Applied Data Science Capstone

Marcelo Almeida Gomes

22/12/2022

OUTLINE

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

EXECUTIVE SUMMARY

- Capstone Introduction
- Data Collection
  - Data Collection API
  - Data Collection with Web scraping
  - Data Wrangling
- Exploratory Data Analysis with Python
  - EDA using SQL
  - EDA using Pandas and Matplotlib
- Interactive Visual and Dashboard
- Predictive Analysis with Machine Learning
- Conclusions

INTRODUCTION

- SpaceX publishes rocket launches on its website
- The launches are very expensive
- SpaceX can launch with a cost of $62 MM, less than other providers because can reuse first stage
- The job consists to determine
  - If the first stage will land successfully
  - Cost of a launch
- To answers theses questions was necessary to gather information, creating dashboards and training a machine learning model

METHODOLOGY

- Data Extraction
- Data Preparation
- Exploratory Data Analysis
- Machine Learning Model
  - Training
  - Evaluation
  - Validation
- Analysis of Results

METHODOLOGY - PIPELINE

1 - Data Extraction

2 – Data Preparing

3 – Data Analysis

4 – Model Training

5 - Deployment

METHODOLOGY – COLLECTION AND WRANGLING

- Data Collection from web[1] using BeautifulSoup lib from Python
- Create a DataFrame using Pandas
- Data Wrangling using Pandas and Numpy

```
FlightNumber        int64
Date               object
BoosterVersion     object
PayloadMass       float64
Orbit              object
LaunchSite         object
Outcome            object
Flights             int64
GridFins             bool
Reused               bool
Legs                 bool
LandingPad         object
Block             float64
ReusedCount         int64
Serial             object
Longitude         float64
Latitude          float64
dtype: object
```

```
# Apply value_counts on Orbit column
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
```

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

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

```
0.6666666666666666
```

1 https://en.wikipedia.org/wiki/List_of_Falcon_9_and_Falcon_Heavy_launches

METHODOLOGY – CREATE DATAFRAME

```
df=pd.read_csv(dataset_part_1_csv)
df.head(10)
```

	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins	Reused	Legs	LandingPad	Block	ReusedCount	Serial	Longitude	Latitude
0	1	2010-06-04	Falcon 9	6104.959412	LEO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B0003	-80.577366	28.561857
1	2	2012-05-22	Falcon 9	525.000000	LEO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B0005	-80.577366	28.561857
2	3	2013-03-01	Falcon 9	677.000000	ISS	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B0007	-80.577366	28.561857
3	4	2013-09-29	Falcon 9	500.000000	PO	VAFB SLC 4E	False Ocean	1	False	False	False	NaN	1.0	0	B1003	-120.610829	34.632093
4	5	2013-12-03	Falcon 9	3170.000000	GTO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B1004	-80.577366	28.561857
5	6	2014-01-06	Falcon 9	3325.000000	GTO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B1005	-80.577366	28.561857
6	7	2014-04-18	Falcon 9	2296.000000	ISS	CCAFS SLC 40	True Ocean	1	False	False	True	NaN	1.0	0	B1006	-80.577366	28.561857
7	8	2014-07-14	Falcon 9	1316.000000	LEO	CCAFS SLC 40	True Ocean	1	False	False	True	NaN	1.0	0	B1007	-80.577366	28.561857
8	9	2014-08-05	Falcon 9	4535.000000	GTO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B1008	-80.577366	28.561857
9	10	2014-09-07	Falcon 9	4428.000000	GTO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B1011	-80.577366	28.561857

METHODOLOGY – EDA

- First Exploratory Data Analysis (EDA) using Matplotlib and Seaborn libs
- Using SQL to some Analysis
- Iterative Dashboard with Plotly and Dash
- Map analysis with Folium

matplotlib

seaborn

Dash by plotly

SQL

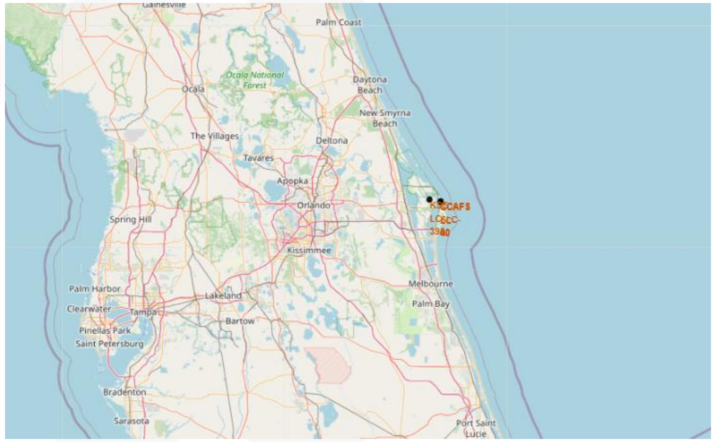
Folium

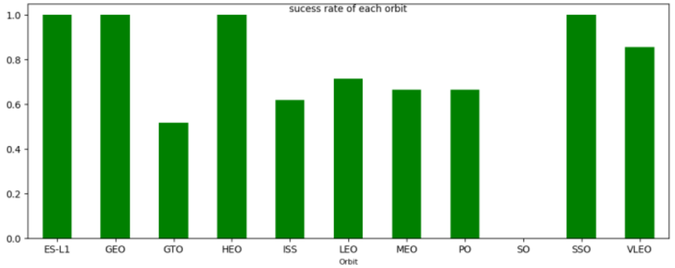
plotly

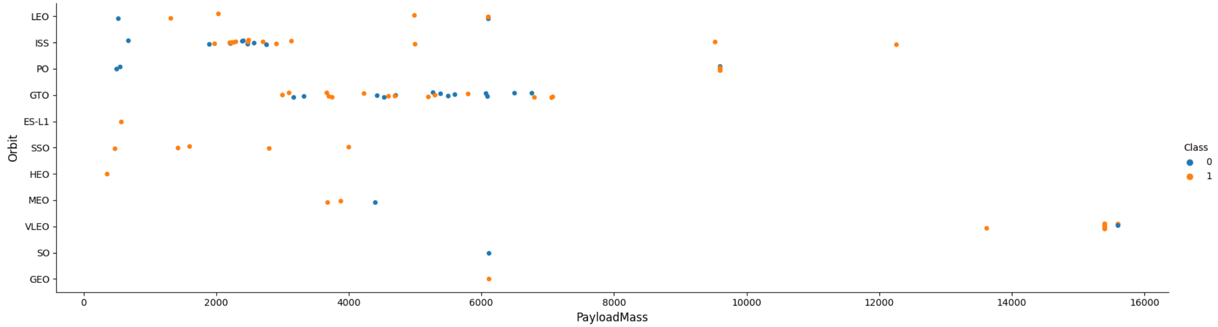
METHODOLOGY – EDA

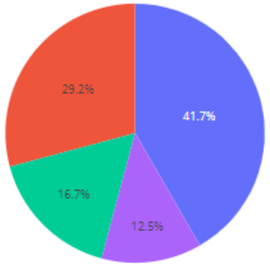
- Exploratory Data Analysis with tables, scatter, maps, bar, pie charts to make findings
- See correlation between variables, counts total, per group and percent

	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins	Reused	Legs	LandingPad	Block	ReusedCount	Serial	Longitude	Latitude	Class
0	1	2010-06-04	Falcon 9	6104.959412	LEO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B0003	-80.577366	28.561857	0
1	2	2012-05-22	Falcon 9	525.000000	LEO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B0005	-80.577366	28.561857	0
2	3	2013-03-01	Falcon 9	677.000000	ISS	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B0007	-80.577366	28.561857	0
3	4	2013-09-29	Falcon 9	500.000000	PO	VAFB SLC 4E	False Ocean	1	False	False	False	NaN	1.0	0	B1003	-120.610829	34.632093	0
4	5	2013-12-03	Falcon 9	3170.000000	GTO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B1004	-80.577366	28.561857	0







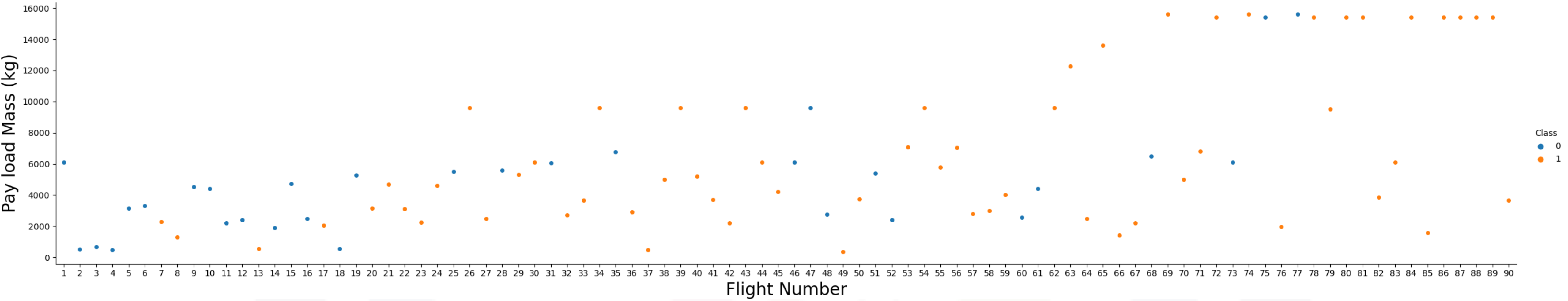


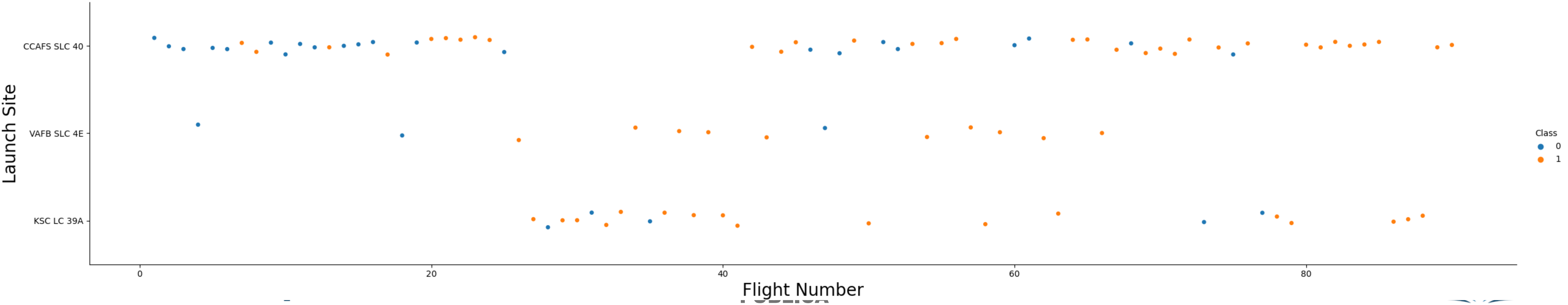
METHODOLOGY – PREDICTIVE ANALYSIS

- How we want to predict whether the launch will be successful the classification algorithms in machine learning was selected
- This is a problem that target attribute is a categorical variable, class can be 0 if launch failed or 1 if successful
- The classification models applied
    - Logistic Regression
    - SVM
    - Decision Tree Classifier
    - KNN
- Use Scikit Learn to build, training, test and validate the model
- The set to data test was 0.2 and then data train 0.8
- Then find the method with best performs
- The Accuracy was used to find method which performs best

RESULTS – EDA

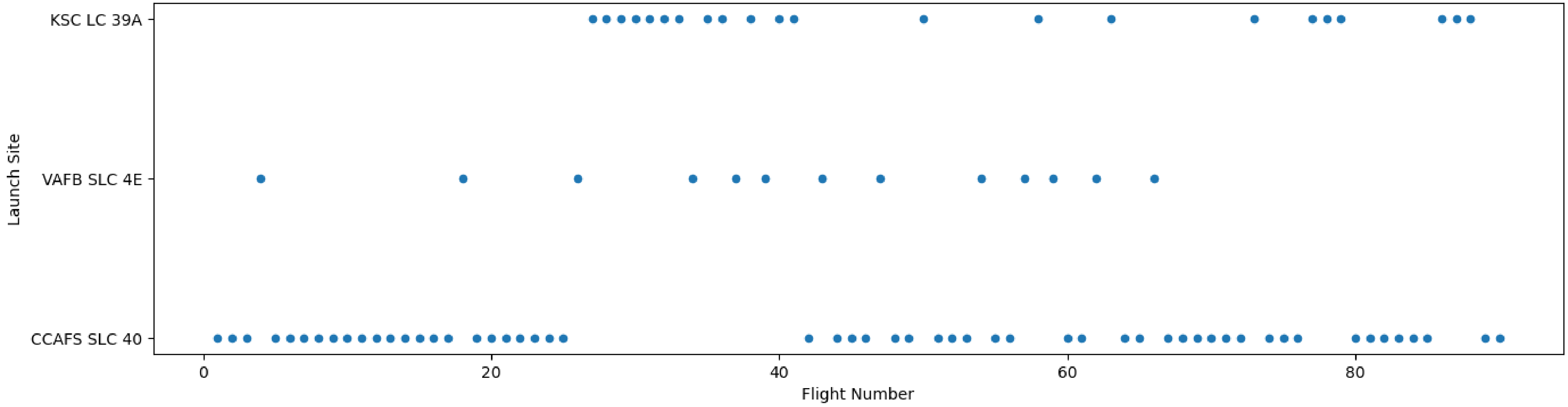
Relation between FlightNumber vs PayLoad and FlightNumber vs Launch Site

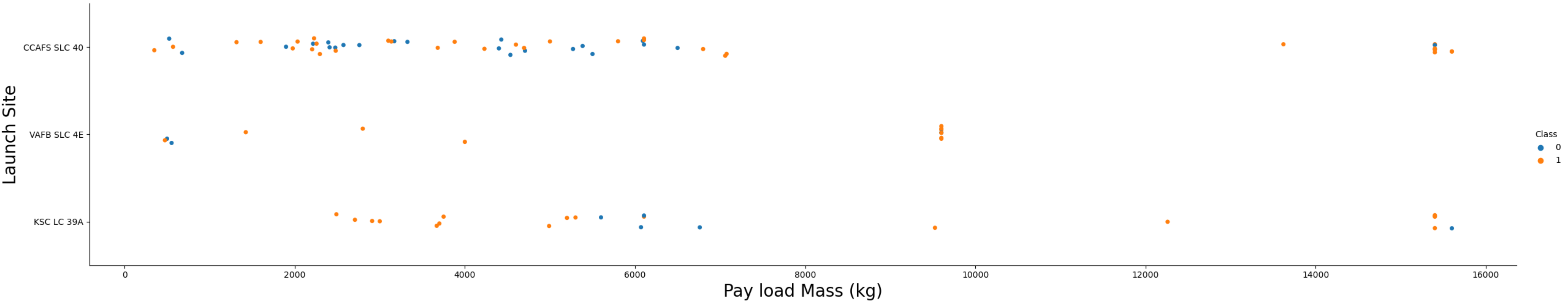




RESULTS – EDA

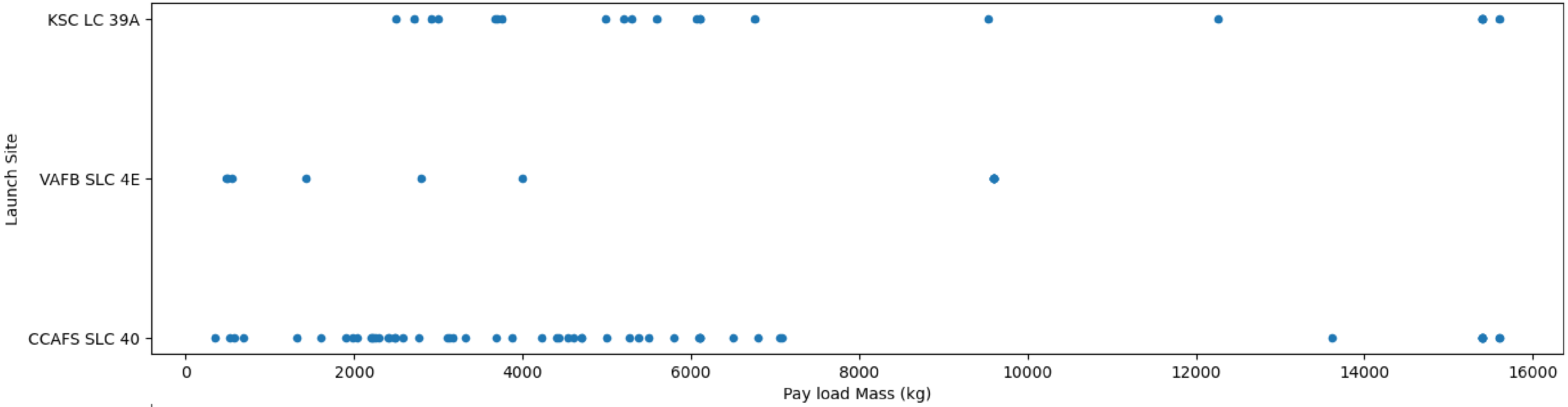
Relation between FlightNumber vs FlightNumber with scatter plot and PayLoad vs Launch Site

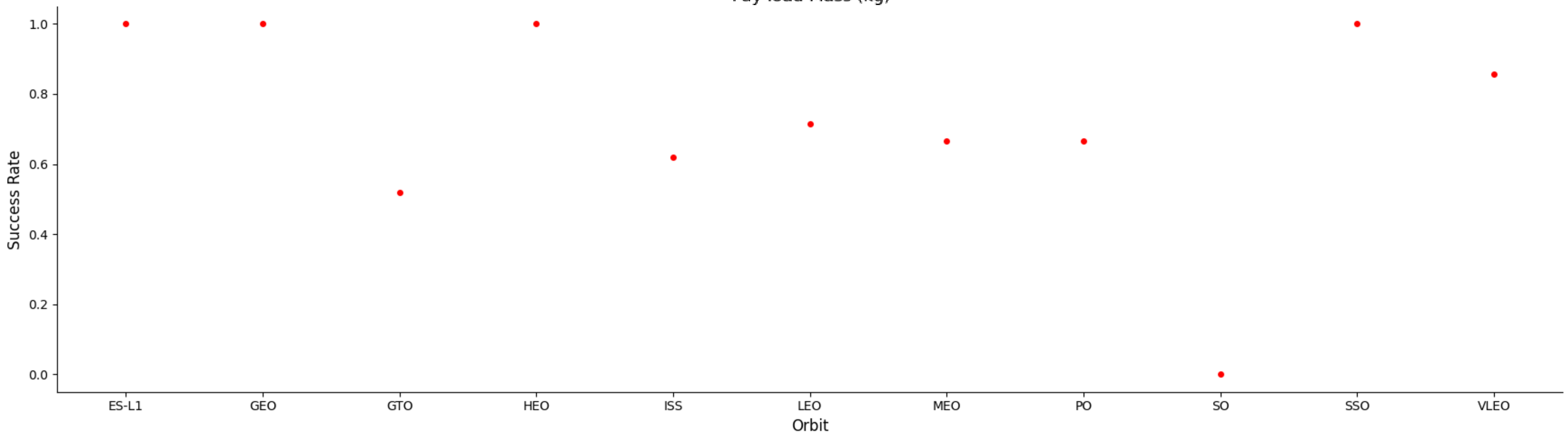




RESULTS – EDA

Relation between PayLoad vs Launch Site with scatter plot and Orbit vs Success Rate

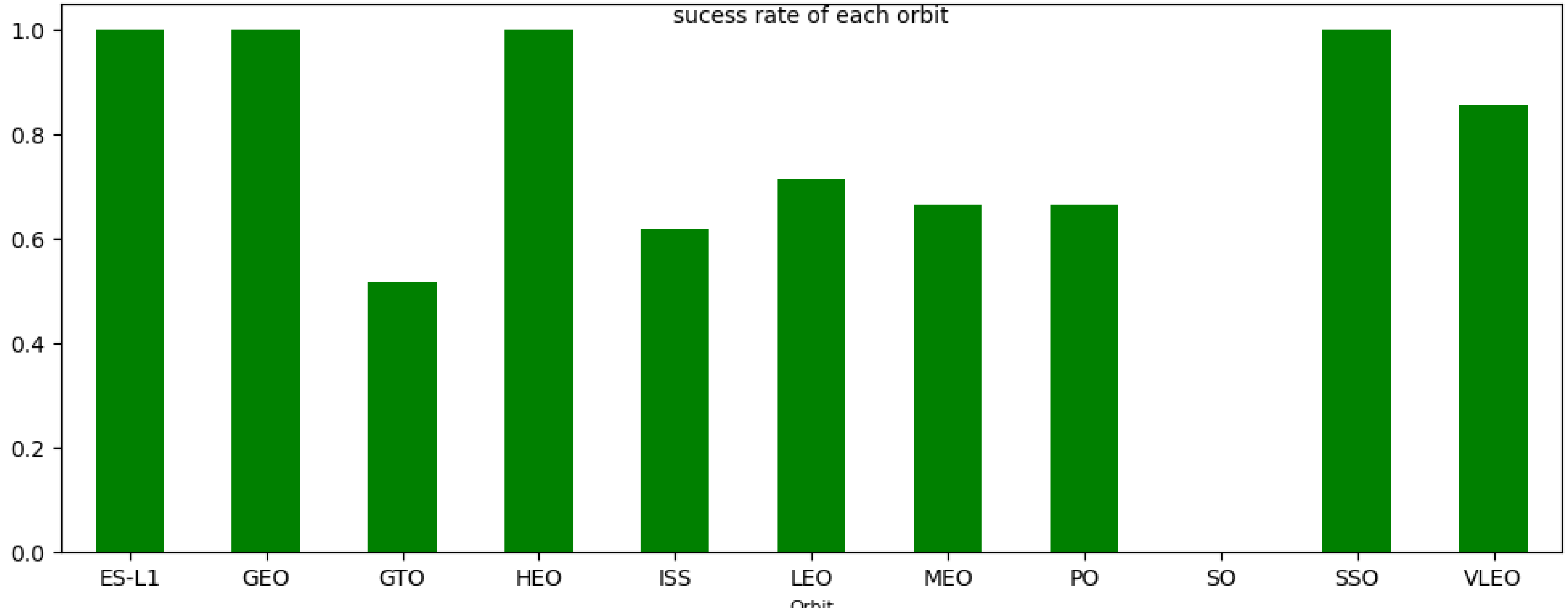




IBM Dev ... RK

RESULTS – EDA

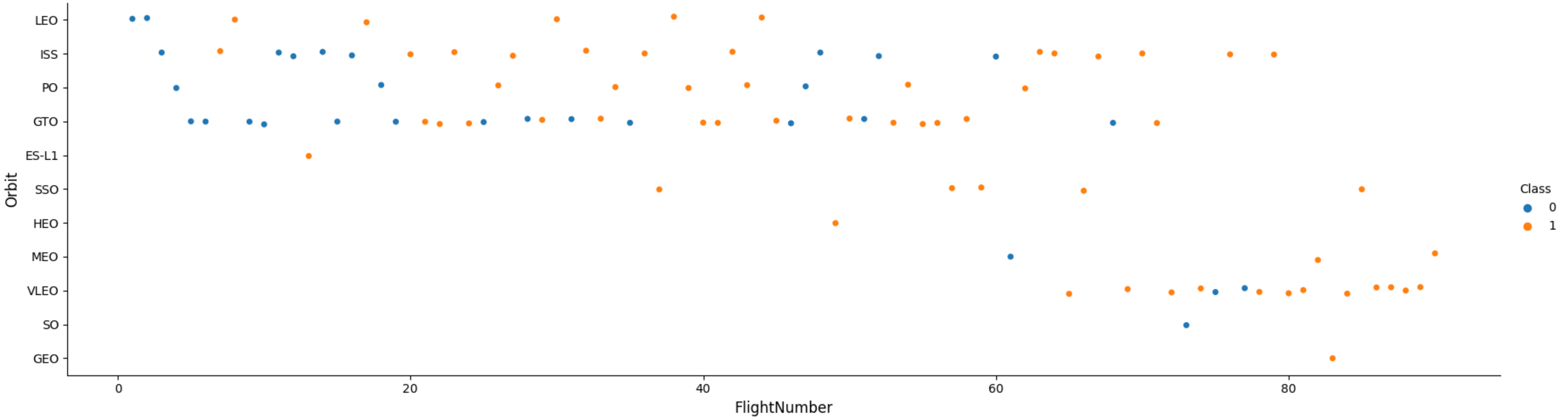
Bar graph showing Success Rate of each orbit

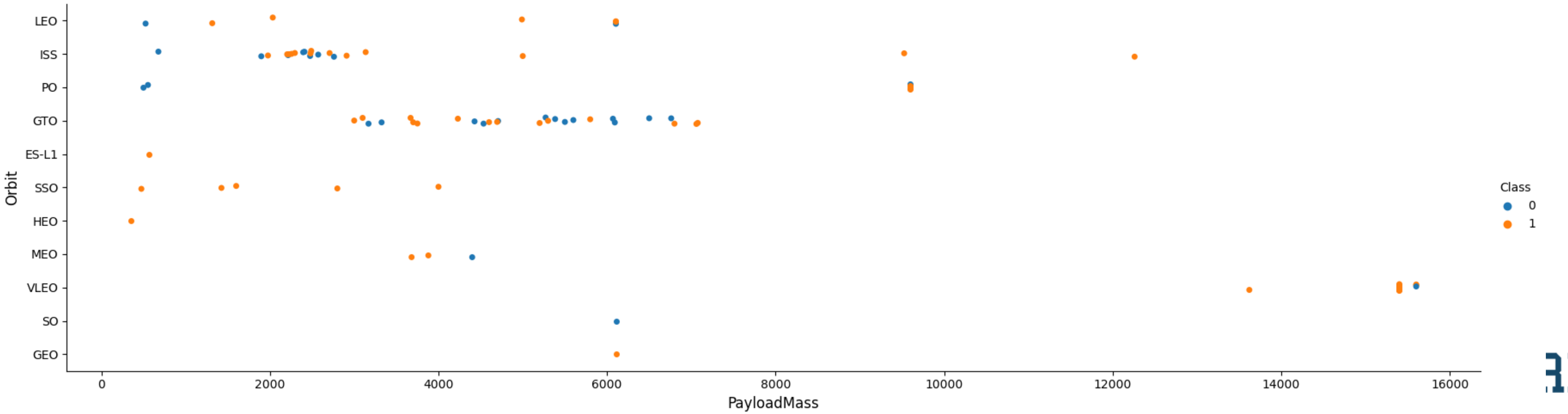



```
# HINT use groupby method on Orbit column and get the mean of Class column
df.groupby("Orbit").mean()['Class'].plot(kind='bar', color = 'green', figsize=(10,4),rot = 0)
plt.xlabel("Orbit",fontsize=8)
plt.ylabel("Success Rate",fontsize=20)
plt.title('sucess rate of each orbit',fontsize=10,loc='center',y=0.94)
plt.show()
```

RESULTS – EDA

Relation between FligthNumber vs Orbit and PayloadMass vs Orbit

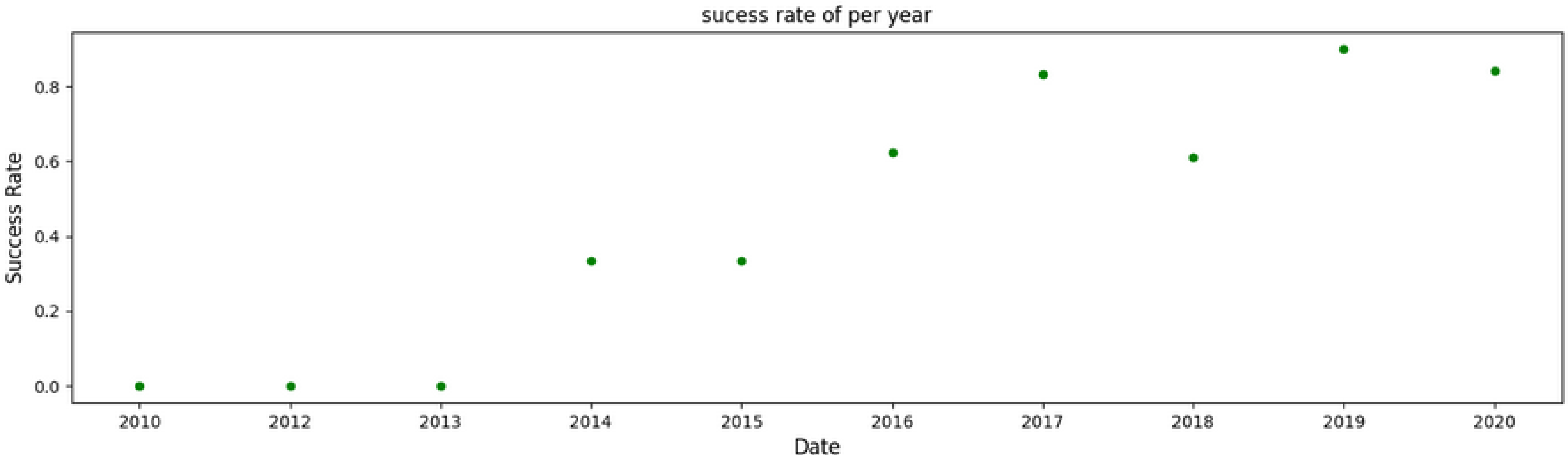




IBM

RESULTS – EDA

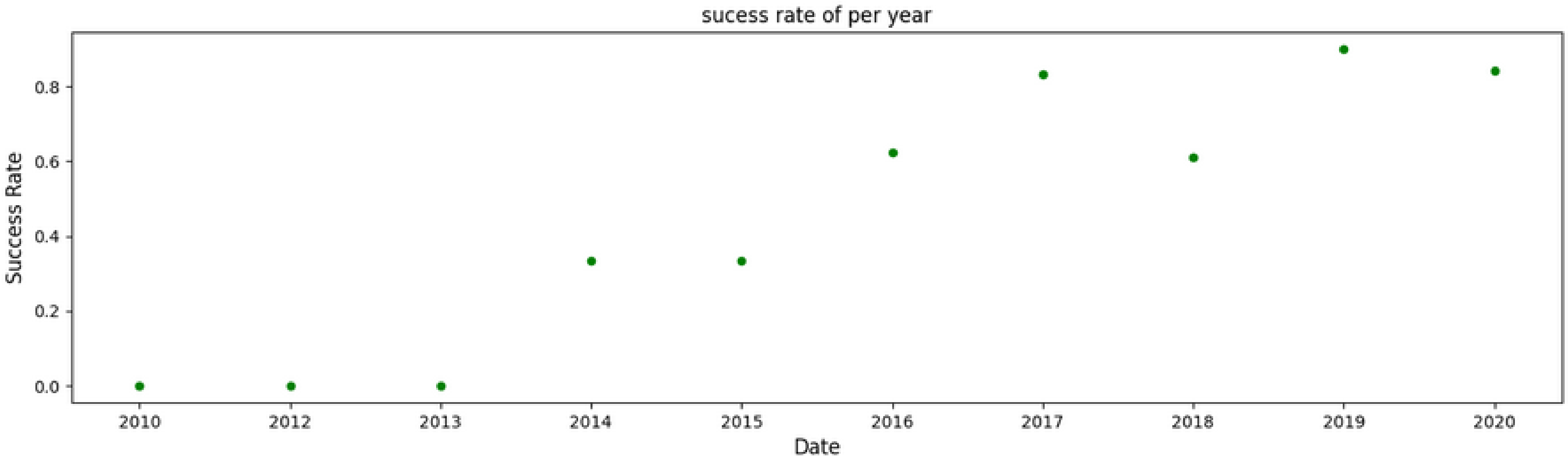
Relation between Success Rate vs Year



It's show that the success rate increasing since 2013 till 2020

RESULTS – EDA

Relation between Success Rate vs Year



It’s show that the success rate increasing since 2013 till 2020

RESULTS – SQL

Task 1

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

```
%sql select distinct(Launch_Site) from SPACEXTBL
```

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

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

Task 2

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

```
%sql select* from SPACEXTBL where Launch_Site like 'CCA%' limit 5
```

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

Date	Time (UTC)	Booster_Version	Launch_Site	Payload	PAYLOAD_MASS__KG_	Orbit	Customer	Mission_Outcome	Landing _Outcome
04-06-2010	18:45:00	F9 v1.0 B0003	CCAFS LC-40	Dragon Spacecraft Qualification Unit	0	LEO	SpaceX	Success	Failure (parachute)
08-12-2010	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)
22-05-2012	07:44:00	F9 v1.0 B0005	CCAFS LC-40	Dragon demo flight C2	525	LEO (ISS)	NASA (COTS)	Success	No attempt
08-10-2012	00:35:00	F9 v1.0 B0006	CCAFS LC-40	SpaceX CRS-1	500	LEO (ISS)	NASA (CRS)	Success	No attempt
01-03-2013	15:10:00	F9 v1.0 B0007	CCAFS LC-40	SpaceX CRS-2	677	LEO (ISS)	NASA (CRS)	Success	No attempt

RESULTS – SQL

Task 3

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

```
%sql select sum(PAYLOAD_MASS__KG_) from SPACEXTBL where Customer like '%NASA (CRS)%'
#%sql select sum(PAYLOAD_MASS__KG_) from SPACEXTBL where Customer in (select upper(Customer) FROM SPACEXTBL where Customer like '%NASA% %CRS)%')
```

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

sum(PAYLOAD_MASS__KG_)
48213

Task 4

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

```
%sql select avg(PAYLOAD_MASS__KG_) as "avg_payload_mass_F9_v1.1_kg" from SPACEXTBL where Booster_Version = 'F9 v1.1'
```

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

avg_payload_mass_F9_v1.1_kg
2928.4

RESULTS – SQL

Task 5

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

Hint:Use min function

```
#%sql select * from SPACEXTBL where "Landing _Outcome" like '%ground pad%' order by Date ASC limit 1
```

```
%sql select min(Date),"Landing _Outcome" from (select * from SPACEXTBL where "Landing _Outcome" like '%ground pad%')
```

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

min(Date)	Landing _Outcome
01-05-2017	Success (ground pad)

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

```
%sql select distinct(Booster_Version) from SPACEXTBL where "Landing _Outcome" like '%Success (drone ship)%' and PAYLOAD_MASS__KG_ BETWEEN 4000 AND 6000
```

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

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

RESULTS – SQL

Task 7

List the total number of successful and failure mission outcomes

```
%sql select Mission_Outcome,count(Mission_Outcome) from SPACEXTBL group by Mission_Outcome like "%Suc%"
```

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

Mission_Outcome	count(Mission_Outcome)
Failure (in flight)	1
Success	100

Task 8

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

```
%sql select Booster_Version from SPACEXTBL where PAYLOAD_MASS__KG_ = (select max(PAYLOAD_MASS__KG_) from SPACEXTBL)
```

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

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

RESULTS – 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, 4, 2) as month to get the months and substr(Date,7,4)='2015' for year.

```
%%sql SELECT substr(Date, 4, 2) as month,booster_version,"Landing _Outcome"
from SPACEXTBL where "Landing _Outcome"
='Failure (drone ship)' and substr(Date,7,4)='2015'
```

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

month	Booster_Version	Landing _Outcome
01	F9 v1.1 B1012	Failure (drone ship)
04	F9 v1.1 B1015	Failure (drone ship)

Task 10

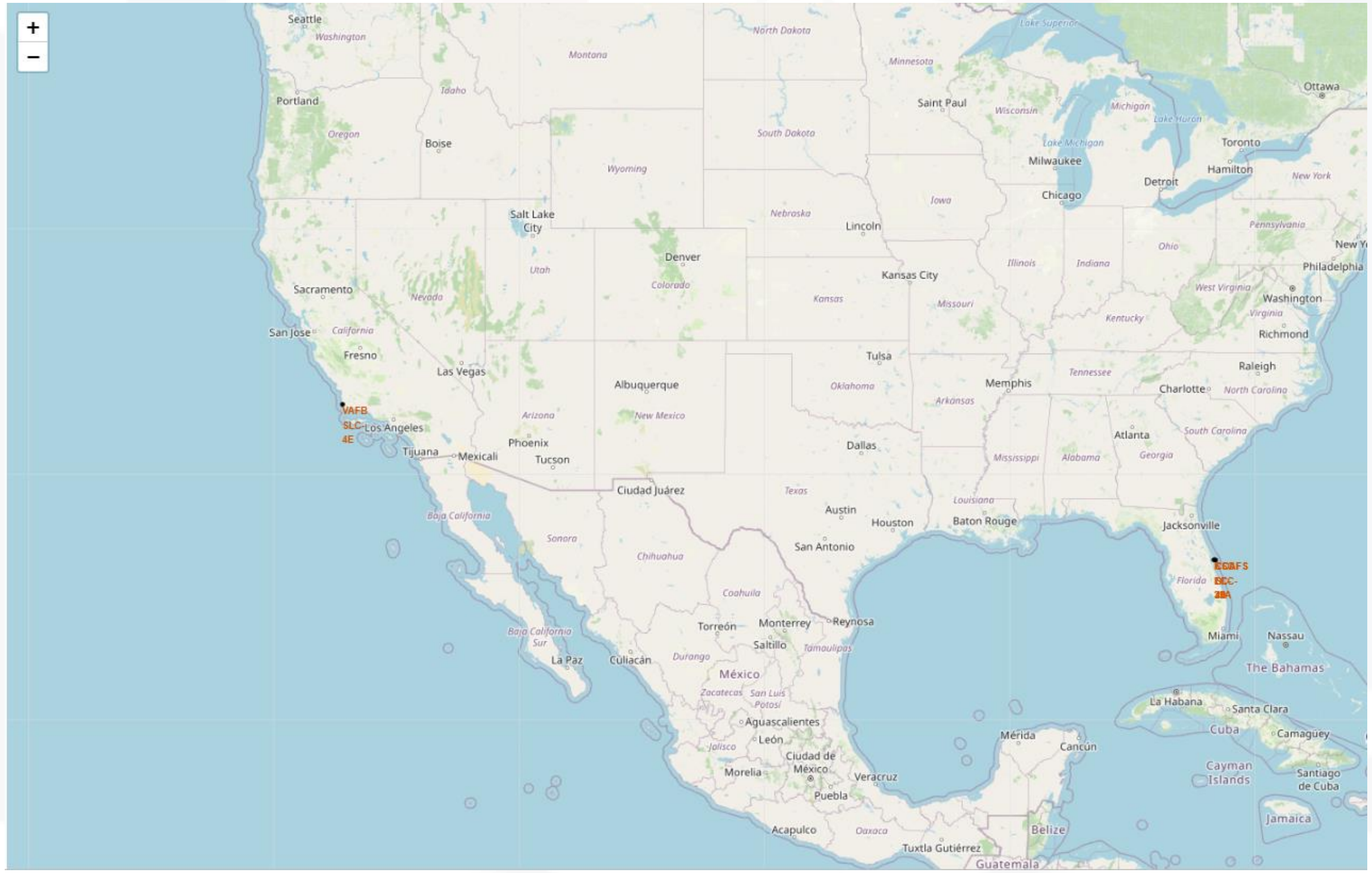
Rank the count of successful landing_outcomes between the date 04-06-2010 and 20-03-2017 in descending order.

```
%%sql
select count("Landing _Outcome") as count_landing_outcome, "Landing _Outcome" from SPACEXTBL
where "Landing _Outcome" like "%Suc%" and substr(Date,7,4) || substr(Date,4,2) || substr(Date,1,2) between '20100604'and '20170320'
group by "Landing _Outcome"
order by count("Landing _Outcome") desc
```

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

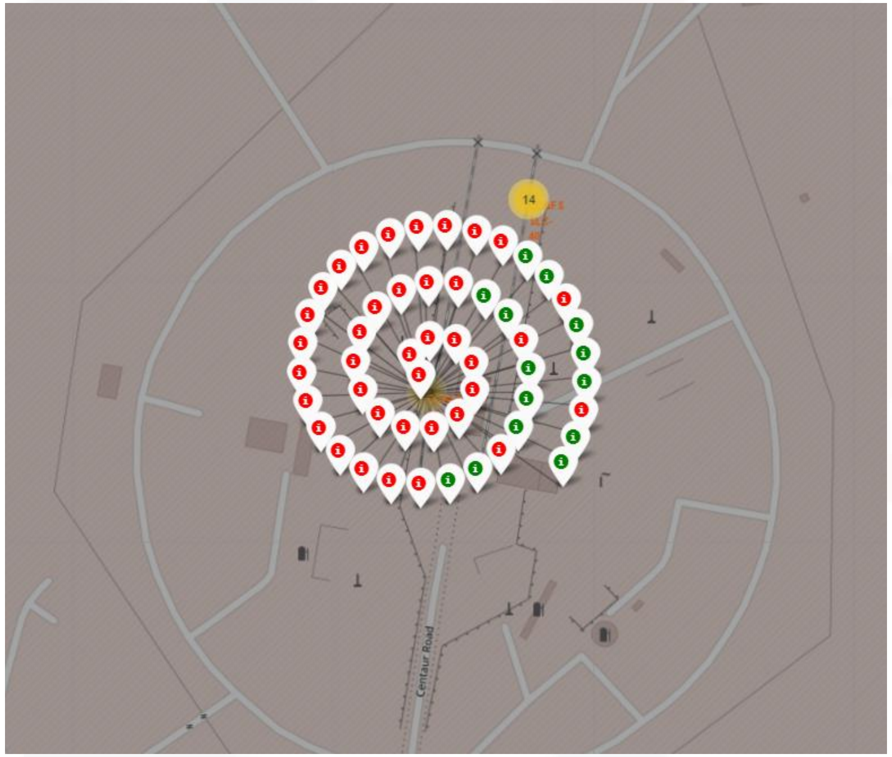
count_landing_outcome	Landing _Outcome
5	Success (drone ship)
3	Success (ground pad)

RESULTS – INTERACTIVE MAP



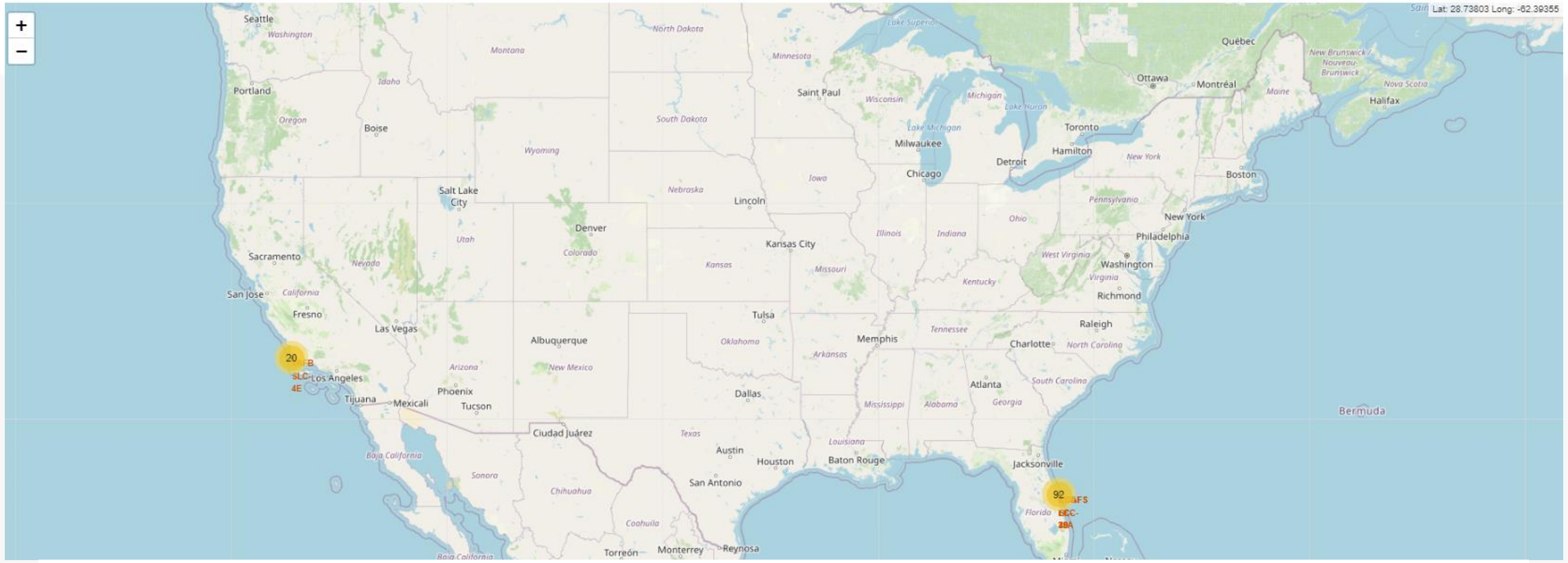
RESULTS – INTERACTIVE MAP

RESULTS – INTERACTIVE MAP



	Launch Site	Lat	Long	class	marker_color
46	KSC LC-39A	28.573255	-80.646895	1	green
47	KSC LC-39A	28.573255	-80.646895	1	green
48	KSC LC-39A	28.573255	-80.646895	1	green
49	CCAFS SLC-40	28.563197	-80.576820	1	green
50	CCAFS SLC-40	28.563197	-80.576820	1	green
51	CCAFS SLC-40	28.563197	-80.576820	0	red
52	CCAFS SLC-40	28.563197	-80.576820	0	red
53	CCAFS SLC-40	28.563197	-80.576820	0	red
54	CCAFS SLC-40	28.563197	-80.576820	1	green
55	CCAFS SLC-40	28.563197	-80.576820	0	red

RESULTS – INTERACTIVE MAP

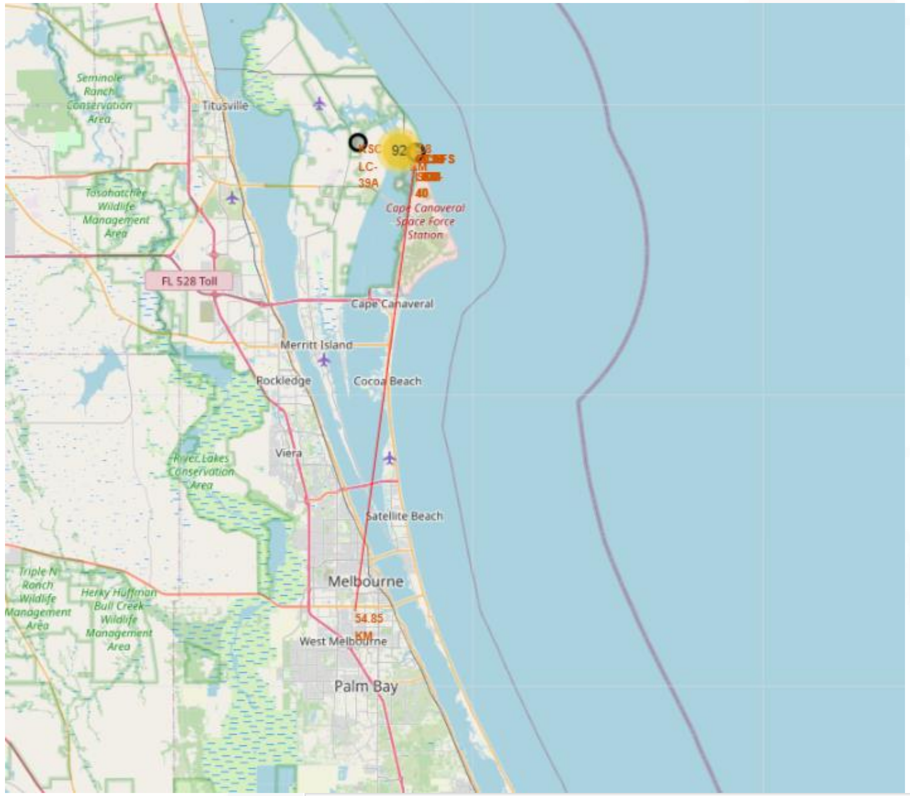



```
# find coordinate of the closet coastline
# e.g.,: Lat: 28.56367  Lon: -80.57163
# distance_coastline = calculate_distance(launch_site_lat, launch_site_lon, coastline_lat, coastline_lon)
# coordinate launch_site 7
launch_site_lat = 28.56323
launch_site_lon = -80.5768
coastline_lat = 28.5632
coastline_lon = -80.56756
distance_coastline = calculate_distance(launch_site_lat, launch_site_lon, coastline_lat, coastline_lon)
print("{:.3}".format(distance_coastline) , 'km')
```

```
0.903 km
```

RESULTS – INTERACTIVE MAP

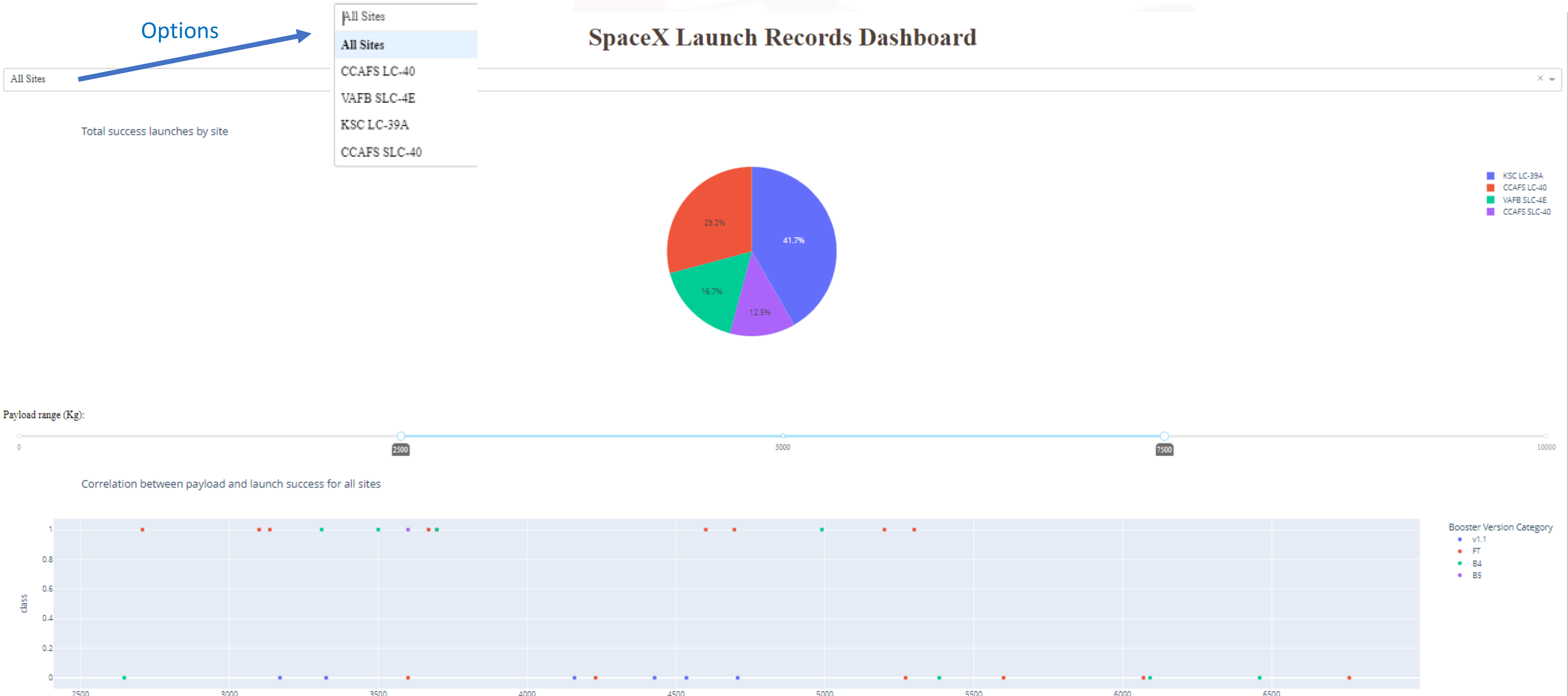
RESULTS – INTERACTIVE MAP



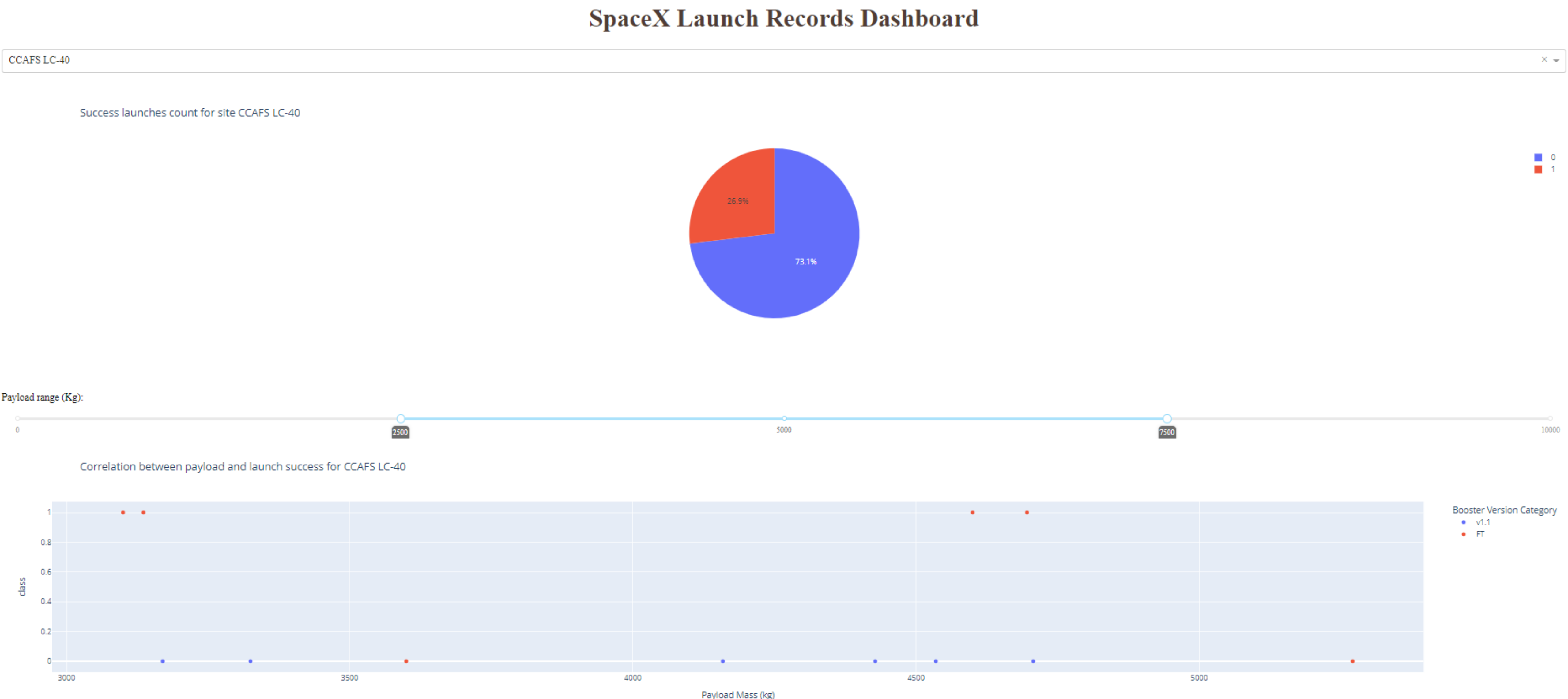

```
# Create a marker with distance to a closest city, railway, highway, etc.
# Draw a line between the marker to the launch site
coordinate_closest_city = [28.0744,-80.65063]
coordinate_closest_railway = [28.57323,-80.58521]
coordinate_closest_highway = [28.56313,-80.57075]
distance_closest_city = calculate_distance(launch_site_lat, launch_site_lon, coordinate_closest_city[0], coordinate_closest_city[1])
distance_closest_railway = calculate_distance(launch_site_lat, launch_site_lon, coordinate_closest_railway[0], coordinate_closest_railway[1])
distance_closest_highway = calculate_distance(launch_site_lat, launch_site_lon, coordinate_closest_highway[0], coordinate_closest_highway[1])
print("{:.3}".format(distance_closest_city) , 'km',"{:.3}".format(distance_closest_railway) , 'km',"{:.3}".format(distance_closest_highway) , 'km')
```

```
54.9 km 1.38 km 0.591 km
```

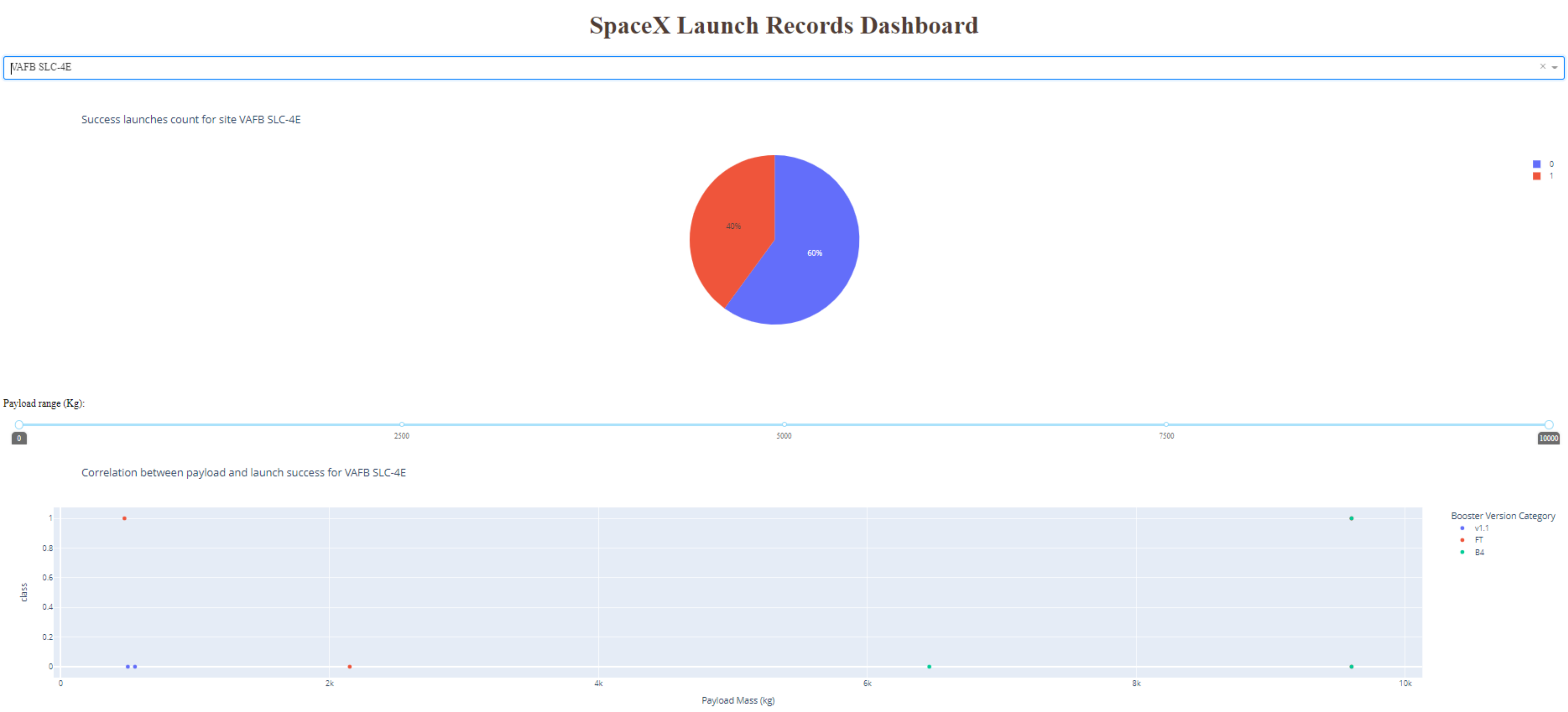
RESULTS – DASHBOARD



RESULTS – DASHBOARD



RESULTS – DASHBOARD



RESULTS – PREDICTIVE ANALYSIS

TASK 1

Create a NumPy array from the column `Class` in `data`, by applying the method `to_numpy()` the

```
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.

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

RESULTS – PREDICTIVE ANALYSIS

TASK 1

Create a NumPy array from the column `Class` in `data`, by applying the method `to_numpy()` the

```
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.

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

TASK 3

Use the function train_test_split to split the data X and Y into training and test data. Set the parameter test_size to 0.2 and r

`X_train, X_test, Y_train, Y_test`

```
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.

```
Y_test.shape
```

```
(18,)
```

RESULTS – PREDICTIVE ANALYSIS

TASK 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`.

```
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(estimator=lr, param_grid=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']})
```

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_`.

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

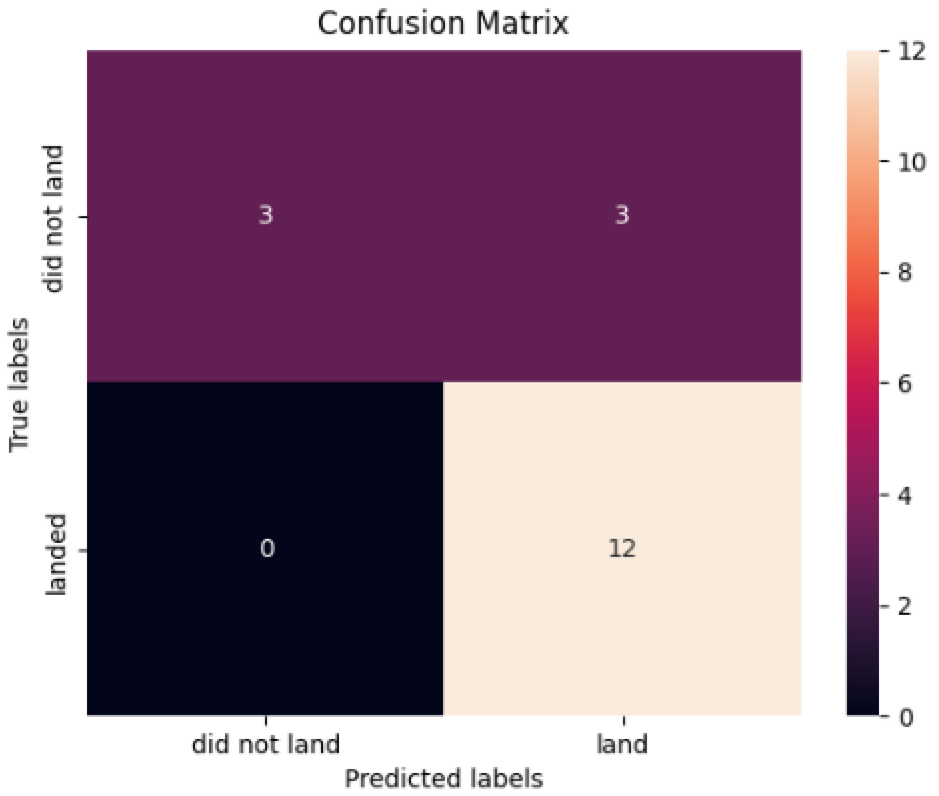
```
tuned hpyerparameters :(best parameters)  {'C': 0.01, 'penalty': 'l2', 'solver': 'lbfgs'}
accuracy : 0.8464285714285713
```

TASK 5

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

```
logscore = logreg_cv.score(X_test, Y_test)
logscore
```

```
0.8333333333333334
```



RESULTS – PREDICTIVE ANALYSIS

TASK 6

Create a support vector machine object then create a `GridSearchCV` object `svm_cv` with cv - 10. Fit the object to find the best parameters from the dictionary `parameters`.

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

```
svm_cv = GridSearchCV(estimator=svm, param_grid=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+00, 3.16227766e+01,
       1.00000000e+03]),
                         'gamma': array([1.00000000e-03, 3.16227766e-02, 1.00000000e+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_)
```

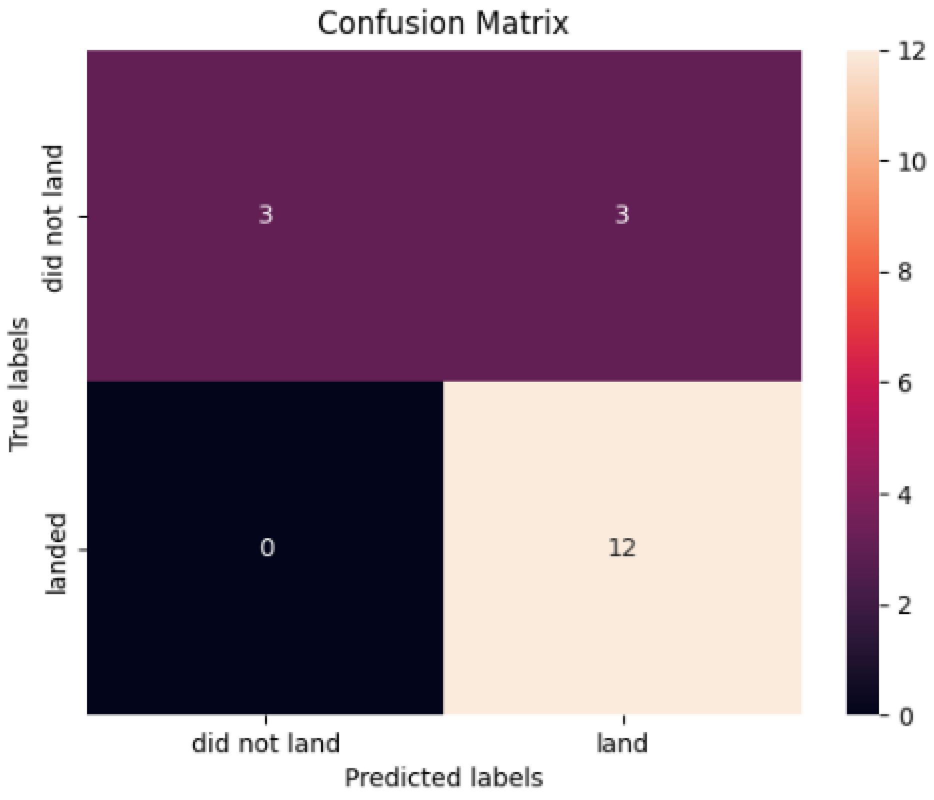
```
tuned hpyerparameters :(best parameters)  {'C': 1.0, 'gamma': 0.03162277660168379, 'kernel': 'sigmoid'}
accuracy : 0.8482142857142856
```

TASK 7

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

```
svmscore = svm_cv.score(X_test, Y_test)
svmscore
```

```
0.8333333333333334
```



RESULTS – PREDICTIVE ANALYSIS

TASK 8

Create a decision tree classifier object then create a `GridSearchCV` object `tree_cv` with cv = 10. Fit the object to find the best parameters from the dictionary `parameters`.

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

```
GridSearchCV(cv=10, estimator=DecisionTreeClassifier(),
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': [2, 4, 6, 8, 10, 12, 14, 16, 18],
                         'max_features': ['auto', 'sqrt'],
                         'min_samples_leaf': [1, 2, 4],
                         'min_samples_split': [2, 5, 10],
                         'splitter': ['best', 'random']})
```

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

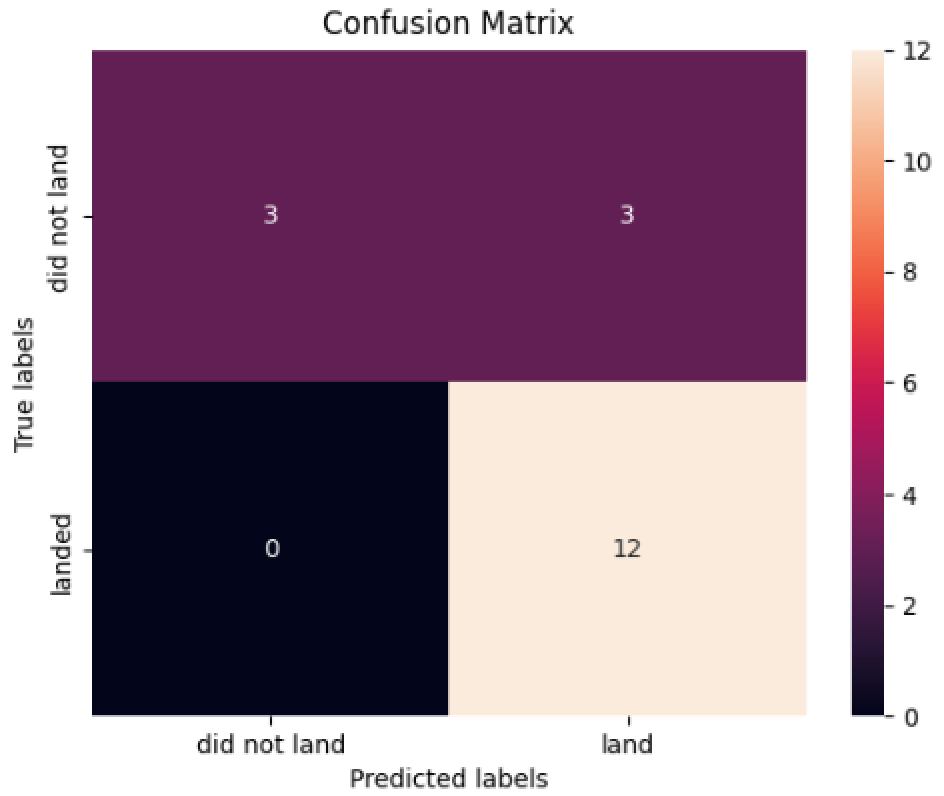
```
tuned hpyerparameters :(best parameters)  {'criterion': 'entropy', 'max_depth': 12, 'max_features': 'sqrt', 'min_samples_leaf': 4, 'min_samples_split': 2, 'splitter': 'random'}
accuracy : 0.8767857142857143
```

TASK 9

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

```
treescore = tree_cv.score(X_test, Y_test)
treescore
```

```
0.8333333333333334
```



RESULTS – PREDICTIVE ANALYSIS

TASK 10

Create a k nearest neighbors object then create a `GridSearchCV` object `knn_cv` with cv = 10. Fit the object to find the best parameters from the dictionary `parameters`.

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

```
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]})
```

```
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
```

TASK 11

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

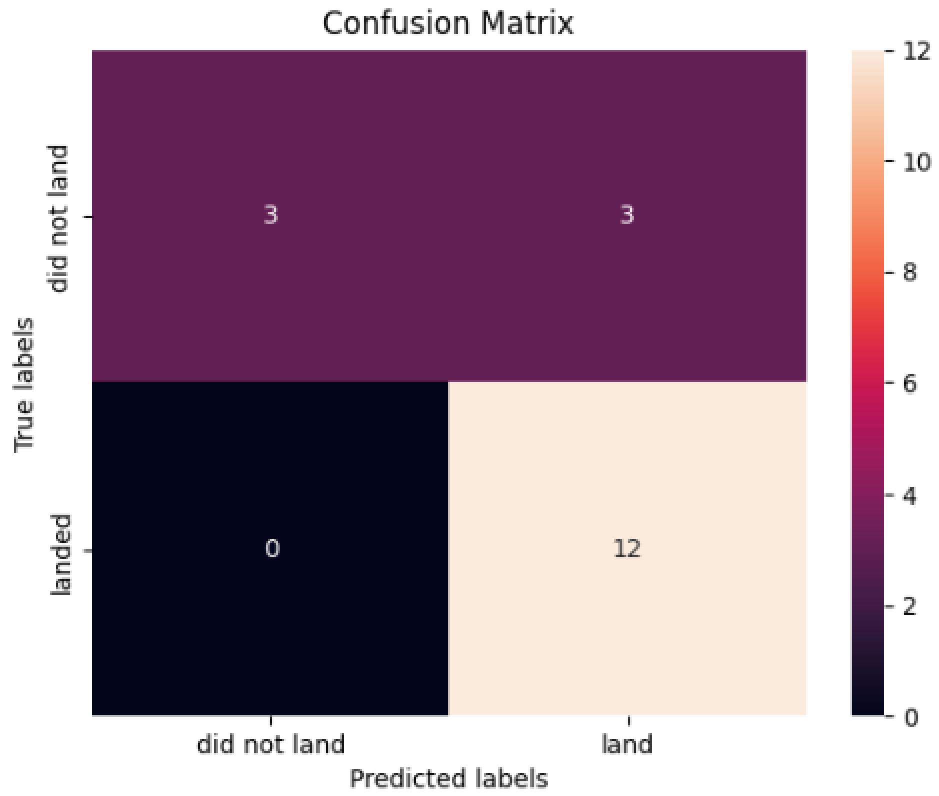
```
knnscore = knn_cv.score(X_test, Y_test)
knnscore
```

```
0.8333333333333334
```

TASK 12

Find the method performs best:

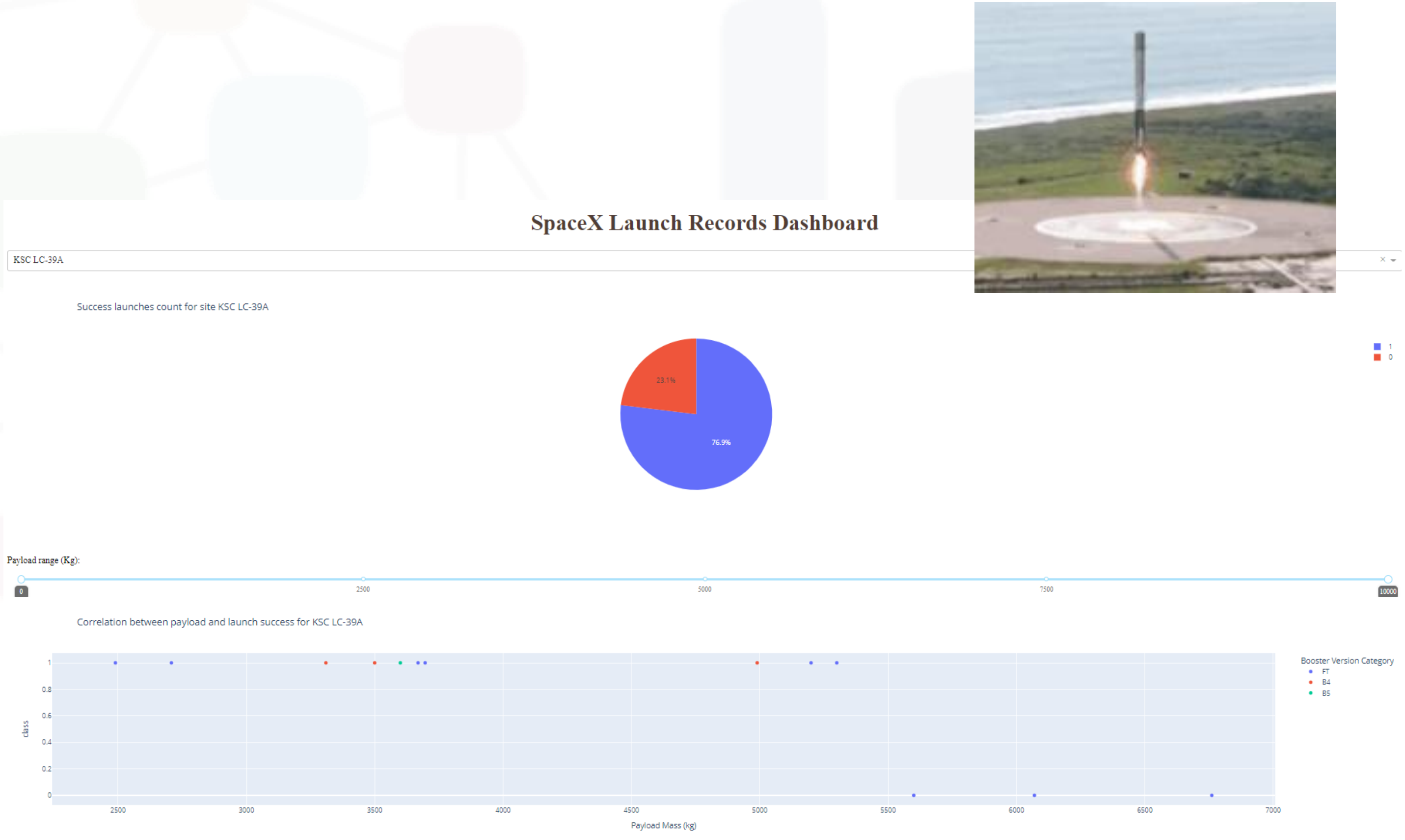
```
All the algorithms are returning the same score 0.833 and almost the same accuracy about 0.84 , except decision tree that was little greater 0.87
```



INOVATIVE INSIGTHS

Findings

- KSC LC-39A has a good rate of success 76.9% and there is no fail to payload mass lower than 5.600 kg



Implications

- Can be a good strategy choose more this launch site with payload less than 5.600 kg

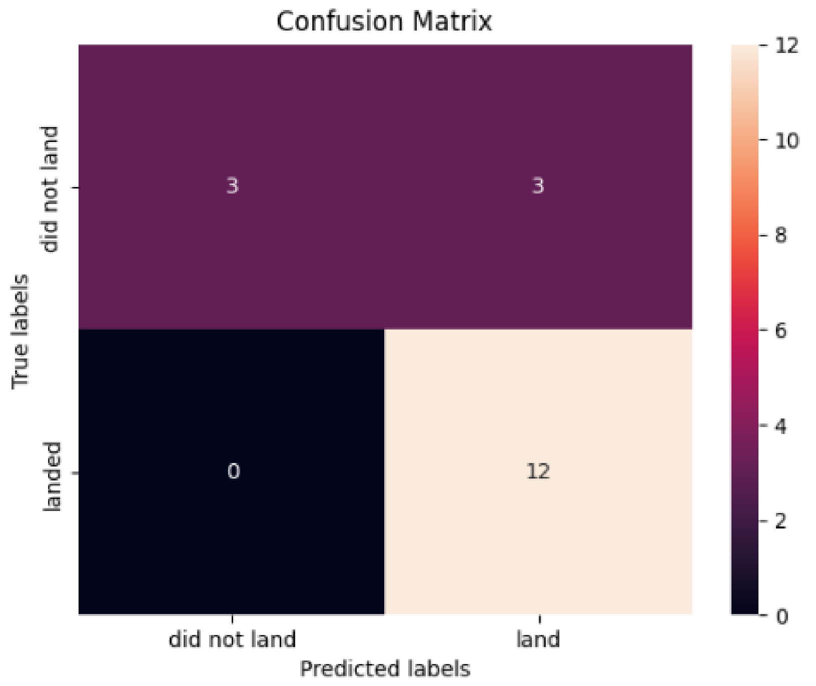
INOVATIVE INSIGTHS

Findings

- Based on confusion matrix It’s better using the model to predict successful than failed, and this is exactly what we want

Implications

- It showed that model can be used to predict launch



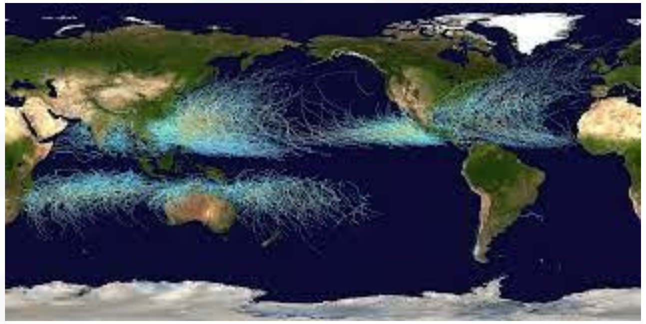
INOVATIVE INSIGTHS

Findings

- Maybe could be interesting available some meteorology data and another launch places in the world

Implications

- It can give insights if there are some condition that improve the success rate and make a launch more safety and cheaper



CONCLUSION

- Classification algorithm showed a good strategy to use to predict launch
- The algorithm used in machine learning Logistic Regression, SVM, Decision Tree Classifier and KNN had the same score 0.833 and almost the same accuracy about 0.84, except Decision Tree that was little greater 0.87
- These four algorithm can be used even decision tree had a little greater accuracy, but practically almost the same
- SpaceX can use the model to help in decision and study the best strategy to launch. Then is possible to reduce risk off failed, accidents and reduce costs