SpaceX Falcon 9 Launch Outcomes Insights

Tanya Radkey

2021-12-12

Table of Contents

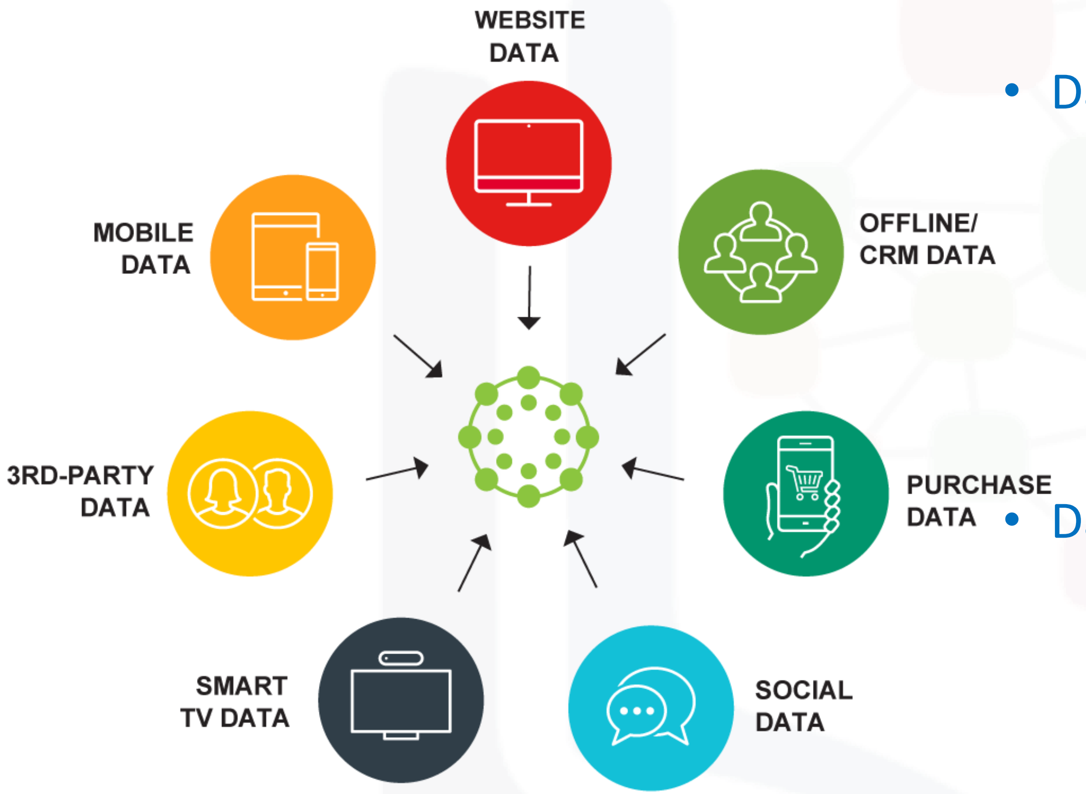
EXECUTIVE SUMMARY

- The business model for SpaceX that has allowed them to offer clients a significantly lower price point for launching payloads into space is based on reuse of the first stage rocket.
- The key point in being able to reuse the first stage for a future launch, is achieving a successful launch/landing outcome for the first stage.
- The goal of this analysis is to determine what factors influence successful launch outcomes for SpaceX, so that these factors can be used to predict future launch outcomes.
- These predictions can be used to calculate launch costs, and to establish pricing for clients, and profit margins.

INTRODUCTION

- Background
  - As identified in the assignment parameters
    - *Space X advertises Falcon 9 rocket launches on its website with a cost of 62 million dollars; other providers cost upward of 165 million dollars each, much of the savings is because Space X can reuse the first stage. Therefore if we can determine if the first stage will land, we can determine the cost of a launch. This information can be used if an alternate company wants to bid against space X for a rocket launch. In this lab, you will create a machine learning pipeline to predict if the first stage will land given the data from the preceding labs.*
- Purpose
  - The purpose of this analysis is to identify the various factors that influence successful launch outcomes for SpaceX.
  - A secondary purpose is to identify the best performing algorithm for machine learning for outcome prediction for SpaceX launch outcomes.

METHODOLOGY
Data Collection and Wrangling

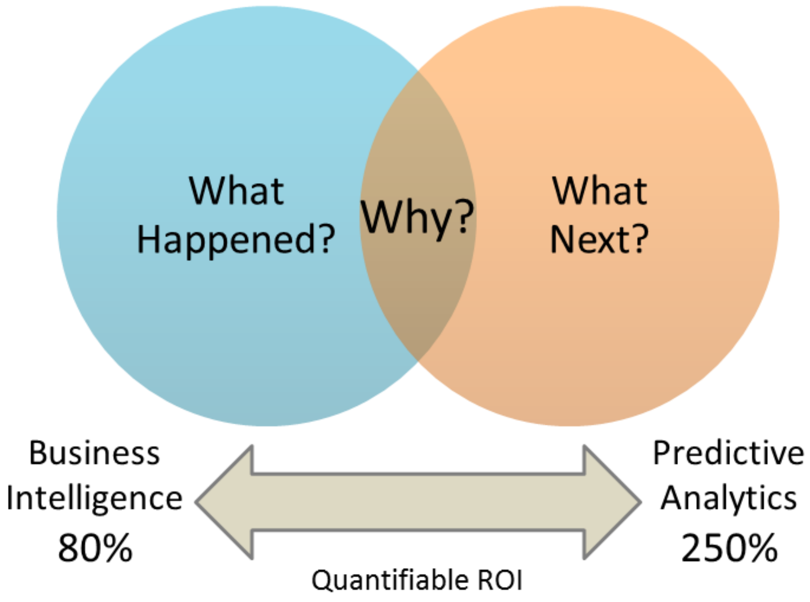


- Data Collection
  - Data was webscraped from Wikipedia page "*List of Falcon 9 and Falcon Heavy launches*" dated 2021-06-09. The webscraped data contained 121 samples, and 5 different launch sites.
  - Use of a RestFUL API, specifically SpaceX REST API to collect data (90 samples)
  - BeautifulSoup was used to parse the webscraped data and create Pandas dataframes.
- Data Preparation
  - The provided dataset "dataset_part_1.csv" containing 90 samples using 3 different launch sites was the base for the data preparation.
  - Missing values were identified by attribute
  - Attribute types were identified
  - Landing Outcomes were identified, categorized as "bad outcome" (value 0) or not, and a landing outcome label was added to the dataframe.
  - Training labels were identified

METHODOLOGY
EDA and Interactive Visual Analytics

- Data Analysis
  - Exploratory Data Analysis
  - Data Feature Engineering
  - Examination of landing success rate by flight number, launch site, orbit, and payload mass (kg) was undertaken using scatterplots.
  - Examination of success rate by orbit type using mean success rate in bar chart format.
  - SQL analysis of outcomes by type for a selected period (additional SQL analysis tasks provided in the Appendix)
  - Launch sites location analysis using an interactive Folium map (outcomes, and location and proximities of a launch site)
  - Interactive launch records analysis using Plotly Dash.

METHODOLOGY
Predictive Analysis

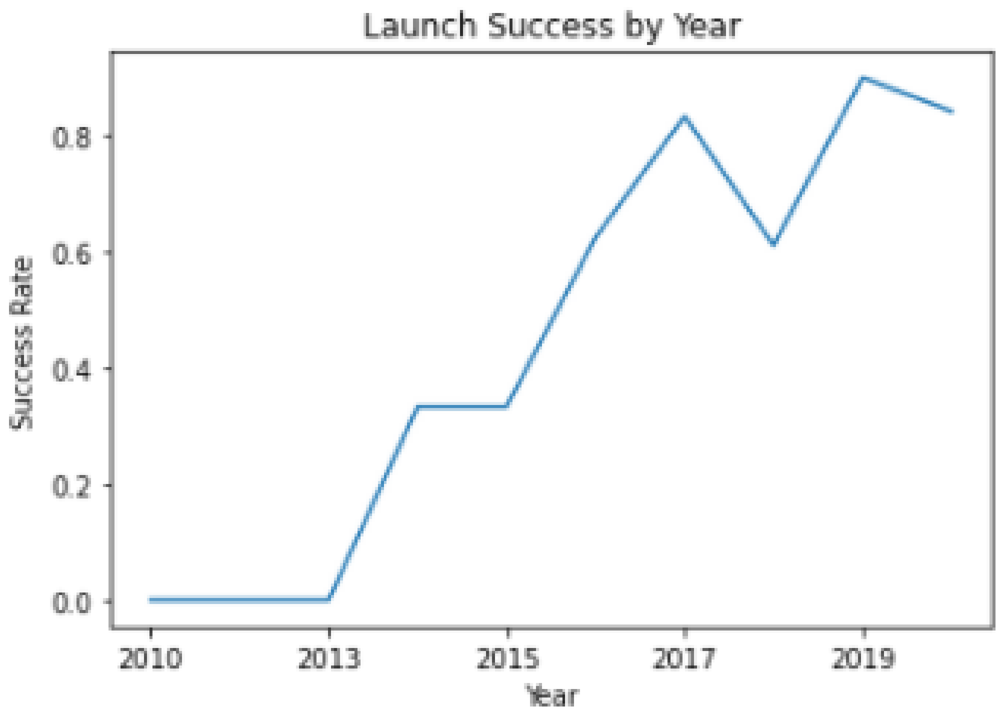


- Prediction
  - Data was converted to a NumPy array
  - Data was preprocessed using StandardScalar, fit, and transform
  - Train/Test split was created, using a 20% test size parameter
  - Logistic Regression Analysis conducted
  - Vector Machine Object Analysis was conducted
  - Decision Tree Classifier Analysis was conducted
  - K Nearest Neighbors Analysis was conducted
  - Accuracy for each analysis method on the Test split was calculated

RESULTS

The goal is to determine if the first stage will land, the factors that influence successful launch/landing, and the machine learning algorithm that will best predict successful outcomes.

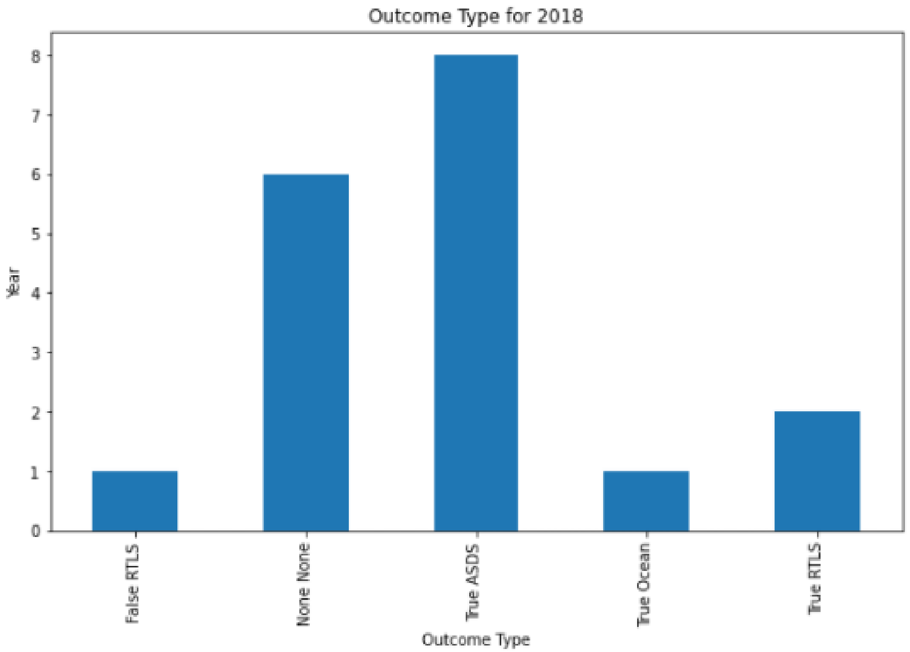
EDA with Visualization Results

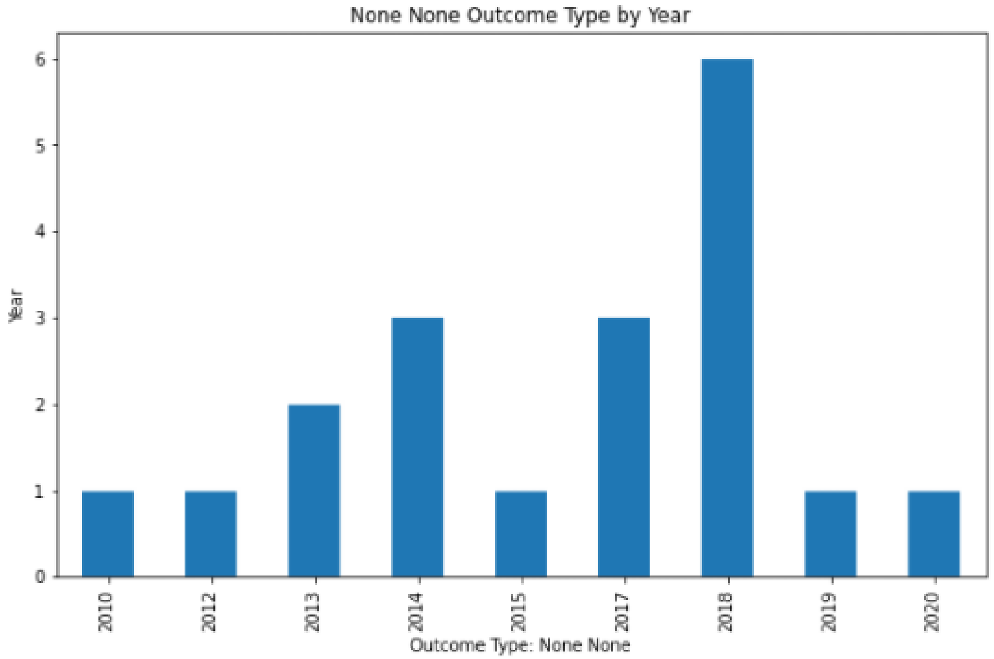


Launch Success Over Time

- The launch success rate improved with the passage of time. The data in graph form shows a trend of increased success from 2013 to 2020, with a dip in 2018.
- Following is a review of the drivers behind the increased success rate, and the dip in the success trend in 2018.

2018 Launch Success Trend Variance

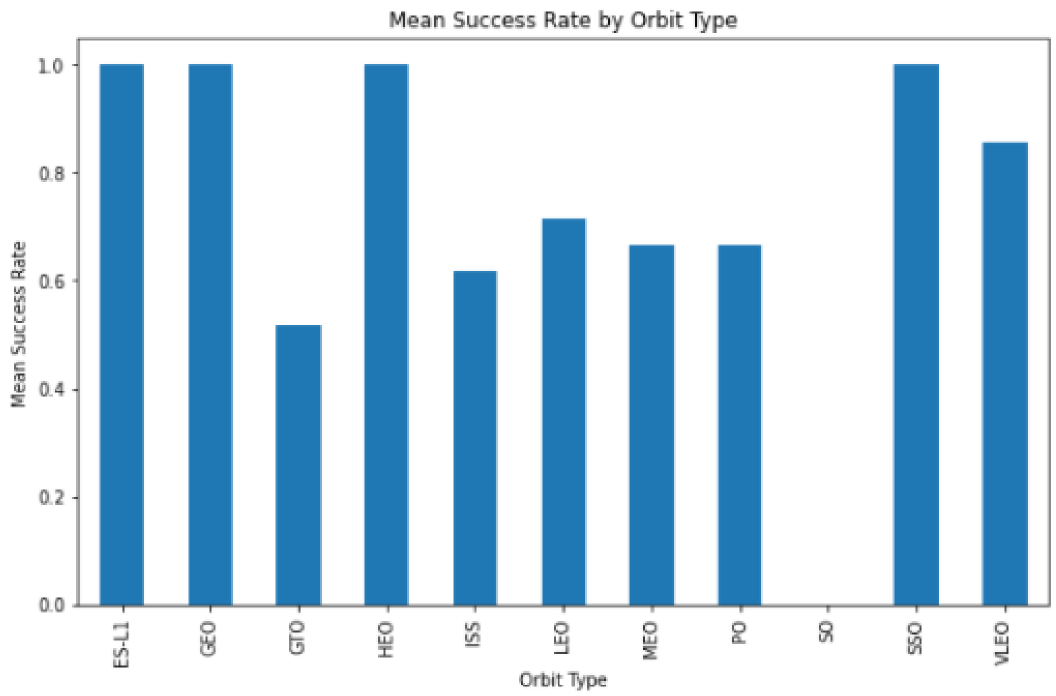


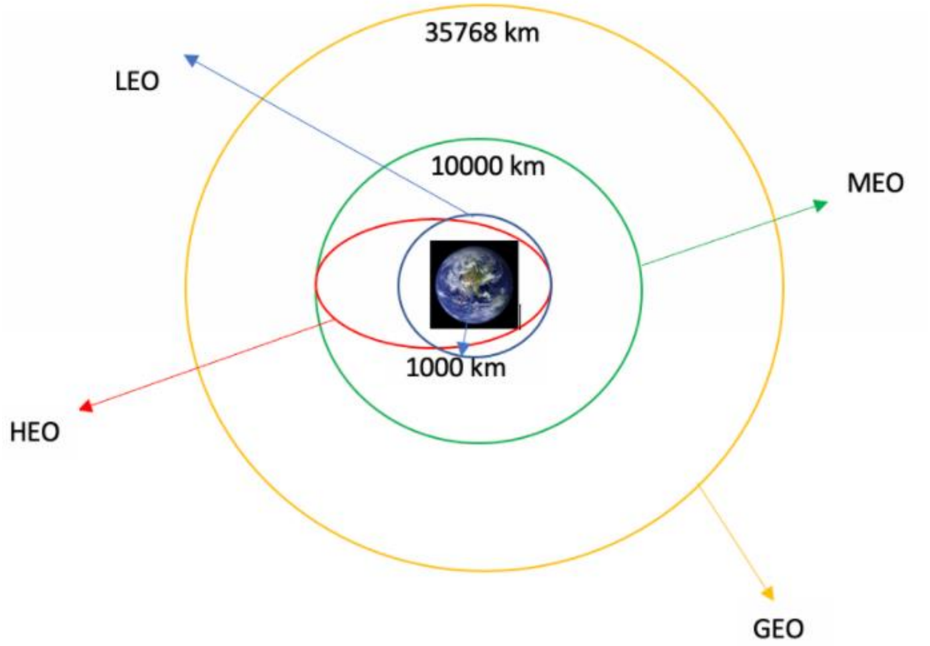


2018 Launch Success Trend Variance

- Viewing the Outcomes for 2018, the « bad outcome » category with the largest count was « None None » (i.e., where there was no attempt to land the unit).
- Viewing the « None None » instances by year, there was a higher number of « no attempts » (i.e., « None None » outcomes for 2018, which depressed the launch success rate for 2018.

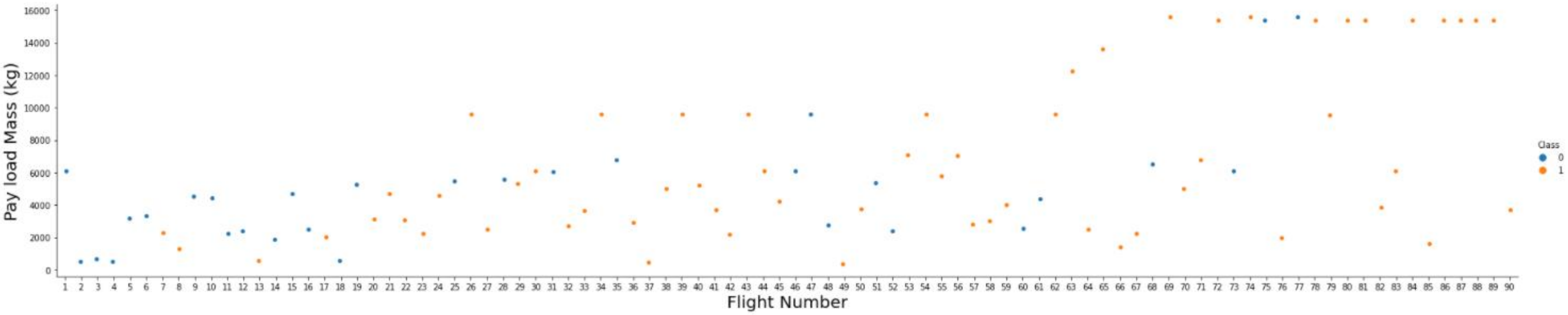
Success Factor: Orbit Type





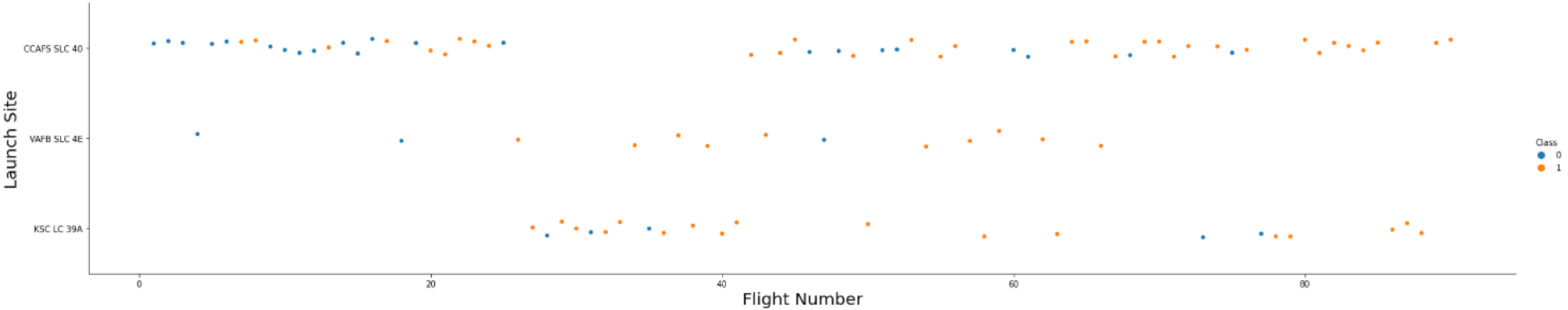
- Not taking year into account, there are orbit types that have a higher success rate than others: ES-L1, GEO, HEO, SSO, VLEO (to a lesser extent)

Success Factor: Flight Number (Time + Experience)



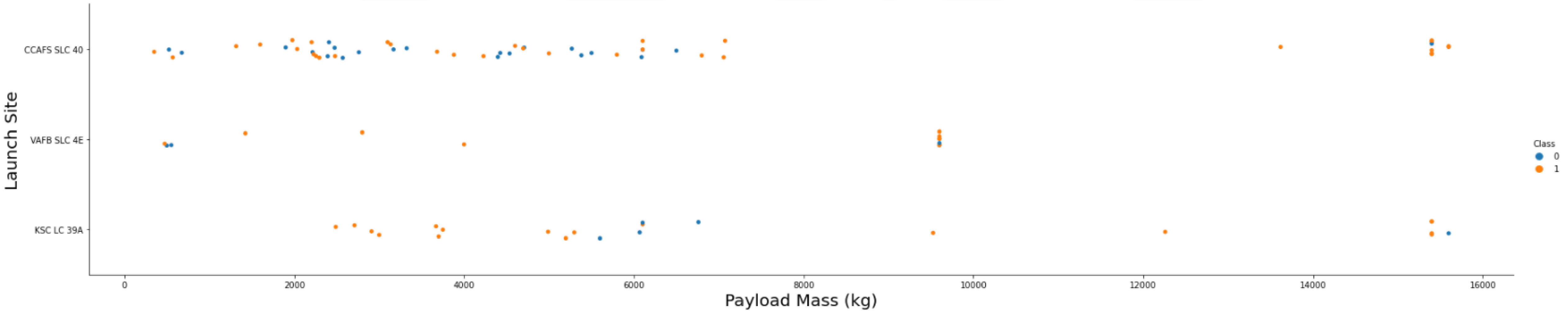
- Assumption that lessons learned from previous flights are applied to the next flight, thus evolving the mission plans and liklihood of success.
- The scatterplot shows fewer undesired outcomes as flight number (time + experience) passes.

Success Factor: Flight Number (by Launch Site)



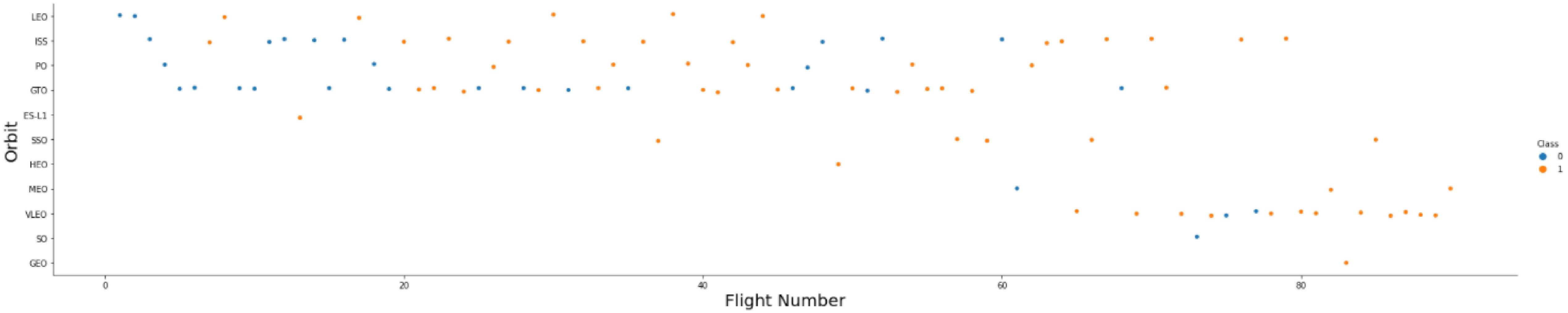
- The scatterplot shows that CCAFS SLC 40 has a lower success rate. However, the majority of the initial flights were launched from this site. Therefore, this site bears the brunt of the learning curve (i.e., initial failures).
- The scatterplot shows no usage of launch site VAFB SLC 4E after launch 66, despite the launch site having a favourable success rate. Although the scatterplot for payload mass by flight number indicates that after launch 66 more than 50% of the launches were for payloads over 10,000kg which could be a contributing factor to the decreased use of this launch site.

Success Factor: Payload and Launch Site



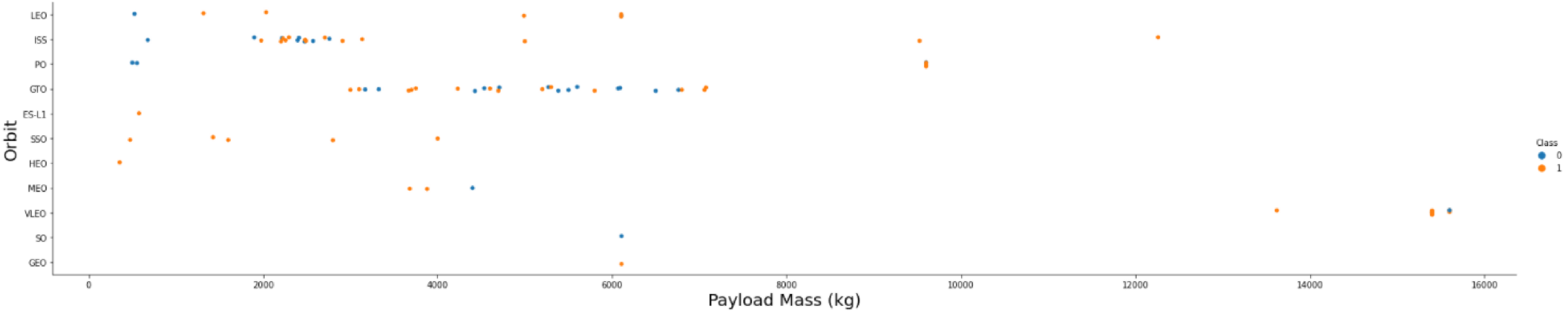
- Launch site VAFB SLC 4E does not have payload mass launches over 10,000kg.
- Successful landings are more frequent for the other two sites with higher payloads, however unsuccessful landings are not eliminated.

Success Factor: Flight Number and Orbit



- Higher rates of successful outcomes based on flight number (passage of time + experience) are identified in the scatterplot for orbit types LEO, ISS, MEO, and VLEO.
- Success rates for the orbit types GTO and PO do not appear to be related to flight number (time + experience)
- With limited launches for the orbit types SSO, HEO, and GEO (only successful outcomes) and the orbit type SO (only an unsuccessful outcome) it would be imprudent to assess any effect of flight nubmer against the success rate for these launch types.

Success Factor: Payload and Orbit



- Higher successful landing rates as payload mass increases are associated with the orbits: Polar, LEO, and ISS.
- The GTO and VLEO orbits do not appear to have payload mass as a factor in positive or negative landing success.

EDA with SQL Results

Landing Outcome	Count of Outcomes	Rank of Outcomes
No attempt	10	8
Success (drone ship)	5	6
Failure (drone ship)	5	6
Success (ground pad)	3	4
Controlled (ocean)	3	4
Uncontrolled (ocean)	2	2
Failure (parachute)	2	2
Precluded (drone ship)	1	1

- Note that this analysis of outcomes is for the date range 2010-06-04 to 2017-03-20 only.
- Given the span of time (almost 7 years), it is interesting to note that "No attempt" has a total outcome count of 10 for those years, and the earlier identification of an anomalous result for the success rate in 2018 had a count of 6 "No attempt" outcomes.
- Of note, the dataset for the SQL analysis included an additional 11 lines of data (101 records versus 90 lines in the datasets used for all other analyses) and 4 launch sites.

See the Appendix for the full slate of SQL tasks for the analysis.

Interactive Map – Launch Site Locations

The SpaceX launch sites are on the east (state of Florida, count of 3) and west (state of California, count of 1) coasts of the United States of America (US).

The locations are also in the southern portion of the US, and given the geopolitical borders, are located in proximity to the equator (given the geopolitical borders for each coast).

This analysis used the provided dataset "*spacex_launch_geo.csv*" which identified 4 launch sites.

Interactive Map – California VAFB SLC-4E

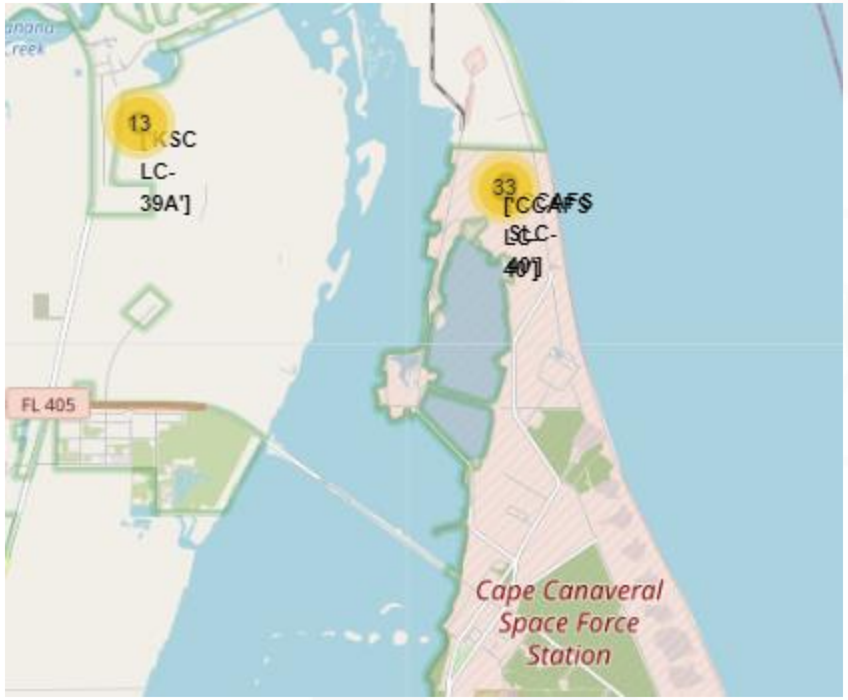
The SpaceX launch site (Vandenberg Space [Air] Force Base) on the west coast (state of California) of the United States of America (US).

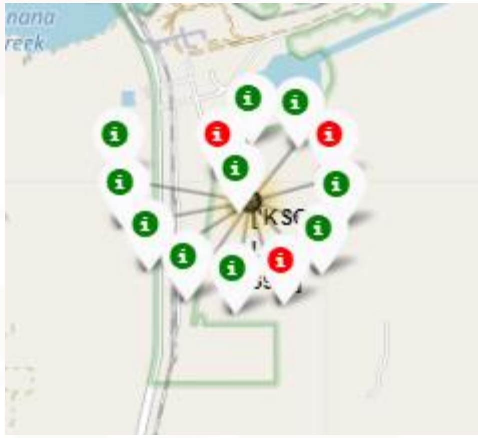
All the launch records for this site have the exact same coordinates. Therefore, the use of Marker clusters has been used to simplify the map presentation of the results.

Expanding the detail behind the marker cluser enables the pop-up markers that provide the detail on the launch site success rate.

Successful launches (class=1), are displayed with a green marker, and unsuccessful launches (class=0), the display uses a red marker.

Interactive Map – Florida KSC-LC 39A





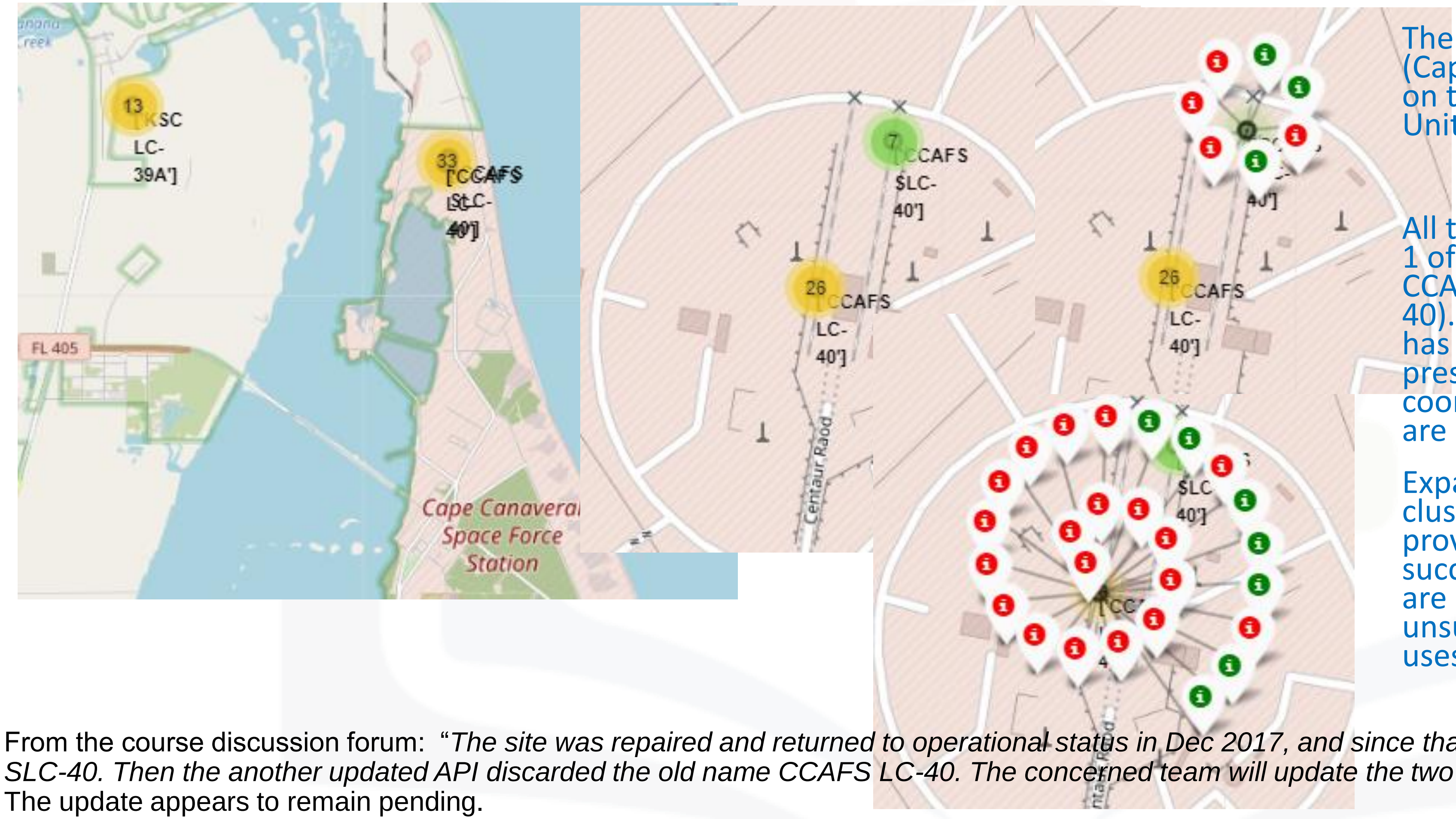
The SpaceX launch site KSC-LC 39A (Kennedy Space Centre) on the east coast (state of Florida) of the United States of America (US).

All the launch records for this site have the exact same coordinates. Therefore, the use of Marker clusters has been used to simplify the map presentation of the results.

Expanding the detail behind the marker cluser enables the pop-up markers that provide the detail on the launch site success rate.

Successful launches (class=1), are displayed with a green marker, and unsuccessful launches (class=0), the display uses a red marker.

Interactive Map – Florida CCAFS (S)LC-40[1]



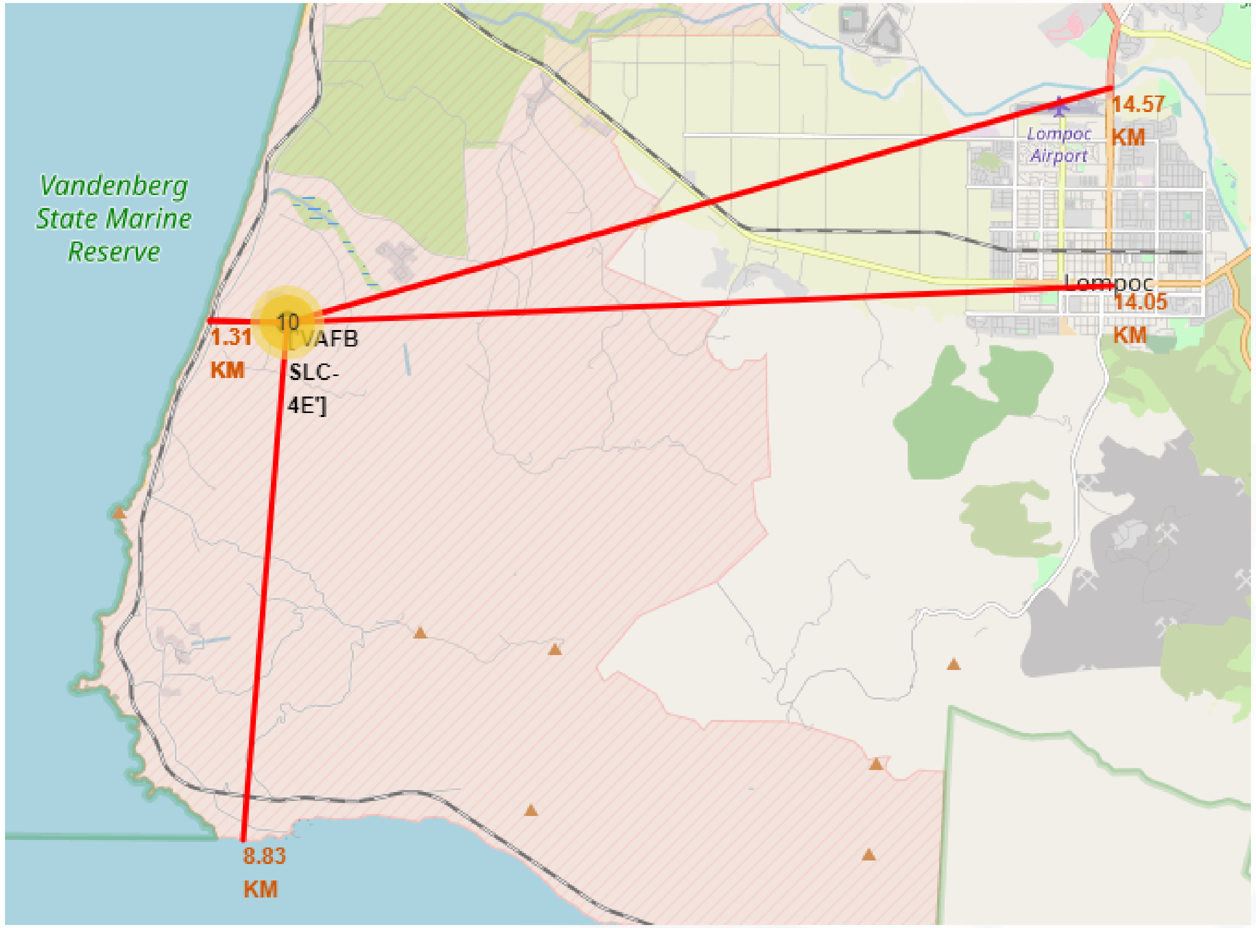
The SpaceX launch site CCAFS (S)LC-40 (Cape Canaveral Space [Air] Force Station) on the east coast (state of Florida) of the United States of America (US).

All the launch records for this site apply to 1 of 2 sets of coordinates ("old name" CCAFS LC-40 and "new name" CCAFS SLC-40). Therefore, the use of Marker clusters has been used to simplify the map presentation of the results, and both coordinate sites for this single launch site are displayed.

Expanding the detail behind the marker cluser enables the pop-up markers that provide the detail on the launch site success rate. Successful launches (class=1), are displayed with a green marker, and unsuccessful launches (class=0), the display uses a red marker.

1. From the course discussion forum: "*The site was repaired and returned to operational status in Dec 2017, and since that time the old API starts to use CCAFS SLC-40. Then the another updated API discarded the old name CCAFS LC-40. The concerned team will update the two visual analytic lab to use latest dataset.*" The update appears to remain pending.

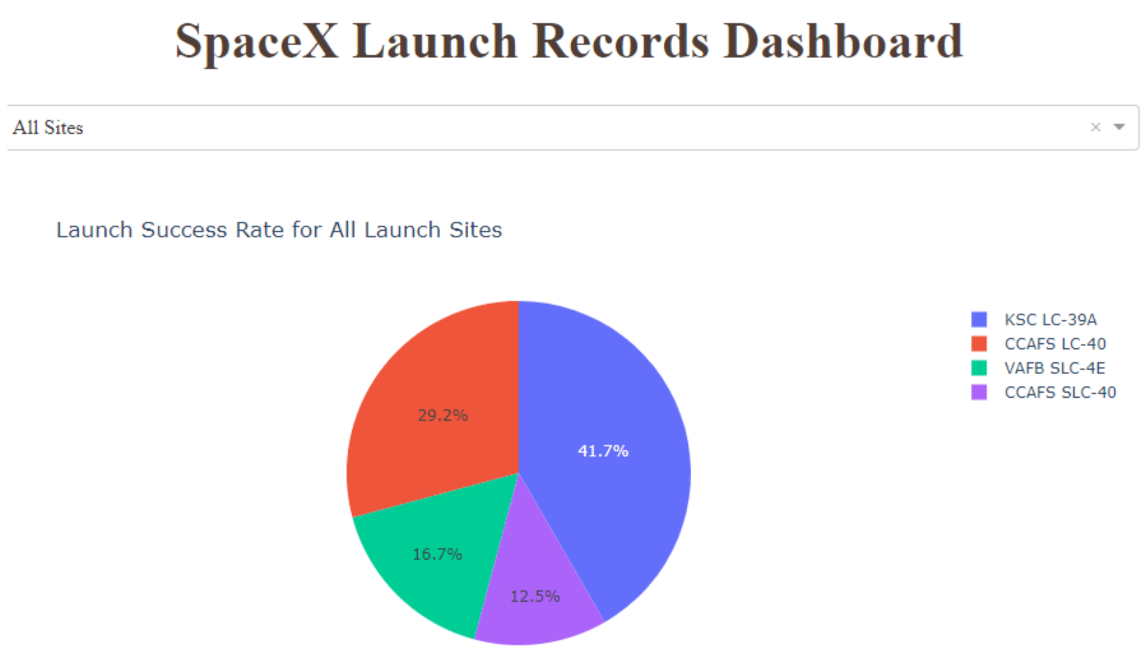
Interactive Map – Landmark Distances

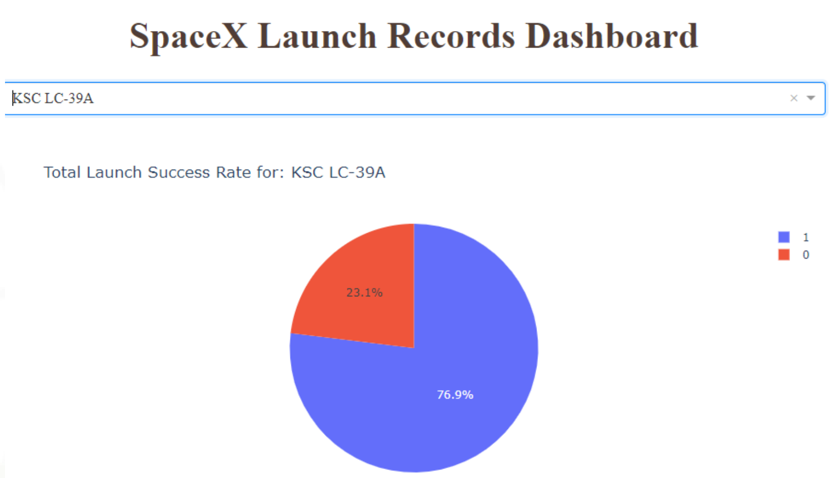


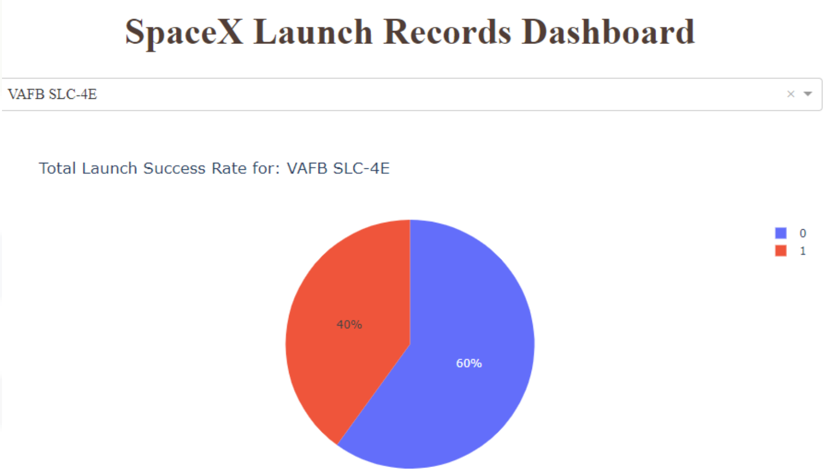
Exploration and analysis of the proximities of launch site: VAFB SLC-4E

- Railway proximity: 1.31km
- Highway proximity: within 15 km
- Coastline proximity: non marine reserve coastline less than 9km; however, coastline within the Vandenberg State Marine Reserve is much closer, similar distance as railway proximity.
- Proximity to city: closest urban area just over 14km

Dashboard Results: Launch Success Rate



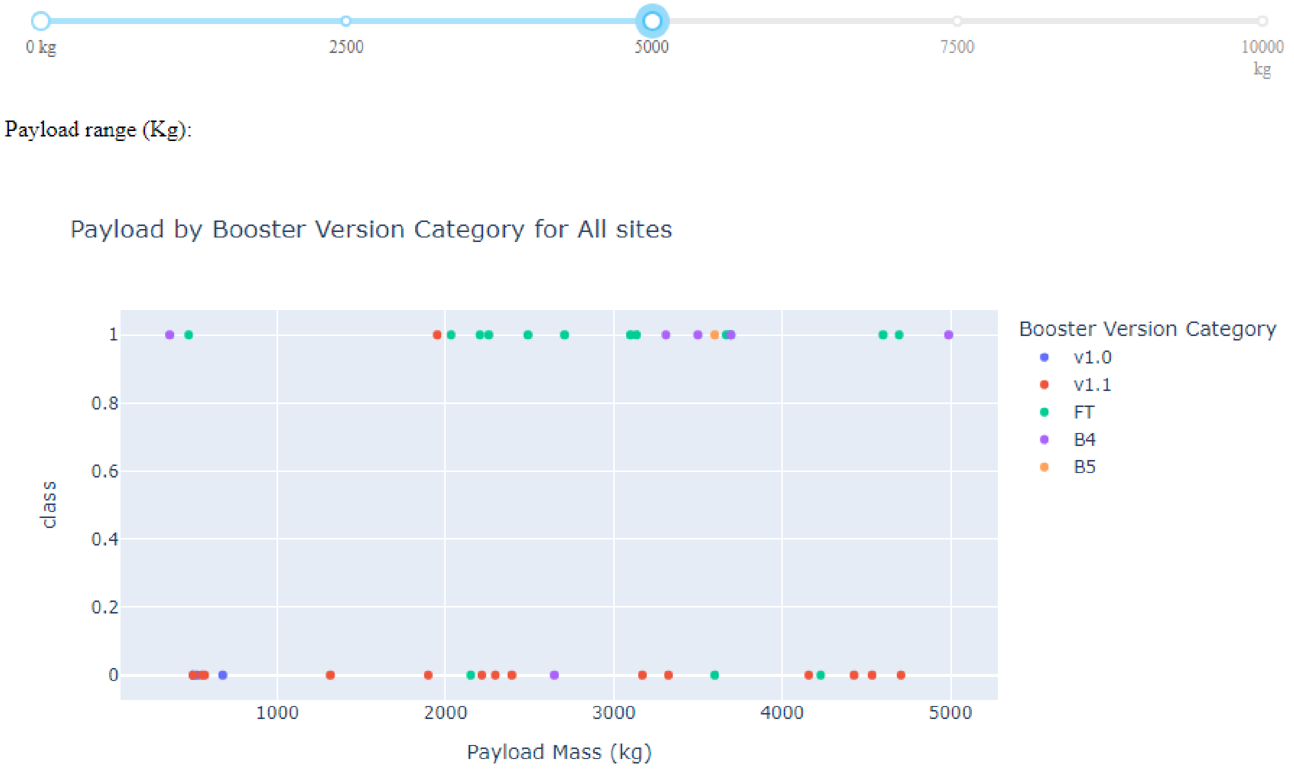


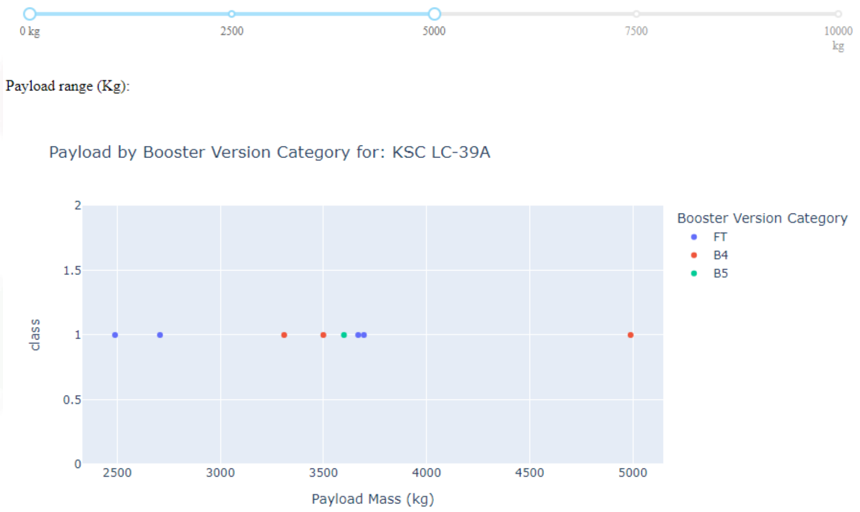


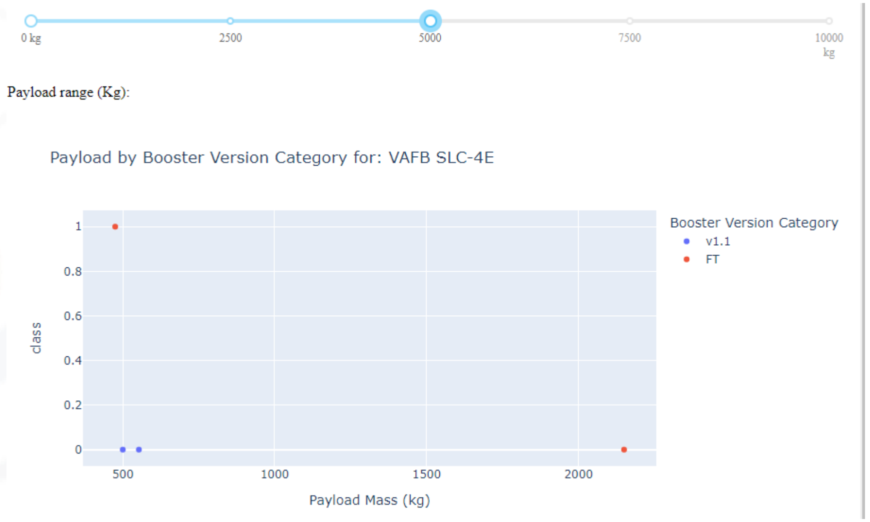
Using the interactive dashboard analysis, the pie chart indicated that the Kennedy Space Centre site had the highest launch success rate. The successful versus unsuccessful launch outcomes for the Kennedy Space Centre were selected using the drop-down menu. For comparison, the successful versus unsuccessful launch outcomes for the Vanderberg Space [Air] Force Base site were also selected for display.

This interactive dashboard analysis used a different provided dataset "*spacex_launch_dash.csv*"

Dashboard Results: Payload 0-5000kg

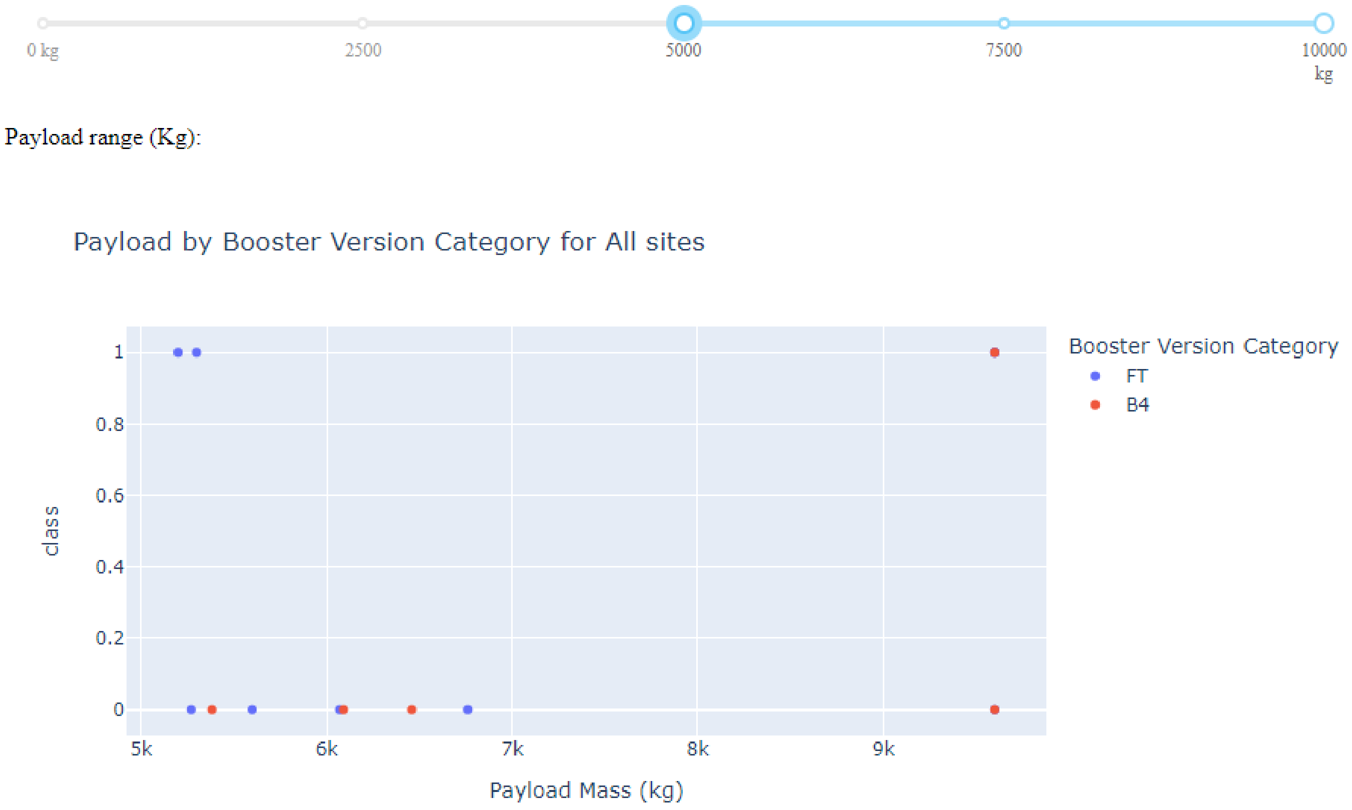


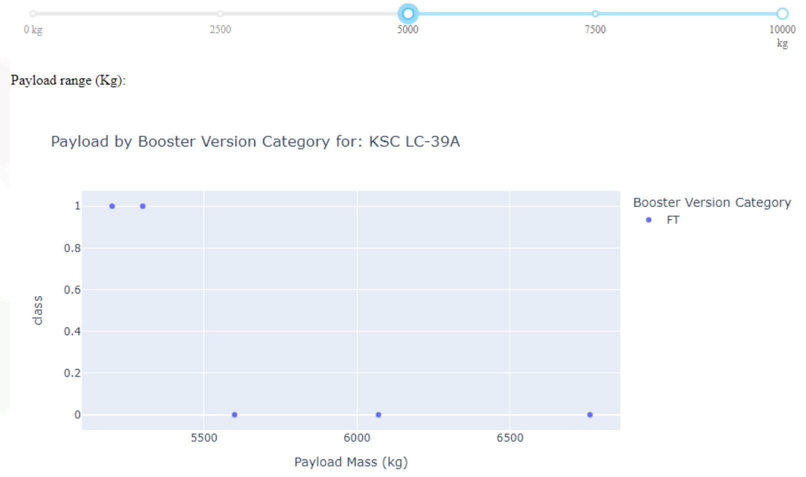




Using the Payload range slider aspect of the dashboard, for all sites the success rate for 0-5000kg payloads is similar to the unsuccessful outcomes, for the Kennedy Space Centre has 100% success for this payload range. In comparison, the Vanderberg Space [Âir] Force Base has a 25% success rate in this payload range.

Dashboard Results





Adjusting the payload mass slider to 5000-10,000kg, the success rate for all sites is significantly lower, with breakouts for the same two sites shown.

Predictive Analysis Results

	Accuracy	Jaccard Index	F1-Score
Logistic Regression	0.846428571429	0.800000000000	0.814814814815
Support Vector Machine Method	**0.848214285714**	0.800000000000	0.814814814815
Decision Tree Model (Test Data)	0.833333333333	0.800000000000	0.814814814815
K Nearest Neighbor Method	0.844400000000	**0.923076923077**	**0.943030303030**

Accuracy identifies the best performing algorithm as Support Vector Machine Method (0.0018 higher than Logistic Regression).

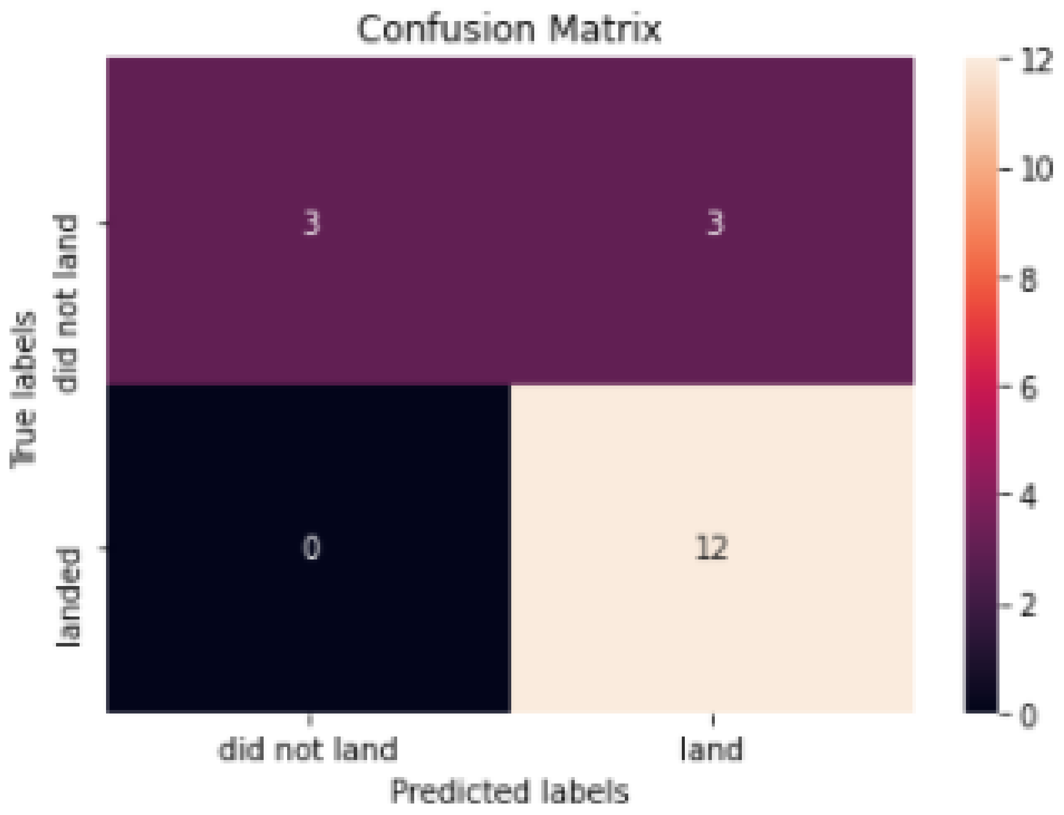
```
tuned hpyerparameters :(best parameters)  {'C': 1.0, 'gamma': 0.03162277660168379, 'kernel': 'sigmoid'}
Accuracy score for the Support Vector Machine Method : 0.8482142857142856
```

Jaccard Index and F1-Score both identify the K Nearest Neighbor algorithm as the best performing method.

```
tuned hpyerparameters :(best parameters)  {'algorithm': 'auto', 'n_neighbors': 5, 'p': 1}
Accuracy for K Nearest Neighbor Method with test data is : {:.2f}%: 0.8444444444444444
```

Predictive Analysis Results

```
SVM Jaccard index:  0.8
SVM F1-score:  0.8148148148148149
```



The Confusion Matrix for the Support Vector Machine Method identifies that the major problem is false positives.

DISCUSSION

- The purpose of this analysis is to identify the various factors that influence successful launch outcomes for SpaceX.
- Lessons learned (time + experience) is a significant factor.
- Payload, orbit, and launch site are interrelated factors.

CONCLUSION

- The best machine learning model for this dataset was the Support Vector Machine model.
- The overarching factor for successful outcomes for launches is related to time and experience (learning curve)
- Success rate for higher weighted payloads is affected by orbit while the success rate for low weighted payloads is affected by launch site.
- Orbit types that have a higher success rate than others are: ES-L1, GEO, HEO, SSO, VLEO (to a lesser extent)
- The instances of “no attempt” to land the stage 1 rocket depress the success rate without providing any useful data on the undesired outcome, as the choice was made to essentially sacrifice the stage 1 rocket for that flight.

APPENDIX

Number of Launches by Site

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

Number of Launches by Orbit

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

Number of Launches by Mission 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: Outcome, dtype: int64
```

APPENDIX EDA using SQL Task 1

Task 1

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

In [11]:
```
%sql select distinct LAUNCH_SITE from SPACEXTBL
```

```
 * ibm_db_sa://wny84718:***@1bbf73c5-d84a-4bb0-85b9-ab1a4348f4a4.c3n41cmd0nqnrk39u98g.databases.appdomain.cloud:32286/bludb
Done.
```

Out[11]:

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

APPENDIX EDA using SQL Task 2

Task 2

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

In [17]:
```
%sql select * from SPACEXTBL where LAUNCH_SITE like 'CCA%' FETCH FIRST 5 ROWS ONLY
```

```
 * ibm_db_sa://wny84718:***@1bbf73c5-d84a-4bb0-85b9-ab1a4348f4a4.c3n41cmd0nqnrk39u98g.databases.appdomain.cloud:32286/bludb
Done.
```

Out[17]:

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	07:44:00	F9 v1.0 B0005	CCAFS LC-40	Dragon demo flight C2	525	LEO (ISS)	NASA (COTS)	Success	No attempt
2012-10-08	00:35:00	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

APPENDIX EDA using SQL Task 3

Task 3

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

In [26]:
```
%sql select customer, sum(payload_mass__kg_) as "total payload mass (kg)" from SPACEXTBL where customer='NASA (CRS)' group by cu
stomer
```

```
 * ibm_db_sa://wny84718:***@1bbf73c5-d84a-4bb0-85b9-ab1a4348f4a4.c3n41cmd0nqnrk39u98g.databases.appdomain.cloud:32286/bludb
Done.
```

Out[26]:

customer	total payload mass (kg)
NASA (CRS)	45596

APPENDIX EDA using SQL Task 4

Task 4

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

```
In [28]: %sql select booster_version, AVG(payload_mass__kg_) as "Average Payload Mass (kg)" from SPACEXTBL where booster_version='F9 v1.
1' group by booster_version
```

```
 * ibm_db_sa://wny84718:***@1bbf73c5-d84a-4bb0-85b9-ab1a4348f4a4.c3n41cmd0nqnrk39u98g.databases.appdomain.cloud:32286/bludb
Done.
```

Out[28]:

booster_version	Average Payload Mass (kg)
F9 v1.1	2928

APPENDIX EDA using SQL Task 5

Task 5

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

Hint:Use min function

In [38]:
```
%sql select min(date) as "Earliest Date", landing__outcome from SPACEXTBL where landing__outcome='Success (ground pad)' group by landing__outcome
```

```
 * ibm_db_sa://wny84718:***@1bbf73c5-d84a-4bb0-85b9-ab1a4348f4a4.c3n41cmd0nqnrk39u98g.databases.appdomain.cloud:32286/bludb
Done.
```

Out[38]:

Earliest Date	landing__outcome
2015-12-22	Success (ground pad)

APPENDIX EDA using SQL Task 6

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 [42]:
```
%sql select booster_version, payload_mass__kg_ from SPACEXTBL where landing__outcome='Success (drone ship)' and payload_mass__kg
_>4000 and payload_mass__kg_<6000
```

```
 * ibm_db_sa://wny84718:***@1bbf73c5-d84a-4bb0-85b9-ab1a4348f4a4.c3n41cmd0nqnrk39u98g.databases.appdomain.cloud:32286/bludb
Done.
```

Out[42]:

booster_version	payload_mass__kg_
F9 FT B1022	4696
F9 FT B1026	4600
F9 FT B1021.2	5300
F9 FT B1031.2	5200

APPENDIX EDA using SQL Task 7

Task 7

List the total number of successful and failure mission outcomes

In [48]:
```
%sql select mission_outcome, count(mission_outcome) as "Count of outcomes" from SPACEXTBL group by mission_outcome
```

```
 * ibm_db_sa://wny84718:***@1bbf73c5-d84a-4bb0-85b9-ab1a4348f4a4.c3n41cmd0nqnrk39u98g.databases.appdomain.cloud:32286/bludb
Done.
```

Out[48]:

mission_outcome	Count of outcomes
Failure (in flight)	1
Success	99
Success (payload status unclear)	1

APPENDIX EDA using SQL Task 8

Task 8

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

In [51]:
```
%sql select distinct booster_version, payload_mass__kg_ from SPACEXTBL where payload_mass__kg_ = (select max(payload_mass__kg_)
from SPACEXTBL)
```

```
 * ibm_db_sa://wny84718:***@1bbf73c5-d84a-4bb0-85b9-ab1a4348f4a4.c3n41cmd0nqnrk39u98g.databases.appdomain.cloud:32286/bludb
Done.
```

Out[51]:

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

APPENDIX EDA using SQL Task 9

Task 9

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

In [56]:
```
%sql select landing__outcome, booster_version, launch_site, date from SPACEXTBL where landing__outcome = ('Failure (drone shi
p)') and year(date)=2015
```

```
 * ibm_db_sa://wny84718:***@1bbf73c5-d84a-4bb0-85b9-ab1a4348f4a4.c3n41cmd0nqnrk39u98g.databases.appdomain.cloud:32286/bludb
Done.
```

Out[56]:

landing__outcome	booster_version	launch_site	DATE
Failure (drone ship)	F9 v1.1 B1012	CCAFS LC-40	2015-01-10
Failure (drone ship)	F9 v1.1 B1015	CCAFS LC-40	2015-04-14

APPENDIX EDA using SQL Task 10

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

In [60]:
```
%sql select landing__outcome, count(landing__outcome), RANK () OVER (ORDER BY count(landing__outcome)) rank_no from SPACEXTBL wh
ere date between '2010-06-04' and '2017-03-20' group by landing__outcome order by count(landing__outcome) DESC
```

```
 * ibm_db_sa://wny84718:***@1bbf73c5-d84a-4bb0-85b9-ab1a4348f4a4.c3n41cmd0nqnrk39u98g.databases.appdomain.cloud:32286/bludb
Done.
```

Out[60]:

landing__outcome	2	rank_no
No attempt	10	8
Success (drone ship)	5	6
Failure (drone ship)	5	6
Success (ground pad)	3	4
Controlled (ocean)	3	4
Uncontrolled (ocean)	2	2
Failure (parachute)	2	2
Precluded (drone ship)	1	1

APPENDIX Folium – distance calculation

In [81]:
```
from math import sin, cos, sqrt, atan2, radians

def calculate_distance(lat1, lon1, lat2, lon2):
    # approximate radius of earth in km
    R = 6373.0

    lat1 = radians(lat1)
    lon1 = radians(lon1)
    lat2 = radians(lat2)
    lon2 = radians(lon2)

    dlon = lon2 - lon1
    dlat = lat2 - lat1

    a = sin(dlat / 2)**2 + cos(lat1) * cos(lat2) * sin(dlon / 2)**2
    c = 2 * atan2(sqrt(a), sqrt(1 - a))

    distance = R * c
    return distance
```

TODO: Mark down a point on the closest coastline using MousePosition and calculate the distance between the coastline point and the launch site.

In [82]:
```
# find coordinate of the closet coastline
# e.g.,: Lat: 28.56367  Lon: -80.57163
coastline_lat = 34.55378
coastline_lon = -120.61899
launch_site_lat = 34.632834
launch_site_lon = -120.610746
#distance_coastline = calculate_distance(launch_site_lat, launch_site_lon, coastline_lat, coastline_lon)
```

APPENDIX Machine Learning: Task 1

TASK 1

Create a NumPy array from the column `Class` in `data`, by applying the method `to_numpy()` then assign it to the variable `Y`,make sure the output is a Pandas series (only one bracket df['name of column']).

In [6]:
```
Y=data['Class'].to_numpy()
Y
```

Out[6]:
```
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])
```

APPENDIX Machine Learning: Task 2

TASK 2

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

In [7]:
```
# students get this
X = preprocessing.StandardScaler().fit(X).transform(X)

X[0:5]
```

Out[7]:
```
array([[-1.71291154e+00, -1.94814463e-16, -6.53912840e-01,
        -1.57589457e+00, -9.73440458e-01, -1.05999788e-01,
        -1.05999788e-01, -6.54653671e-01, -1.05999788e-01,
        -5.51677284e-01,  3.44342023e+00, -1.85695338e-01,
        -3.33333333e-01, -1.05999788e-01, -2.42535625e-01,
        -4.29197538e-01,  7.97724035e-01, -5.68796459e-01,
        -4.10890702e-01, -4.10890702e-01, -1.50755672e-01,
        -7.97724035e-01, -1.50755672e-01, -3.92232270e-01,
         9.43398113e+00, -1.05999788e-01, -1.05999788e-01,
        -1.05999788e-01, -1.05999788e-01, -1.05999788e-01,
        -1.05999788e-01, -1.05999788e-01, -1.05999788e-01,
        -1.05999788e-01, -1.05999788e-01, -1.05999788e-01,
        -1.05999788e-01, -1.05999788e-01, -1.05999788e-01,
        -1.05999788e-01, -1.05999788e-01, -1.05999788e-01,
        -1.05999788e-01, -1.50755672e-01, -1.05999788e-01,
        -1.05999788e-01, -1.05999788e-01, -1.05999788e-01,
        -1.05999788e-01, -1.50755672e-01, -1.05999788e-01,
        -1.50755672e-01, -1.50755672e-01, -1.05999788e-01,
        -1.50755672e-01, -1.50755672e-01, -1.05999788e-01,
```

APPENDIX Machine Learning: Task 3

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 random_state to 2. The training data and test data should be assigned to the following labels.

`X_train, X_test, Y_train, Y_test`

In [8]:
```
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.

In [9]:
```
Y_test.shape
```

Out[9]:
```
(18,)
```

APPENDIX Machine Learning: Task 4

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`.

```
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
         logreg=LogisticRegression(solver='lbfgs').fit(X_train,Y_train)
         logreg_cv=GridSearchCV(logreg,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 of Logistic Regression Method method :",logreg_cv.best_score_)

         tuned hpyerparameters :(best parameters)  {'C': 0.01, 'penalty': 'l2', 'solver': 'lbfgs'}
         Accuracy of Logistic Regression Method method : 0.8464285714285713
```

```
In [13]: from sklearn.metrics import jaccard_score
         from sklearn.metrics import f1_score
```

APPENDIX Machine Learning: Task 5

TASK 5

In [14]:
```
LR=LogisticRegression(C=0.01, penalty='l2', solver='lbfgs').fit(X_train,Y_train)
yhat = LR.predict(X_test)
yhat
```

Out[14]:
```
array([1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1])
```

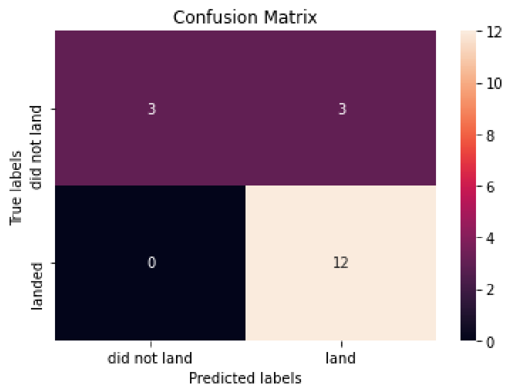
Calculate the accuracy on the test data using the method `score`:

In [15]:
```
print("Jaccard index: ", jaccard_score(Y_test, yhat, pos_label=1))
print("F1-score: ", f1_score(Y_test, yhat, average='weighted'))
```

```
Jaccard index:  0.8
F1-score:  0.8148148148148149
```

Lets look at the confusion matrix:

In [16]:
```
yhat=logreg_cv.predict(X_test)
plot_confusion_matrix(Y_test,yhat)
```



APPENDIX Machine Learning: Task 6

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`.

In [17]:
```
parameters = {'kernel':('linear', 'rbf','poly','rbf', 'sigmoid'),
              'C': np.logspace(-3, 3, 5),
              'gamma':np.logspace(-3, 3, 5)}
svm = SVC()
```

In [18]:
```
svm_cv = GridSearchCV(svm, parameters, cv=10)
svm_cv.fit(X_train, Y_train)
```

Out[18]:
```
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')})
```

In [19]:
```
print("tuned hpyerparameters :(best parameters) ",svm_cv.best_params_)
print("Accuracy score for the Support Vector Machine Method :",svm_cv.best_score_)
```

```
tuned hpyerparameters :(best parameters)  {'C': 1.0, 'gamma': 0.03162277660168379, 'kernel': 'sigmoid'}
Accuracy score for the Support Vector Machine Method : 0.8482142857142856
```

APPENDIX Machine Learning: Task 7

TASK 7

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

In [20]:
```
print("Accuracy score for the Support Vector Machine Method :",svm_cv.best_score_)

SVM_LR=SVC(C = svm_cv.best_params_["C"], gamma = svm_cv.best_params_["gamma"], kernel = svm_cv.best_params_["kernel"])
SVM_LR.fit(X_train,Y_train)
```

```
Accuracy score for the Support Vector Machine Method : 0.8482142857142856
```

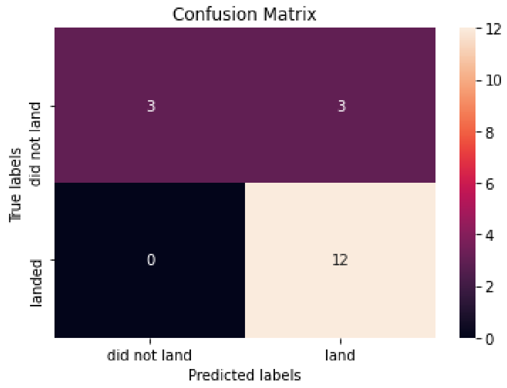
Out[20]:
```
SVC(gamma=0.03162277660168379, kernel='sigmoid')
```

We can plot the confusion matrix

In [21]:
```
SVM_yhat=SVM_LR.predict(X_test)
plot_confusion_matrix(Y_test,SVM_yhat)

print("SVM Jaccard index: ", jaccard_score(Y_test, SVM_yhat, pos_label=1))
print("SVM F1-score: ", f1_score(Y_test, SVM_yhat, average='weighted'))
```

```
SVM Jaccard index:  0.8
SVM F1-score:  0.8148148148148149
```



APPENDIX Machine Learning: Task 8

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`.

In [22]:
```
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()

cv_value=10
tree_cv = GridSearchCV(tree, parameters, scoring='accuracy',cv = cv_value)
tree_cv = tree_cv.fit(X_train, Y_train)
Best_DT = tree_cv.best_estimator_
tree_cv_acc_score = tree_cv.score(X_train, Y_train)
print('Accuracy score of the Decision Tree Model with Training data: ', tree_cv_acc_score )
DT_acc_score = Best_DT.score(X_test, Y_test)
print('Accuracy score of the Decision Tree Model with Test data: ', DT_acc_score )
```

```
Accuracy score of the Decision Tree Model with Training data:  0.9027777777777778
Accuracy score of the Decision Tree Model with Test data:  0.8333333333333334
```

In [23]:
```
print("tuned hpyerparameters :(best parameters) ",tree_cv.best_params_)
print("Accuracy score of the Decision Tree Model using Tuned Hyperparameters :",tree_cv.best_score_)
```

```
tuned hpyerparameters :(best parameters)  {'criterion': 'entropy', 'max_depth': 16, 'max_features': 'sqrt', 'min_samples_leaf': 1, 'min_samples_split': 10, 'splitter': 'random'}
Accuracy score of the Decision Tree Model using Tuned Hyperparameters : 0.8767857142857143
```

APPENDIX Machine Learning: Task 9

TASK 9

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

In [24]:
```
DT_acc_score = Best_DT.score(X_test, Y_test)
print('Accuracy score of the Decision Tree Model with Test data: ', DT_acc_score )
```

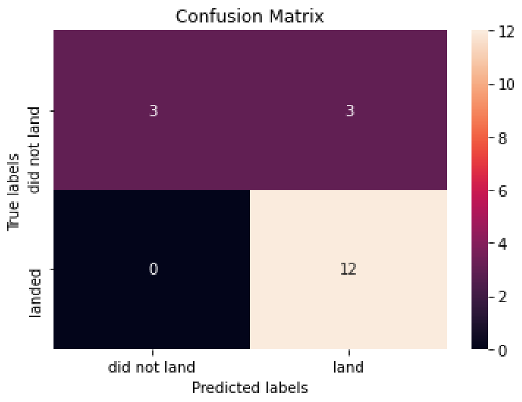
```
Accuracy score of the Decision Tree Model with Test data:  0.8333333333333334
```

We can plot the confusion matrix

In [25]:
```
DT_yhat = tree_cv.predict(X_test)
plot_confusion_matrix(Y_test,DT_yhat)

print("DT Jaccard index: ", jaccard_score(Y_test, DT_yhat, pos_label=1))
print("DT F1-score: ", f1_score(Y_test, DT_yhat, average='weighted'))
```

```
DT Jaccard index:  0.8
DT F1-score:  0.8148148148148149
```



APPENDIX Machine Learning: Task 10a

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`.

In [27]:
```
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()
```

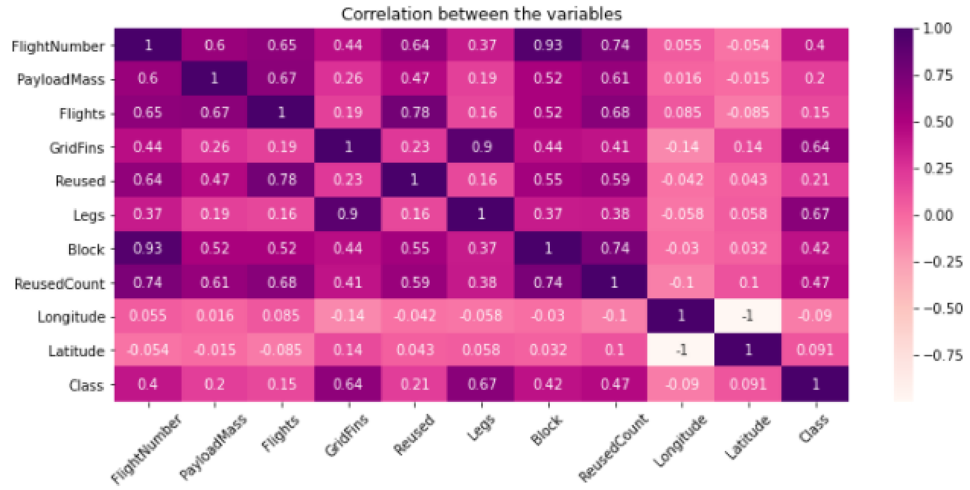
In [81]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 90 entries, 0 to 89
Data columns (total 18 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   FlightNumber    90 non-null     int64
 1   Date            90 non-null     object
 2   BoosterVersion  90 non-null     object
 3   PayloadMass     90 non-null     float64
 4   Orbit           90 non-null     object
 5   LaunchSite      90 non-null     object
 6   Outcome         90 non-null     object
 7   Flights         90 non-null     int64
 8   GridFins        90 non-null     bool
 9   Reused          90 non-null     bool
 10  Legs            90 non-null     bool
 11  LandingPad      64 non-null     object
 12  Block           90 non-null     float64
 13  ReusedCount     90 non-null     int64
 14  Serial          90 non-null     object
 15  Longitude       90 non-null     float64
 16  Latitude        90 non-null     float64
 17  Class           90 non-null     int64
dtypes: bool(3), float64(4), int64(4), object(7)
memory usage: 10.9+ KB
```

APPENDIX Machine Learning: Task 10b

In [28]:
```python
#correlation matrix and the heatmap
plt.subplots(figsize=(12,5))
data_correlation=data.corr()
sns.heatmap(data_correlation,annot=True,cmap='RdPu')
plt.title('Correlation between the variables')
plt.xticks(rotation=45)
```

Out[28]:
```
(array([ 0.5,  1.5,  2.5,  3.5,  4.5,  5.5,  6.5,  7.5,  8.5,  9.5, 10.5]),
 [Text(0.5, 0, 'FlightNumber'),
  Text(1.5, 0, 'PayloadMass'),
  Text(2.5, 0, 'Flights'),
  Text(3.5, 0, 'GridFins'),
  Text(4.5, 0, 'Reused'),
  Text(5.5, 0, 'Legs'),
  Text(6.5, 0, 'Block'),
  Text(7.5, 0, 'ReusedCount'),
  Text(8.5, 0, 'Longitude'),
  Text(9.5, 0, 'Latitude'),
  Text(10.5, 0, 'Class')])
```



APPENDIX Machine Learning: Task 10c

```
In [29]: knn_cv = GridSearchCV(KNN, parameters, cv=10)

         # fitting the model for grid search
         knn_cv=knn_cv.fit(X_train, Y_train)

         print(knn_cv.best_params_)
         knn_train_accuracy = knn_cv.best_score_ *100
         print("Accuracy for K Nearest Neighbor Method with training data is : {:.2f}%".format(knn_train_accuracy) )

         {'algorithm': 'auto', 'n_neighbors': 10, 'p': 1}
         Accuracy for K Nearest Neighbor Method with training data is : 84.82%

In [30]: KNN = KNeighborsClassifier(algorithm = knn_cv.best_params_["algorithm"], n_neighbors = knn_cv.best_params_["n_neighbors"], p = k
         nn_cv.best_params_["p"])

         knn_cv.fit(X, Y)

         knn_yhat=knn_cv.predict(X_test)

         knn_test_accuracy=knn_cv.best_score_*100

         print("Accuracy for K Nearest Neighbor Method with test data is : {:.2f}%".format(knn_test_accuracy) )

         Accuracy for K Nearest Neighbor Method with test data is : 84.44%

In [31]: from sklearn import metrics
         from sklearn.metrics import classification_report,confusion_matrix
         print("\nTrain set Accuracy: ", metrics.accuracy_score(Y_train, knn_cv.predict(X_train)))
         print("Test set Accuracy: ", metrics.accuracy_score(Y_test, knn_yhat))
         print ("\nClassification Report:\n",classification_report(Y_test, knn_yhat))
         print("tuned hpyerparameters :(best parameters) ",knn_cv.best_params_)
         print("Accuracy for K Nearest Neighbor Method with test data is : {:.2f}%:",knn_cv.best_score_)

         Train set Accuracy:  0.875
         Test set Accuracy:  0.9444444444444444

         Classification Report:
                       precision    recall  f1-score   support

                    0       1.00      0.83      0.91         6
                    1       0.92      1.00      0.96        12

             accuracy                           0.94        18
            macro avg       0.96      0.92      0.93        18
         weighted avg       0.95      0.94      0.94        18

         tuned hpyerparameters :(best parameters)  {'algorithm': 'auto', 'n_neighbors': 5, 'p': 1}
         Accuracy for K Nearest Neighbor Method with test data is : {:.2f}%: 0.8444444444444444
```

APPENDIX Machine Learning: Task 11

TASK 11

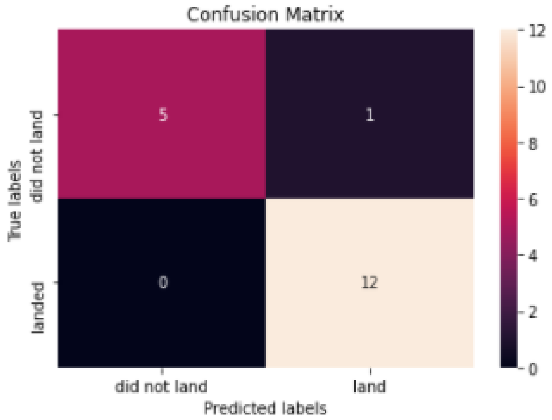
Calculate the accuracy of tree_cv on the test data using the method score:

In [32]:
```
tree_cv_acc_score = tree_cv.score(X_train, Y_train)
print('Accuracy score of the Decision Tree Model with Training data: ', tree_cv_acc_score )
DT_acc_score = Best_DT.score(X_test, Y_test)
print('Accuracy score of the Decision Tree Model with Test data: ', DT_acc_score )
```

```
Accuracy score of the Decision Tree Model with Training data:  0.9027777777777778
Accuracy score of the Decision Tree Model with Test data:  0.8333333333333334
```

We can plot the confusion matrix

In [33]:
```
knn_yhat = knn_cv.predict(X_test)
plot_confusion_matrix(Y_test,knn_yhat)
```



APPENDIX Machine Learning: Task 12

TASK 12

Find the method performs best:

In [34]:
```
print("Accuracy of Logistic Regression Method method :",logreg_cv.best_score_)
print("LR Jaccard index: ", jaccard_score(Y_test, yhat, pos_label=1))
print("LR F1-score: ", f1_score(Y_test, yhat, average='weighted'))

print("Accuracy score for the Support Vector Machine Method :",svm_cv.best_score_)
print("SVM Jaccard index: ", jaccard_score(Y_test, SVM_yhat, pos_label=1))
print("SVM F1-score: ", f1_score(Y_test, SVM_yhat, average='weighted'))

print('Accuracy score of the Decision Tree Model with Test data: ', DT_acc_score )
print("DT Jaccard index: ", jaccard_score(Y_test, DT_yhat, pos_label=1))
print("DT F1-score: ", f1_score(Y_test, DT_yhat, average='weighted'))


print("Accuracy for K Nearest Neighbor Method with test data is : {:.2f}%".format(knn_test_accuracy) )
print("KNN Jaccard index: ", jaccard_score(Y_test, knn_yhat, pos_label=1))
print("KNN F1-score: ", f1_score(Y_test, knn_yhat, average='weighted'))
```

```
Accuracy of Logistic Regression Method method : 0.8464285714285713
LR Jaccard index:  0.8
LR F1-score:  0.8148148148148149
Accuracy score for the Support Vector Machine Method : 0.8482142857142856
SVM Jaccard index:  0.8
SVM F1-score:  0.8148148148148149
Accuracy score of the Decision Tree Model with Test data:  0.8333333333333334
DT Jaccard index:  0.8
DT F1-score:  0.8148148148148149
Accuracy for K Nearest Neighbor Method with test data is : 84.44%
KNN Jaccard index:  0.9230769230769231
KNN F1-score:  0.9430303030303031
```

REFERENCES

- SpaceX Falcon 9 landing image (cover page) sourced from Wikipedia page "*Falcon 9*": https://en.wikipedia.org/wiki/Falcon_9
- SpaceX data source: Wikipedia page "*List of Falcon 9 and Falcon Heavy launches*" as of 2021-06-09: https://en.wikipedia.org/w/index.php?title=List_of_Falcon_9_and_Falcon_Heavy_launches&oldid=1027686922
- Table of contents image source Dreamstime: table-contents-write-document-isolated-blue-background-166839122.jpg (800×533) (dreamstime.com)
- Interactive Dashboard data source: https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/datasets/spacex_launch_dash.csv
- Executive Summary image from Shutterstock.com
- Welcome (Introduction) word cloud source: istockphoto.com -- welcome-phrase-words-cloud-concept-picture-id481560677 (612×442) (istockphoto.com)
- Data collection image from www.ceros.com -- data-collection.png (1013×786) (xcademy.in)
- Methodology word cloud source: istockphoto.com -- methodology-word-cloud-concept-4-illustration-id683486486 (1024×739) (istockphoto.com)
- Predictive analytics image from ibm.com: predictive_bi.png (693×515)
- Discussion image from Discussionlistservices.com: discussion.jpg (1280×960) (discussionlistservices.com)
- Conclusion image (« The End ») from Excelsior College: conclusion.jpg (660×340) (excelsior.edu)