IBM Developer
SKILLS NETWORK

# Winning Space Race with Data Science

Miguel Huidobro García
May, 2024

## Outline

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

## Executive Summary

- **Spatial missions** involved vast amounts of money in the different parts of the mission, but more specifically in the production of the rockets. The **impressive cost reduction** (around 60%) of SpaceX has motivated us to build a model to **predict** the final cost based on their methodology.

- We have observed that the main reason for the cost reduction is the **recovery of the first stage of the rocket**. Hence, we perform a **detailed analysis** of the data deployed by the SpaceX to achieve our goal. First, we **collect** the data from the SpaceX REST API and **clean** it for a consequent **graphic visualization** (static and interactive). Then, we construct and evaluate multiple **machine learning** algorithms to find the optimal predictive model.

- The results obtained from our analysis remark that a **Decision Tree** model is the best choice to predict the costs based on the **score** and on the **confusion matrix**.

## Introduction

- The typical cost of rocket production stands around 165 million dollars, however, the Falcon 9 SpaceX rocket has reached the amount of 62 million dollars, which supposes a **62% reduction of the cost**. The main reason for this success lies in the reuse of the first stage of the rocket, being the most expensive part. However, this is not always possible since an **incorrect landing** might occurs producing the destruction of the first stage. Additionally, the mission parameters, like the **orbit** or **payload**, might preclude the **recovery** of this part of the rocket.

- From a different perspective, **data analysis and machine learning** techniques are powerful tools widely used for the understanding and prediction of the possible outcomes in a specific situation.

- For this reason, it is our aim to find the possible **relations** between the mission parameters and **predict the final cost** of the rocket launching using the mentioned framework using the data from previous attempts deployed by the SpaceX agency.

Section 1

## Methodology

# Methodology

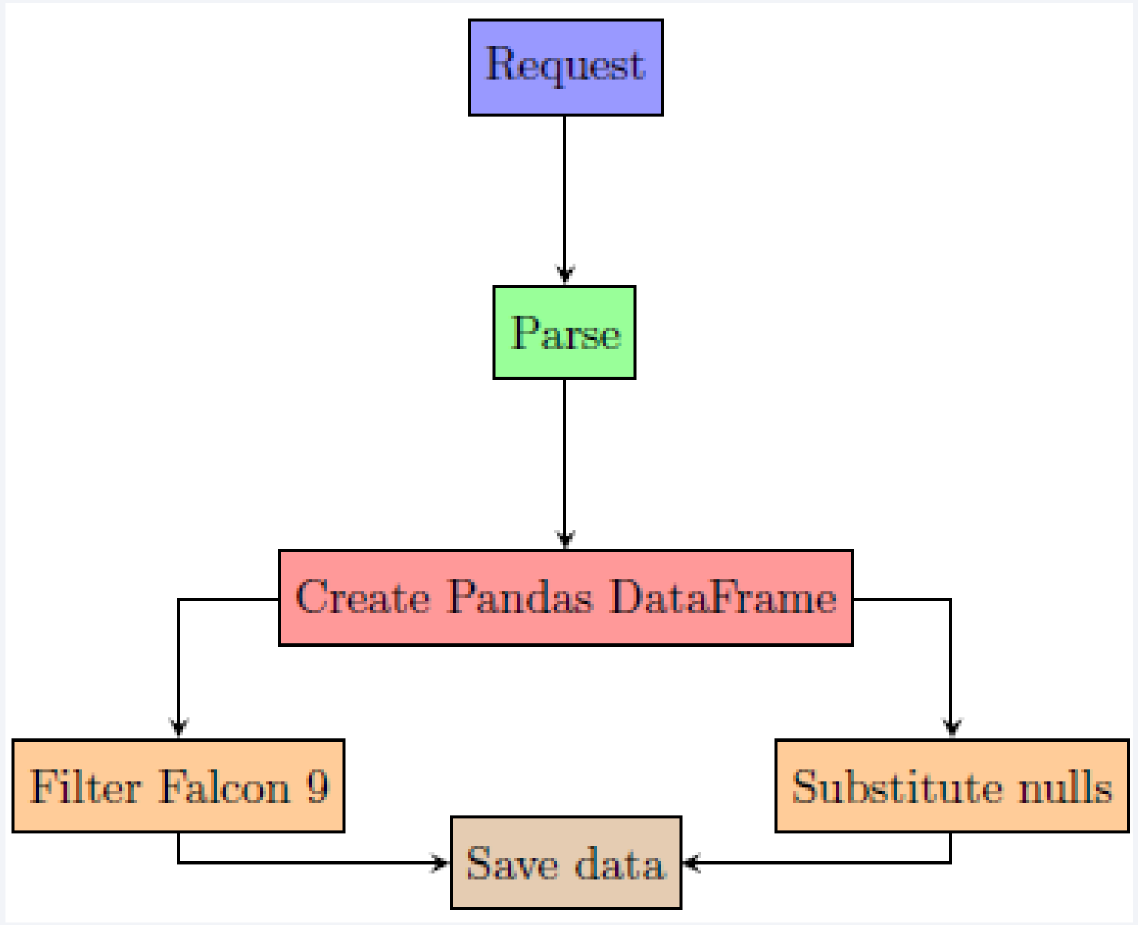
## Executive Summary

- Data collection methodology:
  - SpaceX REST API and WebScraping + BeautifulSoup
- Perform data wrangling
  - Python library: Pandas
- Perform exploratory data analysis (EDA) using visualization and SQL
- Perform interactive visual analytics using Folium
- Perform predictive analysis using classification models
  - Python library: Scikit-learn

## Data Collection – SpaceX API

1. Request the data from SpaceX url.
2. We need to decode the content from the json file and save the data on a pandas DataFrame.
3. A new DataFrame is created only with the data of interest.
4. We clean the data by filtering the values for Falcon 9 rocket and erasing the null values from the LaunchSite column.
5. We finally save the final data in a csv file.

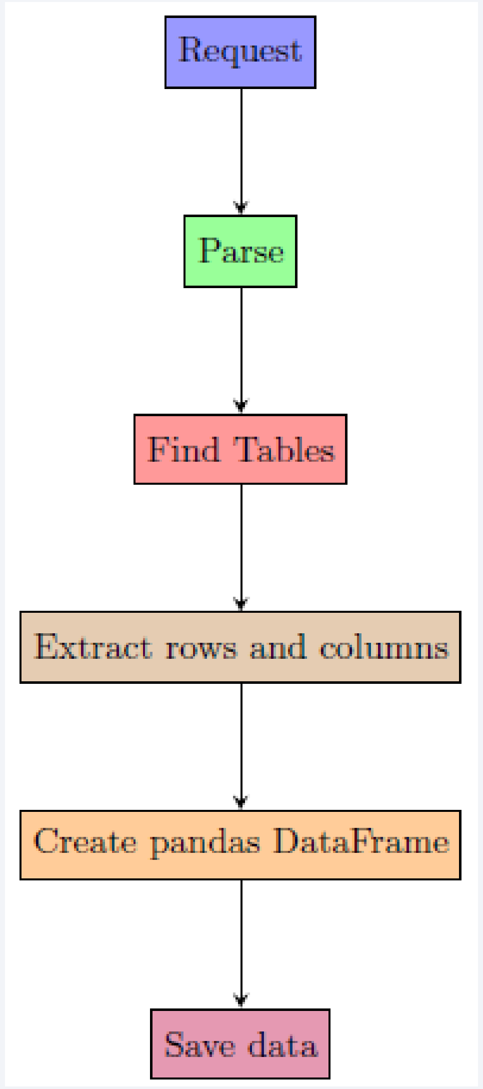
- We leave here the GitHub link to the notebook for the peer-review.



## Data Collection - Scraping

1. Request the Falcon 9 data from the url as before.
2. In this case, we use the BeautifulSoup module to parse the content of the request.
3. The output of the previous step allows us to find the tables easily.
4. We extract the content of the table with the parameters of the Falcon 9 launch.
5. We insert the data into a dictionary which is later converted into a pandas DataFrame.
6. We save the data into a csv file.

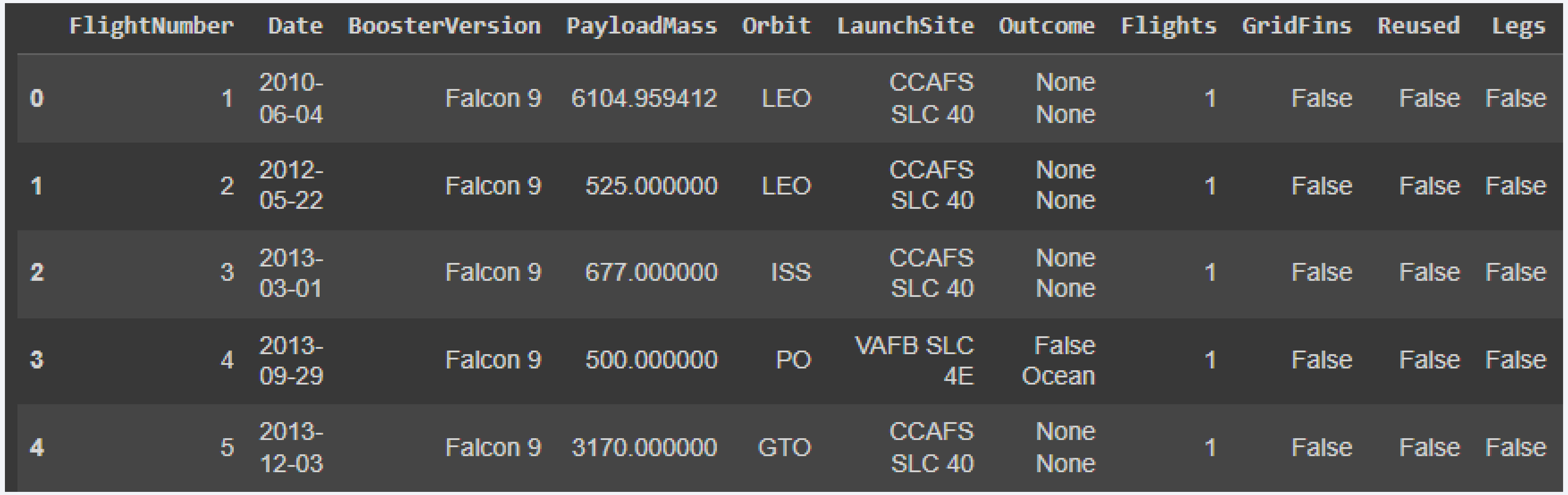
- We leave here the [GitHub link to the notebook]() for the peer-review.



## Data Wrangling

- In this step we read the csv file saved in the previous step and prepare the data for a posterior visualization and the training of a machine learning model.

1. Find the nulls values in the data and identify them as bad landings.
2. Convert the object-type landing values into integers: 0 (if bad) or 1 (if good).
3. We save the new version of the data (see image below) into a csv file.

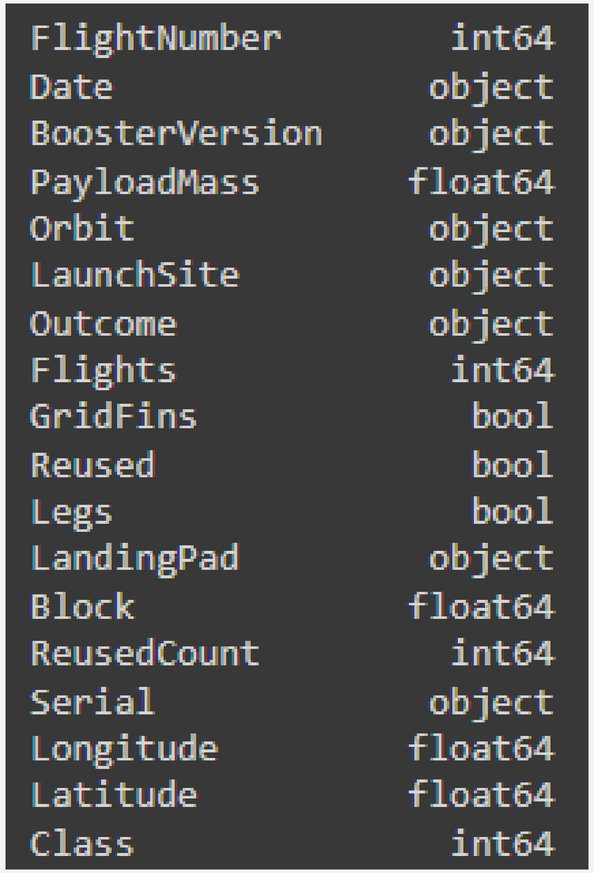


| | FlightNumber | Date | BoosterVersion | PayloadMass | Orbit | LaunchSite | Outcome | Flights | GridFins | Reused | Legs |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 1 | 2010-06-04 | Falcon 9 | 6104.959412 | LEO | CCAFS SLC 40 | None None | 1 | False | False | False |
| **1** | 2 | 2012-05-22 | Falcon 9 | 525.000000 | LEO | CCAFS SLC 40 | None None | 1 | False | False | False |
| **2** | 3 | 2013-03-01 | Falcon 9 | 677.000000 | ISS | CCAFS SLC 40 | None None | 1 | False | False | False |
| **3** | 4 | 2013-09-29 | Falcon 9 | 500.000000 | PO | VAFB SLC 4E | False Ocean | 1 | False | False | False |
| **4** | 5 | 2013-12-03 | Falcon 9 | 3170.000000 | GTO | CCAFS SLC 40 | None None | 1 | False | False | False |

- We leave here the GitHub link to the notebook for the peer-review.

## EDA with Data Visualization

- In this step we represent some of the features of the dataset (shown below):



```
FlightNumber        int64
Date               object
BoosterVersion     object
PayloadMass       float64
Orbit              object
LaunchSite         object
Outcome            object
Flights             int64
GridFins             bool
Reused               bool
Legs                 bool
LandingPad         object
Block             float64
ReusedCount         int64
Serial             object
Longitude         float64
Latitude          float64
Class               int64
```

- We make use of the python library **seaborn** to create the plots.

- The goal is to find possible relations between the features. More specifically, we are interested in relations between: **LaunchSite**, **FlightNumber**, **PayloadMass**, **Orbit**, **Year** and **Class** (good or bad landing).

- From the different possible representations we use multiple scatterplots, a barplot and a lineplot.

- Finally, we convert the remaining object-type features into integer values via **one-hot enconding** procedure.

- We leave here the GitHub link to the notebook for the peer-review.

## EDA with SQL

- At this point we will familiarize with the data by querying information using SQL.
- This task is achieved by loading the SQL extension of python and then introducing our DataFrame into a table allocated in a data base via a **sqlite3** connection.
- Then using the magic command **%sql** or **%%sql** we are allowed to write SQL code to extract information.
- We apply multiple combined queries which are easily executed using SQL language and its built-in functions for a clear understanding of the data. This will be shown in future slides.
- We leave here the GitHub link to the notebook for the peer-review.

## Build an Interactive Map with Folium

- For an illustrative visualization of the data we use **Folium**, with which we can display and highlight different places in a map and label them using the features.
- This is also possible since we have the Latitude and Longitude parameters in our data.
- We use the **Circle** and **Marker** objects of folium to denote the Launch Sites and the nearby regions in the world map and compute the distances between them.
- Additionally, we add the different success/failure cases in each Launch Site in a cluster of makers.
- The results are shown in future slides, and the codes to reproduce the results are given in Appendix II.
- We leave here the GitHub link to the notebook for the peer-review.

## Predictive Analysis (Classification)

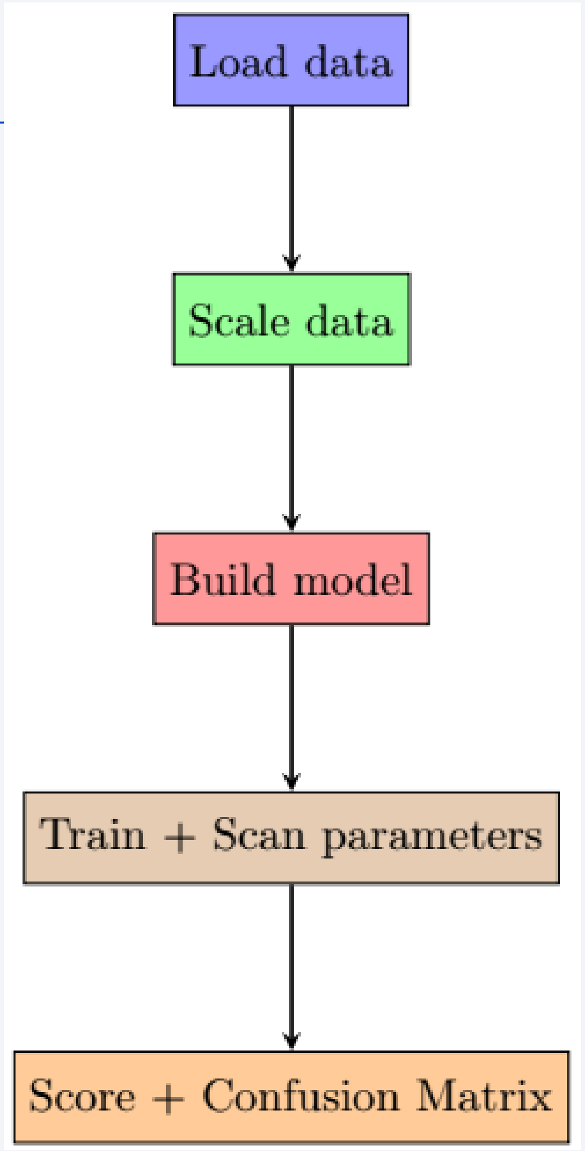
- For a prediction based on our previous analysis we build an appropriate machine learning model, which is a **Classification algorithm,** using the **Scikit-Learn** python library.

- We first read and scale the data using the StandardScaler() function, and split them between **training** and **testing** datasets.

- The models considered are: **Logistic Regression**, **Supported Vector Machine**, **Decision Tree** and **K-Nearest Neighbours** models.

- We perform a **scan** on multiple parameters for each model during the training and evaluate them computing the **score** and **confusion matrix** using the test dataset.

- We leave here the GitHub link to the notebook for the peer-review.

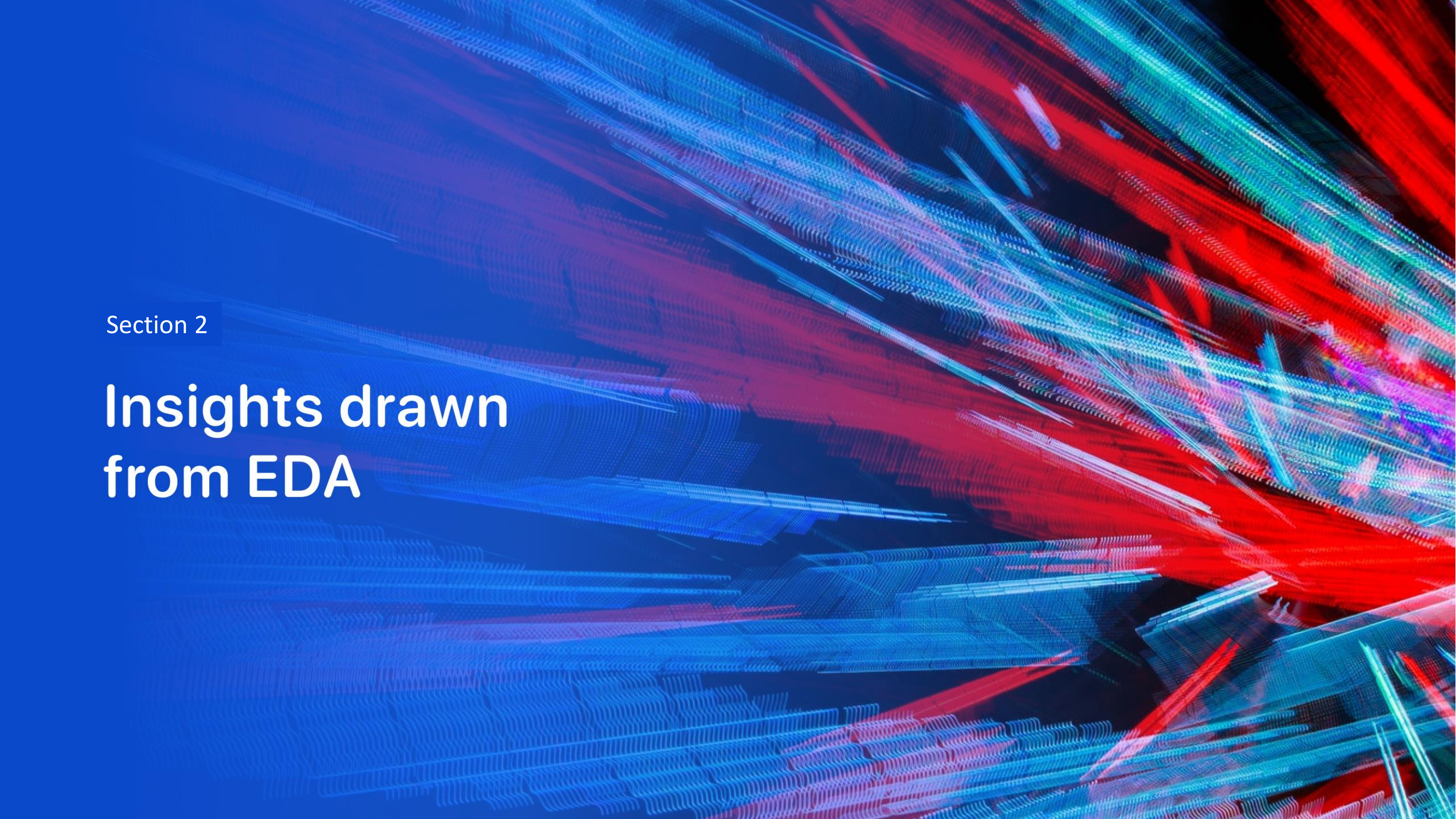


## Summary of results

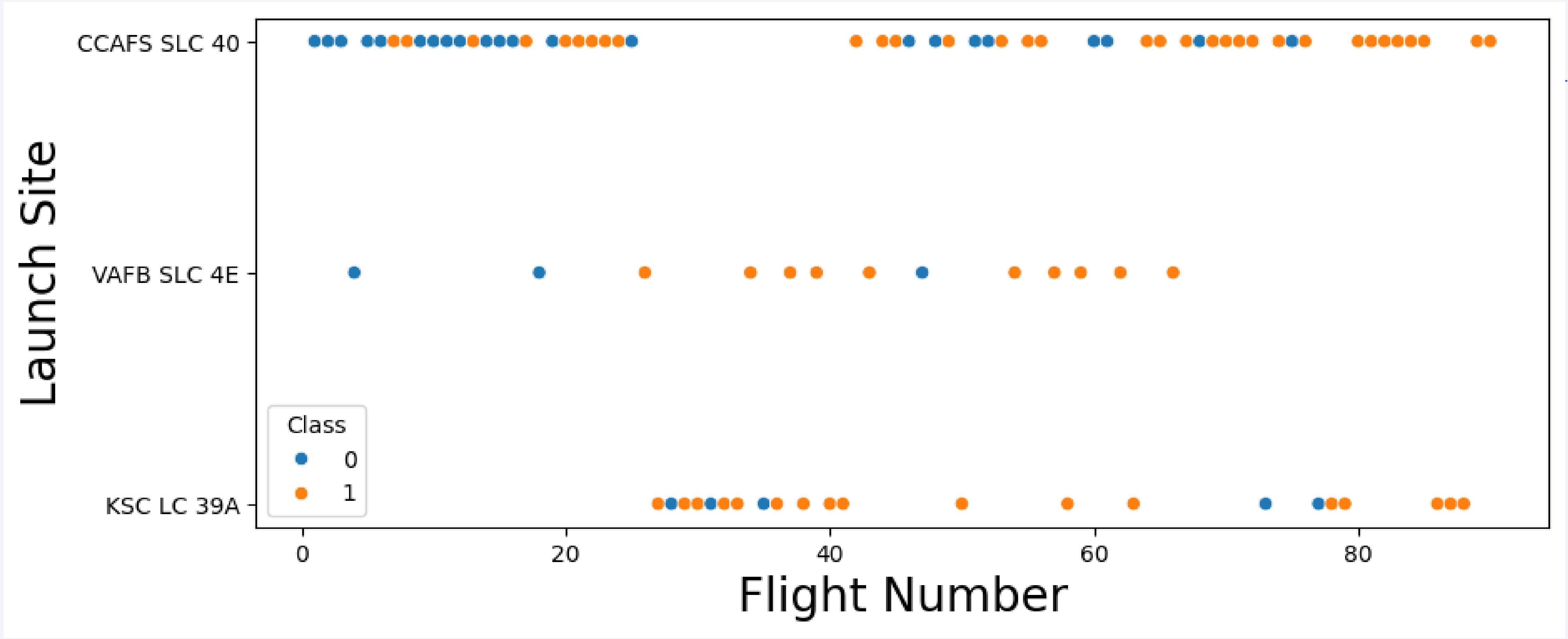
- The results of the different parts in the Data Analysis paradigm, illustrated in the previous slides, is shown in the following.
- As the result from our data collection and wrangling first task is a final csv file with the desired values in the correct format which can be consulted anytime.
- From the EDA part we have observed the possible relations between the data, from which we have extracted the most important features: Orbit, Flight Number, Year, Payload Mass and Landing Outcome.
- Finally, we have compared among multiple machine learning models and obtained the optimal to develop our task. This model may be saved and reused for similar applications.

Section 2

## Insights drawn from EDA

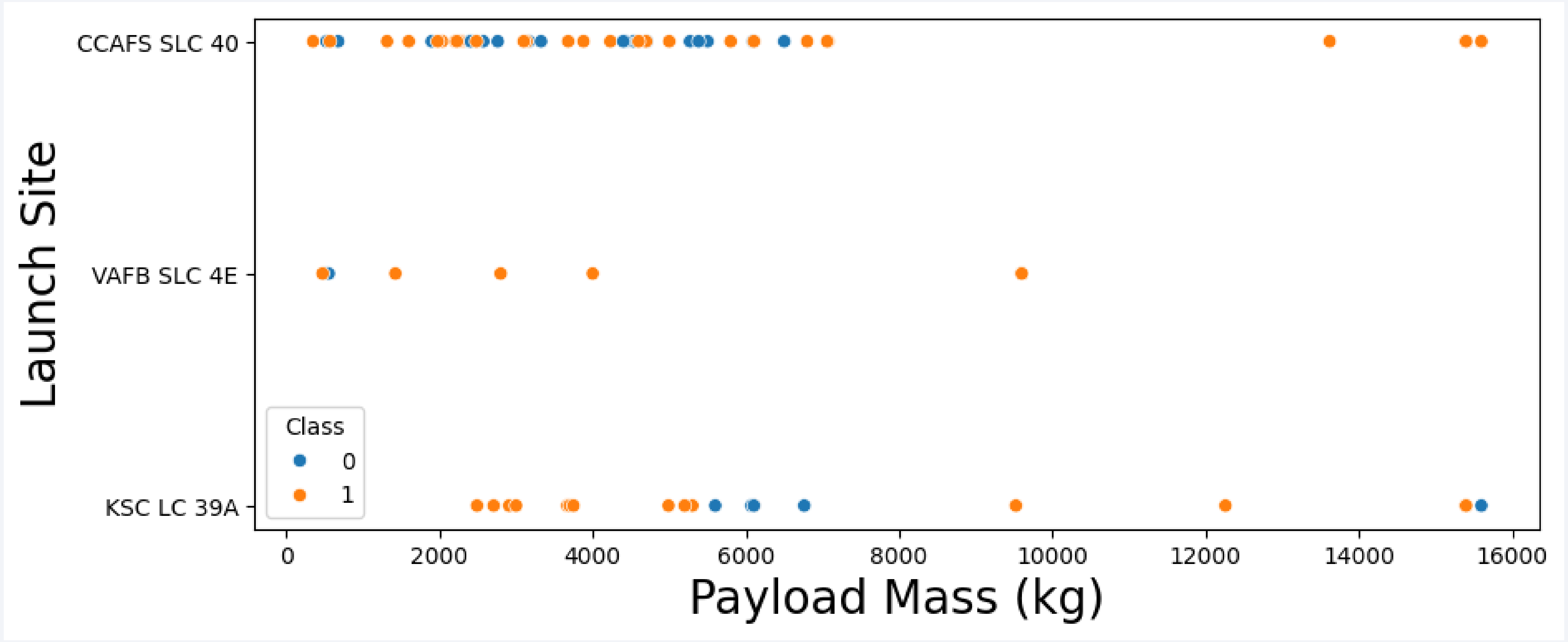


# Flight Number vs. Launch Site



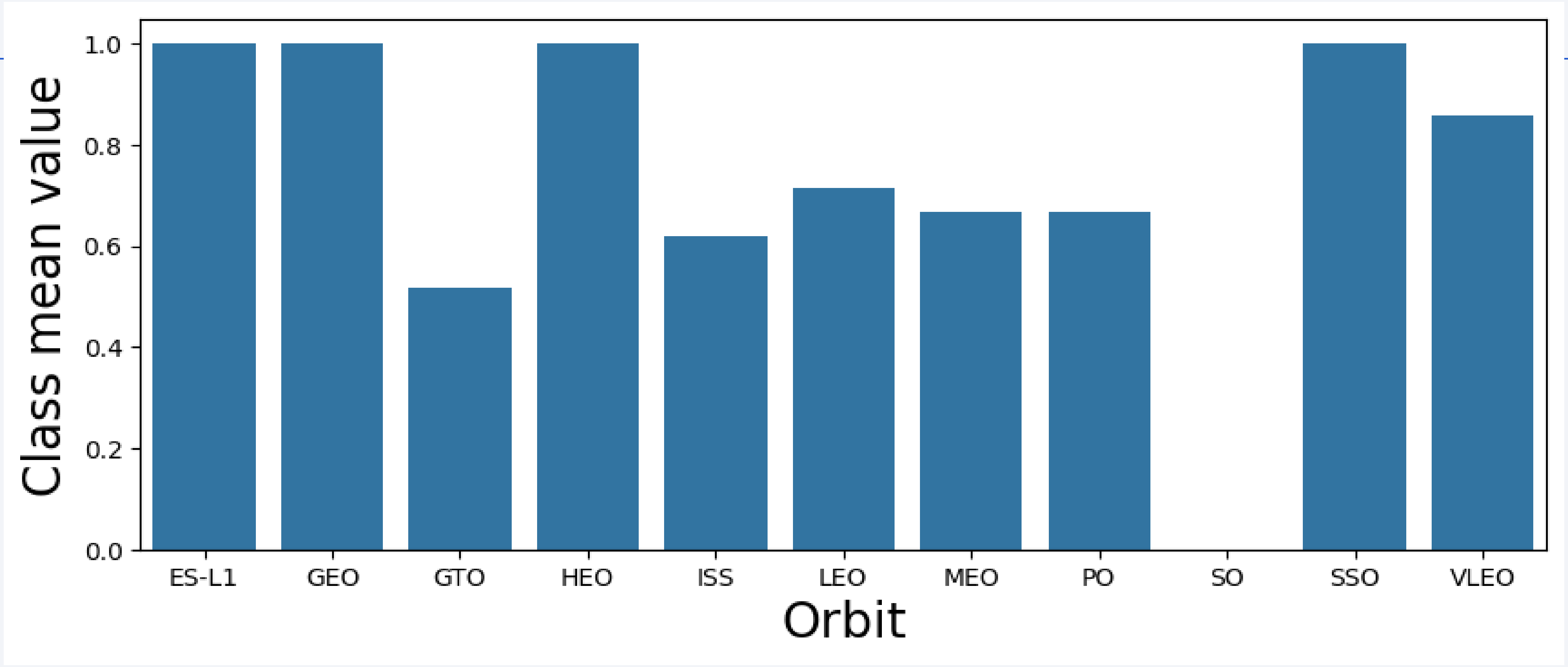
- We show the Launch Site against the Flight Number. The blue dots denote a wrong landing, whilst the orange color represents the correct landed cases.
- We observe that higher flight numbers yield a larger amount of correct landings. We observe the largest amount of launches from “CCAFS SLC 40” site and no launches have been performed from “KSC LC39 A” with flight numbers smaller than 27.

# Payload vs. Launch Site



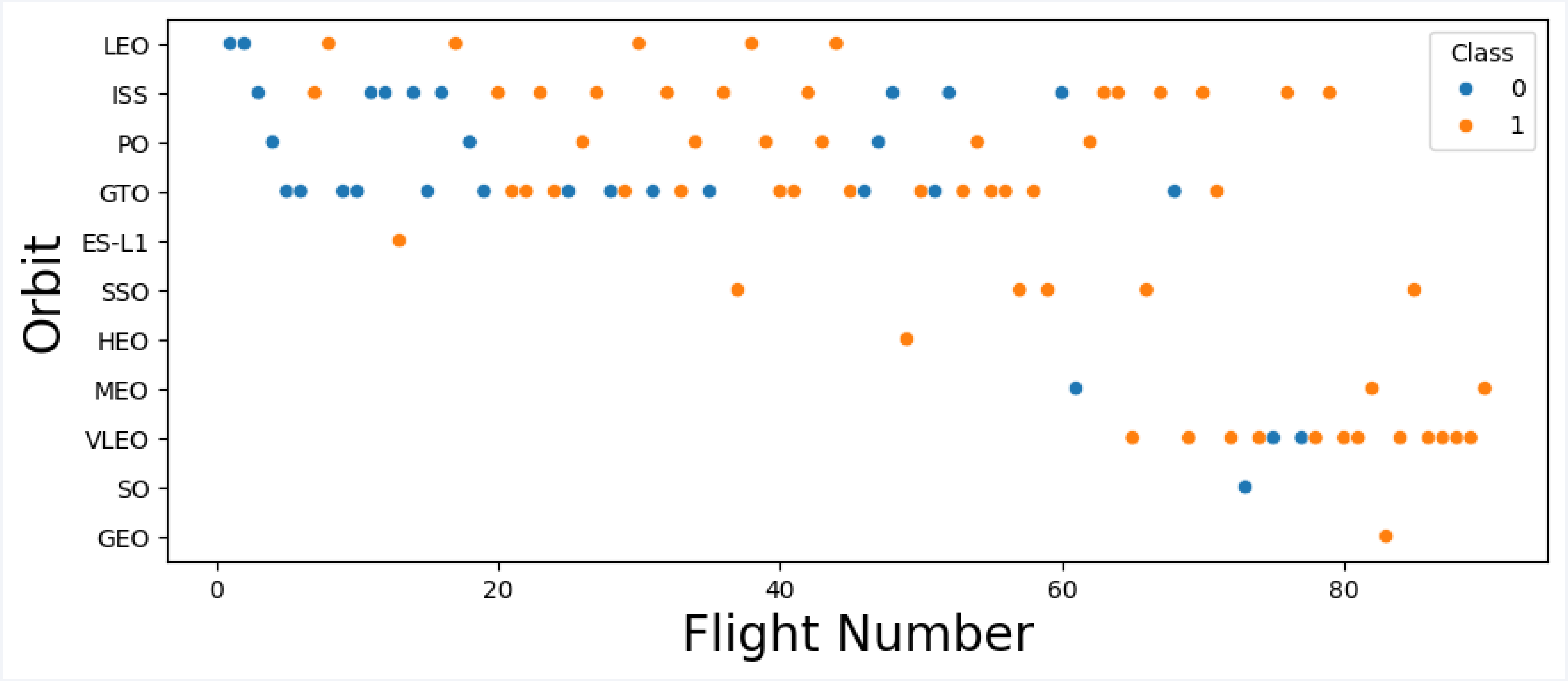
- This plot follows the same convention explained in the previous slide.
- We do not observe a clear relationship between these two variables, but it is found that the maximal payload Mass for the launches from "VAFB SLC 4E" is around 10000 kg.

## Success Rate vs. Orbit Type



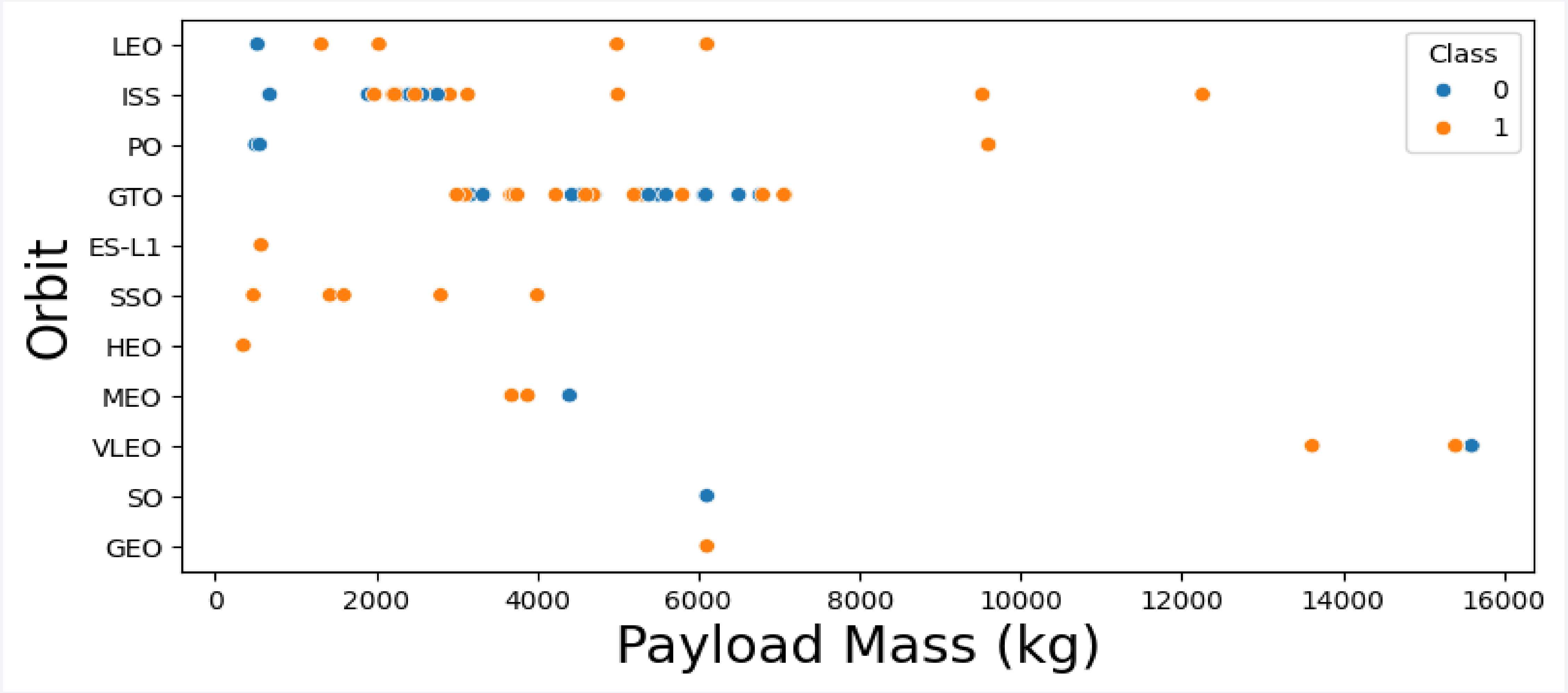
- We represent the mean value of the successes and failures for each type of orbit in a barplot.
- We find that the “ES-L1”, “GEO”, “HEO” and “SO” have a perfect success rate of landing, whilst the “SO” is the only one with the 100% of failures.

## Flight Number vs. Orbit Type



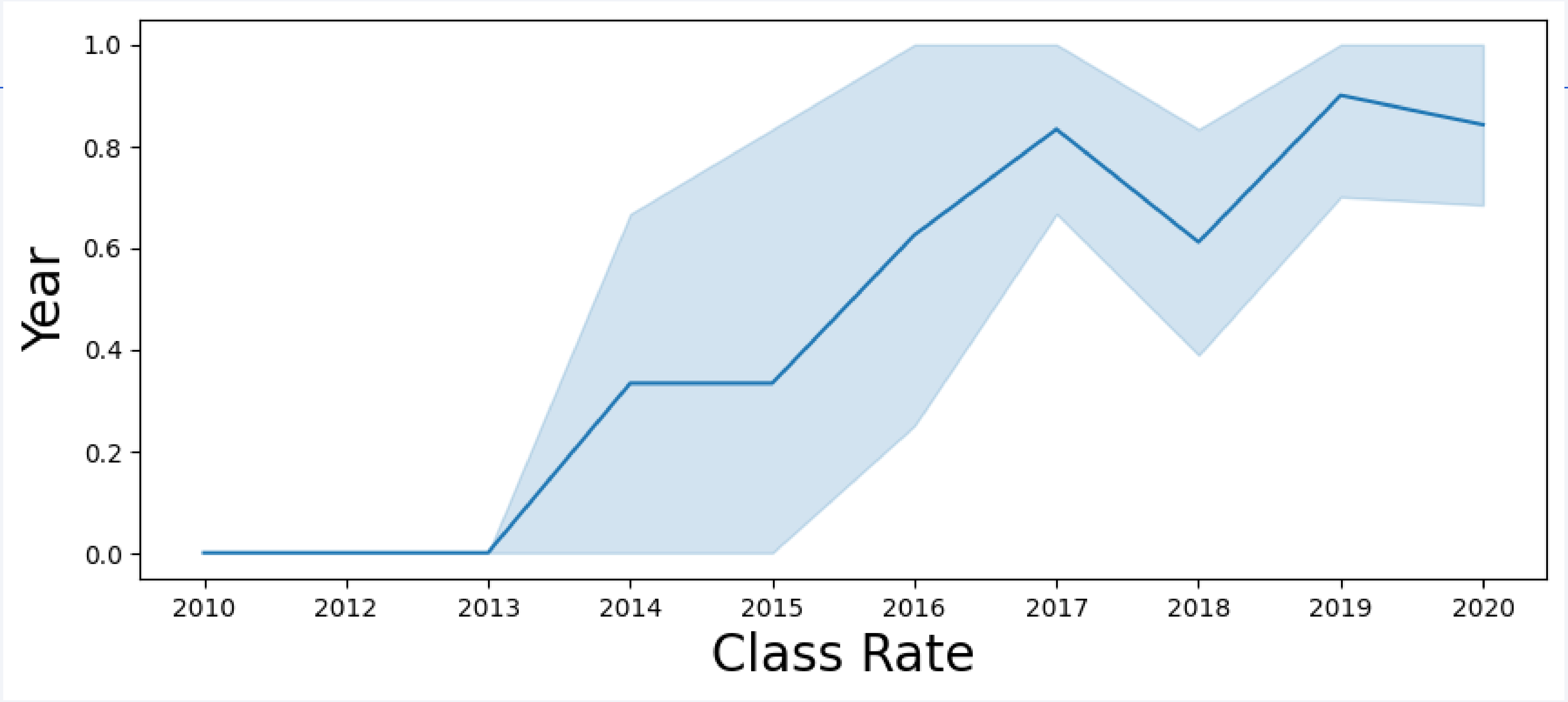
- Here we observe relations between the Orbit and the Flight Number in a scatterplot.
- We see a larger variety of orbits for higher flight numbers, and an increasing success rate.

## Payload vs. Orbit Type



- From this scatterplot we conclude that the high Payload Mass cases (which where launched from “CCAFS” and “KSC”) have the same kind of orbit “VLEO”.

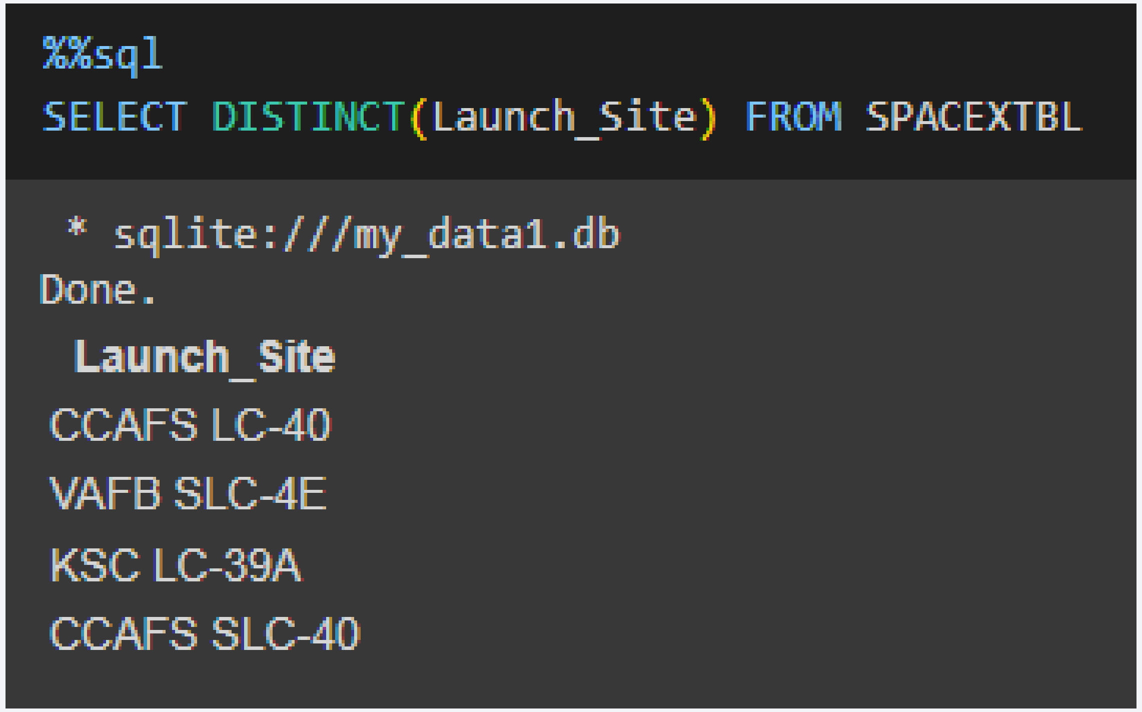
## Launch Success Yearly Trend



- In this lineplot we study the evolution in time of the successes in the landing. The correct outcomes start in the year 2013 and a  sharp increase is observed between 2015 and 2017.

## All Launch Site Names

- This information is easily extracted using SQL on our DataFrame hosted in a database.
- We present below the lines of code and the output of our query. The structure of a query in SQL requires a **SELECT** statement **FROM** the specific table. In this case we make use of the **DISTINCT** function to find the different **Launch_Site** values.

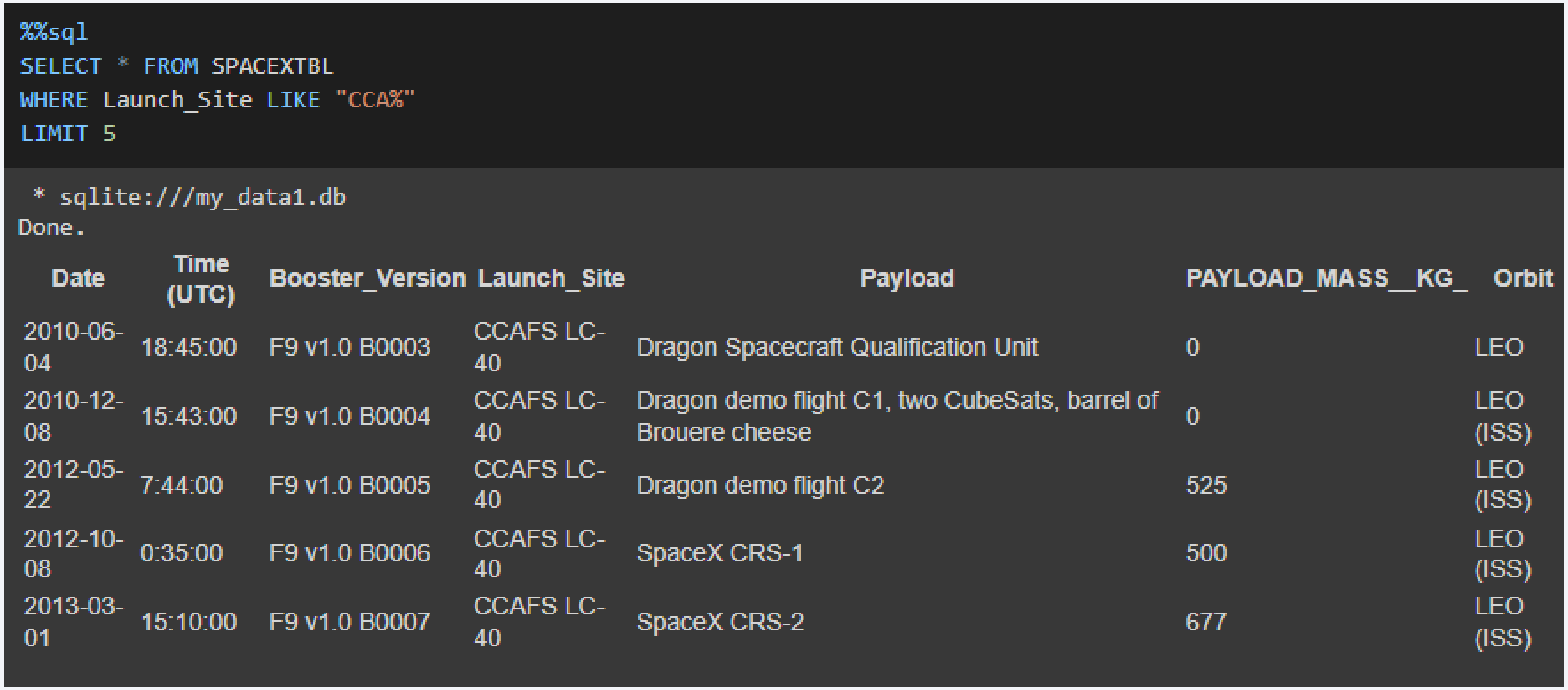


```
%%sql
SELECT DISTINCT(Launch_Site) FROM SPACEXTBL
```

```
 * 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'

- In order to extract all the features we use *. Then we use **WHERE** to filter the values of a specific feature starting with "CCA" using the command **LIKE**. We only show the 5 first records using **LIMIT**.



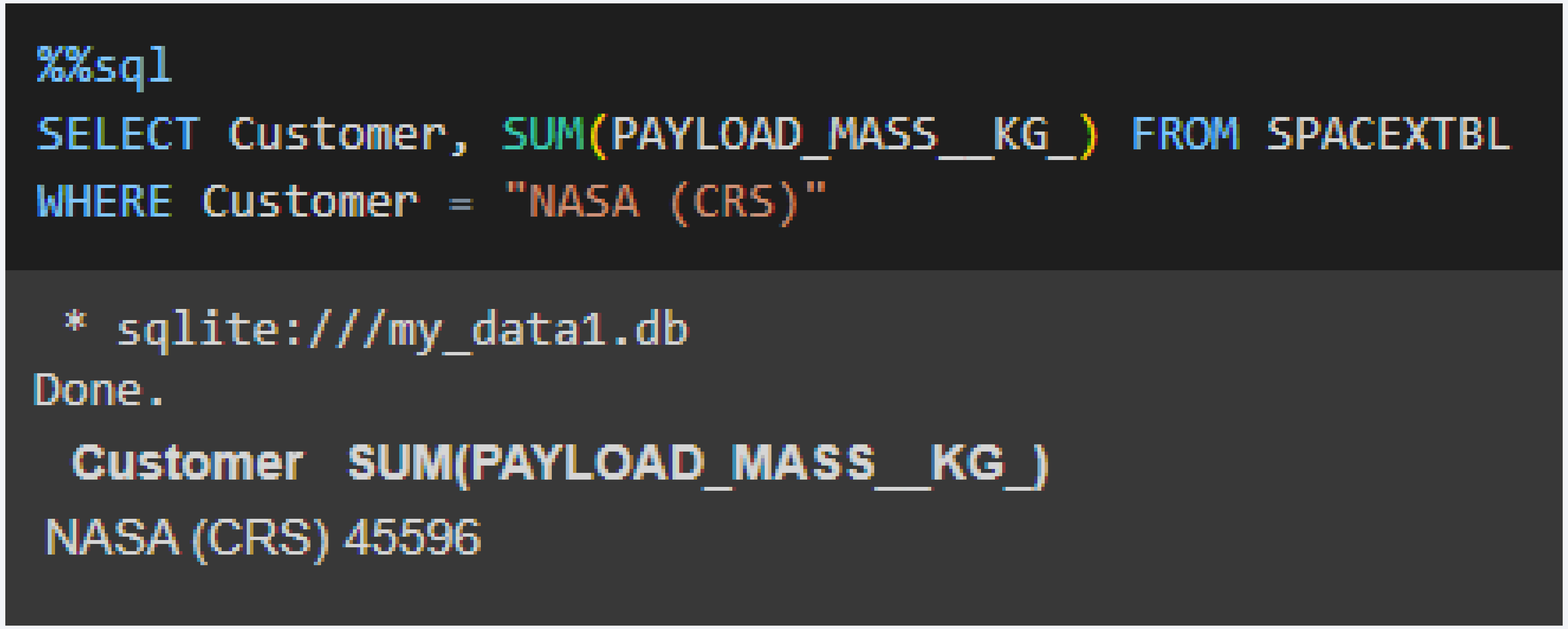
```
%%sql
SELECT * FROM SPACEXTBL
WHERE Launch_Site LIKE "CCA%"
LIMIT 5
```

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

| Date | Time (UTC) | Booster_Version | Launch_Site | Payload | PAYLOAD_MASS__KG_ | Orbit |
|---|---|---|---|---|---|---|
| 2010-06-04 | 18:45:00 | F9 v1.0 B0003 | CCAFS LC-40 | Dragon Spacecraft Qualification Unit | 0 | LEO |
| 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) |
| 2012-05-22 | 7:44:00 | F9 v1.0 B0005 | CCAFS LC-40 | Dragon demo flight C2 | 525 | LEO (ISS) |
| 2012-10-08 | 0:35:00 | F9 v1.0 B0006 | CCAFS LC-40 | SpaceX CRS-1 | 500 | LEO (ISS) |
| 2013-03-01 | 15:10:00 | F9 v1.0 B0007 | CCAFS LC-40 | SpaceX CRS-2 | 677 | LEO (ISS) |

## Total Payload Mass

- For the total value of a numeric-type feature we may use the **SUM()** aggregation function. We also filter the values of Customer feature.

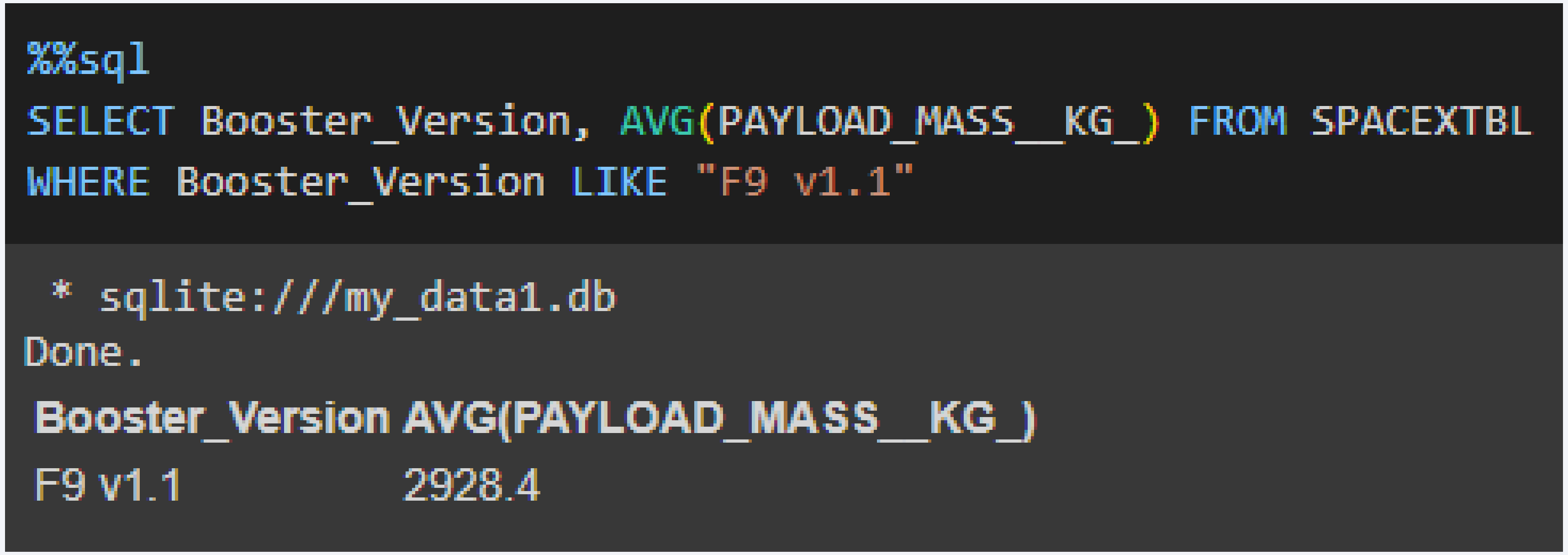


```
%%sql
SELECT Customer, SUM(PAYLOAD_MASS__KG_) FROM SPACEXTBL
WHERE Customer = "NASA (CRS)"
```

```
 * sqlite:///my_data1.db
Done.
 Customer   SUM(PAYLOAD_MASS__KG_)
NASA (CRS) 45596
```

## Average Payload Mass by F9 v1.1

- Another important aggregation function is **AVG()**. This is used to obtain the average value of a numeric-type feature.
- In this figure we ask for the average payload mass of the “F9 v1.1” Booster Version.

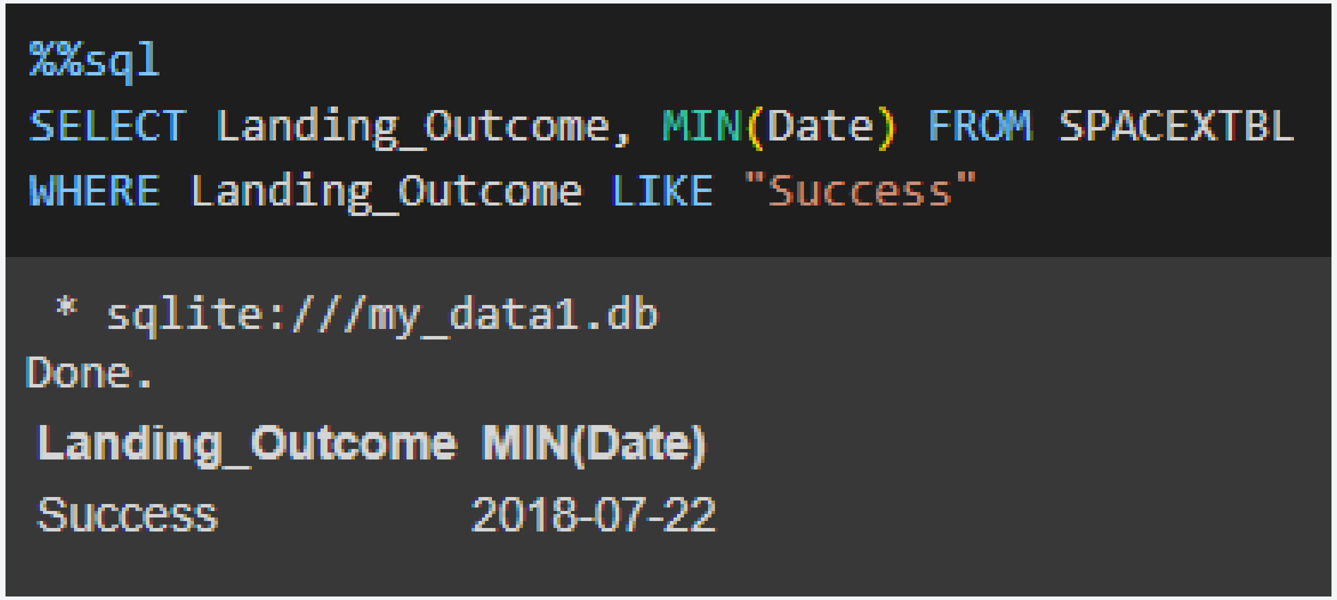


```
%%sql
SELECT Booster_Version, AVG(PAYLOAD_MASS__KG_) FROM SPACEXTBL
WHERE Booster_Version LIKE "F9 v1.1"
```

```
 * sqlite:///my_data1.db
Done.
Booster_Version AVG(PAYLOAD_MASS__KG_)
F9 v1.1             2928.4
```

## First Successful Ground Landing Date

- We display the first successful landing date, which is the determined by the minimal Date with “Success” value in Landing Outcome.

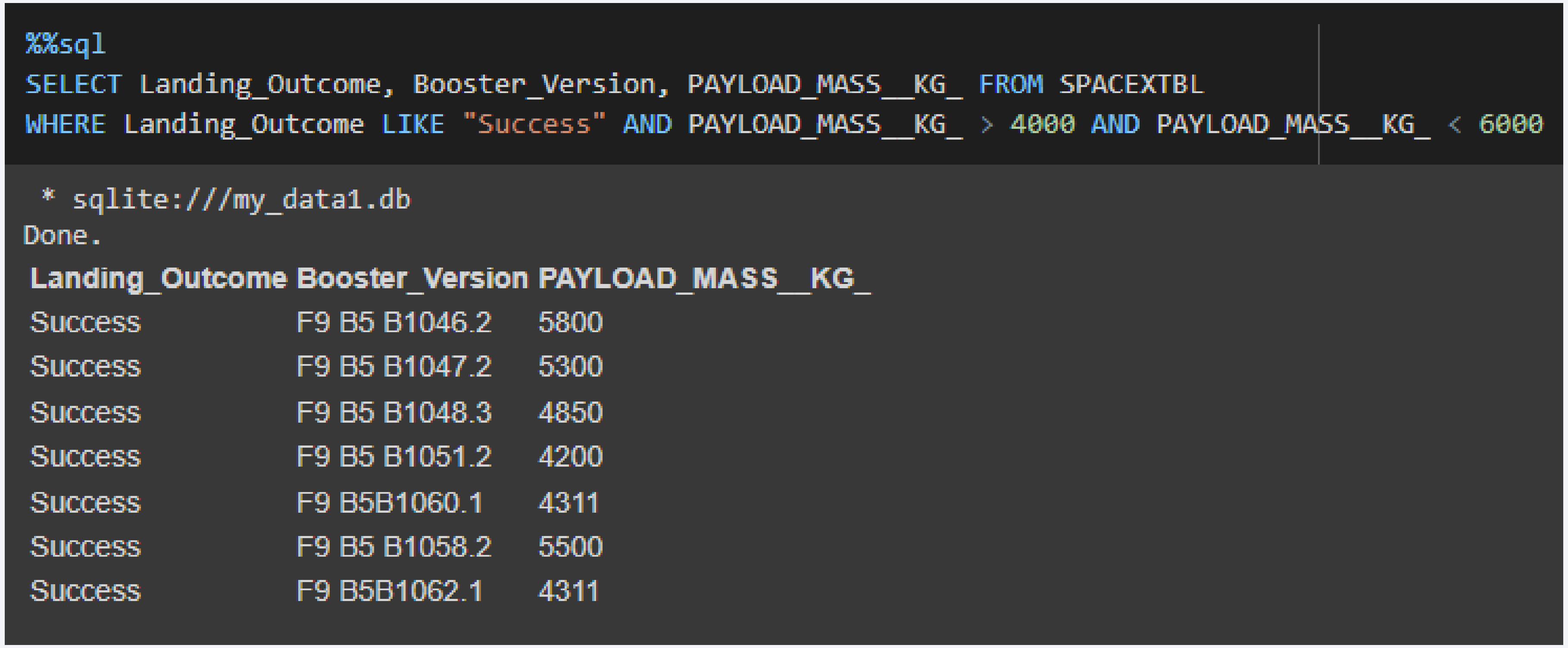


```
%%sql
SELECT Landing_Outcome, MIN(Date) FROM SPACEXTBL
WHERE Landing_Outcome LIKE "Success"
```

```
 * sqlite:///my_data1.db
Done.
Landing_Outcome MIN(Date)
Success         2018-07-22
```

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

- Multiple requirements on the query may be concatenated using **AND** as it is shown below.
- All the success cases with the same range of Payload Mass have similar Booster Versions.

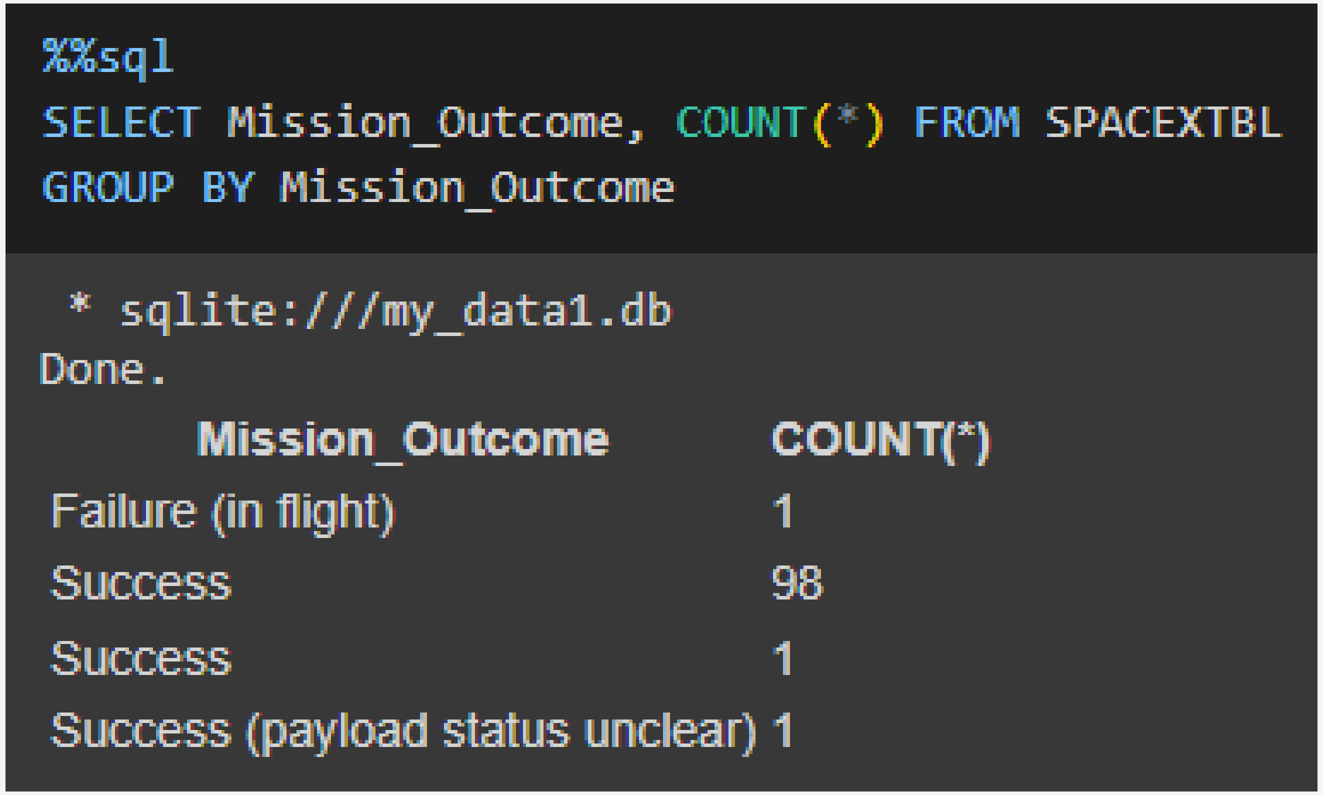


```
%%sql
SELECT Landing_Outcome, Booster_Version, PAYLOAD_MASS__KG_ FROM SPACEXTBL
WHERE Landing_Outcome LIKE "Success" AND PAYLOAD_MASS__KG_ > 4000 AND PAYLOAD_MASS__KG_ < 6000
```

```
 * sqlite:///my_data1.db
Done.
Landing_Outcome Booster_Version PAYLOAD_MASS__KG_
Success         F9 B5 B1046.2   5800
Success         F9 B5 B1047.2   5300
Success         F9 B5 B1048.3   4850
Success         F9 B5 B1051.2   4200
Success         F9 B5B1060.1    4311
Success         F9 B5 B1058.2   5500
Success         F9 B5B1062.1    4311
```

## Total Number of Successful and Failure Mission Outcomes

- To obtain the total number of occurrences we use the **COUNT()** aggregation function. Additionally, we use the **GROUP BY()** command to collect the data by the success/failure outcome.

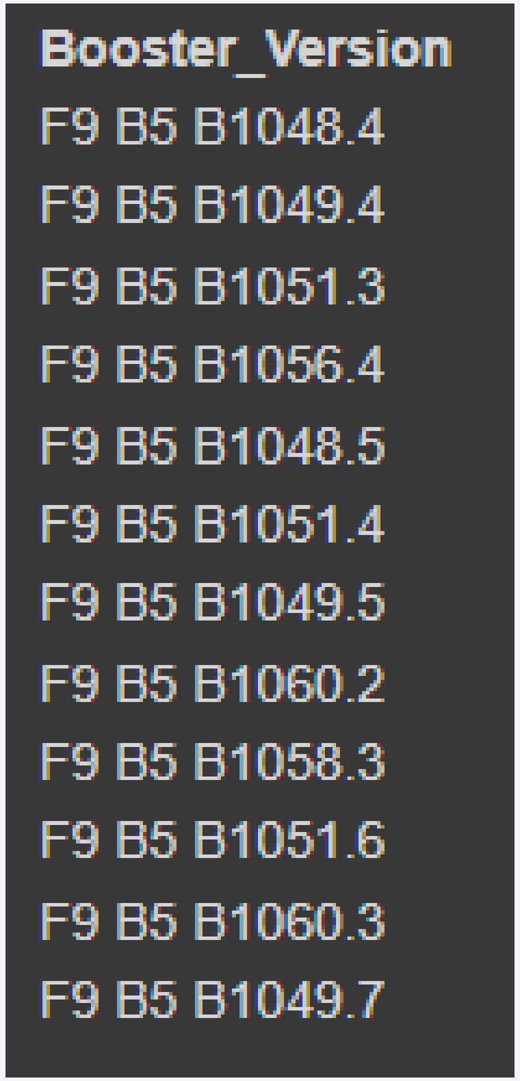


```
%%sql
SELECT Mission_Outcome, COUNT(*) FROM SPACEXTBL
GROUP BY Mission_Outcome
```

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

| Mission_Outcome | COUNT(*) |
|---|---|
| Failure (in flight) | 1 |
| Success | 98 |
| Success | 1 |
| Success (payload status unclear) | 1 |

## Boosters Carried Maximum Payload



| 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 |

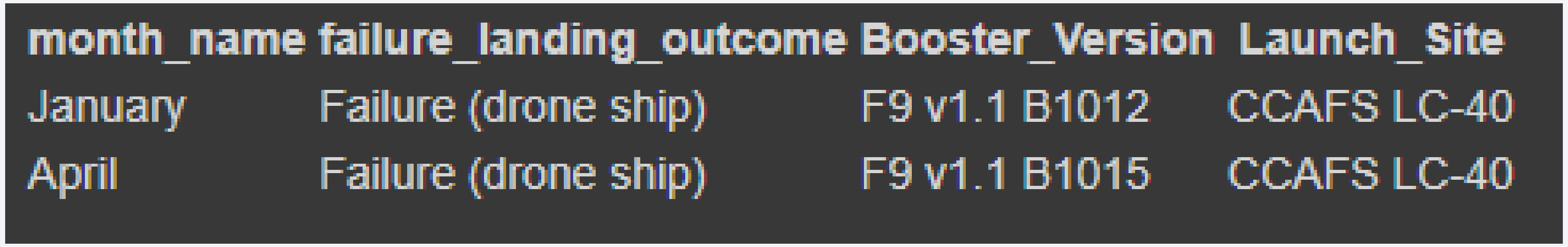
- For this task we need to implement a subquery on the values of the Payload Mass.
- The SQL code is visible in this slide.

## 2015 Launch Records

- In this query we extract multiple columns for the launch records in 2015. For the **month_name** values we implement a **CASE** statement with the months of the year.

- We also require the outcome of the landing to be failure in drone ship.

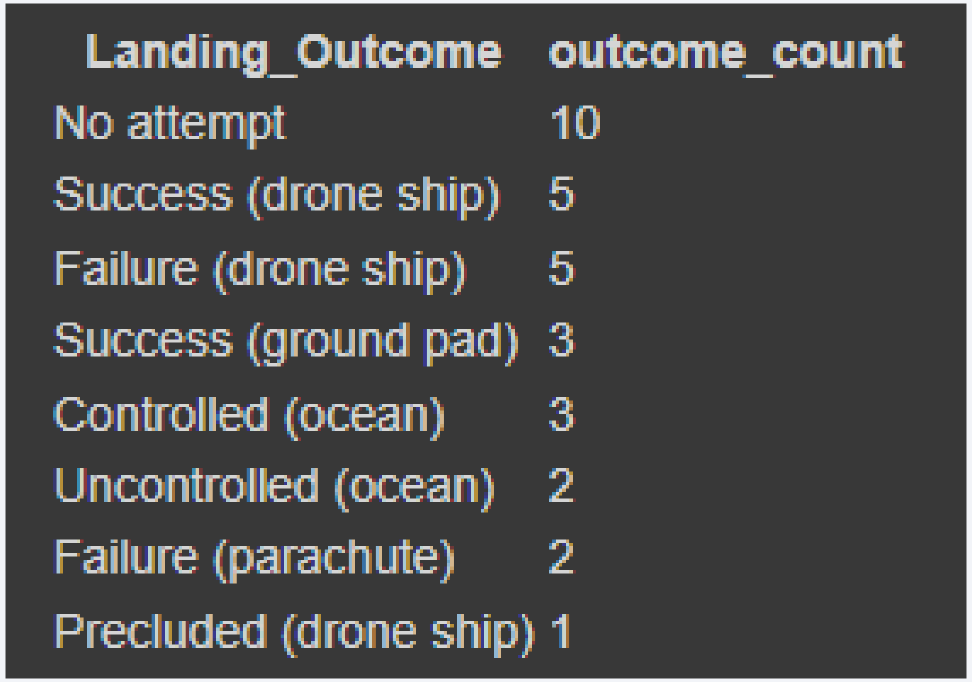
- The SQL code for this query may be seen in this slide.



| month_name | failure_landing_outcome | Booster_Version | Launch_Site |
|---|---|---|---|
| January | Failure (drone ship) | F9 v1.1 B1012 | CCAFS LC-40 |
| April | Failure (drone ship) | F9 v1.1 B1015 | CCAFS LC-40 |

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

- In this final query, we obtain the ranking of landing outcomes in descending order during a specific range of time.
- For this query we have combined many of the previously explained SQL functions. The SQL code may be seen in this slide.



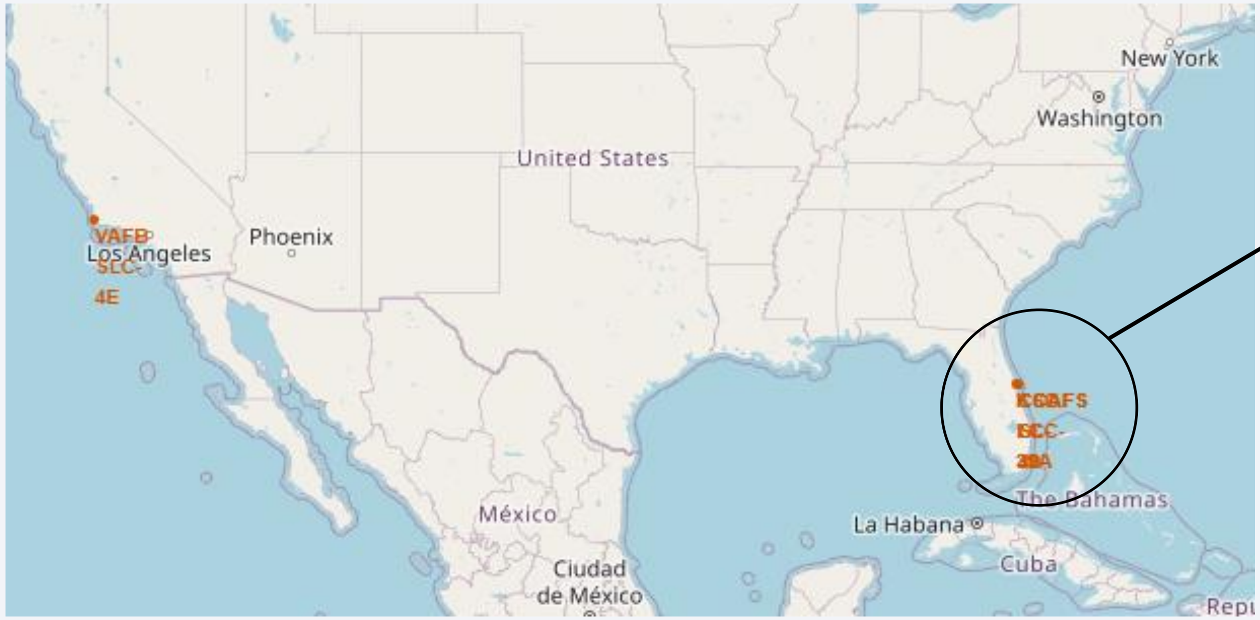
| Landing_Outcome | outcome_count |
|---|---|
| No attempt | 10 |
| Success (drone ship) | 5 |
| Failure (drone ship) | 5 |
| Success (ground pad) | 3 |
| Controlled (ocean) | 3 |
| Uncontrolled (ocean) | 2 |
| Failure (parachute) | 2 |
| Precluded (drone ship) | 1 |

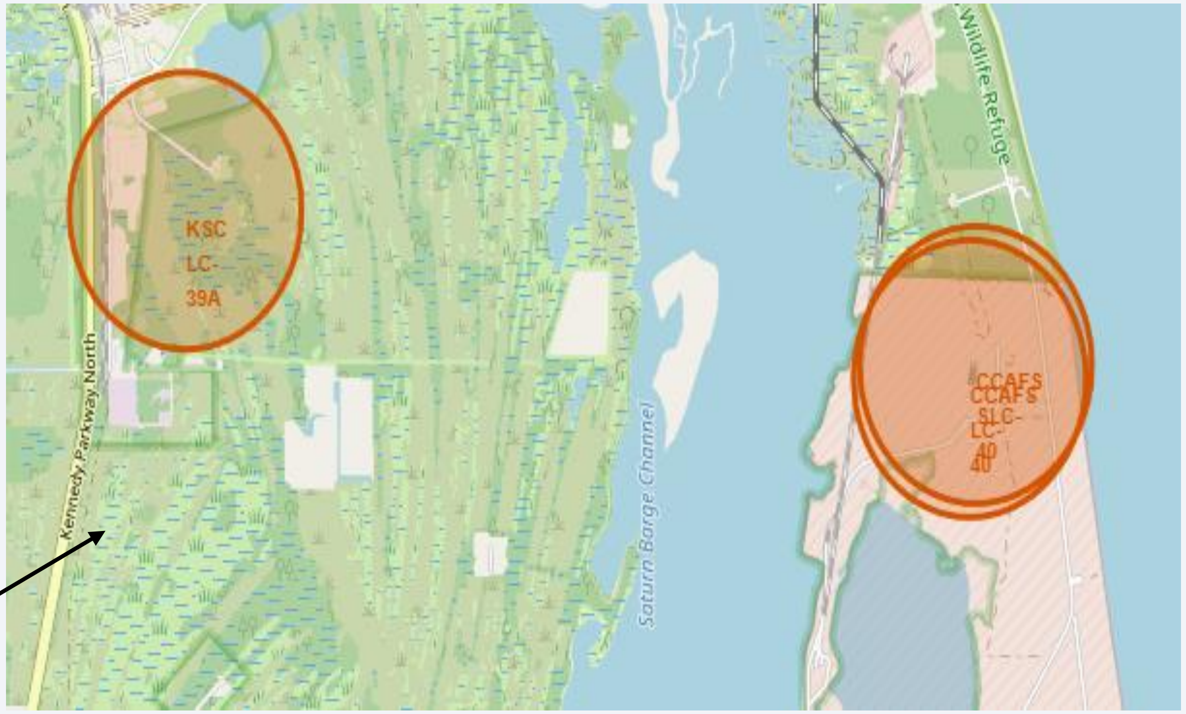
Section 3

## Launch Sites Proximities Analysis

# Launch Sites in the map

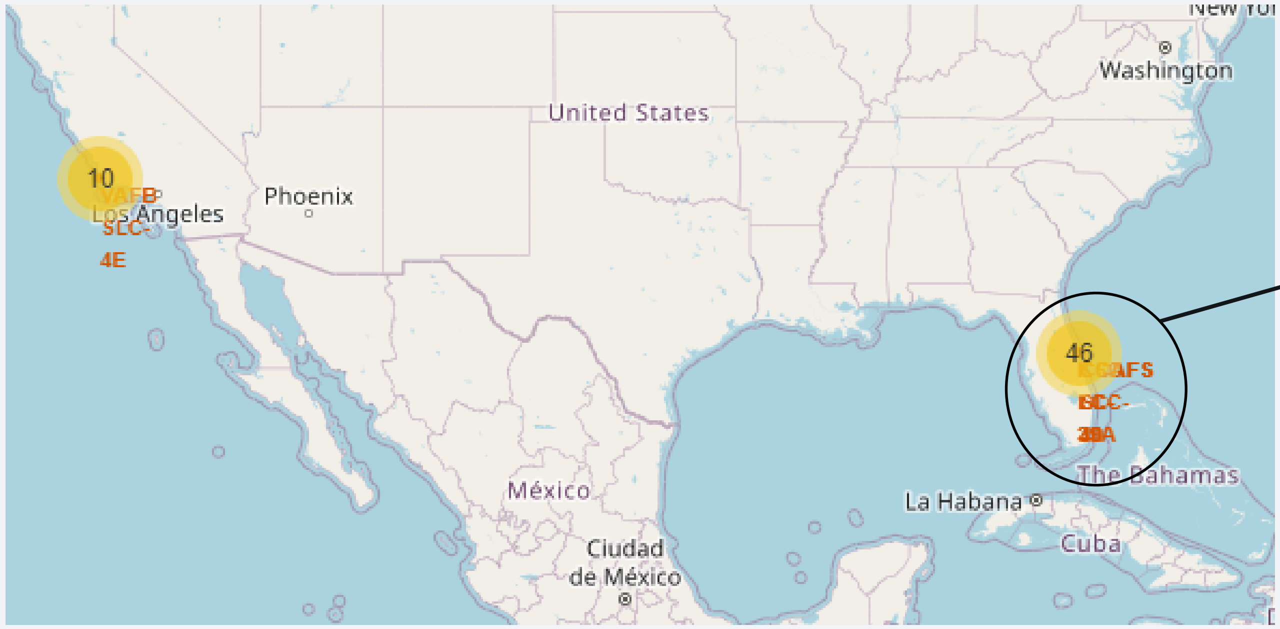
|   | Launch Site | Lat | Long |
|---|---|---|---|
| 0 | CCAFS LC-40 | 28.562302 | -80.577356 |
| 1 | CCAFS SLC-40 | 28.563197 | -80.576820 |
| 2 | KSC LC-39A | 28.573255 | -80.646895 |
| 3 | VAFB SLC-4E | 34.632834 | -120.610745 |

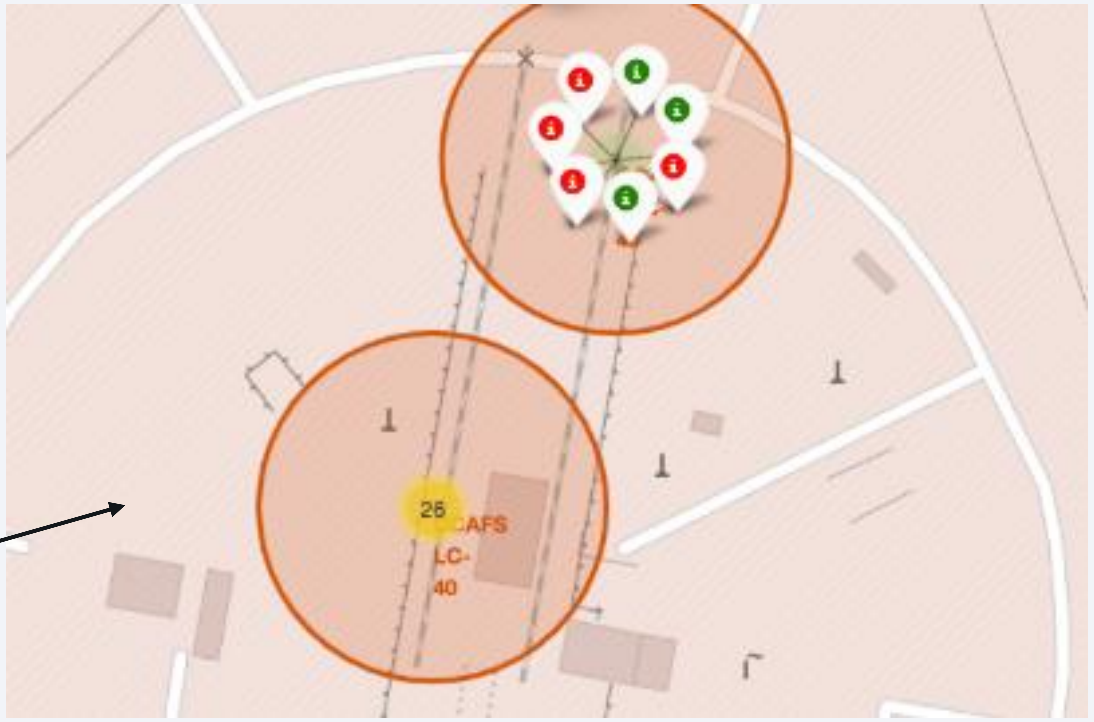




## Success and failure cases in the map

- Zooming the regions, the different locations are displayed. Clicking on the clusters the success/failure cases are shown.

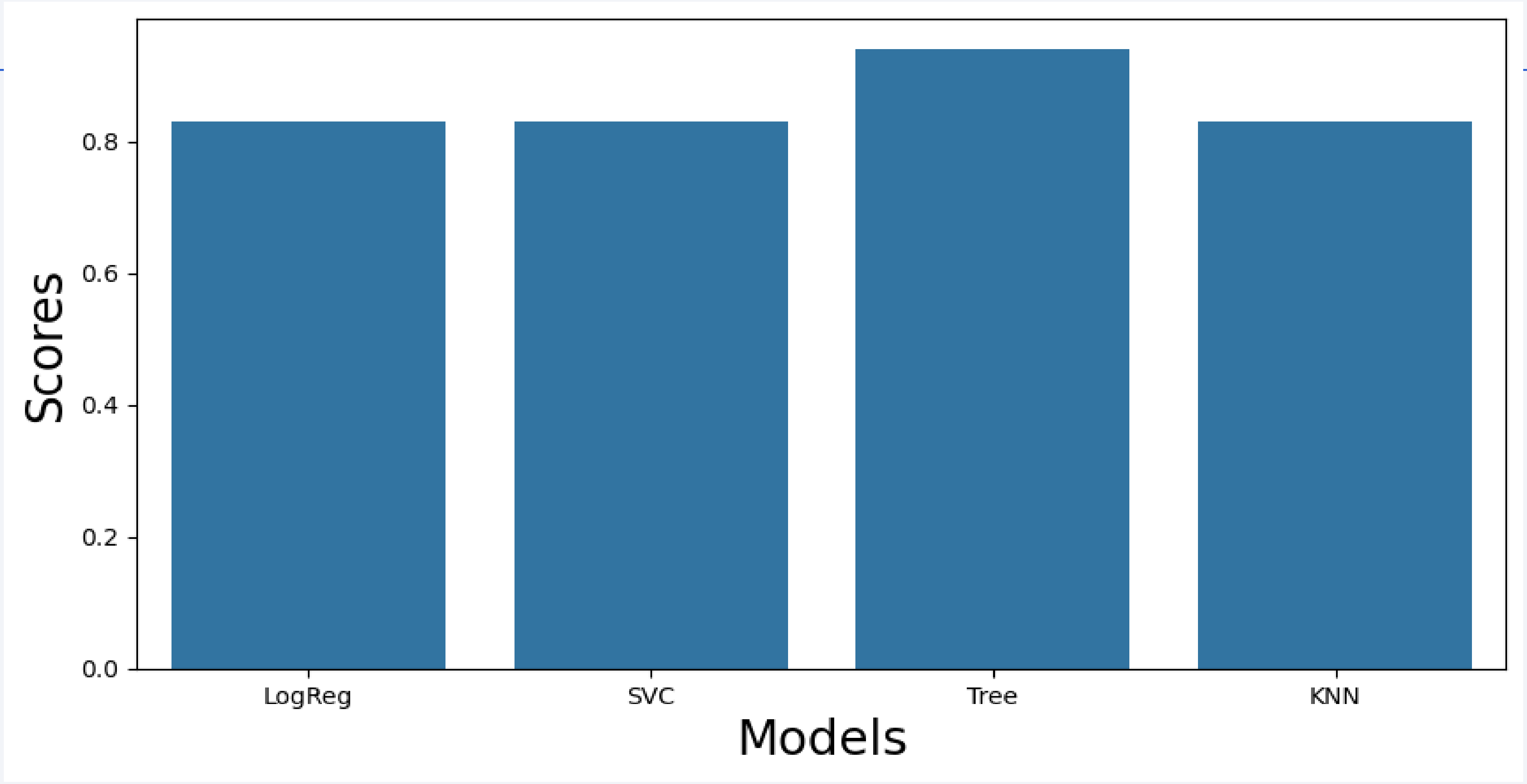




Section 5

## Predictive Analysis (Classification)

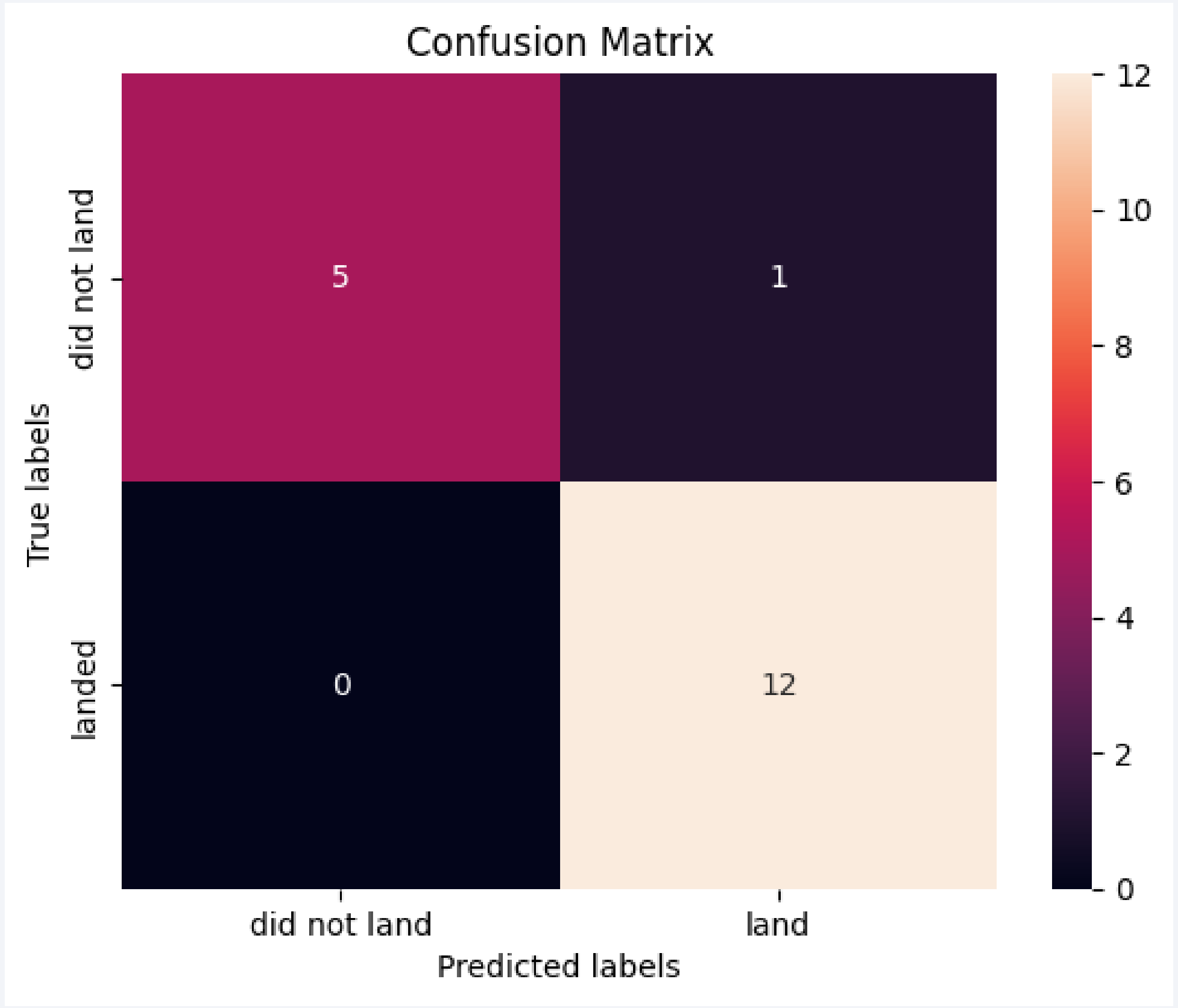
## Classification Accuracy



- The scores of each model are compared in a barplot, from which we conclude that the Decision Tree model is the optimal choice for this task.

# Confusion Matrix

- For a quantitative and visual evaluation of the **Decision Tree** we use the confusion matrix, which is provided by the Scikit-Learn module.
- The elements of this matrix are also known as: **True negatives** (5), **False positives** (1), **False negatives** (0) and **True positives** (12).
- Despite the high values of the score for all the models, the confusion matrix has established the model with the least number of **False positives**.



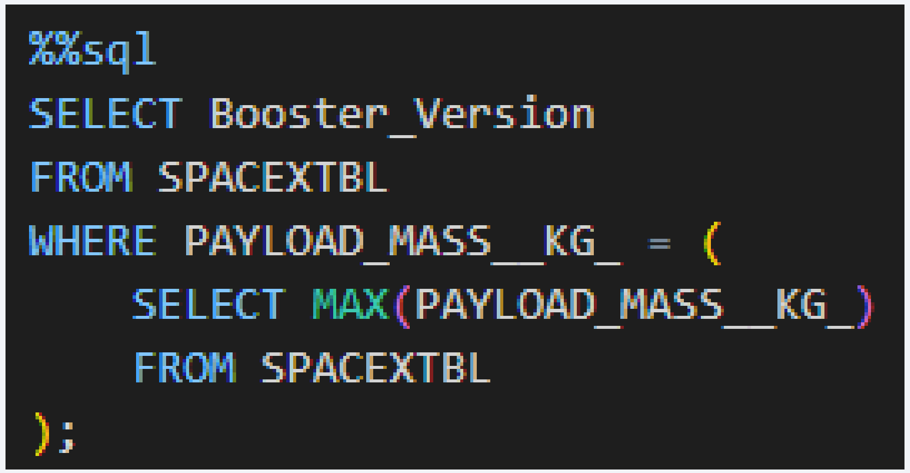
## Conclusions

- The main conclusion in this analysis is that the model with the best performance is the Decision Tree algorithm.
- We also have observed that the rate of success in the landing outcome has been increasing since 2013.
- We conclude that the Flight Number is not a correct indicative of the outcome, but other factors like the type of orbit, the payload mass and the landing site are.

# Appendix I: SQL codes

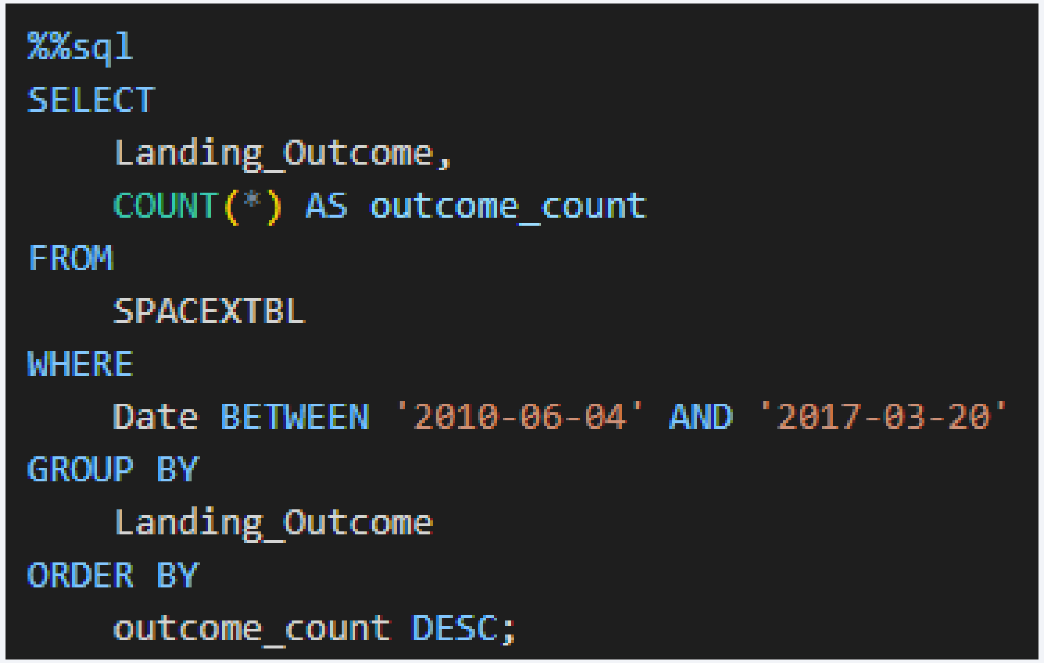
- We show the SQL code for the three last queries with the links to the slides where the results are shown.

Result here



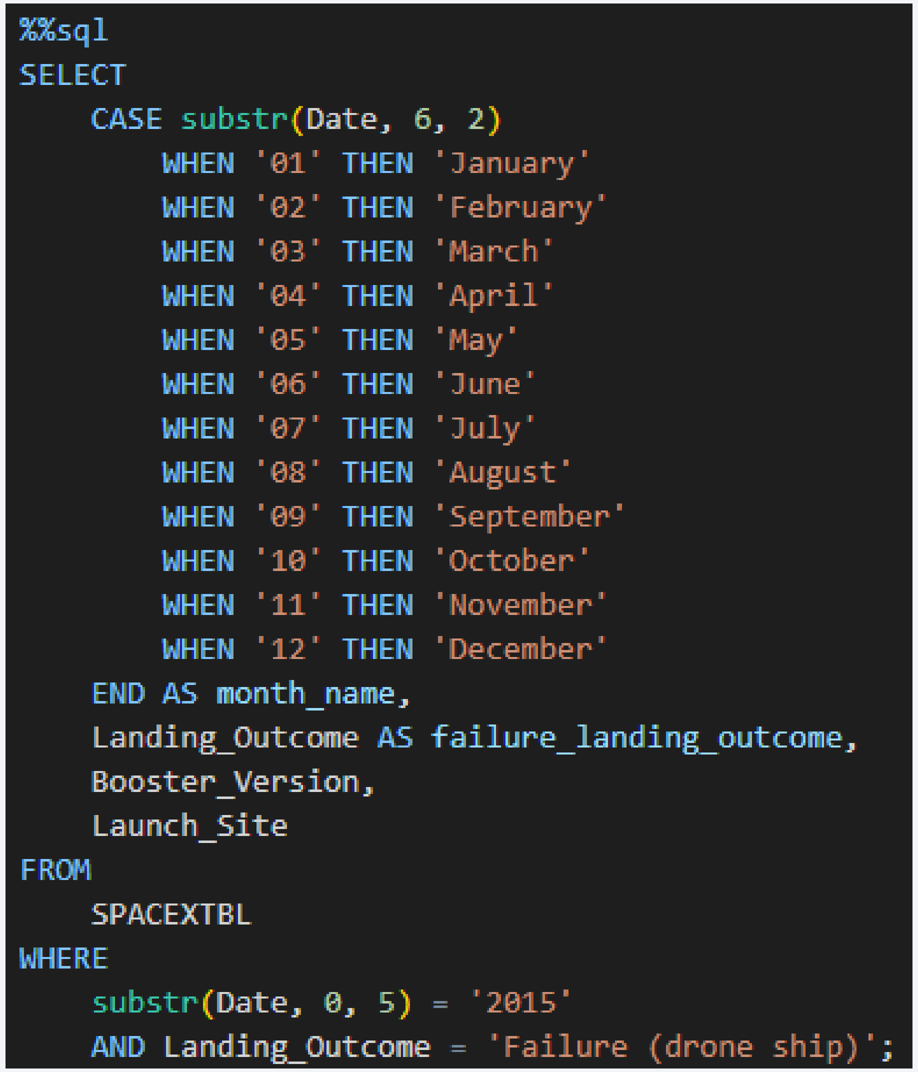
```sql
%%sql
SELECT Booster_Version
FROM SPACEXTBL
WHERE PAYLOAD_MASS__KG_ = (
    SELECT MAX(PAYLOAD_MASS__KG_)
    FROM SPACEXTBL
);
```

Result here



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

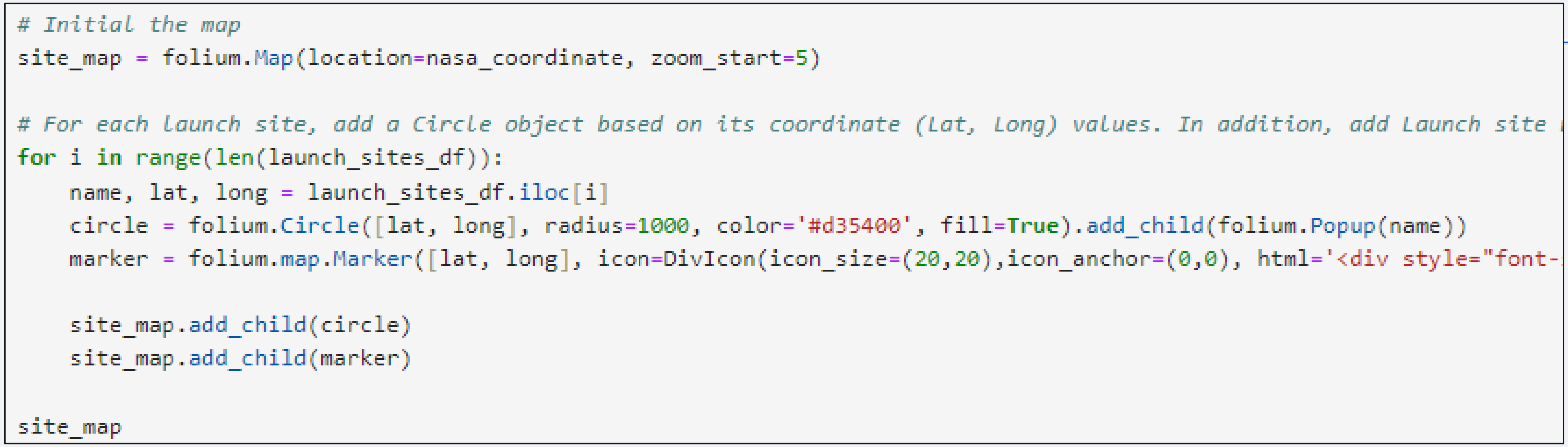
# Appendix I: SQL codes



```
%%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_name,
    Landing_Outcome AS failure_landing_outcome,
    Booster_Version,
    Launch_Site
FROM
    SPACEXTBL
WHERE
    substr(Date, 0, 5) = '2015'
    AND Landing_Outcome = 'Failure (drone ship)';
```

Result here

# Appendix II: Folium codes



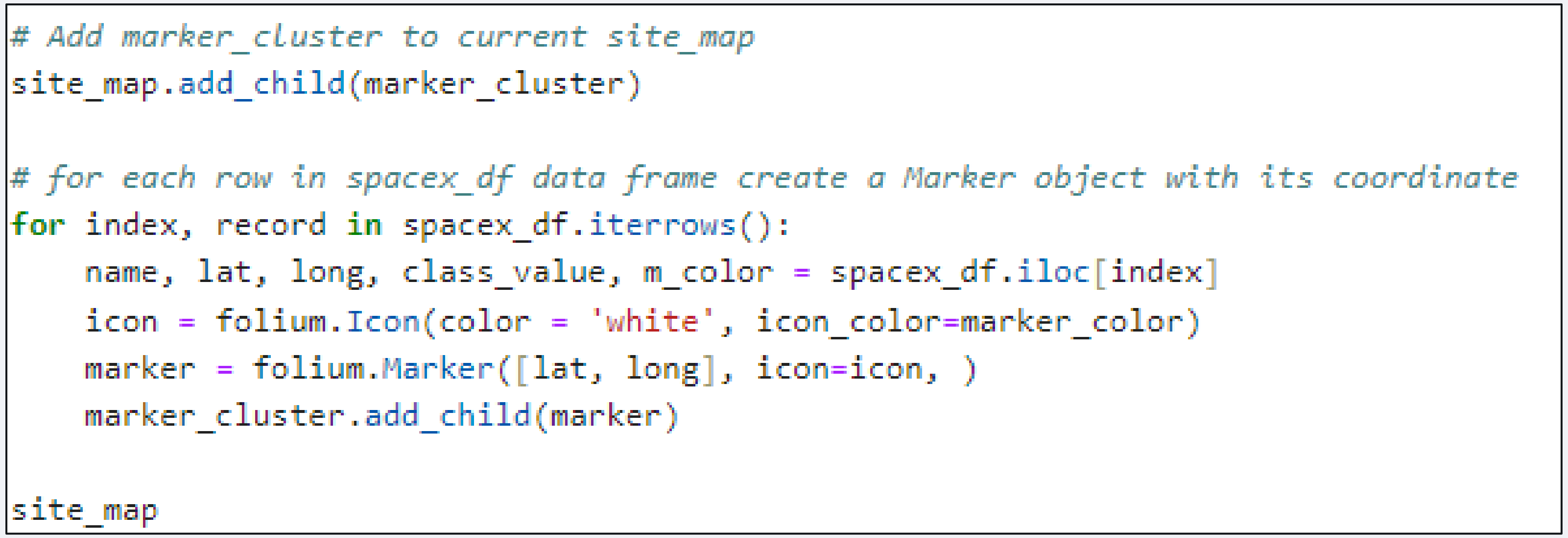
```
# Initial the map
site_map = folium.Map(location=nasa_coordinate, zoom_start=5)

# For each launch site, add a Circle object based on its coordinate (Lat, Long) values. In addition, add Launch site
for i in range(len(launch_sites_df)):
    name, lat, long = launch_sites_df.iloc[i]
    circle = folium.Circle([lat, long], radius=1000, color='#d35400', fill=True).add_child(folium.Popup(name))
    marker = folium.map.Marker([lat, long], icon=DivIcon(icon_size=(20,20),icon_anchor=(0,0), html='<div style="font-

    site_map.add_child(circle)
    site_map.add_child(marker)


site_map
```

Result here



```
# Add marker_cluster to current site_map
site_map.add_child(marker_cluster)

# for each row in spacex_df data frame create a Marker object with its coordinate
for index, record in spacex_df.iterrows():
    name, lat, long, class_value, m_color = spacex_df.iloc[index]
    icon = folium.Icon(color = 'white', icon_color=marker_color)
    marker = folium.Marker([lat, long], icon=icon, )
    marker_cluster.add_child(marker)

site_map
```

Result here

## Thank you!