' % 'CCAFS LC-40'))
circleCCAFS_SLC = folium.Circle(nasa_coordinate_CCAFS_SLC, radius=1000, color='#d35400', fill=True).add_child(folium.Popup
markerCCAFS_SLC = folium.map.Marker(nasa_coordinate_CCAFS_SLC,
     icon=DivIcon(icon_size=(20,20), icon_anchor=(0,0),
        html='<div style="font-size: 12; color:#d35400;"><b>%s</b></div>' % 'CCAFS SLC-40'))
circleKSC_LC = folium.Circle(nasa_coordinate_KSC_LC, radius=1000, color='#d35400', fill=True).add_child(folium.Popup('Laun
markerKSC_LC = folium.map.Marker(nasa_coordinate_KSC_LC,
     icon=DivIcon(icon_size=(20,20), icon_anchor=(0,0),
        html='<div style="font-size: 12; color:#d35400;"><b>%s</b></div>' % 'KSC LC-39A'))
circleVAFB_SLC = folium.Circle(nasa_coordinate_VAFB_SLC, radius=1000, color='#d35400', fill=True).add_child(folium.Popup('
markerVAFB_SLC = folium.map.Marker(nasa_coordinate_VAFB_SLC,
     icon=DivIcon(icon_size=(20,20), icon_anchor=(0,0),
        html='<div style="font-size: 12; color:#d35400;"><b>%s</b></div>' % 'VAFB SLC-4E'))
circles = [circleCCAFS_LC, circleCCAFS_SLC, circleKSC_LC, circleVAFB_SLC]
markers = [markerCCAFS_LC, markerCCAFS_SLC, markerKSC_LC, markerVAFB_SLC ]
```

```python
spacex_df['marker_color'] = spacex_df['class']
for x in spacex_df.index:
    if spacex_df.loc[x, 'marker_color'] == 0:
        spacex_df.loc[x, 'marker_color'] = 'red'
    else:
        spacex_df.loc[x, 'marker_color'] = 'green'
spacex_df.tail(5)
# Apply a function to check the value of `class` column
# If class=1, marker_color value will be green
# If class=0, marker_color value will be red
```

	Launch Site	Lat	Long	class	marker_color
51	CCAFS SLC-40	28.563197	-80.57682	0	red
52	CCAFS SLC-40	28.563197	-80.57682	0	red
53	CCAFS SLC-40	28.563197	-80.57682	0	red

```python
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
    coordinates = [record['Lat'], record['Long']]
    marker_cluster = MarkerCluster().add_to(site_map)
    marker = folium.Marker(coordinates, icon=folium.Icon(color='white', icon_color=record['marker_color'])).add_to(marker
    marker_cluster.add_child(marker)

site_map
```

```python
# Add Mouse Position to get the coordinate (Lat, Long) for a mouse
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
```

```python
coordinates = [nasa_coordinate_CCAFS_SLC, coordinate_coast]
lines=folium.PolyLine(locations=coordinates, weight=1)
site_map.add_child(lines)
```

- Find coordinates fos all launch sites -> Create and add folium.Circle and folium.Marker for each launch site on the site map -> create marker for launch outcome -> add them to map for each site -> add mouse position to show coordinates -> Draw a PolyLine between a launch site to the selected coast point -> repeat for all other selected points (city, railway, highway, airport) to show distance to launch sites
- further reference: **https://github.com/ioana-t07/Applied-Data-Science-Capstone/blob/main/lab_jupyter_launch_site_location.jupyterlite.ipynb**

Build a Dashboard with Plotly Dash

- 2 charts with user-interaction (view the chart for all sites or select a specific one, selection of Payload Mass range) were added to the dashboard:

- a pie chart to show the total successful launches count for all sites or for a specific launch site the Success vs. Failed counts

- a scatter chart to show the correlation between payload and launch success, for all sites or the selected one and for a specific payload range

- This way the viewer has the possibility to explore in detail the relationships between launch site, payload and outcome

- further reference: https://github.com/ioana-t07/Applied-Data-Science-Capstone/blob/main/spacex_dash_app%20(1).py

Predictive Analysis (Classification)

```
Y = data['Class'].to_numpy()
Y
```

```
array([0, 0, 0, 0, 0, 0, 1, 1, 0,
       1, 1, 0, 1, 1, 0, 1, 1, 0,
       1, 0, 0, 0, 1, 1, 0, 0, 1,
       1, 0, 1, 1, 1, 1, 0, 1, 0,
       1, 1], dtype=int64)
```

```
X = X.astype(float)
```

```
# students get this
transform = preprocessing.StandardScaler()
X = transform.fit(X).transform(X)
X
```

```
array([[-1.71291154e+00, -1.94814463e-16, -6.53912840e-01, .
        -8.35531692e-01,  1.93309133e+00, -1.93309133e+00],
       [-1.67441914e+00, -1.19523159e+00, -6.53912840e-01, .
        -8.35531692e-01,  1.93309133e+00, -1.93309133e+00],
       [-1.63592675e+00, -1.16267307e+00, -6.53912840e-01, .
```

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.2, random_state = 2)
```

```
parameters ={"C":[0.01,0.1,1],'penalty':['l2'], 'solver':['lbfgs']
lr=LogisticRegression()
logreg_cv = GridSearchCV(lr, parameters, cv = 10)
logreg_cv.fit(X_train, Y_train)
```

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

```
svm_cv = GridSearchCV(svm, parameters, cv = 10)
svm_cv.fit(X_train, Y_train)
```

```
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()
```

```
tree_cv = GridSearchCV(tree, parameters, cv = 10)
tree_cv.fit(X_train, Y_train)
```

```
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()
```

```
knn_cv = GridSearchCV(KNN, parameters, cv = 10)
knn_cv.fit(X_train, Y_train)
```

- Create a NumPy array from the column Class -> Standardize the data in X -> split the data X and Y into training and test data -> create the LR Model – create the SVM Model – create the Decision Tree Model – create the LR Model -> evaluate Models : all are equally accurate
- further reference: https://github.com/ioana-t07/Applied-Data-Science-Capstone/blob/main/SpaceX_Machine_Learning_Prediction_Part_5.jupyterlite.ipynb

Results

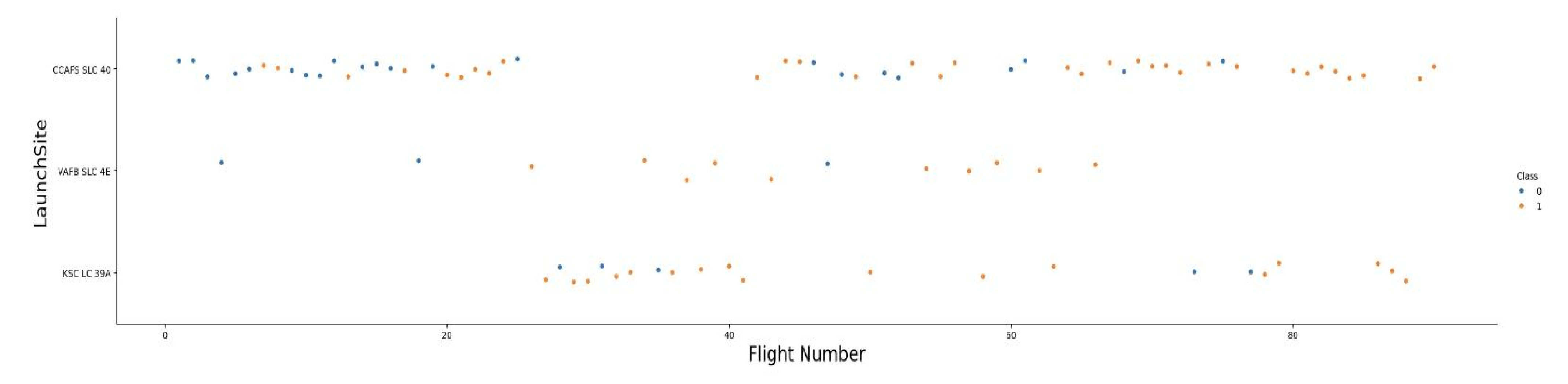
- Exploratory data analysis results
- Interactive analytics demo in screenshots
- Predictive analysis results

Section 2

Insights drawn from EDA

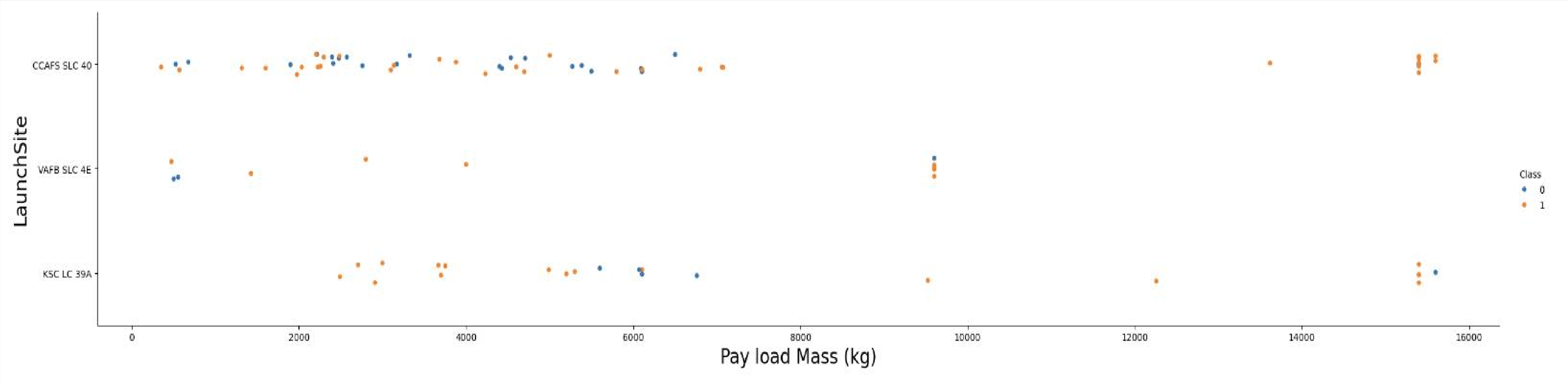
Flight Number vs. Launch Site

- Visualised as scatter plot with 2 classes: landing of the first stage failure (0, blue color) and success(1, orange color)
- As we can see, with the increase in number of launches, increases the number of succesful landings



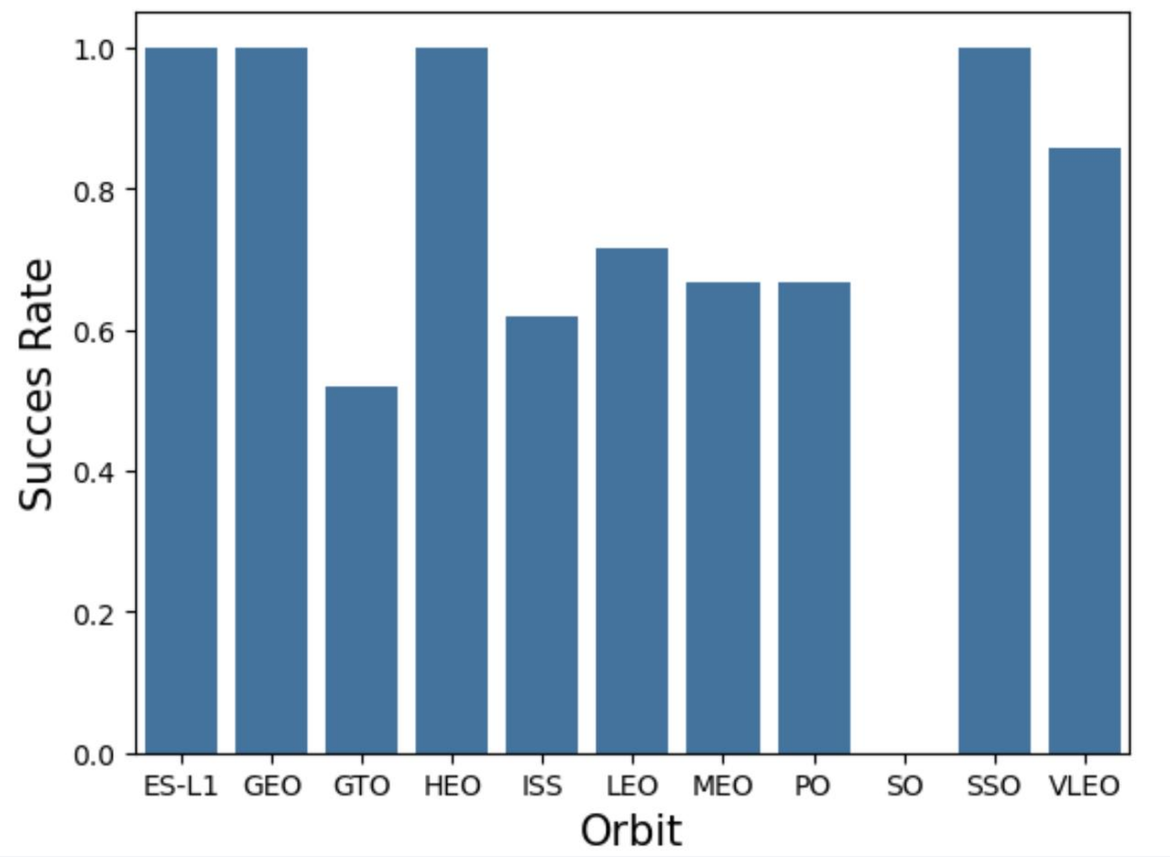
Payload vs. Launch Site

- Visualised as scatter plot with 2 classes: landing of the first stage failure (0, blue color) and success(1, orange color)
- The payload mass reveals it's importance, as it seems the more massive the payload, the less likely the first stage will return
- By VAFB-SLC launchsite there are no rockets launched for heavypayload mass(greater than 10000)



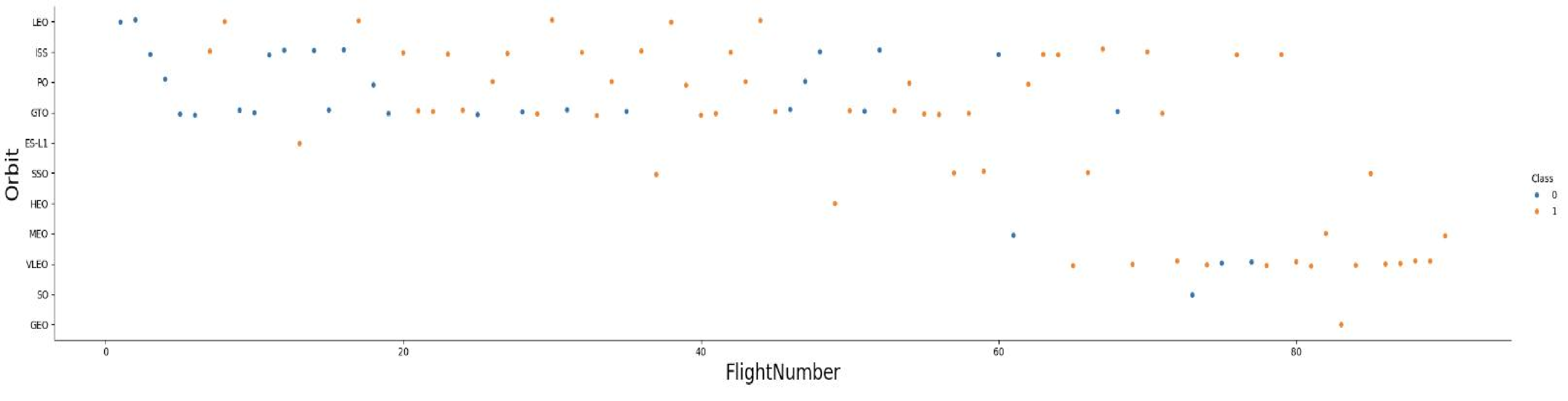
Success Rate vs. Orbit Type

- The best succes rate appears for:
  - the geocentric high-altitude orbits(GEO, HEO)
  - at Lagrange points(ES-L1)
  - the sun-syncronous orbit(SSO)
- the worst appears to be the transfer orbit(GTO)
- see appendix for orbit types and description



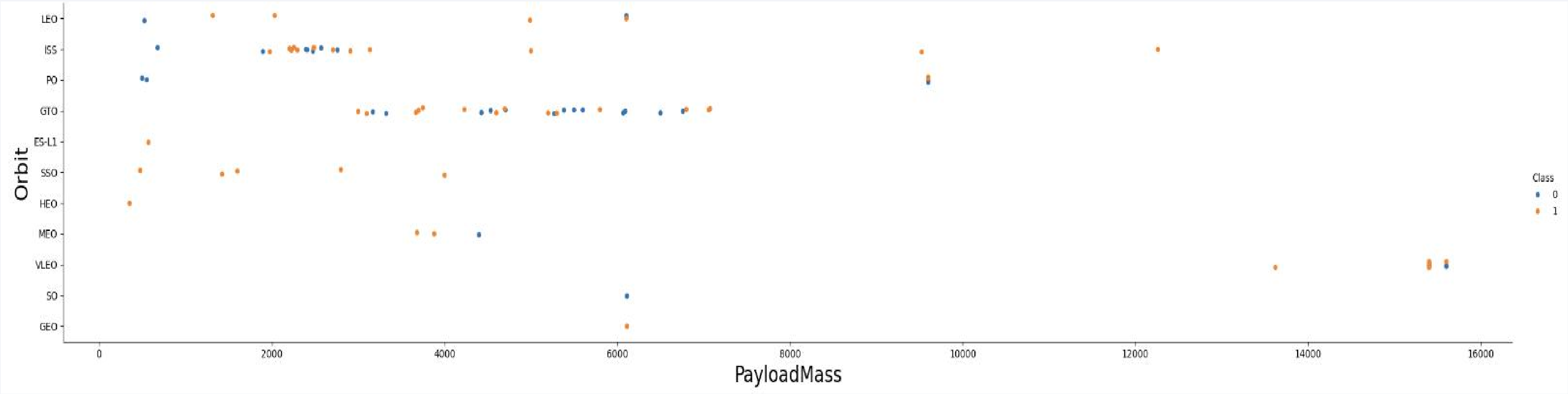
Flight Number vs. Orbit Type

- Visualised as scatter plot with 2 classes: landing of the first stage failure (0, blue color) and success(1, orange color)
- The Success appears to be related to the number of flights, more evident in the LEO orbit, however there seems to be no relationship between flight number when in GTO orbit



Payload vs. Orbit Type

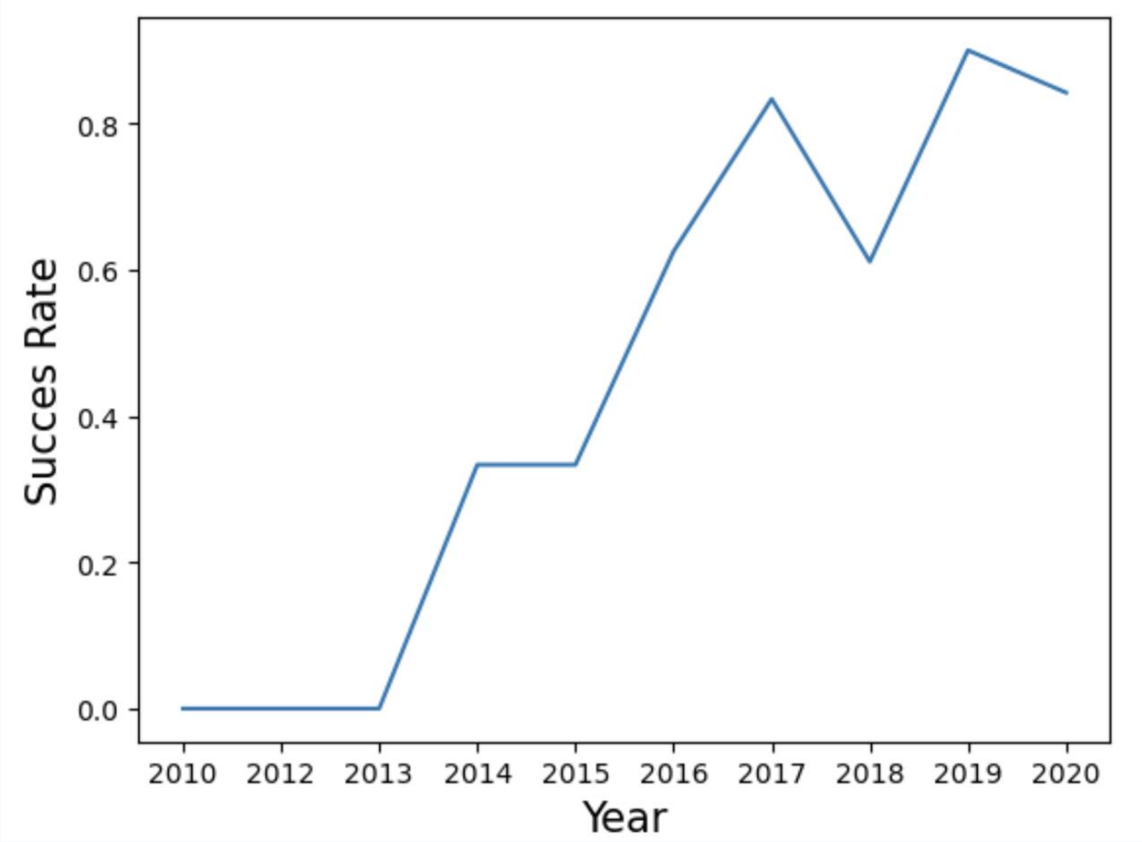
- Visualised as scatter plot with 2 classes: landing of the first stage failure (0, blue color) and success(1, orange color)
- heavy payloads don't come on high-altitude orbits
- With heavy payloads the successful landing rates are more for Polar,LEO and ISS, while for GTO there`s no significant difference



Launch Success Yearly Trend

- The line chart of yearly average success rate with values from 0.0 to 1.0

- we can see that the sucess rate began improving in 2013 and kept increasing till 2020



All Launch Site Names

- The names of the unique launch sites are as shown:
- CCAFS LC-40 and CCAFS SLC-40: Cape Canaveral Space Launch Complex 40
- VAFB SLC-4E: Vandenberg Air Force Base Space Launch Complex 4E (SLC-4E)
- KSC LC 39A: Kennedy Space Center Launch Complex 39A

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

Launch Site Names Begin with 'CCA'

- Query example for 5 records where launch sites begin with `CCA` demonstrating available information and extent of details

Date	Time (UTC)	Booster_Version	Launch_Site	Payload	PAYLOAD_MASS__KG_	Orbit	Customer	Mission_Outcome	Landing_Outcome
2010-06-04	18:45:00	F9 v1.0 B0003	CCAFS LC-40	Dragon Spacecraft Qualification Unit	0	LEO	SpaceX	Success	Failure (parachute)
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	Failure (parachute)
2012-05-22	7:44:00	F9 v1.0 B0005	CCAFS LC-40	Dragon demo flight C2	525	LEO (ISS)	NASA (COTS)	Success	No attempt
2012-10-08	0:35:00	F9 v1.0 B0006	CCAFS LC-40	SpaceX CRS-1	500	LEO (ISS)	NASA (CRS)	Success	No attempt
2013-03-01	15:10:00	F9 v1.0 B0007	CCAFS LC-40	SpaceX CRS-2	677	LEO (ISS)	NASA (CRS)	Success	No attempt

Total Payload Mass

- The total payload carried by boosters from NASA, calculated as 45596 units
- The Total Payload Mass is the combined mass of all payloads carried by the flights

sum("PAYLOAD_MASS__KG_")
45596

Average Payload Mass by F9 v1.1

- The average payload mass carried by booster version F9 v1.1

avg("PAYLOAD_MASS__KG_")
2928.4

First Successful Ground Landing Date

- The date of the first successful landing outcome on ground pad

min("Date")
2015-12-22

Successful Drone Ship Landing with Payload between 4000 and 6000

- The names of boosters which have successfully landed on drone ship and had payload mass greater than 4000 but less than 6000 units

Booster_Version
F9 FT B1022
F9 FT B1026
F9 FT B1021.2
F9 FT B1031.2

Total Number of Mission Outcomes

- The total number of successful and failure mission outcomes

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

Boosters Carried Maximum Payload

- These boosters have carried the maximum payload mass

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

2015 Launch Records

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

Month	Landing_Outcome	Booster_Version	Launch_Site
01	Failure (drone ship)	F9 v1.1 B1012	CCAFS LC-40
04	Failure (drone ship)	F9 v1.1 B1015	CCAFS LC-40

Ranking Outcomes Between 2010-06-04 and 2017-03-20

- The Ranking for 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
- Most attempts were by drone ship, but ground pad brought most sucess

Date	Landing_Outcome	Count
2016-04-08	Success (drone ship)	12
2012-05-22	No attempt	12
2015-12-22	Success (ground pad)	8
2015-01-10	Failure (drone ship)	5
2014-04-18	Controlled (ocean)	4
2013-09-29	Uncontrolled (ocean)	2
2010-06-04	Failure (parachute)	2
2015-06-28	Precluded (drone ship)	1

Section 3

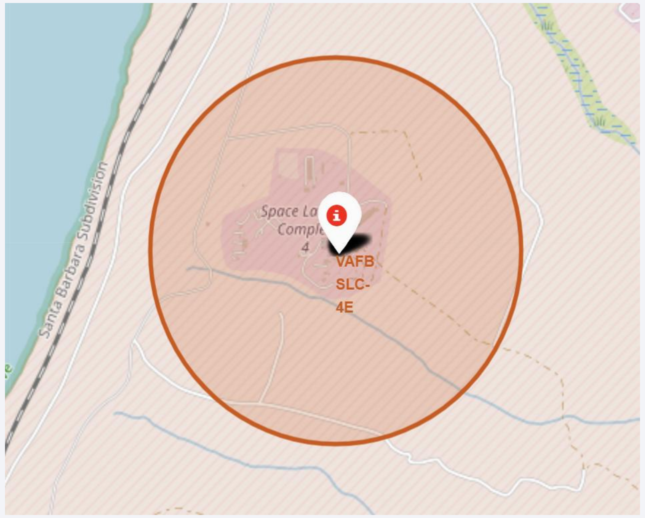
Launch Sites Proximities Analysis

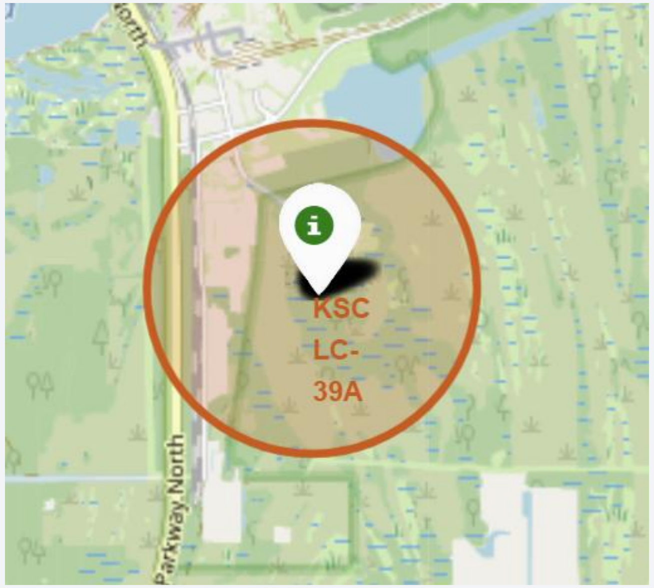
All launch site locations markers on a global map

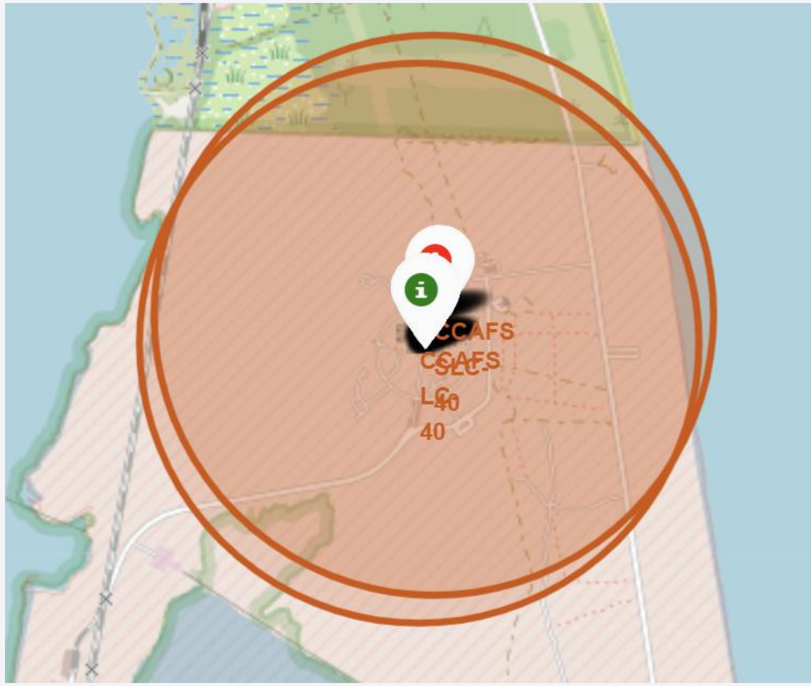
- The launch sites'location is in the USA, mainly near to Cape Canaveral
- All of them are near to the coastline, with good connection possibilities, but not too near to inhabited areas

Launch outcomes of every site location

- The color-labeled launch outcomes for every site also marked on the map
- The launch site with a high launch success is KSC LC-39A

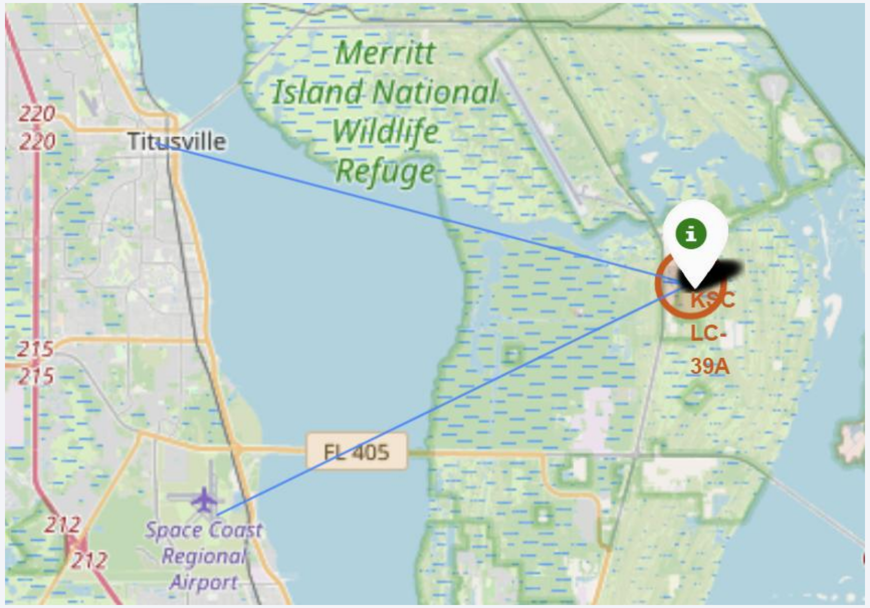






The selected Launch sites and their proximities: coastline, railway, city, highway, airport

- nearest coastline is to launch-site CCAFS SLC-40 : 0.8636358651673924 KM
- nearest railway is to launch-site CCAFS SLC-40 : 1.2305877082226684KM
- nearest city is to launch-site KSC LC-39A : 16.844967380865 84 KM
- nearest highway is to launch-site KSC LC-39A : 0.848905914731 7743 KM
- nearest airport is to launch-site KSC LC-39A : 16.096133642186423 KM

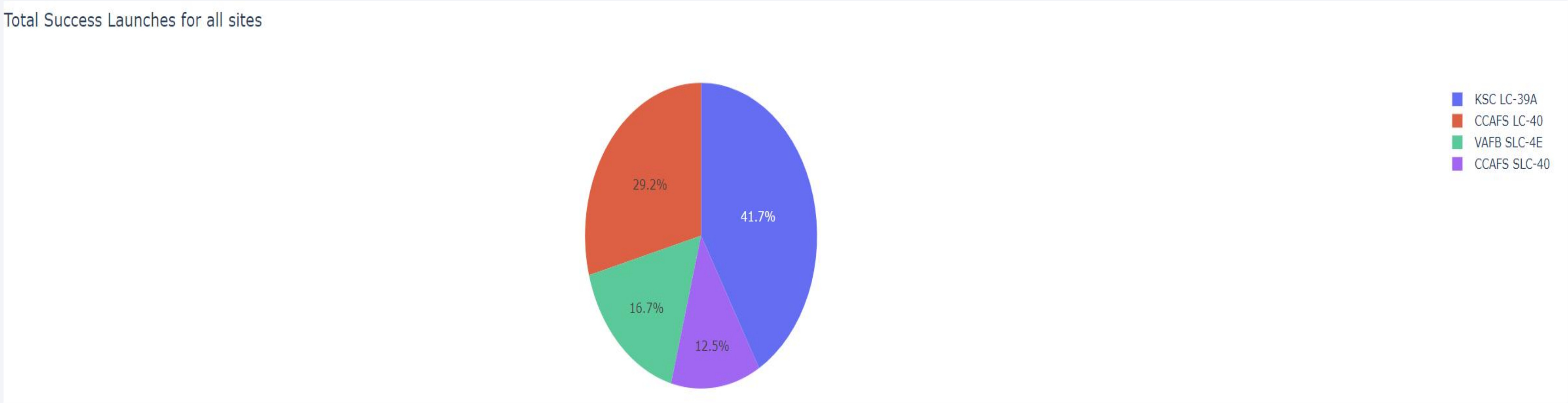


Section 4

Build a Dashboard with Plotly Dash

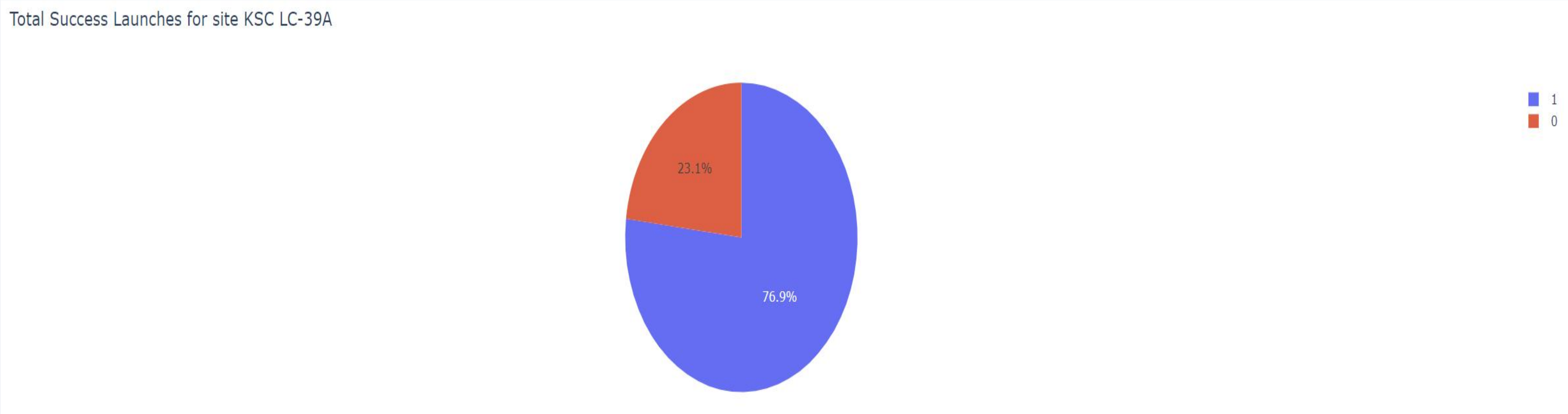
Launch success count for all sites

- The launch site with highest launch success was KSC LC-39A
- The lowest count was recorded by CCAFS SLC-40



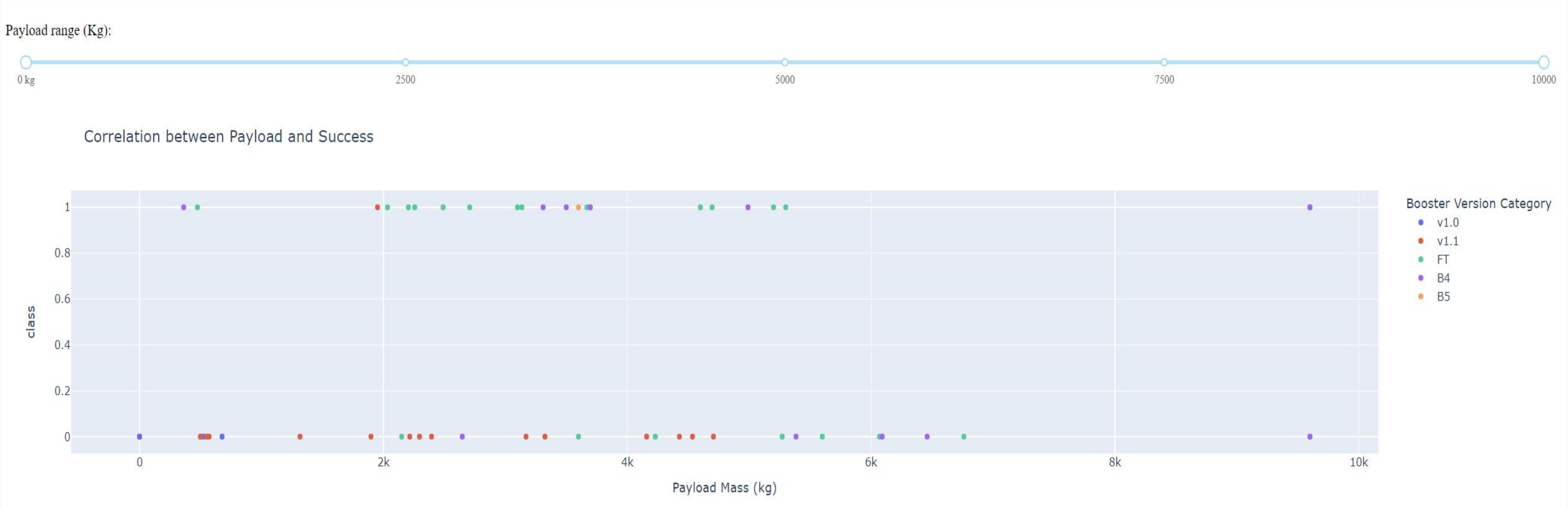
The highest launch success ratio pro site

- The launch site with highest launch success ratio was also KSC LC-39A
- Succes rate was by 76,9%



Payload vs. Launch Outcome

- Shown as scatter plot for all sites, with different payload selected in the range slider
- The best performance is reached with the Booster FT and worst with v1.1
- Once again the negative correlation between Payload Mass and outcome



Section 5

Predictive Analysis (Classification)

Classification Models (1)

Logistic Regression

```
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)
```

```
GridSearchCV(cv=10, estimator=LogisticRegression(),
             param_grid={'C': [0.01, 0.1, 1], 'penalty': ['l2'],
                         'solver': ['lbfgs']})
```

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

```
svm_cv = GridSearchCV(svm, parameters, cv = 10)
svm_cv.fit(X_train, Y_train)
```

```
GridSearchCV(cv=10, estimator=SVC(),
             param_grid={'C': array([1.00000000e-03, 3.16227766e-02, 1.00000000e+0
0, 3.16227766e+01,
       1.00000000e+03]),
                         'gamma': array([1.00000000e-03, 3.16227766e-02, 1.00000000
e+00, 3.16227766e+01,
       1.00000000e+03]),
                         'kernel': ('linear', 'rbf', 'poly', 'rbf', 'sigmoid')})
```

```
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
```

Support Vector Machine

Classification Models (2)

K Nearest Neighbors

```
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()
```

```
knn_cv = GridSearchCV(KNN, parameters, cv = 10)
knn_cv.fit(X_train, Y_train)
```

```
/lib/python3.11/site-packages/threadpoolctl.py:1019: RuntimeWarning: libc not found. The ctyp
es module in Python 3.11 is maybe too old for this OS.
  warnings.warn(
GridSearchCV(cv=10, estimator=KNeighborsClassifier(),
             param_grid={'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute'],
                         'n_neighbors': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
                         'p': [1, 2]})
```

```
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()
```

```
tree_cv = GridSearchCV(tree, parameters, cv = 10)
tree_cv.fit(X_train, Y_train)
```

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

```
tuned hpyerparameters :(best parameters)  {'criterion': 'entropy', 'max_depth': 16, 'max_feat
ures': 'sqrt', 'min_samples_leaf': 4, 'min_samples_split': 5, 'splitter': 'random'}
accuracy : 0.8892857142857142
```

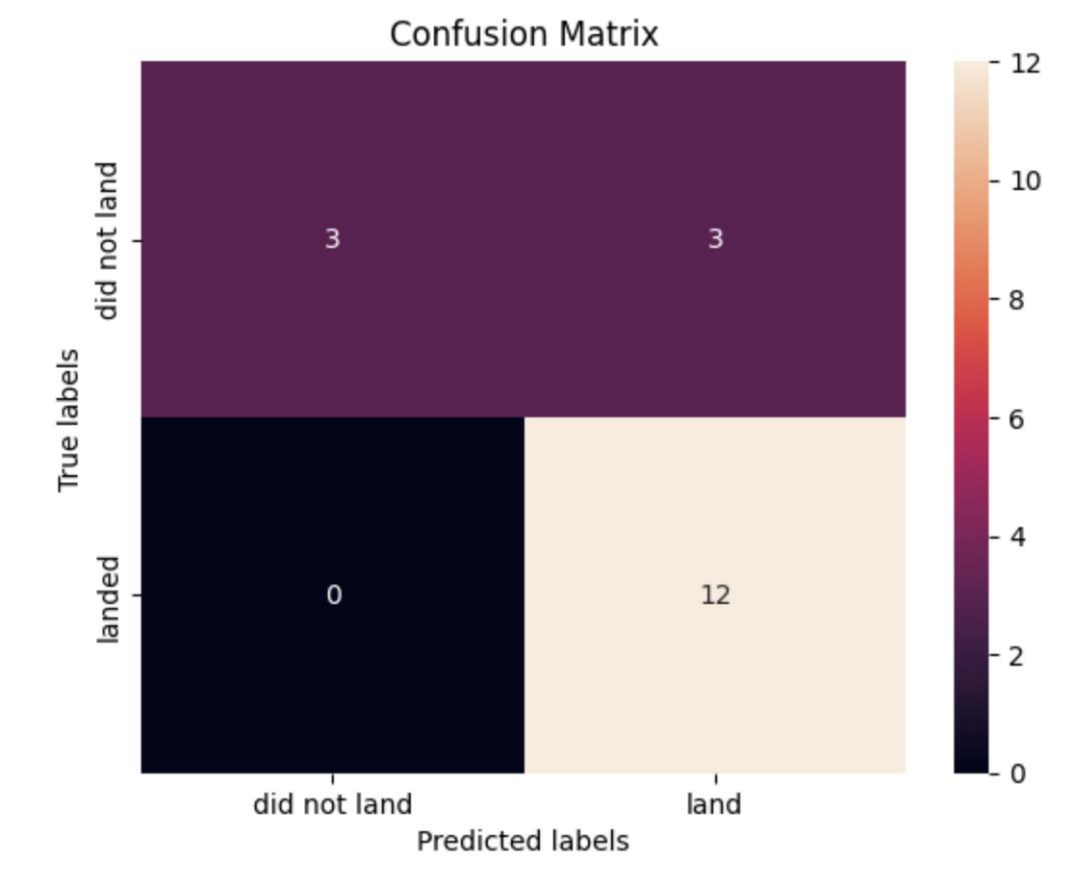
Decision Tree

Classification Accuracy

- all models had the same accuracy, namely

  0.8333333333333334

- the confusion matrix shows the correct prediction of failed landings with an error only by the few failed landings predicted as sucess (3)



Conclusions

- Point 1 All Launch Sites are near to the coastline, with good connection possibilities (railway, highway, airport), but not too near to inhabited areas
- Point 2 The Success appears to be related to the number of flights, with the exception of flights in GTO orbit
- Point 3 There's a negative correlation between Payload Mass and outcome
- Point 4 The best succes rate appears for the geocentric high-altitude orbits, at Lagrange points and the sun-syncronous orbit
- **Point most important:** using predictive analysis one can estimate costs and find ways to improve outcome

Appendix

- a short description of the common orbit types from Data wrangling Lab

- the Dashboard outputs for all launch sites as quick refference

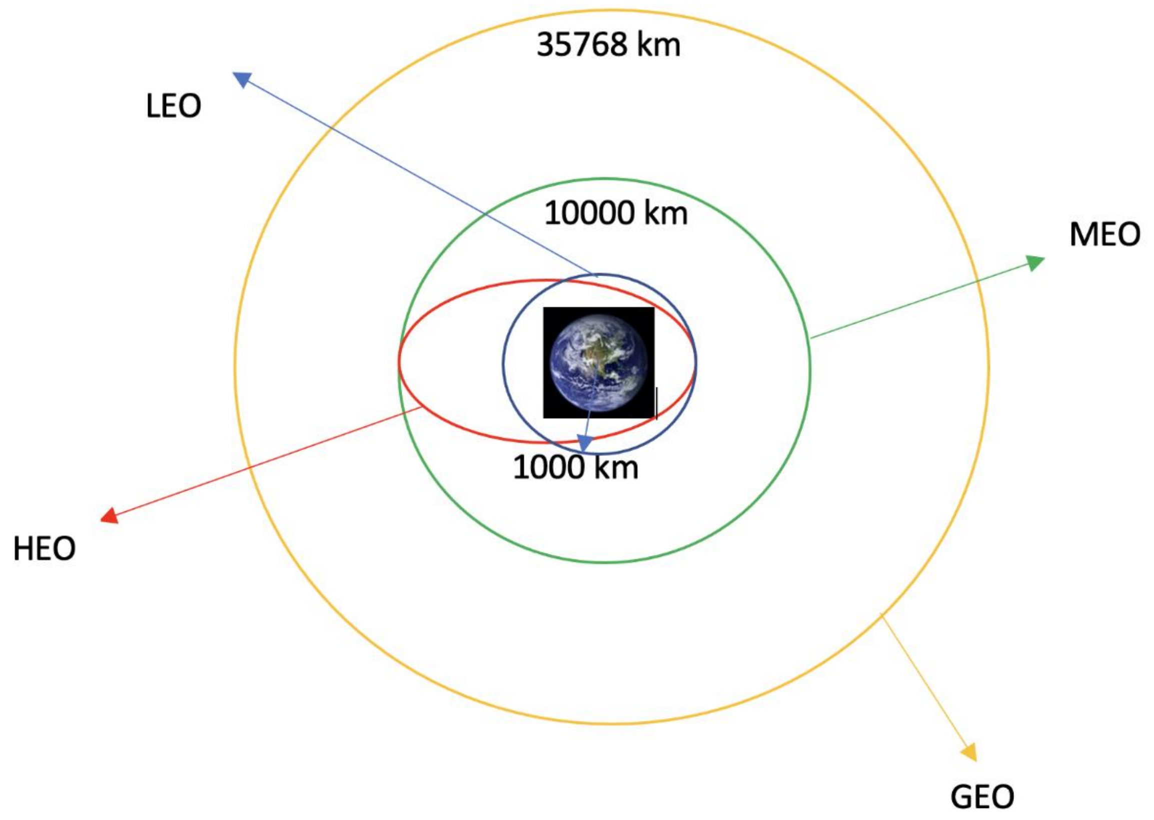
Short description of common orbit types(1)

- **LEO** Low Earth orbit (LEO)is an Earth-centred orbit with an altitude of 2,000 km (1,200 mi) or less (approximately one-third of the radius of Earth),[1] or with at least 11.25 periods per day (an orbital period of 128 minutes or less) and an eccentricity less than 0.25.[2] Most of the manmade objects in outer space are in LEO [1].
- **VLEO** Very Low Earth Orbits (VLEO) can be defined as the orbits with a mean altitude below 450 km. Operating in these orbits can provide a number of benefits to Earth observation spacecraft as the spacecraft operates closer to the observation[2].
- **GTO** A geosynchronous orbit is a high Earth orbit that allows satellites to match Earth's rotation. Located at 22,236 miles (35,786 kilometers) above Earth's equator, this position is a valuable spot for monitoring weather, communications and surveillance. Because the satellite orbits at the same speed that the Earth is turning, the satellite seems to stay in place over a single longitude, though it may drift north to south," NASA wrote on its Earth Observatory website [3]. It's also used for transfering satelites between orbits
- **SSO (or SO)** It is a Sun-synchronous orbit also called a heliosynchronous orbit is a nearly polar orbit around a planet, in which the satellite passes over any given point of the planet's surface at the same local mean solar time [4]
- **ES-L1** At the Lagrange points the gravitational forces of the two large bodies cancel out in such a way that a small object placed in orbit there is in equilibrium relative to the center of mass of the large bodies. L1 is one such point between the sun and the earth [5] .

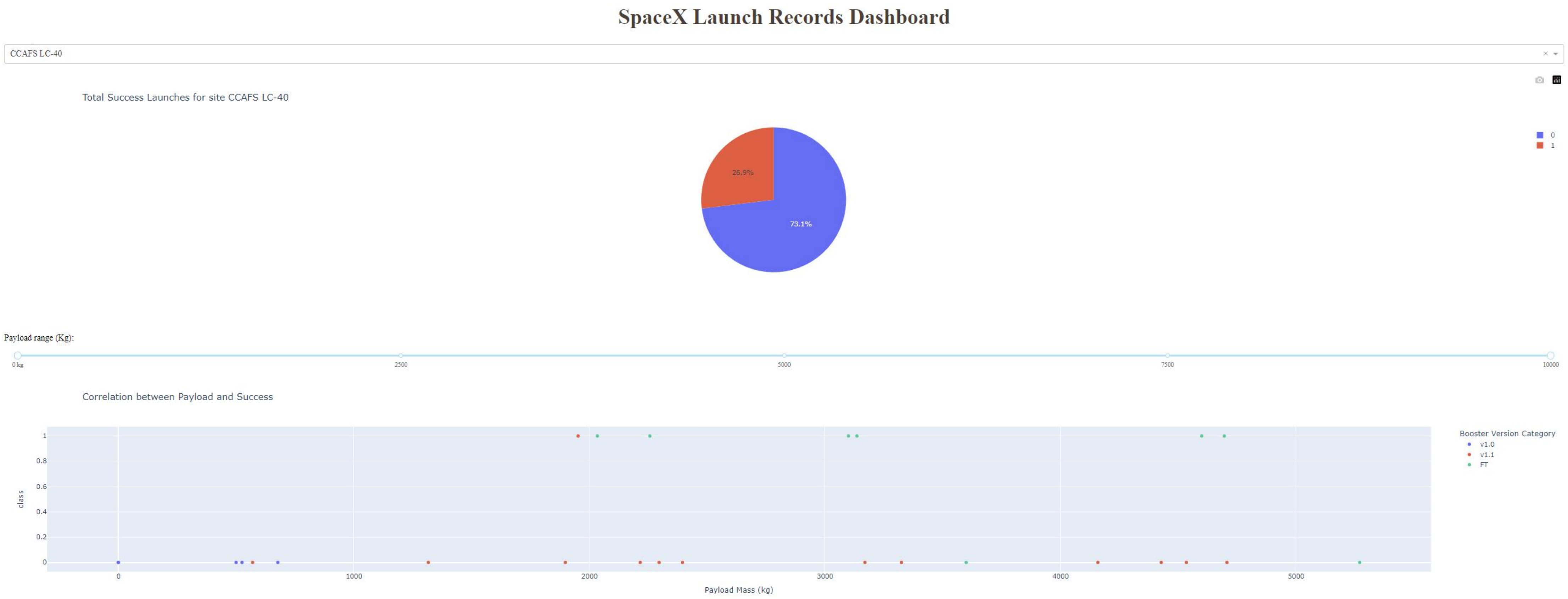
Short description of common orbit types(2)

- **HEO** A highly elliptical orbit, is an elliptic orbit with high eccentricity, usually referring to one around Earth [6].
- **ISS** A low Earth orbit where there is also the modular space station, a multinational collaborative project between five participating space agencies: NASA (United States), Roscosmos (Russia), JAXA (Japan), ESA (Europe), and CSA (Canada) [7]
- **MEO** Geocentric orbits ranging in altitude from 2,000 km (1,200 mi) to just below geosynchronous orbit at 35,786 kilometers (22,236 mi). Also known as an intermediate circular orbit. These are "most commonly at 20,200 kilometers (12,600 mi), or 20,650 kilometers (12,830 mi), with an orbital period of 12 hours [8]
- **HEO** Geocentric orbits above the altitude of geosynchronous orbit (35,786 km or 22,236 mi) [9]
- **GEO** It is a circular geosynchronous orbit 35,786 kilometres (22,236 miles) above Earth's equator and following the direction of Earth's rotation [10]
- **PO** It is one type of satellites in which a satellite passes above or nearly above both poles of the body being orbited (usually a planet such as the Earth [11]

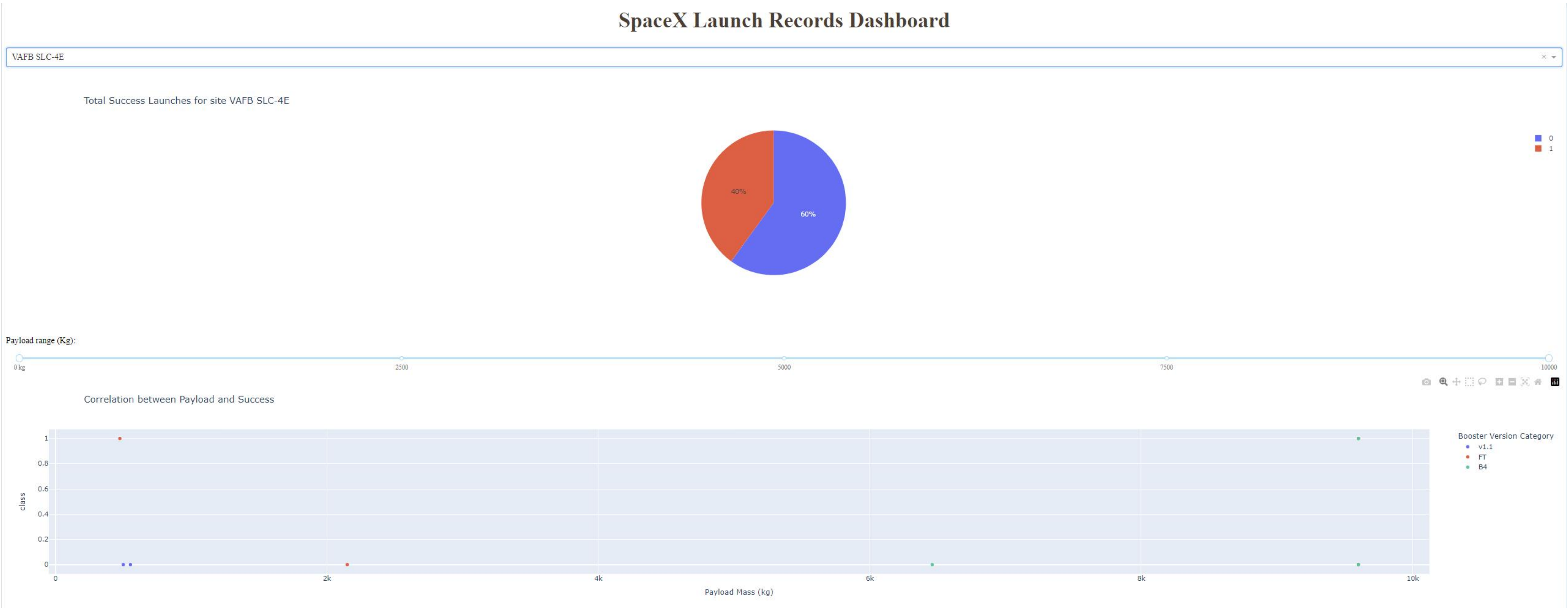
Visual graph of some orbit types



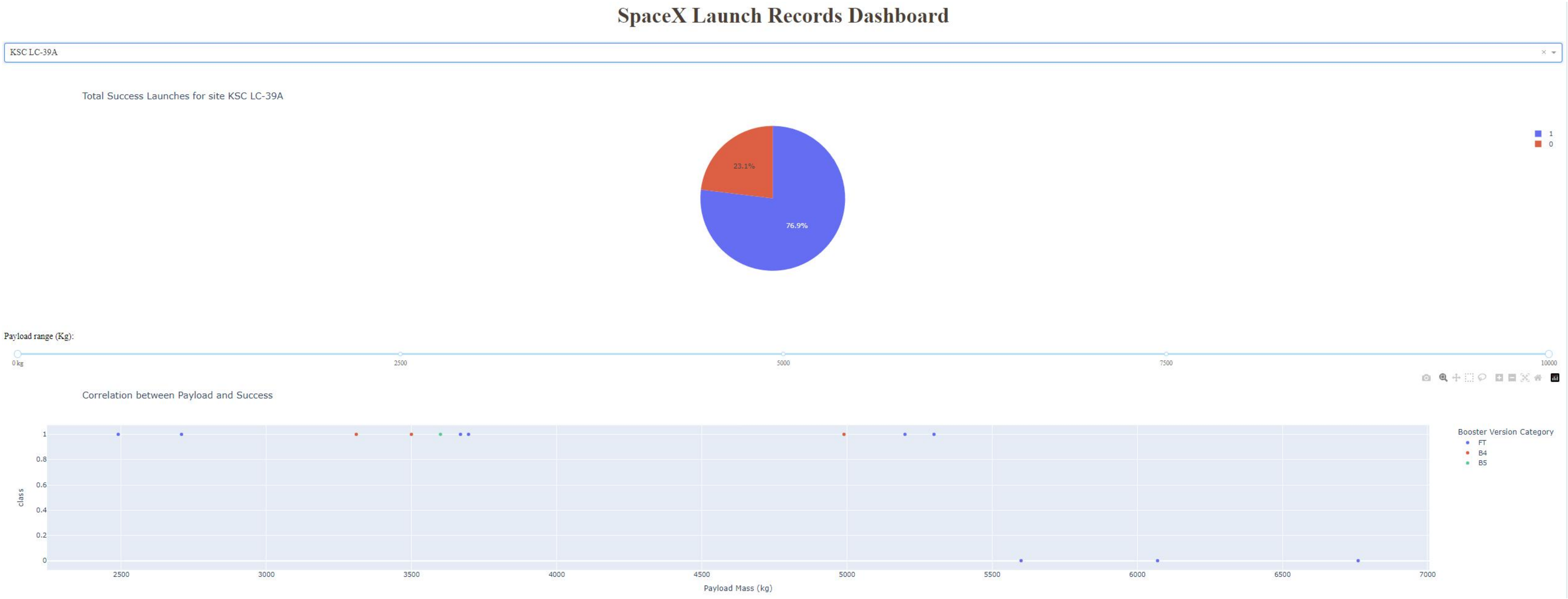
SpaceX Launch Records Dashboard for Site CCAFS LC-40



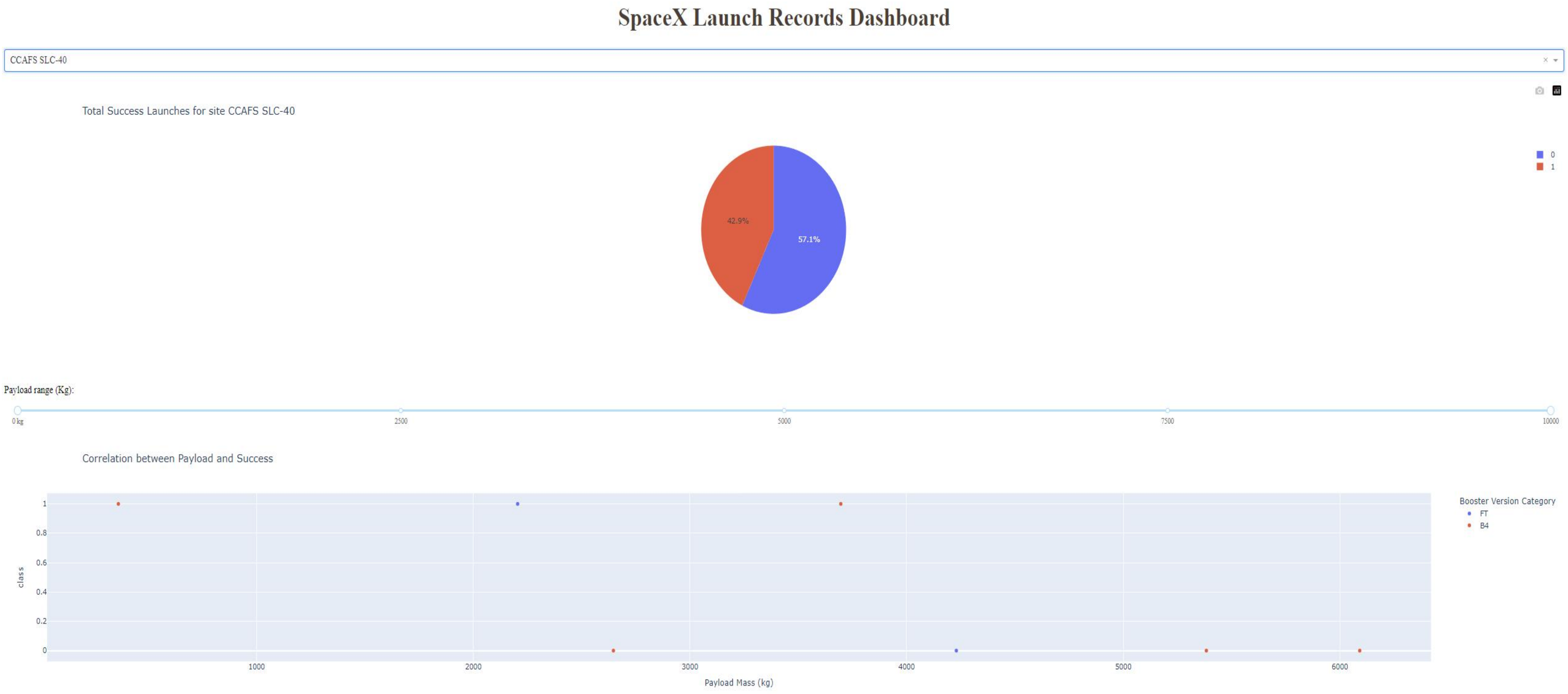
SpaceX Launch Records Dashboard for Site VAFB SLC-4E



SpaceX Launch Records Dashboard for Site KSC LC-39A



SpaceX Launch Records Dashboard for Site CCAFS SLC-40



Thank you!