SpaceX - To infinity and beyond"

Leonardo Araújo

07/01/2022

OUTLINE

- Executive Summary
- Introduction
- Methodology
- Results
  - Visualization – Charts
  - Dashboard
- Discussion
  - Findings & Implications
- Conclusion

Appendix

EXECUTIVE SUMMARY

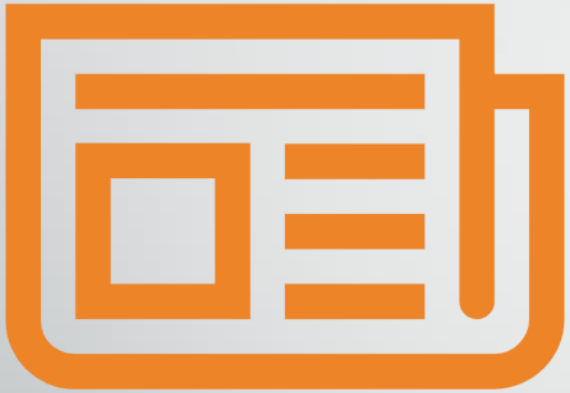
- Introduction
- Methodology
  - Data Collection and Data Wrangling
    - Results
  - Exploratory Data Analysis
  - Predictive Analysis
- Results
  - Exploratory Data Analysis
  - SQL
  - Folium
  - Dashboard
  - Predictive Analysis
- Conclusion
- Appendix

INTRODUCTION

- SpaceX is a space company that manages to reuse early stage rockets, making launching costs much cheaper. However, landings do not always occur successfully. So, predicting the success/failure rate of releases is crucial for planning the best financial results to the company.
- In this report, we will analyze historical data from rocket launches and test which machine learning technique is best suited to predicting the success rate of launches.

METHODOLOGY

Data Collection and Data Wrangling



Data were collected from two sources:

- Using a API
- Using Webscramping from Wikipedia Page

The main Python libraries used were:

- Requests
- Pandas
- Numpy
- Bs4

METHODOLOGY

Data Collection – API - Selected codes

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

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

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

Out[10]: 200

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

In [11]:
```
# Use json_normalize meethod to convert the json result into a dataframe
dados=response.json()
data=pd.json_normalize(dados)
```

Task 3: Dealing with Missing Values

Calculate below the mean for the `PayloadMass` using the `.mean()`. Then use the mean and the `.replace()` function to replace `np.nan` values in the data with the mean you calculated.

In [27]:
```
# Calculate the mean value of PayloadMass column
a=data_falcon9['PayloadMass'].mean()
data_falcon9.loc[:,'PayloadMass']=data_falcon9['PayloadMass'].replace(np.nan,a)
# Replace the np.nan values with its mean value
data_falcon9
```

```
/opt/conda/envs/Python-3.7-OpenCE/lib/python3.7/site-packages/pandas/core/indexing.py:966: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  self.obj[item] = s
```

Out[27]:

	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins	Reused	Legs	LandingPad	Block	ReusedCount	Serial	Longitude	Latitude
4	1	2010-06-04	Falcon 9	6123.547647	LEO	CCSFS SLC 40	None None	1	False	False	False	None	1.0	0	B0003	-80.577366	28.561857
5	2	2012-05-22	Falcon 9	525.000000	LEO	CCSFS SLC 40	None None	1	False	False	False	None	1.0	0	B0005	-80.577366	28.561857
6	3	2013-03-01	Falcon 9	677.000000	ISS	CCSFS SLC 40	None None	1	False	False	False	None	1.0	0	B0007	-80.577366	28.561857
7	4	2013-09-29	Falcon 9	500.000000	PO	VAFB SLC 4E	False Ocean	1	False	False	False	None	1.0	0	B1003	-120.610829	34.632093
8	5	2013-12-03	Falcon 9	3170.000000	GTO	CCSFS SLC 40	None None	1	False	False	False	None	1.0	0	B1004	-80.577366	28.561857
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
89	86	2020-09-03	Falcon 9	15600.000000	VLEO	KSC LC 39A	True ASDS	2	True	True	True	5e9e3032383ecb6bb234e7ca	5.0	8	B1060	-80.603956	28.608058
90	87	2020-10-06	Falcon 9	15600.000000	VLEO	KSC LC 39A	True ASDS	3	True	True	True	5e9e3032383ecb6bb234e7ca	5.0	8	B1058	-80.603956	28.608058
91	88	2020-10-18	Falcon 9	15600.000000	VLEO	KSC LC 39A	True ASDS	6	True	True	True	5e9e3032383ecb6bb234e7ca	5.0	10	B1051	-80.603956	28.608058
92	89	2020-10-24	Falcon 9	15600.000000	VLEO	CCSFS SLC 40	True ASDS	3	True	True	True	5e9e3033383ecbb9e534e7cc	5.0	8	B1060	-80.577366	28.561857
93	90	2020-11-05	Falcon 9	3681.000000	MEO	CCSFS SLC 40	True ASDS	1	True	False	True	5e9e3032383ecb6bb234e7ca	5.0	3	B1062	-80.577366	28.561857

METHODOLOGY

Data Collection – WebScraping – Selected codes

TASK 1: Request the Falcon9 Launch Wiki page from its URL

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

In [5]:
```
# use requests.get() method with the provided static_url
# assign the response to a object
response=requests.get(static_url)
```

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

In [9]:
```
# Use BeautifulSoup() to create a BeautifulSoup object from a response text content
soup=BeautifulSoup(response.text,'html5lib')
```

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

In [11]:
```
soup.title
```

Out[11]:
```
<title>List of Falcon 9 and Falcon Heavy launches - Wikipedia</title>
```

In [10]:
```
column_names = []

for n in first_launch_table.find_all('th'):
    x=extract_column_from_header(n)
    if x is not None and len(x)>0:
        column_names.append(x)
# 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
column_names
```

Out[10]:
```
['Flight No.',
 'Date and time ( )',
 'Launch site',
 'Payload',
 'Payload mass',
 'Orbit',
 'Customer',
 'Launch outcome']
```

Check the extracted column names

In [11]:
```
print(column_names)
```

```
['Flight No.', 'Date and time ( )', 'Launch site', 'Payload', 'Payload mass', 'Orbit', 'Customer', '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.

In [15]:
```
df= pd.DataFrame({ key:pd.Series(value) for key, value in launch_dict.items() })

df.head()
```

Out[15]:

	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 v1.0B0003.1	Failure	4 June 2010	18:45
1	2	CCAFS	Dragon	0	LEO	NASA (COTS)\nNRO	Success	F9 v1.0B0004.1	Failure	8 December 2010	15:43
2	3	CCAFS	Dragon	525 kg	LEO	NASA (COTS)	Success	F9 v1.0B0005.1	No attempt\n	22 May 2012	07:44
3	4	CCAFS	SpaceX CRS-1	4,700 kg	LEO	NASA (CRS)	Success\n	F9 v1.0B0006.1	No attempt	8 October 2012	00:35
4	5	CCAFS	SpaceX CRS-2	4,877 kg	LEO	NASA (CRS)	Success\n	F9 v1.0B0007.1	No attempt\n	1 March 2013	15:10

METHODOLOGY

Data Collection - Data wrangling

TASK 4: Create a landing outcome label from Outcome column

Using the `Outcome`, create a list where the element is zero if the corresponding row in `Outcome` is in the set `bad_outcome`; otherwise, it's one. Then assign it to the variable `landing_class`:

In [33]:
```
# landing_class = 0 if bad_outcome
# landing_class = 1 otherwise
landing_class=[]
for n in df['Outcome']:
    if n in bad_outcomes:
        n=0
        landing_class.append(n)
    else:
        n=1
        landing_class.append(n)

landing_class
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
/tmp/wsuser/ipykernel_414/112306575.py in <module>
     10         landing_class.append(n)
     11
---> 12 landing_class.value_counts()

AttributeError: 'list' object has no attribute 'value_counts'
```

This variable will represent the classification variable that represents the outcome of each launch. If the value is zero, the first stage did not land successfully; one means the first stage landed Successfully

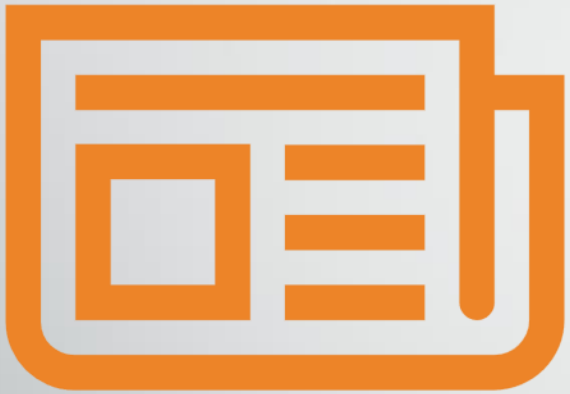
In [35]:
```
df['Class']=landing_class
df[['Class']].head(5)
```

Out[35]:

	Class
0	0
1	0
2	0
3	0
4	0

METHODOLOGY

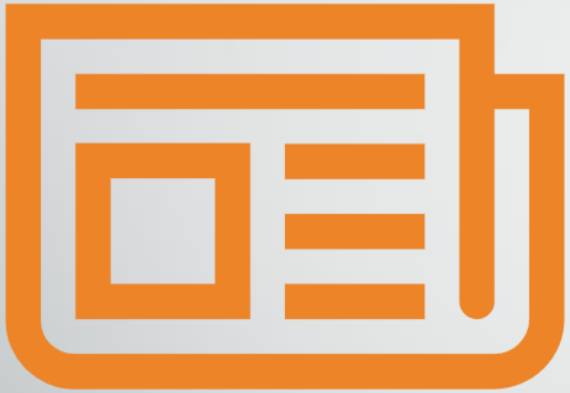
Exploratory Data Analysis and Interractivy



- Data needed to be filtered to analyze only the Falcon 9 rocket
- Some null payload mass records were converted to average.
- The Features that would be used in the model were selected and categoric dimensions were transformed into numerical ones.
- Data were plotted on a Foulium map and a Dash Dashboard.

METHODOLOGY

Predictive Analysis



- After processing the data, 4 machine learning techniques were tested
- to evaluate which of these would be better to predict the success rate of the launches.
  - SVM
  - LogReg
  - KNN
  - Decision Tree

RESULTS
Exploratory Data Analysis - SQL

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

```
In [85]: %sql select DISTINCT LAUNCH_SITE,count(LAUNCH_SITE) from SPACEXTBL group by LAUNCH_SITE
```

```
 * ibm_db_sa://ydk42940:***@98538591-7217-4024-b027-8baa776ffad1.c3n41cmd0nqnrk39u98g.databases.appdomain.cloud:30875/bludb
Done.
```

Out[85]:

launch_site	2
CCAFS LC-40	26
CCAFS SLC-40	34
KSC LC-39A	25
VAFB SLC-4E	16

Task 9

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

```
In [62]: %sql select distinct landing__outcome,booster_version,booster_version from spacextbl where landing__outcome like '%Failure%drone%'
```

```
 * ibm_db_sa://ydk42940:***@98538591-7217-4024-b027-8baa776ffad1.c3n41cmd0nqnrk39u98g.databases.appdomain.cloud:30875/bludb
Done.
```

Out[62]:

landing__outcome	booster_version	booster_version_1
Failure (drone ship)	F9 FT B1020	F9 FT B1020
Failure (drone ship)	F9 FT B1024	F9 FT B1024
Failure (drone ship)	F9 v1.1 B1012	F9 v1.1 B1012
Failure (drone ship)	F9 v1.1 B1015	F9 v1.1 B1015
Failure (drone ship)	F9 v1.1 B1017	F9 v1.1 B1017

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

```
In [49]: %sql select distinct booster_version from spacextbl where payload_mass__kg_=(select max(payload_mass__kg_) from spacextbl )
```

```
 * ibm_db_sa://ydk42940:***@98538591-7217-4024-b027-8baa776ffad1.c3n41cmd0nqnrk39u98g.databases.appdomain.cloud:30875/bludb
Done.
```

Out[49]:

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

Rank the count of landing outcomes (such as Failure (drone ship) or Success (ground pad)) between the date 2010-06-04 and 2017-03-20, in descending order

```
In [84]: %sql select landing__outcome,count(landing__outcome) as x from spacextbl where DATE BETWEEN '2010-06-04' AND '2017-03-20' group by landing__outcome ORDER BY count(landing__outcome) desc

#%sql select * from spacextbl where dayname(date)='Friday'
```

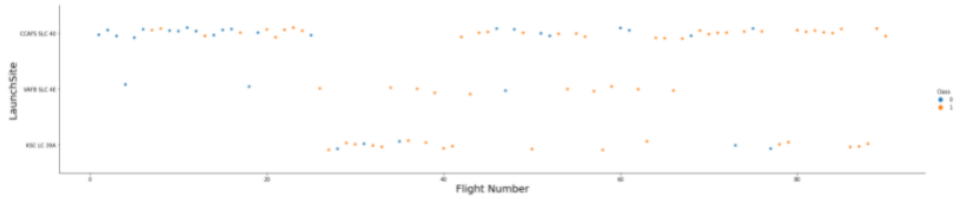
```
 * ibm_db_sa://ydk42940:***@98538591-7217-4024-b027-8baa776ffad1.c3n41cmd0nqnrk39u98g.databases.appdomain.cloud:30875/bludb
Done.
```

Out[84]:

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

RESULTS
Exploratory Data Analysis - Python

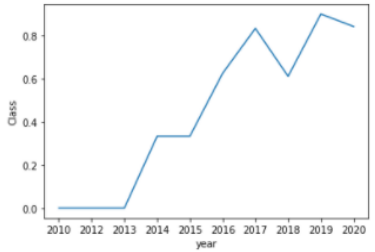
```
In [4]: # Plot a scatter point chart with x axis to be Flight Number and y axis to be the launch site, and hue to be the class value
sns.catplot(y="LaunchSite", x="FlightNumber", hue="Class", data=df, aspect = 5)
plt.xlabel("Flight Number",fontsize=20)
plt.ylabel("LaunchSite",fontsize=20)
plt.show()
```



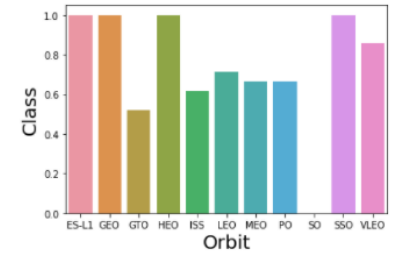

```
In [71]: # Plot a line chart with x axis to be the extracted year and y axis to be the success rate
#year=Extract_year(df['Date'])
#df['year']=year

sns.lineplot(x='year',y='Class',data=df[['year','Class']].groupby(['year'], as_index=False).mean() )
```

```
Out[71]: <AxesSubplot:xlabel='year', ylabel='Class'>
```




```
In [35]: # HINT use groupby method on Orbit column and get the mean of Class column
#List(df)
#df[['Orbit','Class']].groupby(['Orbit']).mean()
sns.barplot(y='Class',x='Orbit',data=df[['Orbit','Class']].groupby(['Orbit'], as_index=False).mean())
plt.xlabel("Orbit",fontsize=20)
plt.ylabel("Class",fontsize=20)
plt.show()
```



Now that our `features_one_hot` dataframe only contains numbers cast the entire dataframe to variable type `float64`

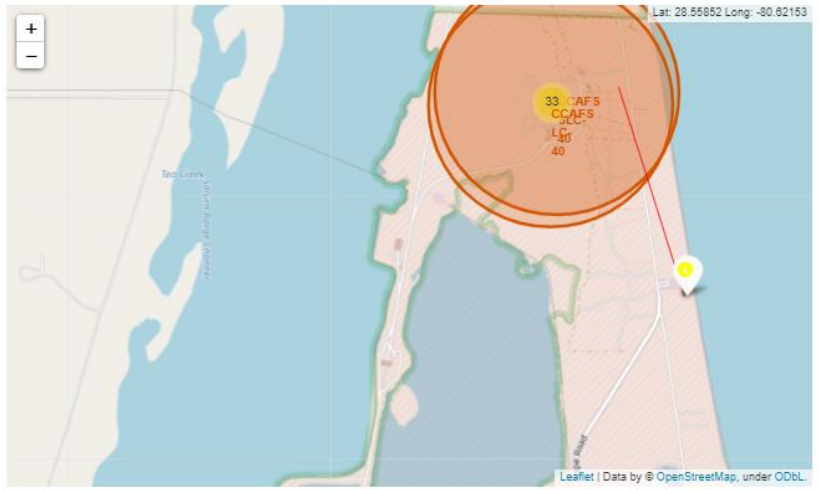
```
In [96]: # HINT: use astype function
features_one_hot=features_one_hot.astype(float)
features_one_hot.shape
```

```
Out[96]: (90, 80)
```

RESULTS
Exploratory Data Analysis - Folium

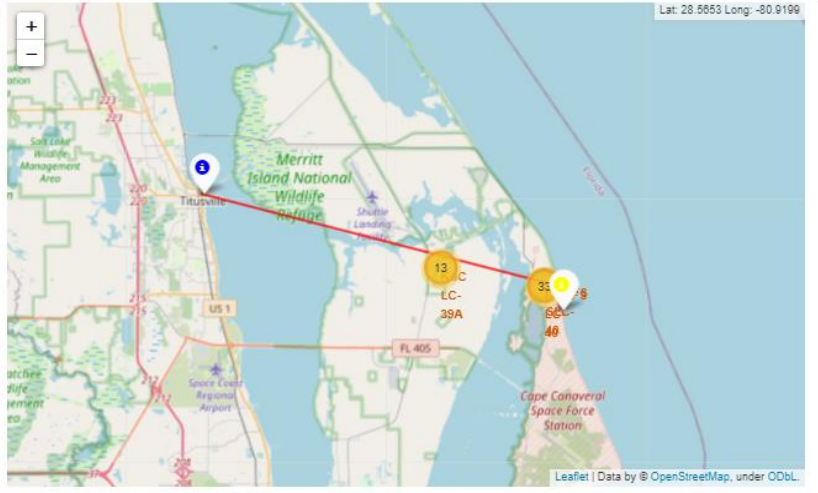
Out[8]:

Out[18]:

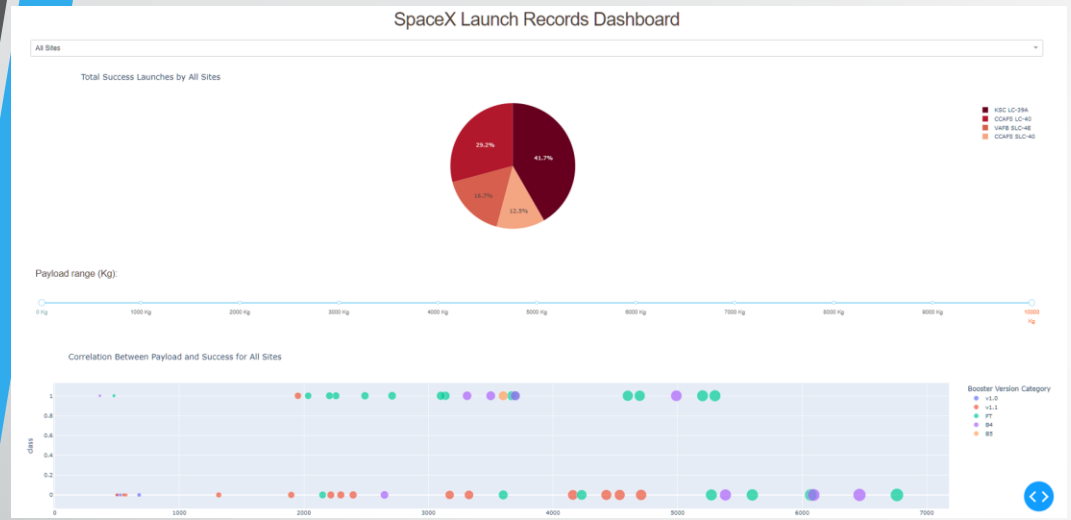


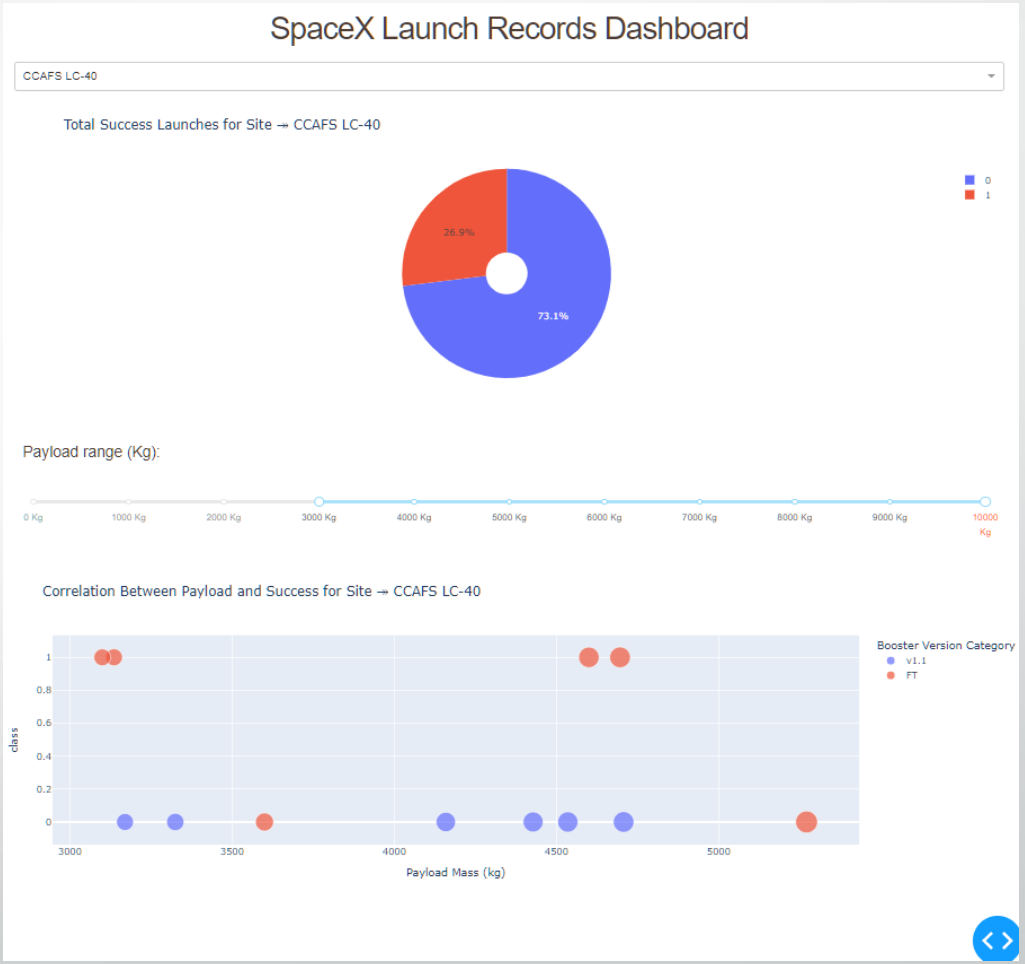
Out[13]:

Out[20]:



RESULTS
Exploratory Data Analysis - Dash





RESULTS
Landing Prediction

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

In [6]:
```
# students get this
transform = preprocessing.StandardScaler()
```

In [7]:
```
X=transform.fit(X).transform(X)
```

In [8]:
```
X_train, X_test, Y_train, Y_test = train_test_split( X, Y, test_size=0.2, random_state=2)
```

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`.

In [10]:
```
parameters ={'C':[0.01,0.1,1],
             'penalty':['l2'],
             'solver':['lbfgs']}
```

In [11]:
```
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)
```

Out[11]:
```
GridSearchCV(cv=10, estimator=LogisticRegression(),
             param_grid={'C': [0.01, 0.1, 1], 'penalty': ['l2'],
                         'solver': ['lbfgs']})
```

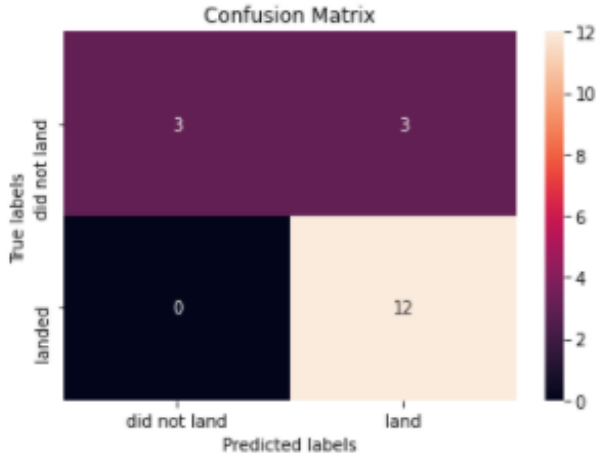
We output the `GridSearchCV` object for logistic regression. We display the best parameters using the data attribute `best_params_` and the accuracy on the validation data using the data attribute `best_score_`.

In [12]:
```
print("tuned hpyerparameters :(best parameters) ",logreg_cv.best_params_)
print("accuracy :",logreg_cv.best_score_)
```

```
tuned hpyerparameters :(best parameters)  {'C': 0.01, 'penalty': 'l2', 'solver': 'lbfgs'}
accuracy : 0.8464285714285713
```

Lets look at the confusion matrix:

In [14]:
```
yhat=logreg_cv.predict(X_test)
plot_confusion_matrix(Y_test,yhat)
```



RESULTS
Landing Prediction

TASK 12

Find the method performs best:

In [33]:
```
a=[logreg_cv.best_score_,svm_cv.best_score_,tree_cv.best_score_,knn_cv.best_score_]
b=['Logistic Regression','Suport Vector Machines','Decision Tree','Knn']
print(a,"Melhor Metodo tree(cv): ",a[2])

d = {'Algorithm': b, 'Accuracy': a}
dfx = pd.DataFrame(data=d)
dfx.sort_values(by=['Accuracy'],ascending=False)
```

```
[0.8464285714285713, 0.8482142857142856, 0.8767857142857143, 0.8482142857142858] Melhor Metodo tree(cv):  0.8767857142857143
```

Out[33]:

	Algorithm	Accuracy
2	Decision Tree	0.876786
3	Knn	0.848214
1	Suport Vector Machines	0.848214
0	Logistic Regression	0.846429

CONCLUSION

- The decision tree is the best algorithm for predicting the rate of successful landings.
- The rate of successful landing launches has been rising since 2013.
- Launches to ES-L1, GEO, HEO and SSO orbits have a higher rate of successful landings