IBM Developer
SKILLS NETWORK

Winning Space Race with Data Science

Anton Kliabeka
2022-05-09
LinkedIn: https://www.linkedin.com/in/anton-kliabeka-69a082155/

Outline

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

Executive Summary

- **Summary of methodologies:**
  - Data collection;
  - Data wrangling;
  - EDA with data visualization;
  - EDA with SQL;
  - Building an interactive map with Folium;
  - Building a Dashboard with Plotly Dash;
  - Predictive analysis (Classification);
- **Summary of all results:**
  - EDA results;
  - Analytics in screenshots;
  - Predictive analysis results;

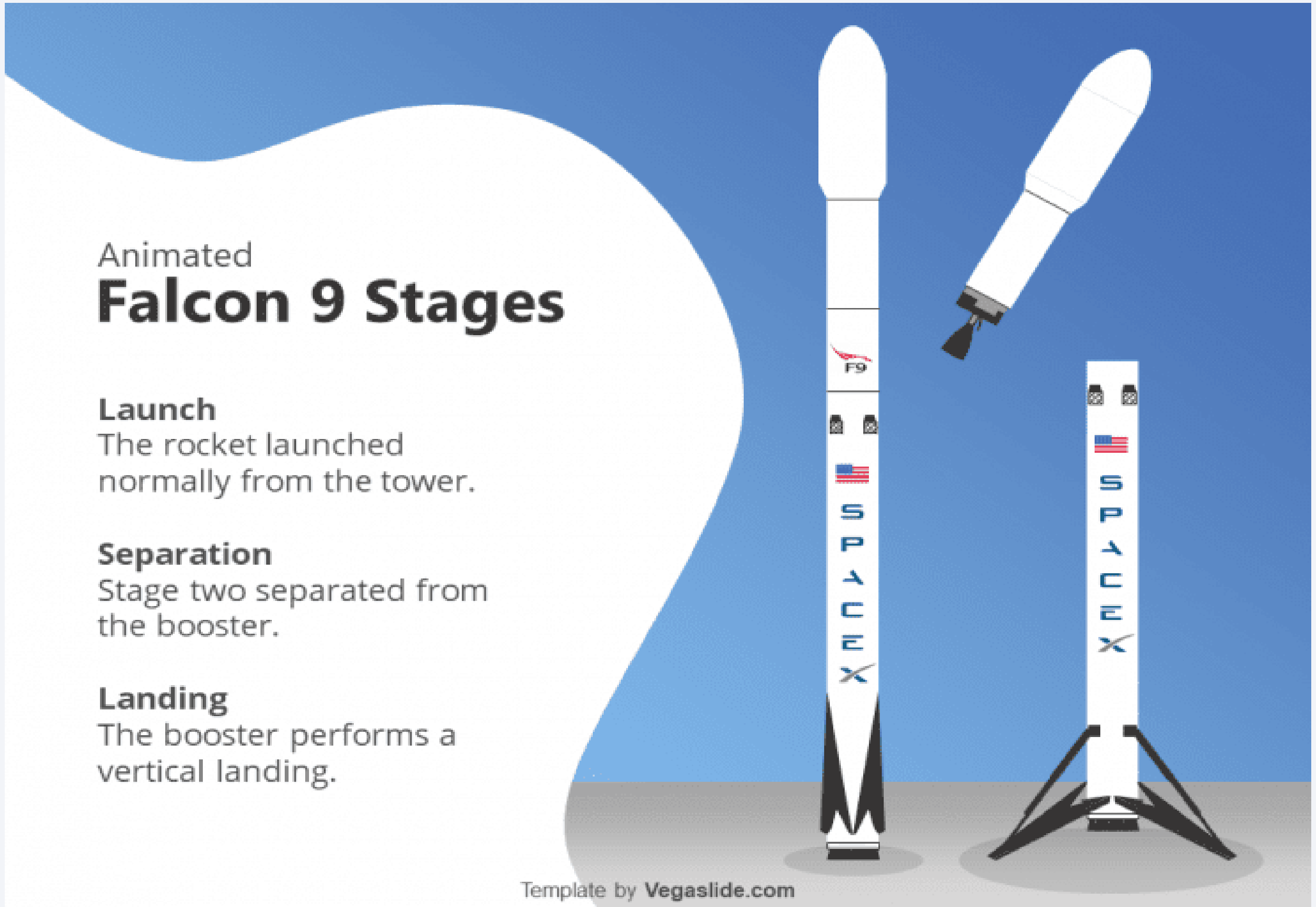
Introduction

<u>Project background and context</u>:

- The commercial space age is here, companies are making space travel affordable for everyone;
- Perhaps the most successful is SpaceX;
- SpaceX launch costs about 65$ million, the others - about 165$ million.
- Much of the savings is because SpaceX can reuse the first stage - SpaceX's Falcon 9 can recover the first stage.
- The first stage is quite large (much larger than the second stage) and expensive, and does most of the work.

Introduction

<u>Project background and context</u>:



Introduction

<u>Project goal</u>:

- To answer the question "*Is it possible to determine if the first stage will land?*" with the help of SpaceX's Falcon 9 launch data. The answer will help to determine the cost of a launch.

Section 1

Methodology

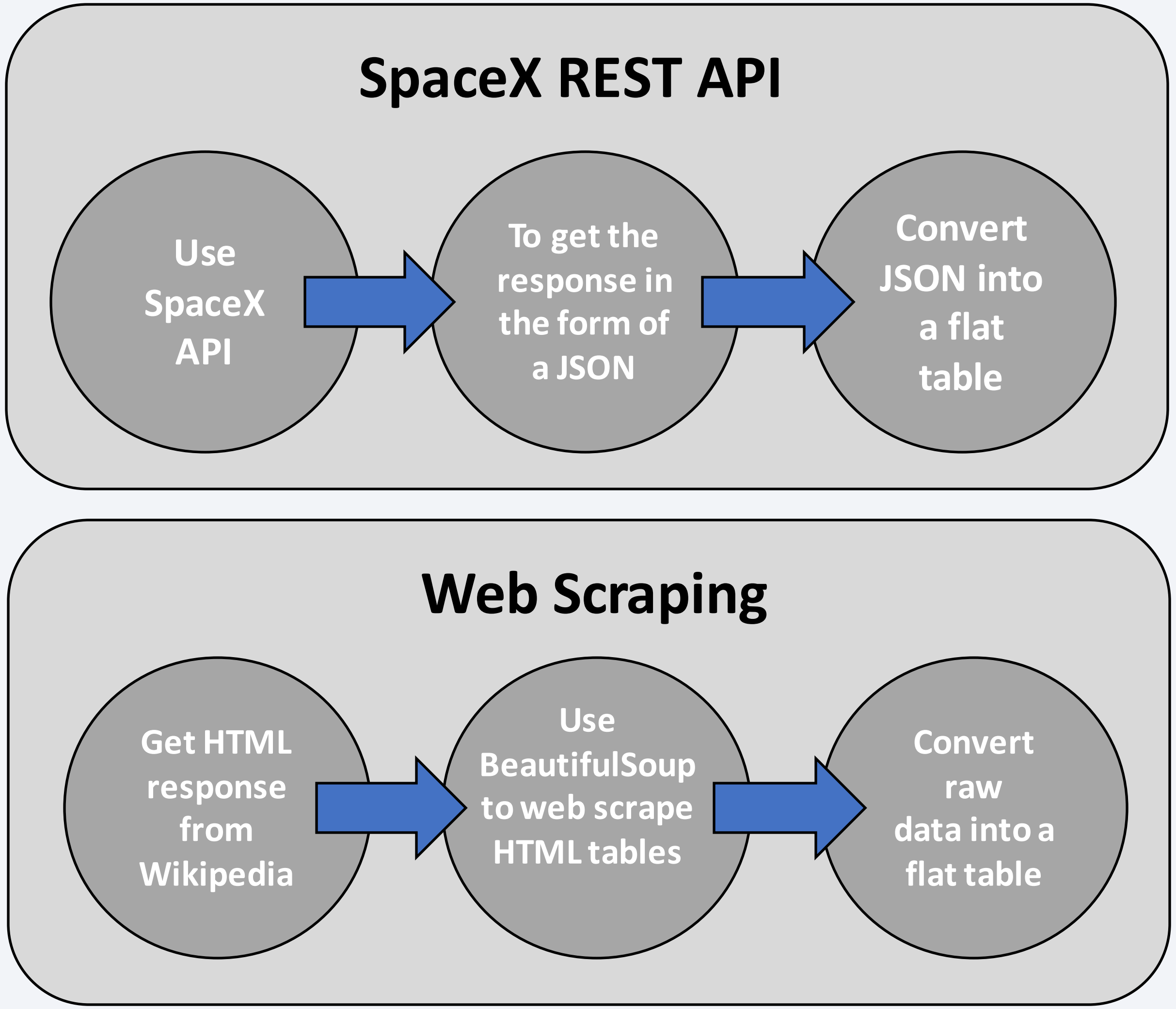
Methodology

- Data collection methodology:
    - **SpaceX REST API**;
    - **Web scraping** related Wikipedia pages;
- Perform data wrangling:
    - **Converting** Landing **Outcomes into Classes** (either 0 or 1);
    - Applying **One-Hot Encoding** technique to categorical predictors;
- Perform exploratory data analysis (EDA) using visualization and SQL:
    - **Using** different types of **charts and diagrams**, and **SQL queries** to **show relationship** between variable, to **reveal patterns** of the data, and to **understand the data**;
- Perform interactive visual analytics using Folium and Plotly Dash;
- Perform predictive analysis using classification models:
    - **Standardizing data**, **splitting data** into training and test sets, **hyperparameters tuning**, **evaluation** and **finding the method performs best**;

Data Collection

- **SpaceX REST API** lets us **get data about launches**, including information about the rocket used, payload delivered, launch specifications, landing specifications, and landing outcome;
- **Web scraping** related Wikipedia pages - r**equest a get response from Wikipedia** and using the Python **BeautifulSoup** package to web scrape some **HTML tables**, parsing the data from those tables and **convert them into a Pandas datframe** for further visualization and analysis;
- **Transform raw data** into a **clean dataset** which provides meaningful data;
- **Filter** the data to have **only Falcon 9** launches;
- Dealing with **NULL** values;

Data Collection – General Flowcharts



Data Collection – SpaceX API

1. Get response from the API

```
import requests
static_json_url = "https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/datasets/API_call_spacex_api.json"
Response_static = requests.get(static_json_url)
```

2. Import to a Pandas dataframe

```
import pandas as pd
df_static = pd.json_normalize(response_static.json())
```

3. Get clean dataset which provides meaningful data

```
# using custom functions
getLaunchSite(data)
getPayloadData(data)
getCoreData(data)

# create a dictionary and fill it with data
launch_dict = {...}
df_launch_data = pd.DataFrame(data = launch_dict)

# filter the dataframe to only include Falcon 9 launches
data_falcon9 = df_launch_data[
        df_launch_data['BoosterVersion'] == 'Falcon 9'
].reset_index(drop = True)

# dealing with null values
Import numpy as np
PayloadMass_mean = data_falcon9['PayloadMass'].mean()
data_falcon9['PayloadMass'].replace(np.nan, PayloadMass_mean, inplace = True)

# save as csv
data_falcon9.to_csv('falcon9_dataset_part_1.csv', index = False)
```

- GitHub URL of the completed SpaceX API calls notebook;

Data Collection – Web Scraping

1. Get response from Wikipedia

```
import requests
static_url =
"https://en.wikipedia.org/w/index.php?title=List_of_Falcon_
9_and_Falcon_Heavy_launches&oldid=1027686922"
Response_static = requests.get(static_url)
```

2. Use BeautifulSoup to web scrape HTML tables

```
from bs4 import BeautifulSoup as bs
data_bs = bs(static_response.text, 'html5lib')
# find all objects with type 'table'
html_tables = data_bs.find_all('table')
# exctarct column names
column_names = []
for row in first_launch_table.find_all('th'):
    name = extract_column_from_header(row)
    if name and len(name) > 0:
        column_names.append(name)
```

3. create a data frame by parsing the launch HTML tables

```
import pandas as pd

# extract records from table rows
launch_dict = dict.fromkeys(column_names)

extracted_row = 0
for table_number, table in enumerate(data_bs.find_all('table', "wikitable
plainrowheaders collapsible")):
  # get table row
   for rows in table.find_all("tr"):
      # check if first table heading is a number corresponding to a lch number
      if rows.th:
         if rows.th.string:
            flight_number = rows.th.string.strip()
            flag = flight_number.isdigit()
      else:
         flag = False
     {...some logic...}

# save data
df = pd.DataFrame(launch_dict)
df.to_csv('spacex_web_scraped.csv', index = False)
```

- GitHub URL of the completed Web Scraping notebook;

Data Wrangling

- **Appling One-Hot Encoding** to categorical columns;
- **Convert Landing Outcomes** to **Classes** (either 0 or 1):
  - 0 is a bad outcome, that is, the booster did not land;
  - 1 is a good outcome, that is, the booster did land.

1. Create dummy variables to categorical columns

```
# read data
...

# Create dummy variables to categorical columns
# columnslist - list of categorical columns
features_one_hot.drop(columns = ['GridFins', 'Reused',
'Legs'], inplace = True)


for column in columnslist:
    features_one_hot = pd.concat(
        [
            features_one_hot,
            pd.get_dummies(features[column], prefix = column)
        ], axis = 1
    )
```

2. Converting Landing Outcomes into Classes (0 or 1)

```
# read data
...

# create classification variable from the "Outcome" column
landing_class = [0 if item in bad_outcomes else 1 for item in df['Outcome']]
df['Class'] = landing_class
```


- GitHub URL 1 and URL 2 of a data wrangling related notebooks;

EDA with Data Visualization

- Plotted charts:
  - **Bar chart** "*Success Rate by Orbit Type*" (shows **relationship between success rate of each Orbit Type**);
  - **Linear Plot** "*Success yearly trend*" (shows the **success rate and its trend** since 2013);
  - **Scatter Plots** (How much **one variable is affected by another**):

    - "Flight Number" vs "Payload Mass";

    - "Flight Number" vs "Launch Site";

    - "Launch Site" vs "Payload Mass";

    - "Flight Number" vs "Orbit Type";

    - "Payload" vs "Orbit Type";

- GitHub URL completed EDA notebook with data visualization;

EDA with SQL

- List of performed **SQL queries** to understand the dataset:
  - Names of the unique launch sites in the space mission;
  - 5 records where launch sites begin with the string 'CCA';
  - Total payload mass carried by boosters launched by NASA (CRS);
  - Average payload mass carried by booster version F9 v1.1;
  - The date when the first successful landing outcome in ground pad was achieved;
  - Names of the boosters which have success in drone ship and have payload mass greater than 4000 but less than 6000;
  - Total number of successful and failure mission outcomes;
  - Names of the Booster Versions which have carried the maximum payload mass using a subquery;
  - Failed Landing Outcomes in drone ship, their booster versions, and launch site names for in year 2015;
  - 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;
- GitHub URL of completed EDA with SQL notebook;

Build an Interactive Map with Folium

- The **launch success rate may depend on the location and proximities of a launch site**, i.e., the initial position of rocket trajectories. Finding an optimal location for building a launch site certainly involves many factors and we could discover some of the factors by analyzing the existing launch site locations;
- Map objects added to a folium map:
  - All launch sites – *Circle* objects;
  - Success/failed launches for each site – *Marker Cluster* objects;
  - The distances between a launch site to its proximities – *Line* objects;
- GitHub URL of completed interactive map with Folium map;

Build a Dashboard with Plotly Dash

- Plotted charts used in the dashboard:
  - **Scatter Plot** showing **relationship** between **Outcome and Payload Mass** for different **Booster Versions.** It shows the correlation between payload mass and launch success;
  - **Pie Chart** showing:

    - **Total successful launches** count for **all sites**;

    - **Success vs. Failed** counts for a **specific site**;
- GitHub URL of Plotly Dash lab source code;

Predictive Analysis (Classification)

PREPARATIONS:

- **Load** and **Transform** data (using *pandas* and *numpy*);
- **Split into train** and **test** (using *scikit-learn*);
- **Choose** ML **algorithms**;

HYPERPARAMETERS TUNING:

- Use **GridSearchCV** to choose **the best set of parameters**;
- **Fit** train **dataset** to **train models**;

PERFOMANCE EVALUATION:

- Check the **Accuracy** Score of each model;
- Check **Confusion Matrix**;

If the quality of the model is not good enough!

FINDING THE BEST PERFORMING MODEL:

- A **model** with **the best Accuracy Score** is considered as **the best performing** model;

- GitHub URL of completed predictive analysis lab;

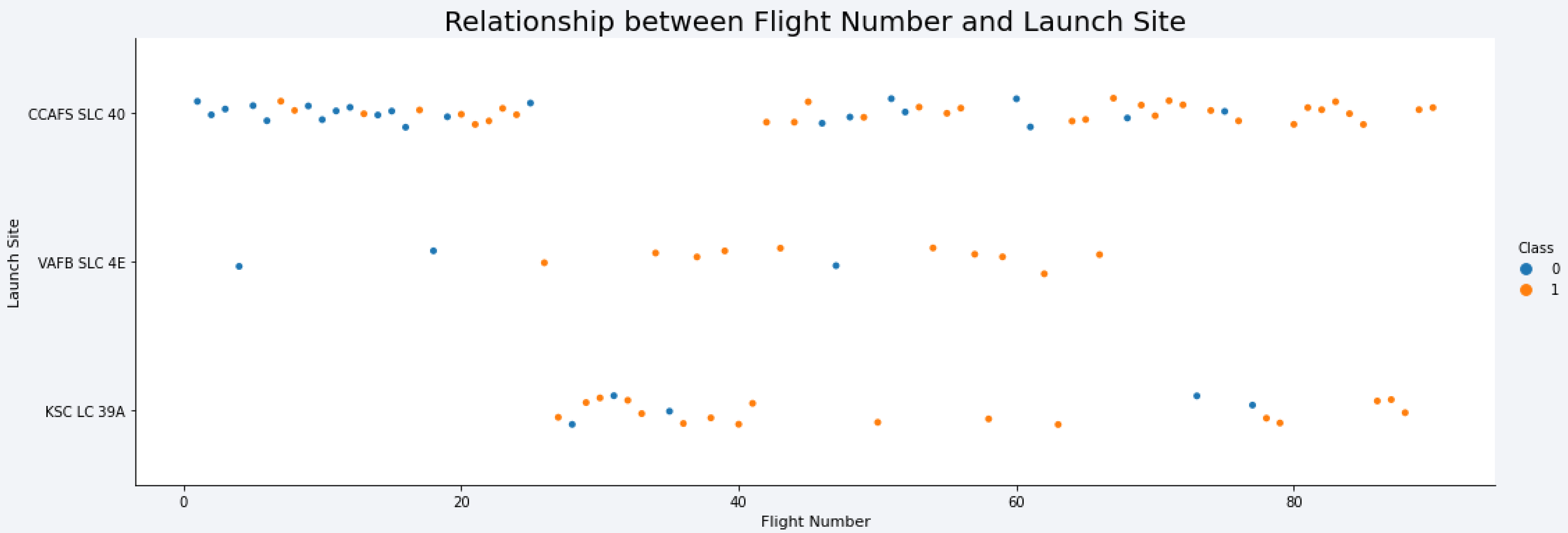
Results

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

Section 2

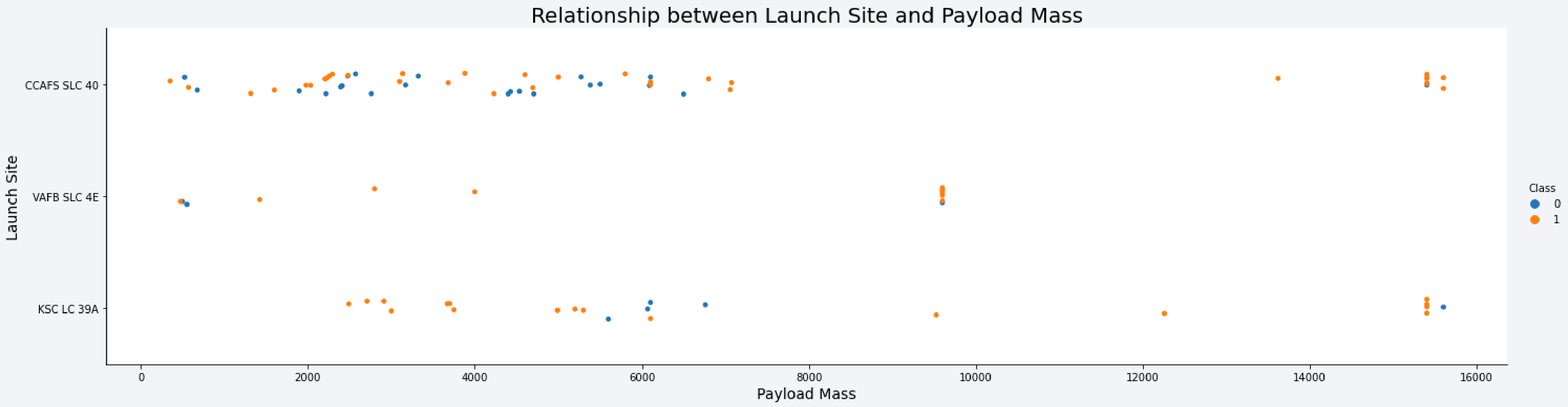
Insights drawn from EDA

Flight Number vs. Launch Site



- Different Launch Sites have different successful rate;
- First 35 Flight Numbers were not very successful;
- Starting from Flight Number 35 the successful tate shows strong improvement in general;

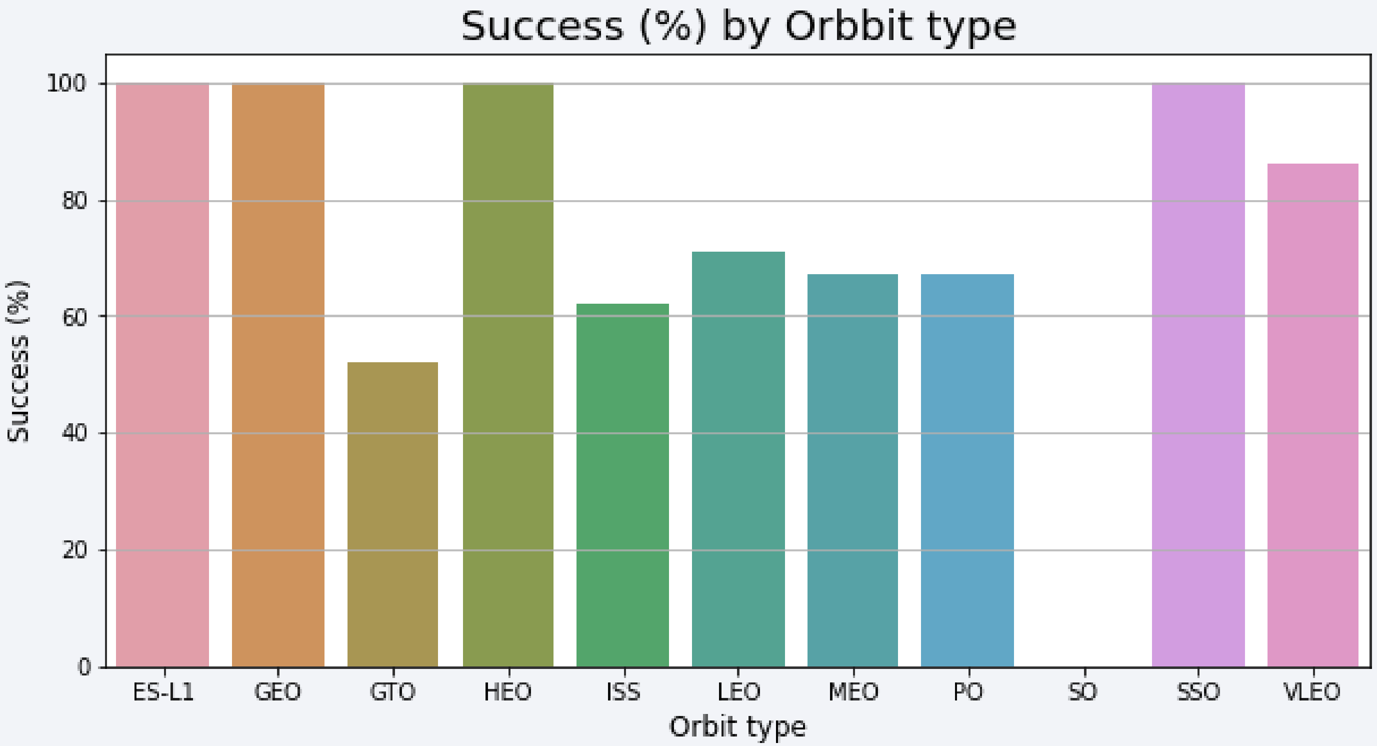
Payload vs. Launch Site



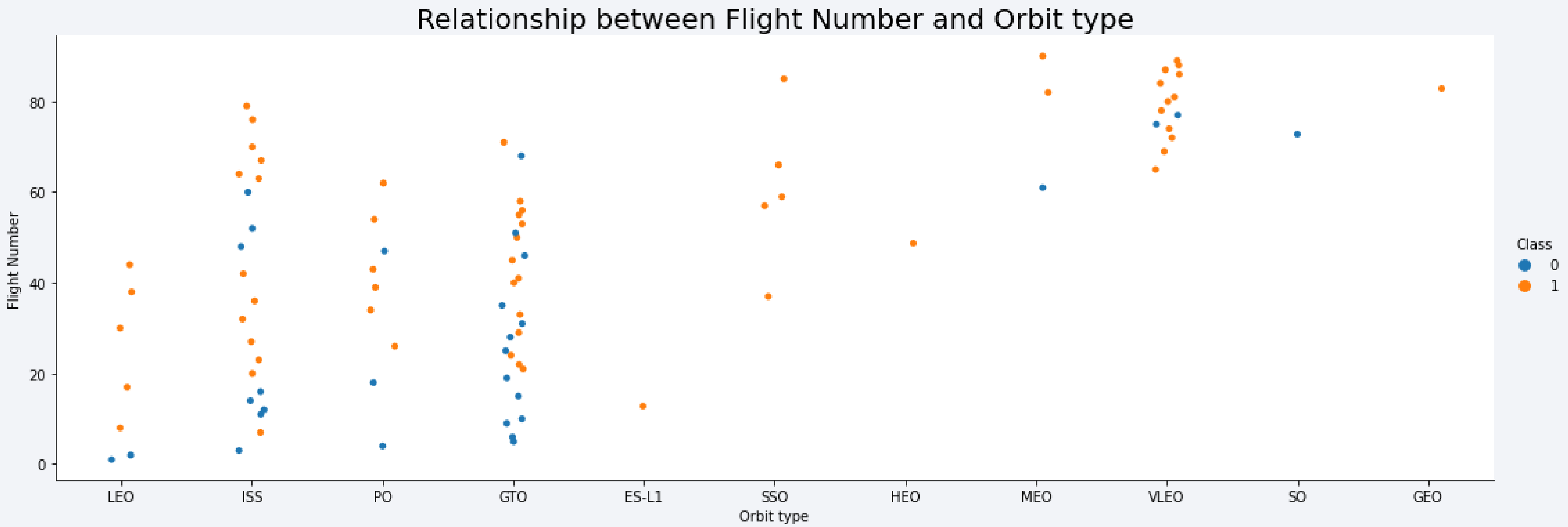
- It feels like the **greater the payload mass**, the **higher the success rate**, but there is **no a clear pattern to be found** to say for sure if a success launch is dependent on the combination of a launch site and payload mass;

Success Rate vs. Orbit Type

- Orbit types *ES-L1* (1), *GEO* (1), *HEO* (1), *SSO* (5) have **the highest success %** - **100%**;
- Orbit type *SO* (1) has **the lowest success %** - **0%**;
- Orbit types' success % with 5 or more cases by success %:
    1. SSO (5) - 100%;
    2. VLEO(14) - 86%;
    3. LEO (7) - 71,5%;
    4. PO (9) - 67%;
    5. ISS (21) - 62%;
    6. GTO (27) - 52%;

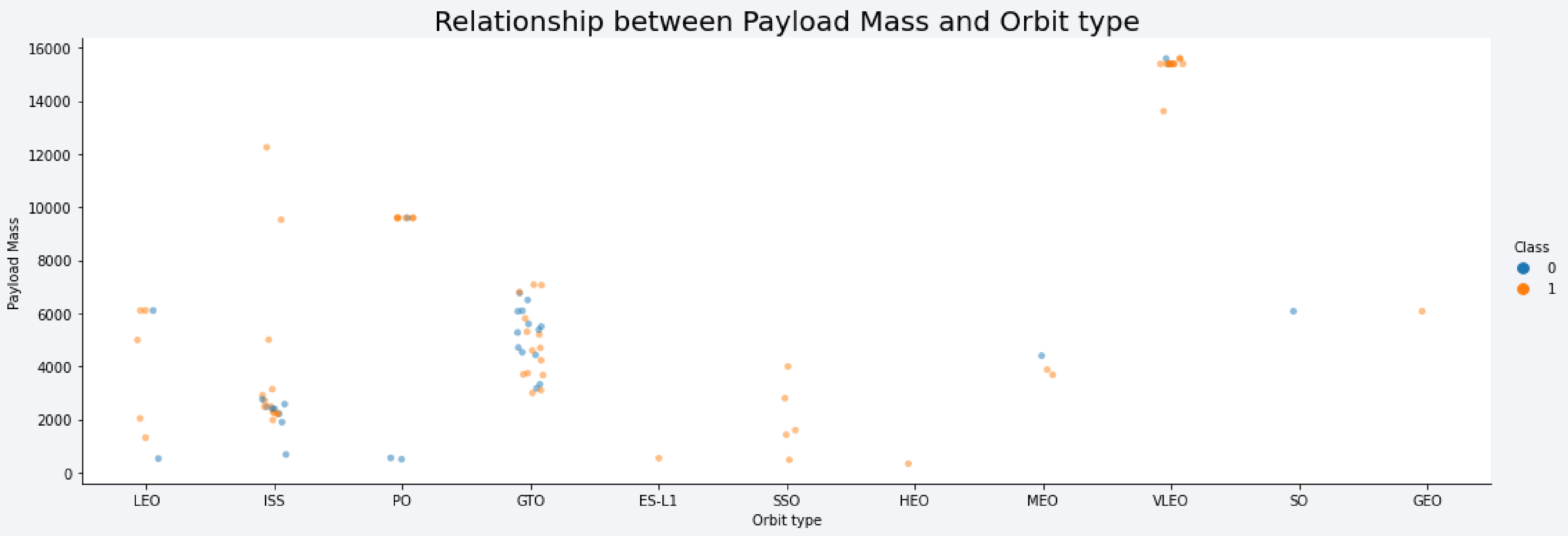


Flight Number vs. Orbit Type



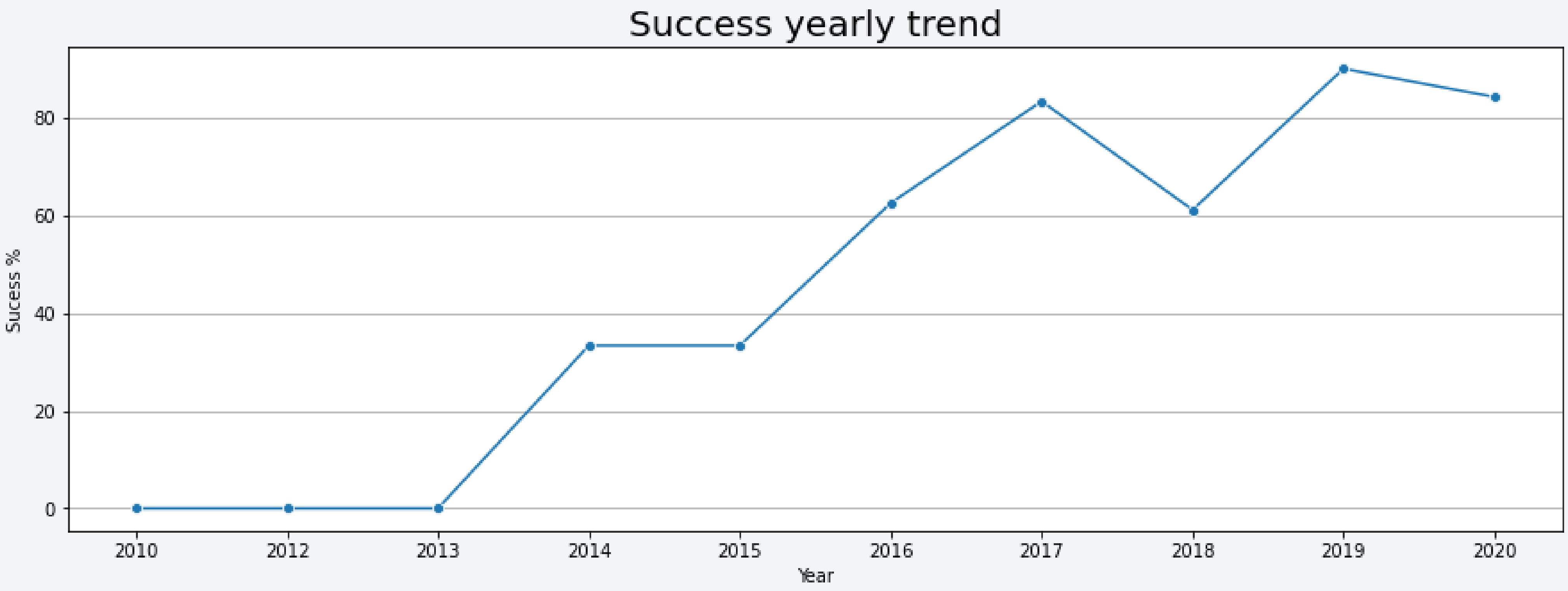
- In the **LEO orbit** the **success appears to be related to the number of flights**;
- In the **GTO orbit there is no relationship between flight number**;

Payload vs. Orbit Type



- With **heavy payloads** the **positive landing** rates are **higher** for **Polar, LEO and ISS**;
- For the **GTO** it is **not possible to distinguish effect of the payload mass**;

Launch Success Yearly Trend



- The **general trend** of the success rate **kept increasing since 2013** till 2020 with a drawdown in 2018;

All Launch Site Names

QUERY:

```
select distinct(Launch_Site)
from Spacex;
```

RESULT:

('CCAFS LC-40',)
('VAFB SLC-4E',)
('KSC LC-39A',)
('CCAFS SLC-40',)

- Display the **names of the unique launch sites** in the space mission;
- The select **DISTINCT** statement is used to return only distinct (different) values;

Launch Site Names Begin with 'CCA'

QUERY:

```
select *
from Spacex
where Launch_Site like ('CCA%')
limit 5;
```

- Display **5 records** where **launch sites begin with** the string **'CCA'**;
- The **LIMIT** clause is used to select a limited number of records;
- The **LIKE** operator is used in a **WHERE** clause to search for a specified pattern in a column;

RESULT:

('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

QUERY:

select Customer, sum(PAYLOAD_MASS__KG_)
from Spacex
where Customer = 'NASA (CRS)';

RESULT:

('NASA (CRS)', 45596)

- Display the **total payload mass** carried by **boosters launched by NASA (CRS)**;
- The **SUM()** function returns the total sum of a numeric column;

Average Payload Mass by F9 v1.1

<u>QUERY</u>:

```
select Booster_Version, avg(PAYLOAD_MASS__KG_)
from Spacex
where Booster_Version = 'F9 v1.1';
```

<u>RESULT</u>:

('F9 v1.1', 2928.4)

- Display **average payload mass** carried by **booster version F9 v1.1**;
- The **AVG()** function returns the average value of a numeric column;

First Successful Ground Landing Date

<u>QUERY</u>:

```
select min(date(substr(Date, 7) || "-" || substr(Date, 4, 2) || "-" || substr(Date, 1, 2)))
from Spacex
where landing__outcome = 'Success (ground pad)'
```

<u>RESULT</u>:

('2015-12-22')

- List **the date** when **the first successful landing outcome** in **ground pad** was acheived;
- The **MIN()** function returns the smallest value of the selected column;
- The **SUBSTR()** function extracts a substring from a string (starting at any position);

Successful Drone Ship Landing with Payload between 4000 and 6000

QUERY:

```
select distinct(Booster_Version)
from Spacex
where 1 = 1
  and upper(Landing__Outcome) like ('SUCCESS%')
  and upper(Landing__Outcome) like ('%DRONE SHIP%')
  and PAYLOAD_MASS__KG_ > 4000
  and PAYLOAD_MASS__KG_ < 6000;
```

RESULT:

('F9 FT B1022',)
('F9 FT B1026',)
('F9 FT B1021.2',)
('F9 FT B1031.2',)

- List the **names of the boosters** which have **success in drone ship** and have **payload mass greater** than **4000** but **less** than **6000;**
- The **1=1** is alway **True** (google it for more info);
- The **UPPER()** function converts a string to upper-case;

Total Number of Successful and Failure Mission Outcomes

QUERY:

```
select 'Success' as outcom, count(*) as item_cnt
from Spacex
where upper(Mission_Outcome) like ('SUCCESS%')
union
select 'Failure' as outcom, count(*) as item_cnt
from Spacex
where upper(Mission_Outcome) like ('FAILURE%');
```

RESULT:

('Success', 100)
('Failure', 1)

- List the **total number of successful and failure mission outcomes**;
- The **UNION** operator is used to combine the result-set of two or more SELECT statements;

Boosters Carried Maximum Payload

QUERY:

```
select distinct(Booster_Version), PAYLOAD_MASS__KG_
from Spacex
where PAYLOAD_MASS__KG_ = (
   select max(PAYLOAD_MASS__KG_)
   from Spacex
);
```

RESULT:

('F9 B5 B1048.4', 15600)
('F9 B5 B1049.4', 15600)
('F9 B5 B1051.3', 15600)
('F9 B5 B1056.4', 15600)
('F9 B5 B1048.5', 15600)
('F9 B5 B1051.4', 15600)
('F9 B5 B1049.5', 15600)
('F9 B5 B1060.2 ', 15600)
('F9 B5 B1058.3 ', 15600)
('F9 B5 B1051.6', 15600)
('F9 B5 B1060.3', 15600)
('F9 B5 B1049.7 ', 15600)

- List the **names of the booster_versions** which have carried the **maximum payload mass;**
- A **Subquery** or **Inner query** or a **Nested query** is a **query within another SQL query** and embedded within the WHERE clause;
- The **MAX()** function returns the largest value of the selected column;

2015 Launch Records

QUERY:

```
select Landing__Outcome, Booster_Version, Launch_Site, date
from Spacex
where 1 = 1
  and upper(Landing__Outcome) like ('%DRONE SHIP%')
  and upper(Landing__Outcome) like ('FAILURE%')
  and date like '%2015';
```

RESULT:

('Failure (drone ship)', 'F9 v1.1 B1012', 'CCAFS LC-40', '10-01-2015')
('Failure (drone ship)', 'F9 v1.1 B1015', 'CCAFS LC-40', '14-04-2015')

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

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

QUERY:

```
select Landing__Outcome, count(*) as Landing__Outcome_count
from Spacex
where 1 = 1
  and date(substr(Date, 7) || "-" || substr(Date, 4, 2) || "-" || substr(Date, 1, 2)) between '2010-06-04' and '2017-03-20'
group by Landing__Outcome
order by Landing__Outcome_count desc;
```

- Rank the **count of landing outcomes between** the date **2010-06-04** and **2017-03-20**, in **descending order**;
- The **BETWEEN** operator selects values within a given range;
- The **GROUP BY** statement groups rows that have the same values into summary rows;
- The **ORDER BY** keyword is used to sort the result-set in ascending or descending order;
- To sort the records in descending order, use the **DESC** keyword;

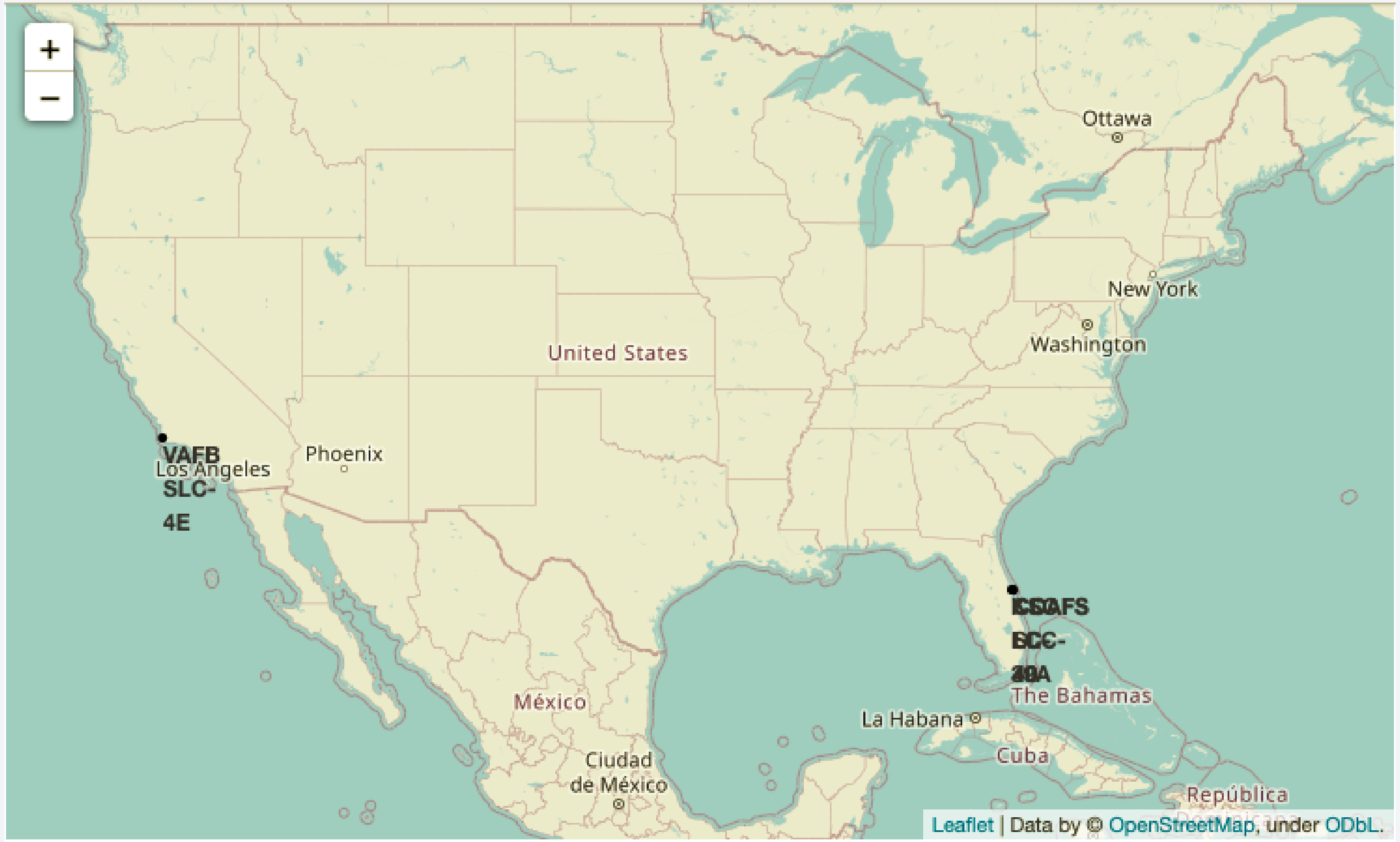
RESULT:

('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

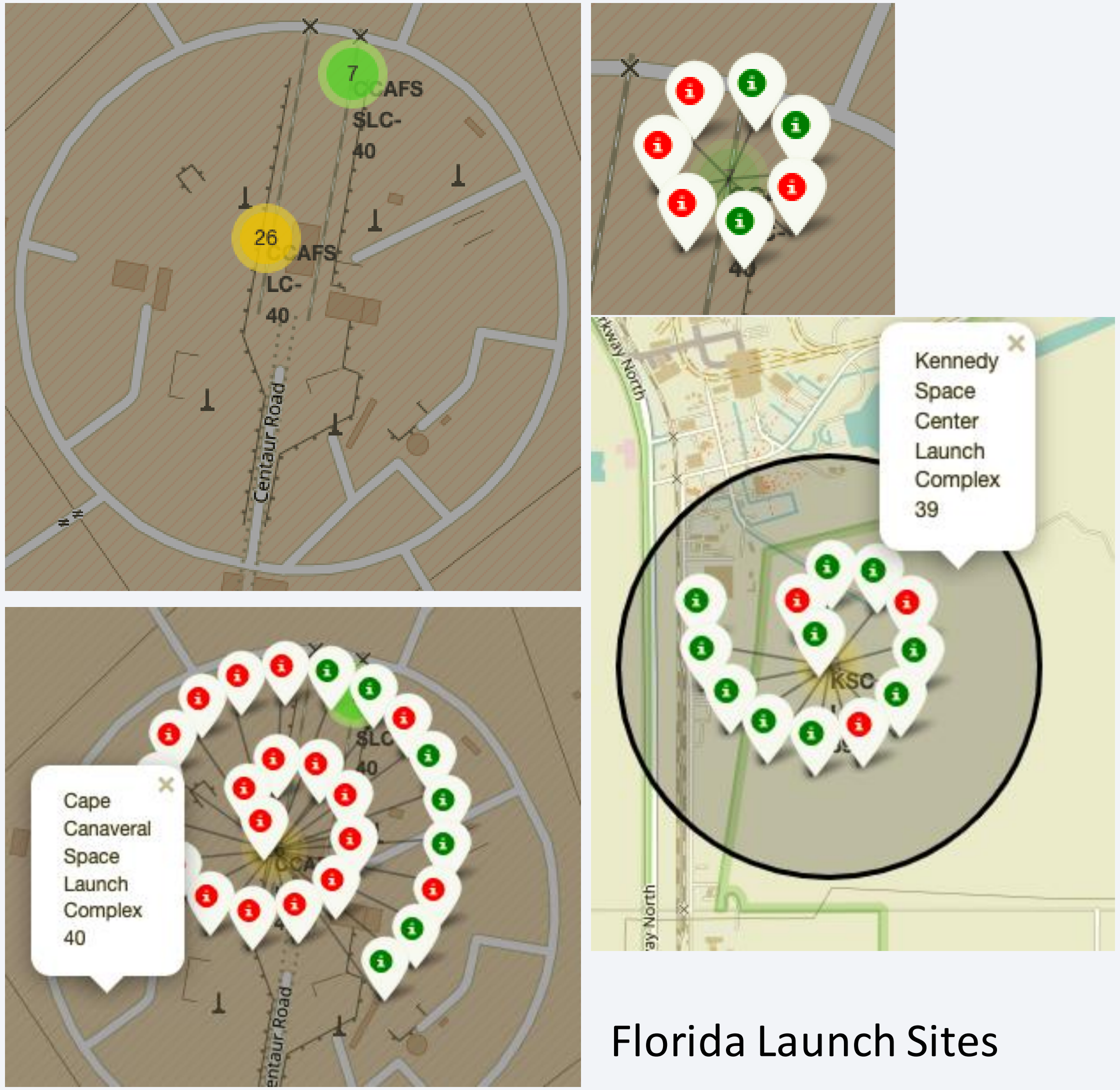
Launch Sites Proximities Analysis

All launch sites on a map

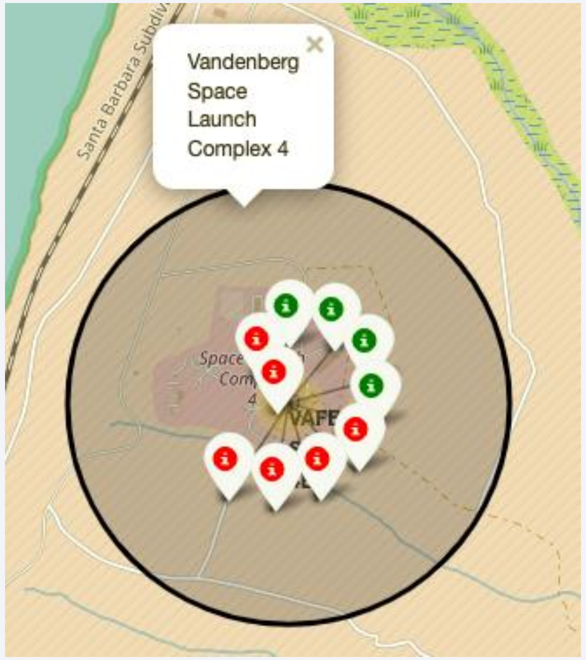


We see that the **SpaceX launch sites** are **located** in the US coasts - **Florida** and **California**;

Colour Labelled Markers



Florida Launch Sites



California Launch Site

- Green Marker - successful launch;
- Red Marker - unsuccessful launch;

Launch Site distances to landmarks (VAFB SLC-4E as a reference)

Q1: Are **launch sites** in **close** proximity to **railways**?
A1: Yes, they are (*max distance is less than 2 KM*);

Q2: Are **launch sites** in **close** proximity to **highways**?
A2: Yes, they are (*max distance is less than 1.5 KM*);

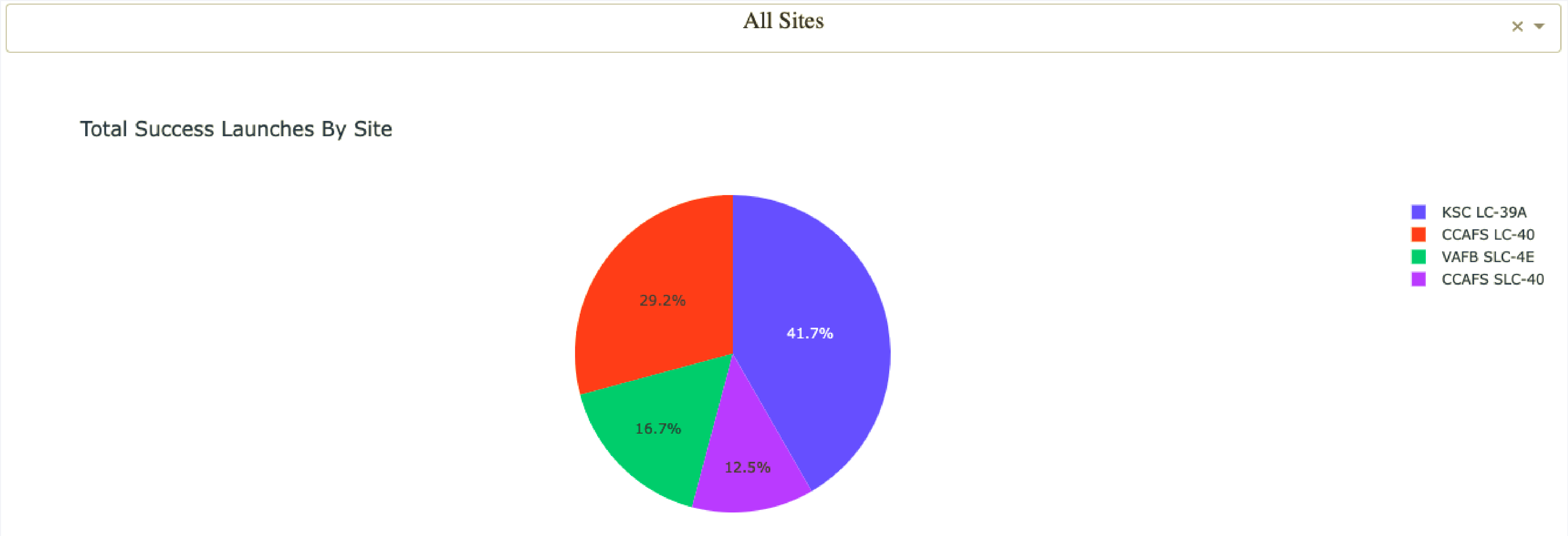
Q3: Are **launch sites** in **close** proximity to **coastline**?
Yes, they are (*max distance is less than 1.5 KM*);

Q4: Do **launch sites keep** certain **distance away from cities**?
A4: If the "*certain distance*" is some distance *higher than 50 KM*, then yes, they do;

Section 4

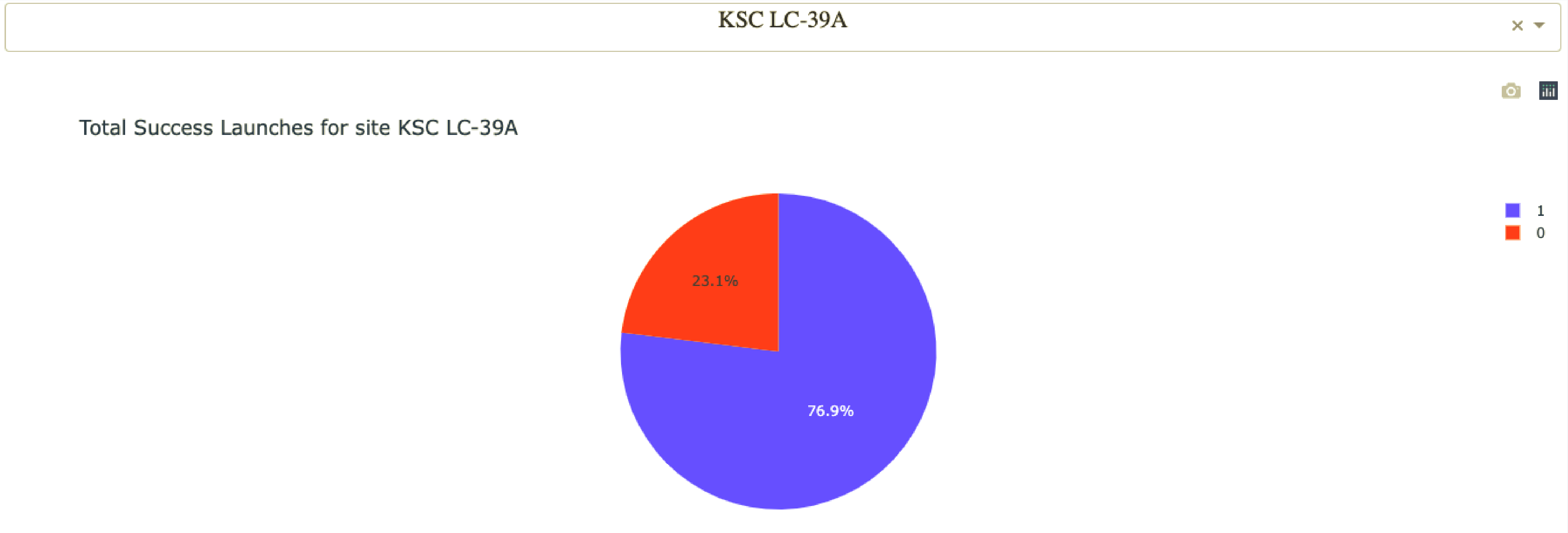
Build a Dashboard with Plotly Dash

SpaceX Launch Records Dashboard



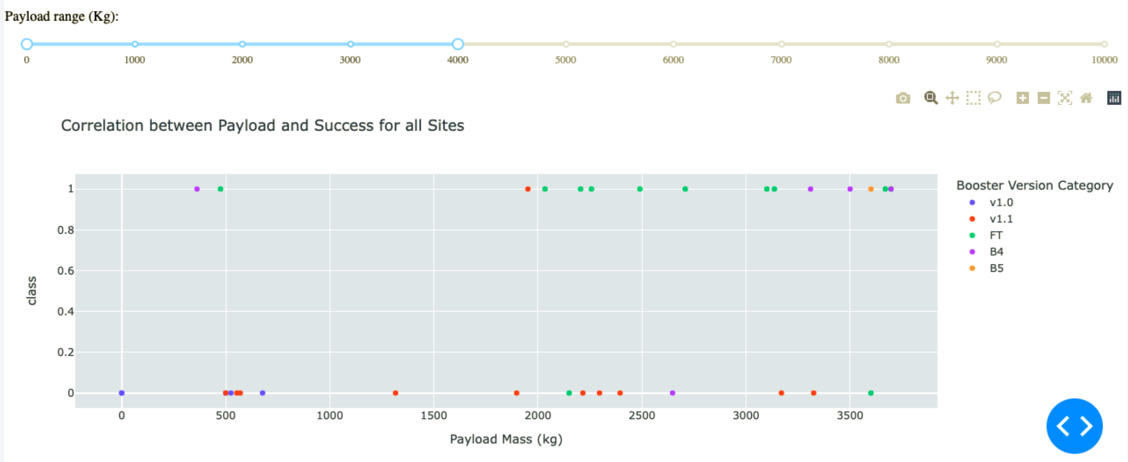
KSC LC-39A has the highest number of successful launches from all the sites;

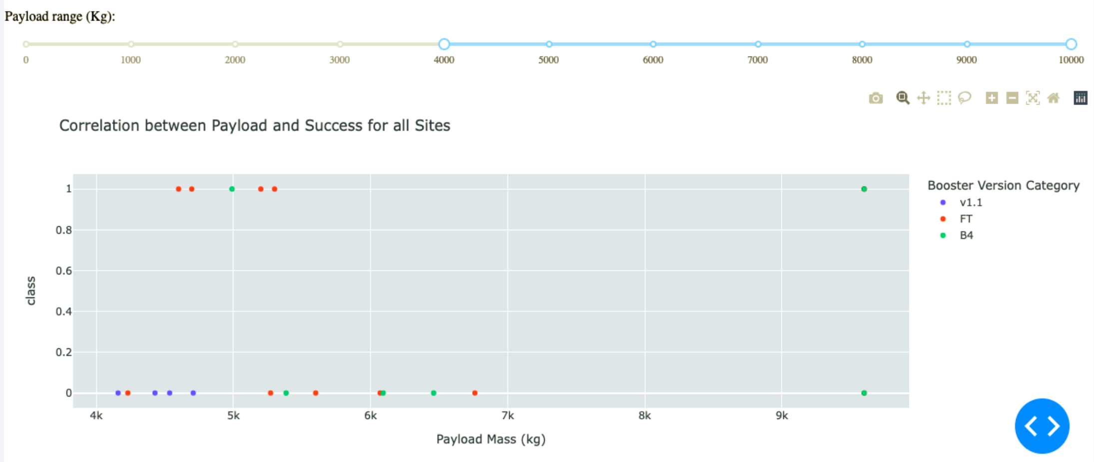
SpaceX Launch Records Dashboard



KSC LC-39A achieves a 76.9% success rate;

SpaceX Launch Records Dashboard





The success rates for cases with Payload Mass 0 kg – 4000 kg is higher than for cases with Payload Mass 4000 kg – 10000kg (for all sites);

Section 5

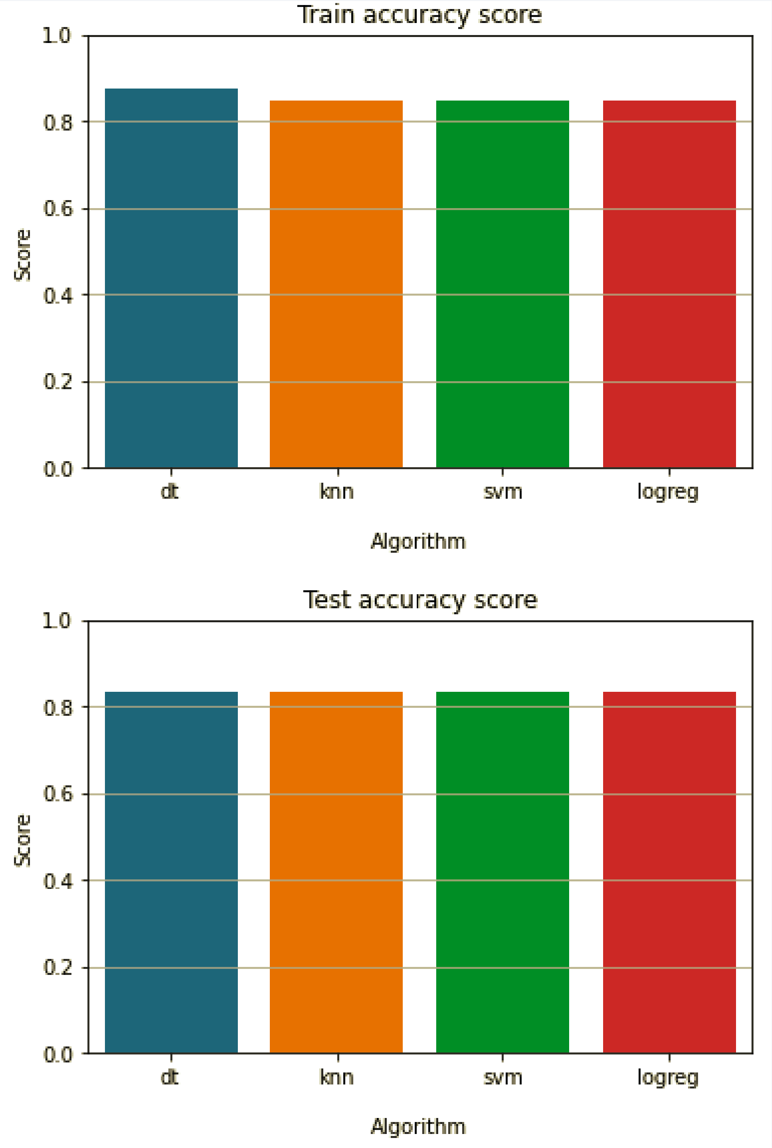
Predictive Analysis (Classification)

Classification Accuracy

- In this case the Test accuracy is the same for all the algorithms – 0.83. Nevertheless I've decided to choose a Decision Tree algorithm, because its Train accuracy is a little bit higher – 0.875;

- The DT classifier best parameters (according to the GridSearchCV results):

{
'criterion': 'gini',
'max_depth': 4,
'max_features': 'sqrt',
'min_samples_leaf': 4,
'min_samples_split': 2,
'splitter': 'random'
}



Confusion Matrix

According to the Confusion Matrix, the DT algorithm can distinguish "land" pretty good, while there is a problem with "did not land" cases – Type I error. We can try to solve the problem by changing a cut off value (default is 0.5), applying a bagging technique, or by adding more data;

Conclusions

- The general trend of the success rate kept increasing since 2013;
- Orbit types ES-L1, GEO, HEO, SSO have the highest success % - 100%;
- All the launch sites keep certain distance from cities and are in close proximity to railways, highways, coastline;
- Launch site KSC LC-39A has the highest number of successful launches from all the sites;
- The success rates of Low weighted payloads (0-4000 kg) is higher than for heavier payloads (4000-10000 kg) among all sites;
- Decision Tree Classifier may be the best algorithm in our case, but there is still some space for making an algorithm's performance better;

Appendix

- SQLite

SQLite

- SQLite is an embedded SQL database engine. Unlike most other SQL databases, SQLite does not have a separate server process. SQLite reads and writes directly to ordinary disk files. A complete SQL database with multiple tables, indices, triggers, and views, is contained in a single disk file.

```
# import libraries
import sqlite3
from sqlite3 import Error
```

sqlite.db <- saved database

```
# create connection
def sql_connection():
   try:
      connection =
sqlite3.connect('sqlite.db')
      return connection
   except Error as er:
      print(er)
```

→

```
# fetch data
def sql_fetch(connection,
select):
   # create a cursor
   cursor =
connection.cursor()
   cursor.execute(select)
   for row in cursor.fetchall():
      print(row)
```

→

```
# run query
select_col = """
select distinct(Launch_Site)
from Spacex;
"""

sql_fetch(sql_connection(),
select_col)
```

→

```
# output
('CCAFS LC-40',)
('VAFB SLC-4E',)
('KSC LC-39A',)
('CCAFS SLC-40',)
```

Thank you!