



IBM Developer
SKILLS NETWORK

Winning Space Race with Data Science

Scott Hagen
March 10, 2023



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 (EDA) with SQL
 - Exploratory Data Analysis (EDA) 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

Executive Summary

- 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

- The data was collected using various methods
 - Data collection was done using get request to the SpaceX API.
 - Next, we decoded the response content as a Json using `.json()` function call and turn it into a pandas dataframe using `.json_normalize()`.
 - 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 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

1. Get request for rocket launch data using API

```
Now let's start requesting rocket launch data from SpaceX API with the following URL:
```

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

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

2. Used json_normalize method to convert json result to dataframe

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

```
In [9]: static_json_url="https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/datasets/API_call_spacex_api.json"
```

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

```
In [10]: response.status_code
```

```
Out[10]: 200
```

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

```
In [16]: # Use json_normalize method to convert the json result into a dataframe
jlist = requests.get(static_json_url).json()
df2 = pd.json_normalize(jlist)
df2.head()
df = pd.read_json(static_json_url)
```

3. Extract only useful columns and store in lists

```
We will now use the API again to get information about the launches using the IDs given for each launch. Specifically we will be using columns rocket, payloads, launchpad, and cores.
```

```
In [18]: # Starting a copy of 'df' dataframe in 'data'
data = df

# Lets take a subset of our dataframe keeping only the features we want and the flight number, and date_utc.
data = data[['rocket', 'payloads', 'cores', 'flight_number', 'date_utc']]

# We will remove rows with multiple cores because those are falcon rockets with 2 extra rocket boosters and rows that have multiple payloads in a single rocket.
data = data[data['cores'].map(len)==1]
data = data[data['payloads'].map(len)==1]

# Since payloads and cores are lists of size 1 we will also extract the single value in the list and replace the feature.
data['cores'] = data['cores'].map(lambda x : x[0])
data['payloads'] = data['payloads'].map(lambda x : x[0])

# We also want to convert the date_utc to a datetime datatype and then extracting the date leaving the time
data['date'] = pd.to_datetime(data['date_utc']).dt.date

# Using the date we will restrict the dates of the launches
data = data[data['date'] <= datetime.date(2020, 11, 13)]
```

4. Construct dataset by combining columns into a dictionary and create a Pandas data frame

```
Finally lets construct our dataset using the data we have obtained. We will combine the columns into a dictionary.
```

```
In [26]: launch_dict = {'FlightNumber': list(data['flight_number']),
                       'Date': list(data['date']),
                       'BoosterVersion': BoosterVersion,
                       'PayloadMass': PayloadMass,
                       'Orbit': Orbit,
                       'LaunchSite': LaunchSite,
                       'Outcome': Outcome,
                       'Flights': Flights,
                       'GridFins': GridFins,
                       'Reused': Reused,
                       'Legs': Legs,
                       'LandingPad': LandingPad,
                       'Block': Block,
                       'ReusedCount': ReusedCount,
                       'Serial': Serial,
                       'Longitude': Longitude,
                       'Latitude': Latitude}
```

```
Then, we need to create a Pandas data frame from the dictionary launch_dict.
```

```
In [27]: # Create a data from launch_dict
data_falcon9 = pd.DataFrame(launch_dict)
```

5. Filter for only Falcon 9 launches and reset launch number

```
Finally we will remove the Falcon 1 launches keeping only the Falcon 9 launches. Filter the data dataframe using the BoosterVersion column to only keep the Falcon 9 launches. Save the filtered data to a new dataframe called data_falcon9.
```

```
In [29]: # Hint data['BoosterVersion']!='Falcon 1'
data_falcon9 = data_falcon9.drop(data_falcon9[data_falcon9['BoosterVersion']!='Falcon 9'].index, inplace = True)
```

```
Now that we have removed some values we should reset the FlightNumber column
```

```
In [30]: data_falcon9.loc[:, 'FlightNumber'] = list(range(1, data_falcon9.shape[0]+1))
data_falcon9
```

```
Out[30]:
```

FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins	Reused	Legs	LandingPad	Block	ReusedCount	Serial	Latitude	Longitude
4	2010-06-04	Falcon 9	NaN	LEO	CCSFS SLC 40	None	1	False	False	False	None	1.0	0	80003	-81.1419	-172.5680

Data Collection - Scraping

1. Request Falcon 9 Launch page from URL

To keep the lab tasks consistent, you will be asked to scrape the data from a snapshot of the `List of Falcon 9 and Falcon Heavy launches` Wikipedia updated on 9th June 2021

```
In [4]: static_url = "https://en.wikipedia.org/w/index.php?title=List_of_Falcon_9_and_Falcon_Heavy_launches&oldid=1027686922"
```

Next, request the HTML page from the above URL and get a `response` object

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 [10]: # use requests.get() method with the provided static_url
# assign the response to a object
data = requests.get(static_url)
data.status_code
```

Out[10]: 200

2. Create BeautifulSoup object from HTML

Create a `BeautifulSoup` object from the HTML response

```
In [12]: # Use BeautifulSoup() to create a BeautifulSoup object from a response text content
soup = BeautifulSoup(data.text, 'html.parser')
```

3. Extract all column/variable 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

```
In [14]: # 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')
```

4. Create a data frame by parsing the launch HTML tables and export to csv file

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

```
In [18]: launch_dict= dict.fromkeys(column_names)

# Remove an irrelevant 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']=[]
```

After you have fill in the parsed launch record values into `launch_dict`, you can create a dataframe from it.

```
In [20]: df=pd.DataFrame(launch_dict)
```

We can now export it to a **CSV** for the next section, but to make the answers consistent and in case you have difficulties finishing this lab.

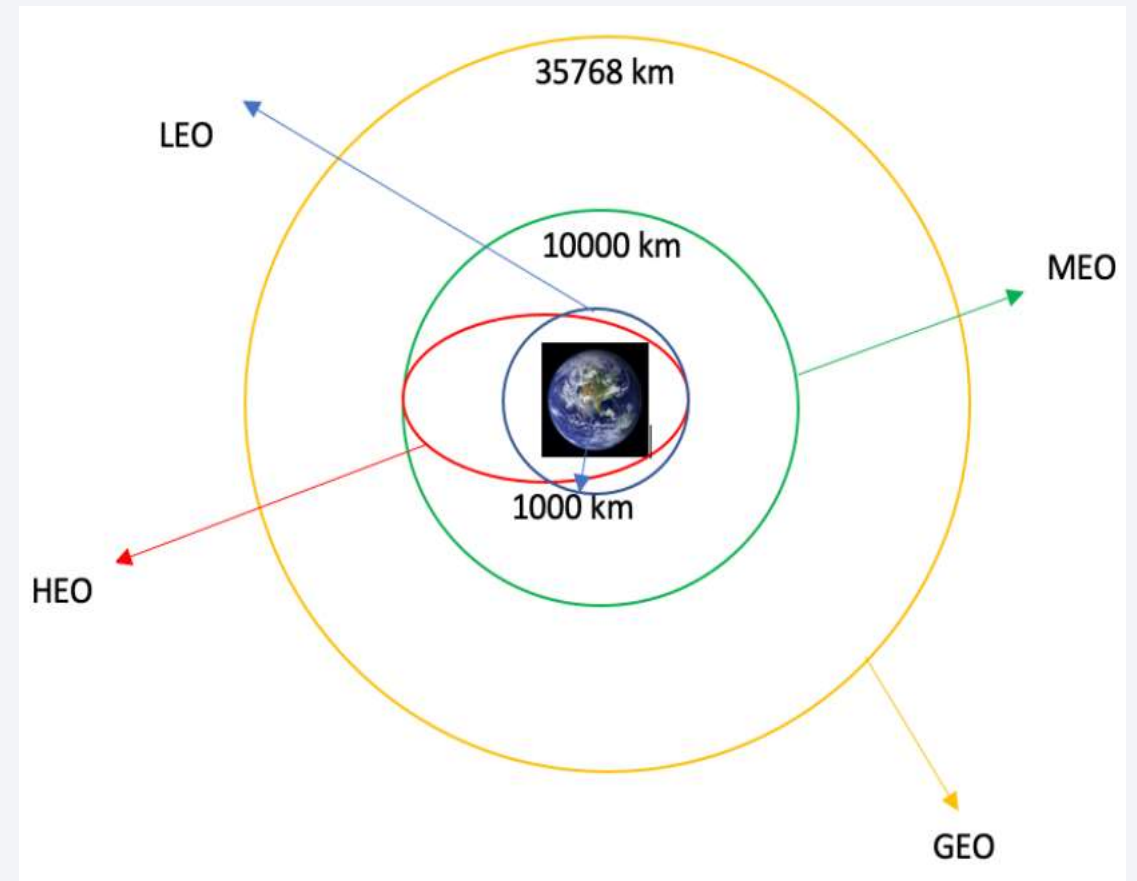
Following labs will be using a provided dataset to make each lab independent.

```
df.to_csv('spacex_web_scraped.csv', index=False)
```

```
In [21]: df.to_csv('spacex_web_scraped.csv', index=False)
```

Data Wrangling

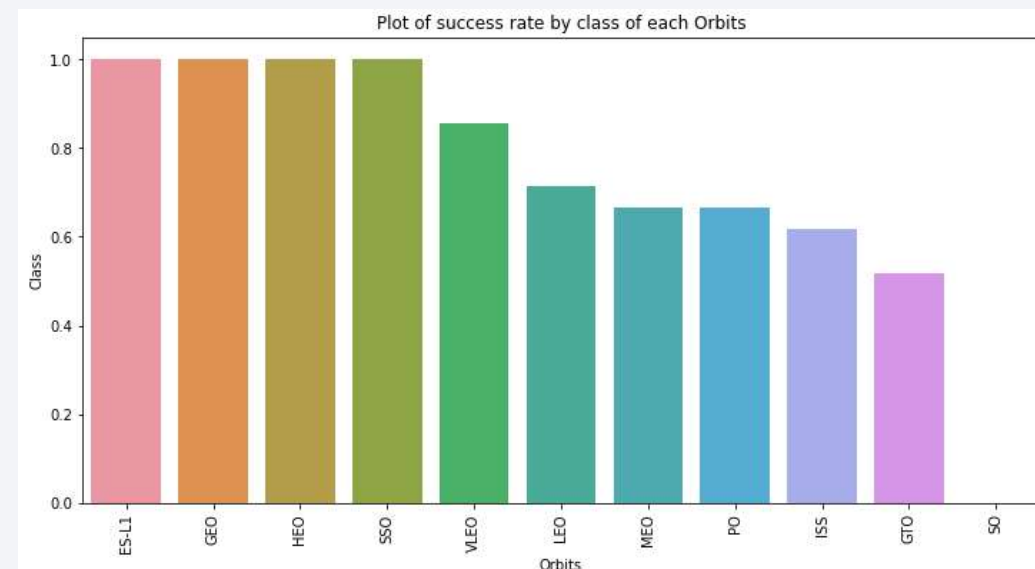
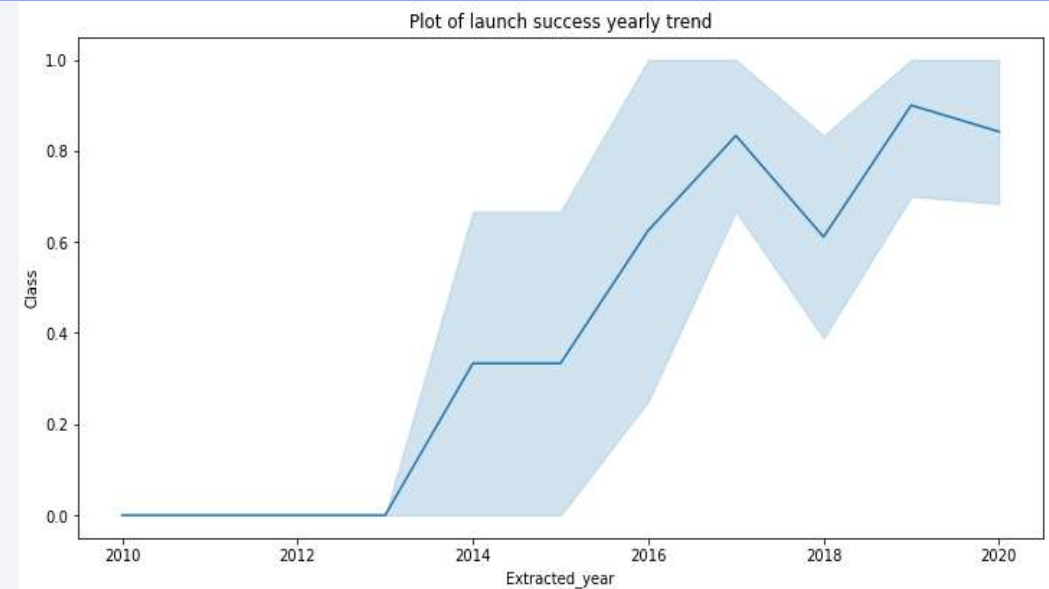
- Performed exploratory data analysis and determined the training labels.
- Calculated the number of launches at each site, and the number and occurrence of each orbits
- Created landing outcome label from outcome column and exported the results to csv.



EDA with Data Visualization

- Explored the data by visualizing the relationship between the following:
 - 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.

[GitHub - Scott Hagen – EDA Data Visualization](#)



EDA with SQL

- Loaded the SpaceX dataset without leaving the jupyter notebook.
- 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.
 - Display 5 records where launch sites begin with the string 'CCA'
 - 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.

Build an Interactive Map with Folium

- Marked all launch sites, and added map objects such as markers, circles, lines to indicate 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.
 - 0 for failure, and 1 for success.
 - Changed the colour of the marker to indicate success (Green) or failure (Red)
- Calculated the distances between a launch site to its proximities and created line objects for a visual representation
 - Answered the following questions for instance:
 - Are launch sites near railways, highways and coastlines. – No, No, Yes
 - Do launch sites keep certain distance away from cities. - Yes

Build a Dashboard with Plotly Dash

- Plotly Dash is an interactive dashboard
- The dashboard contains two charts
 - Pie Charts showing the total launches by a certain sites. This chart is useful as you can visualize the distribution of landing outcomes across all launch sites or show the success rate of launches on individual sites.
 - Scatter Graph showing the relationship with Outcome and Payload Mass (Kg) for the different booster version. This chart is useful as you can visualize how different variables affect the landing outcomes

[GitHub - Scott Hagen – Ploty Dash](#)

Predictive Analysis (Classification)

- Completed the following steps
 - Loaded the data using numpy and pandas
 - Standardized and transformed the data
 - Split our data into training and testing.
- Built different machine learning models and tune different hyperparameters (SVM, Decision Trees, K-Nearest Neighbours and Logistic Regression) using GridSearchCV.
 - Used test data to evaluate models based on their accuracy scores and confusion matrix
 - Found the best performing classification model.

[GitHub - Scott Hagen – Machine Learning Predictions](#)

Results

- Exploratory data analysis indicated the success rate of the Falcon 9 landings was 66%
- Interactive analytics demo in screenshots in the following slides
- Predictive analysis results indicated the Decision Tree algorithm was the best at 87% accuracy

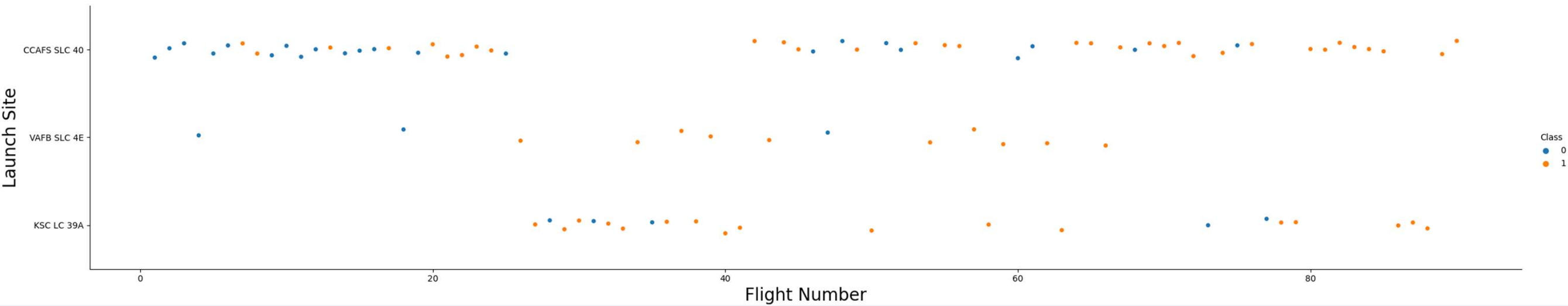
The background of the slide is an abstract composition of numerous thin, overlapping lines and streaks in shades of blue, red, and teal. These lines are oriented diagonally, creating a sense of motion and depth. The lines vary in opacity, with some appearing as bright, sharp streaks and others as more diffuse, textured bands. The overall effect is a complex, layered pattern that resembles a digital or data-driven aesthetic.

Section 2

Insights drawn from EDA

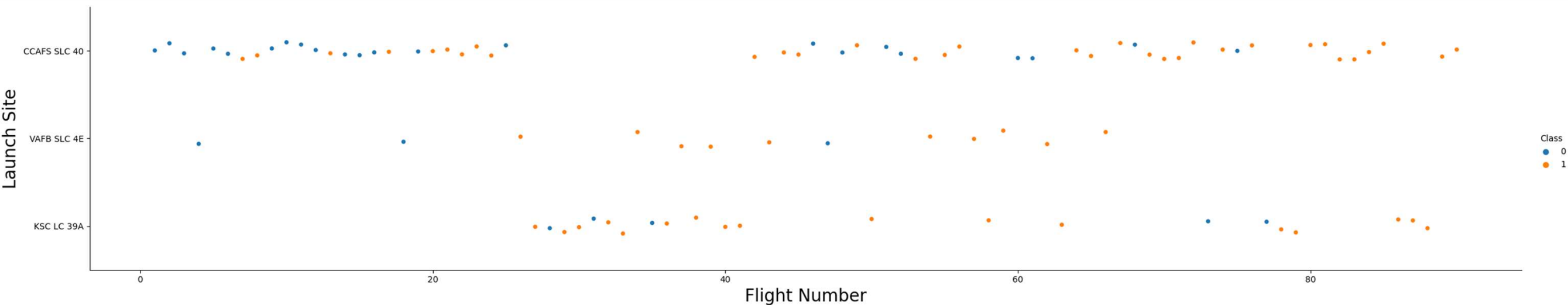
Flight Number vs. Launch Site

- The more flights at a launch site, the greater the success rate at a launch site.



Payload vs. Launch Site

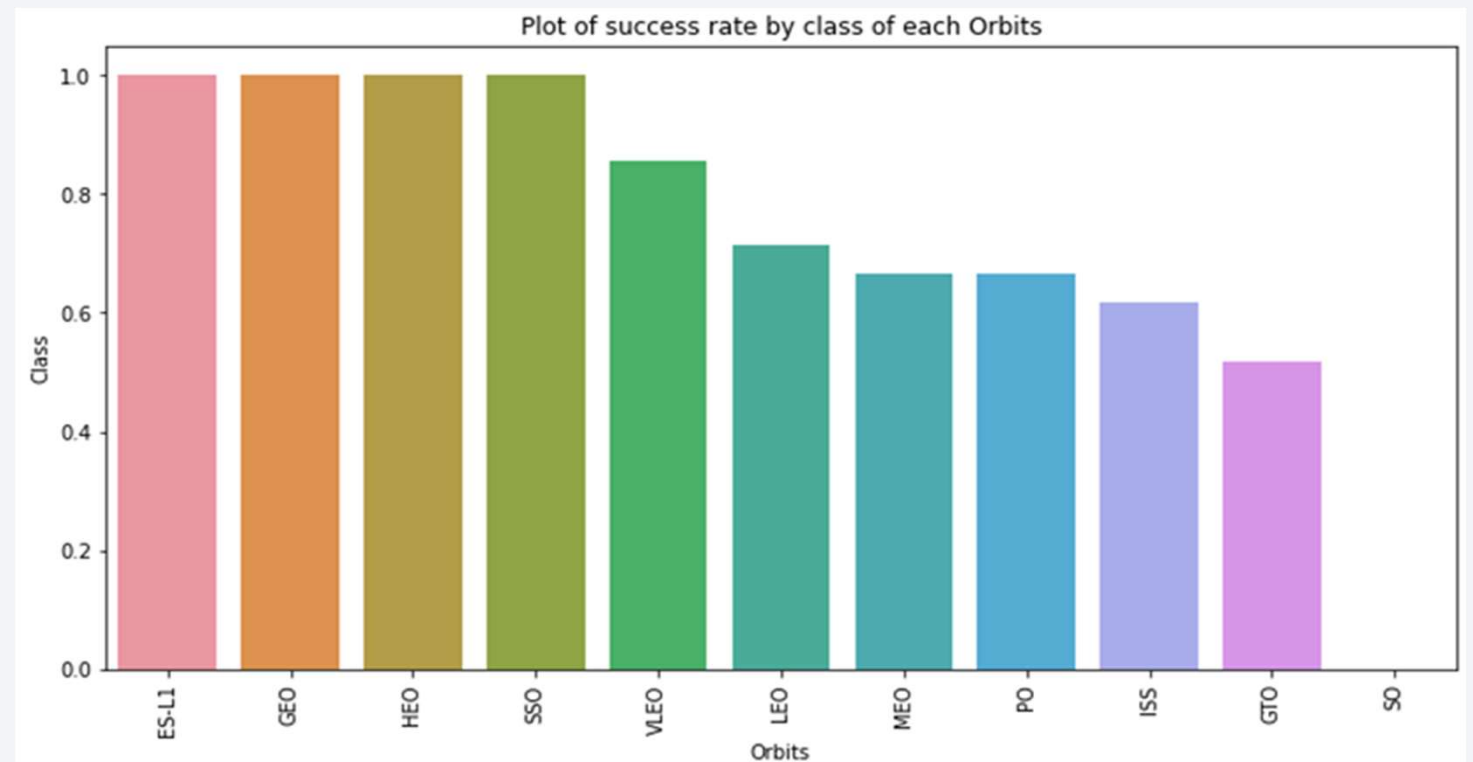
- The greater the payload mass at CCAFS SLC40, the greater the success rate at a launch site.
- No rockets launched for heavypayload mass (greater than 10000) at VAFB-SLC



Success Rate vs. Orbit Type

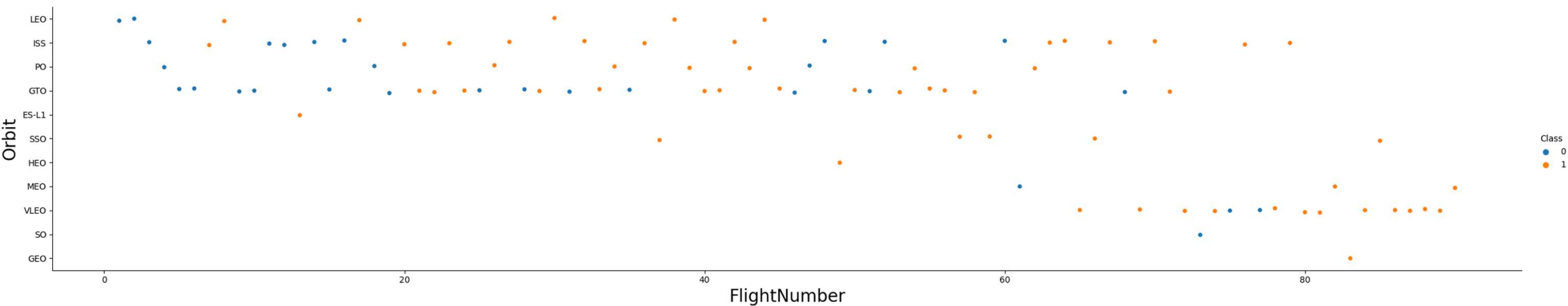
- The following all had the highest success rate:

- ES-L1
- GEO
- HEO
- SSO
- VLEO



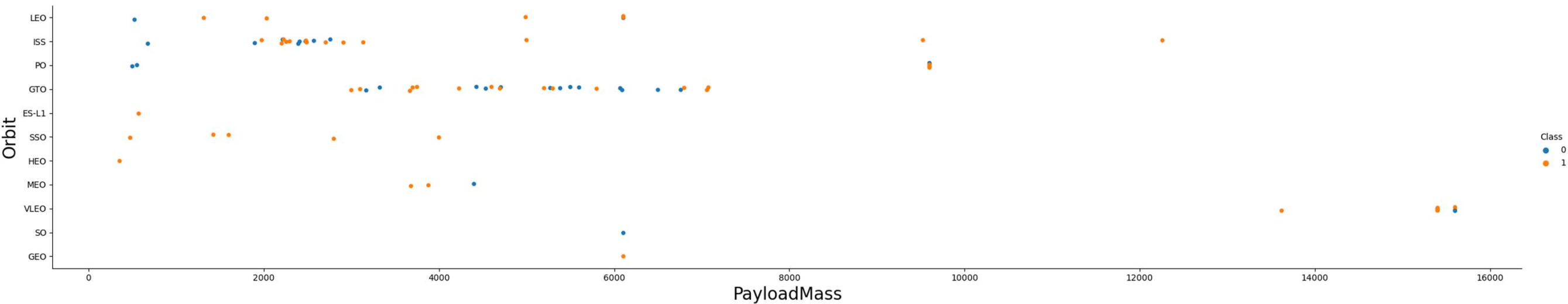
Flight Number vs. Orbit Type

- LEO orbit the Success appears to be related to the number of flights
- There seems to be no relationship between flight number when in GTO orbit.



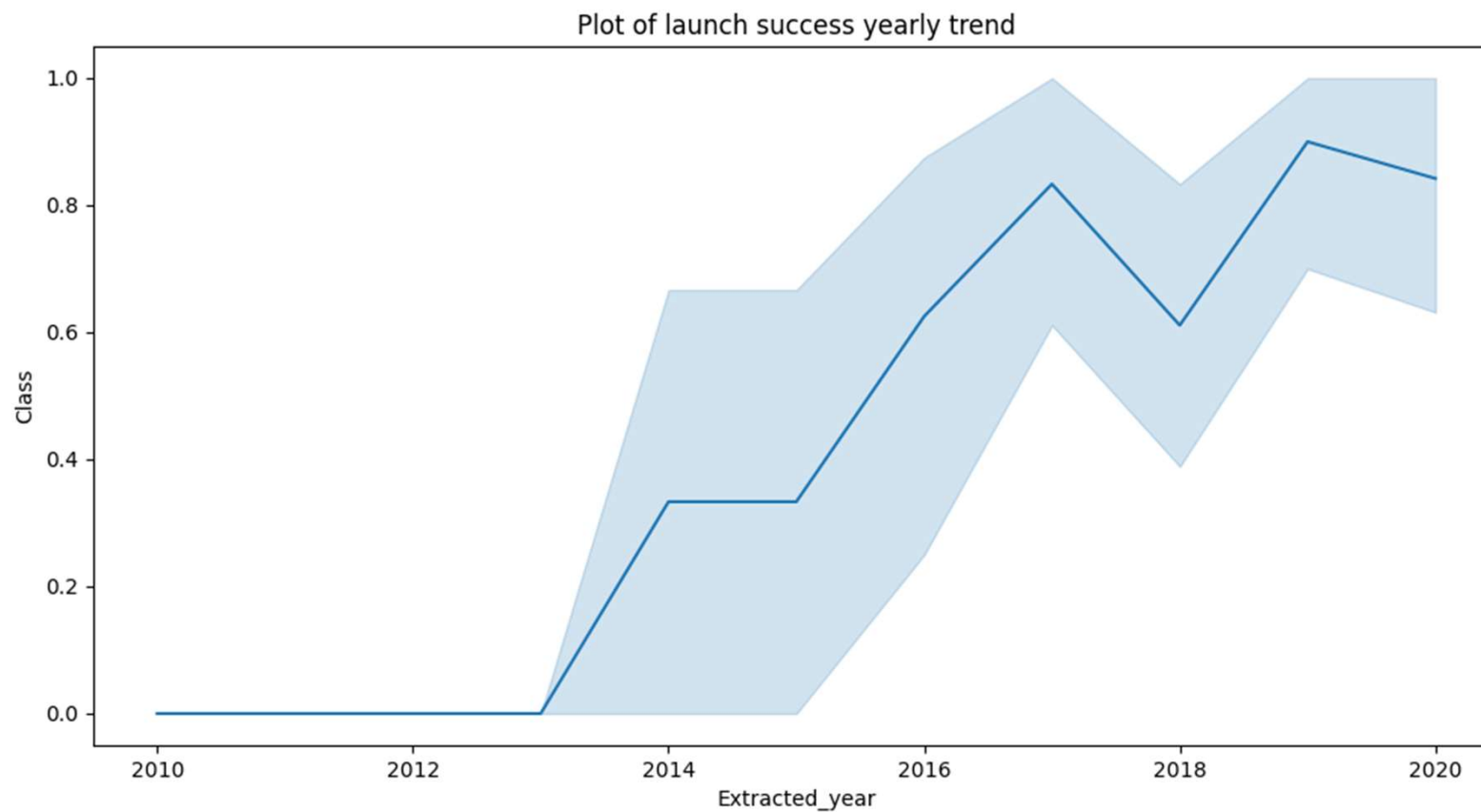
Payload vs. Orbit Type

- With heavy payloads the successful landing are more for Polar, LEO and ISS.



Launch Success Yearly Trend

- Success rate kept increasing from 2013-2020



All Launch Site Names

- DISTINCT is used to show only unique launch sites from the SpaceX data.

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

In [12]: %sql SELECT DISTINCT LAUNCH_SITE as "Launch_Sites" FROM SPACEXTBL;

* sqlite:///my_data1.db
Done.

Out[12]: Launch_Sites
         CCAFS LC-40
         VAFB SLC-4E
         KSC LC-39A
         CCAFS SLC-40
```

Launch Site Names Begin with 'CCA'

- The query below displays 5 records where launch sites begin with 'CCA'

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

```
In [47]: %sql SELECT * FROM SPACEXTBL WHERE LAUNCH_SITE LIKE 'CCA%' LIMIT 5;
```

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

```
Out[47]:
```

Date	Time (UTC)	Booster_Version	Launch_Site	Payload	PAYLOAD_MASS_KG_	Orbit	Customer	Mission_Outcome	Landing_Outcome
04-06-2010	18:45:00	F9 v1.0 B0003	CCAFS LC-40	Dragon Spacecraft Qualification Unit	0	LEO	SpaceX	Success	Failure (parachute)
08-12-2010	15:43:00	F9 v1.0 B0004	CCAFS LC-40	Dragon demo flight C1, two CubeSats, barrel of Brouere cheese	0	LEO (ISS)	NASA (COTS) NRO	Success	Failure (parachute)
22-05-2012	07:44:00	F9 v1.0 B0005	CCAFS LC-40	Dragon demo flight C2	525	LEO (ISS)	NASA (COTS)	Success	No attempt
08-10-2012	00:35:00	F9 v1.0 B0006	CCAFS LC-40	SpaceX CRS-1	500	LEO (ISS)	NASA (CRS)	Success	No attempt
01-03-2013	15:10:00	F9 v1.0 B0007	CCAFS LC-40	SpaceX CRS-2	677	LEO (ISS)	NASA (CRS)	Success	No attempt

Total Payload Mass

- the total payload mass carried by boosters launched by NASA (CRS) is 45596

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

In [20]: `%sql SELECT SUM(PAYLOAD_MASS__KG_) AS "Total Payload Mass by NASA (CRS)" FROM SPACEXTBL WHERE CUSTOMER = 'NASA (CRS)';`

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

Out[20]:

Total Payload Mass by NASA (CRS)
45596

Average Payload Mass by F9 v1.1

- The average payload mass carried by booster version F9 v1.1 is 2928

```
In [21]: %sql SELECT AVG(PAYLOAD_MASS_KG_) AS "Average Payload Mass by Booster Version F9 v1.1" FROM SPACEXTBL \
        WHERE BOOSTER_VERSION = 'F9 v1.1';

* sqlite:///my_data1.db
Done.
Out[21]: Average Payload Mass by Booster Version F9 v1.1
        2928.4
```

First Successful Ground Landing Date

- The date of the first successful landing outcome on ground pad was January 5, 2017

```
In [42]: %sql SELECT MIN(DATE) AS "First Successful Landing Outcome in Ground Pad" FROM SPACEXTBL \
        WHERE "Landing_Outcome" = 'Success (ground pad)';

* sqlite:///my_data1.db
Done.
Out[42]: 

| First Successful Landing Outcome in Ground Pad |
|------------------------------------------------|
| 01-05-2017                                     |


```


Successful Drone Ship Landing with Payload between 4000 and 6000

- List the names of boosters which have successfully landed on drone ship and had payload mass greater than 4000 but less than 6000
- Used the WHERE clause 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

```
In [43]: %sql SELECT BOOSTER_VERSION FROM SPACEXTBL WHERE "LANDING _OUTCOME" = 'Success (drone ship)' \
        AND PAYLOAD_MASS__KG_ > 4000 AND PAYLOAD_MASS__KG_ < 6000;

* sqlite:///my_data1.db
Done.
Out[43]: Booster_Version
         F9 FT B1022
         F9 FT B1026
         F9 FT B1021.2
         F9 FT B1031.2
```

Total Number of Successful and Failure Mission Outcomes

- Calculate the total number of successful and failure mission outcomes
- Used wildcard like '%' to filter for WHERE MissionOutcome was a success or a failure
- Used sum to add if Successful or Failure

```
In [38]: %sql SELECT sum(case when MISSION_OUTCOME LIKE '%Success%' then 1 else 0 end) AS "Successful Mission", \
          sum(case when MISSION_OUTCOME LIKE '%Failure%' then 1 else 0 end) AS "Failure Mission" \
          FROM SPACEXTBL;

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

Out[38]:	Successful Mission	Failure Mission
	100	1

Boosters Carried Maximum Payload

- List the names of the booster which have carried the maximum payload mass
- Using a subquery in the WHERE clause and the MAX() function.

```
In [35]: %sql SELECT DISTINCT BOOSTER_VERSION AS "Booster Versions which carried the Maximum Payload Mass" FROM SPACEXTBL \
        WHERE PAYLOAD_MASS_KG_ =(SELECT MAX(PAYLOAD_MASS_KG_) FROM SPACEXTBL);

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

Out[35]: **Booster Versions which carried the Maximum Payload Mass**

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

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

```
%sql SELECT Date, booster_version, "Landing _Outcome" from SPACEXTBL where "Landing _Outcome"='Failure (drone ship)' AND DATE BETWEEN '2015-01-01' AN
```

* sqlite:///my_data1.db
Done.

	boosterversion	launchsite	landingoutcome
0	F9 v1.1 B1012	CCAFS LC-40	Failure (drone ship)
1	F9 v1.1 B1015	CCAFS LC-40	Failure (drone ship)

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

- 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

```
%sql SELECT "Landing _Outcome", COUNT("Landing _Outcome") FROM SPACEXTBL WHERE DATE BETWEEN '2010-06-04' AND '2017-03-20' GROUP BY "Landing _Outcome"
```

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

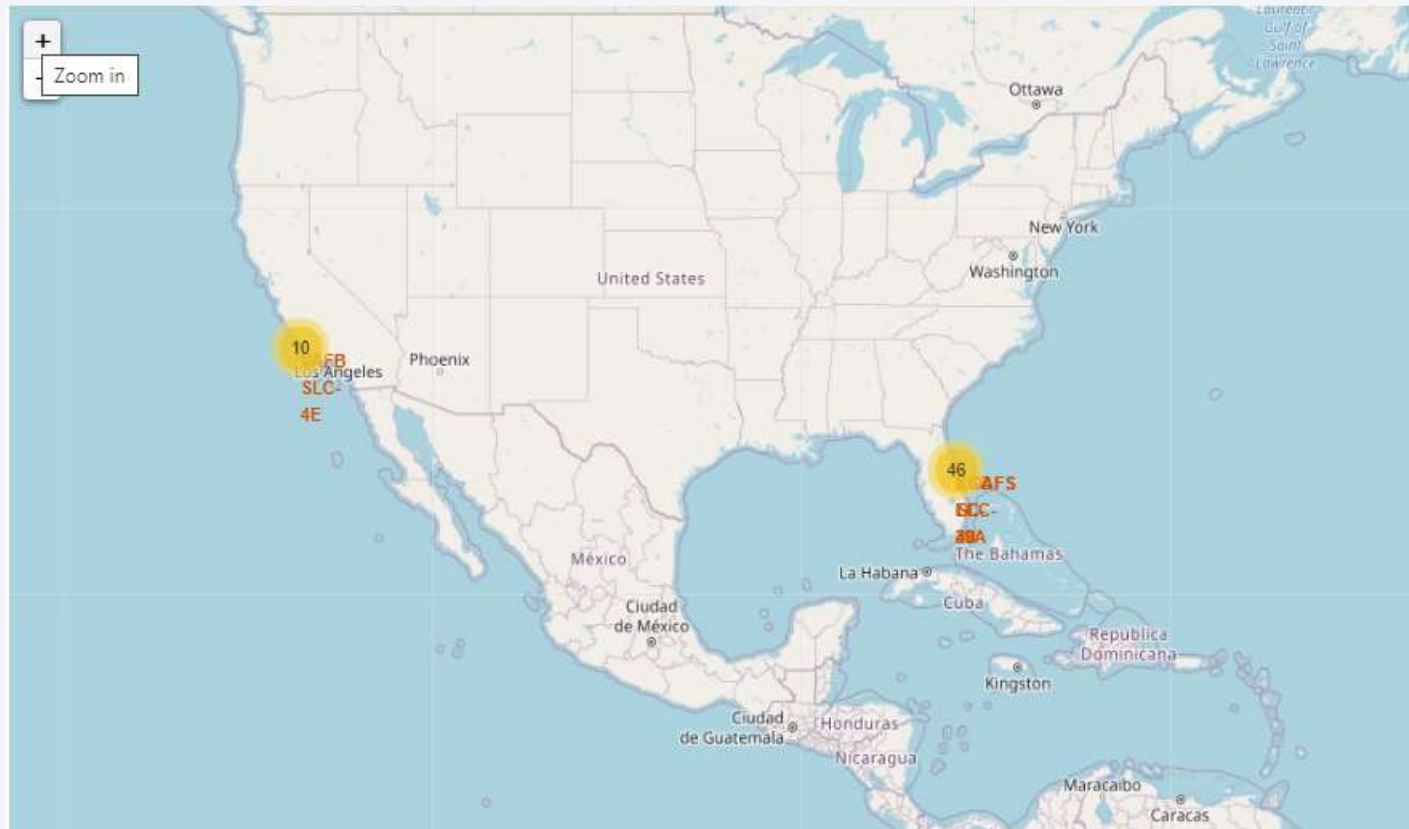
	landingoutcome	count
0	No attempt	10
1	Success (drone ship)	6
2	Failure (drone ship)	5
3	Success (ground pad)	5
4	Controlled (ocean)	3
5	Uncontrolled (ocean)	2
6	Precluded (drone ship)	1
7	Failure (parachute)	1

A satellite view of Earth from space, showing the curvature of the planet and the glow of city lights at night. The image is used as a background for the slide.

Section 3

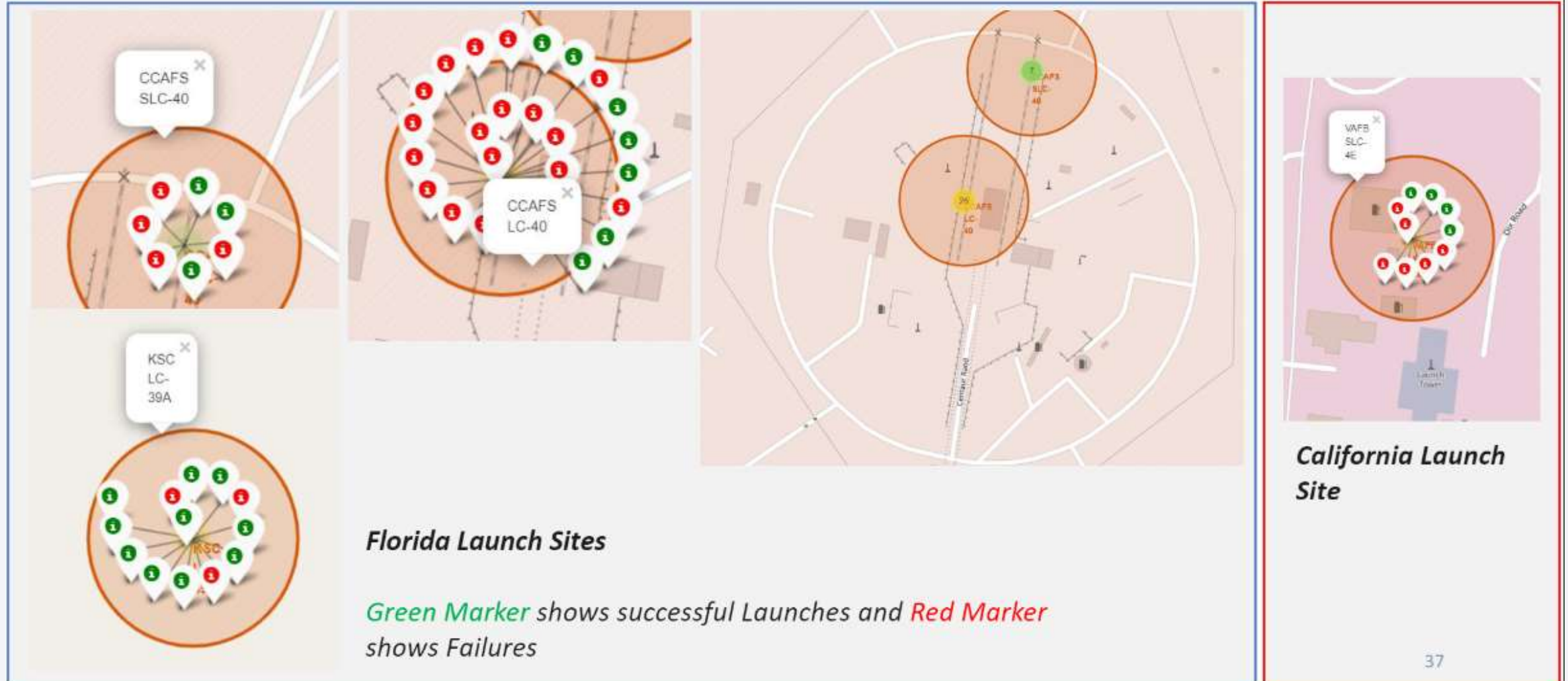
Launch Sites Proximities Analysis

SpaceX Launch Sites Locations

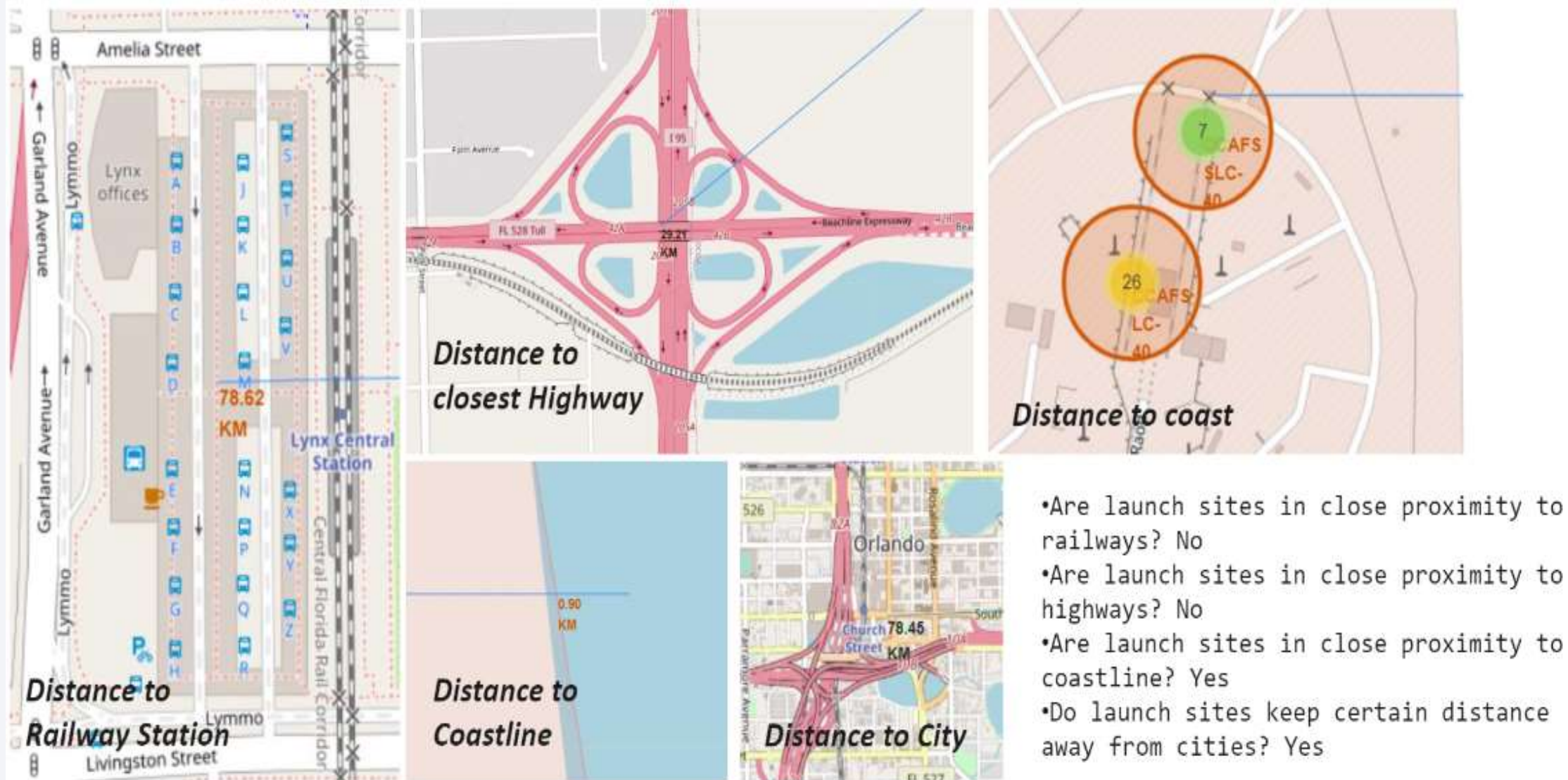


- The yellow markers are indicators of the cluster of locations of the SpaceX launch sites
- All launch sites are situated in the US.
- The launch sites have been strategically placed near the coast

Launch Sites with Colour Labels



Launch Proximity 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

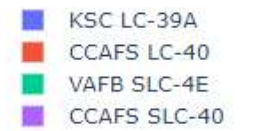
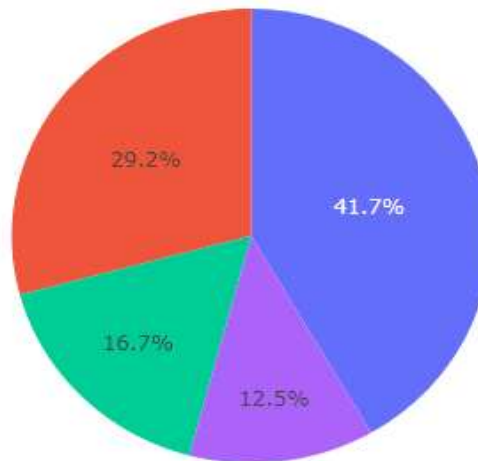


Section 4

Build a Dashboard with Plotly Dash

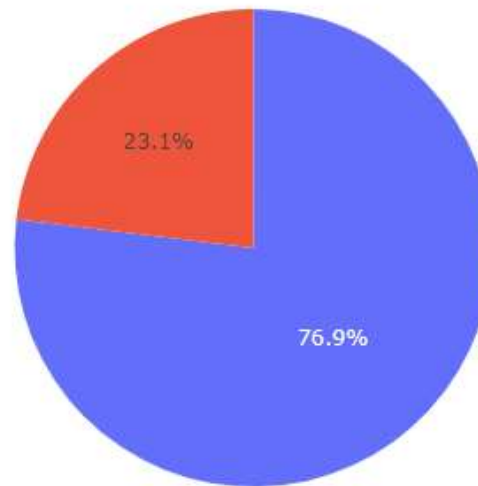
Total Successful Launches By Site

Total Success Launches By Site



Launch Site With Highest Success Ratio

Total Success Launched for site KSC LC-39A



Payload vs Launch Outcome Scatter Plot

- The booster version that has the largest success rate, in both weight ranges is the v1.1.
- The success rate for low weight payloads is higher than heavy weight payloads





Section 5

Predictive Analysis (Classification)

Classification Accuracy

```
models = {'KNeighbors': knn_cv.best_score_,
          'DecisionTree': tree_cv.best_score_,
          'LogisticRegression': logreg_cv.best_score_,
          'SupportVector': svm_cv.best_score_}

bestalgorithm = max(models, key=models.get)
print('Best model is', bestalgorithm, 'with a score of', models[bestalgorithm])
if bestalgorithm == 'DecisionTree':
    print('Best params is :', tree_cv.best_params_)
if bestalgorithm == 'KNeighbors':
    print('Best params is :', knn_cv.best_params_)
if bestalgorithm == 'LogisticRegression':
    print('Best params is :', logreg_cv.best_params_)
if bestalgorithm == 'SupportVector':
    print('Best params is :', svm_cv.best_params_)
```

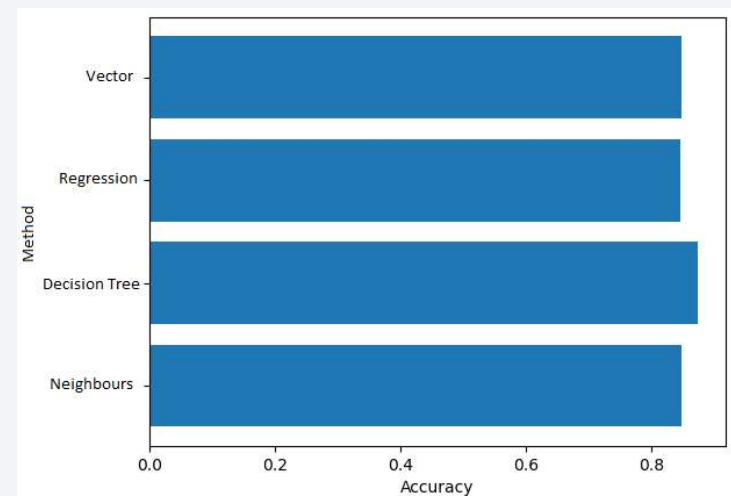
Best model is DecisionTree with a score of 0.8732142857142856

Best params is : {'criterion': 'gini', 'max_depth': 6, 'max_features': 'auto', 'min_samples_leaf': 2, 'min_samples_split': 5, 'splitter': 'random'}

```
import numpy as np
import matplotlib.pyplot as plt

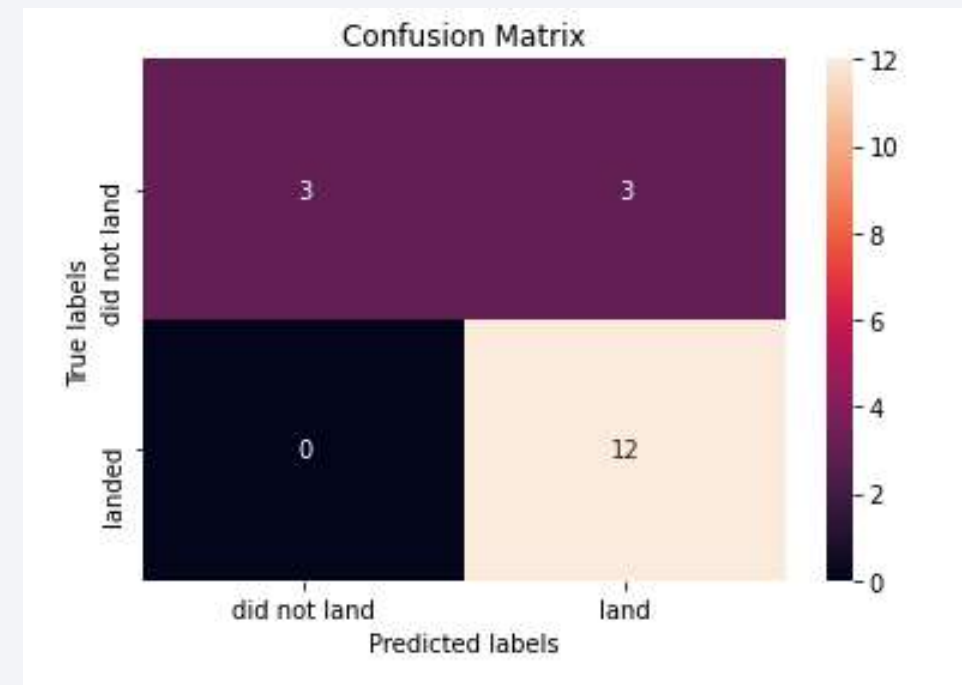
plt.barh(['Neighbours', 'Decision Tree', 'Regression', 'Vector'], [knn_cv.best_score_, tree_cv.best_score_, logreg_cv.best_score_, svm_cv.best_score_])
plt.xlabel('Accuracy')
plt.ylabel('Method')
plt.show()
```

The Decision Tree algorithm was the best at 87% accuracy



Confusion Matrix

- The confusion matrix for the best performing decision tree classifier shows that the classifier can distinguish between the different classes.
- The main problem are false positives
 - Unsuccessful landing marked as successful landing by the classifier.














Conclusions

- The more flights at a launch site, the greater the success rate.
- Launch success rate steadily increased from 2013 to 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.

Appendix

- Github screenshot

 README.md	Initial commit
 Scott Hagen - Week 1 - Space X - Da...	Add files via upload
 Scott Hagen - Week 1 - Space X - Da...	Add files via upload
 Scott Hagen - Week 1 - Space X - We...	Add files via upload
 Scott Hagen - Week 2 - Space X - ED...	Add files via upload
 Scott Hagen - Week 2 - Space X - ED...	Add files via upload
 Scott Hagen - Week 3 - Space X - Fol...	Add files via upload
 Scott Hagen - Week 3 - Space X - Plo...	Add files via upload
 Scott Hagen - Week 4 - Space X - Ma...	Add files via upload
 dataset_part_1.csv	Add files via upload
 spacex_web_scraped.csv	Add files via upload
README.md	
<h2>IBM-Applied-Data-Science</h2>	

Thank you!

