IBM Developer
SKILLS NETWORK

Winning Space Race with Data Science

Baihan Jiang
May 14, 2023

Outline

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

Executive Summary

- Summary of methodologies
  - Data Collection through API
  - Data Collection with Web Scraping
  - Data Wrangling
  - Exploratory Data Analysis with SQL
  - Exploratory Data Analysis with Data Visualization
  - Interactive Visual Analytics with Folium
  - Machine Learning Prediction
- Summary of all results
  - Exploratory Data Analysis result
  - Interactive analytics in screenshots
  - Predictive Analytics result

Introduction

- Project background and context

  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. This goal of the project is to create a machine learning pipeline to predict if the first stage will land successfully.

- Problems you want to find answers

  - What factors determine if the rocket will land successfully?
  - The interaction amongst various features that determine the success rate of a successful landing.
  - What operating conditions needs to be in place to ensure a successful landing program.

Section 1

Methodology

Methodology

- Data collection methodology: Data was collected using SpaceX API and web scraping from Wikipedia.
- Perform data wrangling: One-hot encoding was applied to categorical features
- 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

- Grequest to the SpaceX API.
- Decoded the response content as a Json using .json() function call and turn it into a pandas dataframe using .json_normalize().
- Cleaned the data, checked for missing values and replace it while necessary.
- Perform web scraping from Wikipedia for Falcon 9 launch records with BeautifulSoup.
- The objective was to extract the launch records as HTML table, parse the table and convert it to a pandas dataframe for future analysis.

Data Collection – SpaceX API

- Request to SpaceX API for Data Collection, clean the requested data and process data wrangling and formatting.
- Reference: https://github.com/charles0624/IBM-Capstone-Project---SpaceX-Falcon-9-first-stage-Landing-Prediction/blob/main/Data%20Collection%20API.ipynb

SpaceX API

1.Request to get rocket lunch data through API

```
In [1]: space_url="https://api.spacexdata.com/v4/launches/past"
```

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

2.Use json_normalize method to convert json result to dataframe

```
In [ ]: static_json_df=res.jason()
```

```
In [ ]: data=pd.jason_normalize(static_json_df)
```

3.Perform data cleaning and filling in the missing values

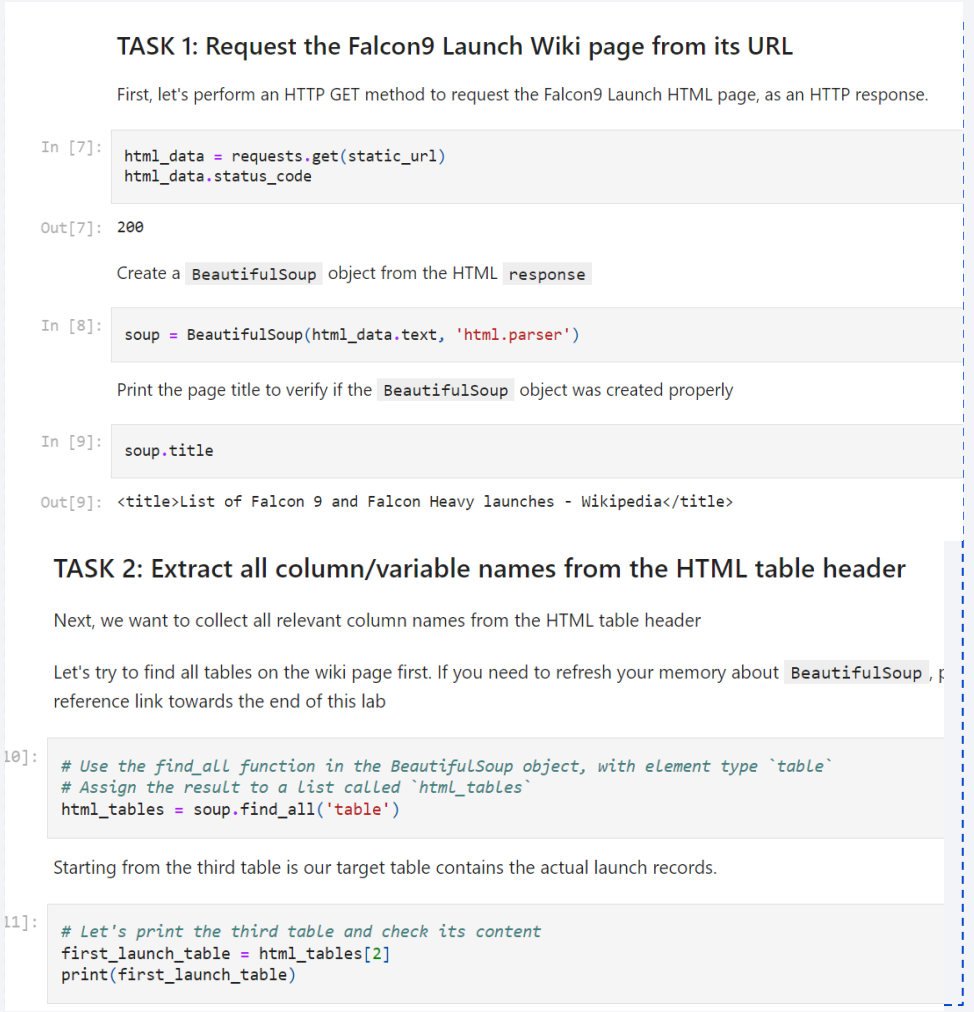
```
In [ ]: rows=data_falcon9['PayLoadMass'].values.tolist()[0]

        df_rows=pd.DataFrame(rows)
        df_rows=df_rows.replace(np.nan, PayLoadMass)

        data_falcon9['PayLoadMass'][0]=df_rows.values
        data_falcon9
```

Data Collection - Scraping

- Applied web scrapping to webscrap Falcon 9 launch records with BeautifulSoup
- Parsed the table and converted it into a pandas dataframe.
- Reference: https://github.com/charles0624/IBM-Capstone-Project---SpaceX-Falcon-9-first-stage-Landing-Prediction/blob/main/Data%20Collection%20with%20Web%20Scraping.ipynb



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

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

In [7]:
```
html_data = requests.get(static_url)
html_data.status_code
```

Out[7]: 200

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

In [8]:
```
soup = BeautifulSoup(html_data.text, 'html.parser')
```

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

In [9]:
```
soup.title
```

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

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

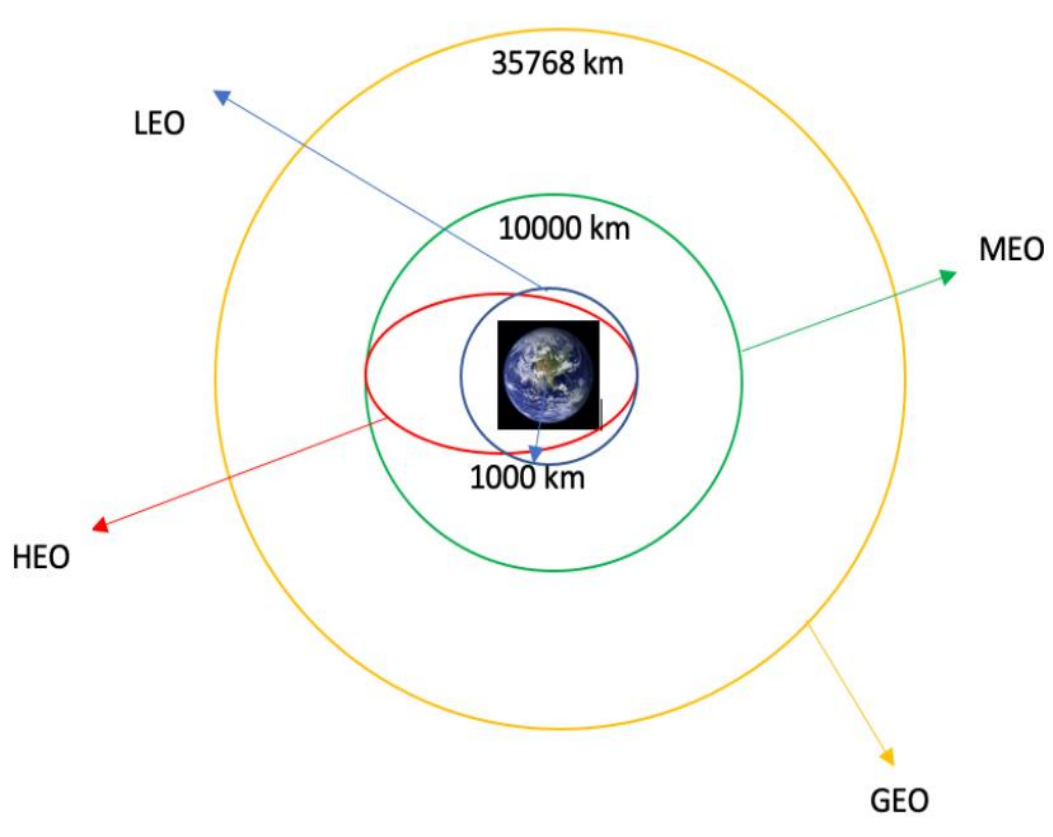
Let's try to find all tables on the wiki page first. If you need to refresh your memory about `BeautifulSoup`, p
reference link towards the end of this lab

10]:
```
# 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.

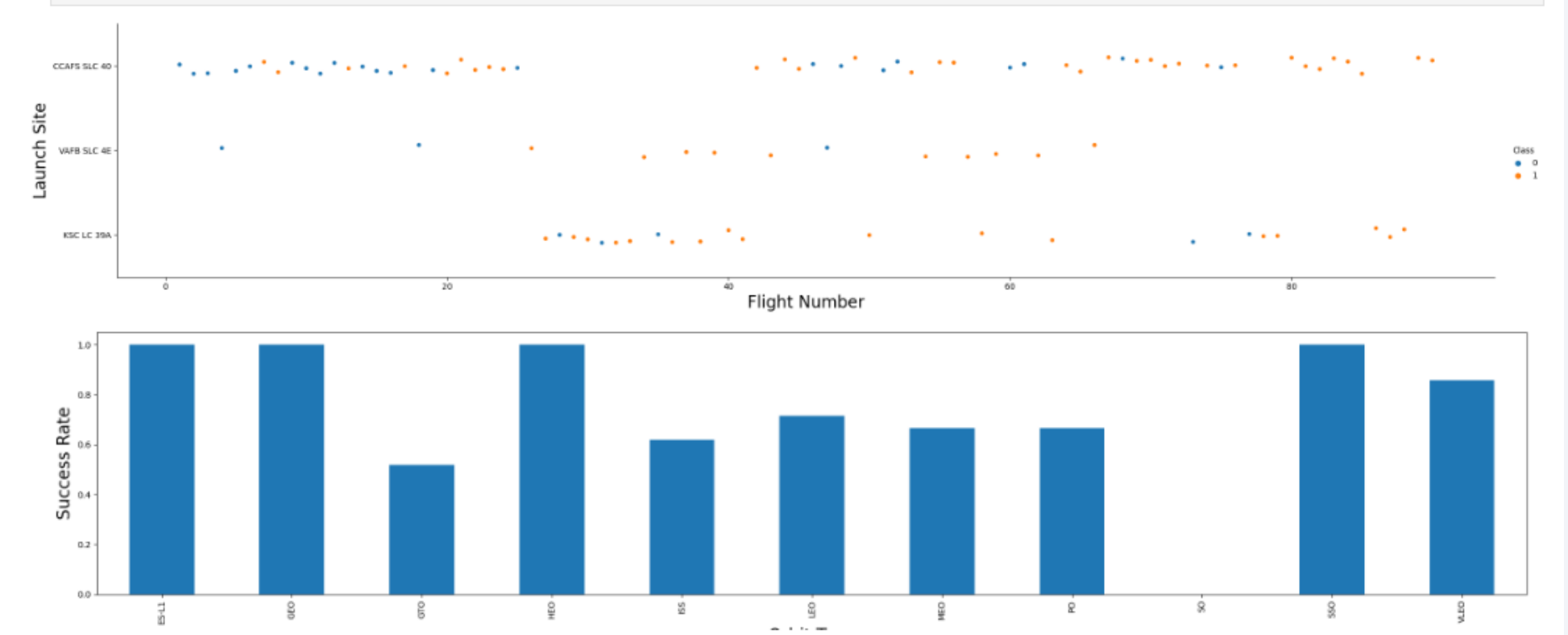
11]:
```
# Let's print the third table and check its content
first_launch_table = html_tables[2]
print(first_launch_table)
```

Data Wrangling



- Performed exploratory data analysis and determined the training labels.
- Calculated the number of launches at each site, and the number and occurrence of each orbits
- Created landing outcome label from outcome column
- Exported the results to csv.
- Reference: https://github.com/charles0624/IBM-Capstone-Project---SpaceX-Falcon-9-first-stage-Landing-Prediction/blob/main/Data%20Wrangling.ipynb

EDA with Data Visualization



Scatterplots and barchart were used to visualize the relationship between pair of features:

- Payload Mass X Flight Number
- Launch Site X Flight Number
- Launch Site X Payload Mass
- Orbit and Flight Number
- Payload and Orbit

Reference: IBM-Capstone-Project---SpaceX-Falcon-9-first-stage-Landing-Prediction/Data Visualization.ipynb at main · charles0624/IBM-Capstone-Project---SpaceX-Falcon-9-first-stage-Landing-Prediction · GitHub

EDA with SQL

Reference: https://github.com/charles0624/IBM-Capstone-Project---SpaceX-Falcon-9-first-stage-Landing-Prediction/blob/main/Data%20Visualization.ipynb

- The following SQL queries were performed:\
- Names of the unique launch sites in the space mission;
- Top 5 launch sites whose name begin with the string 'CCA';
- Total payload mass carried by boosters launched by NASA (CRS);
- Average payload mass carried by booster version F9 v1.1;
- Date when the first successful landing outcome in ground pad was achieved;
- Names of the boosters which have success in drone ship and have payload mass between 4000 and 6000 kg;
- Total number of successful and failure mission outcomes;
- Names of the booster versions which have carried the maximum payload mass;
- Failed landing outcomes in drone ship, their booster versions, and launch site names for in year 2015;
- Rank of the count of landing outcomes (such as Failure (drone ship) or Success (ground pad)) between the date 2010-06-04 and 2017-03-20.

Build an Interactive Map with Folium

- Marked all launch sites, and added map objects such as markers, circles, lines to mark the success or failure of launches for each site on the folium map.
- Assigned the feature launch outcomes (failure or success) to class 0 and 1.i.e., 0 for failure, and 1 for success.
- Using the color-labeled marker clusters, we identified which launch sites have relatively high success rate.
- alculated the distances between a launch site to its proximities. We answered some question for instance:
  - Are launch sites near railways, highways and coastlines.
  - Do launch sites keep certain distance away from cities.

Reference: IBM-Capstone-Project---SpaceX-Falcon-9-first-stage-Landing-Prediction/Folium Launch Site Location.ipynb at main · charles0624/IBM-Capstone-Project---SpaceX-Falcon-9-first-stage-Landing-Prediction · GitHub

Build a Dashboard with Plotly Dash

The following graphs and plots were used to visualize data

- Percentage of launches by site
- Payload range

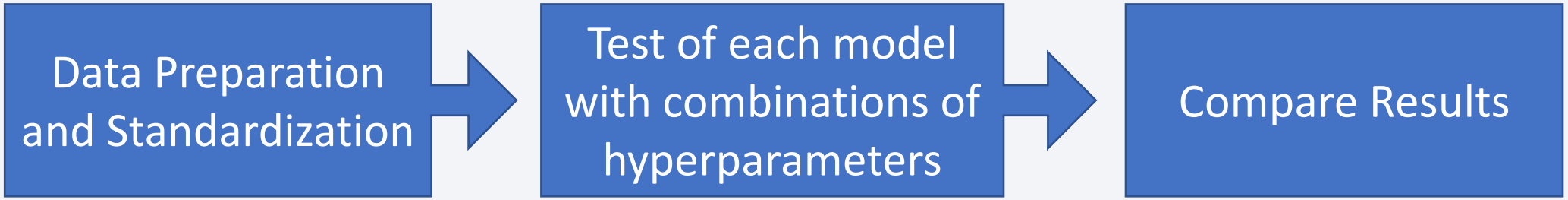
This combination allowed to quickly analyze the relation between payloads and launch sites, helping to identify where is best place to launch according to payloads.

- Reference: https://github.com/charles0624/IBM-Capstone-Project---SpaceX-Falcon-9-first-stage-Landing-Prediction/blob/main/Ploty.py

Predictive Analysis (Classification)

Four classification models were applied:

- logistic regression
- support vector machine
- decision tree and k nearest neighbors



- Reference: https://github.com/charles0624/IBM-Capstone-Project---SpaceX-Falcon-9-first-stage-Landing-Prediction/blob/main/ML%20Predict.ipynb

Results

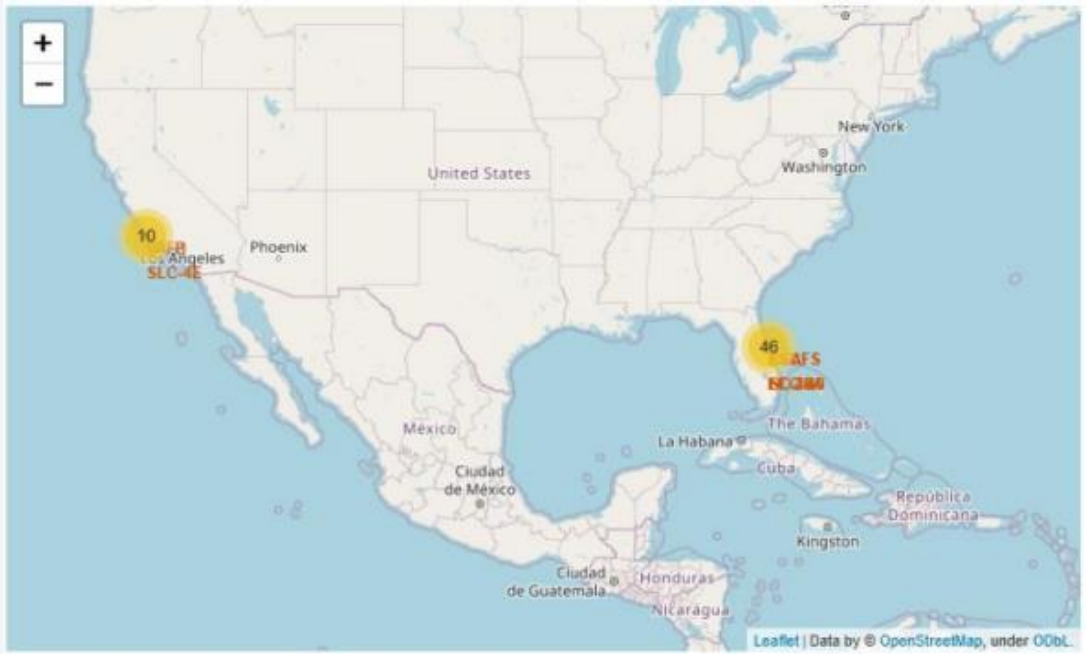
Exploratory data analysis results:

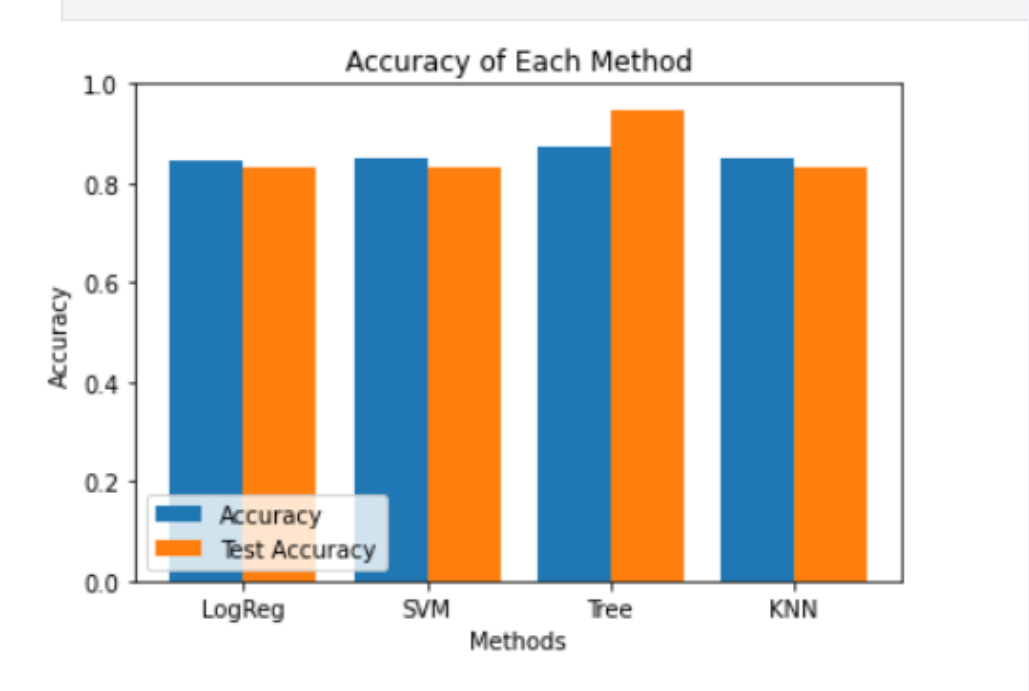
- Space X uses 4 different launch sites;
- The first launches were done to Space X itself and NASA;
- The average payload of F9 v1.1 booster is 2,928 kg;
- The first success landing outcome happened in 2015 fiver year after the first launch;
- Many Falcon 9 booster versions were successful at landing in drone ships having payload above the average;
- Almost 100% of mission outcomes were successful;
- Two booster versions failed at landing in drone ships in 2015: F9 v1.1 B1012 and F9 v1.1 B1015;
- The number of landing outcomes became as better as years passed

Results

1. Successful launches in East and West Coast of US, There are more launches happened in Eastern(count:46) than Western(count:10)
2. Predictive Analysis showed that Decision Tree Classifier is the best model to predict successful landings:

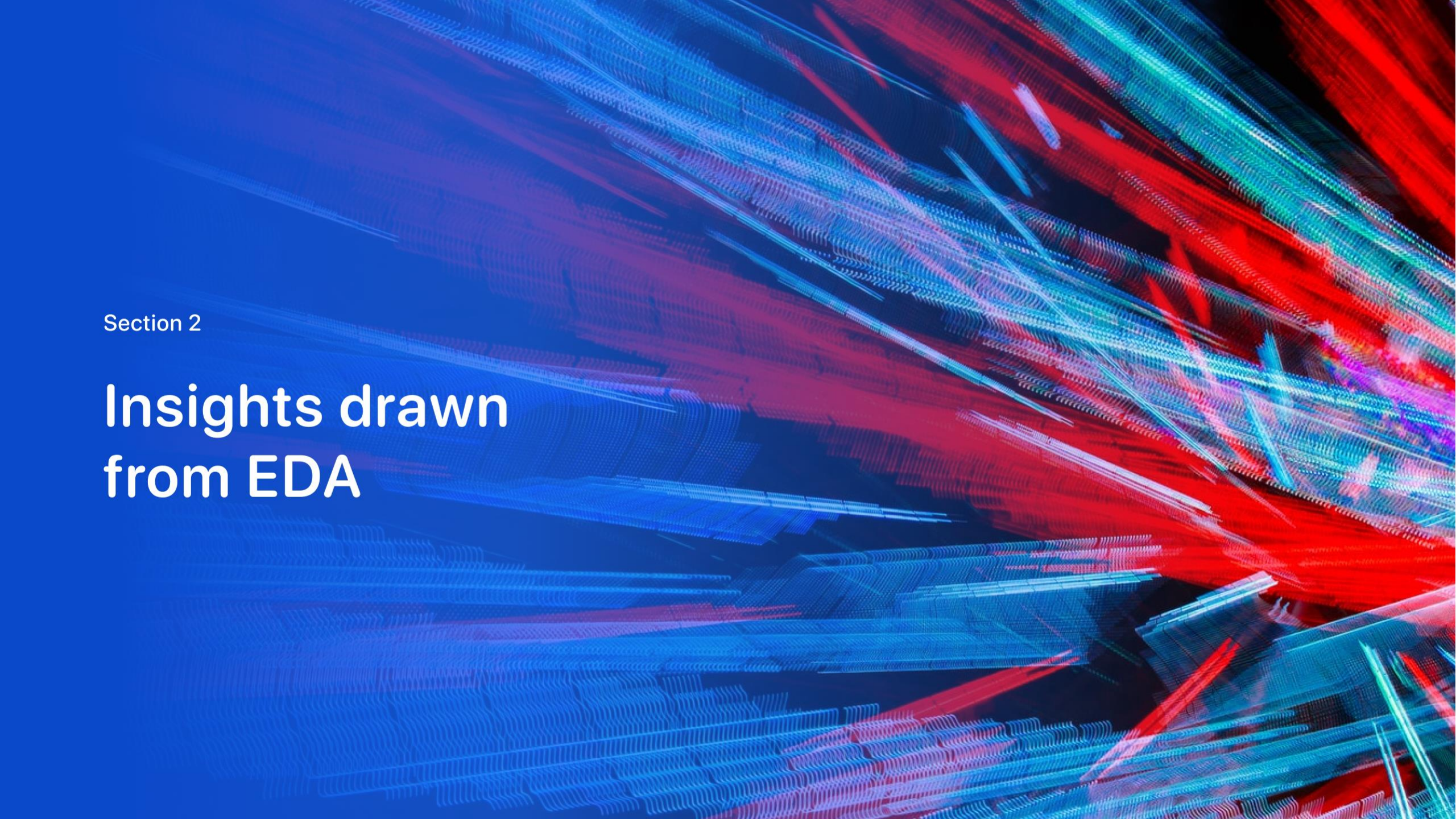
- having accuracy over 87%
- accuracy for test data over 94%





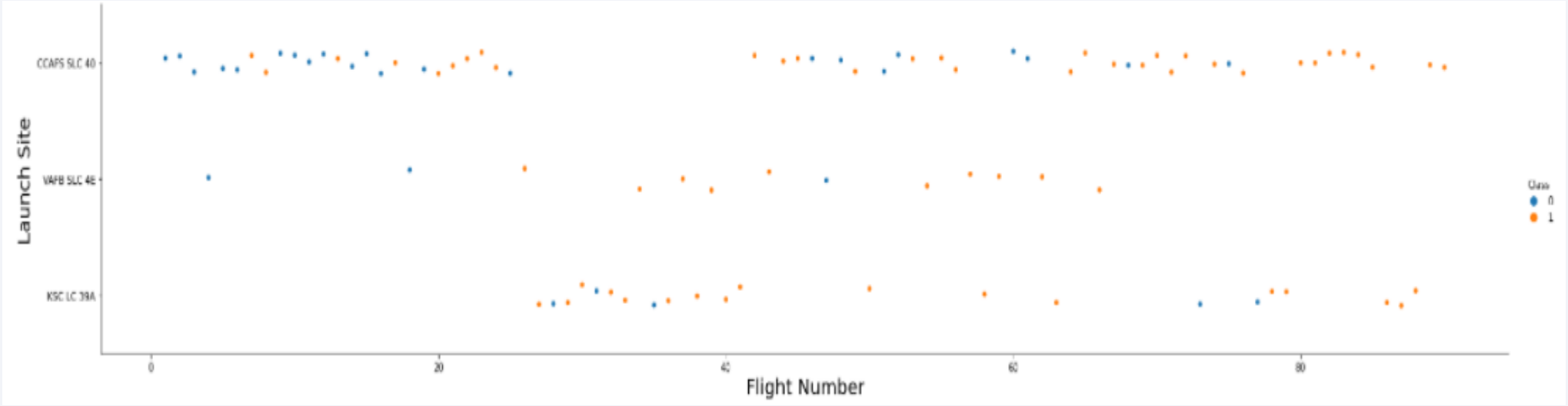
Section 2

Insights drawn from EDA



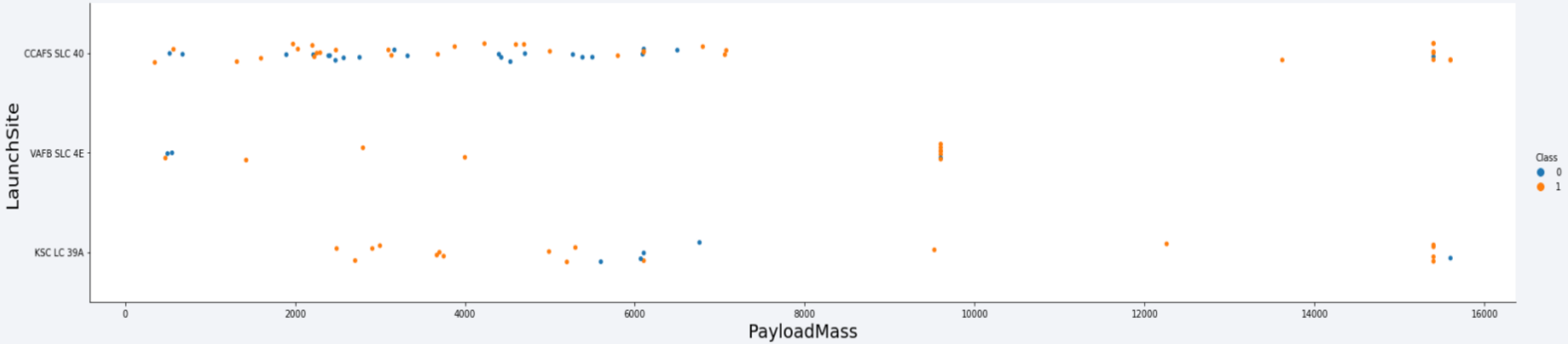
Flight Number vs. Launch Site

- According to the plot above, it's possible to verify that the best launch site nowadays is CCAF5 SLC 40, where most of recent launches were successful;
- In second place VAFB SLC 4E and third place KSC LC 39A;
- It's also possible to see that the general success rate improved over time.



Payload vs. Launch Site

- Payloads over 9,000kg have excellent success rate;
- Payloads over 12,000kg seems to be possible only on CCAFS SLC 40 and KSC LC 39A launch sites



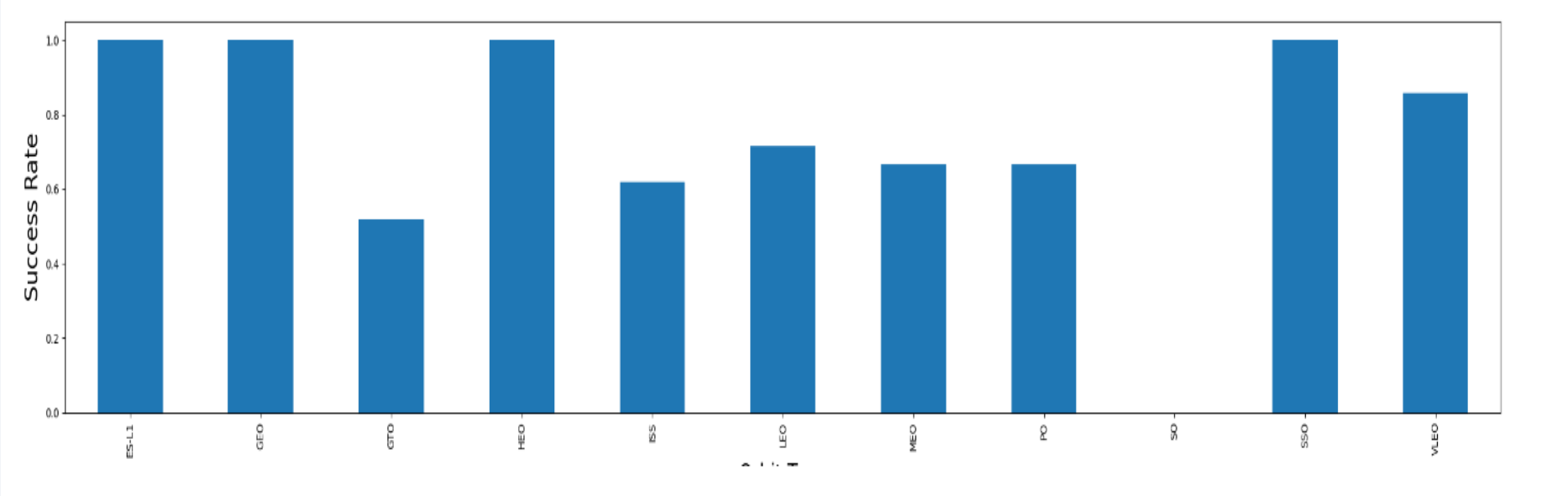
Success Rate vs. Orbit Type

The biggest success rates happens to orbits:

- ES-L1;
- GEO;
- HEO; and
- SSO.

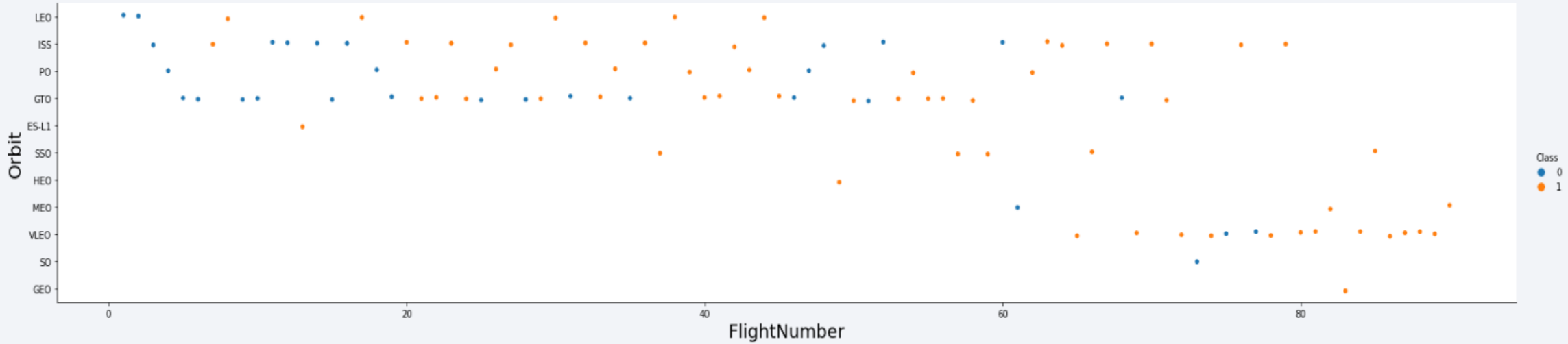
Followed by:

- VLEO (above 80%);
- LFO (above 70%).

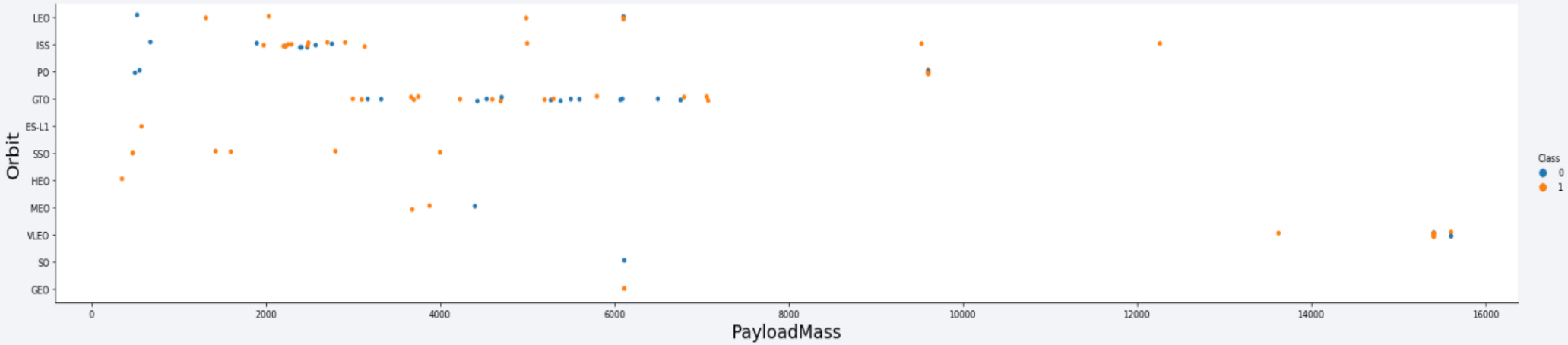


Flight Number vs. Orbit Type

- VLEO orbit seems a new business opportunity, due to recent increase of its frequency.



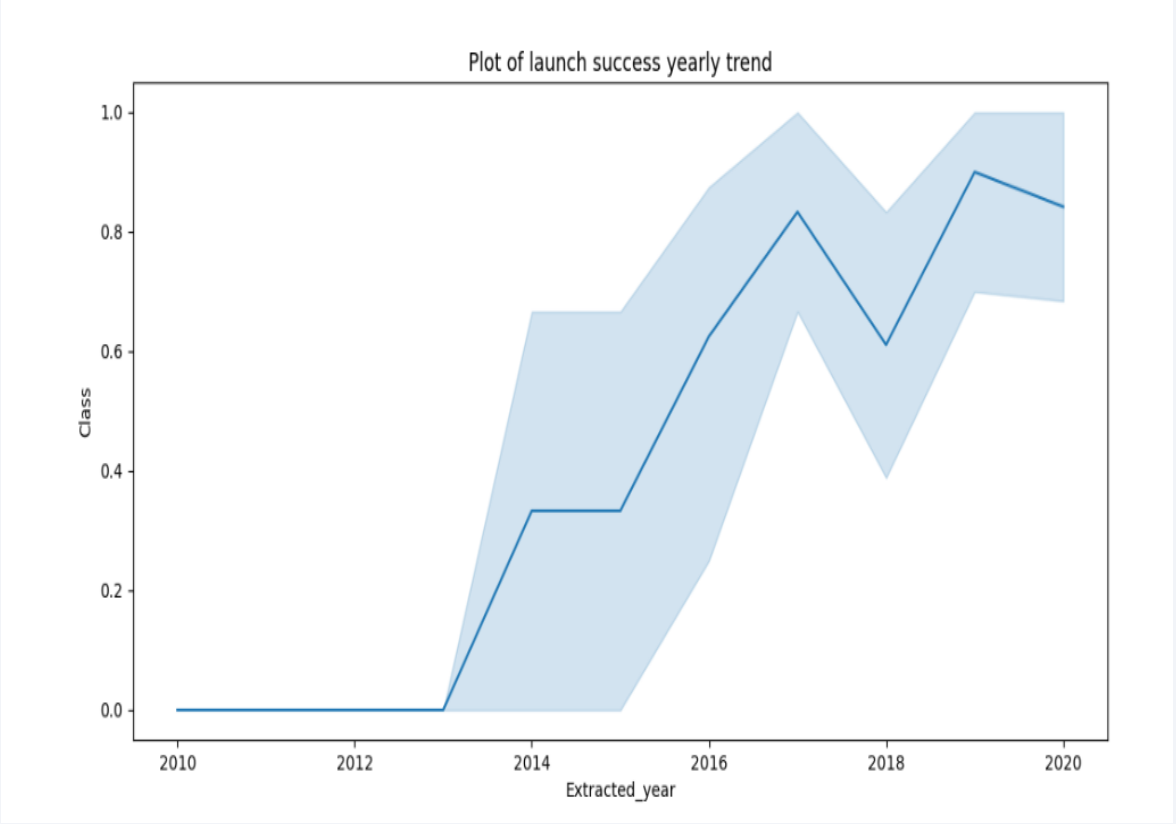
Payload vs. Orbit Type



- There is no relation between payload and success rate to orbit GTO;
- ISS orbit has the widest range of payload and a good rate of success;
- Few launches to the orbits SO and GEO.

Launch Success Yearly Trend

- Rate of successful launch are increasing from 2013 – 2020.
- Impliedly speaking, there are launch preparation from 2010-2013
- Even if the success rate dropped around 2017, the overall trend is still growing



All Launch Site Names

- Four launch sites presented:
- They are obtained by selecting unique occurrences of “launch_site” values from the dataset

	Launch Site
0	CCAFS LC-40
1	CCAFS SLC-40
2	KSC LC-39A
3	VAFB SLC-4E

Launch Site Names Begin with 'CCA'

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

Total Payload Mass

- Total payload carried by boosters from NASA:

Total payload calculated below, by summing all payloads whose codes contain ‘CRS’, which corresponds to NASA

total_payload
111268

Average Payload Mass by F9 v1.1

- Average payload mass carried by booster version F9 v1.1:

avg_payload
2928

First Successful Ground Landing Date

- First successful landing outcome on ground pad:
- By filtering data by successful landing outcome on ground pad and getting the minimum value for date it’s possible to identify the first occurrence, that happened on 12/22/2015

First Landing Date
2015-12-22

Successful Drone Ship Landing with Payload between 4000 and 6000

- Boosters which have successfully landed on drone ship and had payload mass greater than 4000 but less than 6000
- Selecting distinct booster versions according to the filters below, these 4 are the result

Booster Version
F9 FT B1021.2
F9 FT B1031.2
F9 FT B1022
F9 FT B1026

Total Number of Successful and Failure Mission Outcomes

- Number of successful and failure mission outcomes:
- Grouping mission outcomes and counting records for each group led us to the summary below.

Mission Outcome	Occurrences
Success	99
Success(payload status unclear)	1
Failure(in flight)	1

Boosters Carried Maximum Payload

- Boosters which have carried the maximum payload mass
- These are the boosters which have carried the maximum payload mass registered in the dataset

Booster Version(…)	Booster Version
F9 B4 B1048.4	F9 B4 B1051.4
F9 B4 B1048.5	F9 B4 B1051.6
F9 B4 B1049.4	F9 B4 B1056.4
F9 B4 B1049.5	F9 B4 B1058.3
F9 B4 B1049.7	F9 B4 B1060.2
F9 B4 B1051.3	F9 B4 B1060.3

2015 Launch Records

- Failed landing outcomes in drone ship, their booster versions, and launch site names for in year 2015
- The list below has the only two occurrences.

booster_version	launch_site
F9 v1.1 B1012	CCAFS LC-40
F9 v1.1 B1015	CCAFS LC-40

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

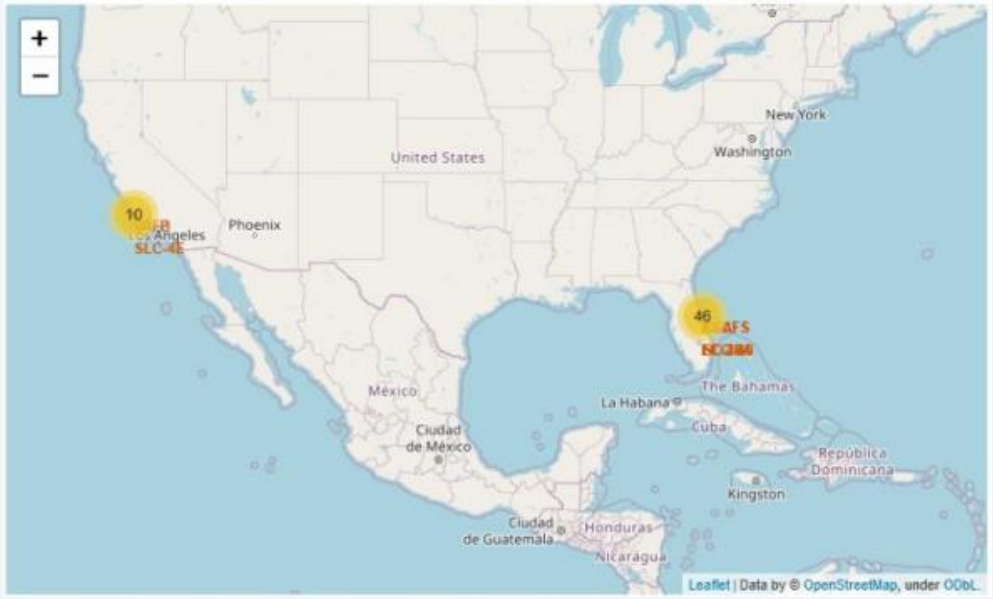
- Ranking of all landing outcomes between the date 2010-06-04 and 2017-03-20:
- This view of data alerts us that “No attempt” must be taken in account

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

Section 4

Launch Sites Proximities Analysis

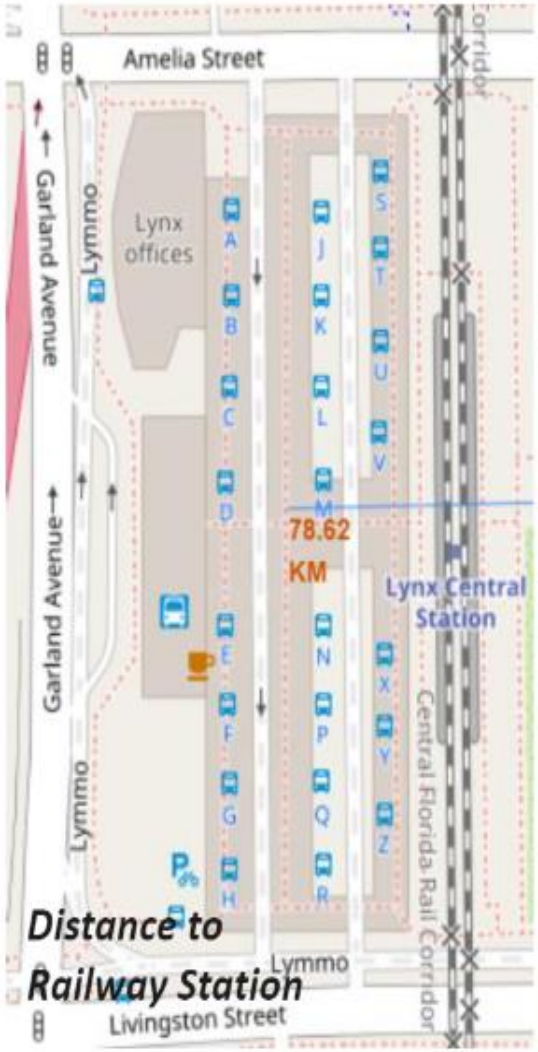
All launch sites global map markers



Successful launches in East and West Coast of US, There are more launches happened in Eastern than Western

Markers showing launch sites with color labels

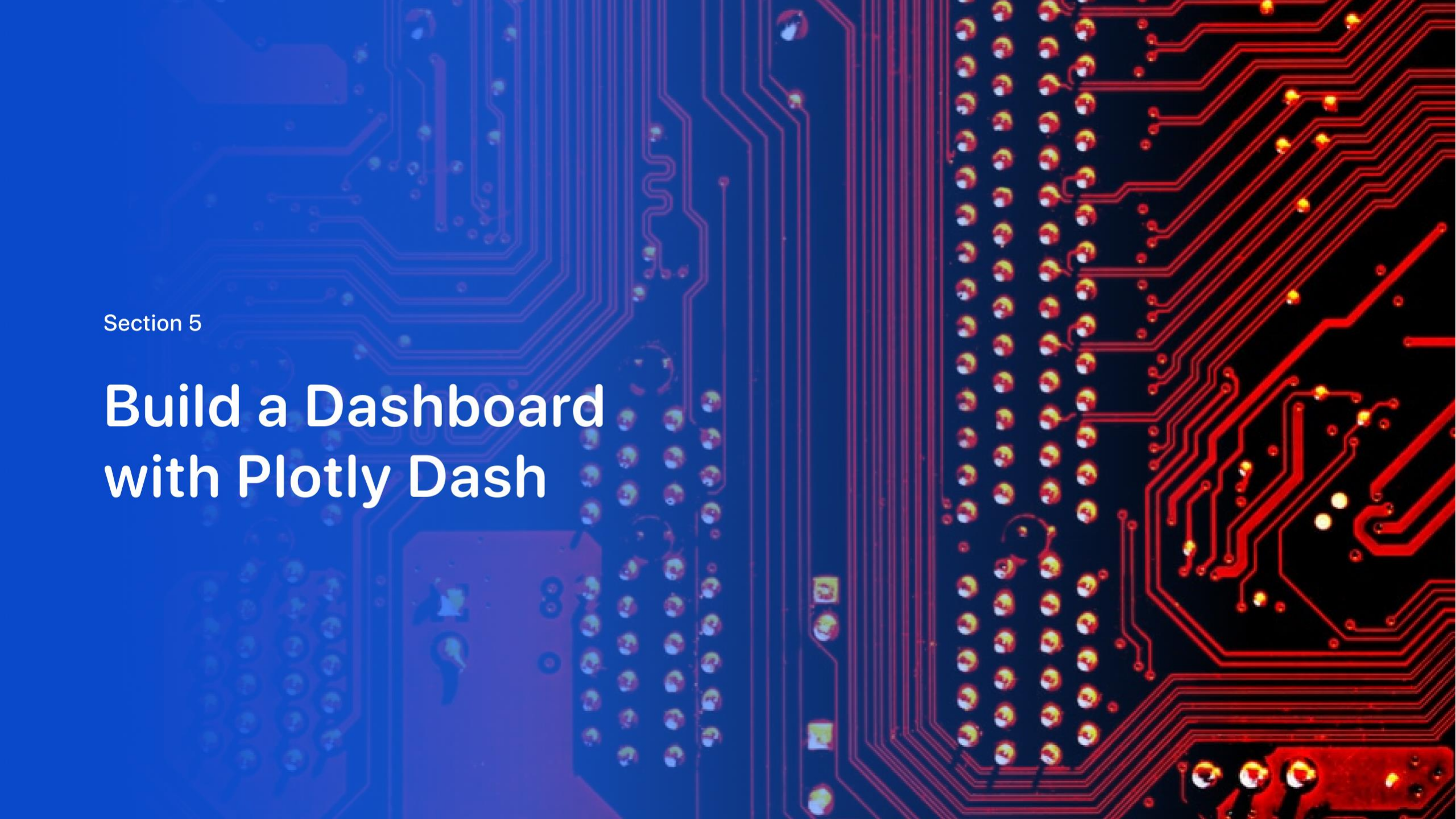
Launch Site distance to landmarks



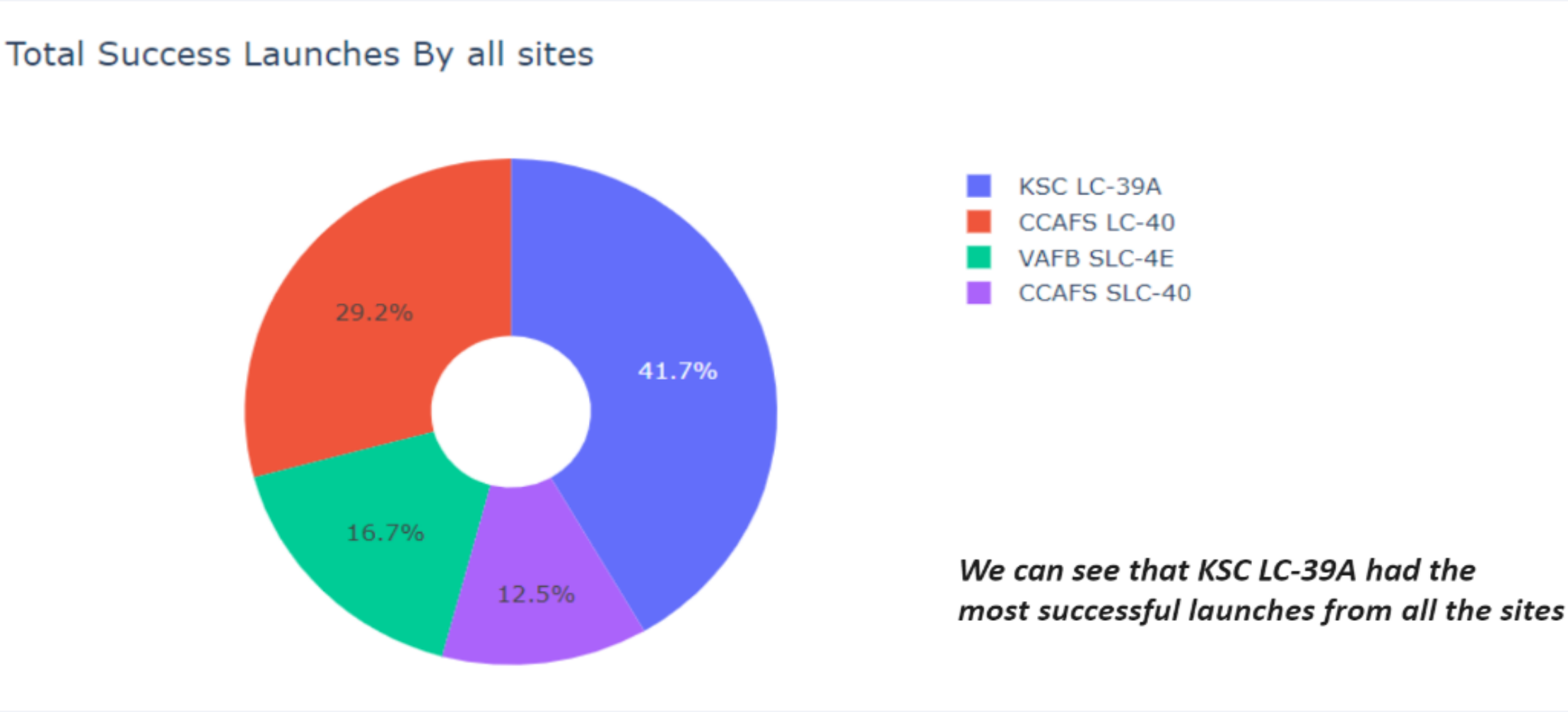
- Are launch sites in close proximity to railways? No
- Are launch sites in close proximity to highways? No
- Are launch sites in close proximity to coastline? Yes
- Do launch sites keep certain distance away from cities? Yes

Section 5

Build a Dashboard with Plotly Dash

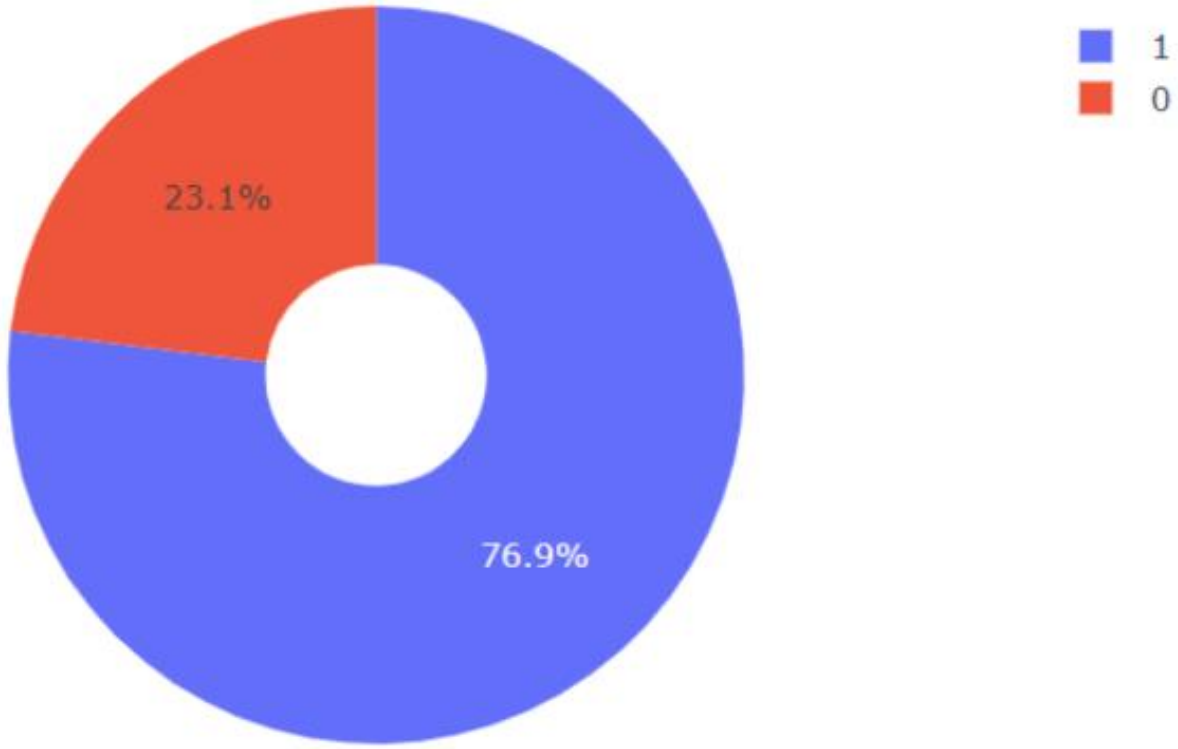


Pie chart showing the success percentage achieved by each launch site



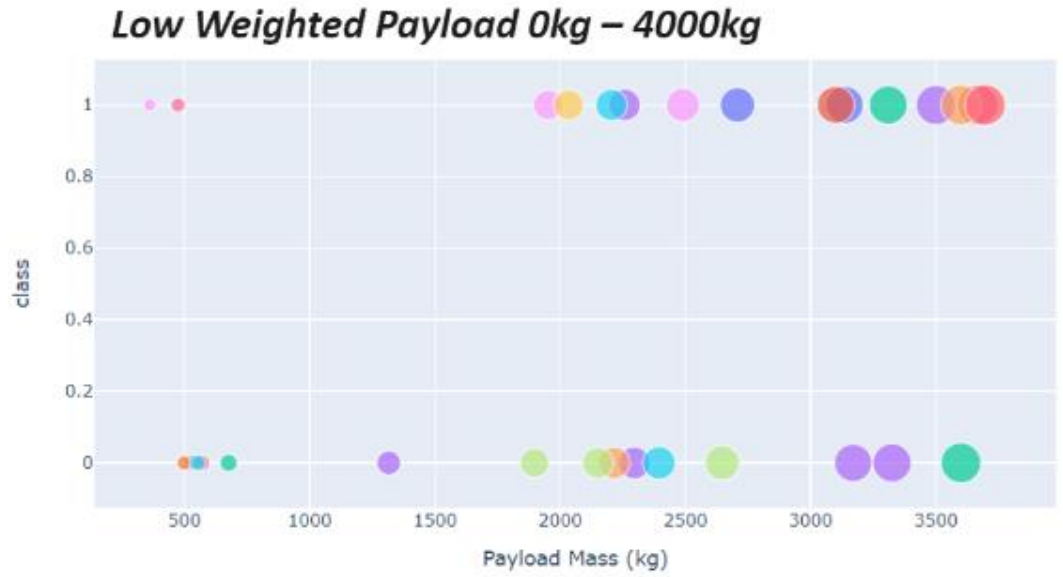
We can see that KSC LC-39A had the most successful launches from all the sites

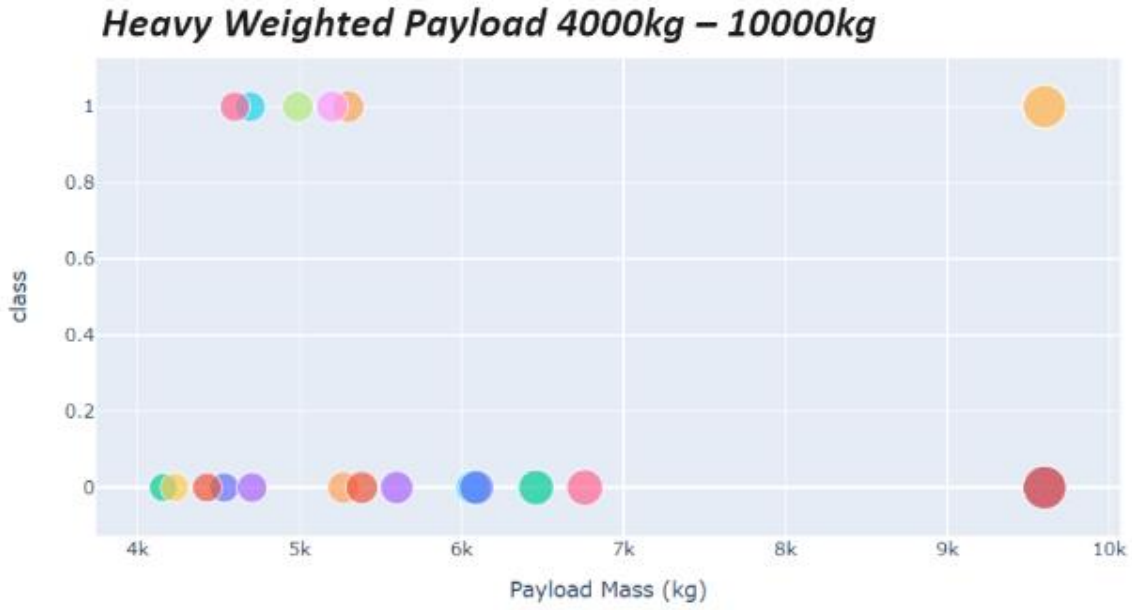
Pie chart showing the Launch site with the highest launch success ratio



KSC LC-39A achieved a 76.9% success rate while getting a 23.1% failure rate

Scatter plot of Payload vs Launch Outcome for all sites, with different payload selected in the range slider





We can see the success rates for low weighted payloads is higher than the heavy weighted payloads

Section 6

Predictive Analysis (Classification)

KNN Model

- Use KNN to find the best parameter
- The accuracy: 0.8482142857142858

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 [26]:
```
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 [27]:
```
knn_cv = GridSearchCV(estimator=KNN, cv=10, param_grid=parameters)
knn_cv.fit(X_train, Y_train)
```

Out[27]:
```
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]})
```

In [28]:
```
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
```

Logistic Regression Model Performance

- Use Logistic Regression to find the best parameter with the calculation presented in the image
- The accuracy: 0.8464285714285713

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 [11]:
```
parameters ={'C':[0.01,0.1,1],
             'penalty':['l2'],
             'solver':['lbfgs']}
```

In [12]:
```
parameters ={"C":[0.01,0.1,1],'penalty':['l2'], 'solver':['lbfgs']}# l1 lasso l2 ridge
lr=LogisticRegression()

logreg_cv = GridSearchCV(estimator=lr, cv=10, param_grid=parameters)
logreg_cv.fit(X_train, Y_train)
```

Out[12]:
```
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 [13]:
```
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
```

Decision Tree Model

- Use Logistic Regression to find the best parameter with the calculation presented in the image
- The accuracy: 0.8767857142857143

In [21]:
```
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()
```

In [22]:
```
tree_cv = GridSearchCV(estimator=tree, cv=10, param_grid=parameters)
tree_cv.fit(X_train, Y_train)
```

Out[22]:
```
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']})
```

In [23]:
```
print("tuned hpyerparameters :(best parameters) ",tree_cv.best_params_)
print("accuracy :",tree_cv.best_score_)
```

```
tuned hpyerparameters :(best parameters)  {'criterion': 'gini', 'max_depth': 4, 'max_features': 'auto', 'min_samples_lea
f': 1, 'min_samples_split': 2, 'splitter': 'best'}
accuracy : 0.8767857142857143
```

SVM Model

- Use SVM to find the best parameter with the calculation presented in the image
- The accuracy: 0.8482142857142856

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 [11]:
```
parameters ={'C':[0.01,0.1,1],
             'penalty':['l2'],
             'solver':['lbfgs']}
```

In [12]:
```
parameters ={"C":[0.01,0.1,1],'penalty':['l2'], 'solver':['lbfgs']}# l1 lasso l2 ridge
lr=LogisticRegression()

logreg_cv = GridSearchCV(estimator=lr, cv=10, param_grid=parameters)
logreg_cv.fit(X_train, Y_train)
```

Out[12]:
```
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 [13]:
```
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
```

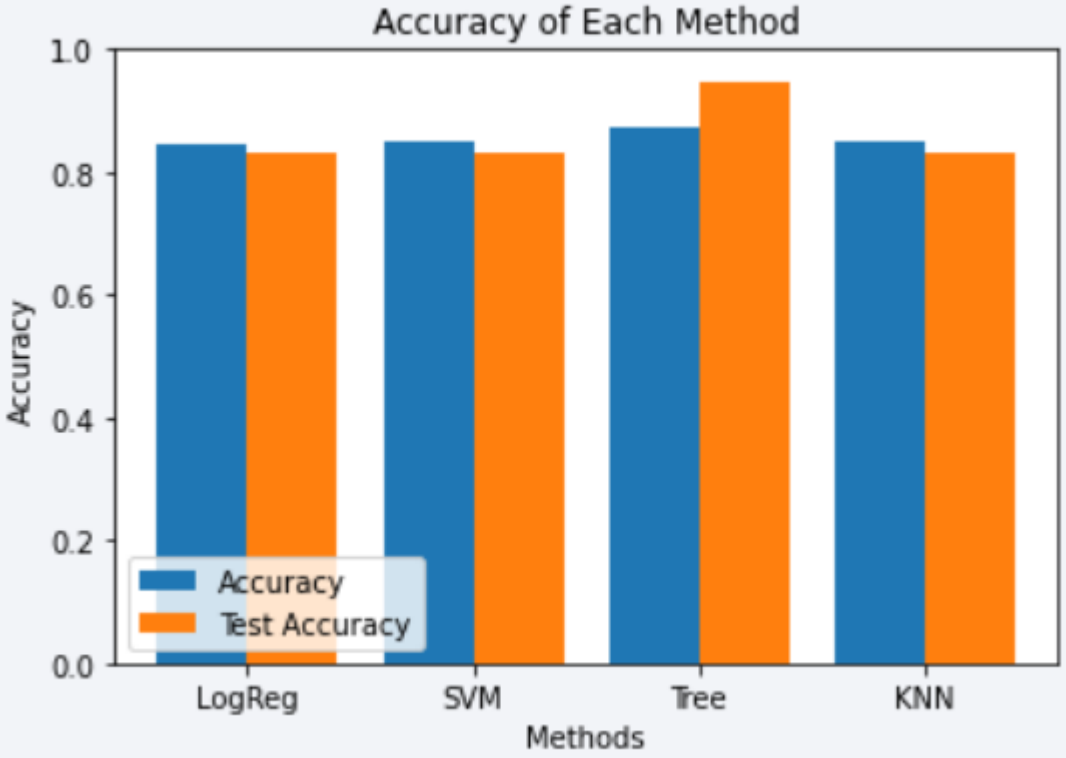
Models Comparation

Model	Accuracy	Test_Accuracy
KNN	0.84643	0.83333
Tree	0.84821	0.83333
SVM	0.87679	0.72222
LogReg	0.84821	0.83333

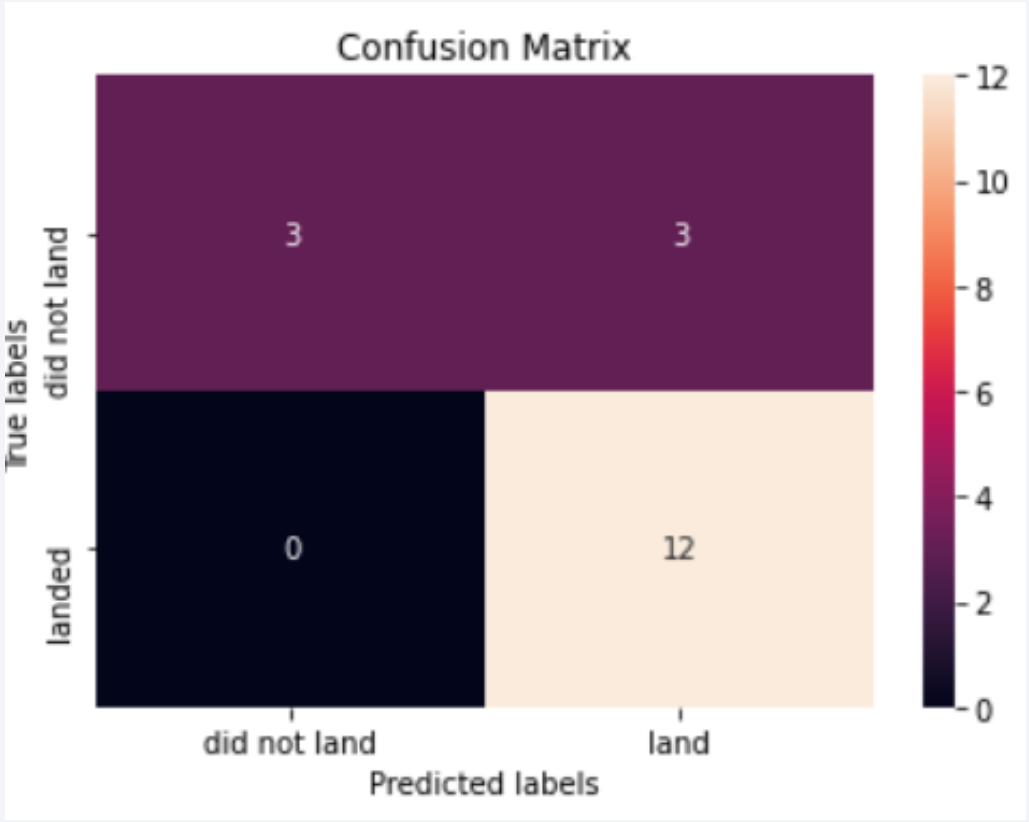
Models Comparation

Four classification models were tested, and their accuracies are plotted beside;

The model with the highest classification accuracy is **Decision Tree Classifier**, which has accuracies over than 87%



Confusion Matrix



Confusion matrix of **Decision Tree Classifier proves its accuracy** by showing the big numbers of true positive and true negative compared to the false ones

Conclusions

- The larger the flight amount at a launch site, the greater the success rate at a launch site.
- Launch has increasing success from 2013 till 2020.
- Orbits ES-L1, GEO, HEO, SSO, VLEO had the most success rate.
- KSC LC-39A had the most successful launches of any sites.
- The **Decision tree classifier** is the best machine learning algorithm for this task.

Thank you!