IBM Developer
SKILLS NETWORK

Winning Space Race with Data Science

Spencer Ng
26/05/2024

Section 1

Methodology

Outline

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

Executive Summary

- Summary of methodologies

- SpaceX Data Collection using SpaceX API
- SpaceX Data Collection with Web Scraping
- SpaceX Data Wrangling
- SpaceX EDA using SQL
- Space-X EDA Data Visulaization with Pandas and Matplotlib
- Space-X Launch Sites Analysis with Folium and PlotlyDash
- SpaceX Machine Learning Landing Prediction

- Summary of all results

- EDA results
- Interactive Visualization and Dashboards
- Machine Learning delivering predictive analytics

Methodology

Executive Summary

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

Data Collection

- Describe how data sets were collected.

- Data was first collected using SpaceX API (a REST API) by making a get request to the SpaceX API.
- This API gives us info about payload mass, launchsites, orbits, landing outcomes and other useful datapoints that can be used to making predictions
- Another source for getting Space X data is webscraping Wikipedia using Beatiful soup

Data Collection – SpaceX API

- Data collected using SpaceX API . Input get request to the SpaceX API and decoded the response content as a Json result which is then converted into a Pandas data frame

- Add the GitHub URL of the completed SpaceX API calls notebook https://github.com/SuperVerkaufer/IBM-data-science-capstone/blob/5273bb1d13679321338e0d582b9767dc04218309/2.%20Data%20Collection.ipynb

Flow diagram

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

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

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

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

```
response.status_code
```

```
200
```

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

```
# Use json_normalize meethod to convert the json result into a dataframe
respjson = response.json()
DATA = pd.json_normalize(respjson)
```

Data Collection - Scraping

- Scraped Falcon 9 historical launch records from Wikipedia using BeautifulSoup request, to extract the Falcon 9 launch records from the HTML table of the Wikipedia page, then created a data frame by parsing the launch HTML.

- Add the GitHub URL of the completed web scraping notebook, as an external reference and peer-review purpose

https://github.com/SuperVerkaufer/IBM-data-science-capstone/blob/5273bb1d13679321338e0d582b9767dc04218309/3.%20webscraping.ipynb

Flow diagram

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.

```
# 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`

```
# Use BeautifulSoup() to create a BeautifulSoup object from a response text content
soup = BeautifulSoup(response.content, 'html.parser')
```

TASK 2: Extract all column/variable names from the HTML table header

Next, we want to collect all relevant column names from the HTML table header

Let's try to find all tables on the wiki page first. If you need to refresh your memory about `BeautifulSoup`, please check the external reference link towards the end of this lab

```
# Use the find_all function in the BeautifulSoup object, with element type `table`
# Assign the result to a list called `html_tables`
html_tables = soup.find_all('table')
```

Starting from the third table is our target table contains the actual launch records.

```
# Let's print the third table and check its content
first_launch_table = html_tables[2]
print(first_launch_table)
```

TASK 3: Create a data frame by parsing the launch HTML tables

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

```
launch_dict= dict.fromkeys(column_names)

# Remove an irrelvant column
del launch_dict['Date and time ( )']

# Let's initial the launch_dict with each value to be an empty list
launch_dict['Flight No.'] = []
launch_dict['Launch site'] = []
launch_dict['Payload'] = []
launch_dict['Payload mass'] = []
launch_dict['Orbit'] = []
launch_dict['Customer'] = []
launch_dict['Launch outcome'] = []
# Added some new columns
launch_dict['Version Booster']=[]
launch_dict['Booster landing']=[]
launch_dict['Date']=[]
launch_dict['Time']=[]
```

Data Wrangling

- Enumerate the various outcomes and then group them by good or bad outcomes
- Subsequently code the outcomes as either good or bad outcomes
- Add the GitHub URL of your completed data wrangling related notebooks, as an external reference and peer-review purpose
- https://github.com/SuperVerkaufer/IBM-data-science-capstone/blob/5273bb1d13679321338e0d582b9767dc04218309/4.%20Data%20wrangling.ipynb

Flow diagram

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

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

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

```
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
```

```
bad_outcomes=set(landing_outcomes.keys()[[1,3,5,6,7]])
bad_outcomes
```

```
{'False ASDS', 'False Ocean', 'False RTLS', 'None ASDS', 'None None'}
```

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`:

```
# landing_class = 0 if bad_outcome
# landing_class = 1 otherwise
df['Class'] = df['Outcome'].apply(lambda x: 0 if x in bad_outcomes else 1)
df['Class'].value_counts()
```

```
1    60
0    30
Name: Class, dtype: int64
```

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

```
landing_class=df['Class']
df[['Class']].head(8)
```

EDA with Data Visualization

- Used scatter plots to visualize the relationship between 1) Flight Number VS Launch Site, 2) Payload mass and Launch Site, 3) FlightNumber and Orbit type, 4) Payload mass and Orbit type.
- Used Bar chart to visualize the success rate of each orbit type
- Line plot to show the launch success yearly increasing trend.
- Add the GitHub URL of your completed EDA with data visualization notebook, as an external reference and peer-review purpose
- https://github.com/SuperVerkaufer/IBM-data-science-capstone/blob/5273bb1d13679321338e0d582b9767dc04218309/6.%20eda%20data%20visualization.ipynb

EDA with SQL

- Display the names of the unique launch sites in the space mission
- Display 5 records where launch sites begin with the string 'CCA‘
- Display the total payload mass carried by boosters launched by NASA (CRS)
- Display average payload mass carried by booster version F9 v1.1
- List the date when the first successful landing outcome in ground pad was achieved
- List the names of the boosters which have success in drone ship and have payload mass greater than 4000 but less than 6000
- List the total number of successful and failure mission outcomes

Add the GitHub URL of your completed EDA with SQL notebook, as an external reference and peer-review purpose

- https://github.com/SuperVerkaufer/IBM-data-science-capstone/blob/5273bb1d13679321338e0d582b9767dc04218309/5.%20eda%20SQL.ipynb

Build an Interactive Map with Folium

- Developed folium map to mark all the launch sites, and created map objects such as markers, circles, lines to mark the success or failure of launches for each launch site.
- launch outcomes were denoted as failure=0 or success=1.

Add the GitHub URL of your completed interactive map with Folium map, as an external reference and peer-review purpose

- https://github.com/SuperVerkaufer/IBM-data-science-capstone/blob/5273bb1d13679321338e0d582b9767dc04218309/7.%20Launch%20site%20location%20analysis.ipynb

Build a Dashboard with Plotly Dash

- Built an interactive dashboard application with Plotlydash by:
- Created a Launch Site Drop-down Input Component
- Created a callback function to render success-pie-chart based on selected site in the dropdown menu
- Created a Range Slider to Select Payload
- Created a callback function to render the success-payload-scatter-chart to showcase success by payload mass

Add the GitHub URL of your completed Plotly Dash lab, as an external reference and peer-review purpose

- https://github.com/SuperVerkaufer/IBM-data-science-capstone/blob/5273bb1d13679321338e0d582b9767dc04218309/8.%20Interactive%20Dashboard%20with%20Plotly%20-%20SpaceX%20Dash

Predictive Analysis (Classification)

- Prepared the data for modelling. First creating a Numpy array, then standardizing the data and finally splitting the data into training (to fit the data into the model) and test sets (to validate the model)

Add the GitHub URL of your completed predictive analysis lab, as an external reference and peer-review purpose

- https://github.com/SuperVerkaufer/IBM-data-science-capstone/blob/fa8728e617f190b21d2c0b8dfbf109936a37e1a1/9.%20SpaceX_Machine%20Learning%20Prediction_Part_5.ipynb

Flow diagram

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']).

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

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()
X = transform.fit_transform(X)
X
```

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`

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

Predictive Analysis (Classification)

- Built various models so that we can identify the best one.
- For each of the models, the GridsearchCV object was created with cv=10, then fit the training data into the GridSearch object
- The table shows the test data accuracy score for each model comparing them to show which performed best

Flow diagram

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()

# GridSearchCV object
logreg_cv = GridSearchCV(lr, parameters, cv=10)

#Fit the training data into the GridSearch object
logreg_cv.fit(X_train, Y_train)
```

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()
```

```
# GridSearchCV object with cv = 10
svm_cv = GridSearchCV(svm, parameters, cv= 10)

#Fit the training data into the GridSearch object
svm_cv.fit(X_train, Y_train)
```

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()
```

```
# GridSearchCV object with cv = 10
tree_cv = GridSearchCV(tree, parameters, cv= 10)

#Fit the training data into the GridSearch object
tree_cv.fit(X_train, Y_train)
```

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()
```

```
# GridSearchCV object with cv = 10
knn_cv = GridSearchCV(KNN, parameters, cv= 10)

#Fit the training data into the GridSearch object
knn_cv.fit(X_train, Y_train)
```

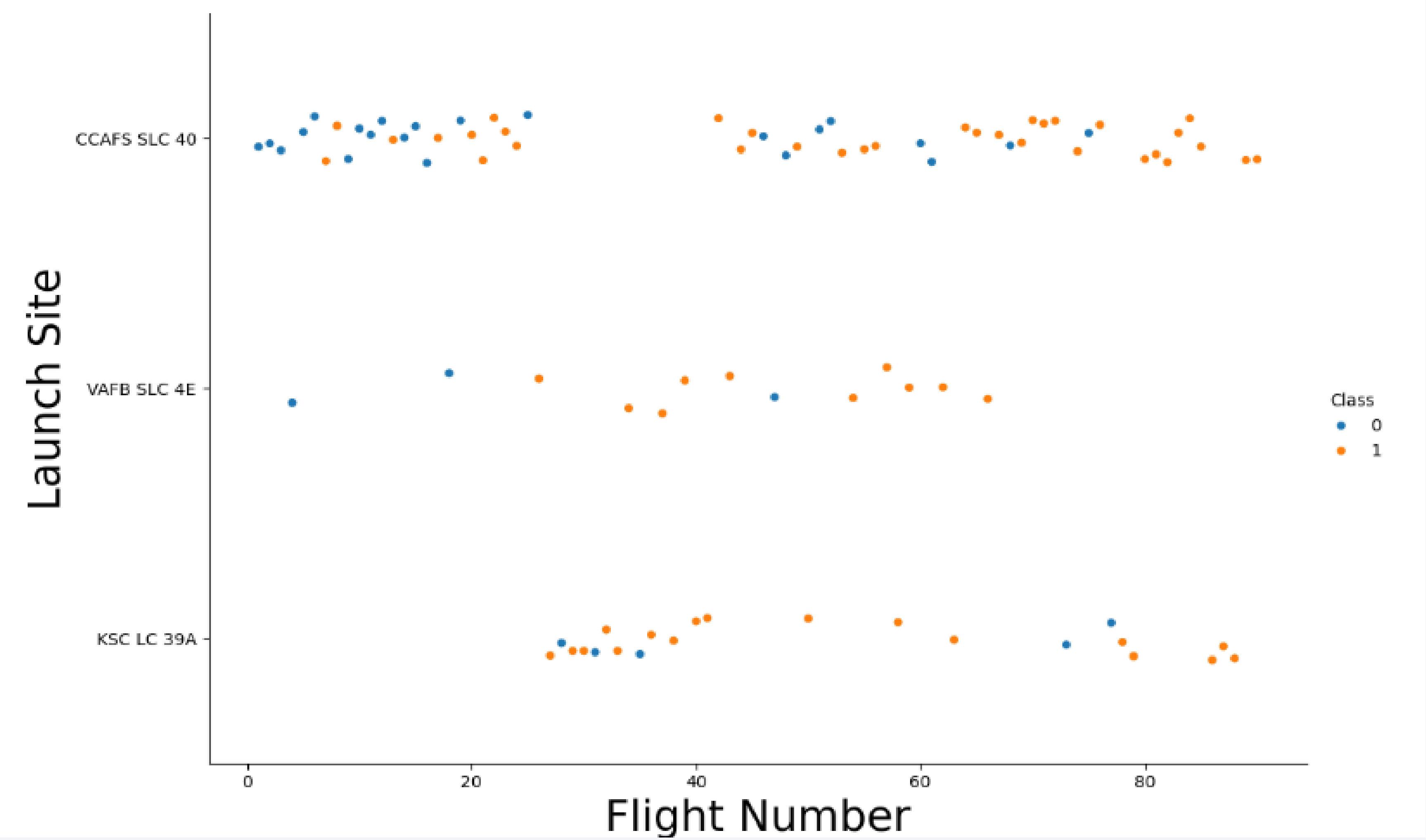
	0
Method	TestData Accuracy
Log Reg	0.833333
SVM	0.833333
Decision Tree	0.777778
KNN	0.833333

Section 2

Insights drawn from EDA

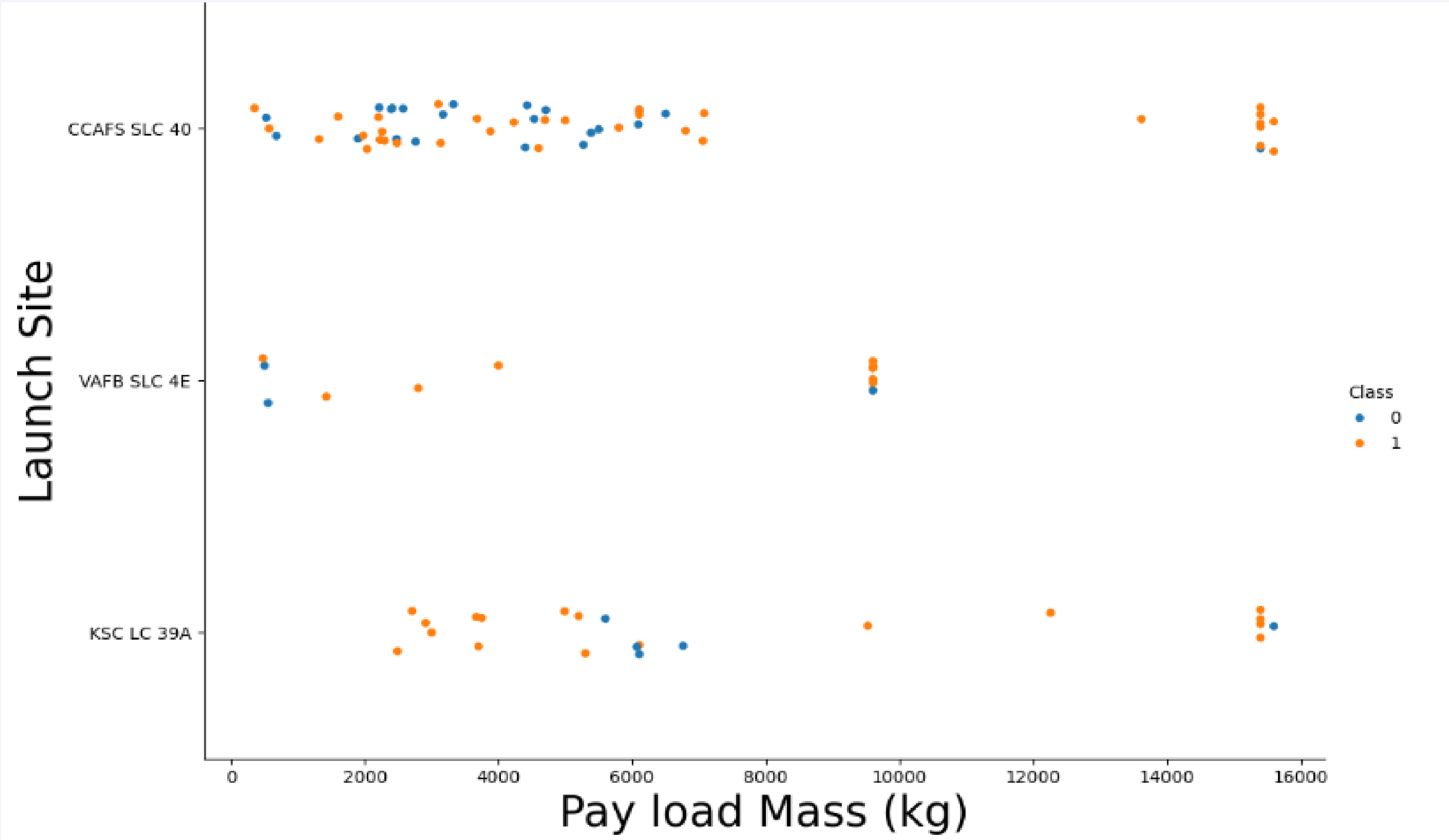
Flight Number vs. Launch Site

- Across launch sites; Successful landing (1) increases as flight number increases



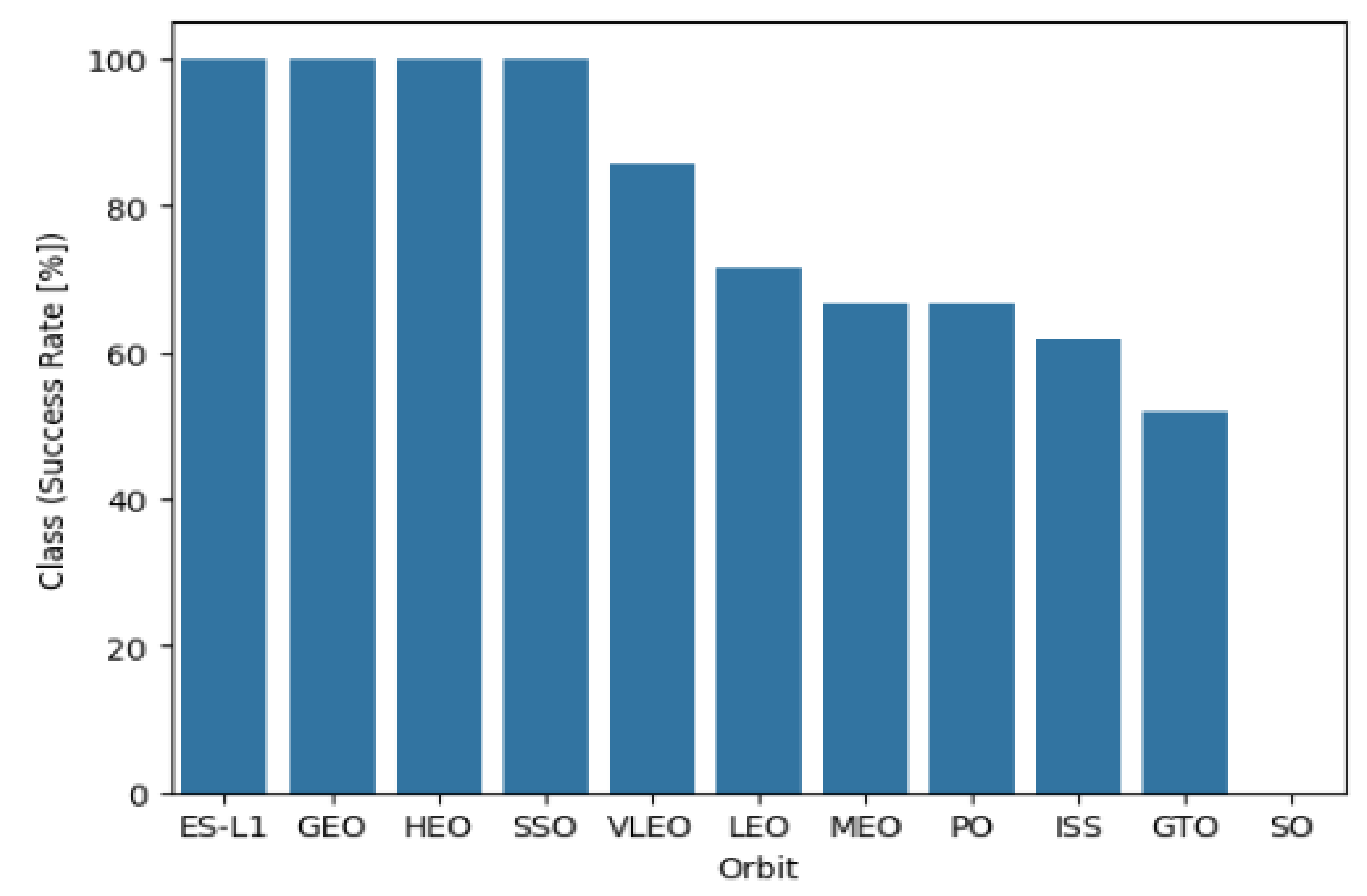
Payload vs. Launch Site

- Across launch sites; Successful landing (1) increases as Pay Load Mass increases



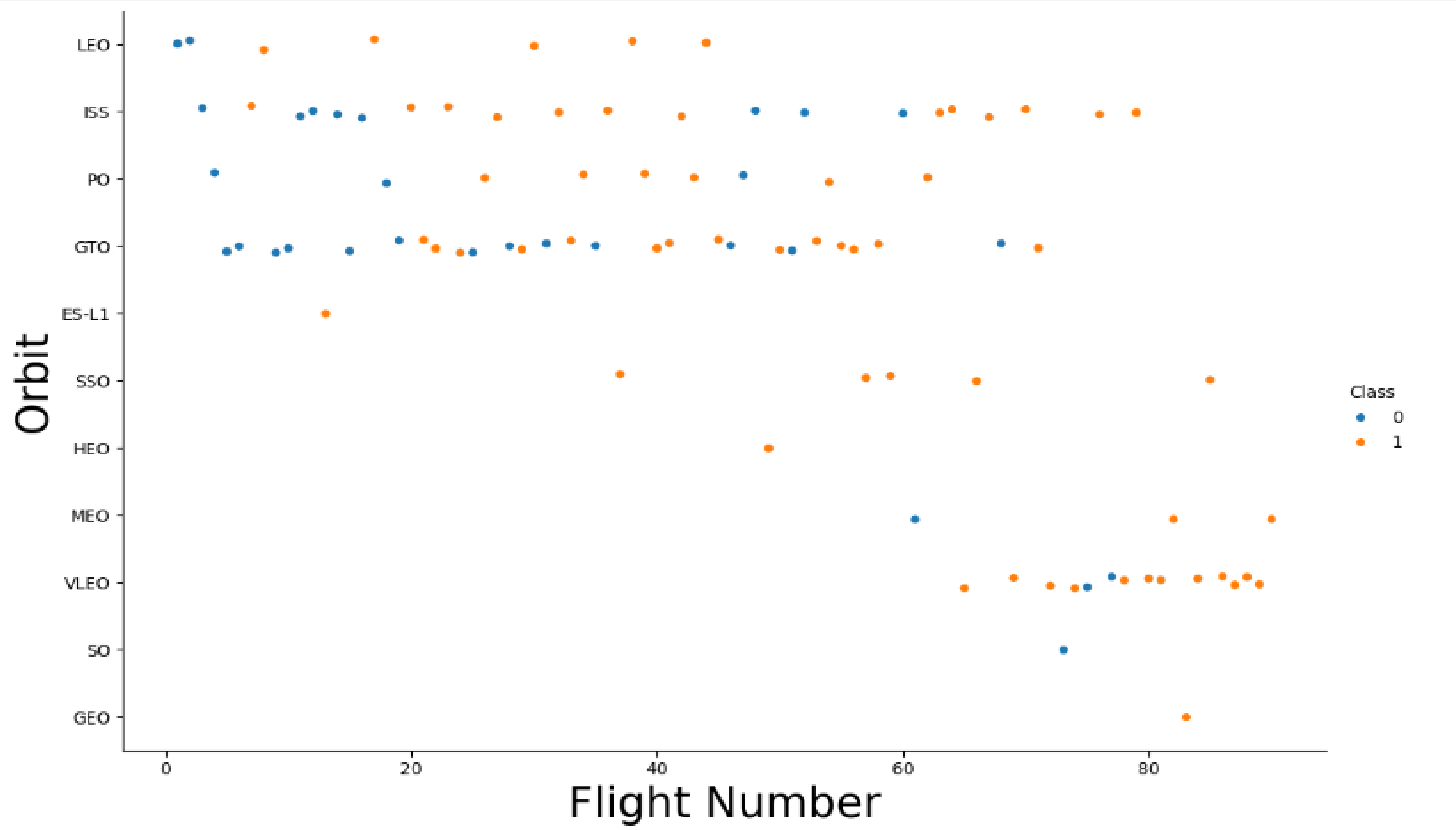
Success Rate vs. Orbit Type

- Orbits ES-L1, GEO, HEO, SSO have 100% success rate of landing



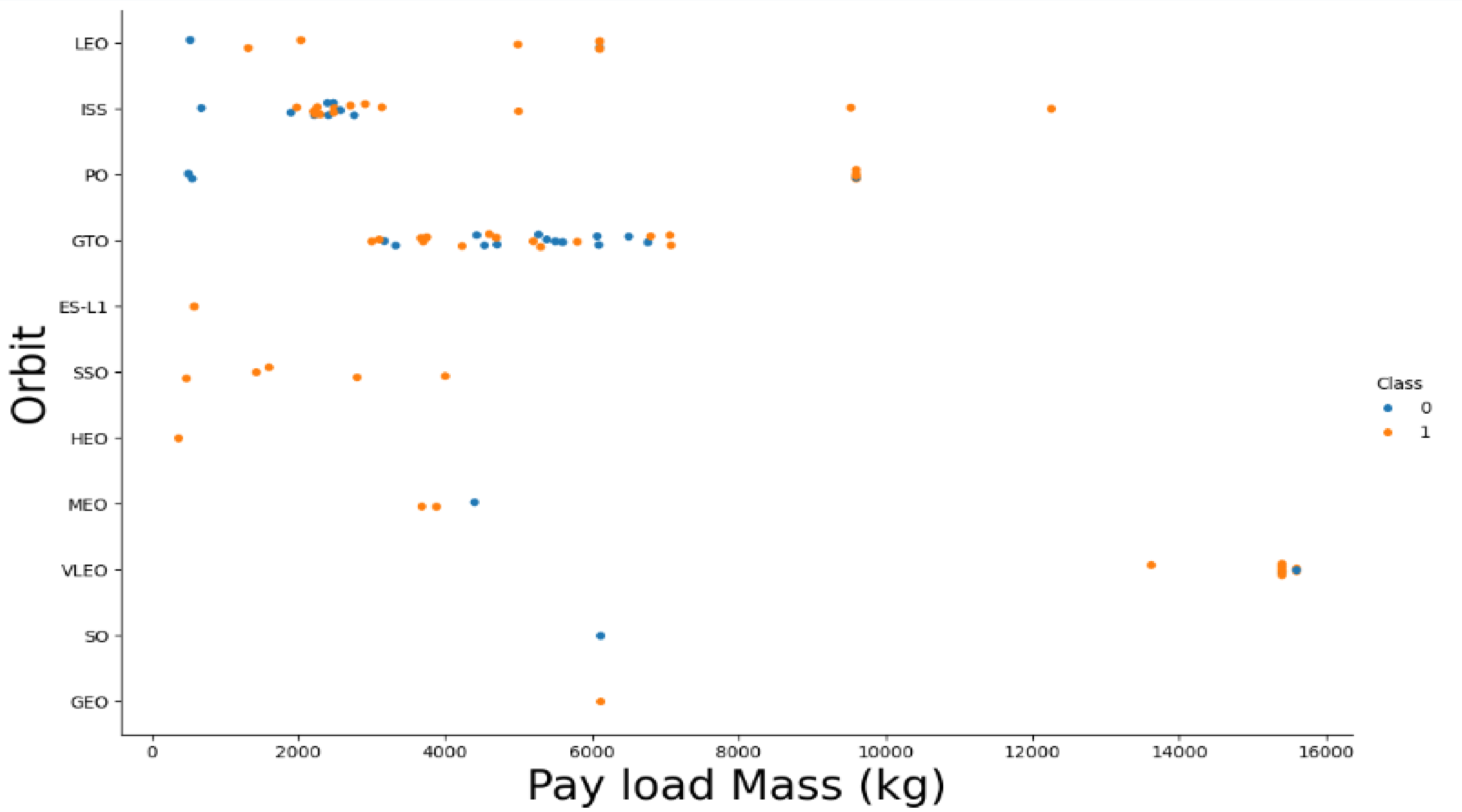
Flight Number vs. Orbit Type

- Only LEO orbit shows clear increasing successful landing with increasing flight number



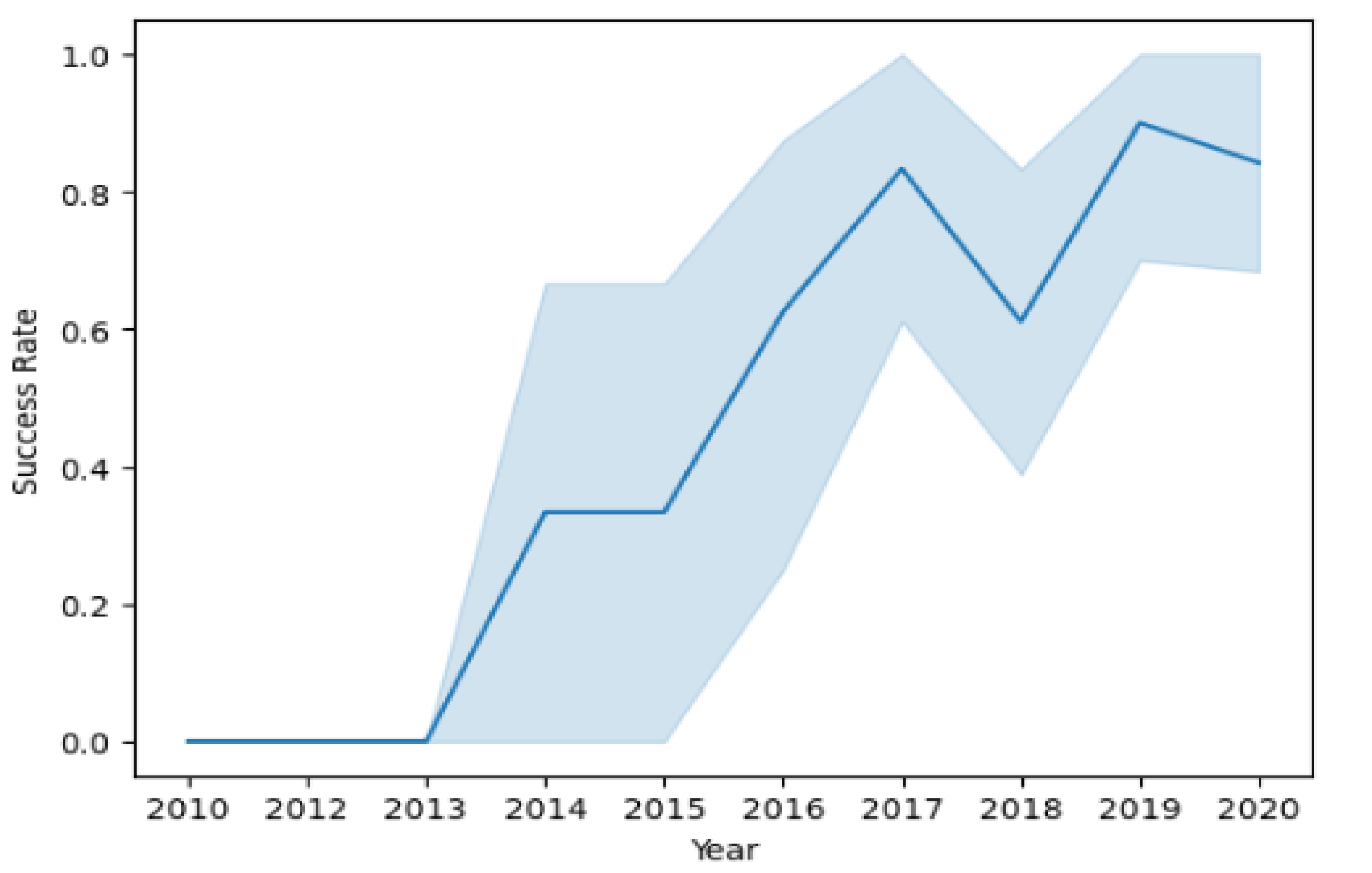
Payload vs. Orbit Type

- Only LEO, ISS & PO shows clear increasing successful landing with increasing payload mass



Launch Success Yearly Trend

- Increase in successful landing rate since 2013



All Launch Site Names

- SQL Select Distinct used to find and show unique launch sites

Task 1

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

In [11]:

```
#import pandas as pd
#import random
#unique_launch=df["Launch_Site"].unique().tolist()
#unique_launch
%sql SELECT distinct(Launch_Site) FROM SPACEXTABLE
#%sql select Unique(LAUNCH_SITE) from SPACEXTABLE;
```

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

Out[11]:

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

Launch Site Names Begin with 'CCA'

- Select instances where launchsites correspond to %CCA% from table and limit 5 instances

```
%sql SELECT * FROM SPACEXTABLE WHERE Launch_Site LIKE '%CCA%' LIMIT 5
```

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

Out[12]:

Date	Time (UTC)	Booster_Version	Launch_Site	Payload	PAYLOAD_MASS__KG_	Orbit	Customer	Mission_Outcome	Landing_Outcome
2010-06-04	18:45:00	F9 v1.0 B0003	CCAFS LC-40	Dragon Spacecraft Qualification Unit	0	LEO	SpaceX	Success	Failure (parachute)
2010-12-08	15:43:00	F9 v1.0 B0004	CCAFS LC-40	Dragon demo flight C1, two CubeSats, barrel of Brouere cheese	0	LEO (ISS)	NASA (COTS) NRO	Success	Failure (parachute)
2012-05-22	7:44:00	F9 v1.0 B0005	CCAFS LC-40	Dragon demo flight C2	525	LEO (ISS)	NASA (COTS)	Success	No attempt
2012-10-08	0:35:00	F9 v1.0 B0006	CCAFS LC-40	SpaceX CRS-1	500	LEO (ISS)	NASA (CRS)	Success	No attempt
2013-03-01	15:10:00	F9 v1.0 B0007	CCAFS LC-40	SpaceX CRS-2	677	LEO (ISS)	NASA (CRS)	Success	No attempt

Total Payload Mass

- Sum of group by corresponding to NASA customer produces the total payload mass for NASA

Task 3

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

```
#launch_site_CCA=[]
#for site in df["Launch_Site"]:
#    if "CCA" in site:
#        launch_site_CCA.append(site)
#random.choices(launch_site_CCA, k=5)
#launch_site_CCA=pd.Series(launch_site_CCA)
#launch_site_CCA.value_counts()
%sql SELECT Customer, SUM(PAYLOAD_MASS__KG_) FROM SPACEXTABLE WHERE Customer = "NASA (CRS)" GROUP BY Customer;
```

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

Customer	SUM(PAYLOAD_MASS__KG_)
NASA (CRS)	45596

Average Payload Mass by F9 v1.1

- Avg of group by corresponding to booster version F9 V1.1

Task 4

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

```
%sql SELECT Customer, AVG(PAYLOAD_MASS__KG_) FROM SPACEXTABLE WHERE Booster_Version = "F9 v1.1"
```

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

Customer	AVG(PAYLOAD_MASS__KG_)
SES	2928.4

First Successful Ground Landing Date

- Identify earliest successful landing by applying MIN on Date amongst successful landing outcomes

Task 5

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

Hint:Use min function

```
%sql SELECT Landing_Outcome, MIN(Date) FROM SPACEXTABLE WHERE Landing_Outcome = "Success (ground pad)"
```

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

Landing_Outcome	MIN(Date)
Success (ground pad)	2015-12-22

Successful Drone Ship Landing with Payload between 4000 and 6000

- Select booste version, amongst successful drone ship landing as landing outcome

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 Booster_Version, Landing_Outcome, PAYLOAD_MASS__KG_ FROM SPACEXTABLE WHERE Landing_Outcome = "Success (drone sh
```

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

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

Total Number of Successful and Failure Mission Outcomes

- Use Group By and count mission outcome within each group

Task 7

List the total number of successful and failure mission outcomes

```
%sql SELECT "Mission_Outcome", COUNT("Mission_Outcome") as Total FROM SPACEXTBL GROUP BY "Mission_Outcome";
```

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

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

Boosters Carried Maximum Payload

- Names of boosters identified by applying MAX to Payload Mass from SPACEXTABLE

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

```
select BOOSTER_VERSION, PAYLOAD_MASS__KG_ from SPACEXTABLE where PAYLOAD_MASS__KG_=(select max(PAYLOAD_MASS__KG_) from SPACEX
 select BOOSTER_VERSION, PAYLOAD_MASS__KG_ from SPACEXTABLE where PAYLOAD_MASS__KG_= (select max(PAYLOAD_MASS__KG_) from SPAC
```

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

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

2015 Launch Records

- Apply the ‘subsrt()’ in the select statement to get month & year from the date variable where substr(Date,7,4)='2015' for year and Landing_outcome was ‘Failure (drone ship’) and return the records

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.

```
%sql SELECT substr(Date,7,4), substr(Date, 4, 2),"Booster_Version", "Launch_Site", Payload, "PAYLOAD_MASS__KG_", "Mission_Outcome", "Landing _Outcome"
```

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

substr(Date,7,4)	substr(Date, 4, 2)	Booster_Version	Launch_Site	Payload	PAYLOAD_MASS__KG_	Mission_Outcome	Landing _Outcome
2015	01	F9 v1.1 B1012	CCAFS LC-40	SpaceX CRS-5	2395	Success	Failure (drone ship)
2015	04	F9 v1.1 B1015	CCAFS LC-40	SpaceX CRS-6	1898	Success	Failure (drone ship)

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

- Rank order by dates but includes both successful landings and failures

Task 10

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

```
%sql SELECT * FROM SPACEXTBL WHERE "Landing _Outcome" LIKE 'Success%' AND (Date BETWEEN '04-06-2010' AND '20-03-2017') ORDER BY Date DESC;
```

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

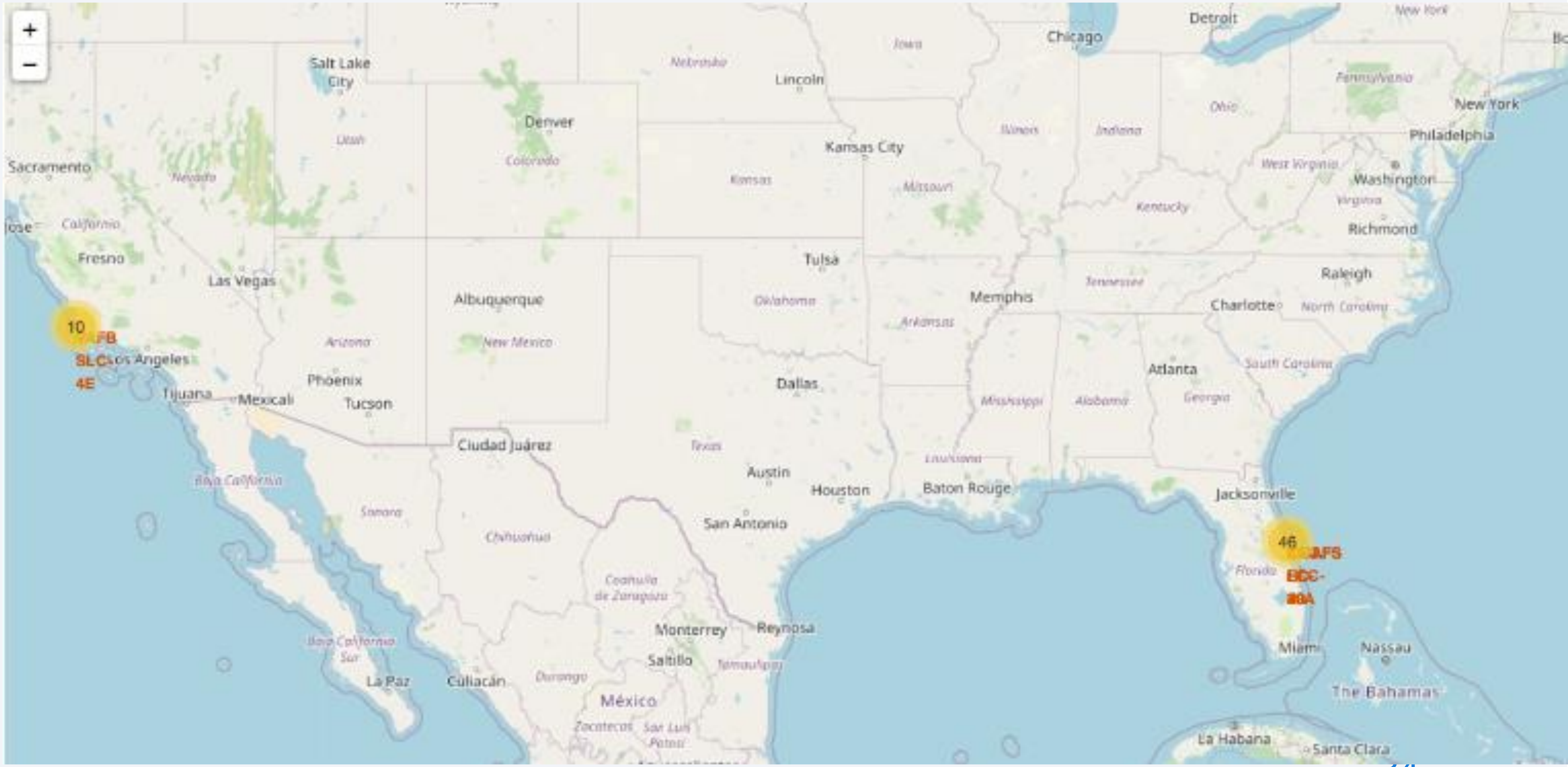
Date	Time (UTC)	Booster_Version	Launch_Site	Payload	PAYLOAD_MASS__KG_	Orbit	Customer	Mission_Outcome	Landing _Outcome
19-02-2017	14:39:00	F9 FT B1031.1	KSC LC-39A	SpaceX CRS-10	2490	LEO (ISS)	NASA (CRS)	Success	Success (ground pad)
18-10-2020	12:25:57	F9 B5 B1051.6	KSC LC-39A	Starlink 13 v1.0, Starlink 14 v1.0	15600	LEO	SpaceX	Success	Success
18-08-2020	14:31:00	F9 B5 B1049.6	CCAFS SLC-40	Starlink 10 v1.0, SkySat-19, -20, -21, SAOCOM 1B	15440	LEO	SpaceX, Planet Labs, PlanetIQ	Success	Success
18-07-2016	04:45:00	F9 FT B1025.1	CCAFS LC-40	SpaceX CRS-9	2257	LEO (ISS)	NASA (CRS)	Success	Success (ground pad)
18-04-2018	22:51:00	F9 B4 B1045.1	CCAFS SLC-40	Transiting Exoplanet Survey Satellite (TESS)	362	HEO	NASA (LSP)	Success	Success (drone ship)

Section 3

Launch Sites Proximities Analysis

All launch site locations

- All launch sites are at the coastline



Launch Outcomes

- CCAFS SLC40 has 3 successful launch outcomes out of 7

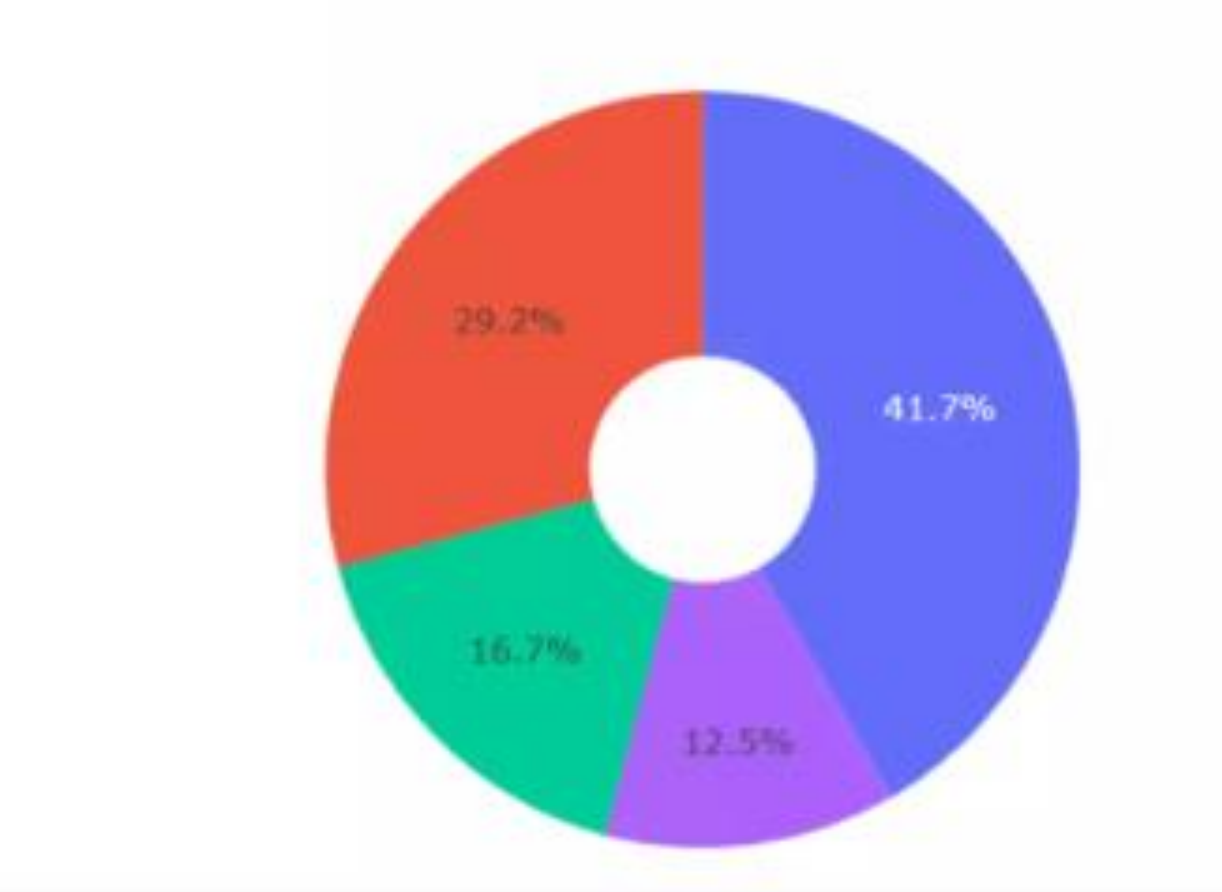
Proximity of launch site to coastline

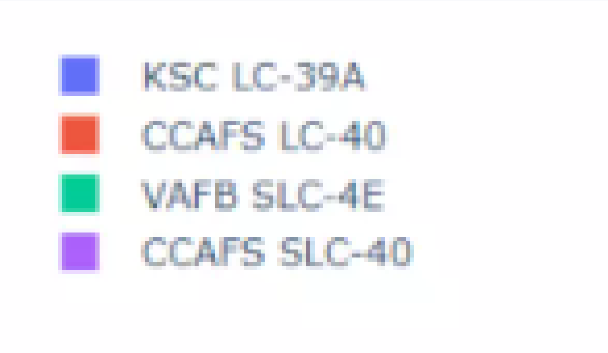
- Proximity of launchsite is 0.9KM to coastline

Section 4

Build a Dashboard with Plotly Dash

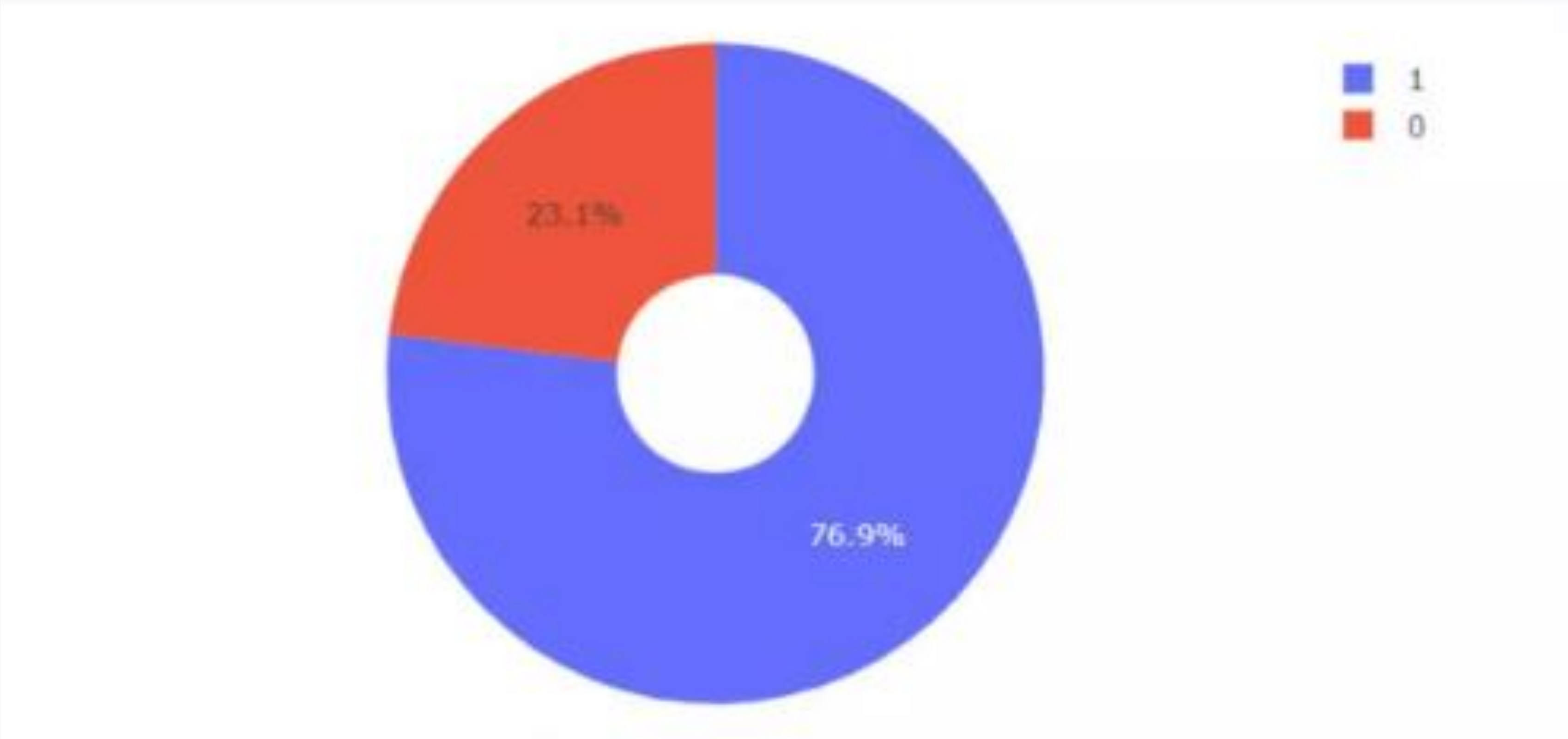
Total success launches across all sites





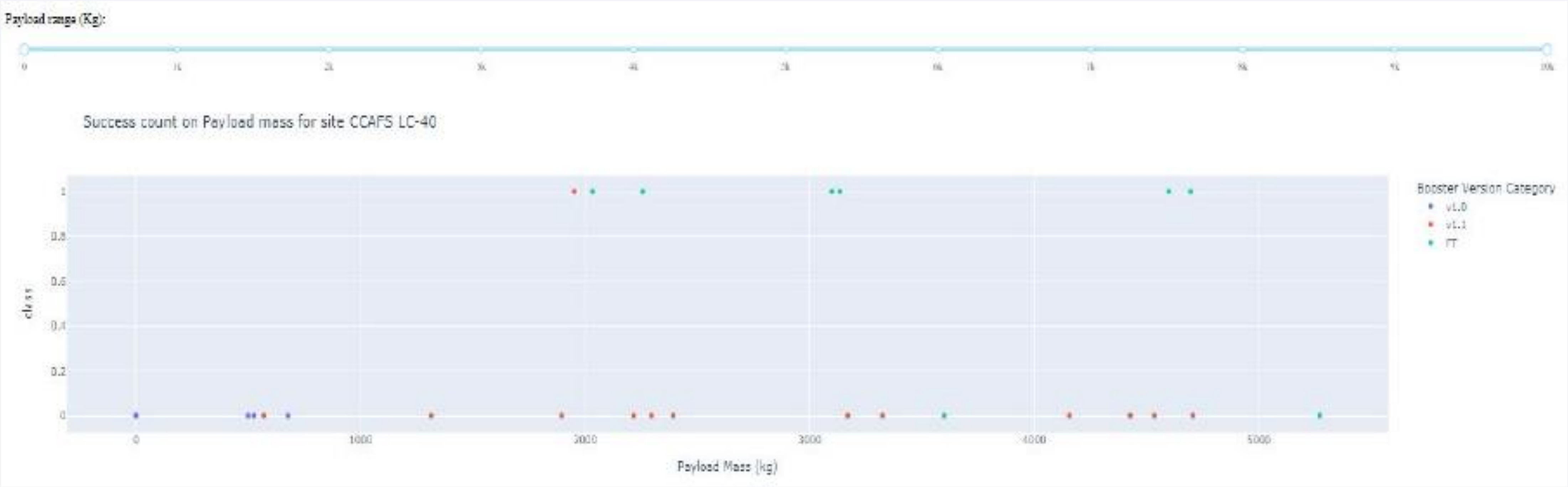
- Here you can see success by all sites, led by KSC LC39A

Success rate by site



- KSC LC 39-A achieved a 76.9% success rate and thus the site with highest success rate

Payload mass vs launch outcome by sites



- Success rate improves with rising payload mass

Section 5

Predictive Analysis (Classification)

Classification Accuracy

- All models had the same level of accuracy except for decision tree which has the poorest accuracy

TASK 12

Find the method performs best:

```
COMPARISON = pd.DataFrame({'Method' : ['TestData Accuracy']})

knn_accuracy=knn_cv.score(X_test, Y_test)
Decision_tree_accuracy=tree_cv.score(X_test, Y_test)
SVM_accuracy=svm_cv.score(X_test, Y_test)
Logistic_Regression=logreg_cv.score(X_test, Y_test)

COMPARISON['Log Reg'] = [Logistic_Regression]
COMPARISON['SVM'] = [SVM_accuracy]
COMPARISON['Decision Tree'] = [Decision_tree_accuracy]
COMPARISON['KNN'] = [knn_accuracy]

COMPARISON.transpose()
```

	0
Method	TestData Accuracy
Log Reg	0.833333
SVM	0.833333
Decision Tree	0.777778
KNN	0.833333

Confusion Matrix

- Most of the models had the same accuracy – meaning it could predict accurately all the successful outcomes. But it also produced false positives. The KNN model is shown as an example

TASK 11

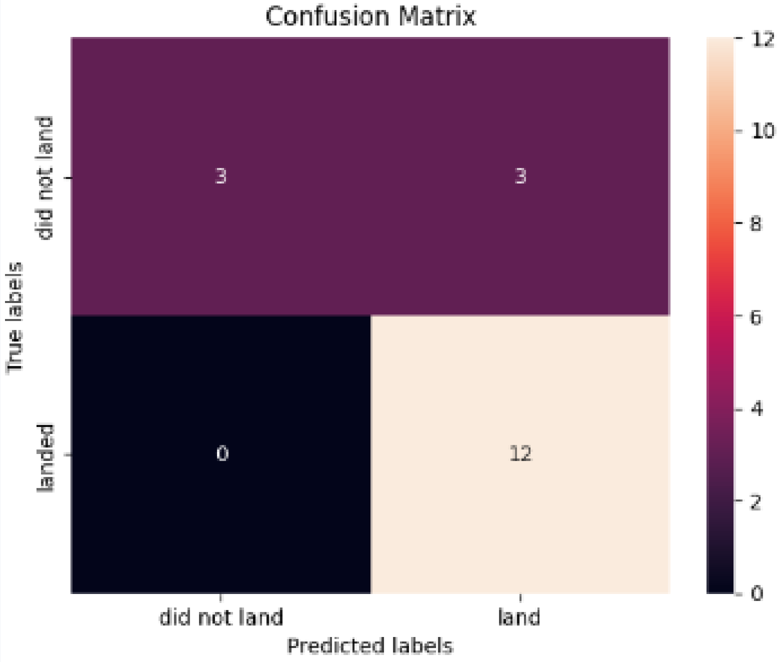
Calculate the accuracy of knn_cv on the test data using the method `score` :

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

```
0.8333333333333334
```

We can plot the confusion matrix

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



Conclusions

- The most predictive variables of a successful launchsite is 1) PayLoad Mass, 2) Number of Flights, 3) Orbits ES-L1, GEO, HEO & SSO are the most successful
  - Furthmore, launches after 2013 were also successful
- Various models were tried to see which model achieved the best fit. Apart from decision tree, which achieved the poorest accuracy, the rest of the models achieved comparable accuracy

Thank you!