IBM Developer
SKILLS NETWORK

Winning Space Race with Data Science

David Michael Rook
21.07.24

Outline

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

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

- Data analysis results using SQL/Pandas/Matplotlib
- Interactive Analytics Dashboard
- Using ML for Predictive Analytics

Introduction

Project background and context:

SpaceX offers relatively inexpensive rocket launches due to the reuse of the Falcon 9's first stage, which costs $62 million compared to other providers' $165 million. Our task is to gather information on SpaceX and create dashboards to predict if the first stage will be reused, using machine learning and public data to estimate launch costs.

Problems you want to find answers to?

- 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

Executive Summary

- Data collection methodology:
  - Data collect web scraping results from Wiki and using SpaceX API
  - Perform data wrangling
  - Using Pandas and Numpy to remove missing values and to see the number of launches for each site.
- 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 collection was done using get request to the SpaceX API.
- Taken the dataset and uses the rocket column to call the API and append the data to the list to a .JSON file into a Pandas dataframe using the call function
- We then cleaned the data, checked for missing values and fill in missing values where necessary.
- In addition, we performed web scraping from Wikipedia for Falcon 9 launch records
- Finally; extract a Falcon 9 launch records HTML table from Wikipedia and parsed the table and convert it into a Pandas data frame

Data Collection – SpaceX API

- We used the get request to the SpaceX API to collect data, clean the requested data and did some basic data wrangling and formatting.

- https://github.com/Drook93/IBM-Data-Science-Capstone-SpaceX/blob/main/jupyter-labs-webscraping.ipynb

1. Get request for rocket launch data using API

In [6]:
```
spacex_url="https://api.spacexdata.com/v4/launches/past"
```

In [7]:
```
response = requests.get(spacex_url)
```

2. Use json_normalize method to convert json result to dataframe

In [12]:
```
# Use json_normalize method to convert the json result into a dataframe

# decode response content as json
static_json_df = res.json()
```

In [13]:
```
# apply json_normalize
data = pd.json_normalize(static_json_df)
```

3. We then performed data cleaning and filling in the missing values

In [30]:
```
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

- I applied web scrapping to webscrap Falcon 9 launch records with BeautifulSoup
- We parsed the table and converted it into a pandas dataframe.
- https://github.com/Drook93/IBM-Data-Science-Capstone-SpaceX/blob/main/jupyter-labs-webscraping.ipynb

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

```
static_url = "https://en.wikipedia.org/w/index.php?title=List_of_Falcon_9_and_Falcon_Heavy_launches&oldid=1027686922"

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.text, 'html.parser')
```

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

```
print(soup.title)
```

```
<title>List of Falcon 9 and Falcon Heavy launches - Wikipedia</title>
```

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

```
html_tables = soup.find_all("table")
print(html_tables)
```

```
[<table class="col-basic" role="presentation">
```

Next, we just need to iterate through the `<th>` elements and apply the provided `extract_column_from_header()` to extract column name one by one

```
column_names = []

temp = soup.find_all('th')
for x in range(len(temp)):
    try:
     name = extract_column_from_header(temp[x])
     if (name is not None and len(name) > 0):
        column_names.append(name)
    except:
     pass
```

Check the extracted column names

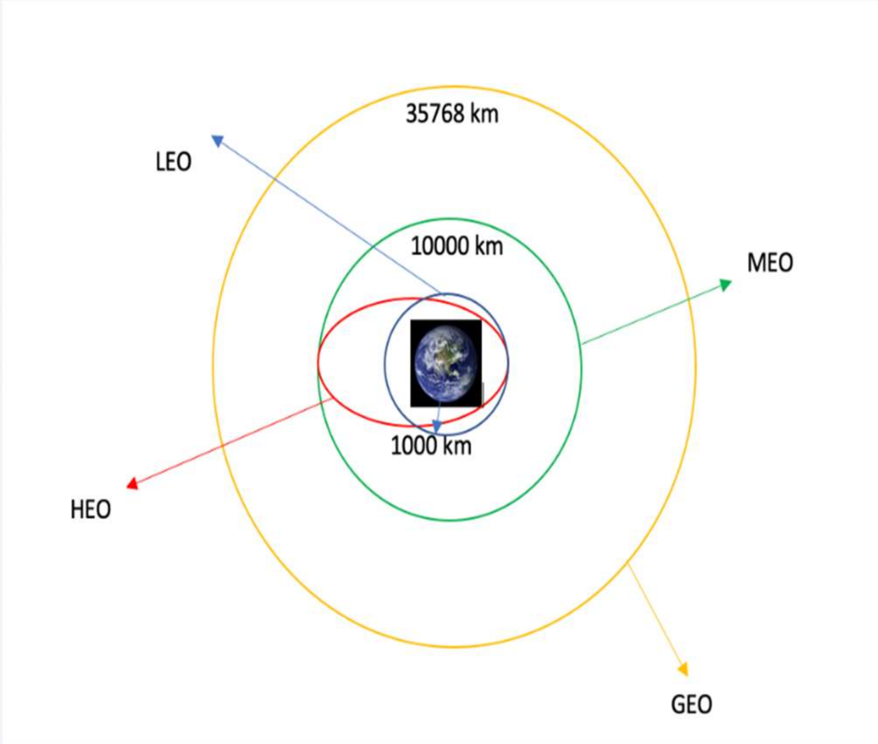
```
print(column_names)
```

```
['Flight No.', 'Date and time ( )', 'Launch site', 'Payload', 'Payload mass', 'Orbit', 'Customer', 'Launch outcome', 'Flight No.', 'Date and time ( )', 'Launch site', 'Payload', 'Payload mass', 'Orbit', 'Customer', 'Launch outcome', 'Flight No.',
'Date and time ( )', 'Launch site', 'Payload', 'Payload mass', 'Orbit', 'Customer', 'Launch outcome', 'Flight No.', 'Date and time ( )', 'Launch site', 'Payload', 'Payload mass', 'Orbit', 'Customer', 'Launch outcome', 'N/A', 'Flight No.', 'Date an
time ( )', 'Launch site', 'Payload', 'Payload mass', 'Orbit', 'Customer', 'Launch outcome', 'Flight No.', 'Date and time ( )', 'Launch site', 'Payload', 'Payload mass', 'Orbit', 'Customer', 'Launch outcome', 'Flight No.', 'Date and time ( )', 'Lau
ch site', 'Payload', 'Payload mass', 'Orbit', 'Customer', 'Launch outcome', 'FH 2', 'FH 3', 'Flight No.', 'Date and time ( )', 'Launch site', 'Payload', 'Payload mass', 'Orbit', 'Customer', 'Launch outcome', 'Date and time ( )', 'Launch site', 'Pa
load', 'Payload mass', 'Orbit', 'Customer', 'Launch outcome', 'Date and time ( )', 'Launch site', 'Payload', 'Orbit', 'Customer', 'Date and time ( )', 'Launch site', 'Payload', 'Orbit', 'Customer', 'Date and time ( )', 'Launch site', 'Payload', 'O
bit', 'Customer', 'Date and time ( )', 'Launch site', 'Payload', 'Orbit', 'Customer', 'Demonstration', 'logistics', 'Crewed', 'Commercial satellites', 'Scientific satellites', 'Military satellites', 'Rideshares', 'Transporter', 'Bandwagon', 'Curre
t', 'In development', 'Retired', 'Cancelled', 'Spacecraft', 'Cargo', 'Crewed', 'Test vehicles', 'Current', 'Retired', 'Unflown', 'Orbital', 'Atmospheric', 'Landing sites', 'Other facilities', 'Support', 'Contracts', 'R&D programs', 'Key people', '
elated', 'General', 'General', 'People', 'Vehicles', 'Launches by rocket type', 'Launches by spaceport', 'Agencies, companies and facilities', 'Other mission lists and timelines']
```

Data Wrangling

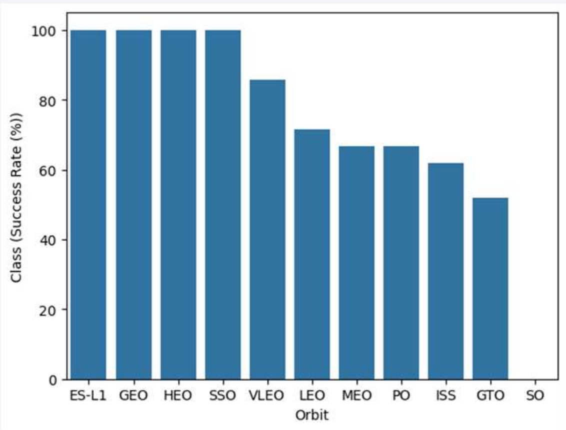
- We performed exploratory data analysis and determined the training labels.
- We calculated the number of launches at each site, and the number and occurrence of each orbits
- We created landing outcome label from outcome column and exported the results to csv.

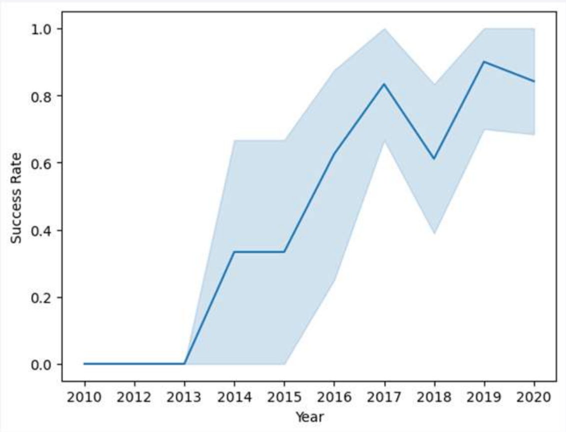
- https://github.com/Drook93/IBM-Data-Science-Capstone-SpaceX/blob/main/labs-jupyter-spacex-Data%20wrangling.ipynb



EDA with Data Visualization

- We explored the data by visualizing the relationship between flight number and launch Site, payload and launch site, success rate of each orbit type, flight number and orbit type, the launch success yearly trend.





- https://github.com/Drook93/IBM-Data-Science-Capstone-SpaceX/blob/main/EDA%20with%20Data%20Visulisation.ipynb

EDA with SQL

- We loaded the SpaceX dataset into a SQL database without leaving the jupyter notebook.
- We applied EDA with SQL to get insight from the data. We wrote queries to find out for instance:
  - The names of unique launch sites in the space mission.
  - The total payload mass carried by boosters launched by NASA (CRS)
  - The average payload mass carried by booster version F9 v1.1
  - The total number of successful and failure mission outcomes
  - The failed landing outcomes in drone ship, their booster version and launch site names.
- https://github.com/Drook93/IBM-Data-Science-Capstone-SpaceX/blob/main/jupyter-labs-eda-sql-coursera_sqllite.ipynb

Build an Interactive Map with Folium

- We 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.
- We 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.
- We calculated the distances between a launch site to its proximities. We asked questions such as:
- **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 a certain distance away from cities? YES**
- By asking these questions and building an interactive map with coordinates I was able to determine the best proximity locations for the rocket launch.
- https://github.com/Drook93/IBM-Data-Science-Capstone-SpaceX/blob/main/Locations%20Analysis%20with%20Folium.ipynb

Predictive Analysis (Classification)

- Summarize how you built, evaluated, improved, and found the best performing classification model
- I loaded the data using numpy and pandas, transformed the data, split our data into training and testing.
- I built different machine learning models and tuned different hyperparameters using GridSearchCV along with using other methods such as SVM, Classification Trees and Logistic Regression.
- I tested the accuracy as the metric for the model across various algorithms and improved the model using feature engineering.
- We found the best performing classification model with was GrideSearchCV with a marginaly better accuracy.

https://github.com/Drook93/IBM-Data-Science-Capstone-SpaceX/blob/main/SpaceX_Machine%20Learning%20Prediction_Part_5.ipynb

Results

- Exploratory data analysis results
- Interactive analytics demo in screenshots
- Predictive analysis results

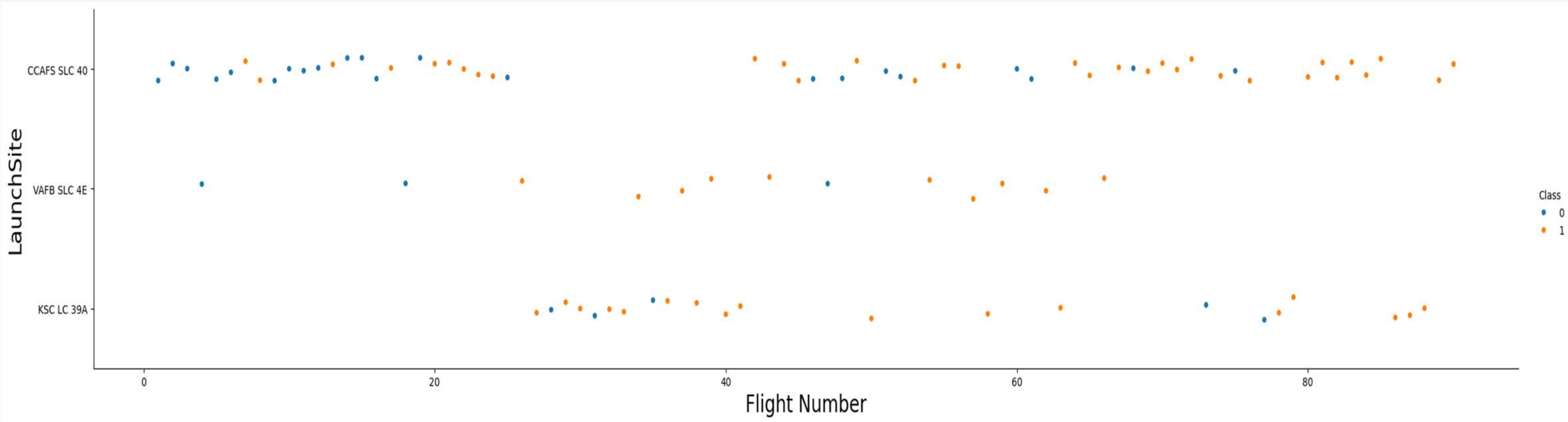
Section 2

Insights drawn from EDA

Flight Number vs. Launch Site

Plot of Flight Number vs. Launch Site

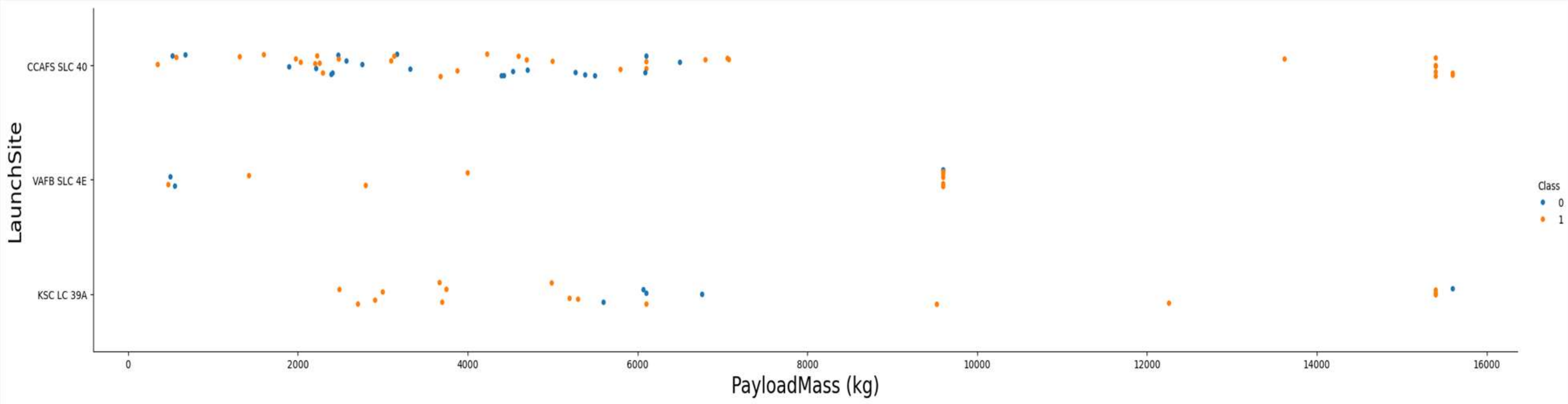
- From the plot, we found that the larger the flight amount at a launch site, the greater the success rate at a launch site for the rockets to land.



Payload vs. Launch Site

Plot of Payload vs. Launch Site

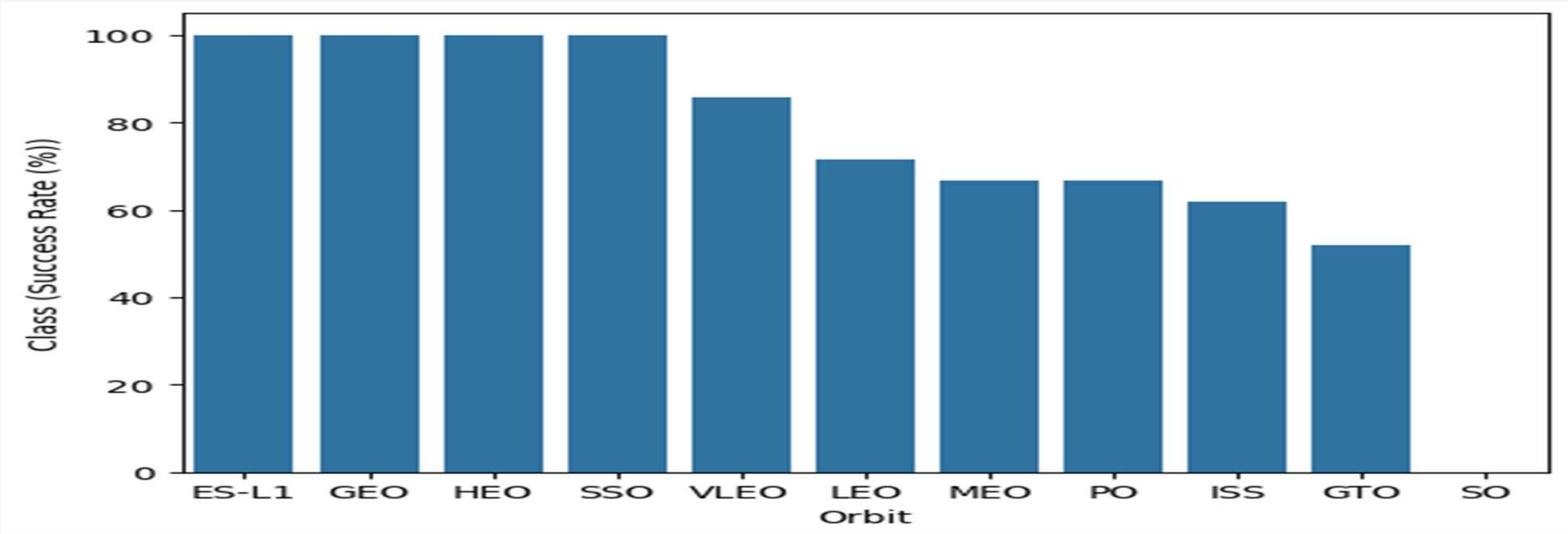
Found that below 7000 (kg) has an overall increased amount of a successful land.



Success Rate vs. Orbit Type

Bar chart for the success rate of each orbit type

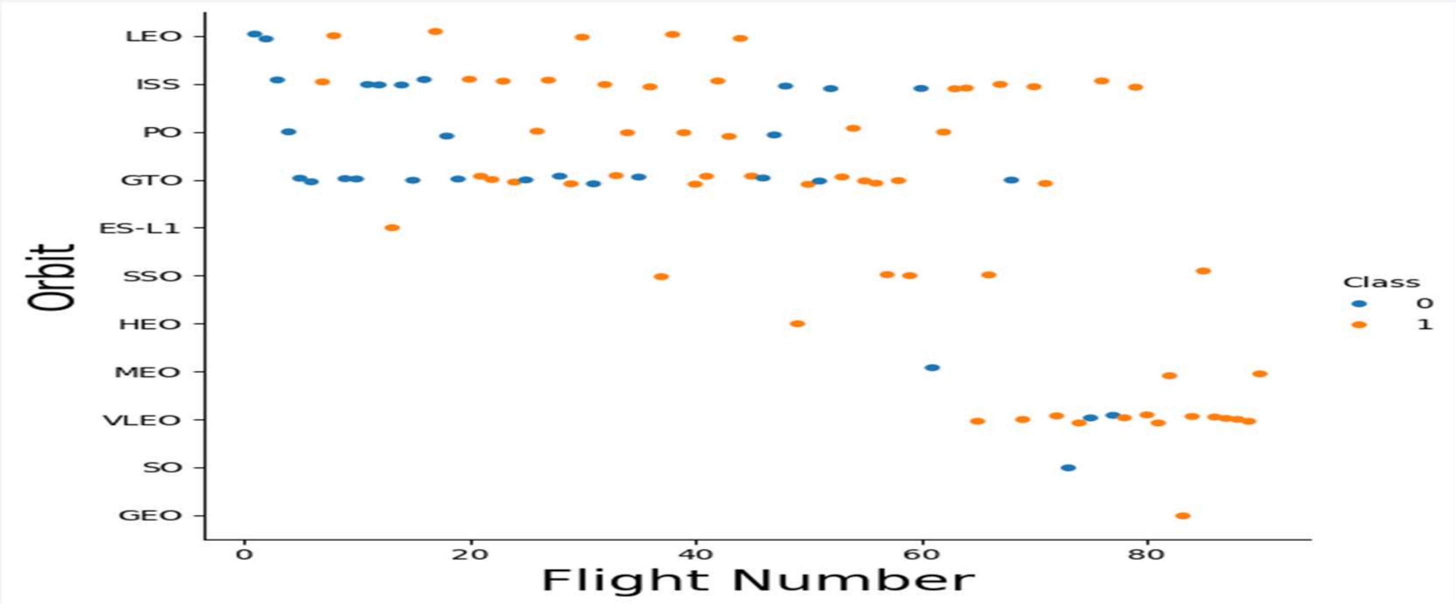
- From the bar chat you can see that the highest success rates where at the site locations ES-L1, GEO, HEO and SSO.



Flight Number vs. Orbit Type

Scatter point of Flight number vs. Orbit type

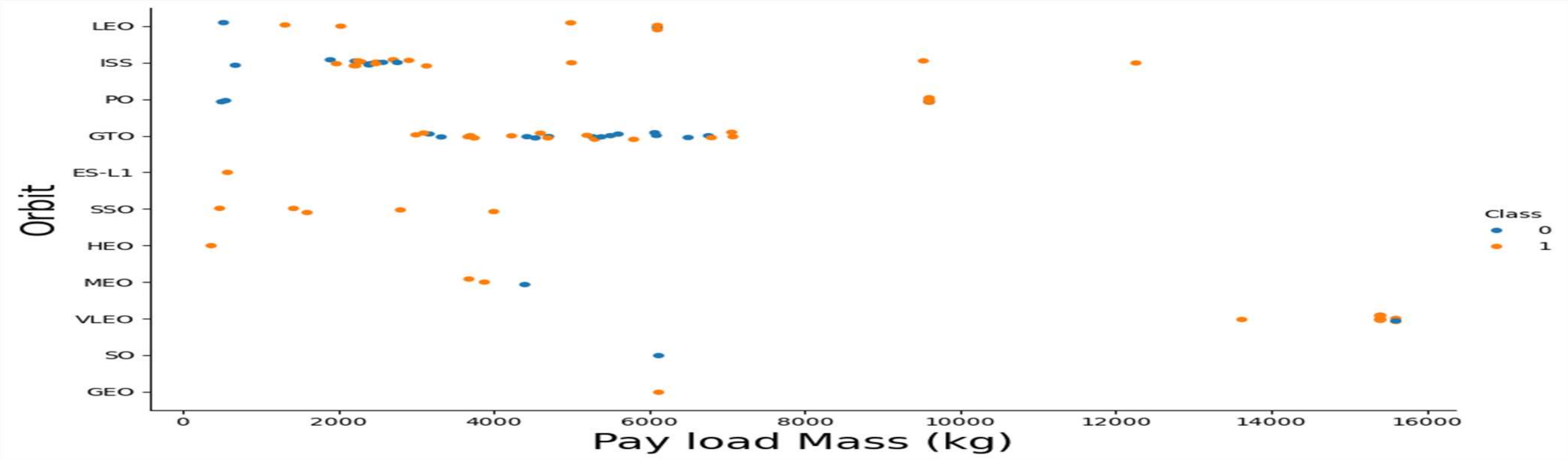
- The plot below shows the Flight Number vs. Orbit type. With the LEO orbit, success is related to the increased number of flights, whereas in the GTO orbit there is no relationship between flight number and the orbit.



Payload vs. Orbit Type

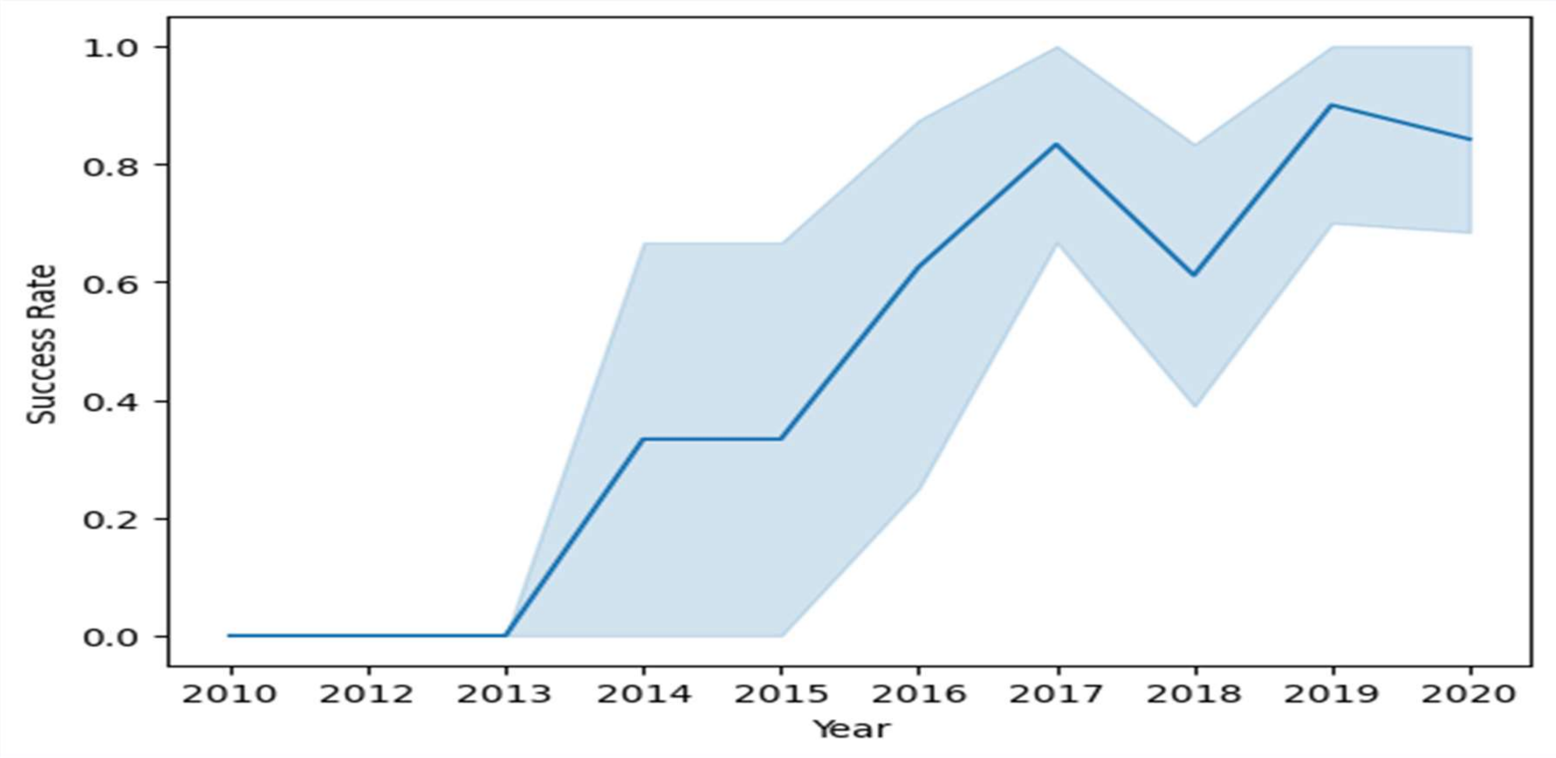
Scatter point of payload vs. orbit type

I can observe that with heavy payloads, the successful landing are more for PO, LEO and ISS orbits.



Launch Success Yearly Trend

- From the plot, we can observe that success rate since 2013 kept on increasing till 2020.



All Launch Site Names

- I used the key word DISTINCT to show only unique launch sites from the SpaceX data.

Task 1

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

In [39]:

```
%sql select DISTINCT "Launch_Site" FROM SPACEXTBL
```

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

Out[39]:

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

Launch Site Names Begin with 'CCA'

Task 2

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

In [41]:
```
%sql SELECT * FROM SPACEXTBL WHERE LAUNCH_SITE LIKE 'CCA%' LIMIT 5;
```

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

Out[41]:

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

- I calculated the total payload carried by boosters from NASA CRS using the query below

Task 3

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

In [42]:
```
%sql SELECT SUM(PAYLOAD_MASS__KG_) from SPACEXTBL WHERE Customer = 'NASA (CRS)';
```

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

Out[42]:

SUM(PAYLOAD_MASS__KG_)
45596

Average Payload Mass by F9 v1.1

- I calculated the average payload mass carried by booster version F9 v1.1 as 2928.4

Task 4

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

In [43]:
```
%sql SELECT AVG(PAYLOAD_MASS__KG_) from SPACEXTBL WHERE Booster_Version = 'F9 v1.1';
```

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

Out[43]:

AVG(PAYLOAD_MASS__KG_)
2928.4

First Successful Ground Landing Date

- Find the dates of the first successful landing outcome on ground pad
- We observed that the dates of the first successful landing outcome on ground pad was 22nd December 2015

Task 5

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

Hint:Use min function

In [44]:
```
%sql SELECT MIN(DATE) FROM SPACEXTBL WHERE Landing_Outcome = 'Success (ground pad)'
```

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

Out[44]:

MIN(DATE)
2015-12-22

Successful Drone Ship Landing with Payload between 4000 and 6000

- The WHERE clause was used to filter for boosters which have successfully landed on drone ship and applied the AND condition to determine successful landing with payload mass greater than 4000 but less than 6000

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 [45]:
```
%sql SELECT * FROM SPACEXTBL WHERE PAYLOAD_MASS__KG_ BETWEEN 4000 AND 6000 AND Landing_Outcome = 'Success (drone ship)';
```

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

Out[45]:

Date	Time (UTC)	Booster_Version	Launch_Site	Payload	PAYLOAD_MASS__KG_	Orbit	Customer	Mission_Outcome	Landing_Outcome
2016-05-06	5:21:00	F9 FT B1022	CCAFS LC-40	JCSAT-14	4696	GTO	SKY Perfect JSAT Group	Success	Success (drone ship)
2016-08-14	5:26:00	F9 FT B1026	CCAFS LC-40	JCSAT-16	4600	GTO	SKY Perfect JSAT Group	Success	Success (drone ship)
2017-03-30	22:27:00	F9 FT B1021.2	KSC LC-39A	SES-10	5300	GTO	SES	Success	Success (drone ship)
2017-10-11	22:53:00	F9 FT B1031.2	KSC LC-39A	SES-11 / EchoStar 105	5200	GTO	SES EchoStar	Success	Success (drone ship)

Total Number of Successful and Failure Mission Outcomes

- Used wildcard like '%' to filter for WHERE Mission Outcome was a success or a failure.

Task 7

List the total number of successful and failure mission outcomes

In [48]:
```
%sql SELECT Mission_Outcome, COUNT(*) as Total_Number FROM SPACEXTBL GROUP BY MISSION_OUTCOME;
```

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

Out[48]:

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

Boosters Carried Maximum Payload

- I determined the booster that have carried the maximum payload using a subquery in the **WHERE** clause and the **MAX()** function.

Task 8

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

In [52]:
```
%sql SELECT Booster_Version FROM SPACEXTBL WHERE PAYLOAD_MASS__KG_ = (SELECT MAX(PAYLOAD_MASS__KG_) FROM SPACEXTBL);
```

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

Out[52]:

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

2015 Launch Records

- I used a combination of the CASE clause, WHEN, FROM, and WHERE conditions to filter for failed landing outcomes in drone ship, their booster versions, and launch site names for year 2015 and assigned the numbers to the relevant months and specified in the subtr CASE.

Task 9

List the records which will display the month names, failure landing_outcomes in drone ship ,booster versions, launch_site for the months in year 2015.

Note: SQLLite does not support monthnames. So you need to use substr(Date, 6,2) as month to get the months and substr(Date,0,5)='2015' for year.

[55]:

```
%%sql
SELECT CASE substr(DATE, 6, 2)
            WHEN '01' THEN 'January'
            WHEN '02' THEN 'February'
            WHEN '03' THEN 'March'
            WHEN '04' THEN 'April'
            WHEN '05' THEN 'May'
            WHEN '06' THEN 'June'
            WHEN '07' THEN 'July'
            WHEN '08' THEN 'August'
            WHEN '09' THEN 'September'
            WHEN '10' THEN 'October'
            WHEN '11' THEN 'November'
            WHEN '12' THEN 'December'
       END AS month,
       Booster_Version,
       Launch_Site,
       Landing_Outcome
FROM SPACEXTBL
WHERE Landing_Outcome = 'Failure (drone ship)'
  AND substr(DATE, 1, 4) = '2015';
```

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

[55]:

month	Booster Version	Launch Site	Landing Outcome
January	F9 v1.1 B1012	CCAFS LC-40	Failure (drone ship)
April	F9 v1.1 B1015	CCAFS LC-40	Failure (drone ship)

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

- I selected Landing outcomes and the COUNT of landing outcomes from the data and used the WHERE clause to filter for landing outcomes BETWEEN 2010-06-04 to 2010-03-20.
- We applied the GROUP BY clause to group the landing outcomes and the ORDER BY clause to order the grouped landing outcome in descending order.

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 [70]:

```
%%sql
SELECT LANDING__OUTCOME, COUNT(LANDING__OUTCOME) AS TOTAL_NUMBER
FROM SPACEXTBL
WHERE DATE BETWEEN '2010-06-04' AND '2017-03-20'
GROUP BY LANDING__OUTCOME
ORDER BY TOTAL_NUMBER DESC
```

```
 * ibm_db_sa://xcg80731:***@ba99a9e6-d59e-4883-8fc0-d6a8c9f7a08f.c1ogj3sd0tgtu0lqde00.databases.appdomain.cloud:31321/bludb
Done.
```

Out[70]:

landing__outcome	total_number
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

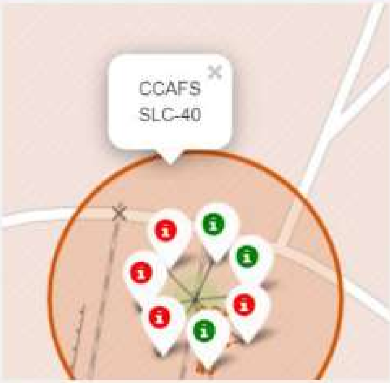
Section 3

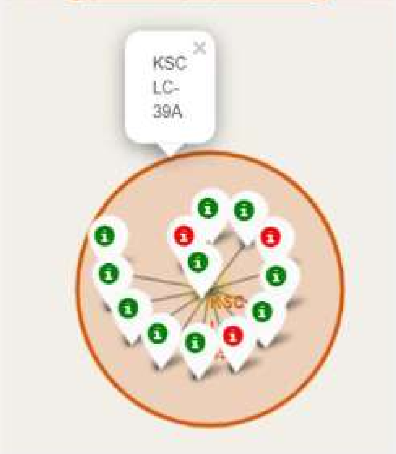
Launch Sites Proximities Analysis

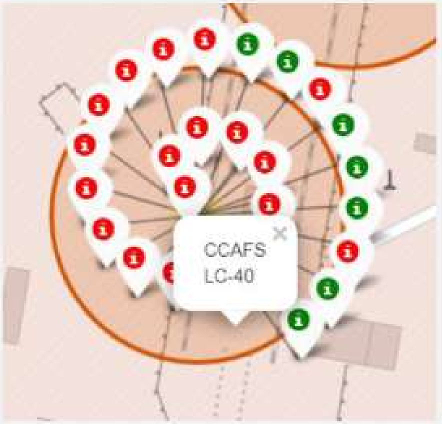
<Folium Map Screenshot 1>

All launch sites global map markers

Markers showing launch sites with color labels







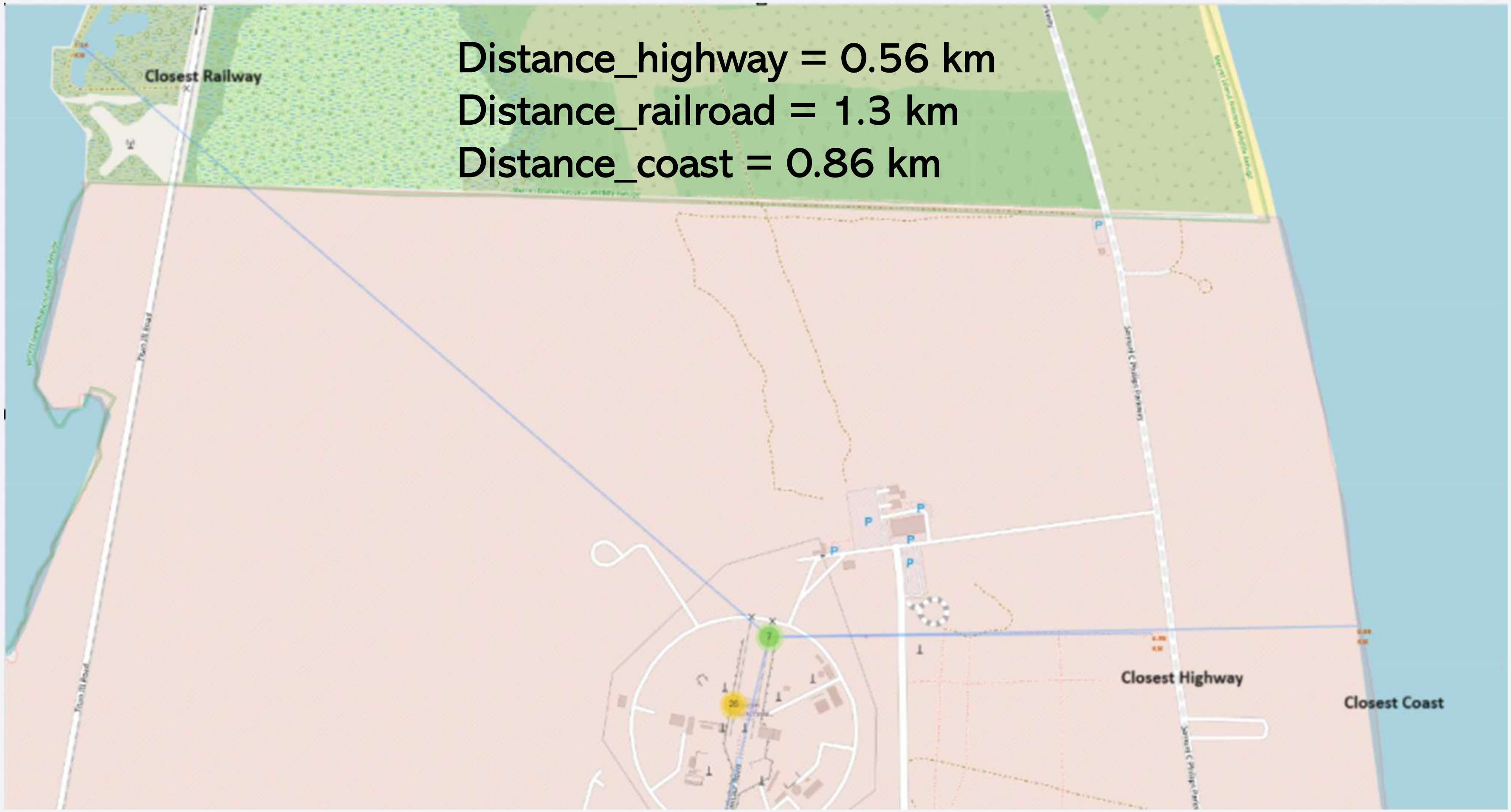
Florida Launch Sites

Green Marker shows successful Launches and Red Marker shows Failures

California Launch Site

37

Launch Site distance to landmarks



Section 4

Build a Dashboard with Plotly Dash

Section 5

Predictive Analysis (Classification)

Classification Accuracy

- The decision tree classifier is the model with the highest classification accuracy

Create a logistic regression object then create a GridSearchCV object log...
...om the dictionary parameters.

```
[13]: parameters ={'C':[0.01,0.1,1],
                   'penalty':['l2'],
                   'solver':['lbfgs']}
      lr=LogisticRegression()

[14]: logreg_cv = GridSearchCV(lr,parameters,cv=10)
      logreg_cv.fit(X_train, Y_train)

[14]: GridSearchCV(cv=10, error_score='raise-deprecating',
             estimator=LogisticRegression(C=1.0, class_weight=None, du...
                intercept_scaling=1, max_iter=100, multi_class='warn',
                n_jobs=None, penalty='l2', random_state=None, solver='...
                tol=0.0001, verbose=0, warm_start=False),
             fit_params=None, iid='warn', n_jobs=None,
             param_grid={'C': [0.01, 0.1, 1], 'penalty': ['l2'], 'sol...
             pre_dispatch='2*n_jobs', refit=True, return_train_score='...
             scoring=None, verbose=0)
```

We output the GridSearchCV object for logistic regression. We display t...
the accuracy on the validation data using the data attribute best_score_...

```
[15]: print("tuned hpyerparameters :(best parameters) ",logreg_cv.bes...
      print("accuracy :",logreg_cv.best_score_)

tuned hpyerparameters :(best parameters)  {'C': 0.01, 'penalty': 'l2', 'solver': 'lbfgs'}
accuracy : 0.8472222222222222
```

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

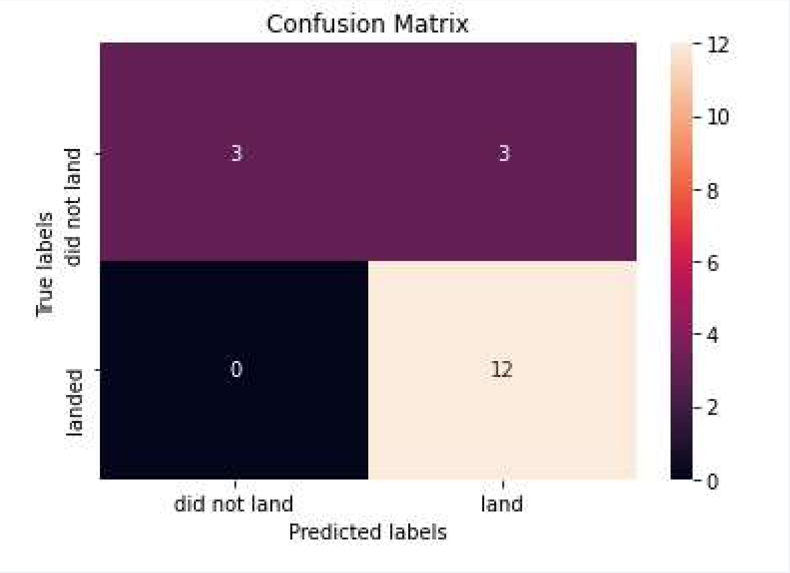
```
parameters = {'criterion': ['gini', 'entropy'],
     'splitter': ['best', 'random'],
     'max_depth': [2*n for n in range(1,10)],
     'max_features': ['auto', 'sqrt'],
     'min_samples_leaf': [1, 2, 4],
     'min_samples_split': [2, 5, 10]}

tree = DecisionTreeClassifier()

tree_cv = GridSearchCV(tree,parameters,cv=10)
tree_cv.fit(X_train, Y_train)

GridSearchCV(cv=10, error_score='raise-deprecating',
       estimator=DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best'),
       fit_params=None, iid='warn', n_jobs=None,
       param_grid={'criterion': ['gini', 'entropy'], 'splitter': ['best', 'random'], '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]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring=None, verbose=0)

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

tuned hpyerparameters :(best parameters)  {'criterion': 'entropy', 'max_depth': 12, 'max_features': 'sqrt', 'min_samples_leaf': 4, 'min_samples_split': 2, 'splitter': 'random'}
accuracy : 0.8888888888888888
```

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

```
...'rbf', 'sigmoid'),

...ng',
...s_weight=None, coef0=0.0,
...='auto_deprecated',
...random_state=None,

...poly', 'rbf', 'sigmoid'), 'C': array([1.00000e-03, 3.16228e-02, 1.00000e+00,
...000e-03, 3.16228e-02, 1.00000e+00, 3.16228e+01, 1.00000e+03])},
...rn_train_score='warn',

...",svm_cv.best_params_)

tuned hyperparameters :(best parameters)  {'C': 1.0, 'gamma': 0.03162277660168379, 'kernel': 'sigmoid'}
accuracy : 0.8472222222222222
```

Confusion Matrix

- The confusion matrix for the decision tree classifier shows that the classifier can distinguish between the different classes. The major problem is the false positives .i.e., unsuccessful landing marked as successful landing by the classifier.



Conclusions

We can conclude that:

- The larger the flight amount at a launch site, the greater the success rate at a launch site.
- Launch success rate started to increase in 2013 till 2020.
- Orbits ES-L1, GEO, HEO and SSO had the most success rate.
- The Decision tree classifier is the best machine learning algorithm for this task.


Thank you!