IBM Developer
SKILLS NETWORK

Winning Space Race with Data Science

Andy Moran MSc MPLan
December 2025

Outline

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

Executive Summary

Can we predict how often a SpaceX rocket lands without crashing?

Summary of methodologies:

- Data Collection
- Data Wrangling
- Exploratory data analysis with data visualization and SQL
- Building an interactive map with Folium
- Building a Dashboard with Plotly Dash
- Predictive analysis (classification)

Summary of results:

Logistic Regression is best model • Stable, predictable business outcomes • Interpretable for engineering and regulatory review • Fast predictions for real-time launch decisions • Provides probability scores for risk assessment • Best balance of stability, interpretability, and performance

Decision tree test model unstable. Other models over complicated for marginal gains.

Introduction

This capstone project is part of the IBM Data Science Professional Certificate. The project will show proficiency in data science and machine learning techniques using a real-world data from the SpaceX and their Falcon 9 rocket.

A rival company to SpaceX, Space Y, uses rocket data to find out how often the rocket first stage lands. Space Y also uses this data to determine the net-cost of a Falcon 9 launch. Space Y will use the data to bid against SpaceX for a future rocket launch. A Falcon 9 rocket launch is $62M for a customer. Other companies rocket launches cost $165M or more.

Python Jupyter notebooks are used to perform the data collection and analysis. These Jupyter notebooks and the final *.pdf report are saved here: AndyMoran/Capstone

This report includes data collection methodology, data wrangling, exploratory data analysis (EDA), interactive data visualization, machine learning (ML) classification model development, and model evaluation. Finally, the accuracy of different ML algorithms are compared.

Section 1

Methodology

Methodology

Data collection, wrangling, and formatting: SpaceX API, Web Scraping

Exploratory data analysis (EDA): Pandas, Numpy, SQL

Data visualization: Matplotlib, Seaborn, Folium, Dash

Machine learning prediction: Logistic Regression, Support vector machine (SVM), Decision tree, K-nearest neighbors (KNN)

Data Collection

Data collection combined:

- API requests from SpaceX REST API
- Web Scraping data from a table in SpaceX's Wikipedia entry.

Both collection methods needed for detailed analysis.

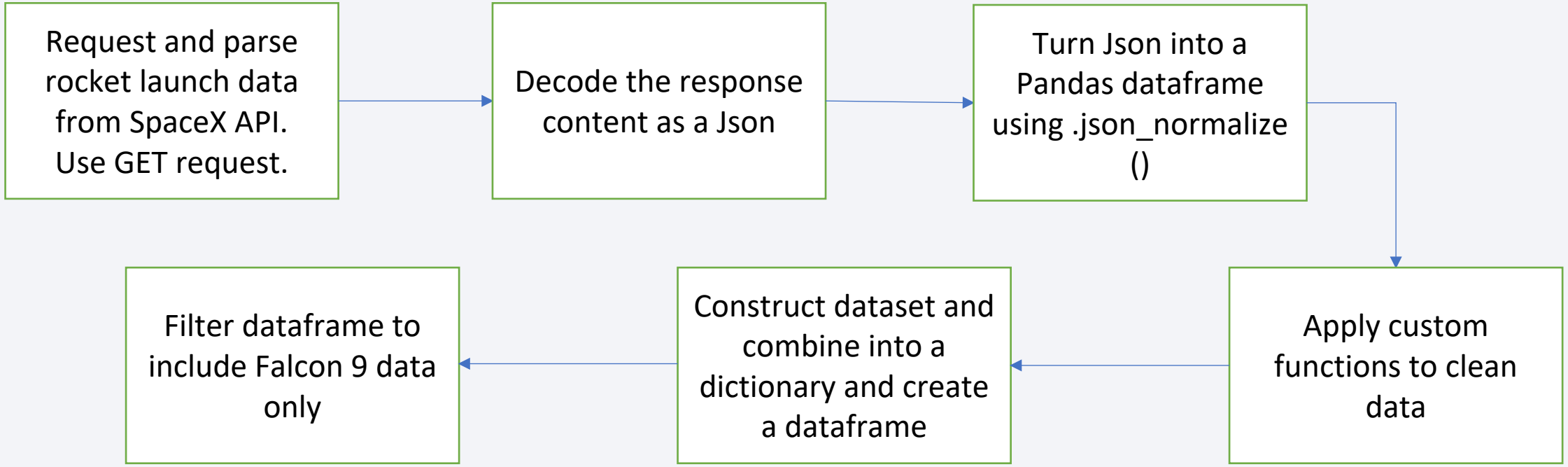
Data Columns from SpaceX REST API:

- FlightNumber, Date, BoosterVersion, PayloadMass, Orbit, LaunchSite, Outcome, Flights, GridFins, Reused, Legs, LandingPad, Block, ReusedCount, Serial, Longitude, Latitude

Data Columns from Wikipedia:

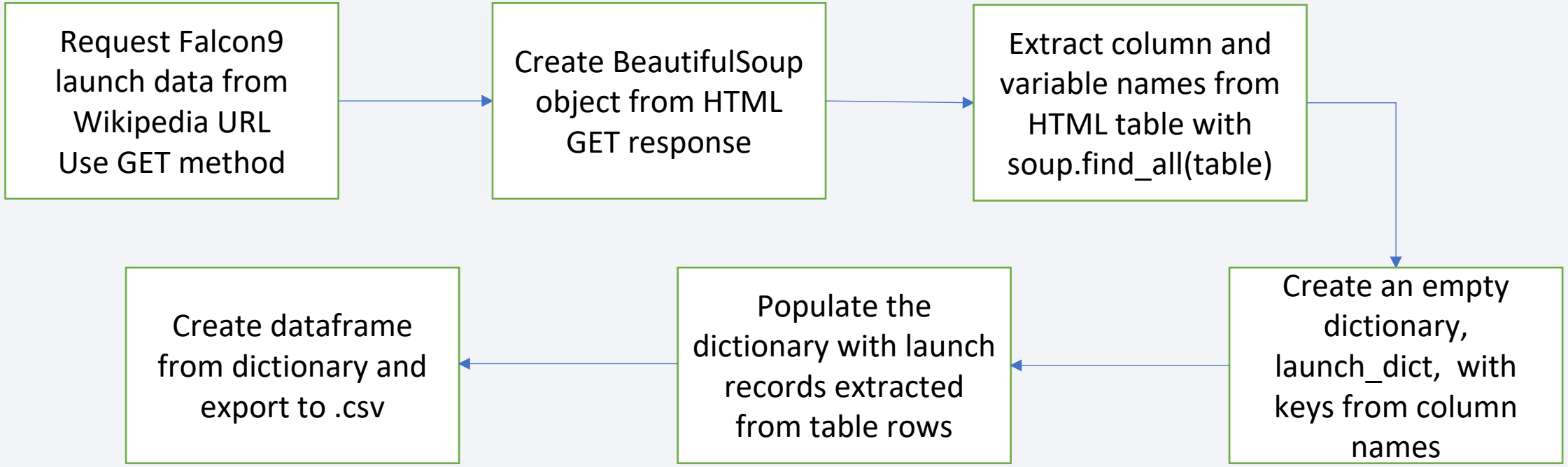
- Flight No., Launch site, Payload, PayloadMass, Orbit, Customer, Launch outcome, Version Booster, Booster landing, Date, Time

Data Collection – SpaceX API



[Link: Data Collection]

Data Collection – Web Scraping



[Link: Web Scraping]

Data Collection/Data Wrangling

Data wrangling transforms raw, messy, and unstructured data into a clean, structured, usable format. As an example we will simply address missing values here, **See: Github.com/Data Wrangling**

Missing values were found in the SpaceX API data:

- LandingPad
- PayloadMass

LandingPad missing values are useful as they demonstrate that the LandingPad was not used.

PayloadMass missing values are problematic and can be addressed by finding np.nan, then replacing with the mean of PayloadMass.

Find missing values in dataset:
data_falcon9.isnull().sum()

↓

Evaluate missing data

↓

Address missing values:

```
# Calculate the mean from data
payload_mean = data_falcon9['PayloadMass'].mean()

# Replace NaNs in data
data_falcon9['PayloadMass'] =
data_falcon9['PayloadMass'].replace(np.nan, payload_mean)
```

Data Wrangling/EDA

Use Data Wrangling and Exploratory Data Analysis (EDA) to find patterns in data and determine labels for training supervised models

Data Wrangling:

- Identify and calculate the percentage of the missing values in each attribute
- Identify which columns are numerical and categorical

EDA

- (1) Calculate the number of launches from each launch site
- (2) Calculate the type and number of each orbit
- (3) Determine mission outcome of orbits
- (4) Create a landing outcome label from Outcome column
- (5) Determine the success rate

[Link: Data Wrangling]

```
(1) launch_counts = df['LaunchSite'].value_counts()
```

```
(2) orbit_counts = df['Orbit'].value_counts()
```

```
(3) landing_outcomes = df['Outcome'].value_counts()
for i,outcome in enumerate(landing_outcomes.keys()):
print(i,outcome)
bad_outcomes=set(landing_outcomes.keys()[[1,3,5,6,7]])
```

```
(4) landing_class = [0 if outcome in bad_outcomes else 1 for
outcome in df['Outcome’]]
```

```
(5) df['Class']=landing_classdf[['Class']].head(8)
df["Class"].mean()
```

EDA with Data Visualization

Categorical Plots:
- FlightNumber and Payload
- FlightNumber and LaunchSite
- Payload Mass and LaunchSite

Bar Charts:
- Orbit Type and Success Rate (Mean)

Scatter Plots:

- FlightNumber and OrbitType
- PayloadMass and OrbitType

Line Chart:

- Year and SuccessRate

Data Visualization:
between Flight Number and Launch Site
Payload Mass and Launch Site
success rate of each orbit type
FlightNumber and Orbit type
Payload Mass and Orbit type
launch success yearly trend

Feature Selection:
- Select features
- Create dummy variables for categorical columns
- Display features
- Cast all columns to numeric values

[Link: EDA and Data Viz]

EDA with SQL

Sql Queries:

- %sql SELECT DISTINCT Launch_Site FROM SPACEXTABLE;
- %sql SELECT * FROM SPACEXTABLE WHERE Launch_Site LIKE 'CCA%' LIMIT 5;
- %%sqlSELECT SUM("PAYLOAD_MASS__KG_") AS Total_Payload_MassFROM SPACEXTABLEWHERE Customer = 'NASA (CRS)’;
- %%sqlSELECT AVG("PAYLOAD_MASS__KG_") AS Avg_Payload_MassFROM SPACEXTABLEWHERE Booster_Version = 'F9 v1.1’;
- %%sqlSELECT MIN(Date) AS First_Successful_Ground_LandingFROM SPACEXTABLEWHERE Landing_Outcome = 'Success (ground pad)’;
- %%sqlSELECT Booster_VersionFROM SPACEXTABLEWHERE Landing_Outcome = 'Success (drone ship)' AND "PAYLOAD_MASS__KG_" > 4000 AND "PAYLOAD_MASS__KG_" < 6000;
- %%sqlSELECT CASE WHEN Mission_Outcome LIKE 'Success%' THEN 'Success' WHEN Mission_Outcome LIKE 'Failure%' THEN 'Failure' ELSE 'Other' END AS Outcome_Category, COUNT(*) AS TotalFROM SPACEXTABLEGROUP BY Outcome_Category;
- %%sqlSELECT Booster_Version, "PAYLOAD_MASS__KG_", Date, CustomerFROM SPACEXTABLEWHERE "PAYLOAD_MASS__KG_" = (SELECT MAX("PAYLOAD_MASS__KG_") FROM SPACEXTABLE);
- %%sqlSELECT strftime('%Y', Date) AS Year, strftime('%m', Date) AS Month_Number, CASE strftime('%m', Date) 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_Name, Landing_Outcome, Booster_Version, Launch_SiteFROM SPACEXTABLEWHERE Landing_Outcome LIKE 'Failure%drone ship%' AND strftime('%Y', Date) = '2015’;
- %%sqlSELECT Landing_Outcome, COUNT(*) AS Outcome_CountFROM SPACEXTABLEWHERE Date BETWEEN '2010-06-04' AND '2017-03-20'GROUP BY Landing_OutcomeORDER BY Outcome_Count DESC;

[Link: EDA with SQL]

Build an Interactive Map with Folium

Step 1: Use Folium to map launch sites and add markers using longitude and latitude to show the geographical location

Step 2: Mark the success and failed launches for each site on the map using colored markers

Step 3: Calculate the distance of launch site to geographical features, railways, highways, coastline population centres

Link: Interactive map with folium

Build a Dashboard with Plotly Dash

Plotly Dash is an interactive application that enables users to perform interactive visual analytics on data in real-time.
This dashboard application contains input components such as a dropdown list and a range slider to interact with a pie chart and a scatter point chart.

Tasks completed in Plotly Dash, add a:

- Launch Site Drop-down Input Component
- Callback function to render success-pie-chart based on selected site dropdown
- Range Slider to Select Payload
- Callback function to render the success-payload-scatter-chart scatter plot

Link: Dashboard created by Plotly Dash

Note: Code works in Jupyter Notebook, but not within Github. Screencaps provided as examples of working Dashboard.

Predictive Analysis (Classification)

- Split data into training and test data set

Predict on Test Data

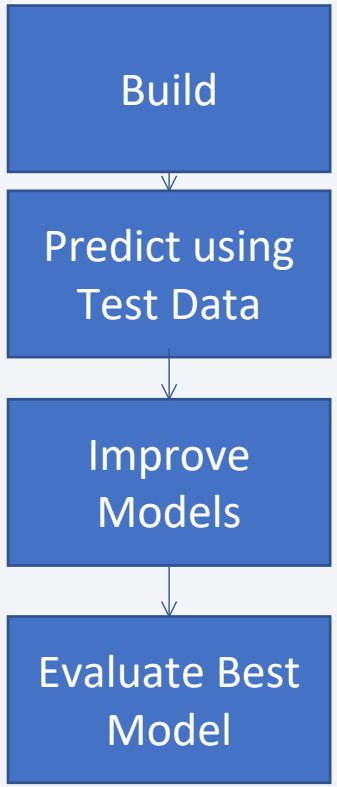
- Create logistic regression/SVM/Decision Tree/KNN objects, create GridSearchCV
- Display best parameters and accuracy on the validation data
- Check accuracy for each model using test data: score method
- Plot Confusion Matrix

Improve

- Feature Engineering
- Algorithm Tuning

Evaluate Performance of Prediction Models -Score

- Best model wins



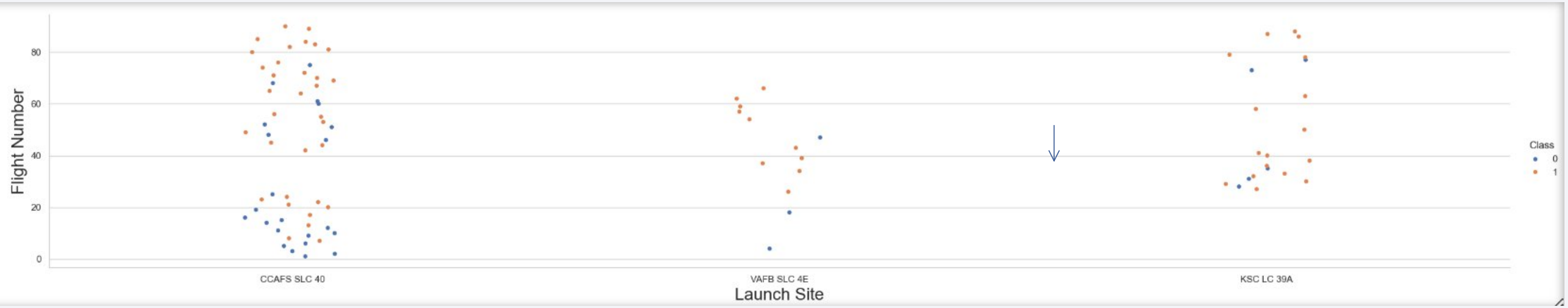
Results

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

Section 2

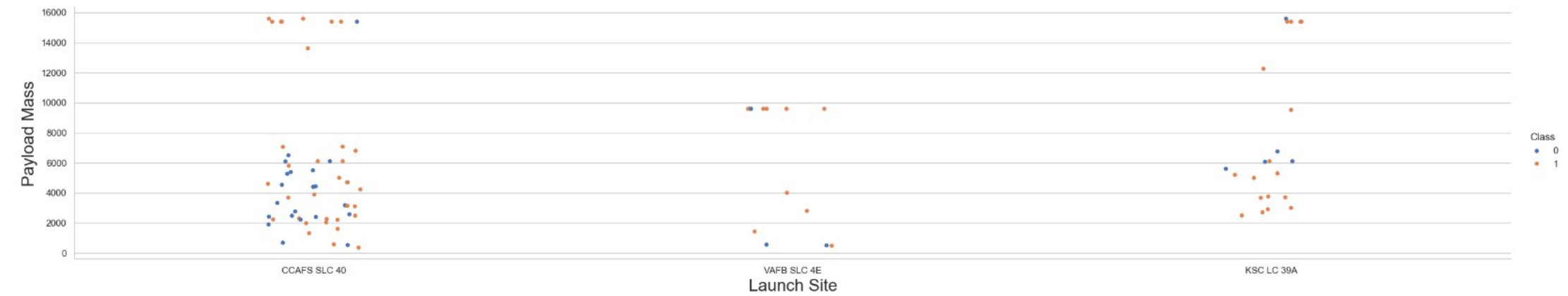
Insights drawn from EDA

Flight Number vs. Launch Site



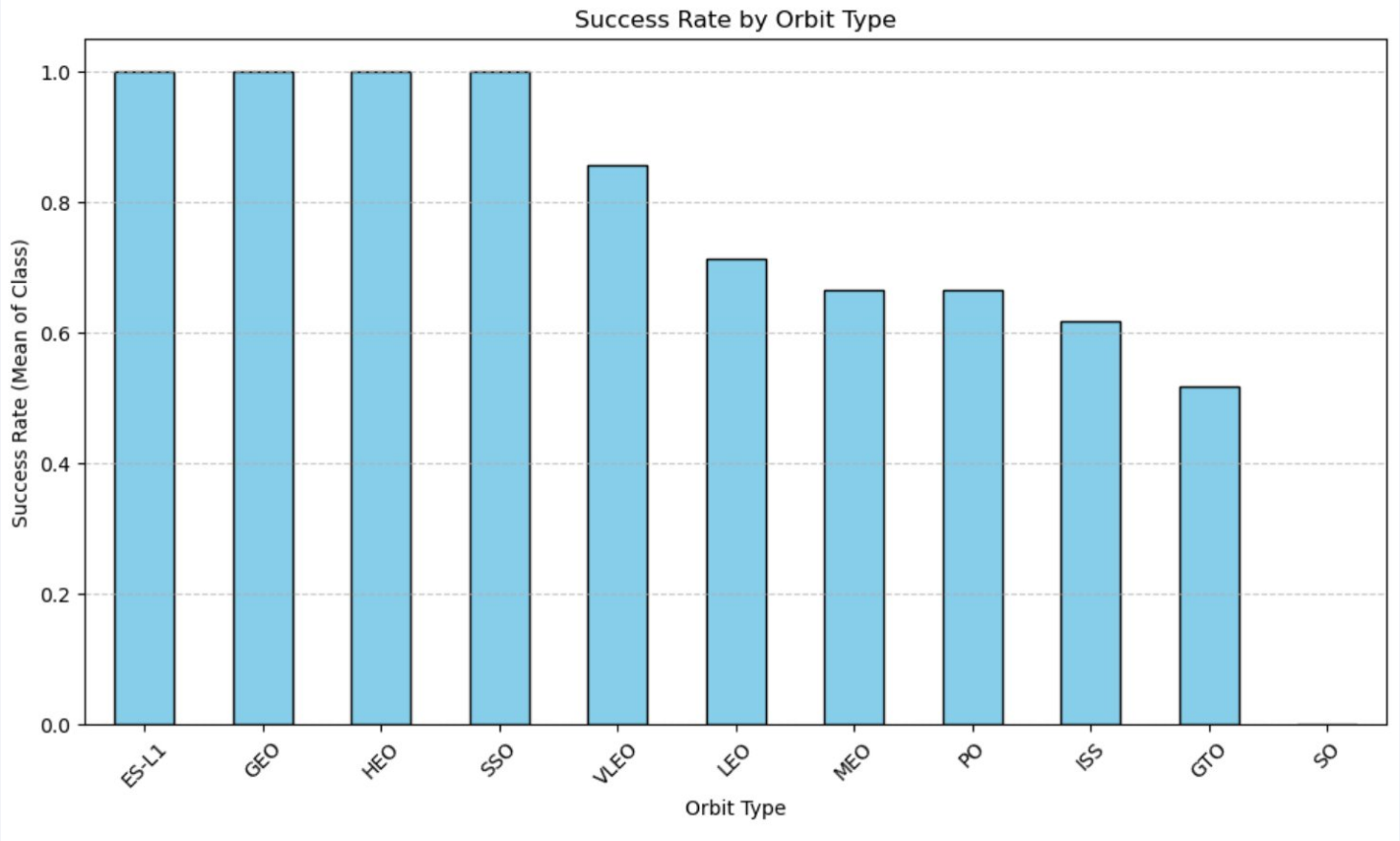
The more flights at a launch site the better the success rate

Payload vs. Launch Site



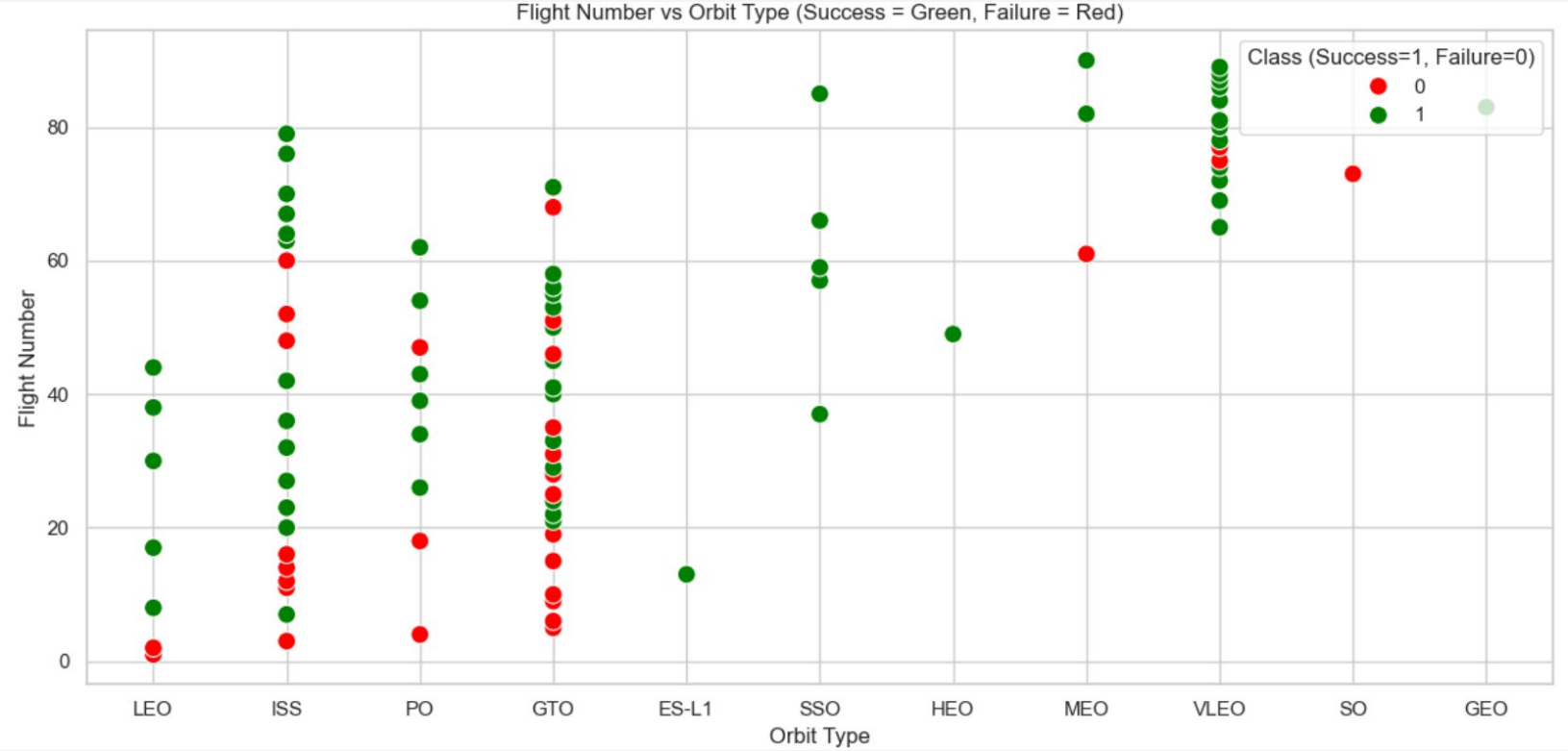
For the VAFB-SLC launch site there are no rockets launched with a payload mass (greater than 10,000)

Success Rate vs. Orbit Type



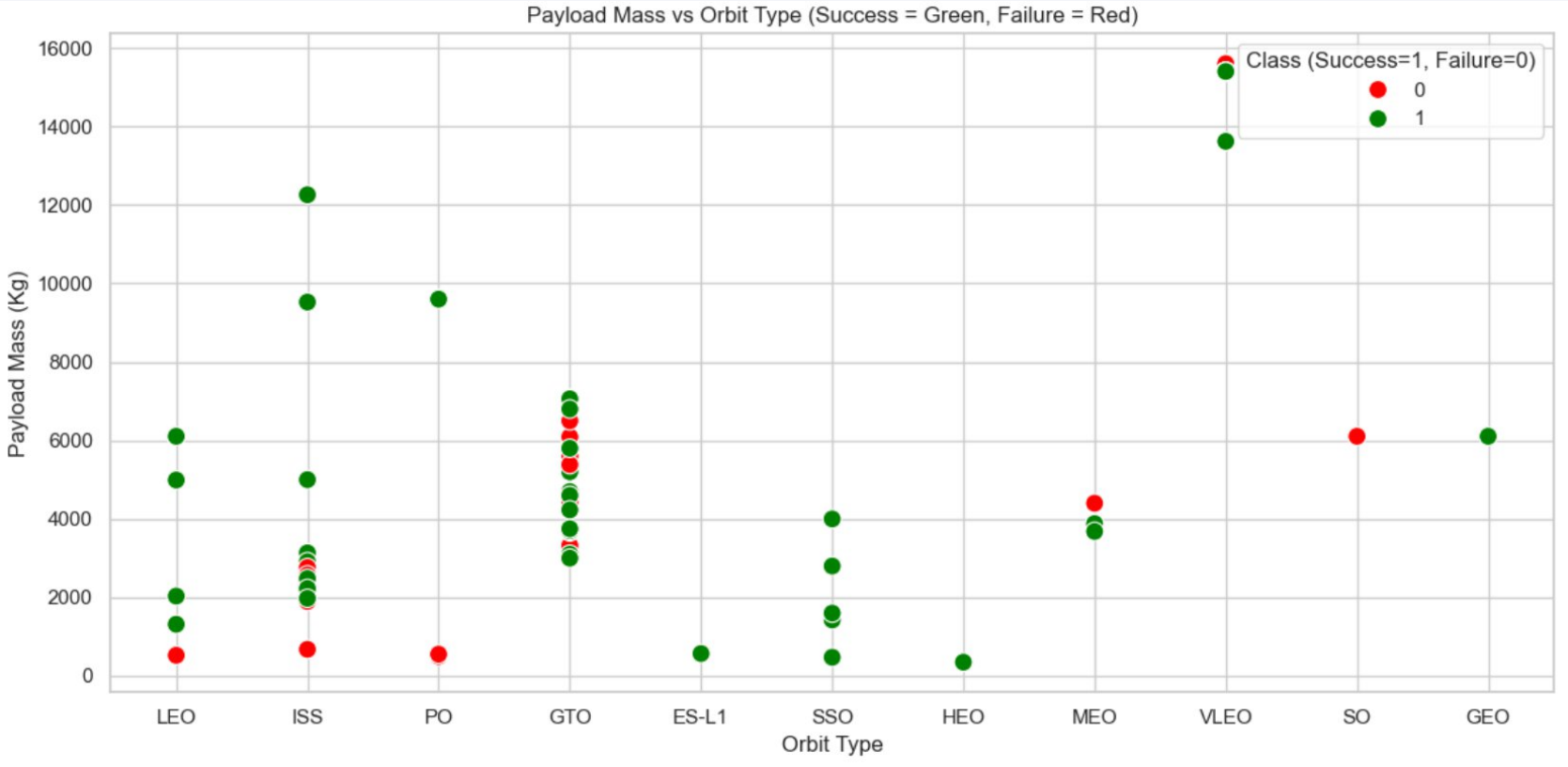
Analyze the plotted bar chart to identify which orbits have the highest success rates.

Flight Number vs. Orbit Type



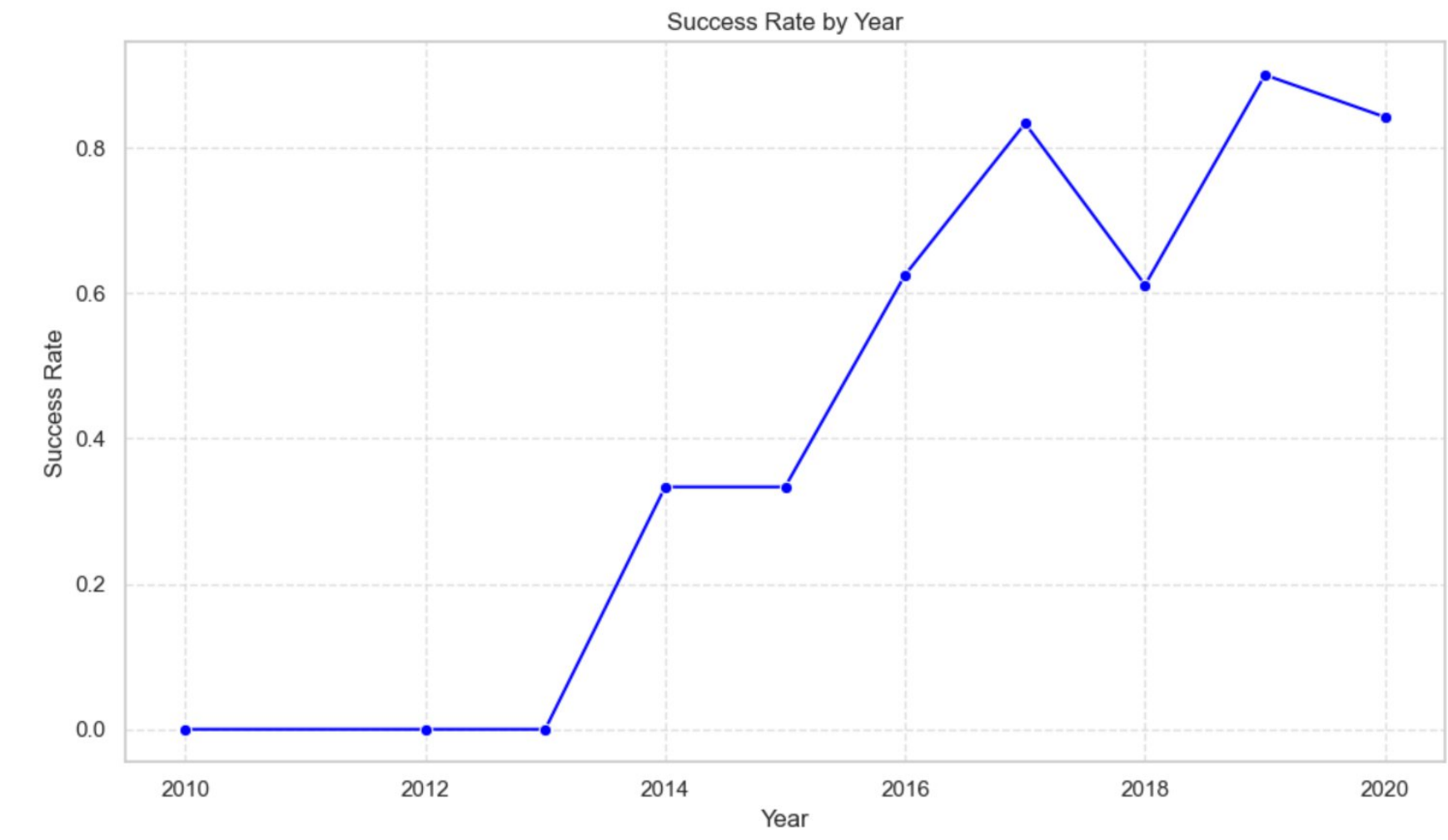
You can observe that in the LEO orbit, success seems to be related to the number of flights. Conversely, in the GTO orbit, there appears to be no relationship between flight number and success.

Payload vs. Orbit Type



With heavy payloads the successful landing or positive landing rate are more for Polar,LEO and ISS.
However, for GTO, it's difficult to distinguish between successful and unsuccessful landings as both outcomes are present.

Launch Success Yearly Trend



The success rate from 2013 kept increasing till 2019

All Launch Site Names

```
%sql SELECT DISTINCT Launch_Site FROM SPACEXTABLE;
```

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

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

USE DISTINCT to find Unique values in the Launch_Site column in the SPACEXTABLE

Launch Site Names Begin with 'CCA'

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

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

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

Use LIKE ‘CCA%’ and LIMIT to display 5 records where launch sites begin with the string 'CCA'.

Total Payload Mass

```
%%sql
SELECT SUM("PAYLOAD_MASS__KG_") AS Total_Payload_Mass
FROM SPACEXTABLE
WHERE Customer = 'NASA (CRS)';
```

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

Total_Payload_Mass
45596

Using SUM adds all the totals in the column “PAYLOAD_MASS_KG_”. The WHERE clause filters the dataset to only perform calculations on Customer “NASA (CRS)”.

Average Payload Mass by F9 v1.1

```
%%sql
SELECT AVG("PAYLOAD_MASS__KG_") AS Avg_Payload_Mass
FROM SPACEXTABLE
WHERE Booster_Version = 'F9 v1.1';
```

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

Avg_Payload_Mass
2928.4

AVG
works out the average in column "PAYLOAD_MASS_KG_ ".
WHERE
filters the dataset to only perform calculations on Booster_version = "F9 v1.1"

First Successful Ground Landing Date

```
%%sql
SELECT MIN(Date) AS First_Successful_Ground_Landing
FROM SPACEXTABLE
WHERE Landing_Outcome = 'Success (ground pad)';
```

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

First_Successful_Ground_Landing
2015-12-22

MIN(Date)
finds the earliest (minimum) date in the dataset.
AS First_Successful_Ground_Landing
renames the output column as First_Successful_Ground_Landing.
FROM SPACEXTABLE
This specifies the table
WHERE Landing_Outcome = 'Success (ground pad)'
This filters the rows, only rows where the landing outcome was “Success (ground pad)” are considered. Other outcomes (like “Failure” or “Success (drone ship)”) are ignored.

Successful Drone Ship Landing with Payload between 4000 and 6000

```
%%sql
SELECT Booster_Version
FROM SPACEXTABLE
WHERE Landing_Outcome = 'Success (drone ship)'
  AND "PAYLOAD_MASS__KG_" > 4000
  AND "PAYLOAD_MASS__KG_" < 6000;
```

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

Booster_Version
F9 FT B1022
F9 FT B1026
F9 FT B1021.2
F9 FT B1031.2

SELECT Booster_Version
Retrieves the values in the Booster_Version column.
FROM SPACEXTABLE
Specifies table
WHERE Landing_Outcome = 'Success (drone ship)'
Filters rows where rocket successfully landed drone ship. Other landing outcomes (like "Failure" or "Success (ground pad)") are excluded.
AND "PAYLOAD_MASS__KG_" > 4000
Filters rows where the payload mass is greater than 4000 kg.
AND "PAYLOAD_MASS__KG_" < 6000
Filters rows where the payload mass is less than 6000 kg.

Total Number of Successful and Failure Mission Outcomes

```
%%sql
SELECT
  CASE
    WHEN Mission_Outcome LIKE 'Success%' THEN 'Success'
    WHEN Mission_Outcome LIKE 'Failure%' THEN 'Failure'
    ELSE 'Other'
  END AS Outcome_Category,
  COUNT(*) AS Total
FROM SPACEXTABLE
GROUP BY Outcome_Category;
```

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

Outcome_Category	Total
Failure	1
Success	100

CASE ... END AS Outcome_Category
This is SQL's version of if/else logic.
It looks at the value in the Mission_Outcome column and assigns it to a category:
If Mission_Outcome starts with "Success" label as "Success".
If Mission_Outcome starts with "Failure" label as "Failure".
Otherwise label as "Other".
The result of this logic is given the alias Outcome_Category.
COUNT(*) AS Total
Counts how many rows fall into each category, label as Total.
FROM SPACEXTABLE
Specifies the table.
GROUP BY Outcome_Category
Groups rows by the new categories (Success, Failure, Other).
Ensures that COUNT(*) is calculated separately for each category.

Boosters Carried Maximum Payload

```
%%sql
SELECT Booster_Version, "PAYLOAD_MASS__KG_", Date, Customer
FROM SPACEXTABLE
WHERE "PAYLOAD_MASS__KG_" = (
    SELECT MAX("PAYLOAD_MASS__KG_")
    FROM SPACEXTABLE
);
```

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

Booster_Version	PAYLOAD_MASS__KG_	Date	Customer
F9 B5 B1048.4	15600	2019-11-11	SpaceX
F9 B5 B1049.4	15600	2020-01-07	SpaceX
F9 B5 B1051.3	15600	2020-01-29	SpaceX
F9 B5 B1056.4	15600	2020-02-17	SpaceX
F9 B5 B1048.5	15600	2020-03-18	SpaceX
F9 B5 B1051.4	15600	2020-04-22	SpaceX
F9 B5 B1049.5	15600	2020-06-04	SpaceX, Planet Labs
F9 B5 B1060.2	15600	2020-09-03	SpaceX
F9 B5 B1058.3	15600	2020-10-06	SpaceX
F9 B5 B1051.6	15600	2020-10-18	SpaceX
F9 B5 B1060.3	15600	2020-10-24	SpaceX
F9 B5 B1049.7	15600	2020-11-25	SpaceX

SELECT Booster_Version, "PAYLOAD_MASS__KG_", Date, Customer
This chooses columns to display in the final result
FROM SPACEXTABLE
Specifies the table
WHERE "PAYLOAD_MASS__KG_" = (...)
This filters rows based on a condition.
Only rows where the payload mass equals the result of the subquery are included.
Subquery: **SELECT MAX("PAYLOAD_MASS__KG_") FROM SPACEXTABLE**
This finds the maximum payload mass in the entire table.

2015 Launch Records

```
SELECT
  strftime('%Y', Date) AS Year,
  strftime('%m', Date) AS Month_Number,
  CASE strftime('%m', Date)
    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_Name,
  Landing_Outcome,
  Booster_Version,
  Launch_Site
FROM SPACEXTABLE
WHERE Landing_Outcome LIKE 'Failure%drone ship%'
  AND strftime('%Y', Date) = '2015';
```

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

Year	Month_Number	Month_Name	Landing_Outcome	Booster_Version	Launch_Site
2015	01	January	Failure (drone ship)	F9 v1.1 B1012	CCAFS LC-40
2015	04	April	Failure (drone ship)	F9 v1.1 B1015	CCAFS LC-40

strftime('%Y', Date) AS Year
Extracts the **year** from the Date column using SQLite's strftime function.
Example: if Date = '2015-06-28', this returns '2015'.
strftime('%m', Date) AS Month_Number
Extracts the **month number** (01–12) from the Date.
Example: '2015-06-28' '06'.
CASE strftime('%m', Date) ... END AS Month_Name
Converts the numeric month into a **human-readable month name**.
Example: '06' , 'June'.
This is done with a CASE statement (SQL's version of if/else).
Landing_Outcome, Booster_Version, Launch_Site
These columns are selected directly to give more context about the mission.
FROM SPACEXTABLE
Specifies the table being queried.
WHERE Landing_Outcome LIKE 'Failure%drone ship%'
Filters rows to only those where the landing outcome starts with **"Failure" and contains "drone ship".**
Example matches: "Failure (drone ship)", "Failure due to weather (drone ship)".
AND strftime('%Y', Date) = '2015'
Further filters rows to only those from the year **2015**.

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

```
%%sql
SELECT Landing_Outcome, COUNT(*) AS Outcome_Count
FROM SPACEXTABLE
WHERE Date BETWEEN '2010-06-04' AND '2017-03-20'
GROUP BY Landing_Outcome
ORDER BY Outcome_Count DESC;
```

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

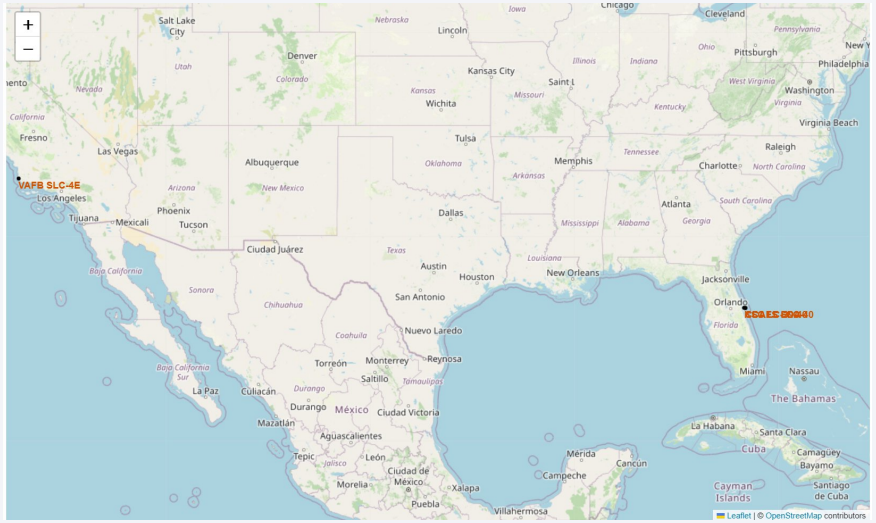
Landing_Outcome	Outcome_Count
No attempt	10
Success (drone ship)	5
Failure (drone ship)	5
Success (ground pad)	3
Controlled (ocean)	3
Uncontrolled (ocean)	2
Failure (parachute)	2
Precluded (drone ship)	1

SELECT Landing_Outcome, COUNT(*) AS Outcome_Count
Retrieves the type of landing result (e.g., "Success (drone ship)", "Failure", etc.).
COUNT(*) AS Outcome_Count counts how many rows (missions) fall into each landing outcome category and the alias makes the count column easier to read.
FROM SPACEXTABLE
Specifies the table
WHERE Date BETWEEN '2010-06-04' AND '2017-03-20'
Filters the rows to only include missions that happened between June 4, 2010 and March 20, 2017 (inclusive). Any mission outside this date range is ignored.
GROUP BY Landing_Outcome
Groups the filtered rows by landing outcome.
This ensures that COUNT(*) is calculated separately for each outcome category.
ORDER BY Outcome_Count DESC
Sorts the results by the count, in descending order.
The most frequent landing outcome appears first.

Section 3

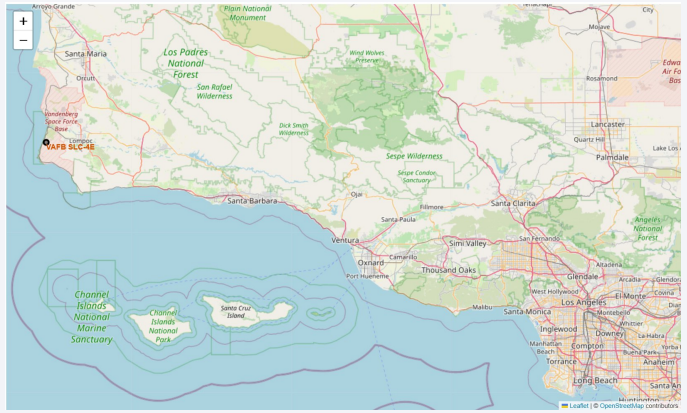
Launch Sites Proximities Analysis

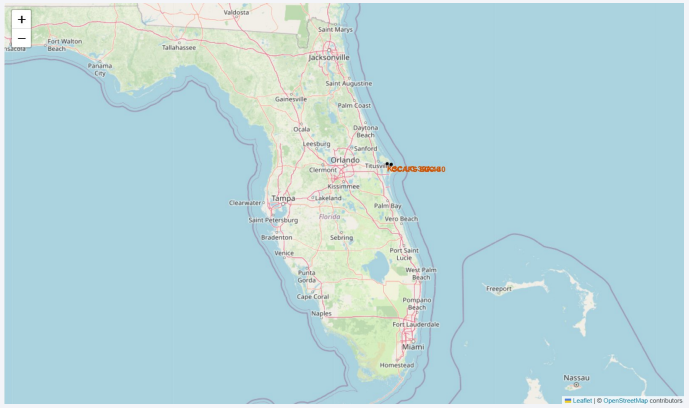
USA Launch Sites — California and Florida



Launch sites are sited close to the Equator to enable easier launches from Earth.
Launch sites are close to the coast to minimize the risk of catastrophic failure near large populations

[DataScienceIBM/Capstone-SpaceX/IBM-spaceX-launch-site-location.ipynb at main · AndyMoran/DataScienceIBM]

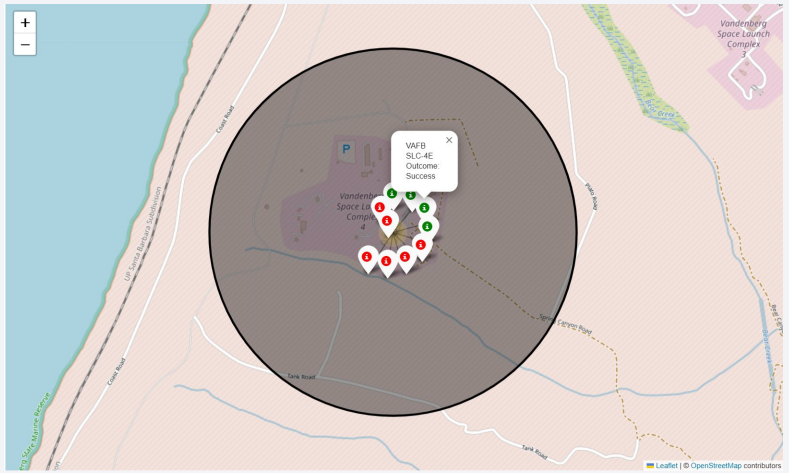


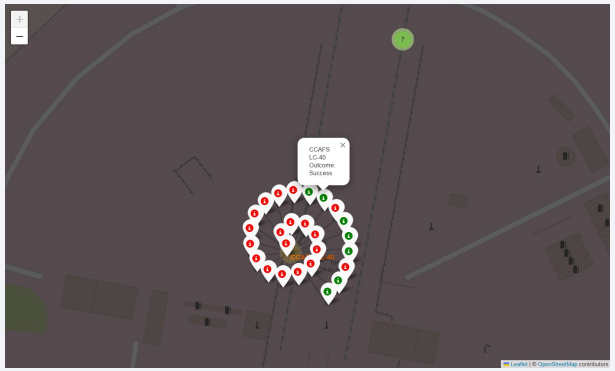


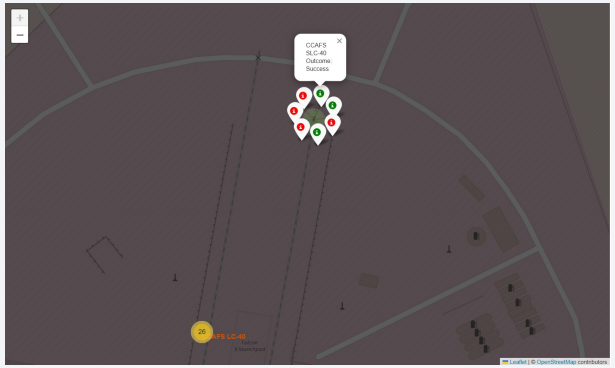
Launch Success and Failures

Total Launches

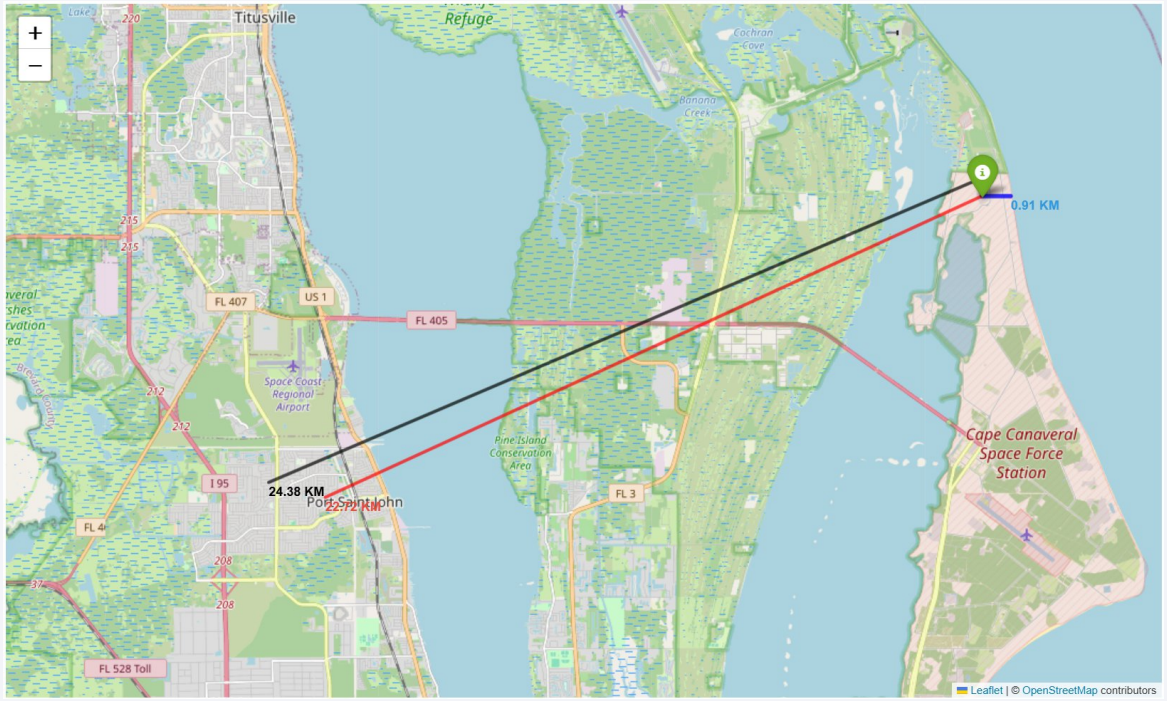
Success and Failure by Launch Site







Launch Site Proximity to Infrastructure and Sea



Distance from Falcon 9 launch site in Florida

Blue line indicates distance from coastline

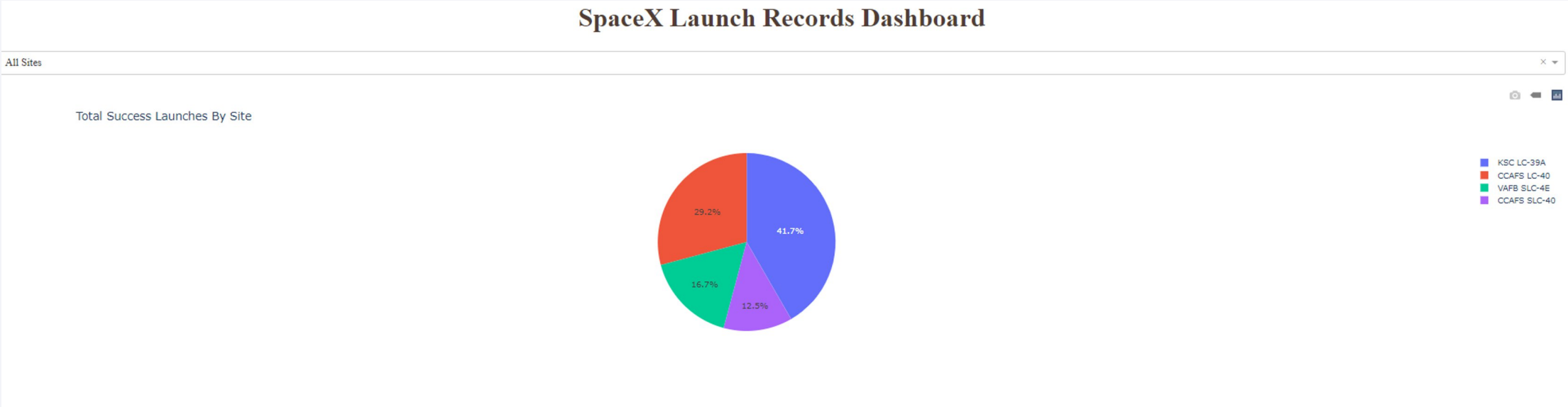
Red line indicates distance to nearest population centre —Port Saint John

Black line indicates approx. distance to the I95 highway

Section 4

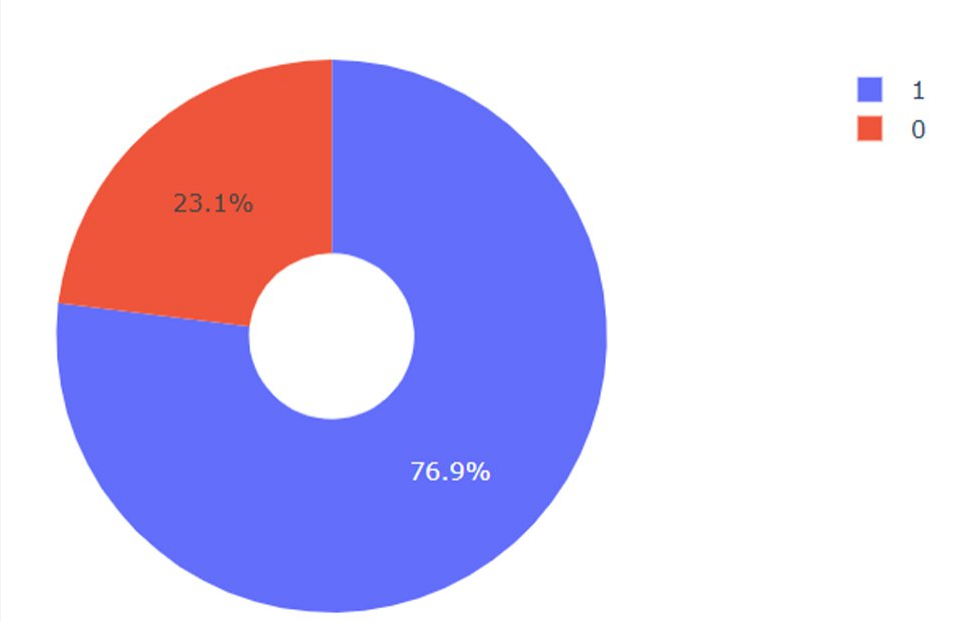
Build a Dashboard with Plotly Dash

Total Launch Success by Site — Dashboard



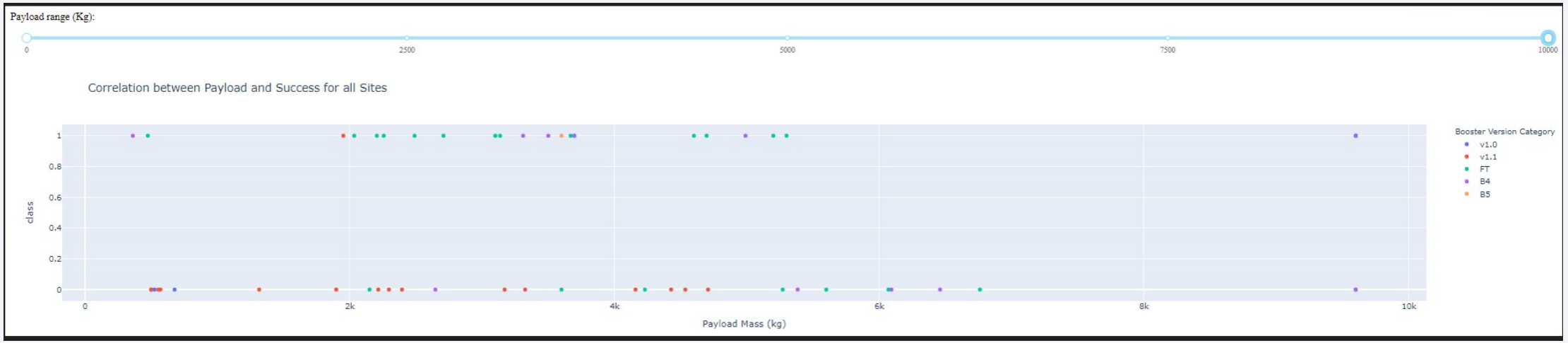
KSC LC-39A is the most successful launch site with a 41.7% success rate
CCAFS LC40 is the second most successful launch site with a 29.2% success rate

Highest Site Success Ratio — KSC LC-39A



Site KSC LC-39A achieved a 76.9% launch success and a 23.1% failure rate

Payload vs. Launch Outcome for all Sites

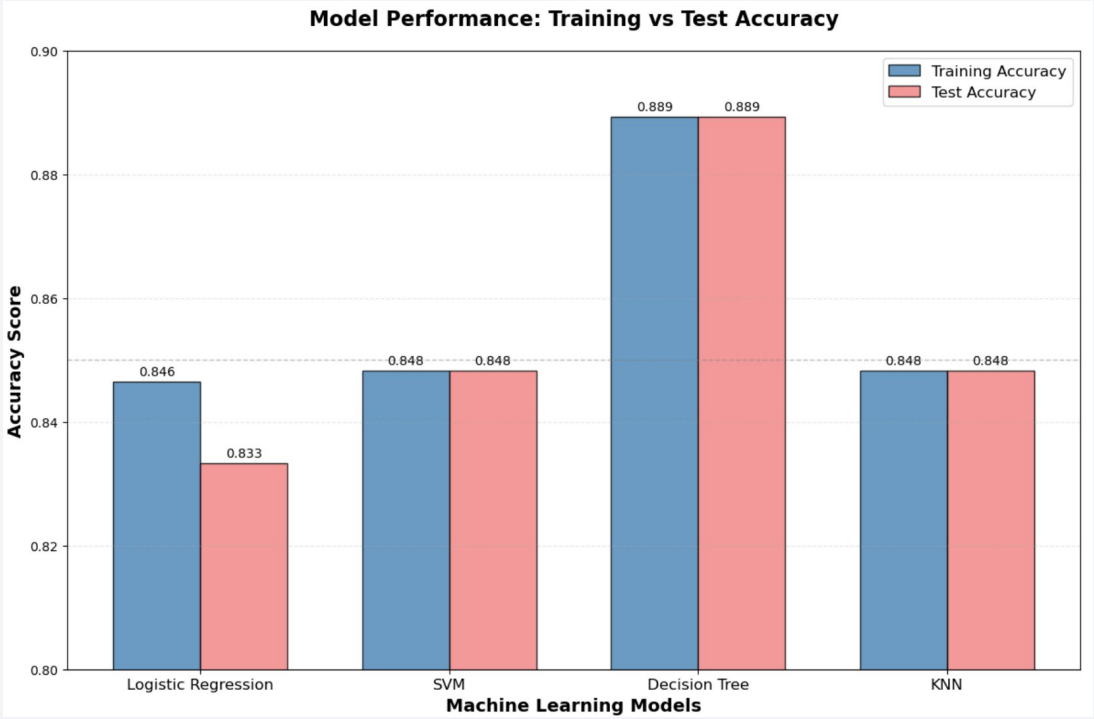


Highest success rate for payloads is between 2000 and 5500 Kgs

Section 5

Predictive Analysis (Classification)

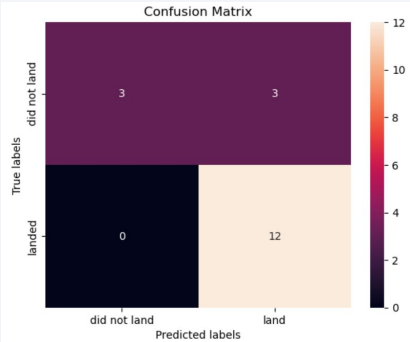
Classification Accuracy



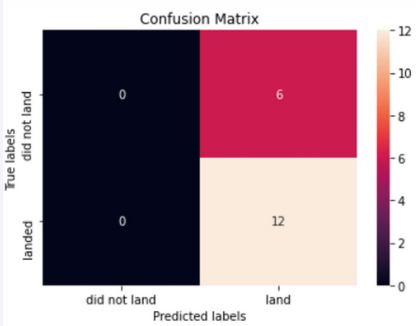

```
============================================================
MODEL PERFORMANCE SUMMARY
============================================================
              Model  Training Accuracy  Test Accuracy  Difference (Train-Test)
Logistic Regression           0.846429       0.833333                 0.013095
                SVM           0.848214       0.848214                 0.000000
      Decision Tree           0.889286       0.889286                 0.000000
                KNN           0.848214       0.848214                 0.000000

============================================================
KEY OBSERVATIONS:
============================================================
1. Best Test Accuracy: Decision Tree (88.929%)
2. Smallest Train-Test Gap: SVM (0.000000)
3. Most Consistent Model: Decision Tree (identical train/test accuracy)
```

Confusion Matrix



Logistic Regression, SVM, KNN, DecisionTree results



Decision Tree results (Run1) showing an example of instabilty in Confusion Matrix

Decision Tree Confusion Matrix (Run 1): TN=0, FP=6, TP=12, FN=0 Precision: 67% Would lose 6 rockets ($180M) • Business impact: Disastrous

Decision Tree Confusion Matrix(Run 2): TN=3, FP=3, TP=12, FN=0 Precision: 80% Would lose 3 rockets ($90M) • Business impact: Acceptable (same as other models)

Variability of Decision Tree unacceptable. A model that gives variable business outcomes each time is dangerous for mission-critical applications.

Recommend Model: Logistic Regression: TN=3, FP=3, TP=12, FN=0 Precision: 80%

All other models, but not decision tree, have similar results. But Logistic regression is, stable, interpretable, gives predictable business outcomes, simpler, faster, provides confidence scores

Conclusions

Critical Discovery: Model Stability Variability • Decision Tree produces different confusion matrices between runs • Business impact varies from losing $90M to $180M in rockets • This unpredictability is unacceptable for mission-critical applications

Stable models: Logistic Regression, SVM, KNN • All produce consistent, predictable results • All have identical business outcomes (80% precision, 100% recall) • All generate $654M net value reliably

Accuracy Tradeoff: • Decision Tree: 88.9% (highest but unstable) • SVM/KNN: 84.8% (stable) • Logistic Regression: 83.3% (stable) • The 1.5-5.6% accuracy difference is insignificant compared to stability

Final Reccommendation: Logistic Regression • Stable, predictable business outcomes • Interpretable for engineering and regulatory review • Fast predictions for real-time launch decisions • Provides probability scores for risk assessment • Best balance of stability, interpretability, and performance

LESSON LEARNED: Always test model stability by running multiple times For mission-critical applications, stability is more important than marginal accuracy gains Interpretability enables better decision-making and regulatory approval

Appendix 1

📊 Statistical Performance Comparison

Model	Accuracy	Precision	Recall (Sensitivity)	F1-Score	Specificity (TNR)	FP Rate (FPR)	FN Rate (FNR)	TP	TN	FP	FN	Total Predictions
Logistic Regression	83.33%	80.00%	100.00%	88.89%	50.00%	50.00%	0.00%	12	3	3	0	18
SVM	84.82%	80.00%	100.00%	88.89%	50.00%	50.00%	0.00%	12	3	3	0	18
Decision Tree	88.93%	80.00%	100.00%	88.89%	50.00%	50.00%	0.00%	12	3	3	0	18
KNN	84.82%	80.00%	100.00%	88.89%	50.00%	50.00%	0.00%	12	3	3	0	18

💰 TABLE 2: KEY BUSINESS IMPACT

Model	Rockets Lost	Missed Launches	Successful Launches	Rocket Loss	Revenue Loss	Revenue	Net Value	Risk Level
Logistic Regression	3	0	12	$90M	$0M	$744M	$654M	✅ LOW
SVM	3	0	12	$90M	$0M	$744M	$654M	✅ LOW
Decision Tree	3	0	12	$90M	$0M	$744M	$654M	⚠️ HIGH
KNN	3	0	12	$90M	$0M	$744M	$654M	✅ LOW

⚠️ IMPORTANT NOTE: Decision Tree shows instability between runs
Observed variability: Could lose 3 rockets ($90M) OR 6 rockets ($180M)
This unpredictability makes it unsuitable for production use

Appendix 2

```
================================================================================
Final Recommendation: Logistic Regression
================================================================================

🥇 Logistic Regression:
   • Stable, predictable outcomes
   • Highly interpretable (critical for engineering review)
   • Fast predictions for real-time decisions
   • Provides probability confidence scores
   • Slightly lower accuracy (83.33%) but identical business impact

🥈 SVM (Alternative):
   • Same business outcomes as LR
   • Slightly higher accuracy (84.82%)
   • Less interpretable ("black box")
   • Good choice if pure accuracy is paramount

🚫 Decision Tree (Rejected):
   • Highest accuracy (88.93%) but UNSTABLE
   • Business outcomes vary unpredictably
   • Could lose $180M instead of $90M
   • Unacceptable risk for mission-critical application

🥉 KNN (Consider):
   • Same business outcomes as LR/SVM
   • Same accuracy as SVM (84.82%)
   • Slow predictions (stores all training data)
   • Good for small datasets, poor scalability
```

Thank you!