



IBM Developer  
SKILLS NETWORK

# Winning Space Race with Data Science

Travis  
4/5/2023



# Outline

---

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

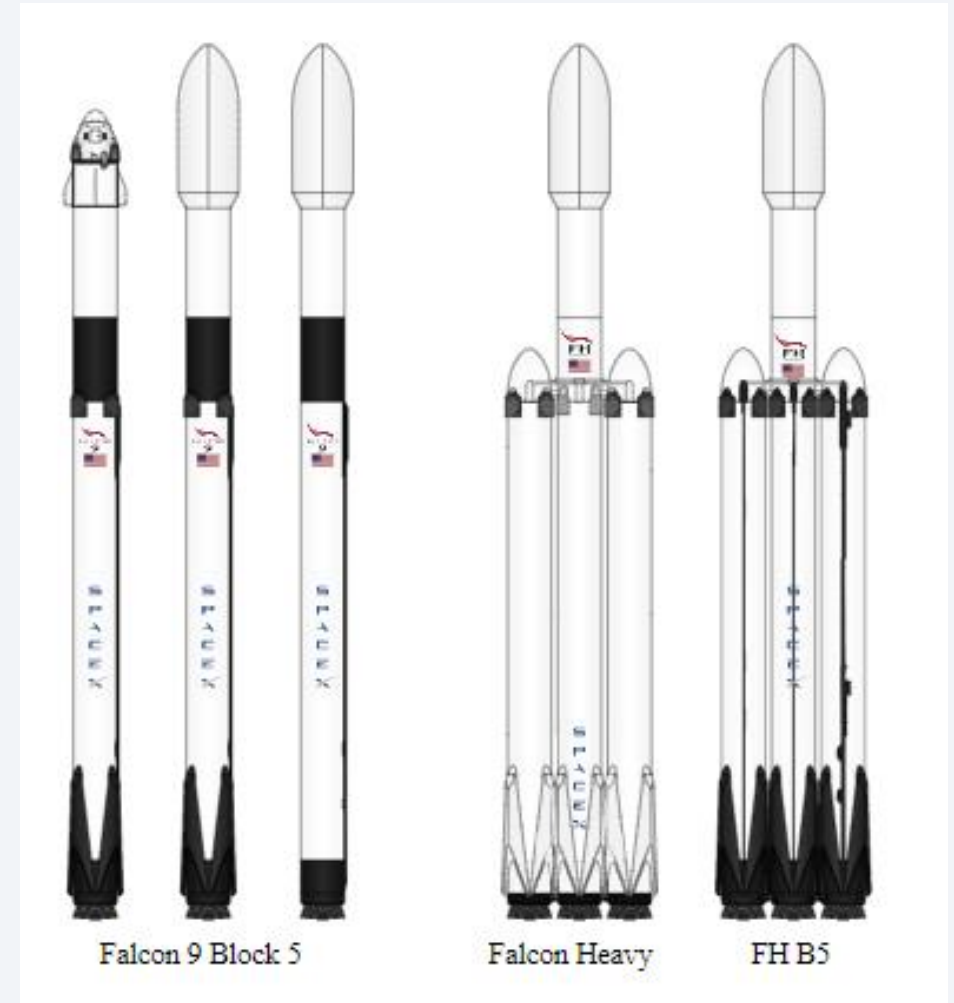
# Executive Summary

---

- Summary of methodologies
  - Data collection (SpaceX API and web scraping)
  - Data Wrangling
  - Exploratory Data Analysis (SQL queries and data visualization)
  - Interactive visualizations (folium maps and Plotly dashboard interface)
  - Machine Learning based prediction modeling
- Summary of all results
  - Launch success varies depending on a number of factors
  - Success rates for Space X have generally improved over time
  - Kennedy Space Center Complex is most successful launch site
  - Decision Tree classification algorithm makes the most accurate predictions for launch success utilizing the data available
  - False positives in predictions remain an area for improvement with the model

# Introduction

- Project background and context
  - Space X advertises Falcon 9 rocket launches on its website with a cost of \$62M; other providers cost as much as \$165M. Much of the difference is because Space X can reuse the first stage. If we can determine if the first stage will land, we can determine the cost of a launch. In this analysis, we created a machine learning pipeline to predict if the first stage will land utilizing data from Space X.
- Problems you want to find answers to
  - What are the most relevant factors that impact the outcome of a rocket launch
  - What are the best models/parameters to predict the outcome of a rocket launch
  - Will the rocket land successfully





Section 1

# Methodology

# Methodology

---

## Executive Summary

- Data collection methodology:
  - Get request to the SpaceX API and web scraping
- Perform data wrangling
  - Cleaned data, utilized one-hot encoding to process categorical features, and created an outcome feature “Class” to indicate landing success
- Perform exploratory data analysis (EDA) using visualization and SQL
- Perform interactive visual analytics using Folium and Plotly Dash
- Perform predictive analysis using classification models
  - Split dataset into training and test sets, optimized various ML models with hyperparameters, and scored test set accuracy for each for comparison

# Data Collection

---

- Data utilized in this analysis were collected using two primary methodologies:
- Get request to the SpaceX API
  - JSON response was converted into a Pandas DataFrame
  - Data was then cleaned, organized, assessed for missing values, and filled in as needed
- Web Scraping from Wikipedia
  - Wikipedia page with data on Falcon 9 launch/landing records were scraped utilizing the BeautifulSoup library
  - Tables embedded in the HTML for the page were parsed and converted into a Pandas DataFrame

# Data Collection – SpaceX API

- Get request to the SpaceX API
  - JSON response was converted into a Pandas DataFrame
  - Data was then cleaned, organized, assessed for missing values, and filled in as needed
- <https://github.com/eisen178/coursera/blob/2a5cda13c8fe8ec5fe8f748b079ec4b0fec8a346/jupyter-labs-spacex-data-collection-api.ipynb>

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

# We will remove rows with multiple cores because those are falcon rockets with 2 extra rocket boosters
data = data[data['cores'].map(len)==1]
# We will also remove rows with multiple payloads on a single rocket
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 lists by the value
data['cores'] = data['cores'].map(lambda x : x[0])
data['payloads'] = data['payloads'].map(lambda x : x[0])

# We 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

# Finally, we will restrict the dates of the launches to the last 100 launches
data = data[data['date'] <= datetime.date(2020, 11, 13)]
```

## Request and parse the SpaceX launch data using the GET request

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

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

and see that the request was successful 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



# Data Collection - Scraping

- Web Scraping from Wikipedia
  - Wikipedia page with data on Falcon 9 launch/landing records were scraped utilizing the BeautifulSoup library
  - Tables embedded in the HTML for the page were parsed and converted into a Pandas DataFrame
- <https://github.com/eisen178/coursera/blob/2a5cda13c8fe8ec5fe8f748b079ec4b0fec8a346/jupyter-labs-webscraping.ipynb>

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

```
requests.get() method with the provided static_url  
Assign the response to a object  
response = requests.get(static_url).text  
0.2s
```

Create a BeautifulSoup object from the HTML response

```
# Use BeautifulSoup() to create a BeautifulSoup object from a response text content  
soup = BeautifulSoup(response, 'html.parser')  
✓ 0.4s
```

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

```
# Use soup.title attribute  
print(soup.title)  
[10] ✓ 0.0s  
... <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, please 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')
```

✓ 0.0s

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

Python

# Data Wrangling

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

Create a landing outcome label from Outcome column

- Cleaned data, utilized one-hot encoding to process categorical features, assessed the number of launches by site and the types of orbits
- Created an outcome feature “Class” to indicate landing success for further use in modeling later
- <https://github.com/eisen178/coursera/blob/2a5cda13c8fe8ec5fe8f748b079ec4b0fec8a346/labs-jupyter-spacex-Data%20wrangling.ipynb>

```
for outcome, create a list where the element is zero if the corresponding row in Outcome is in the set bad_outcome;
otherwise, it is one. Then assign it to the variable landing_class:

landing_class = 0 if bad_outcome
landing_class = 1 otherwise

landing_class = []

for outcome in set(bad_outcomes):
    landing_class.append(0)

for outcome in set(landing_outcomes.keys()):
    landing_class.append(1)

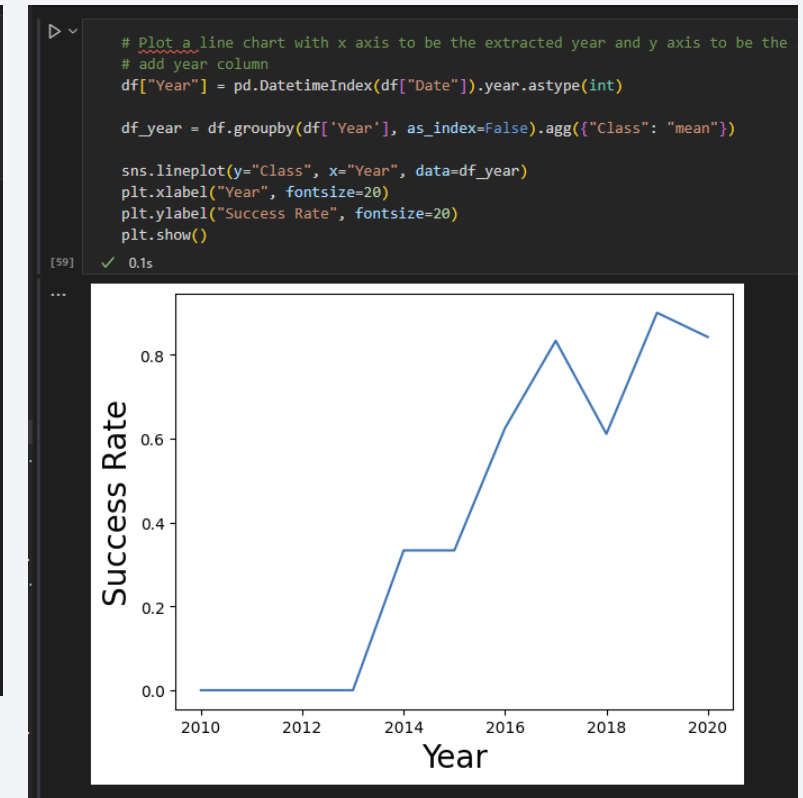
# Create a new column using numpy
landing_class = np.where(df['Outcome'].isin(set(bad_outcomes)), 0, 1)
```

Python

landing\_class 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

# EDA with Data Visualization

- Explored potential relationships in the data visually with scatterplots of various combinations of features
- Used bar charts and line graphs to plot the variable success rate of launches by orbit type and over time



- <https://github.com/eisen178/coursera/blob/2a5cda13c8fe8ec5fe8f748b079ec4b0fec8a346/jupyter-labs-eda-dataviz.ipynb>

the total number of successful and failure missi

## EDA with SQL

```
%sql SELECT "Mission_Outcome", COUNT(Mission_Outcome)
```

0.0s

- Using the SQLite library in Python, a number of SQL queries were written to help explore the [ite:///my\\_data1.db](https://github.com/eisen178/coursera/blob/master/jupyter-labs-eda-sql-coursera_sqlite.ipynb) SpaceX launch data previously collected, including:

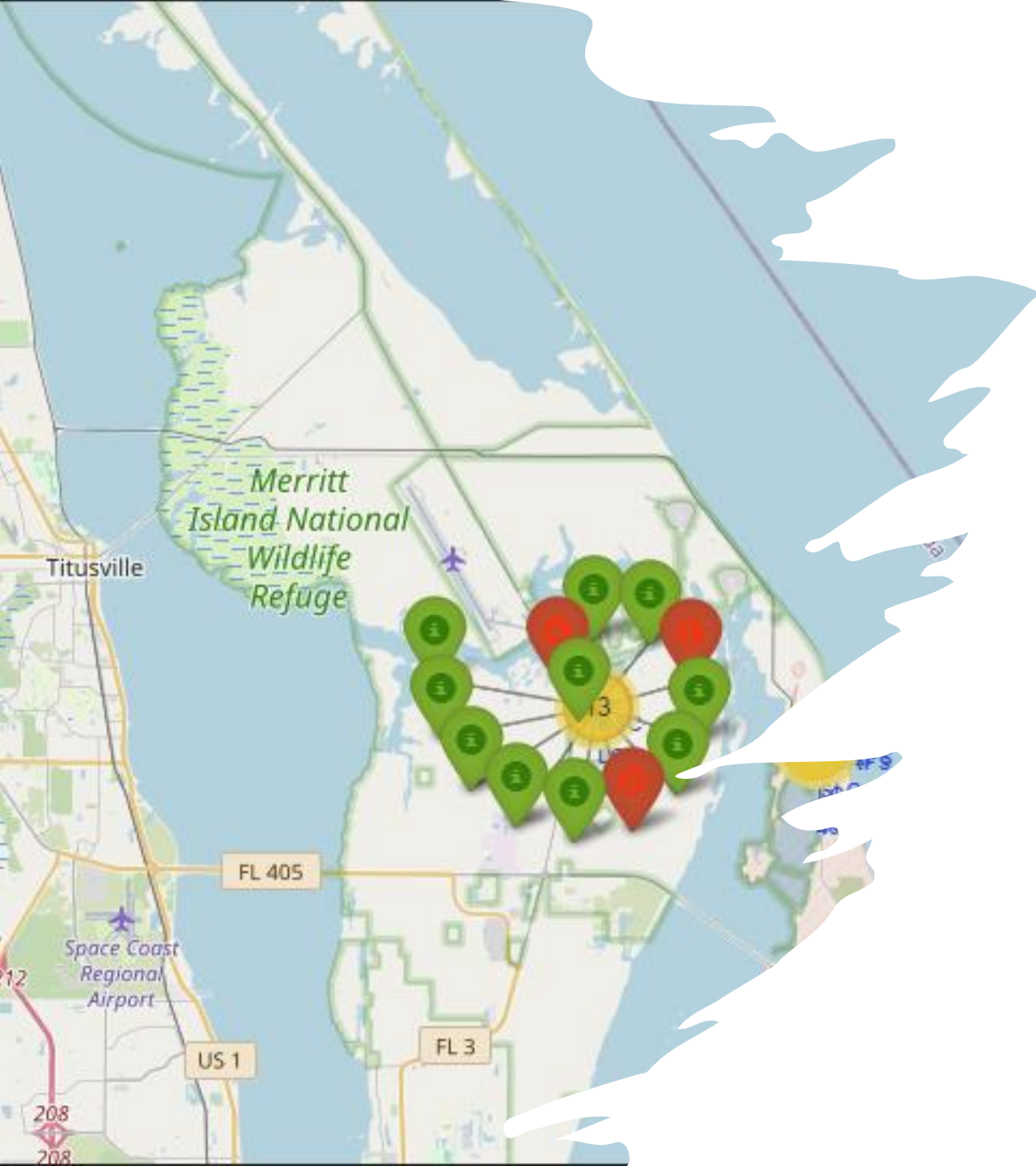
- Displaying unique launch sites
- Finding records for specific launch sites
- Filtering data specific to certain customers or booster types
- Determining the landing outcome (success/failure) and the types of landings achieved

- [https://github.com/eisen178/coursera/blob/master/jupyter-labs-eda-sql-coursera\\_sqlite.ipynb](https://github.com/eisen178/coursera/blob/master/jupyter-labs-eda-sql-coursera_sqlite.ipynb)

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

# Build an Interactive Map with Folium

- Using the launch data collected, maps were built to display objects such as markers, circles, lines, etc. on a folium map to note geographic information on the launches
- Using the created feature outcome “Class”, values of 0 and 1 were mapped to different markers to display the success rate by geographic location
- Additionally, launch site proximity to other nearby resources and locations were also mapped to better understand the qualities of the launch sites
- [https://github.com/eisen178/coursera/blob/ab59bbb84e3cb82842a3c1f5af84cefd51e90d56/lab\\_jupyter\\_launch\\_site\\_location.ipynb](https://github.com/eisen178/coursera/blob/ab59bbb84e3cb82842a3c1f5af84cefd51e90d56/lab_jupyter_launch_site_location.ipynb)





## SpaceX Launch Records Dashboard

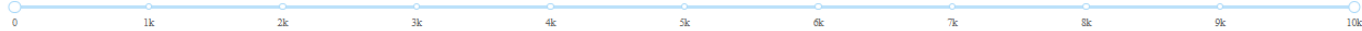
All Sites

X

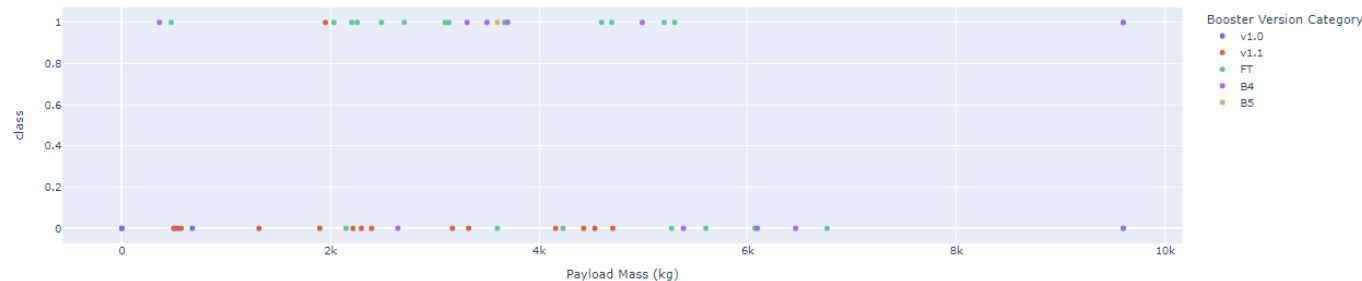
Launch Success Rate For All Sites



Payload range (Kg):



Launch Success Rate For All Sites

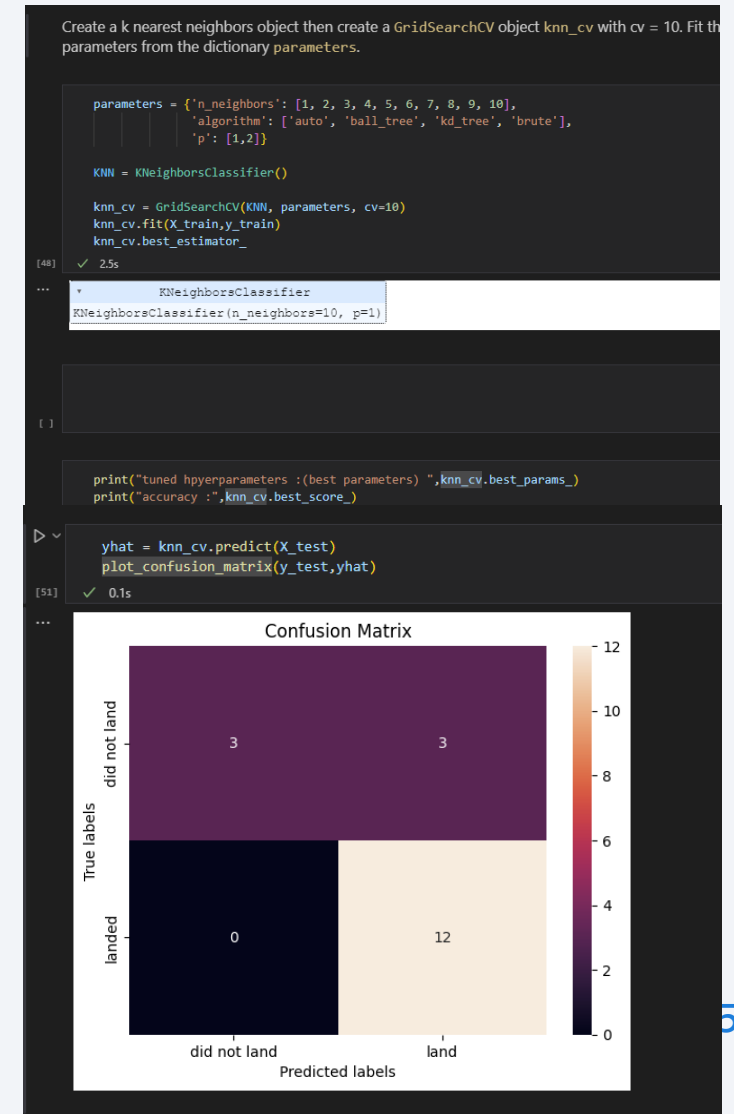


## Build a Dashboard with Plotly Dash

- Interactive dashboard built in Plotly Dash to display key data on launch success rates by site and booster version, with options to filter by payload range
- This was done to better visualize the outcomes across different data dimensions and make it more accessible to end users
- [https://github.com/eisen178/coursera/blob/ab59bbb84e3cb82842a3c1f5af84cefd51e90d56/SpaceX\\_dash\\_app.py](https://github.com/eisen178/coursera/blob/ab59bbb84e3cb82842a3c1f5af84cefd51e90d56/SpaceX_dash_app.py)

# Predictive Analysis (Classification)

- After pulling all of the data together into a DataFrame and preparing it for modeling, the dataset was then split into two subsets, a training and test set, to use in fitting and evaluating the performance of the models
- After fitting, the models were then optimized using hyperparameters, and scored on the test set to determine the accuracy for each for comparison
- Predictions were evaluated using a confusion matrix
- [https://github.com/eisen178/coursera/blob/ab59bbb84e3cb82842a3c1f5af84cefd51e90d56/SpaceX\\_Machine%20Learning%20Prediction\\_Part\\_5.ipynb](https://github.com/eisen178/coursera/blob/ab59bbb84e3cb82842a3c1f5af84cefd51e90d56/SpaceX_Machine%20Learning%20Prediction_Part_5.ipynb)



# Results

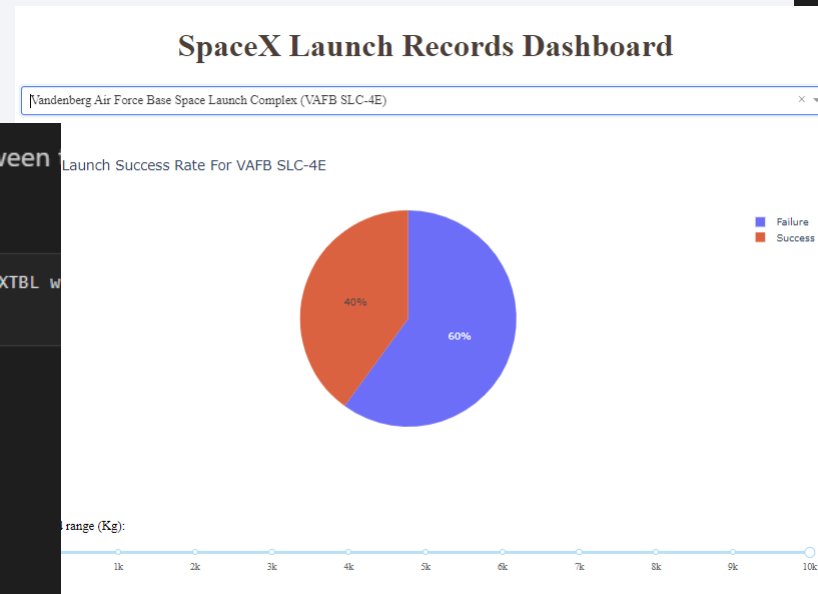
- Exploratory data analysis results
- Interactive analytics demo in screenshots
- Predictive analysis results

Rank the count of successful landing\_outcomes between descending order.

```
[77] ✓ 0.0s
```

... \* [sqlite:///my\\_data1.db](#)  
Done.

Landing_Outcome	Count_Success
Success	11
No attempt	6
Success (ground pad)	5
Success (drone ship)	5
Failure (drone ship)	4
Controlled (ocean)	3
Failure	2



Find the method performs best:

```
knn_score = [knn_cv.score(X_test, y_test)]  
tree_score = [tree_cv.score(X_test, y_test)]  
lr_score = [logreg_cv.score(X_test, y_test)]  
svm_score = [svm_cv.score(X_test, y_test)]
```

[52] ✓ 0.0s

```
eval_data = [['K-Nearest Neighbors', knn_score], ['D  
Report = pd.DataFrame(data=eval_data, columns=['Mode  
Report
```

✓ 0.0s

	Model Type	Accuracy
0	K-Nearest Neighbors	[0.8333333333333334]
1	Decision Tree	[0.8333333333333334]
2	Logistic Regression	[0.8333333333333334]
3	SVM	[0.8333333333333334]



The background of the slide is an abstract composition. It features a dark blue base color. Overlaid on this are numerous diagonal streaks in shades of red and cyan. A faint, light blue grid pattern is also visible, particularly in the lower-left quadrant. The overall effect is dynamic and technological.

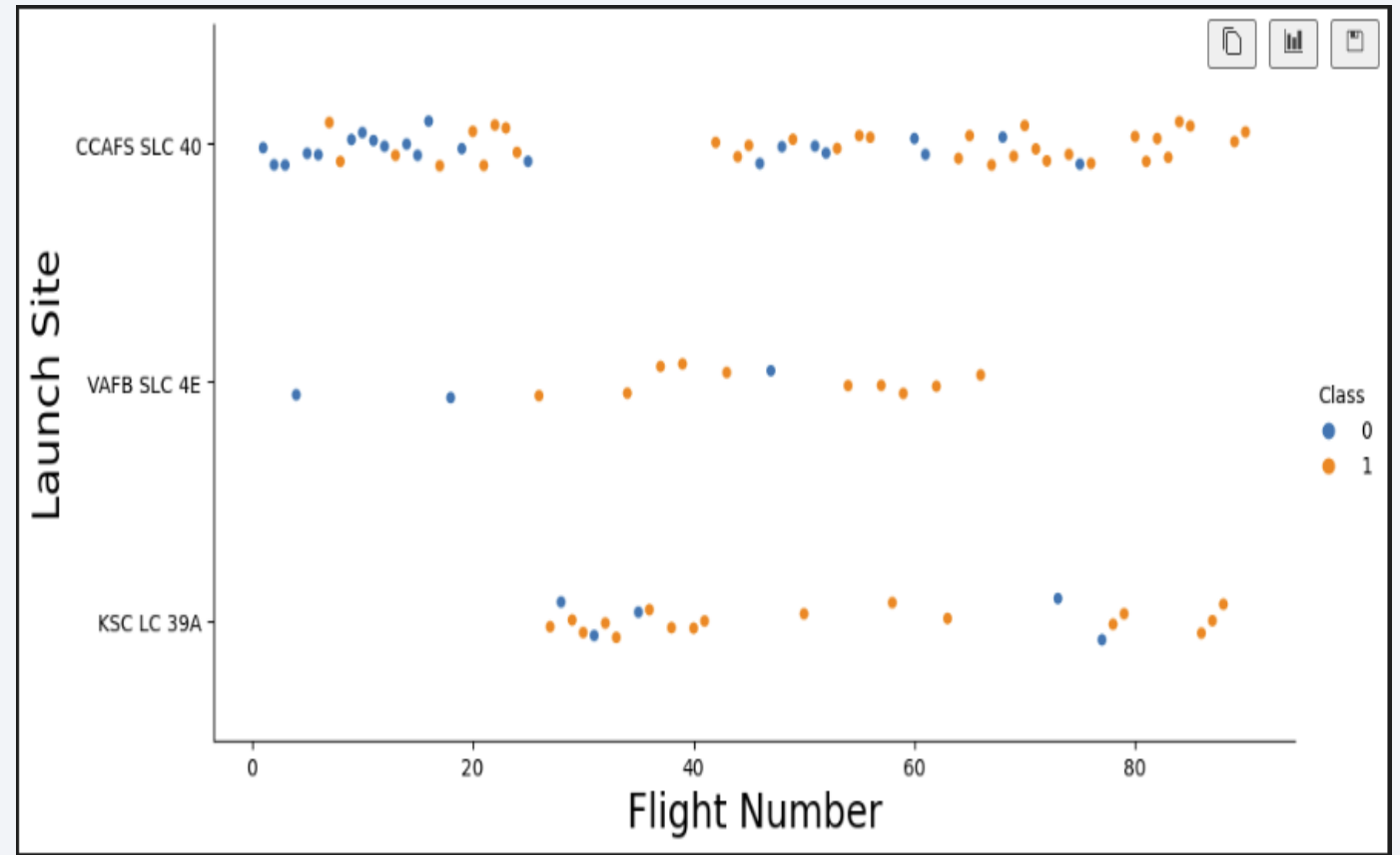
Section 2

# Insights drawn from EDA



# Flight Number vs. Launch Site

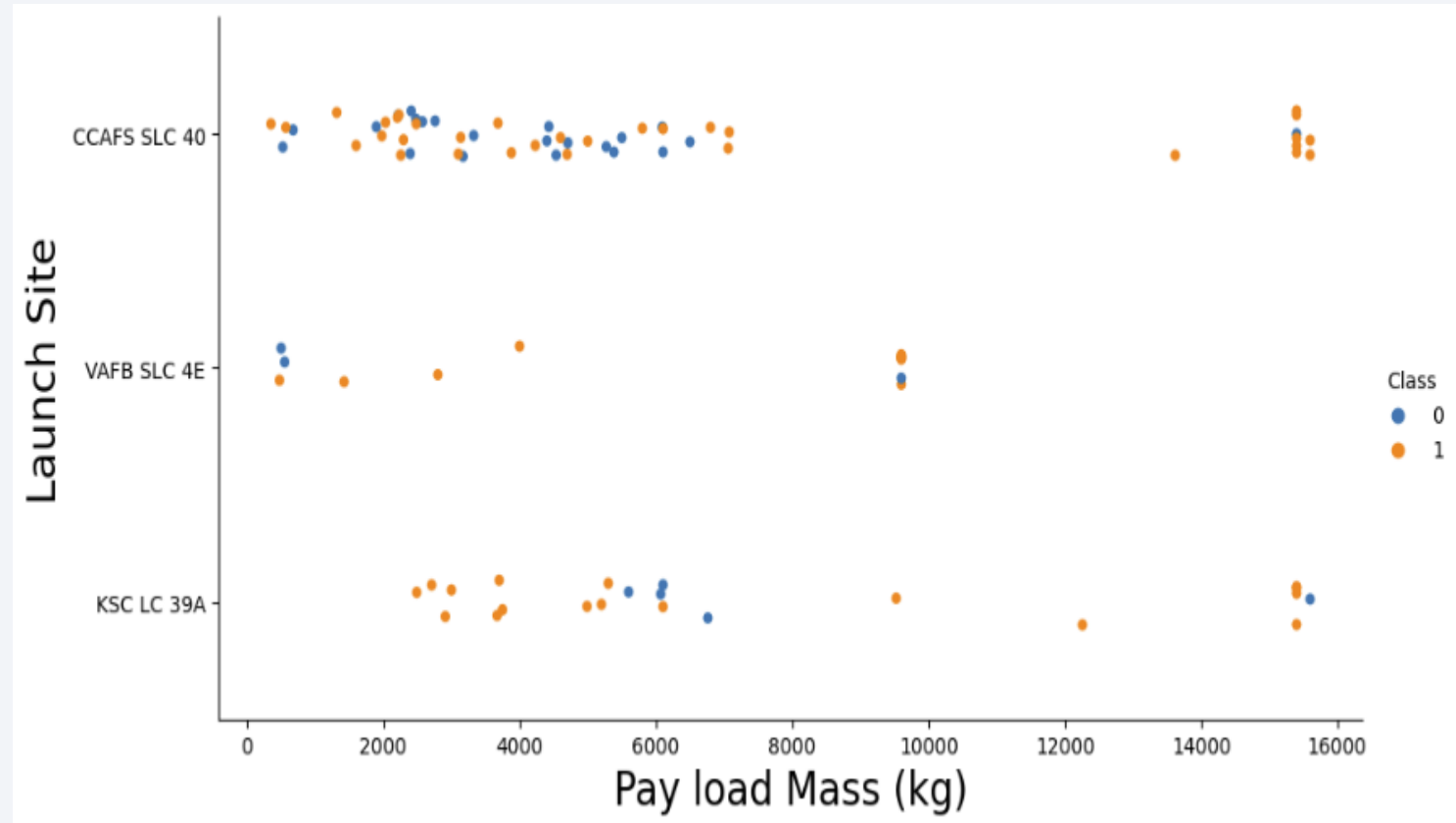
- Plotting flight number for each launch site with the success/failure feature “Class” we can see that the number of successes appears to increase as the flight number increases





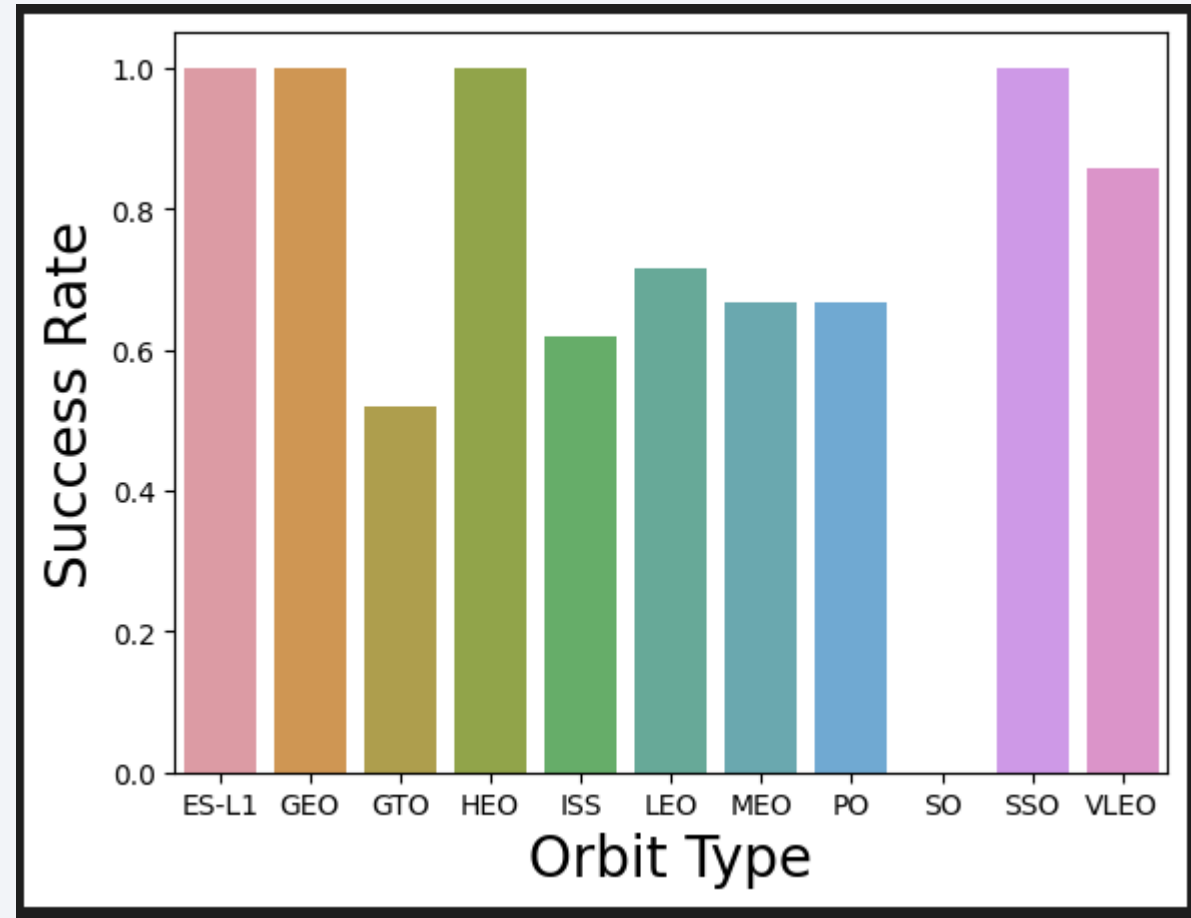
# Payload vs. Launch Site

- Depending on payload mass, the success of launches varied across sites
- CCAFS site showed high success at heavy payloads while KSC was best at low payloads



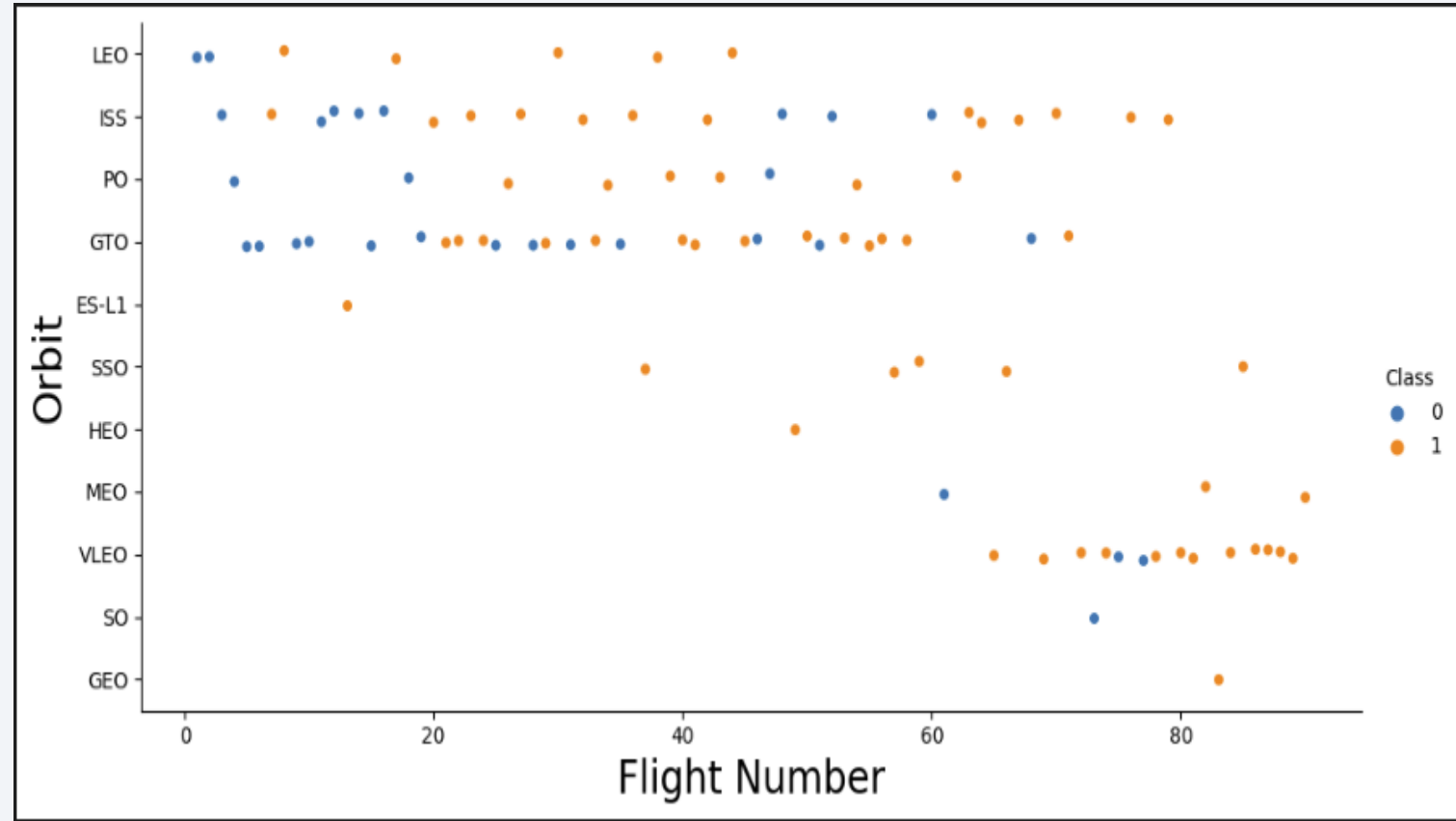
# Success Rate vs. Orbit Type

- Depending on the type of orbit achieved by a given launch, the success rate varied
- GTO, ISS, LEO, MEO, and PO launches all experienced relatively less success than other types of orbital launches



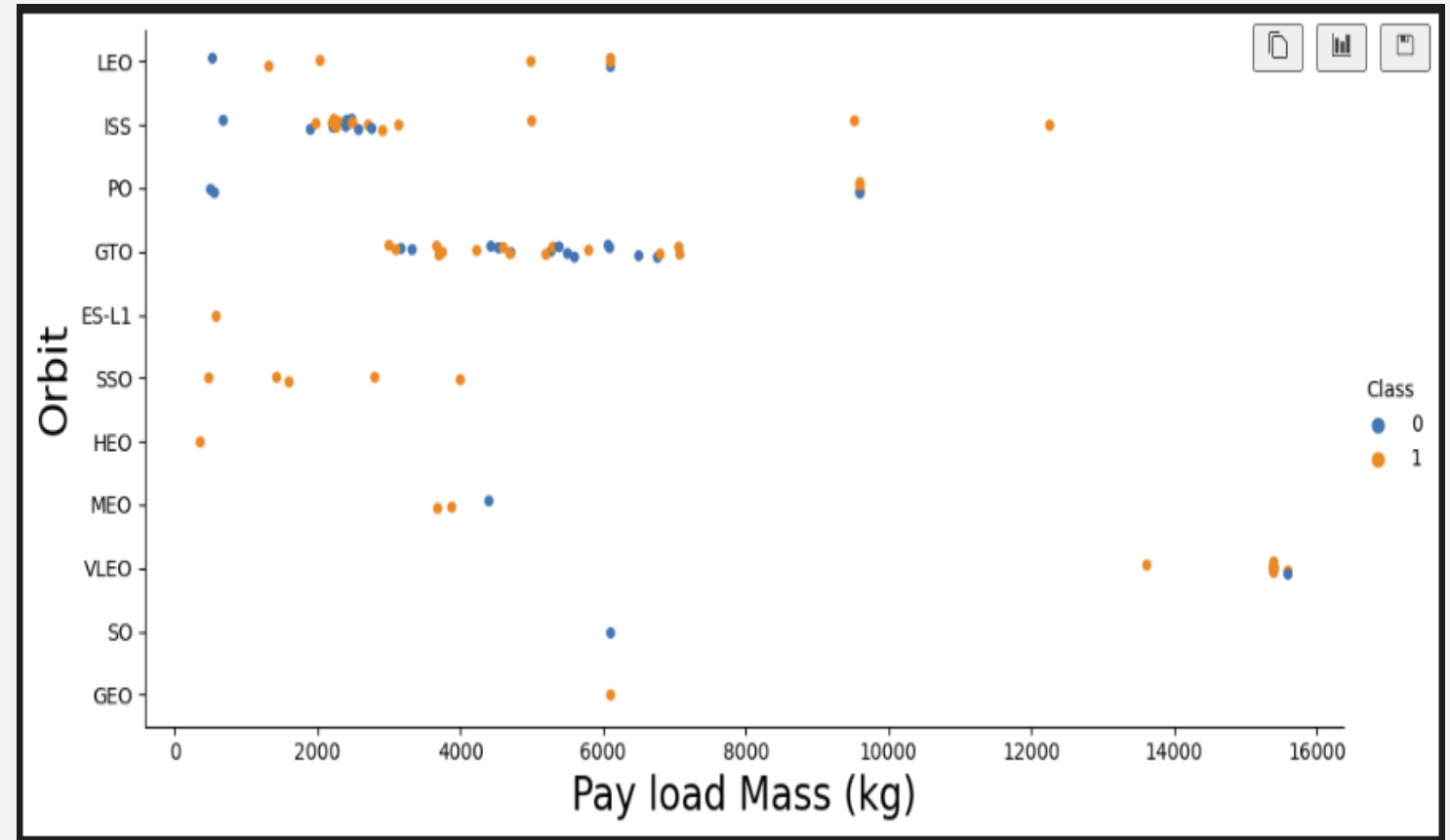
# Flight Number vs. Orbit Type

- Here it can be seen that the type of orbital launches attempted has varied across flight numbers
- Lower flight numbers were primarily associated with LEO, ISS, PO, and GTO orbits
- Later flight numbers with a broader variety of orbit types



# Payload vs. Orbit Type

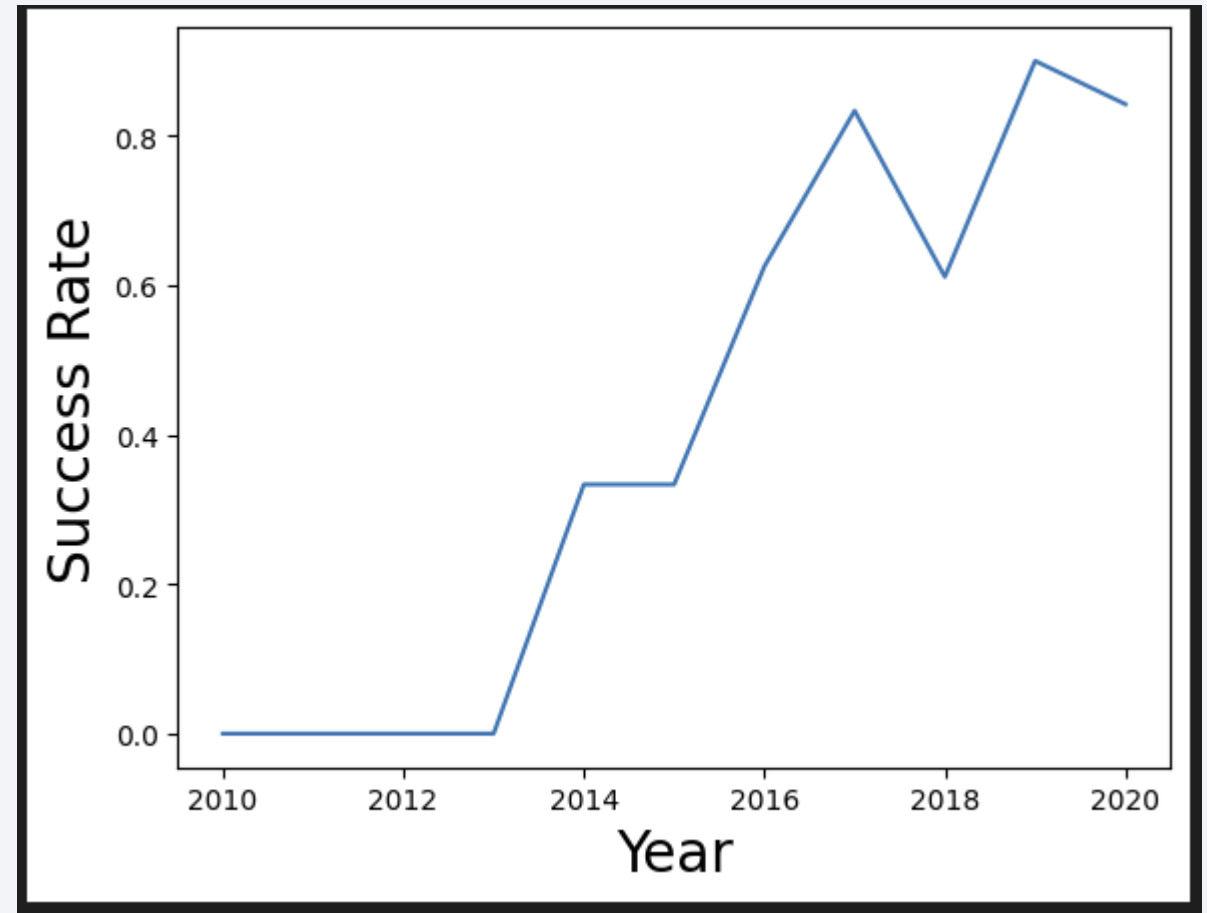
- The mass of payload onboard launches varied by orbit type
- GTO launches carried loads in the range of only ~3,000 to ~7,500 kg, whereas VLEO launches carried much greater loads, >13,000 kg
- Success rates varied



# Launch Success Yearly Trend

---

- Success rates were initially 0 (all failed) in the earlier years but then increased greatly from 2014
- Year over Year changes in success have generally been positive, with the exception of 2018 and 2020 showing negative changes in success rate





# All Launch Site Names

---

- Using the DISTINCT modifier in the SELECT SQL statement on the SPACEX table data collected provided the unique names for the four launch sites from which SpaceX launched rockets

```
%sql SELECT DISTINCT "Launch_Site" FROM SPACEXTBL;
[46] ✓ 0.0s
... * sqlite:///my\_data1.db
Done.

</> Launch_Site
    CCAFS LC-40
    VAFB SLC-4E
    KSC LC-39A
    CCAFS SLC-40
```

# Launch Site Names Begin with 'CCA'

- Using the SQL “LIKE” modifier with the desired string “CCA”, records specific to the launch sites CCAFS LC-40 and CCAFS SLC-40 could be filtered out from the rest of the data

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

```
%sql SELECT * FROM SPACEXTBL WHERE "Launch_Site" like 'CCA%' LIMIT 5;
```

[47] ✓ 0.0s

... \* [sqlite:///my\\_data1.db](#)

Done.

Date	Time (UTC)	Booster_Version	Launch_Site	Payload	PAYLOAD_MASS_KG	C
4/6/2010	18:45:00	F9 v1.0 B0003	CCAFS LC-40	Dragon Spacecraft Qualification Unit	0	
8/12/2010	15:43:00	F9 v1.0 B0004	CCAFS LC-40	Dragon demo flight C1, two CubeSats, barrel of Brouere cheese	0	
22-05-2012	7:44:00	F9 v1.0 B0005	CCAFS LC-40	Dragon demo flight C2	525	
8/10/2012	0:35:00	F9 v1.0 B0006	CCAFS LC-40	SpaceX CRS-1	500	
1/3/2013	15:10:00	F9 v1.0 B0007	CCAFS LC-40	SpaceX CRS-2	677	

# Total Payload Mass

---

- Using the SUM function in SQL, the total payload (for NASA) of all rockets launched by SpaceX was calculated

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

%sql SELECT SUM("PAYLOAD_MASS_KG_ ") FROM SPACEXTBL WHERE "Customer" = 'NASA (CRS)';
[48] ✓ 0.0s

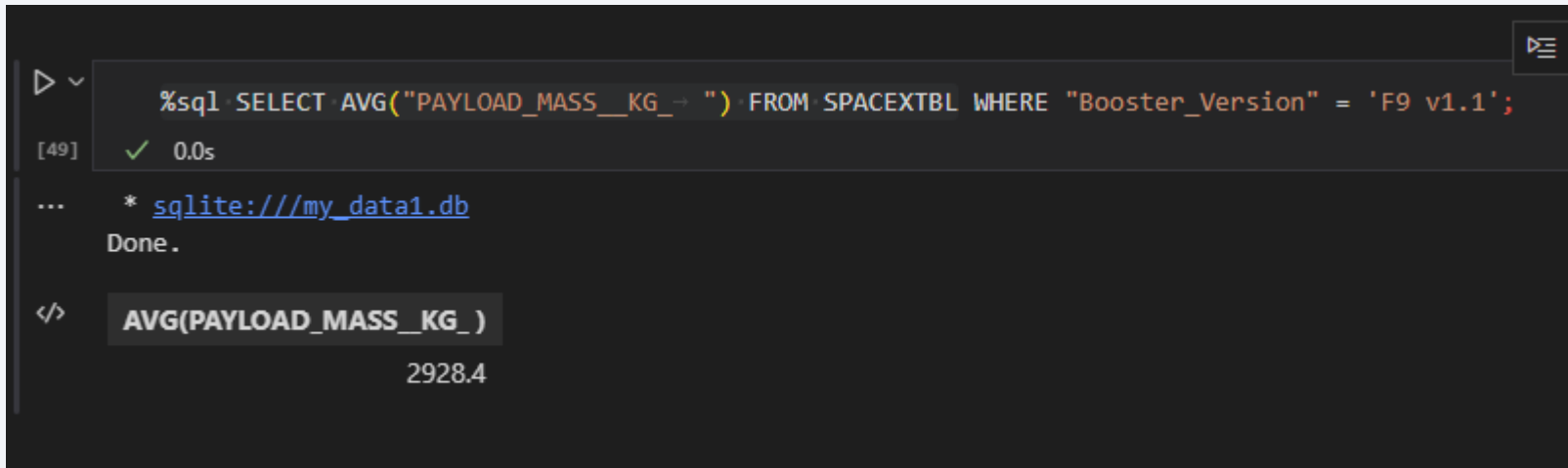
... * sqlite:///my\_data1.db
Done.

SUM(PAYLOAD_MASS_KG_)
45596
```

# Average Payload Mass by F9 v1.1

---

- Using the AVG function in SQL, the average payload carried by F9 v1.1 rockets launched by SpaceX was calculated

A screenshot of a SQL query execution in a terminal window. The query is: `%sql SELECT AVG("PAYLOAD_MASS_KG_") FROM SPACEXTBL WHERE "Booster_Version" = 'F9 v1.1';`. The execution status is shown as `[49] ✓ 0.0s`. Below the query, the database connection is listed as `* sqlite:///my_data1.db` and the status is `Done.`. The result of the query is displayed in a table with one column, `AVG(PAYLOAD_MASS_KG_)`, and one row with the value `2928.4`.

```
%sql SELECT AVG("PAYLOAD_MASS_KG_") FROM SPACEXTBL WHERE "Booster_Version" = 'F9 v1.1';
[49] ✓ 0.0s
... * sqlite:///my_data1.db
Done.
</>
AVG(PAYLOAD_MASS_KG_)
2928.4
```

# First Successful Ground Landing Date

- Using the LIKE function in SQL to find records where the landing outcome contained “success (ground pad), the first such mission launched by SpaceX was found (22-12-2015)

```
%sql SELECT * FROM SPACEXTBL WHERE "Landing_Outcome" like '%ground pad%';
```

✓ 0.0s Python

\* [sqlite:///my\\_data1.db](#)

Done.

Date	Time (UTC)	Booster_Version	Launch_Site	Payload	PAYLOAD_MASS_KG_	Orbit	Customer	Mission_Outcome	Landi
22-12-2015	1:29:00	F9 FT B1019	CCAFS LC-40	OG2 Mission 2 11 Orbcomm-OG2	2034	LEO	Orbcomm	Success	Suc



## Successful Drone Ship Landing with Payload between 4000 and 6000

---

- Using multiple WHERE conditions, we then listed the names of boosters which have successfully landed on drone ship and had payload mass greater than 4000 but less than 6000, as shown below

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", "PAYLOAD_MASS_KG_" FROM SPACEXTBL WHERE "Landing_Outcome" = 'Success (drone ship)'
```

✓ 0.0s

Python

\* [sqlite:///my\\_data1.db](#)

Done.

Booster_Version	PAYLOAD_MASS_KG_
F9 FT B1022	4696
F9 FT B1026	4600
F9 FT B1021.2	5300
F9 FT B1031.2	5200

# Total Number of Successful and Failure Mission Outcomes

---

- Using the COUNT function and the GROUP BY function, we then calculated the total number of successful and failure mission outcomes, as shown below

```
List the total number of successful and failure mission outcomes
```

```
%sql SELECT "Mission_Outcome", COUNT(Mission_Outcome) AS OUTCOME FROM SPACEXTBL group by "Mission_Outcome";
```

[64] ✓ 0.0s Pyth

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

Done.

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

# Boosters Carried Maximum Payload

- Using a subquery to first find the MAX payload mass, the names of the boosters which have carried the maximum payload mass was then found, as shown here (all of those listed had equal payload values)

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

%sql SELECT "Booster_Version" FROM SPACEXTBL where "PAYLOAD_MASS_KG_" = (select max("PAYLOAD_MASS_KG_") FROM SPA

[68] ✓ 0.0s Python
... * sqlite:///my_data1.db
Done.
</>
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

---

- A list the failed landings associated with drone ships, their booster versions, and launch site names for year 2015 can be seen below
- A subquery and WHERE conditions were used to find these records

```
▶ %sql SELECT "Date", "Landing_Outcome", "Booster_Version", "Launch_Site" FROM SPACEXTBL WHERE "Landing_Outcome" =
```

[73] ✓ 0.0s Python

... \* [sqlite:///my\\_data1.db](#)  
Done.

```
</>
```

Date	Landing_Outcome	Booster_Version	Launch_Site
10/1/2015	Failure (drone ship)	F9 v1.1 B1012	CCAFS LC-40
14-04-2015	Failure (drone ship)	F9 v1.1 B1015	CCAFS LC-40

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

---

- A ranking of the count of landing outcomes between the date 2010-06-04 and 2017-03-20, in descending order was then found using WHERE conditions and GROUP BY to sort the records according to outcome

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

```
Landing_Outcome", count(*) as "Count_Success" from SPACEXTBL where "Date" between '04-06-2010' and '20-03-2017'
```

✓ 0.0s

\* [sqlite:///my\\_data1.db](#)

Done.

Landing_Outcome	Count_Success
Success	11
No attempt	6
Success (ground pad)	5
Success (drone ship)	5
Failure (drone ship)	4
Controlled (ocean)	3
Failure	2



A satellite view of Earth from space, showing the curvature of the planet and city lights at night. The background is a deep blue gradient.

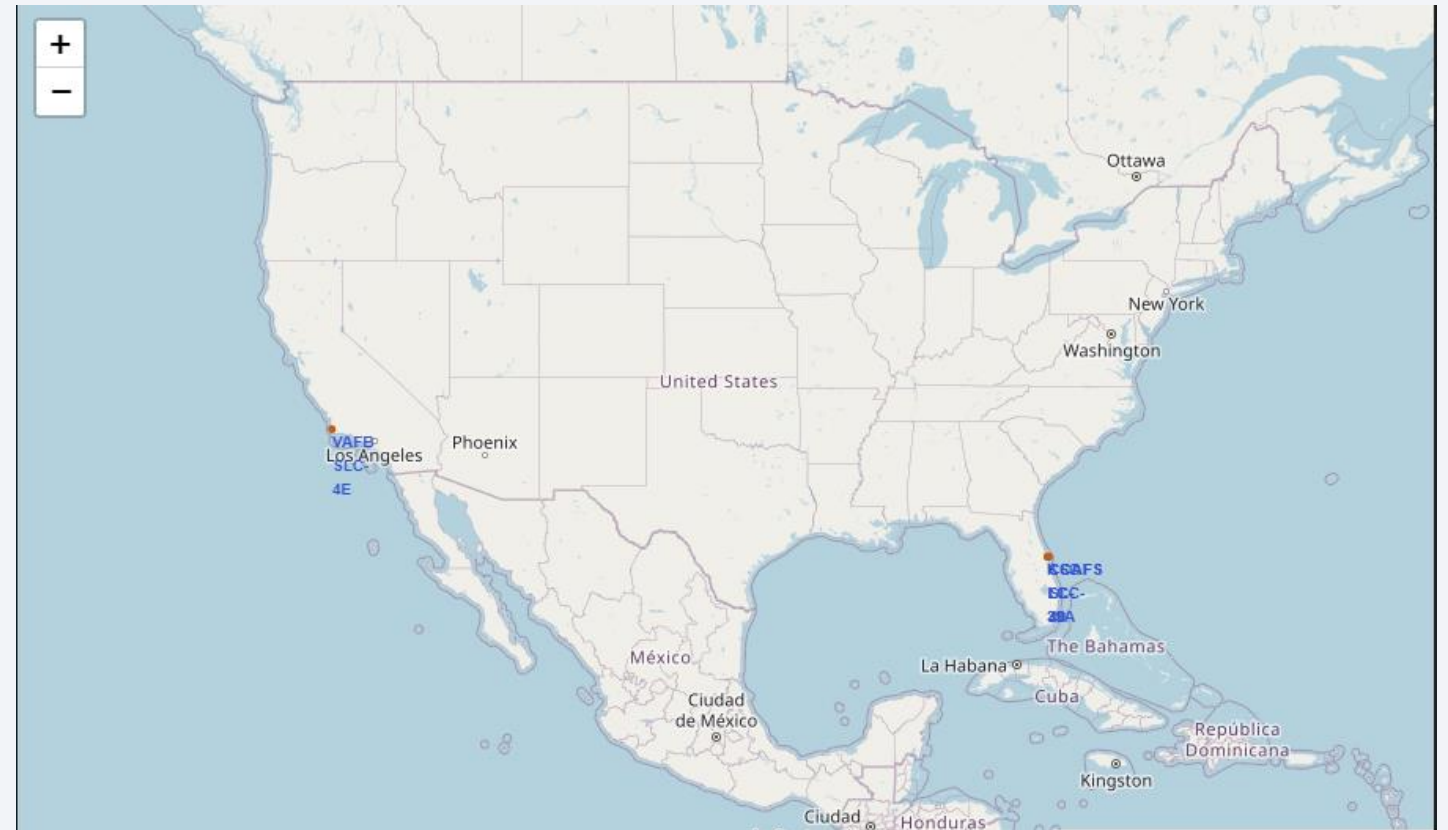
Section 3

# Launch Sites Proximities Analysis

# Map of SpaceX Launch Sites in US

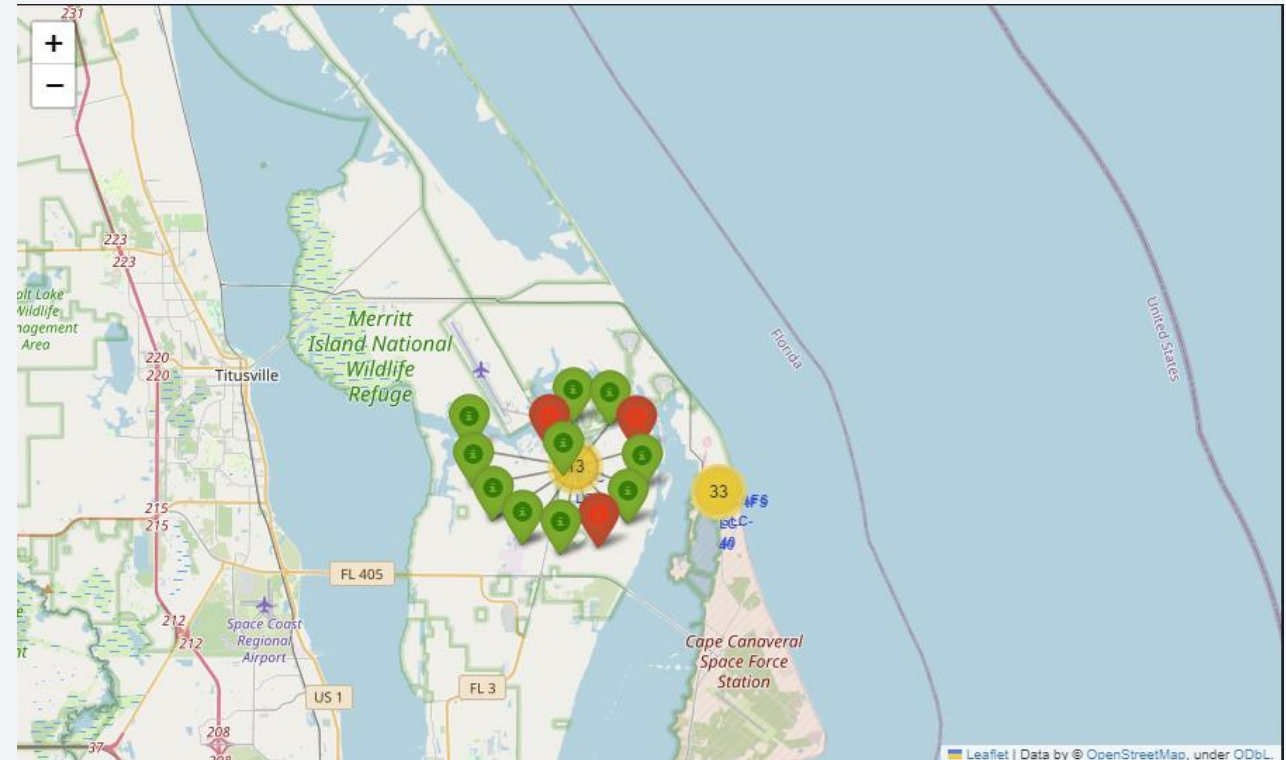
---

- SpaceX launch sites are located on both the east and west coasts of the US, in California and Florida
- Launches from Florida occur at three different sites near the town of Cape Canaveral, while launches from California occur at just a single site near the town of Lompac



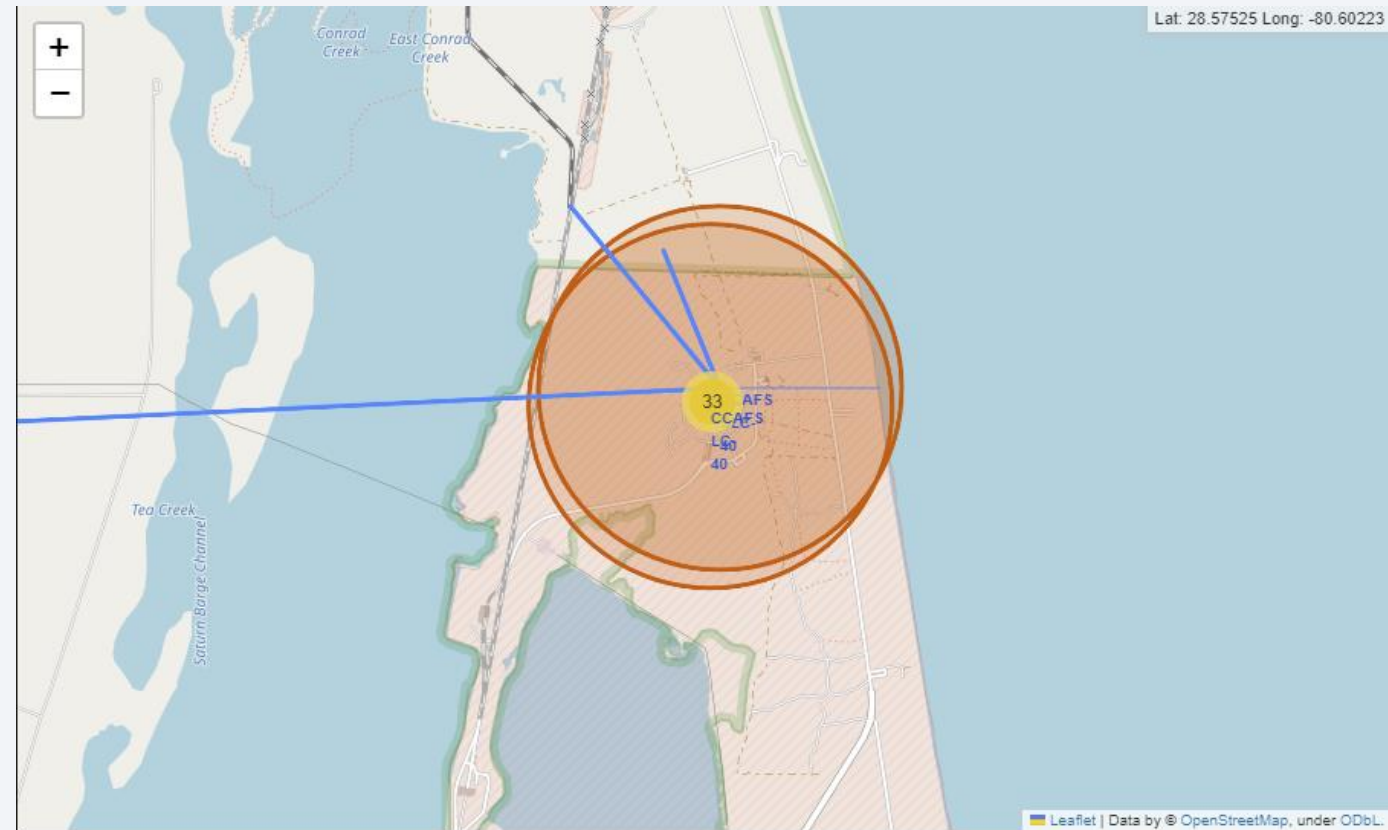
# Map of Florida Launches by Outcome

- Applying differential coloring to markers on the map, we can show how many launches were attempted for each site and display whether the launch was successful or not according to the coloration (green being success and red being failure) to quickly visualize the variable success rate for each site



# Map of Launch Site Proximity to Nearby Resources

- Here can be seen a map with lines plotted on it to display the approximate distance from a particular launch site to nearby points of interest, such as the closest coastline, railroad lines, and the nearest city (Orlando, not visible here)







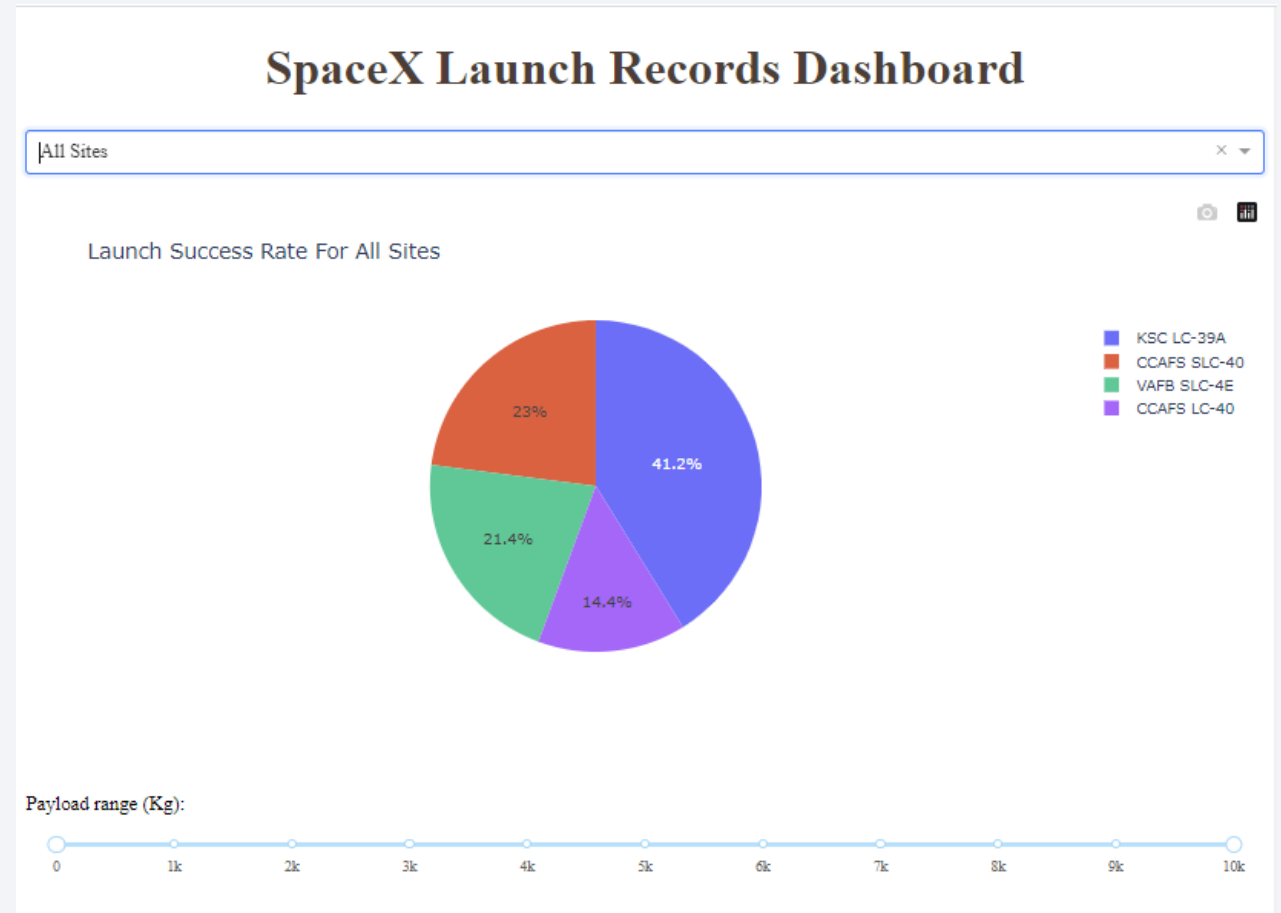
Section 4

# Build a Dashboard with Plotly Dash



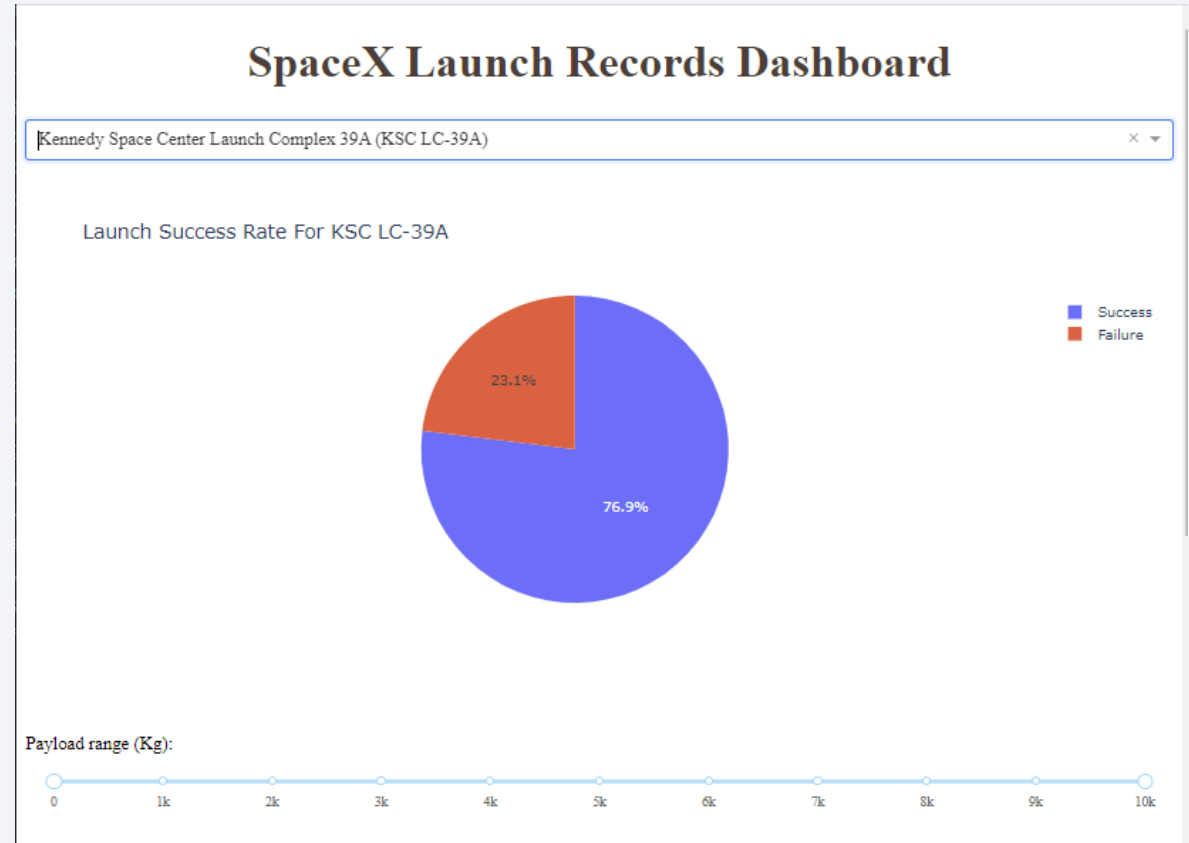
# Proportion of launch success by site

- Using the interactive dashboard built previously, here can be seen a pie chart showing the proportion of launch success counts from each of the four SpaceX launch sites
- It can be seen that the KSC LC-39A launch site contributed the largest number of successful launches to the total



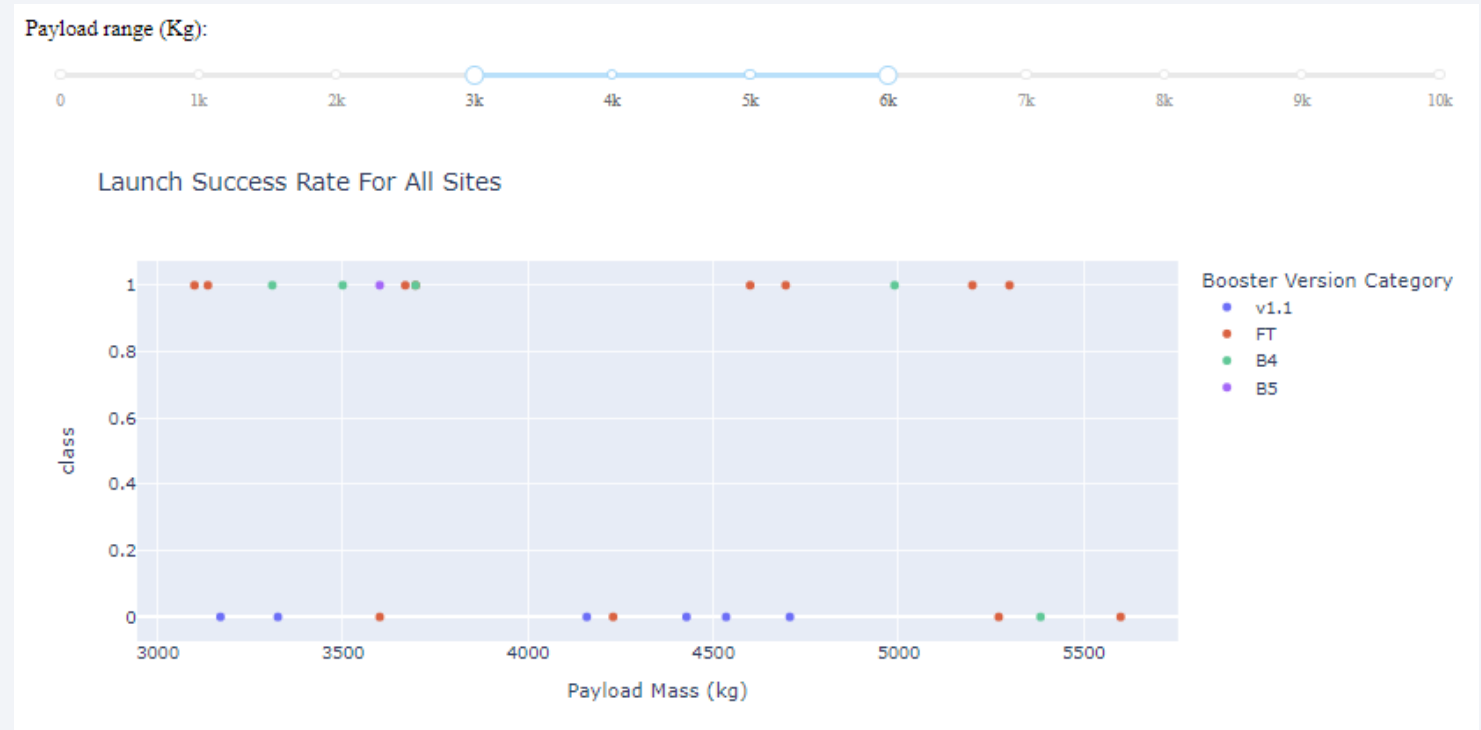
# Kennedy Space Center Launch Complex Success

- The pie chart shown here displays the proportion of success vs failure specific to the Kennedy Space Center Launch Complex site
- This site had the highest rate of success of the four analyzed, with approximately 77% of launches from the facility resulting in successful outcomes



# Scatter plot of success by payload range

- Using the interactive dashboard again to limit the displayed data to only show results from launches with payloads in the range of 3k to 6k, we can see the different outcomes for launches in this range
- There were several failures and no successful launches in the payload range of 4k to 4.5k kg



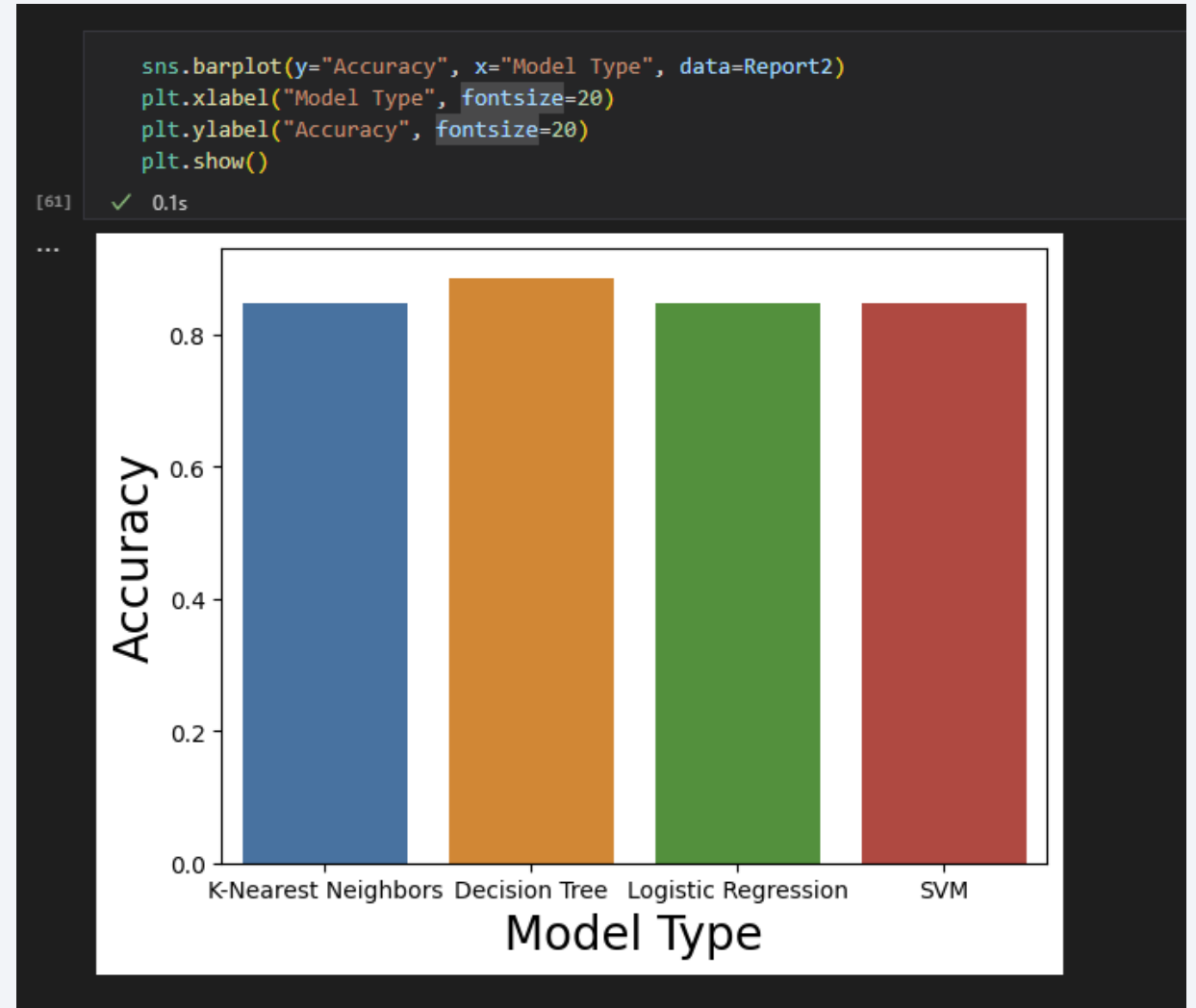


Section 5

# Predictive Analysis (Classification)

# Classification Accuracy

- Based on the accuracy score calculated for each of the four different models employed in the analysis, though each was fairly similar in accuracy, the Decision Tree model beat out the others by a slight margin, as can be seen in the bar chart here



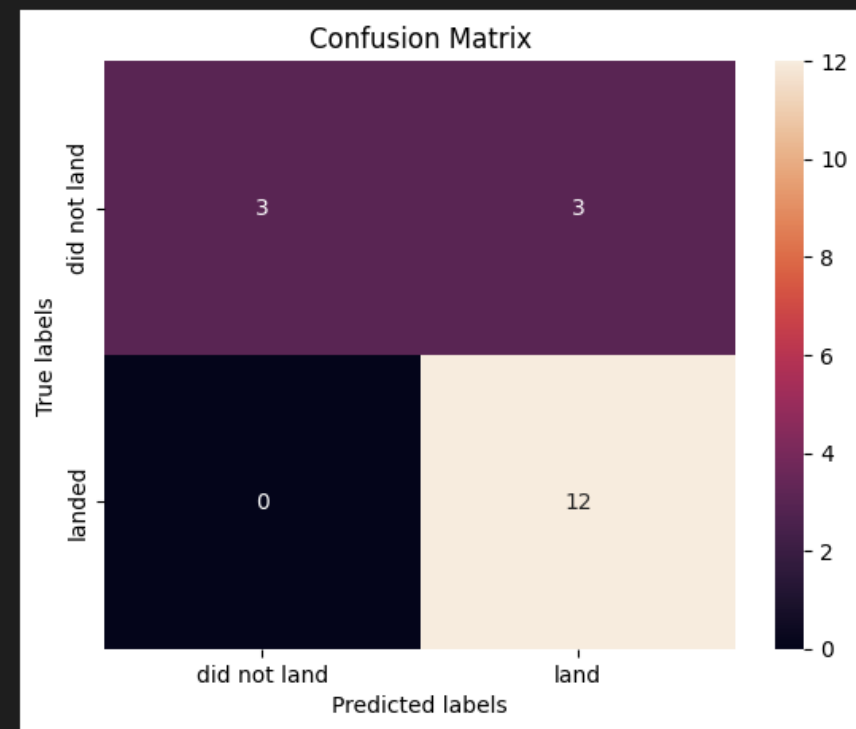


# Confusion Matrix

- The confusion matrix of the best performing model, the classification tree method, can be seen here
- From the 12 results in the test set that were True “landed” the model predicted all 12 of them correctly
- From the 6 results in the test set that were True “did not land” the model only correctly predicted half of them
- False positives are the main issue with the model predictions

We can plot the confusion matrix

```
▶ yhat = tree_cv.predict(X_test)  
plot_confusion_matrix(y_test,yhat)  
[62] ✓ 0.1s
```



# Conclusions

---

- Launch success varies with a number of other features, including launch site, flight number, payload range, and orbit type
- Over time and as flight number increased for each site, the success rate generally increased
- Launches for ES-L1, GEO, HEO, SSO, and VLEO orbits all had high success rates
- The Kennedy Space Center Complex had the highest success rate of all sites
- Using a Decision Tree classification algorithm, we were able to correctly predict the launch outcome on the test data set with approximately 88% accuracy utilizing the available SpaceX data
- Room for improvement remains in the model, specifically to address the issue with false positives, with the model incorrectly classifying failed launches as successes for 50% of the test set



# Appendix

---

```
features_one_hot = pd.get_dummies(features_one_hot)
features_one_hot = pd.concat([features_one_hot, features_one_hot], axis=1)
features_one_hot.head(10)
```

	FlightNumber	PayloadMass
0	1	6104.959412
1	2	525.000000
2	3	677.000000
3	4	500.000000
4	5	3170.000000
5	6	3325.000000
6	7	2296.000000
7	8	1316.000000
8	9	4535.000000
9	10	4428.000000

```
features_one_hot = pd.get_dummies(features_one_hot)
features_one_hot = pd.concat([features_one_hot, features_one_hot], axis=1)
features_one_hot.head(10)
```



Thank you!

