IBM Capstone Project

Filip Jojovic

02/2024

IBM Developer

SKILLS NETWORK

OUTLINE

- Executive Summary
- Introduction
- Recommendation
    - Findings & Implications
- Key results
- Methodology
    - EDA (SQL + Pandas)
    - Visualization – Charts
    - Dashboard
    - Predictive Analysis

EXECUTIVE SUMMARY

- The objective is to predict our Falcon 9 first stage landing success to support future bid decisions on projects.
- Predicting first stage landing success is crucial for determining launch cost, as they are the largest driver.
- The question is, how the landing success will behave in future?
- The recommendation is to bid lower prices in future tenders, due to following findings from the data analysis:
  - The probability of launch failure is reducing with every Falcon 9 launch.
  - Payload masses are increasing, as the learning curve is growing. The payload masses are not correlating with the success or failure of the launch.

INTRODUCTION

- The goal is to predict the success of the Falcon 9's first stage landing in order to gain insight into whether bids for upcoming tenders should be placed higher or lower.
- Importance: Predicting first stage landing success is crucial for determining launch cost.
- Cost Comparison: SpaceX's Falcon 9 launches cost $62 million, significantly lower than competitors' launches priced at $165 million, mainly due to the ability to reuse the first stage.

IBM Developer
SKILLS NETWORK

RECOMMENDATION

- The recommendation from the data analysis is to bid lower prices in future call for tenders for rocket launches, due to following reasons:
  - The probability of launch failure is reducing with every Falcon 9 launch.
  - Payload masses are increasing, as the learning curve is growing. The payload masses are not correlating with the success or failure of the launch.
- The risk remains, that the call for tender is expecting to launch from a site and/or to an orbit with a low success rate leading to higher launch costs.
- Next Steps: An assessment of the individual launch site and orbit for the specific call for tender.

KEY INSIGHTS

- Launch failure decreased over time, leading to steadily lower launch cost over time. This is mainly due to learning curve growth.
- Payload masses are increasing, as learning curve is growing. The payload masses do not have an immediate impact on the success or failure of the launch.
- There are several launch sites and orbits of high failure and on the other side, high success. This is also applicable for the booster version.
  - Highest orbit success rate: ES-L1, GEO, HEO, SSO
  - Lowest orbit success rate: GTO, SO
  - KSC LC-39A with the highest success to failure rate.
  - CCAFS SLC-40 with the lowest success to failure rate.
- Predictions can be made with all models. However, decision tree modelling resulted in the highest accuracy. Confusion matrix probably has same values for all models, due to the low amount of samples and high euclidic distance between data points.

METHODOLOGY & RESULTS

- Collecting the data via APIs and Webscraping.
- Data pre-processing via data wrangling incl. one-hot encoding.
- Exploratory Data Analysis (EDA) for analysis of data.
- Graphical visualization via Folium and Dash to understand cause- and effect relationships.
- Creation of supervised, but also unsupervised models to predict the launch cost. This includes Logistic Regression, Decision Tree modelling, Supported Vector modelling, K-Nearest-Neighbour

METHODOLOGY: Data collection (1/2)

1. Data collection: Getting Data via an API request.

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

```
[9]: static_json_url='https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/datasets/API_call_spacex_api.json'
```

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

```
[10]: response.status_code
[10]: 200
```

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

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

Using the dataframe `data` print the first 5 rows

```
[15]: # Get the head of the dataframe
data.head(3)
```

2. Filtering data via for the objects of interests.

```
[1]: # Hint data['BoosterVersion']!='Falcon 1'
data_falcon9 = launch_data[launch_data['BoosterVersion'] == 'Falcon 9']
data_falcon9.shape
```

3. Data wrangling: Filling out missing values.

```
[81]: # Calculate the mean value of PayloadMass column
mean_payload = data_falcon9['PayloadMass'].mean()
# Replace the np.nan values with its mean value
data_falcon9['PayloadMass'] = data_falcon9['PayloadMass'].replace(np.nan, mean_payload)
data_falcon9.isnull().sum()

# To CSV, landing pad rows will be removed afterwards
data_falcon9.to_csv('dataset_part_1.csv', index = False)
```

METHODOLOGY: Data collection (2/2)

4. Webscraping: Getting further data via Wikipedia table.

First, let's perform an HTTP GET method to request the Falcon9 Launch HTML page, as an HTTP response.

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

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

```
[8]: # Use BeautifulSoup() to create a BeautifulSoup object from a response text content
     soup = BeautifulSoup(response)
```

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

```
[9]: # Use soup.title attribute
     soup.title
[9]: <title>List of Falcon 9 and Falcon Heavy launches - Wikipedia</title>
```

5. Creating a Dataframe from Wikipedia table content.

We will create an empty dictionary with keys from the extracted column names in the previous task. Later, this dictionary will be converted into a Pandas dataframe

```
[14]: 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']=[]
```

Next, we just need to fill up the `launch_dict` with launch records extracted from table rows.

Usually, HTML tables in Wiki pages are likely to contain unexpected annotations and other types of noises, such as reference links `B0004.1[8]`, missing values `N/A [e]`, inconsistent formatting, etc.

To simplify the parsing process, we have provided an incomplete code snippet below to help you to fill up the `launch_dict`. Please complete the following code snippet with TODOs or you can choose to write your own logic to parse all launch tables:

```
[16]: 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.`
            launch_dict['Flight No.'].append(flight_number)
            #print(flight_number)
            print(flight_number)
            datatimelist=date_time(row[0])
```

Would you like to receive official Jupyter news?

--> Final Dataframe for analysis.

	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins	Reused	Legs	LandingPad
1	1	2010-06-04	Falcon 9	6123.547647058824	LEO	CCSFS SLC 40	None None	1	False	False	False	
2	2	2012-05-22	Falcon 9	525.0	LEO	CCSFS SLC 40	None None	1	False	False	False	
3	3	2013-03-01	Falcon 9	677.0	ISS	CCSFS SLC 40	None None	1	False	False	False	
4	4	2013-09-29	Falcon 9	500.0	PO	VAFB SLC 4E	False Ocean	1	False	False	False	
5	5	2013-12-03	Falcon 9	3170.0	GTO	CCSFS SLC 40	None None	1	False	False	False	
6	6	2014-01-06	Falcon 9	3325.0	GTO	CCSFS SLC 40	None None	1	False	False	False	
7	7	2014-04-18	Falcon 9	2296.0	ISS	CCSFS SLC 40	True Ocean	1	False	False	True	
8	8	2014-07-14	Falcon 9	1316.0	LEO	CCSFS SLC 40	True Ocean	1	False	False	True	
9	9	2014-08-05	Falcon 9	4535.0	GTO	CCSFS SLC 40	None None	1	False	False	False	
10	10	2014-09-07	Falcon 9	4428.0	GTO	CCSFS SLC 40	None None	1	False	False	False	
11	11	2014-09-21	Falcon 9	2216.0	ISS	CCSFS SLC 40	False Ocean	1	False	False	False	

METHODOLOGY: Data visualization (1/2)

1. Usage of Folium to create an interactive map for visualization purposes.

```
[1]: import piplite
     await piplite.install(['folium'])
     await piplite.install(['pandas'])
```

```
[2]: import folium
     import pandas as pd
```

```
[3]: # Import folium MarkerCluster plugin
     from folium.plugins import MarkerCluster
     # Import folium MousePosition plugin
     from folium.plugins import MousePosition
     # Import folium DivIcon plugin
     from folium.features import DivIcon
```

2. Usage of Folium objects to to add markers to data points.

We could use `folium.Circle` to add a highlighted circle area with a text label on a specific coordinate. For example,

```
[7]: # Create a blue circle at NASA Johnson Space Center's coordinate with a popup label showing its name
     circle = folium.Circle(nasa_coordinate, radius=1000, color='#d35400', fill=True).add_child(folium.Popup('NASA John
     # Create a blue circle at NASA Johnson Space Center's coordinate with a icon showing its name
     marker = folium.map.Marker(
         nasa_coordinate,
         # Create an icon as a text label
         icon=DivIcon(
             icon_size=(20,20),
             icon_anchor=(0,0),
             html='<div style="font-size: 12; color:#d35400;"><b>%s</b></div>' % 'NASA JSC',
             )
         )
     site_map.add_child(circle)
     site_map.add_child(marker)
```

3. Usage of Folium Marker & Polyline to create lines and distance indication between points of interests.

```
orlando_lat= 28.53627
orlando_lon= -81.4
launch_site_lat= 28.562
launch_site_lon= -80.57717
distance_orlando = calculate_distance(launch_site_lat, launch_site_lon, orlando_lat, orlando_lon)

city_marker = folium.Marker(location = [28.53627, -81.4],
                            icon=DivIcon(icon_size = (20,20),
                                         icon_anchor = (0,0),
                                          html='<div style="font-size: 40; color:#d35400;"><b>%s</b></div>' % "{:10.2f}
                                         )
                           )
lines=folium.PolyLine(locations=[[launch_site_lat, launch_site_lon], [orlando_lat, orlando_lon]], weight=1)
site_map.add_child(lines)
site_map.add_child(city_marker)
```

METHODOLOGY: Data visualization (2/2)

1. Addition of Dropdown menu to Dash

```
dcc.Dropdown(id='site-dropdown', options=[
    {'label': 'All Sites', 'value': 'ALL'},
    {'label': 'CCAFS LC-40', 'value': 'CCAFS LC-40'},
    {'label': 'VAFB SLC-4E', 'value': 'VAFB SLC-4E'},
    {'label': 'KSC LC-39A', 'value': 'KSC LC-39A'},
    {'label': 'CCAFS SLC-40', 'value': 'CCAFS SLC-40'}
],
            value='ALL',
            placeholder="SELECT",
            searchable=True,
            clearable=True
            ),
```

2. Creation of callback functions for pie chart.

```
@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(data_frame=filtered_df, values='class', names='Launch Site',title='Success/Failure per launch site')
    else:
        filtered_df = filtered_df[filtered_df['Launch Site'] == entered_site]
        grouped_df = filtered_df.groupby('class').size().reset_index(name = 'count')
        grouped_df['class'] = grouped_df['class'].astype(str)
        fig = px.pie(data_frame=grouped_df,
                     values='count', names='class',
                     title='Success/Failure per launch site')
    return fig
```

3. Addition of a range slider for visualization adjustment.

```
html.P("Payload range (Kg):"),
dcc.RangeSlider(id='payload-slider', min=0, max=10000, step=1000, marks={0: '0', 100: '100'},
                value=[min_payload, max_payload]),
```

4. Creation of callback functions for scatter plotting.

```
@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 get_scatter(entered_site, payload_range):
    filtered_df = spacex_df
    min_payload, max_payload = payload_range
    if entered_site == 'ALL':
        scatter_fig = px.scatter(filtered_df[(filtered_df['Payload Mass (kg)'] >= min_payload) &
                                             (filtered_df['Payload Mass (kg)'] <= max_payload)],
                                 x='Payload Mass (kg)', y='class',
                                 color='Booster Version Category',
                                 title='Relationship Payload Mass and success rate')
    else:
        scatter_fig = px.scatter(filtered_df[(filtered_df['Launch Site'] == entered_site) &
                                             (filtered_df['Payload Mass (kg)'] >= min_payload) &
                                             (filtered_df['Payload Mass (kg)'] <= max_payload)],
                                 x='Payload Mass (kg)', y='class',
                                 color='Booster Version Category',
                                 title='Relationship Payload Mass and success rate')
    return scatter_fig
```

METHODOLOGY: Predictive analytics

1. Defining the "class"/target variable.

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

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

2. Standardizing predictors.

```
[19]: # students get this
      transform = preprocessing.StandardScaler()
      X = transform.fit(X).transform(X.astype(float))
      X
```

```
[19]: array([[-1.71291154e+00, -1.34780356e-17, -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, ...,
              -8.35531692e-01,  1.93309133e+00, -1.93309133e+00],
             ...,
             [ 1.63592675e+00,  1.99100483e+00,  3.49060516e+00, ...,
               1.19684269e+00, -5.17306132e-01,  5.17306132e-01],
             [ 1.67441914e+00,  1.99100483e+00,  1.00389436e+00, ...,
               1.19684269e+00, -5.17306132e-01,  5.17306132e-01],
             [ 1.71291154e+00, -5.19213966e-01, -6.53912840e-01, ...,
              -8.35531692e-01, -5.17306132e-01,  5.17306132e-01]])
```

3. Creation and fitting of several models. Finding of best model parameters with GridSearchCV.

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

```
[29]: 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)
```

```
[29]: ▸ GridSearchCV
      ▸ estimator: LogisticRegression
        ▸ LogisticRegression
```

4. Evaluation of model accuracy via confusion matrix and R2-Score

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

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

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

RESULTS: EDA (SQL)

Task 1

://labs.cognitiveclass.ai/v2/tools/jupyterlab?ulid=ulid-3b2a81d2238ed2815cfd3c00243962e49176a464

:24, 21:00 jupyter-labs-eda-sql-coursera_sqllite

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

```
In [9]: %sql SELECT DISTINCT "Launch_Site" FROM SPACEXTABLE
```

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

Out[9]:

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

Launch site overview

Task 2

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

```
In [21]: %sql SELECT * FROM SPACEXTABLE WHERE "Launch_Site" LIKE 'CCA%' LIMIT 20
```

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

jupyter-labs-eda-sql-coursera_sqllite

[21]:

Date	Time (UTC)	Booster_Version	Launch_Site	Payload	PAYLOAD_MASS__KG_	Orbit	Customer	Mission_Outcome	Landing
2010-06-04	18:45:00	F9 v1.0 B0003	CCAFS LC-40	Dragon Spacecraft Qualification Unit	0	LEO	SpaceX	Success	Failure
2010-12-08	15:43:00	F9 v1.0 B0004	CCAFS LC-40	Dragon demo flight C1, two CubeSats, barrel of	0	LEO (ISS)	NASA (COTS) NRO	Success	Failure

Task 3

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

```
In [20]: %sql SELECT SUM("PAYLOAD_MASS__KG_") as 'TOTAL PAYLOAD NASA' FROM SPACEXTABLE WHERE "Customer" LIKE 'NASA%'
```

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

Out[20]:

TOTAL PAYLOAD NASA
99980

Total payload mass for NASA.

Task 4

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

```
In [29]: %sql SELECT AVG("PAYLOAD_MASS__KG_") AS 'AVERAGE PAYLOAD F9 V1.1' FROM SPACEXTABLE WHERE "Booster_Version" LIKE 'F9
```

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

Out[29]:

AVERAGE PAYLOAD F9 V1.1
2534.6666666666665

Average payload mass.

RESULTS: EDA (SQL)

Task 5

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

Hint:Use min function

s.cognitiveclass.ai/v2/tools/jupyterlab?ulid=ulid-3b2a81d2238ed2815cfd3c00243962e49176a464 7/13

21:00 jupyter-labs-eda-sql-coursera_sqllite

```
In [35]: %sql SELECT MIN("Date") as "EARLIEST DATE FOR SUCCESSFUL LAUNCH" FROM SPACEXTABLE WHERE "Mission_Outcome" is "Succe
 * sqlite:///my_data1.db
Done.
```

Out[35]:

EARLIEST DATE FOR SUCCESSFUL LAUNCH
2010-06-04

First successfull launch

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 [55]: %sql SELECT Booster_Version, PAYLOAD_MASS__KG_ FROM SPACEXTABLE WHERE PAYLOAD_MASS__KG_ < 6000 AND PAYLOAD_MASS__KG
 * sqlite:///my_data1.db
Done.
```

Out[55]:

Booster_Version	PAYLOAD_MASS__KG_
F9 v1.1	4535
F9 v1.1 B1011	4428
F9 v1.1 B1014	4159
F9 v1.1 B1016	4707
F9 FT B1020	5271
F9 FT B1022	4696
F9 FT B1026	4600
F9 FT B1030	5600
F9 FT B1021.2	5300
F9 FT B1032.1	5300
F9 B4 B1040.1	4990
F9 FT B1031.2	5200
F9 B4 B1043.1	5000
F9 FT B1032.2	4230
F9 B4 B1040.2	5384
F9 B5 B1046.2	5800
F9 B5 B1047.2	5300
F9 B5B1054	4400
F9 B5 B1048.3	4850
F9 B5 B1051.2	4200
F9 B5B1060.1	4311
F9 B5 B1058.2	5500
F9 B5B1062.1	4311

Successfull boosters versions with mass between 4000 – 6000kg.

Task 7

List the total number of successful and failure mission outcomes

```
In [76]: %sql SELECT "Mission_Outcome", COUNT(*) FROM SPACEXTABLE GROUP BY "Mission_Outcome"
 * sqlite:///my_data1.db
Done.
```

Out[76]:

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

Success/Failure overview

Task 8

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

```
In [92]: %sql SELECT Booster_Version FROM SPACEXTABLE WHERE PAYLOAD_MASS__KG_ == (SELECT MAX(PAYLOAD_MASS__KG_) FROM SPACEXT
 * sqlite:///my_data1.db
Done.
```

Out[92]:

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

Maximum payload mass boosters.

RESULTS: EDA (SQL)

Task 9

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.

Note: SQLLite does not support monthnames. So you need to use substr(Date, 6,2) as month to get the months and substr(Date,0,5)='2015' for year.

[95]:
```
%sql SELECT substr(Date,6,2) AS "Month", "Landing_Outcome", "Booster_Version", "Launch_Site" FROM SPACEXTABLE WHERE
```

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

gnitiveclass.ai/v2/tools/jupyterlab?ulid=ulid-3b2a81d2238ed2815cfd3c00243962e49176a464 11/

0 jupyter-labs-eda-sql-coursera_sqllite

t[95]:

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

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.

[115_
```
%%sql
SELECT COUNT(*), "Landing_Outcome", "Date"
FROM SPACEXTABLE
WHERE "Date" BETWEEN '2010-06-04' AND '2017-03-20' -- OR: WHERE "Date" >= '2010-06-04' AND "Date" <= '2017-03-20'
GROUP BY "Landing_Outcome"
ORDER BY COUNT(*) DESC
```

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

t[115]:

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

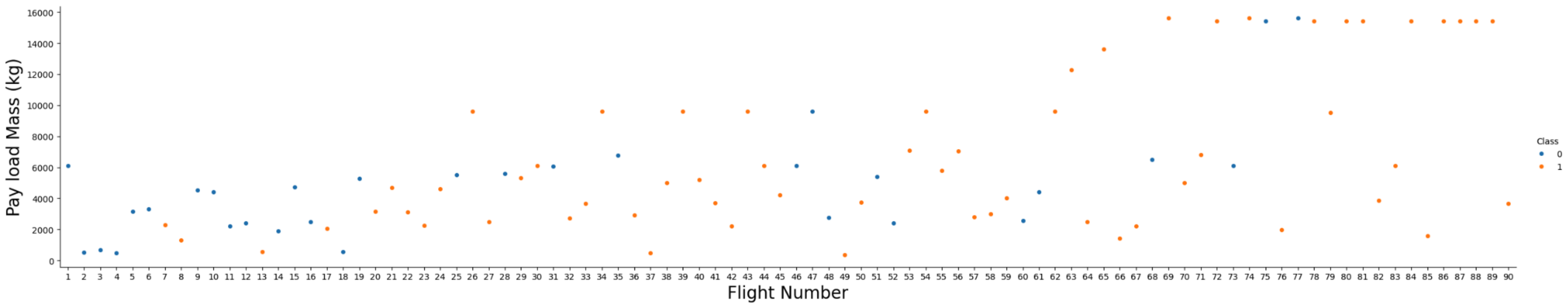
Landing outcomes for targeted date interval.

Results: EDA (Pandas)

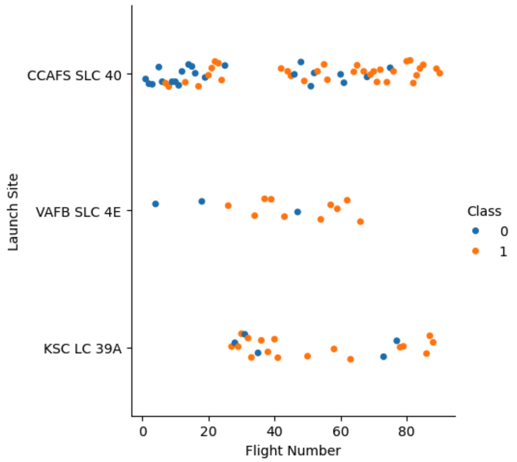
Amber: Success

Blue: Failed

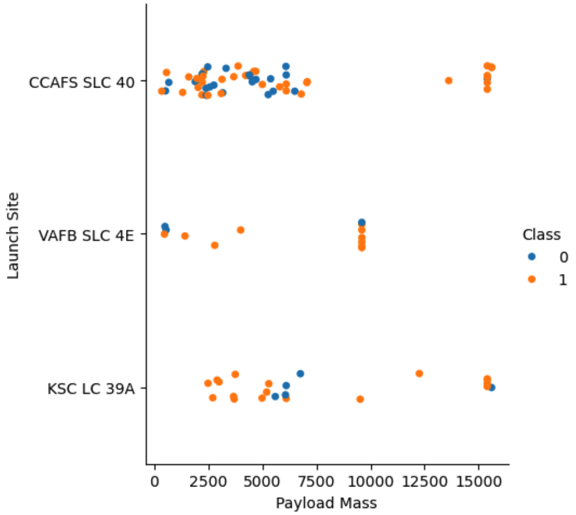
Flight Number over Pay-Load mass (kg) and Launch Site.



- Over time, success increased, failures decreased.



- KSC LC 39A started from 25th flight.
- Highest quantity of flights from CCAFS SLC 40
- First flights are rather unsuccessfull.



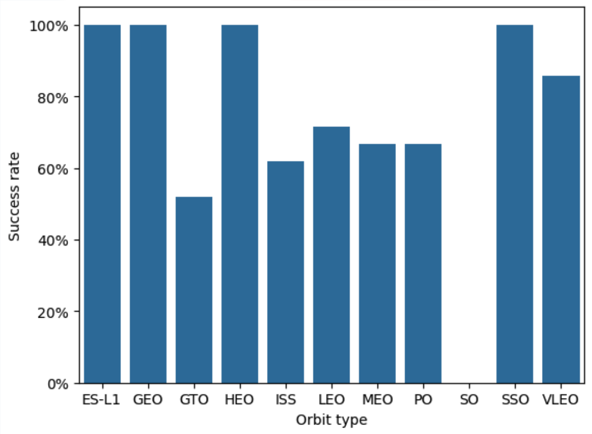
- More low payload launches have failed that high payloads.

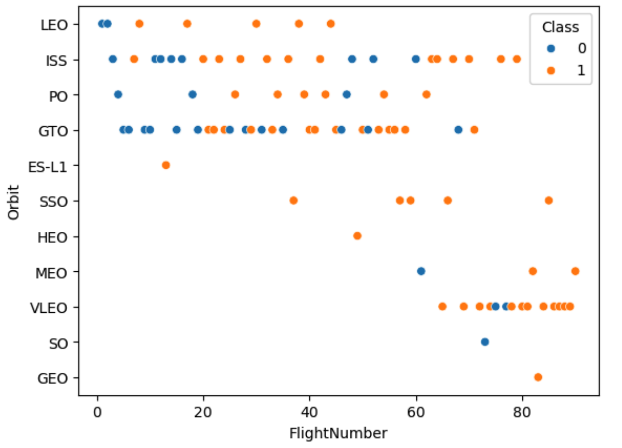
Results: EDA (Pandas)

Amber: Success
Blue: Failed

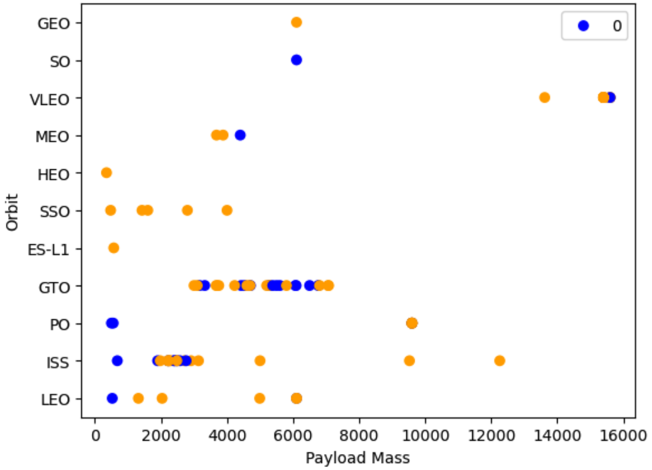
Orbit vs. Success rate vs. Flight Number

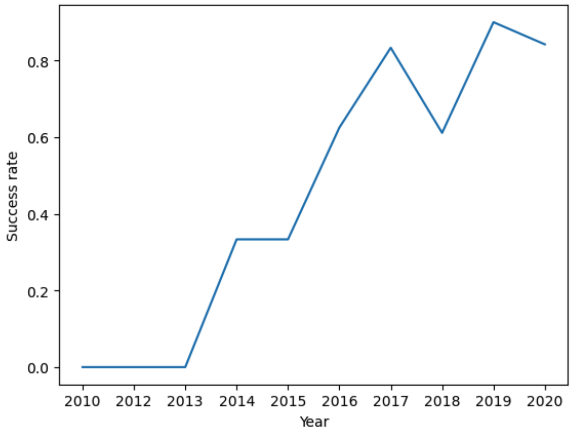
- Highest success rate: ES-L1, GEO, HEO, SSO
- Lowest success rate: GTO, SO





- Success rate increased with flight numbers.
- Top 5 Orbits are LEO, ISS, PO, GTO, VLEO

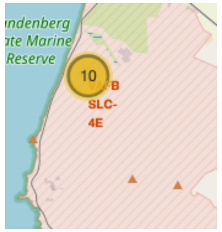




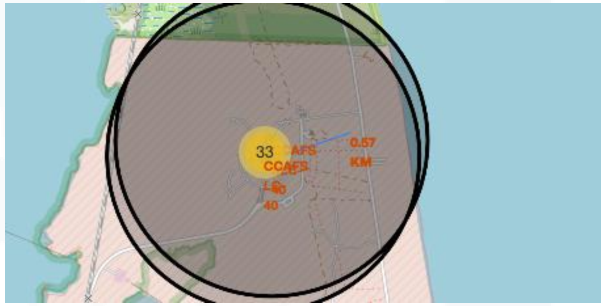
- Success rate increased steadily, except for a downward facing slope in 2017 and 2020.

Results: Visualization with Folium

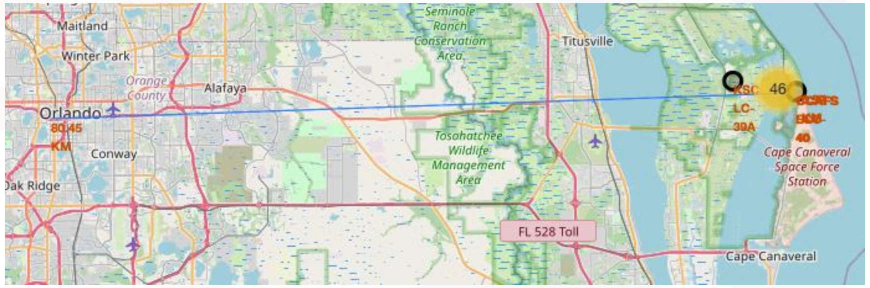
Marker location for successfull and unsuccesfull starts in US.



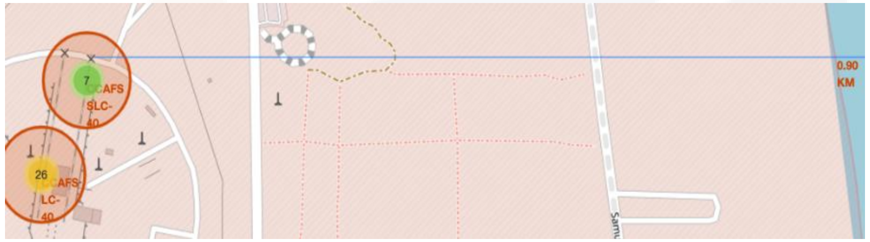
- Distance to nearest main stree t (0.6km).



- Distance to Orlando (80.45km)



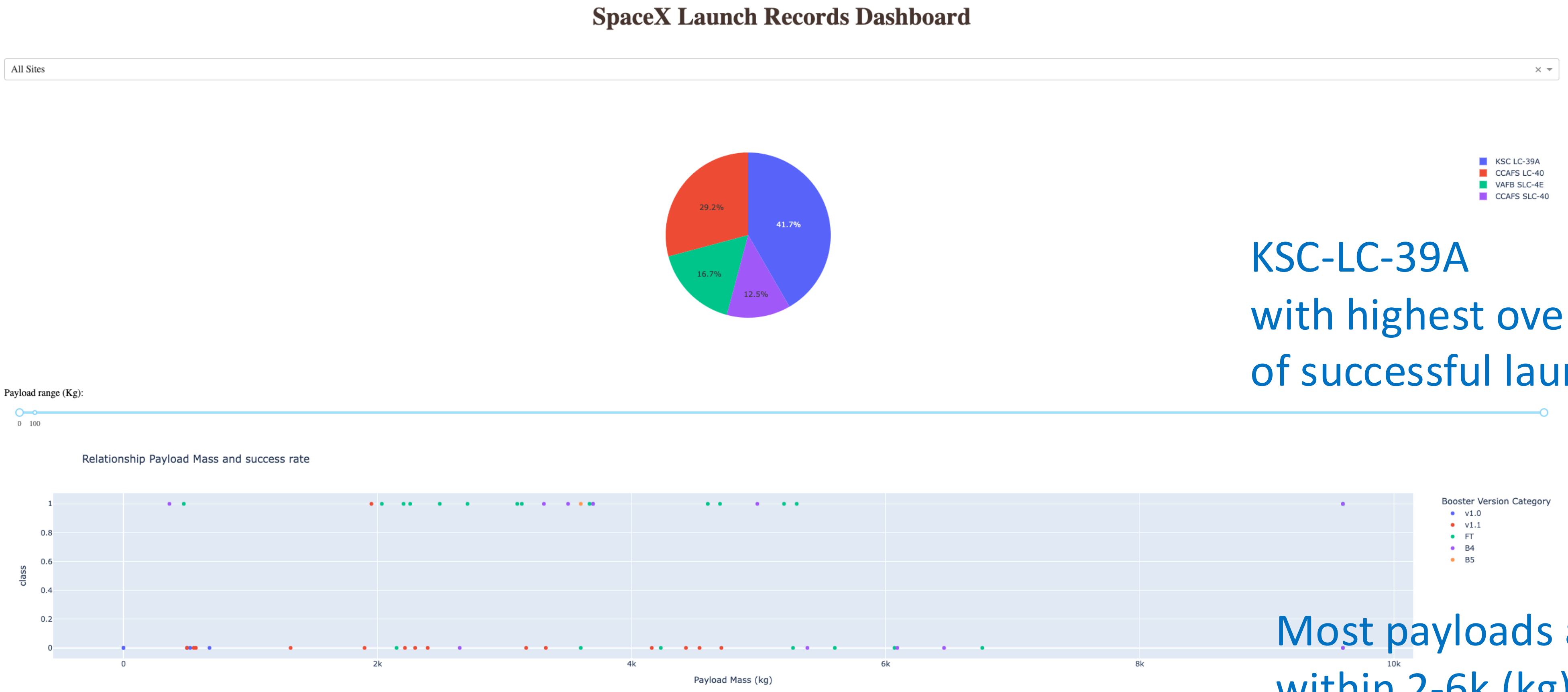
- Distance to ocean (0.9km).



IBM Developer
SKILLS NETWORK

Results: Dashboard (1/5)

Pie chart and scatter plot analysis for all launch sites.

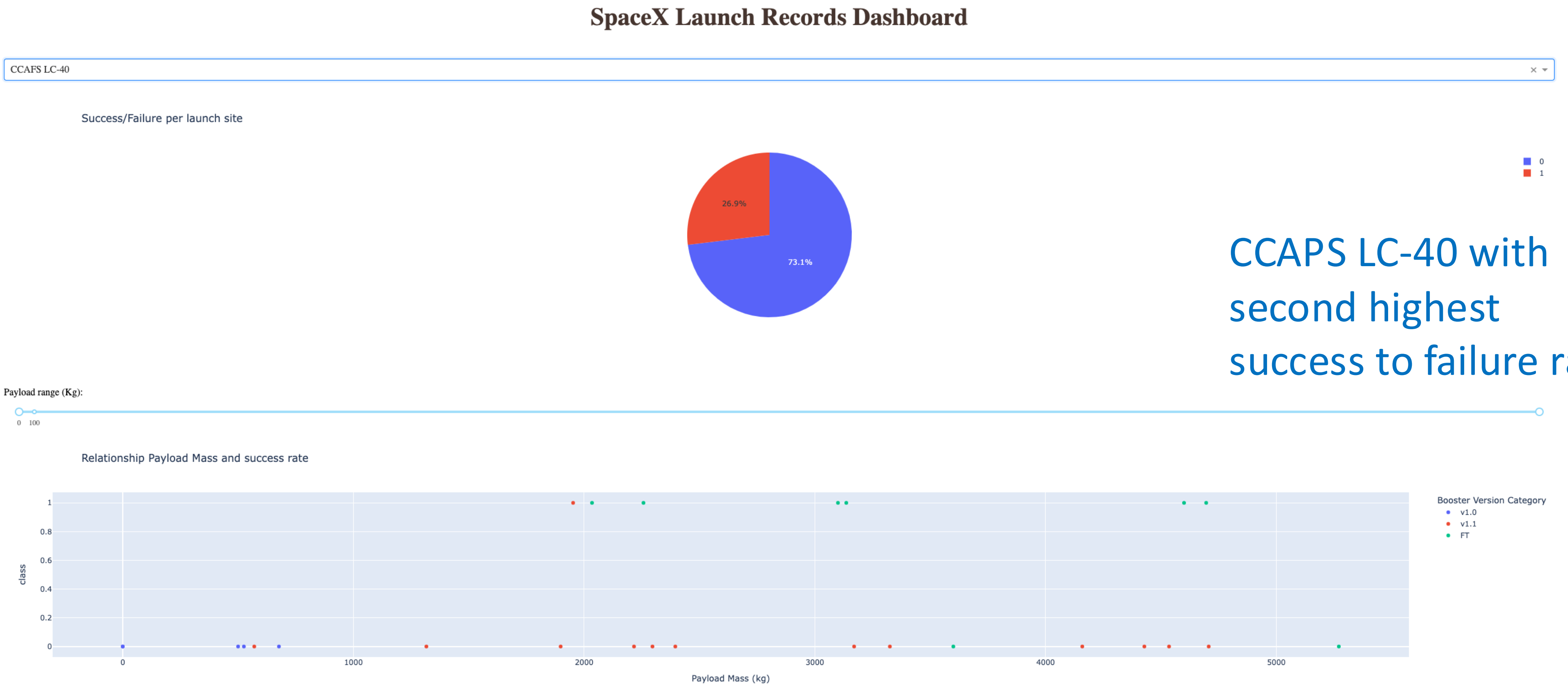


KSC-LC-39A
with highest overall share
of successful launches

Most payloads are
within 2-6k (kg)

Results: Dashboard (2/5)

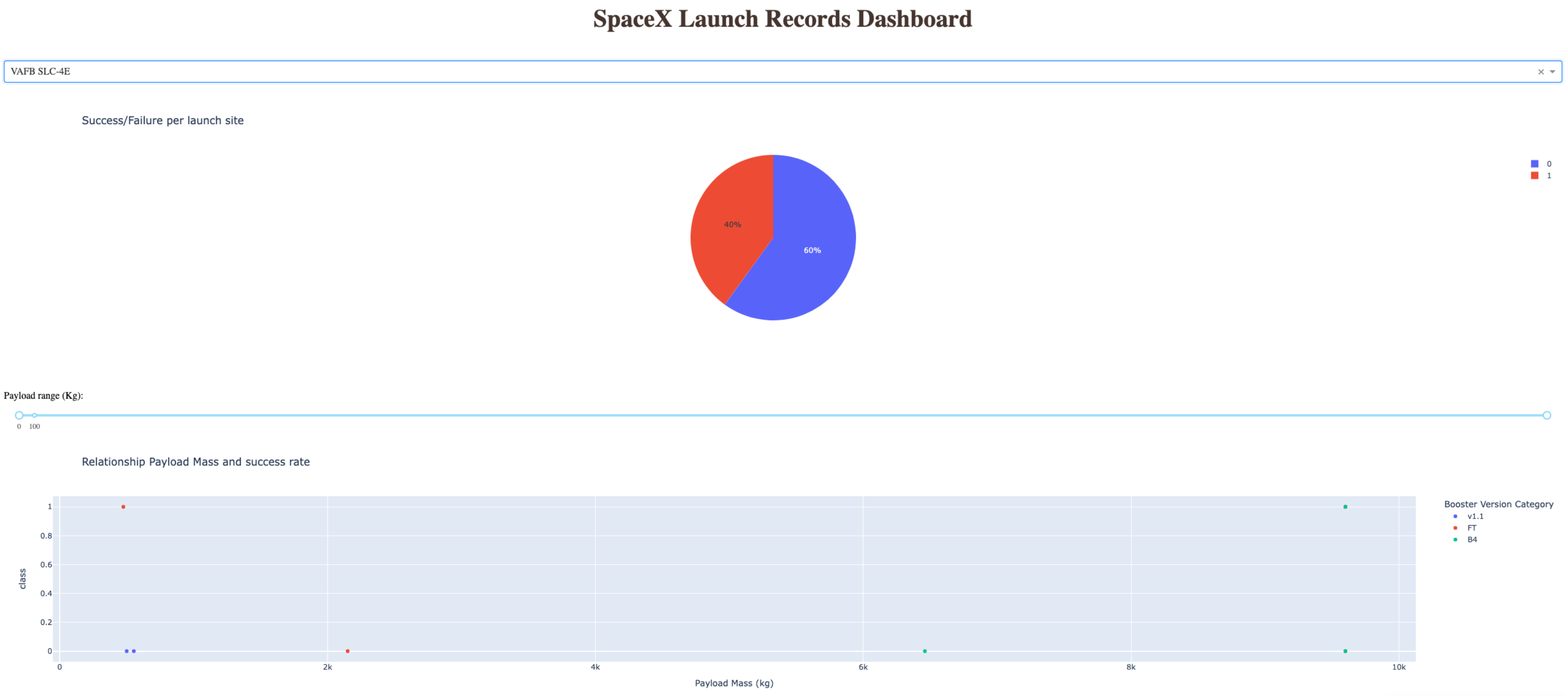
Pie chart and scatter plot analysis for CCAPS LC-40.



CCAPS LC-40 with second highest success to failure rate.

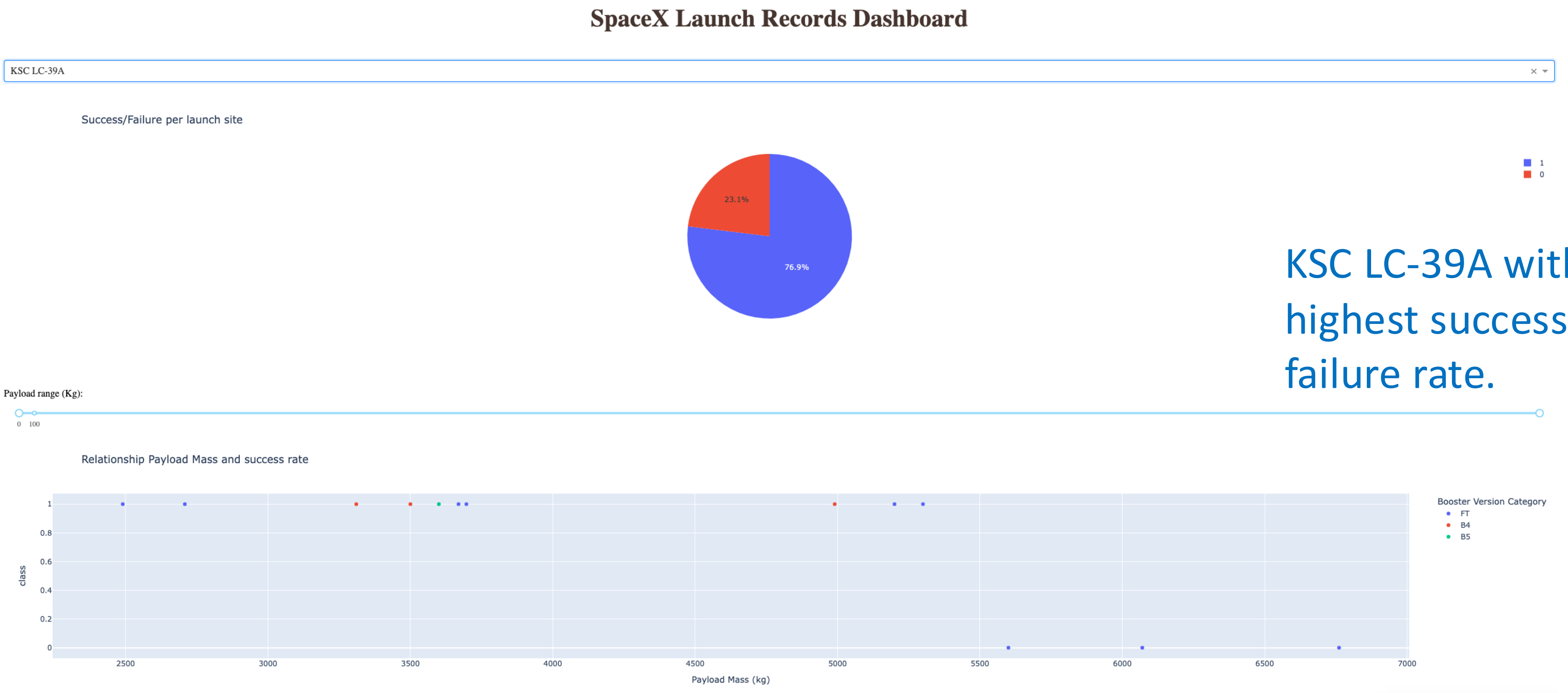
Results: Dashboard (3/5)

Pie chart and scatter plot analysis for VAFB SLLC-4E.



Results: Dashboard (4/5)

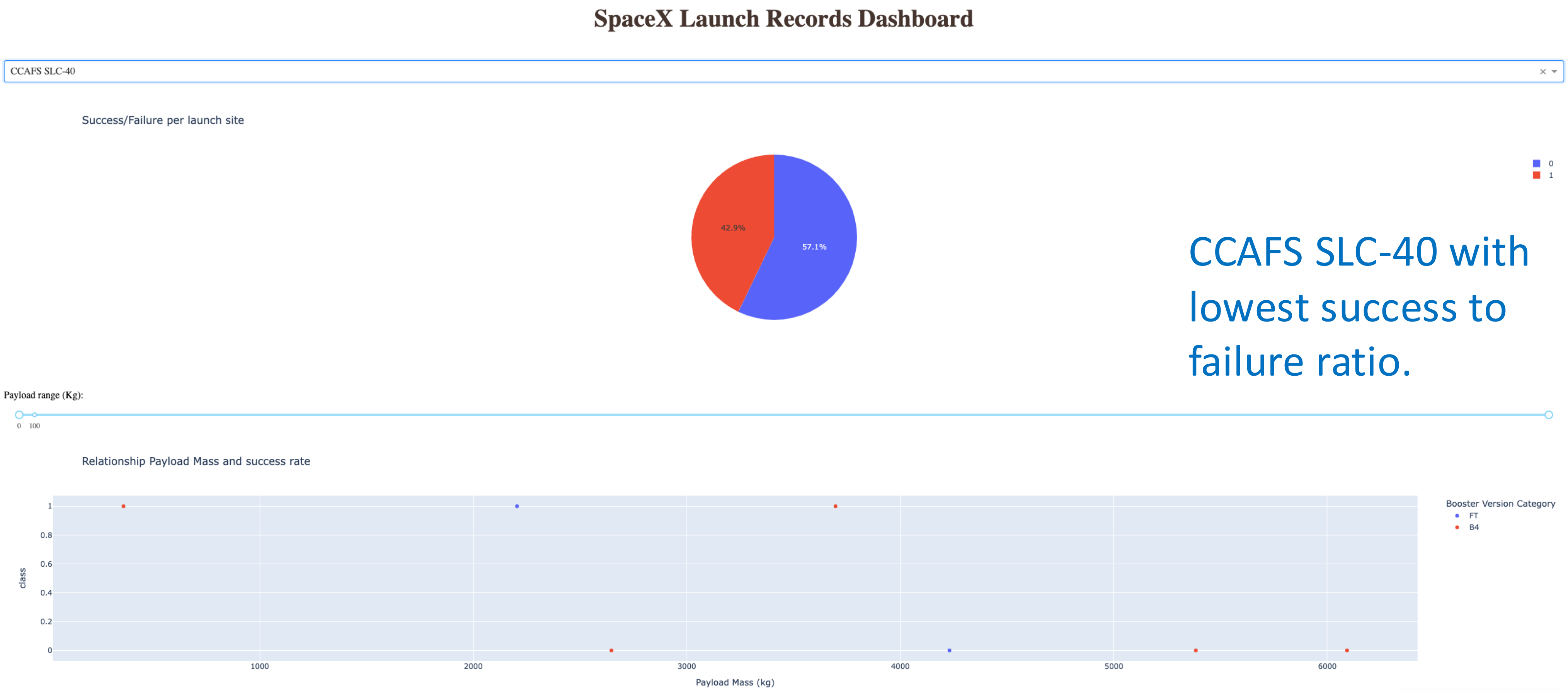
Pie chart and scatter plot analysis for KSC LC-39A.



KSC LC-39A with highest success to failure rate.

Results: Dashboard (5/5)

Pie chart and scatter plot analysis for KSC LC-39A.

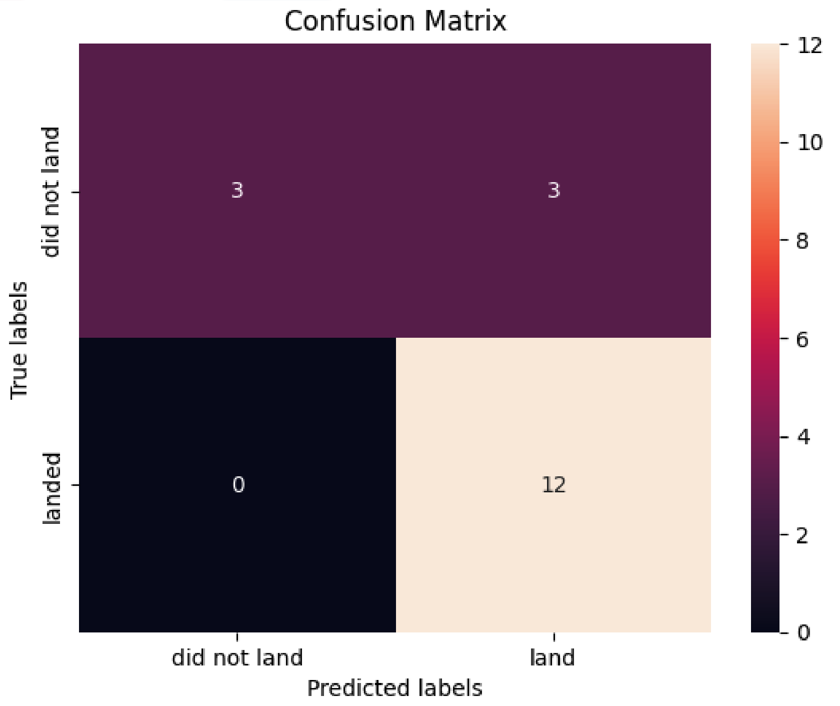


CCAFS SLC-40 with lowest success to failure ratio.

Results: Predictive Analysis

Overview of model accuracy including confusion matrix*

Model	R2 Score	Accuracy
Logistic Regression	0.83	0.846429
Supported Vector Model	0.83	0.848214
Decision Tree	**0.83**	**0.875**
KNN	0.83	0.848214



Decision tree model resulted in highest accuracy. Models are optimized with GridSearchCV for highest accuracy. Out of sample accuracy is relatively high.

*identical confusion matrix values resulted for all models.