IBM Developer
SKILLS NETWORK

Winning Space Race with Data Science

Josue Isaac Soto
Odriozola
09/06/2025

Outline

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

Executive Summary

- Given data from previous SpaceX launches and landings, an exploratory data analysis (EDA) was performed various machine learning models, such as KNN, Logistic Regression, Support Vector Machine (SVM), and Decision Trees, were also tested to create a tool that would best predict the outcome of each future launch.
- Significant characteristics such as launch location, payload mass, orbit, etc. are considered.
- Based on the above, it is found that the best supervised machine learning classification model yields a prediction of the booster recovery outcome with a accuracy of 84% for the model of booster Falcon 9.

Introduction

- SpaceX and its innovative concept of reusing the first stage in space launches represents a revolution and multi-billion-dollar savings for the aerospace industry. It opens the door to the possibility of commercial space travel and more affordable development for future space missions.
- For the purpose of identifying variables correlated with the success and/or failure of launches, understanding the context and impact of historical data from previous launches is essential to be able to try to reach the critical point. Can we predict with a high degree of accuracy whether a launch will be successful or unsuccessful?

Section 1

Methodology

Methodology

Executive Summary

- Data collection methodology:
  - Describe how data was collected
- Perform data wrangling
  - Describe how data was processed
- Perform exploratory data analysis (EDA) using visualization and SQL
- Perform interactive visual analytics using Folium and Plotly Dash
- Perform predictive analysis using classification models
  - How to build, tune, evaluate classification models

Data Collection

- The data was collected using SpaceX REST API by making a get request and then decoding the response content as JSON and then converting it into a data frame using pandas.
- Information was filtered by the Falcon 9 booster data, and drop de n/a’s values.

FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins	Reused	Legs	LandingPad	Block	Re
1	2006-03-24	Falcon 1	20.0	LEO	Kwajalein Atoll	None None	1	False	False	False	None	NaN	
2	2007-03-21	Falcon 1	NaN	LEO	Kwajalein Atoll	None None	1	False	False	False	None	NaN	
4	2008-09-28	Falcon 1	165.0	LEO	Kwajalein Atoll	None None	1	False	False	False	None	NaN	
5	2009-07-13	Falcon 1	200.0	LEO	Kwajalein Atoll	None None	1	False	False	False	None	NaN	
6	2010-06-04	Falcon 9	NaN	LEO	CCSFS SLC 40	None None	1	False	False	False	None	1.0	

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

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

Para que los resultados JSON solicitados sean más consistentes, utilizaremos el siguiente objeto de respuesta estático para este proyecto:

In [11]:
```
static_json_url='https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/datasets/API_
```

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

Deberíamos ver que la solicitud fue exitosa con el código de respuesta de estado 200

In [12]:
```
response=requests.get(static_json_url)
```

In [13]:
```
response.status_code
```

Out[13]: 200

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

Ahora decodificamos el contenido de la respuesta como un JSON usando `.json()` y lo convertimos en un marco de datos de Pandas usando `.json_normalize()`

In [14]:
```
# Utilice el método json_normalize para convertir el resultado json en un marco de datos
data_json = response.json()
data_json
data = pd.json_normalize(data_json)
```

Data Collection – SpaceX API

- The data was collected using SpaceX REST API by making a get request and then decoding the response content as JSON and then converting it into a data frame using pandas.
- GitHub URL of the completed web scraping notebook, https://github.com/JosueOdriozola/Data-Sciene/blob/70710dc2908c445c9dd0769903ee90c16386e99e/jupyter-labs-spacex-data-collection-api.ipynb as an external reference and peer-review purpose

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

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

Para que los resultados JSON solicitados sean más consistentes, utilizaremos el siguiente objeto de respuesta estático para este proyecto:

In [11]:
```
static_json_url='https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/datasets/API_
```

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

Deberíamos ver que la solicitud fue exitosa con el código de respuesta de estado 200

In [12]:
```
response=requests.get(static_json_url)
```

In [13]:
```
response.status_code
```

Out[13]: 200

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

Ahora decodificamos el contenido de la respuesta como un JSON usando `.json()` y lo convertimos en un marco de datos de Pandas usando `.json_normalize()`

In [14]:
```
# Utilice el método json_normalize para convertir el resultado json en un marco de datos
data_json = response.json()
data_json
data = pd.json_normalize(data_json)
```

Task 2: Filter the dataframe to only include `Falcon 9` launches

Finally we will remove the Falcon 1 launches keeping only the Falcon 9 launches. Filter the data dataframe using the `BoosterVersion` column to only keep the Falcon 9 launches. Save the filtered data to a new dataframe called `data_falcon9`.

In [36]:
```
eliminar =  data.loc[data['BoosterVersion'] == "Falcon 1"].index
data_falcon9 = data.drop(eliminar, axis=0)
data_falcon9.describe()
data.dtypes
```

Data Collection - Scraping

- By scraping the web, we obtained historical information on releases from Wikipedia, and then used this information to create a data frame of the relevant information.

- GitHub URL of the completed web scraping notebook, https://github.com/JosueOdriozola/Data-Sciene/blob/70710dc2908c445c9dd0769903ee90c16386e99e/jupyter-labs-webscraping.ipynb as an external reference and peer-review purpose

```
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

            # Append the flight_number into launch_dict with key `Flight No.`
            launch_dict['Flight No.'].append(flight_number)
            #print(flight_number)
            datatimelist=date_time(row[0])

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

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

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

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

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

            # Payload Mass
            # Append the payload_mass into launch_dict with key `Payload mass`
            payload_mass = get_mass(row[4])
            launch_dict['Payload mass'].append(payload_mass)
            #print(payload)
```

```
            # Payload Mass
            # Append the payload_mass into launch_dict with key `Payload mass`
            payload_mass = get_mass(row[4])
            launch_dict['Payload mass'].append(payload_mass)
            #print(payload)

            # Orbit
            # Append the orbit into launch_dict with key `Orbit`
            orbit = row[5].a.string
            launch_dict['Orbit'].append(orbit)

            #print(orbit)

            # Customer
            # Append the customer into launch_dict with key `Customer`

            #customer = row[6].a
            #if not customer:
            #    customer = 'Nan'
            #else:
            customer = row[6].text.strip()
            launch_dict['Customer'].append(customer)
            #print(customer)

            # Launch outcome
            # TODO: Append the launch_outcome into launch_dict with key `Launch outcome`
            launch_outcome = list(row[7].strings)[0]
            launch_dict['Launch outcome'].append(launch_outcome)

            #print(launch_outcome)

            # Booster landing
            # TODO: Append the launch_outcome into launch_dict with key `Booster landing`
            booster_landing = landing_status(row[8])
            launch_dict['Booster landing'].append(booster_landing)

            #print(booster_landing)
```

After you have fill in the parsed launch record values into `launch_dict`, you can create a dataframe from it.

```
df=pd.DataFrame(launch_dict)
df
```

	Flight No.	Launch site	Payload	Payload mass	Orbit	Customer	Launch outcome	Version Booster	Booster landing	Date	Time
0	1	CCAFS	Dragon Spacecraft Qualification Unit	0	LEO	SpaceX	Success\n	F9 v1.07B0003.18	Failure	4 June 2010	18:45
1	2	CCAFS	Dragon	0	LEO	NASA (COTS)\nNRO	Success	F9 v1.07B0004.18	Failure	8 December 2010	15:43
2	3	CCAFS	Dragon	525 kg	LEO	NASA (COTS)	Success	F9 v1.07B0005.18	No attempt\n	22 May 2012	07:44
3	4	CCAFS	SpaceX CRS-1	4,700 kg	LEO	NASA (CRS)	Success\n	F9 v1.07B0006.18	No attempt	8 October 2012	00:35
4	5	CCAFS	SpaceX CRS-2	4,877 kg	LEO	NASA (CRS)	Success\n	F9 v1.07B0007.18	No attempt\n	1 March 2013	15:10
...	...	...	...	...	...	...	...	...	...	...	...
116	117	CCSFS	Starlink	15,600 kg	LEO	SpaceX	Success\n	F9 B5B1051.10657	Success	9 May 2021	06:42
117	118	KSC	Starlink	~14,000 kg	LEO	SpaceX Capella Space and Tyvak	Success\n	F9 B5B1058.8660	Success	15 May 2021	22:56
118	119	CCSFS	Starlink	15,600 kg	LEO	SpaceX	Success\n	F9 B5B1063.2665	Success	26 May 2021	18:59
119	120	KSC	SpaceX CRS-22	3,328 kg	LEO	NASA (CRS)	Success\n	F9 B5B1067.1668	Success	3 June 2021	17:29

Data Wrangling

- For this part, we understand the context and diversity of the data. Whether it's the different launch sites, orbits, and the different possible outcomes of each launch.
- GitHub URL of the completed web scraping notebook, https://github.com/JosueOdriozola/Data-Sciene/blob/70710dc2908c445c9dd0769903ee90c16386e99e/labs-jupyter-spacex-Data%20wrangling.

TASK 1: Calculate the number of launches on each site

The data contains several Space X launch facilities: Cape Canaveral Space Launch Complex 40 **VAFB SLC 4E** , Vandenberg Air Force Base Space Launch Complex 4E **(SLC-4E)**, Kennedy Space Center Launch Complex 39A **KSC LC 39A** .The location of each Launch Is placed in the column `LaunchSite`

Next, let's see the number of launches for each site.

Use the method `value_counts()` on the column `LaunchSite` to determine the number of launches on each site:

```
# Apply value_counts() on column LaunchSite
df["LaunchSite"].value_counts()
```

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

Each launch aims to an dedicated orbit, and here are some common orbit types:

Cada lanzamiento apunta a una órbita dedicada, y estos son algunos tipos de órbitas comunes:

TASK 2: Calculate the number and occurrence of each orbit

Use the method `.value_counts()` to determine the number and occurrence of each orbit in the column `Orbit`

Utilice el método `.value_counts()` para determinar el número y la ocurrencia de cada órbita en la columna `Orbit`

```
# Apply value_counts on Orbit column
df["Orbit"].value_counts()
```

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

TASK 3: Calculate the number and occurence of mission outcome of the orbits

Use the method `.value_counts()` on the column `Outcome` to determine the number of `landing_outcomes` .Then assign it to a variable landing_outcomes.

Utilice el método `.value_counts()` en la columna `Outcome` para determinar la cantidad de `landing_outcomes` . Luego, asígnelo a una variable landing_outcomes.

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

```
Outcome
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: count, dtype: int64
```

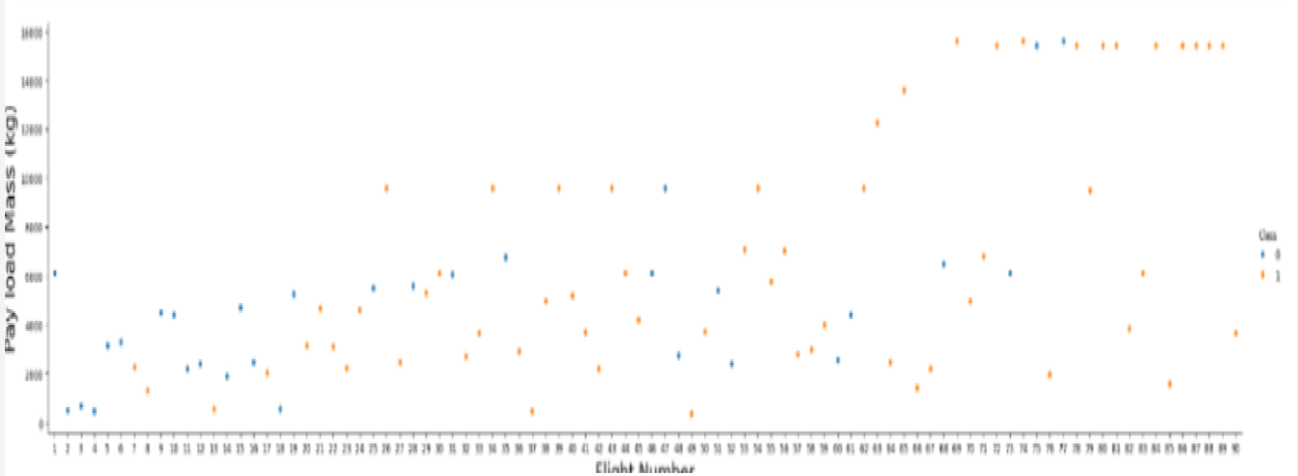
```
for i,outcome in enumerate(landing_outcomes.keys()):
    print(i,outcome)
df
```

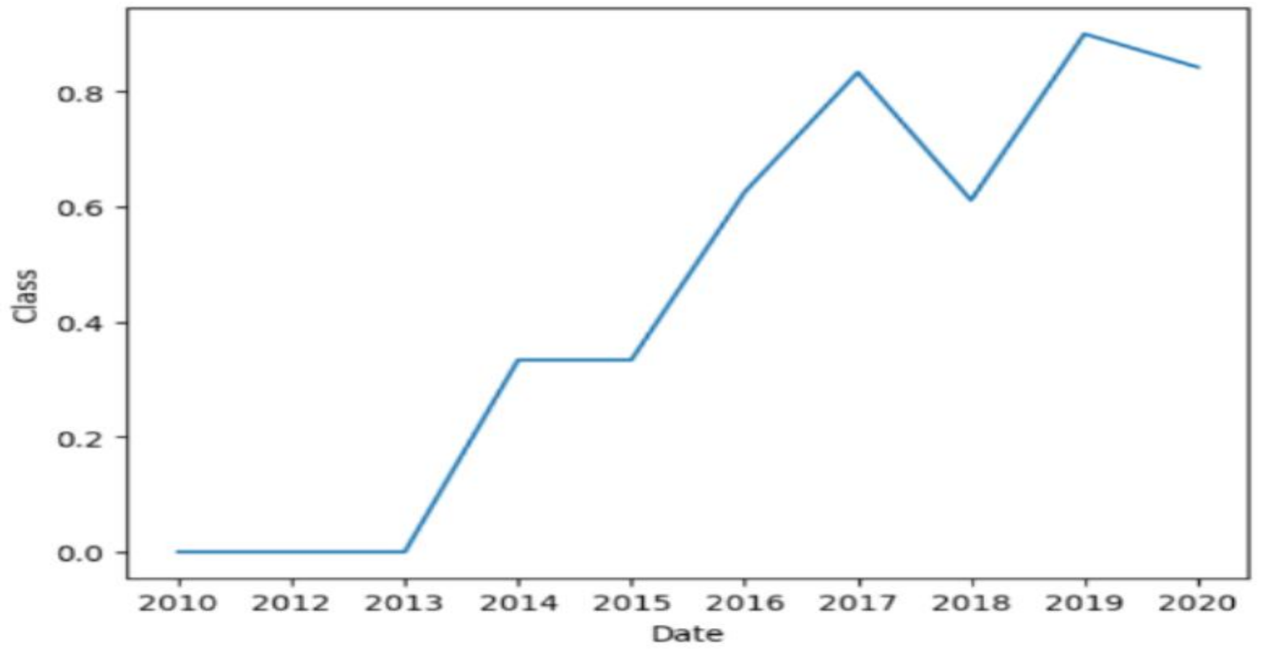
```
0 True ASDS
1 None None
2 True RTLS
3 False ASDS
4 True Ocean
5 False Ocean
6 None ASDS
7 False RTLS
```

	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins	Reused	Legs	La
0	1	2010-06-04	Falcon 9	6104.959412	LEO	CCAFS SLC 40	None None	1	False	False	False	
1	2	2012-05-22	Falcon 9	525.000000	LEO	CCAFS SLC 40	None None	1	False	False	False	
2	3	2013-03-01	Falcon 9	677.000000	ISS	CCAFS SLC 40	None None	1	False	False	False	
3	4	2013-09-29	Falcon 9	500.000000	PO	VAFB SLC 4E	False Ocean	1	False	False	False	
4	5	2013-12-03	Falcon 9	3170.000000	GTO	CCAFS SLC 40	None None	1	False	False	False	
...	...	...	...	...	...	...	...	...	...	...	...	
85	86	2020-09-03	Falcon 9	15400.000000	VLEO	KSC LC 39A	True ASDS	2	True	True	True	5e9e3032383ecb6b
86	87	2020-10-06	Falcon 9	15400.000000	VLEO	KSC LC 39A	True ASDS	3	True	True	True	5e9e3032383ecb6b
87	88	2020-10-18	Falcon 9	15400.000000	VLEO	KSC LC 39A	True ASDS	6	True	True	True	5e9e3032383ecb6b
88	89	2020-10-24	Falcon 9	15400.000000	VLEO	CCAFS SLC 40	True ASDS	3	True	True	True	5e9e3033383ecbb
89	90	2020-11-05	Falcon 9	3681.000000	MEO	CCAFS SLC 40	True ASDS	1	True	False	True	5e9e3032383ecb6b

EDA with Data Visualization

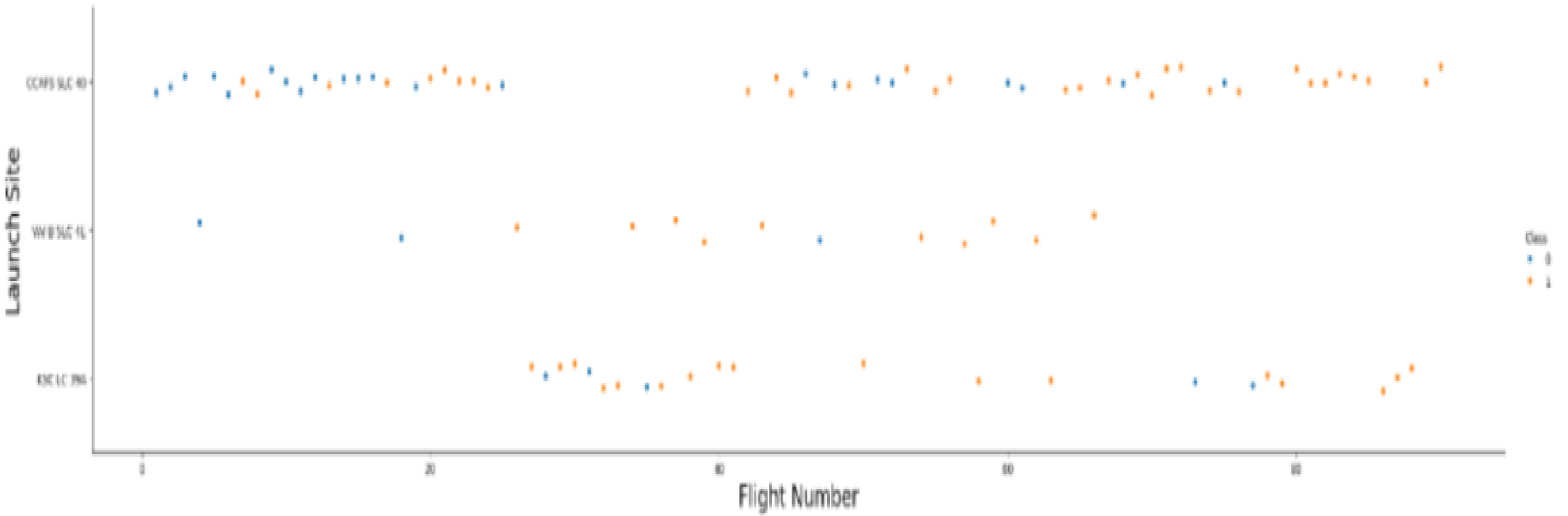
- To perform an exploratory data analysis, we will begin by reviewing some relationships between different pairs of variables and their outcome. For which we will use scatter plots.
- Flight number / Payload: As the flight number increases, the first stage is more likely to land successfully. Payload mass also appears to be a factor; even with heavier payloads, the first stage usually returns successfully.
- We reinforce this hypothesis with the annual success ratio and we observe that as the years go by, this ratio increases significantly.

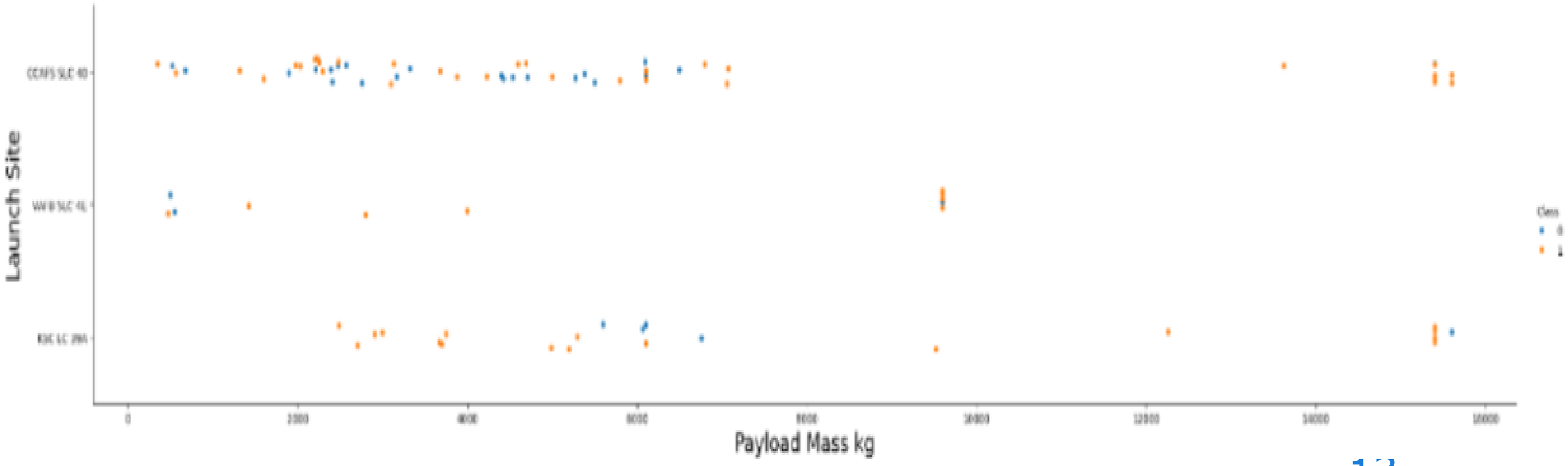




EDA with Data Visualization

- Likewise, through dispersion visualizations we observe that there is a preference for 2 of the 3 different launch points, ceasing to use said launch point from flight number 66 onwards.
- The reason for the disuse of that launch point makes sense when reviewing the payload mass and its result at different launch points.
- GitHub URL of completed EDA with data visualization notebook, https://github.com/JosueOdriozola/Data-Sciene/blob/70710dc2908c445c9dd0769903ee90c16386e99e/pandas-data-EDA.python.ipynb





EDA with SQL

- **Display the names of the unique launch sites in the space mission**
- Display average payload mass carried by booster version F9 v1.1
- **Display 5 records where launch sites begin with the string 'CCA'**
- **Display the total payload mass carried by boosters launched by NASA (CRS)**
- **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 all the booster_versions that have carried the maximum payload mass. Use a subquery.

GitHub URL of complete EDA with SQL notebook,
https://github.com/JosueOdriozola/Data-Sciene/blob/70710dc2908c445c9dd0769903ee90c16386e99e/jupyter-labs-eda-sql-coursera_sqllite.ipynb

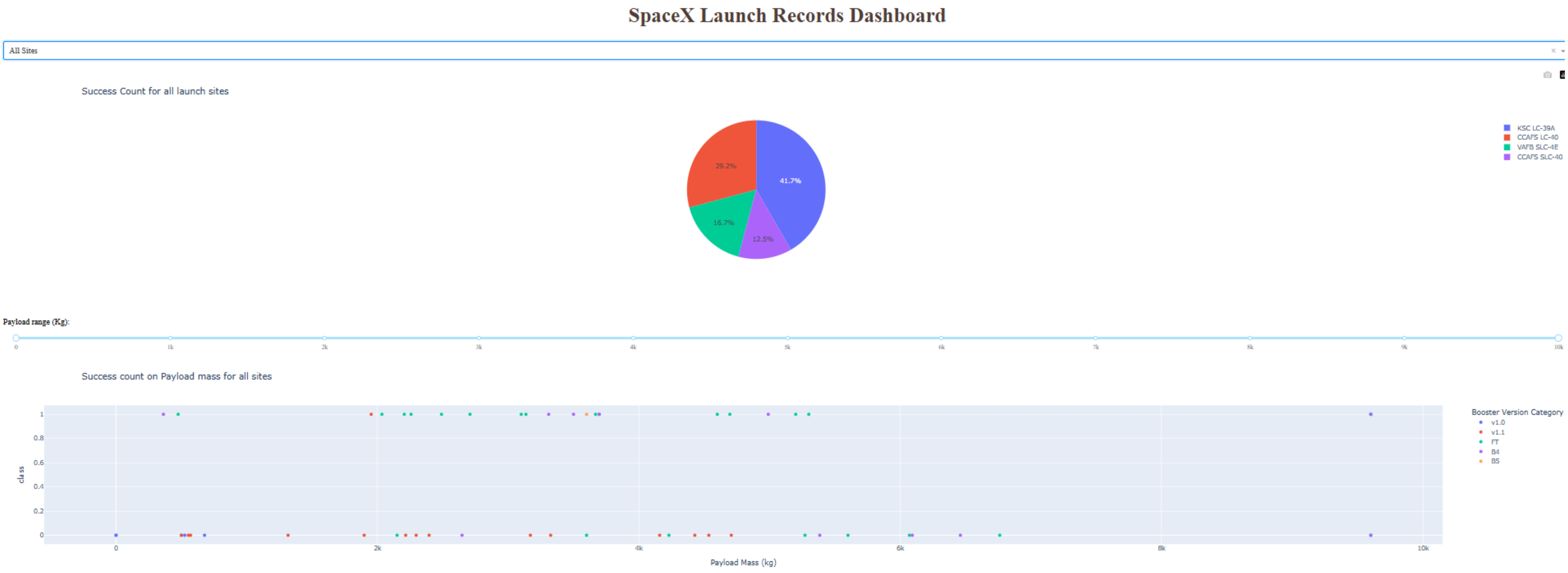
Build an Interactive Map with Folium

- A map have been created with the folium library to marked all the lauch sites, and created map objects such markers, circles, lines to mark the successs (1) or failure (0) of launches for each launch site.
- GitHub URL of complete interactive map with Folium map https://github.com/JosueOdriozola/Data-Sciene/blob/70710dc2908c445c9dd0769903ee90c16386e99e/lab_jupyter_launch_site_location-mapas.ipynb

Build a Dashboard with Plotly Dash

- Within the dashboard, a drop-down menu is implemented in which the specific launch site can be selected, as well as a pie chart where the count of launches made by site can be observed. A scatter plot was also inserted where the success or failure of each launch can be observed by site and by booster version.
- Add the GitHub URL of complete Plotly Dash lab, https://github.com/JosueOdriozola/Data-Sciene/blob/70710dc2908c445c9dd0769903ee90c16386e99e/spacex-dash-app.py

Build a Dashboard with Plotly Dash



Predictive Analysis (Classification)

- After loading the data into a data frame, we identify the target variable "Class“ (Y), which indicates whether a launch was successful or not.
- On the other hand, the remaining features (X) are normalized to then be able to perform the segmentation and training of the models later with the GridSearchCV function.

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

```
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)
```

TASK 2

Standardize the data in `X` then reassign it to the variable `X` using the transform provided below.

Estandarice los datos en `X` y luego reasignelos a la variable `X` usando la transformación proporcionada a continuación.

```
# students get this: definimos y asignamos a la variable "transform" el metodo de estandarización
transform = preprocessing.StandardScaler()
X=transform.fit_transform(X)
X
```

Predictive Analysis (Classification)

- We create a logistic regression object and then create a GridSearchCV object logreg_cv with cv = 10. Tune the object to find the best parameters from the parameters dictionary.
- We output the GridSearchCV object for logistic regression. Display the best parameters using the data attribute best_params_ and the accuracy on the validation data using the data attribute best_score_.
- Then, calculate the accuracy using the test data and generate the confusion matrix. We'll repeat this process for each machine learning method.
- GitHub URL of complete predictive analysis https://github.com/JosueOdriozola/Data-Sciene/blob/d0a9cb6ae1873ffae9844ec70c72d481b9880d2b/SpaceX_Machine%20Learning%20Prediction_Part_5.ipynb

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

we can see we only have 18 test samples.

Podemos ver que solo tenemos 18 muestras de prueba.

```
Y_test.shape

print ('Train set:')
print('X-train= ', X_train.shape,'Y-train= ',Y_train.shape)
print ('Test set:')
print('X-test= ', X_test.shape,  'Y-test= ',Y_test.shape)
```

```
rain set:
-train=  (72, 83) Y-train=  (72,)
est set:
-test=  (18, 83) Y-test=  (18,)
```

TASK 4 / Tarea 4

Create a logistic regression object then create a GridSearchCV object `logreg_cv` with cv = 10. Fit the object to find the best parameters from the dictionary `parameters`.

Cree un objeto de regresión logística y luego cree un objeto GridSearchCV `logreg_cv` con cv = 10. Ajuste el objeto para encontrar los mejores parámetros del diccionario `parameters`.

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

```
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)
```

```
print("Hiperparametros ajustados :(mejores parametros) ",logreg_cv.best_params_)
print("exactitud :",logreg_cv.best_score_)
```

```
Hiperparametros ajustados :(mejores parametros)  {'C': 0.01, 'penalty': 'l2', 'solver': 'lbfgs'}
exactitud : 0.8464285714285713
```

TASK 5

Calculate the accuracy on the test data using the method `score` :

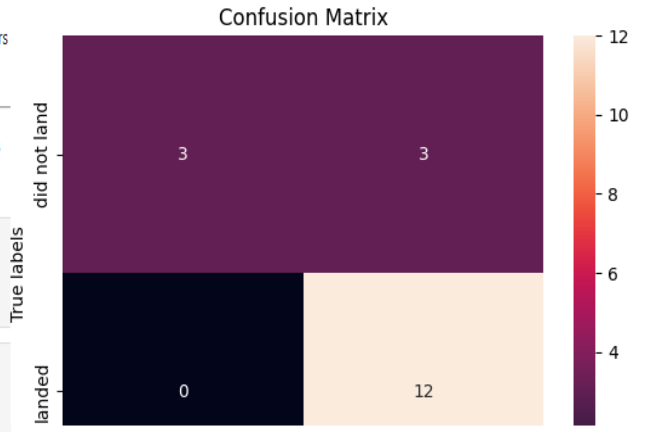
Calcule la precisión de los datos de prueba utilizando el método `score` :

```
print("Prueba de exactitud de datos de Regresión Logistica :",logreg_cv.score(X_test, Y_test))
```

```
Prueba de exactitud de datos de Regresión Logistica : 0.8333333333333334
```

Lets look at the confusion matrix:

```
yhat=logreg_cv.predict(X_test)
plot_confusion_matrix(Y_test,yhat)
```



Results

- Exploratory data analysis results:

There is a relationship between the flight number and the mass load, since the last launches have been for the maximum possible load and with a fairly high success rate.

- Interactive analytics demo in screenshots
- **Predictive analysis results:**
- After comparing the precision values in each of the models, it is obtained that the best model applied to determine if a launch will be successful can be Logistic regression, SVM or KNN with 83.3% of accuracy

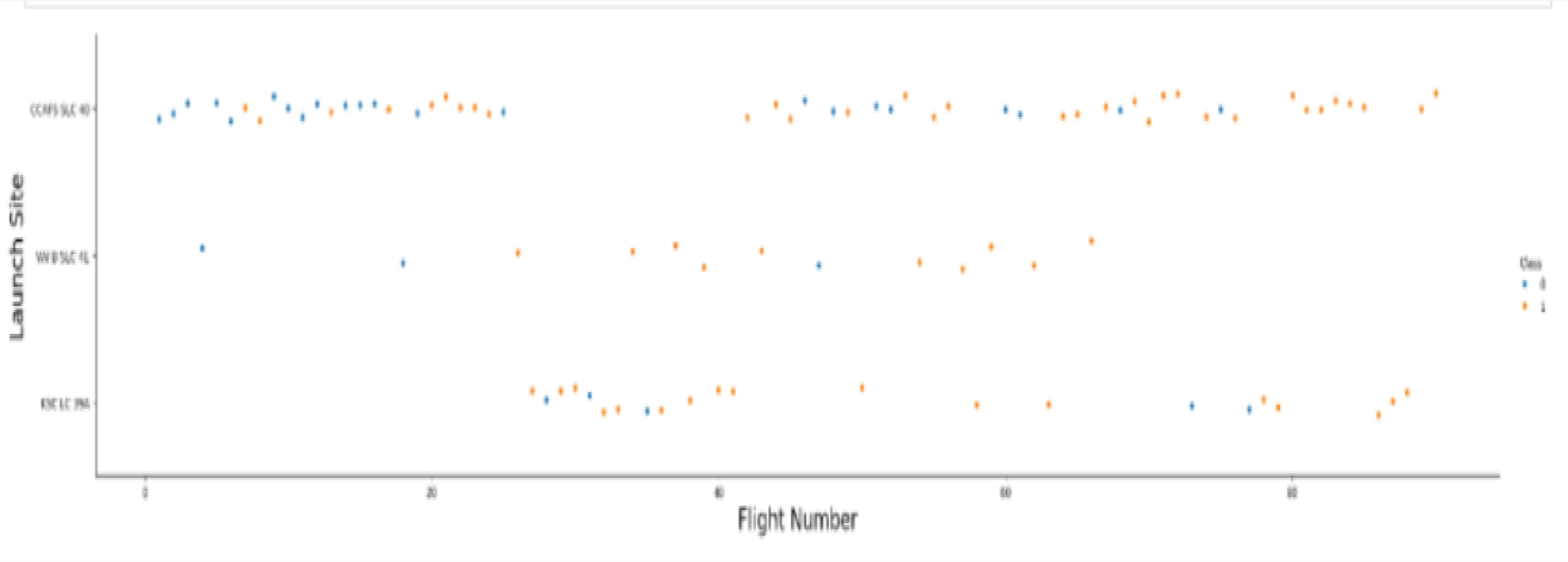
Method	Test Data Accuracy
Logistic_Reg	0.833333
SVM	0.833333
Decision Tree	0.722222
KNN	0.833333

Section 2

Insights drawn from EDA

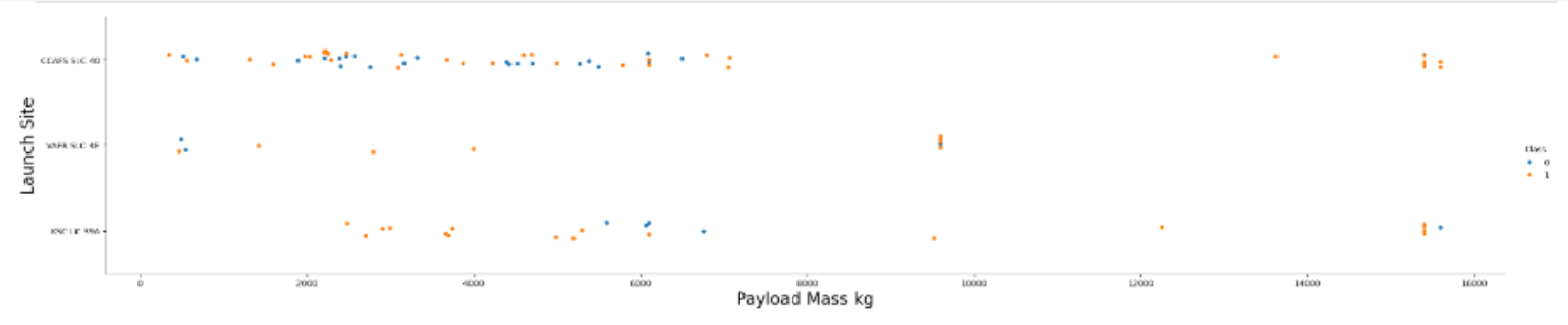
Flight Number vs. Launch Site

We observe that there is a preference for 2 of the 3 different launch points, ceasing to use said launch point from flight number 66 onwards. Even though the last releases were successful, this may be due to another feature that made such a change necessary.



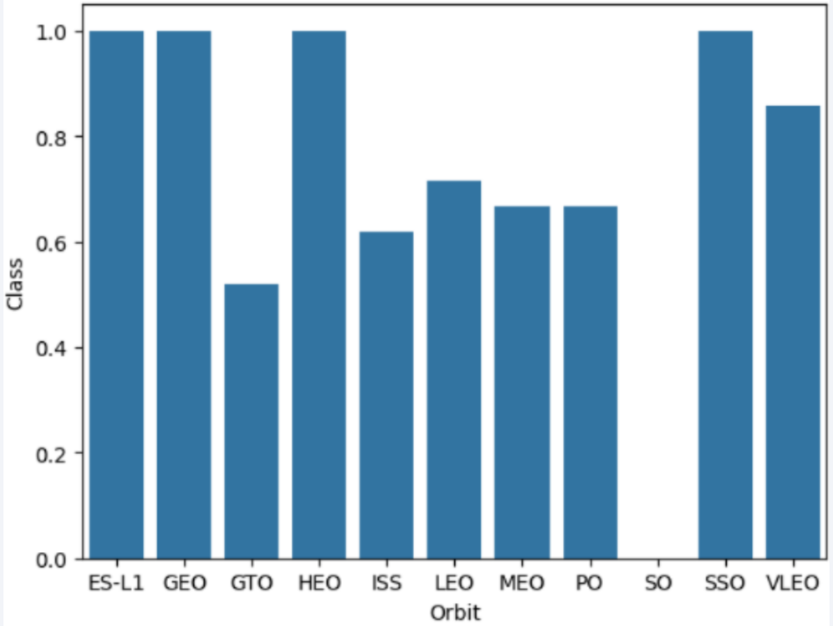
Payload vs. Launch Site

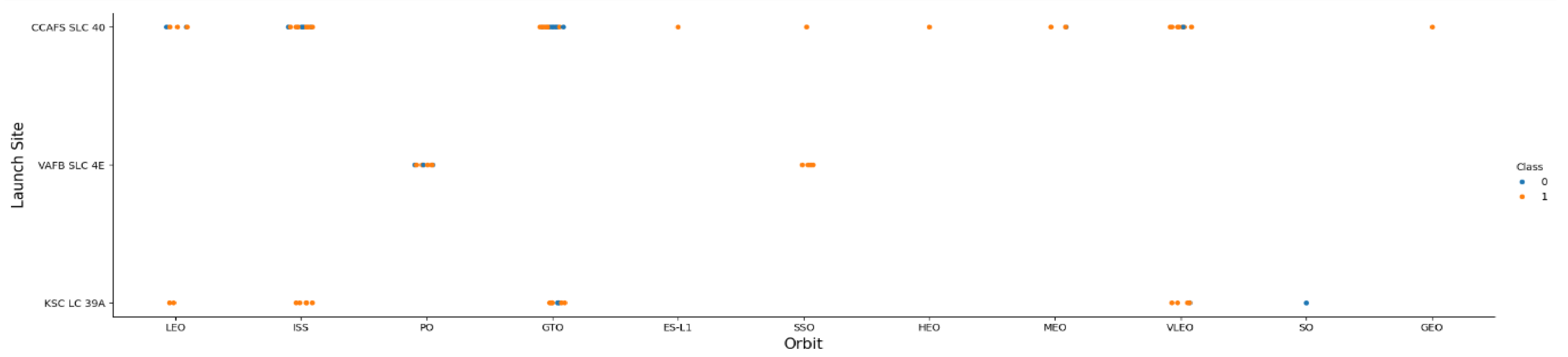
- It is observed that, for payloads greater than 10,000 kg, only two of the launch sites are used. Furthermore, in these cases, the success rate is high compared to average.
- In this case, it means that carrying more mass in fewer trips is not a significant problem.



Success Rate vs. Orbit Type

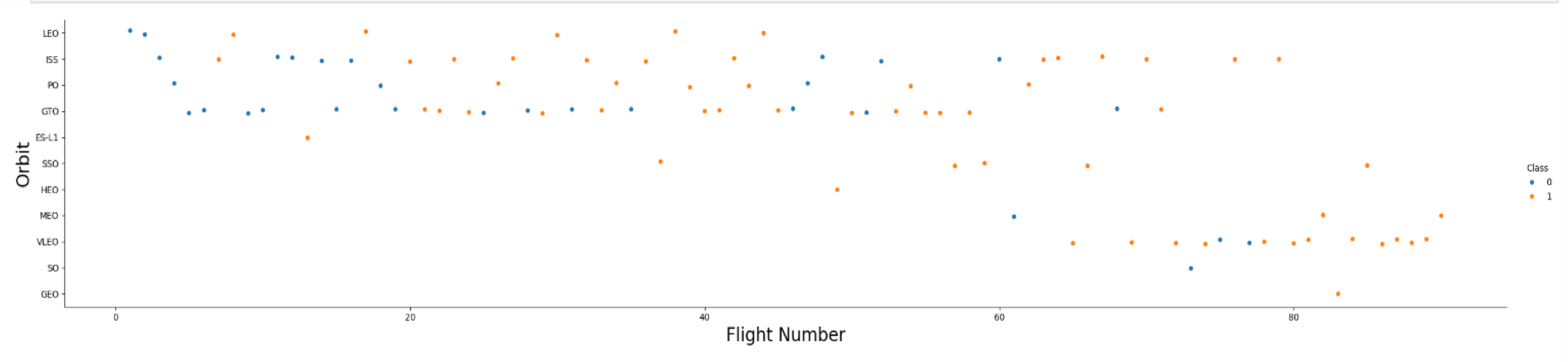
- It is identified that for the ESL-1, GEO, HEO and SSO orbits the success rate is 100%. However, this is a consequence of the fact that only 1 to 5 launches have been carried out.
- The VLEO orbit have a high success rate and it’s one of the most used orbits.





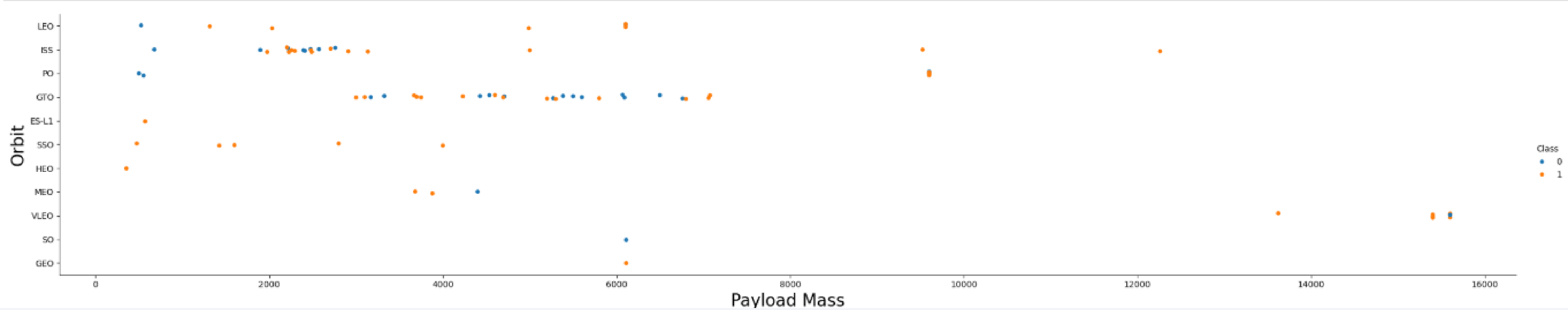
Flight Number vs. Orbit Type

- Complementing the previous section, it is observed that the VLEO orbit is also the most used in launches from the sixty-fifth onwards.
- It has an success rate of 85.7%
- After launch 60 LEO and VLEO are the most used orbits, leaving the remaining ones practically in disuse.



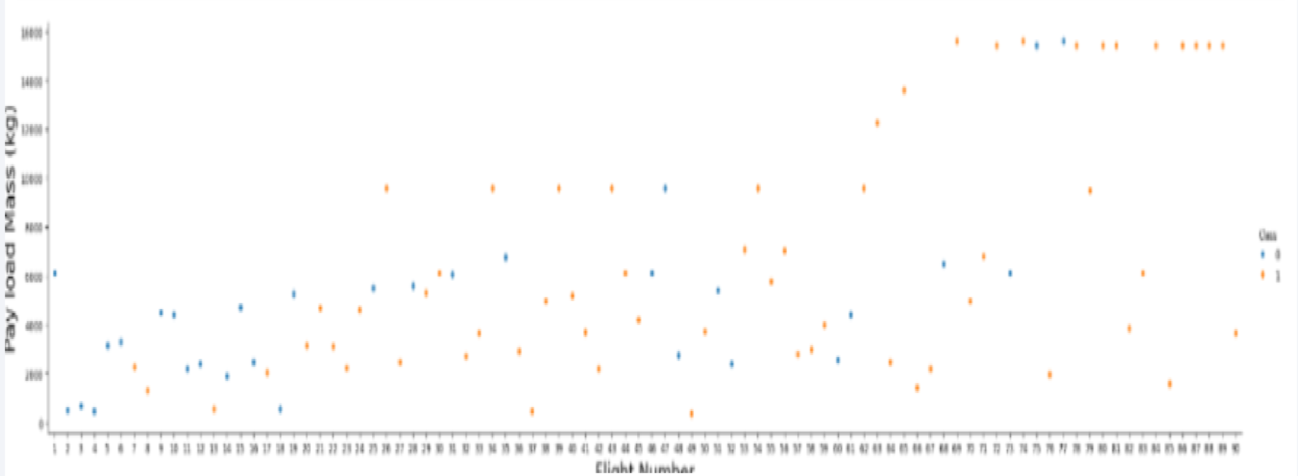
Payload vs. Orbit Type

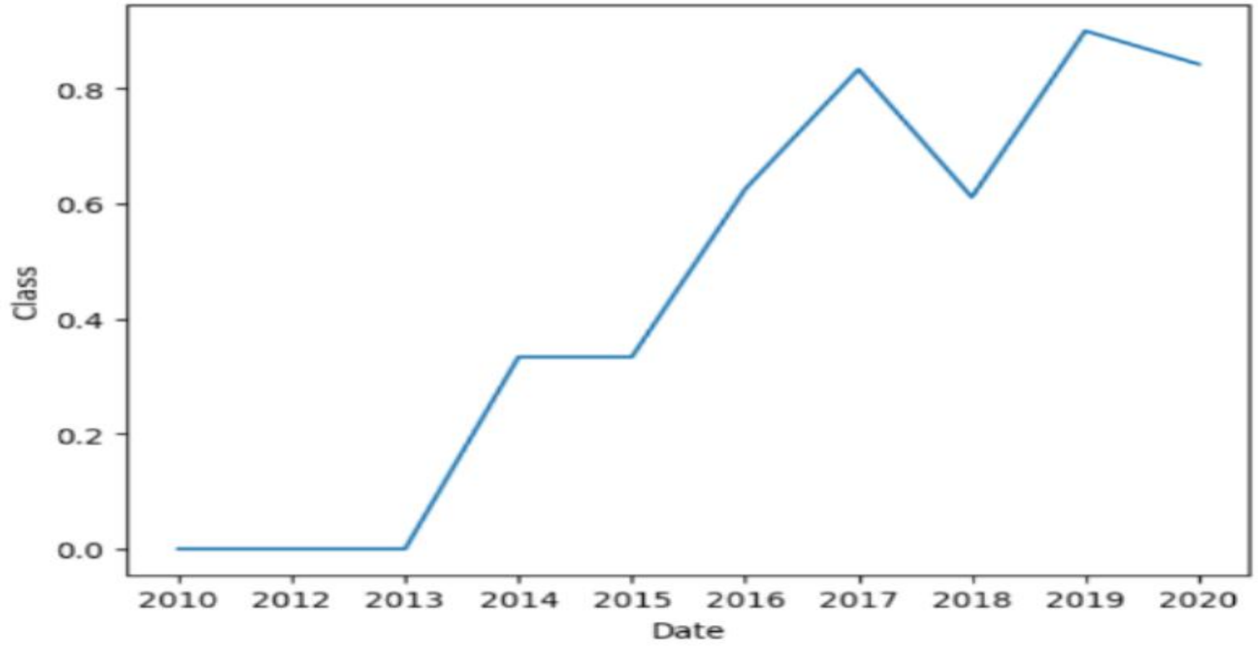
- For Payloads mass greater than 12,000 kg, the VLEO orbit is used exclusively.
- And in the range of 8,000 to 12,000 only ISS and PO orbits are used



Launch Success Yearly Trend

- As time goes by, it seems that SpaceX has measured both its successes and its failures well, since the fact that the more recent the launches are, the higher their success rate may indicate constant improvement.





Although there are 4 different launch sites, there are 2 that are extremely close to each other.

All Launch Site Names

- Although there are 4 different launch sites, there are 2 that are extremely close to each other.

In [14]:

```
%sql select "Launch_Site" from SPACEXTABLE group by "Launch_Site"
```

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

Out[14]:

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

Launch Site Names Begin with 'CCA'

- We get the first 5 records whose launch site starts with "CCA"

```
%sql select * from SPACEXTABLE where "Launch_Site" LIKE"CCA%" limit 5;
```

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

- For the query, a column is selected and defined that will be the sum of the total mass load, it is indicated from which table said query will be obtained and finally the filters are defined to take into account the resulting sum.
- The result of the total Payload Mass in Kg launched by NASA is **111,268 Kg.**

```
%sql select sum("PAYLOAD_MASS__KG_") as "Total_Payload" from SPACEXTABLE where "Payload" like "%CRS%";
```

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

Total_Payload
111268

Average Payload Mass by F9 v1.1

- This is the average of each lauch by the booster version F9 v1.1

```
%sql select AVG("PAYLOAD_MASS__KG_") as "Average_Payload" from SPACEXTABLE WHERE "Booster_Version" like "F9 v1.1%";
```

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

Average_Payload
2534.6666666666665

First Successful Ground Landing Date

- The first succesful landing out come in ground pad was achieved in 22/12/2015.

```
%sql select min("Date"),"Landing_Outcome" from SPACEXTABLE where Landing_Outcome = "Success (ground pad)";
```

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

min("Date")	Landing_Outcome
2015-12-22	Success (ground pad)

Successful Drone Ship Landing with Payload between 4000 and 6000

- This is a list of the names of the booster versions which landed successfully and their payload mass was in the range of 4000 to 6000 kg.

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

```
%%sql
select "Booster_Version","PAYLOAD_MASS__KG_","Landing_Outcome" from SPACEXTABLE
where Landing_Outcome = 'Success (drone ship)'  and PAYLOAD_MASS__KG_ between 4000 and 6000 group by "Booster_Version";
```

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

Booster_Version	PAYLOAD_MASS__KG_	Landing_Outcome
F9 FT B1021.2	5300	Success (drone ship)
F9 FT B1031.2	5200	Success (drone ship)
F9 FT B1022	4696	Success (drone ship)
F9 FT B1026	4600	Success (drone ship)

Total Number of Successful and Failure Mission Outcomes

- The total number of successful and failed launches is presented.
- Successful = 61
- Failed = 10

```
%%sql
select "Landing_Outcome", Count("Landing_Outcome") as "Count" from SPACEXTABLE
    where "Landing_Outcome" like "Success%" or "Landing_Outcome" like "Failure%" group by "Landing_Outcome";
```

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

Landing_Outcome	Count
Failure	3
Failure (drone ship)	5
Failure (parachute)	2
Success	38
Success (drone ship)	14
Success (ground pad)	9

Boosters Carried Maximum Payload

- These booster versions have had the maximum payload mass of all launches

```
%%sql
select DISTINCT("Booster_Version"),"PAYLOAD_MASS__KG_" from SPACEXTABLE
    WHERE "PAYLOAD_MASS__KG_" IN ( SELECT MAX("PAYLOAD_MASS__KG_") FROM SPACEXTABL
```

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

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

2015 Launch Records

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

```
%%sql
select substr("Date",6,2) as "Num_Month",substr("Date",0,5) as "Num_Year", "Booster_Version","Launch_Site","Landing_Outcome"
    from SPACEXTABLE where Landing_Outcome='Failure (drone ship)' and Num_Year = '2015';
```

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

Num_Month	Num_Year	Booster_Version	Launch_Site	Landing_Outcome
01	2015	F9 v1.1 B1012	CCAFS LC-40	Failure (drone ship)
04	2015	F9 v1.1 B1015	CCAFS LC-40	Failure (drone ship)

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

- Rank of 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.
- It is observed that in this data extract there are more failures than successes.

```
%%sql
SELECT DATE, LANDING_OUTCOME FROM SPACEXTBL WHERE DATE BETWEEN '2010-06-04' AND '2017-03-20' AND
 LANDING_OUTCOME LIKE "Success (ground pad)%" OR LANDING_OUTCOME LIKE "Failure (drone ship)%"  ORDER BY "DATE" DESC;
```

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

Date	Landing_Outcome
2017-02-19	Success (ground pad)
2016-07-18	Success (ground pad)
2016-06-15	Failure (drone ship)
2016-03-04	Failure (drone ship)
2016-01-17	Failure (drone ship)
2015-12-22	Success (ground pad)
2015-04-14	Failure (drone ship)
2015-01-10	Failure (drone ship)

Section 3

Launch Sites Proximities Analysis

Launch sites in a map

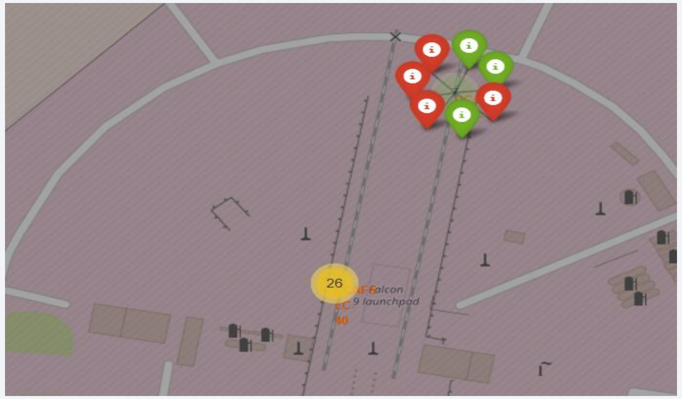
- The map shows some circles in which SpaceX launches are accumulated, representing the different launch sites.

```
    # Add the Circle and Marker to the map
    site_map.add_child(circle)
    site_map.add_child(marker)

# Display the map
site_map
```

Identification on the map

- You can identify by color whether the success rate is high or low at a high level (green, yellow, or red), and at a low level, you can see whether a launch was successful or not (green or red).



Proximity/Security

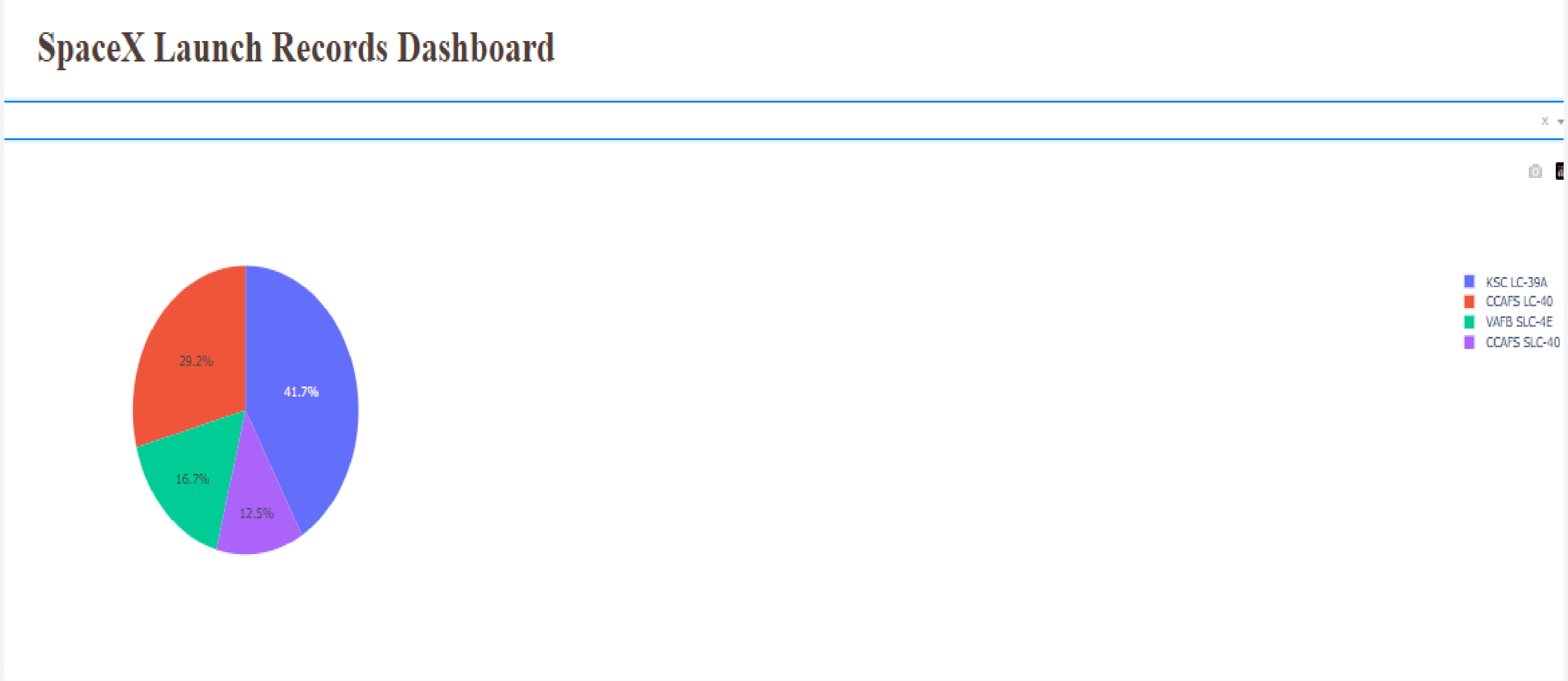
- The proximity to cities, roads and coasts is important for the safety of both the launch and the general population.
- The distance to the nearest road can be identified as less than 0.6 km.

Section 4

Build a Dashboard with Plotly Dash

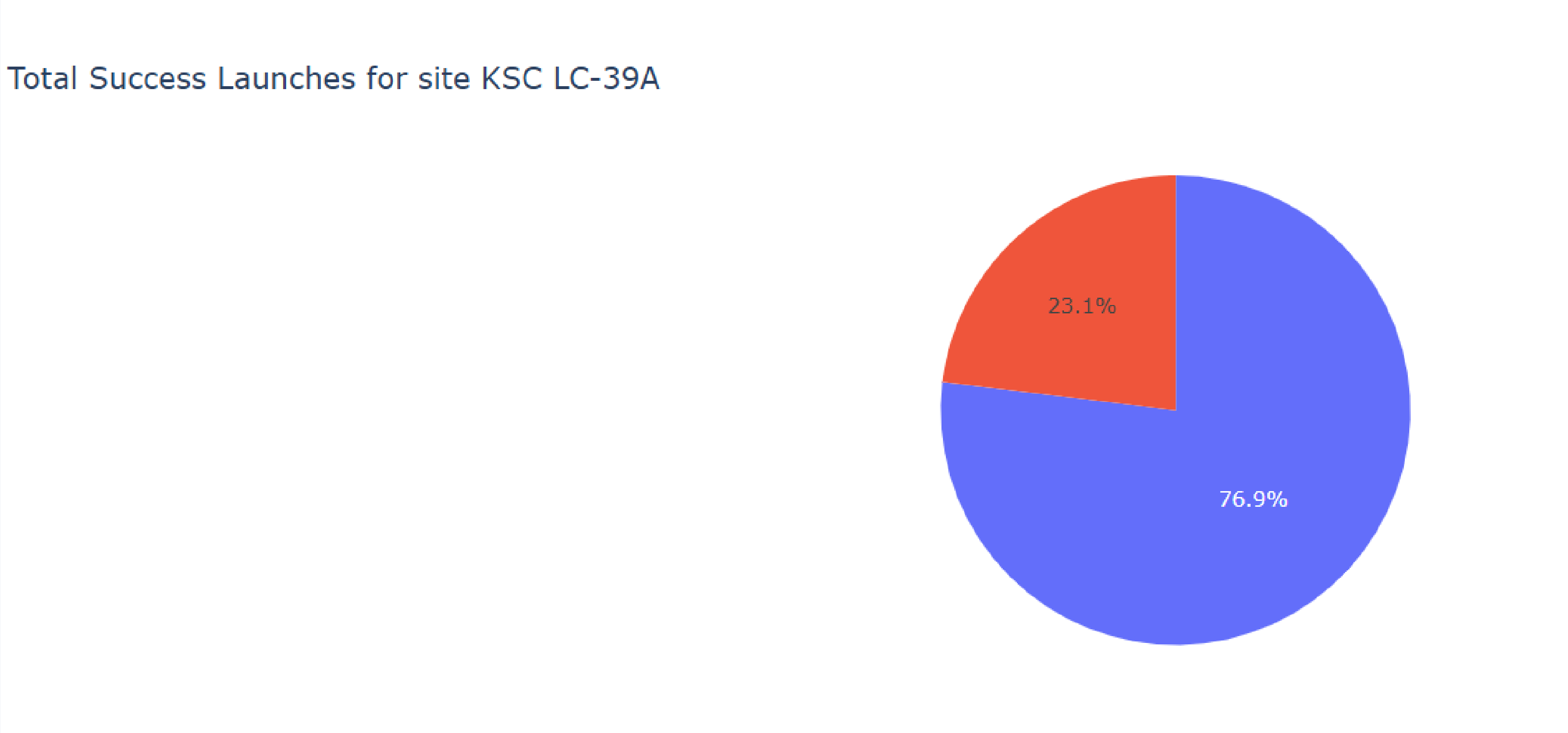
Ratio of launches per launch site

- The proportion of launches carried out by each different site is shown. The KSL LC-39A site accounts for almost 50% of the total.



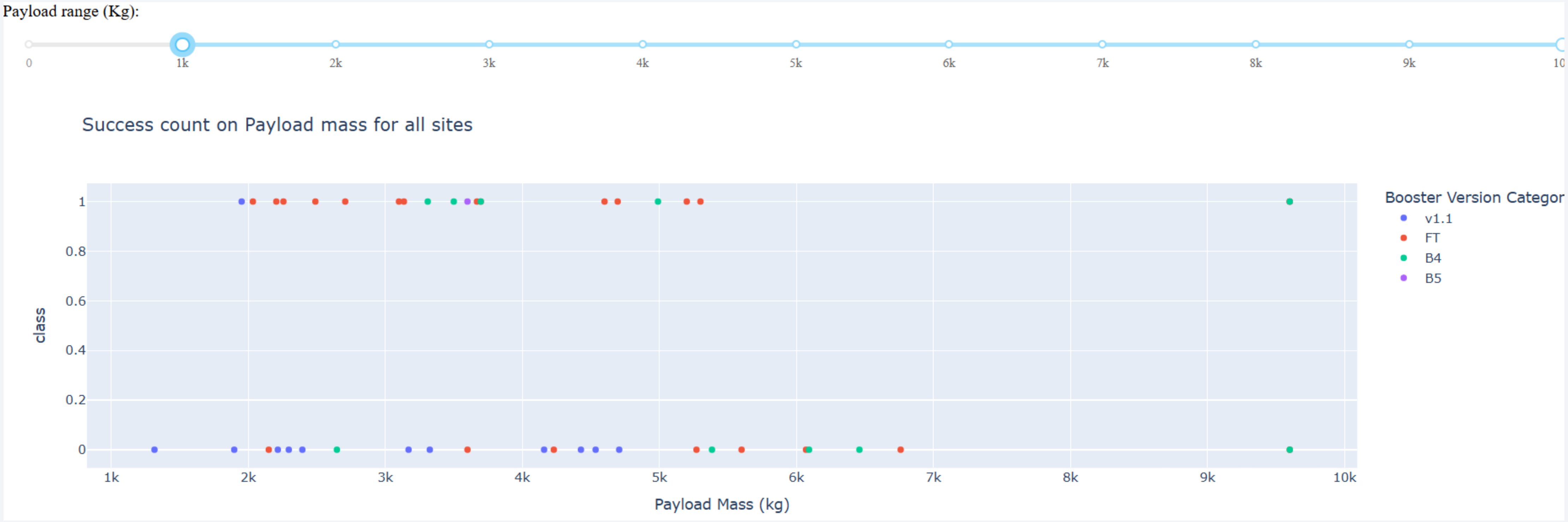
<Dashboard Screenshot 2>

- KSC LC-39A is the launch site with the highest success rate.



<Dashboard Screenshot 3>

- The booster version with the highest success ratio is B4 and FT for Falcon 9.
- The booster version with the highest Payload Mass with success is B4

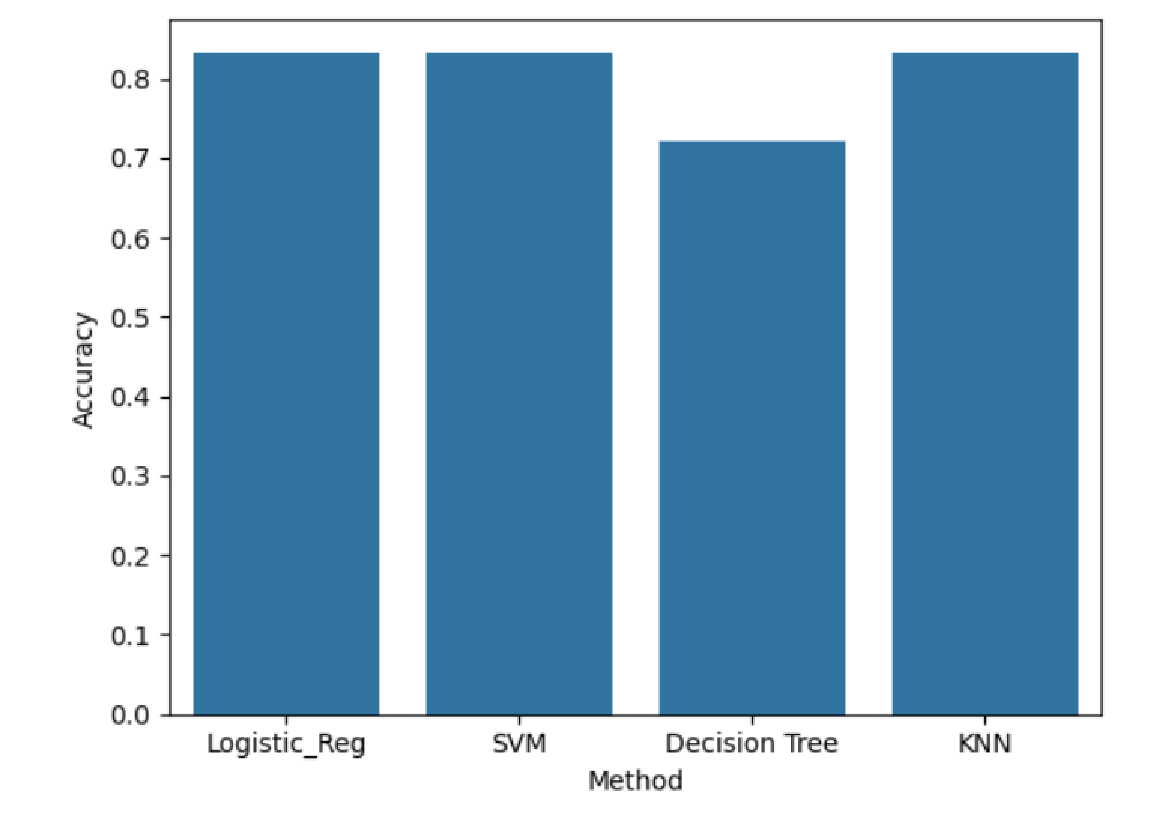


Section 5

Predictive Analysis (Classification)

Classification Accuracy

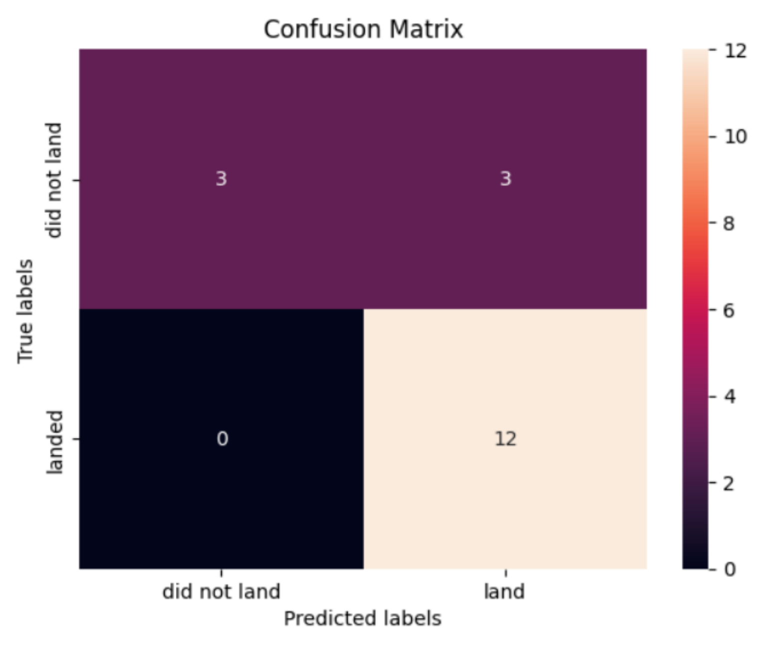
- There is a tie between three models: KNN, Logistic Regression and SVM with 83.3% accuracy.



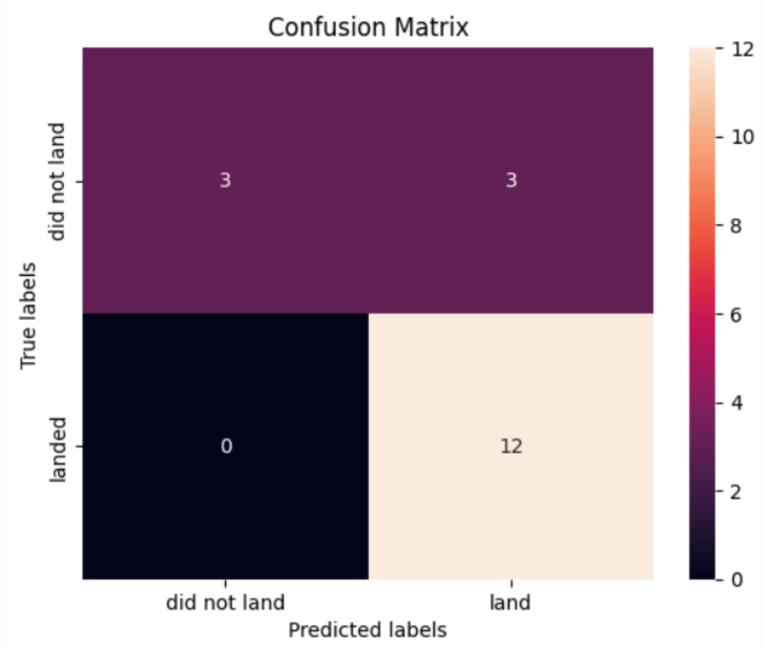
Confusion Matrix

- The confusion matrix of each model is shown, which has an accuracy of 83.3%.

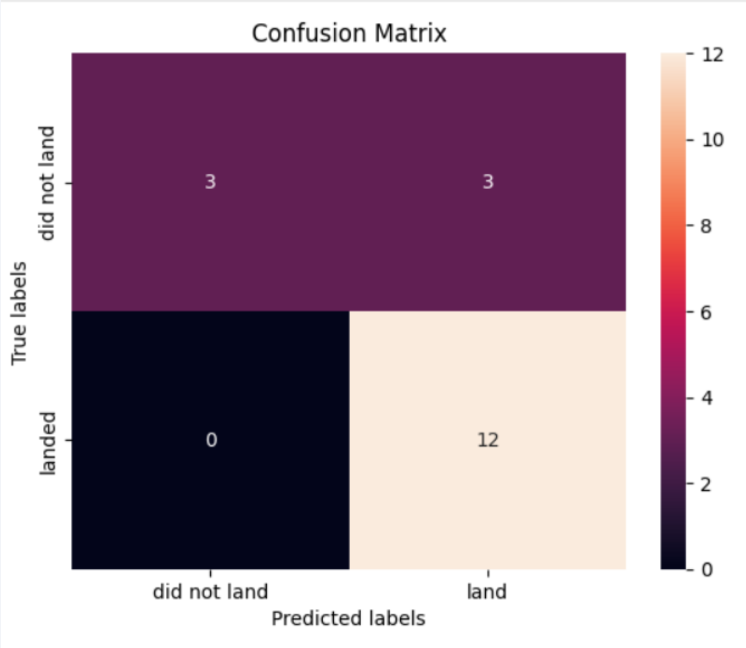
Logistic Regression



SVM



KNN



Conclusions

- SpaceX launches have a tendency to increase their success rate as well as standardize the orbits to follow.
- All the current results are the product of experimentation and continuous improvement, since there are launch configurations which have a success rate of less than 60%, but statistics show significant progress in subsequent models.
- In this particular case and moment, that is, with the context and data currently provided, it is possible to use more than one method to predict whether a launch will be successful or not with an accuracy of 83.3% in its qualification (KNN, SVM or Log Reg)

- While more than one model can be used at this time, this may change as well as future data from SpaceX.

Appendix

Thank you!