IBM Developer
SKILLS NETWORK

Winning Space Race with Data Science

Dr Sam Thompson
October 2023

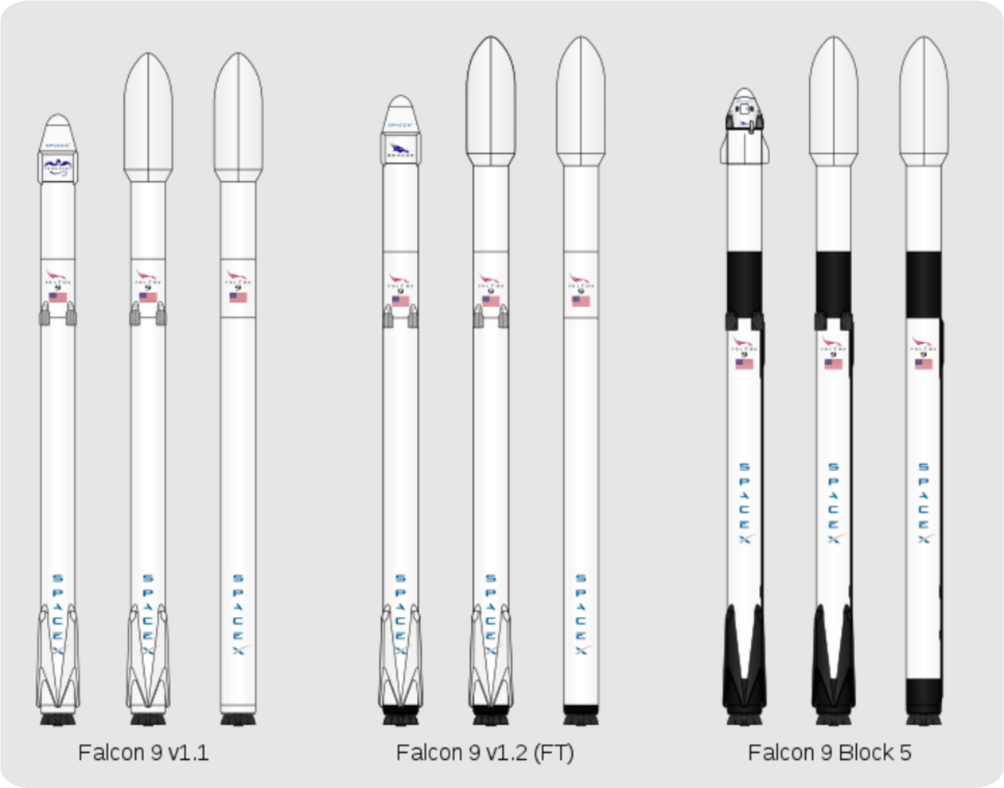
Outline

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

Executive Summary

- Data collected from SpaceX API and web-scraped from Wikipedia
- Data wrangled and encoded for ML
- EDA performed with visualization and SQL
  - Increase in success from 2017 to 2020, best success in ES-L1, GEO, HEO, and SSO orbit types
- Folium
  - Locations of most successful sites and proximities to landmarks
- Dashboard
  - Success rates of each launch site, most successful being CCAFS SLC-40
- ML
  - SVM and KNN produce best predictive result of all algorithms, ~84% accuracte

Introduction



- Working for Space Y, the leading competitor to Space X, we need to understand how our rivals successes can benefit us and lead to greater market share in the reusable rocket race
- Space X's Falcon 9 rocket, their flagship vessel, has a reusable first stage. The first stage serves as the main thruster out of the atmosphere, and is extremely costly to build from scratch for each launch
- In this report we'll look at the cost of each launch, and the probability of being able to reuse the first stage of our rocket in a later trial. To do this, we'll use existing open-access data from Space X.

Section 1

Methodology

Methodology

Data collection

The data was collected directly from Space X using the Space X REST API, api.spacexdata.com/v4/. A comparison was also made using webscraping with BeautifulSoup

Perform data wrangling

Data was cleaned and processed using Pandas and Numpy

Perform exploratory data analysis (EDA) using visualization and SQL

Perform interactive visual analytics using Folium and Plotly Dash

Folium

Perform predictive analysis

Classification models were used to build our predictive capability.

*Data url: https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/datasets/API_call_spacex_api.json

Data Collection – SpaceX API

- Data pulled from SpaceX REST API

```
[2]: # Takes the dataset and uses the rocket column to call the API and append the data to the list
def getBoosterVersion(data):
    for x in data['rocket']:
       if x:
        response = requests.get("https://api.spacexdata.com/v4/rockets/"+str(x)).json()
        BoosterVersion.append(response['name'])
```

```
[3]: # Takes the dataset and uses the launchpad column to call the API and append the data to the list
def getLaunchSite(data):
    for x in data['launchpad']:
       if x:
         response = requests.get("https://api.spacexdata.com/v4/launchpads/"+str(x)).json()
         Longitude.append(response['longitude'])
         Latitude.append(response['latitude'])
         LaunchSite.append(response['name'])
```

```
[4]: # Takes the dataset and uses the payloads column to call the API and append the data to the lists
def getPayloadData(data):
    for load in data['payloads']:
       if load:
        response = requests.get("https://api.spacexdata.com/v4/payloads/"+load).json()
        PayloadMass.append(response['mass_kg'])
        Orbit.append(response['orbit'])
```

```
[5]: # Takes the dataset and uses the cores column to call the API and append the data to the lists
def getCoreData(data):
    for core in data['cores']:
            if core['core'] != None:
                response = requests.get("https://api.spacexdata.com/v4/cores/"+core['core']).json()
                Block.append(response['block'])
                ReusedCount.append(response['reuse_count'])
                Serial.append(response['serial'])
            else:
                Block.append(None)
                ReusedCount.append(None)
                Serial.append(None)
            Outcome.append(str(core['landing_success'])+' '+str(core['landing_type']))
            Flights.append(core['flight'])
            GridFins.append(core['gridfins'])
            Reused.append(core['reused'])
            Legs.append(core['legs'])
            LandingPad.append(core['landpad'])
```

```
#Global variables
BoosterVersion = []
PayloadMass = []
Orbit = []
LaunchSite = []
Outcome = []
Flights = []
GridFins = []
Reused = []
Legs = []
LandingPad = []
Block = []
ReusedCount = []
Serial = []
Longitude = []
Latitude = []
```

[35]:

	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins	Reused	Legs	LandingPad	Block	ReusedCount	Serial	Longitude	Latitude
4	1	2010-06-04	Falcon 9	6123.547647	LEO	CCSFS SLC 40	None None	1	False	False	False	None	1.0	0	B0003	-80.577366	28.561857
5	2	2012-05-22	Falcon 9	525.000000	LEO	CCSFS SLC 40	None None	1	False	False	False	None	1.0	0	B0005	-80.577366	28.561857
6	3	2013-03-01	Falcon 9	677.000000	ISS	CCSFS SLC 40	None None	1	False	False	False	None	1.0	0	B0007	-80.577366	28.561857
7	4	2013-09-29	Falcon 9	500.000000	PO	VAFB SLC 4E	False Ocean	1	False	False	False	None	1.0	0	B1003	-120.610829	34.632093
8	5	2013-12-03	Falcon 9	3170.000000	GTO	CCSFS SLC 40	None None	1	False	False	False	None	1.0	0	B1004	-80.577366	28.561857

1) requests.get(url*)
2) pd.json_normalize

Retrieve the JSON file via the API and convert to pandas dataframe

3) Create subset of dataframe containing only 15 relevant columns of

4) Use specific requests.get() functions to fill columns

5) Isolate only rows for Falcon 9 boosters

6) Convert nulls in Payload to mean for column

GITHUB: IBMDataScienceCapstone/jupyter-labs-spacex-data-collection-api.ipynb at main · SpongeSJT/IBMDataScienceCapstone (github.com)

*Data url: https://en.wikipedia.org/w/index.php?title=List_of_Falcon_9_and_Falcon_Heavy_launches&oldid=1027686922

Data Collection - Scraping

- Comparison made with webscraping from Wikipedia table. BeautifulSoup and Pandas used to make data useable

1) requests.get(url*)

2) BeautifulSoup(url.content, htmlparser)

Retrieve the data file via the API and convert to BeautifulSoup html object

3) Extract all table objects from html data BeautifulSoup.bs.findall(table)

4) For loop to extract all column names ,<th>, from html data and create dictionaries with column names

5) Loop through html data to extract values for columns and add to the dictionaries

Convert filled dictionary to a pd.DataFrame object and save as a CSV

```
def date_time(table_cells):
    """
    This function returns the data and time from the HTML  table cell
    Input: the  element of a table data cell extracts extra row
    """
    return [data_time.strip() for data_time in list(table_cells.strings)][0:2]

def booster_version(table_cells):
    """
    This function returns the booster version from the HTML  table cell 
    Input: the  element of a table data cell extracts extra row
    """
    out=''.join([booster_version for i,booster_version in enumerate( table_cells.strings) if i%2==0][0:-1])
    return out

def landing_status(table_cells):
    """
    This function returns the landing status from the HTML table cell 
    Input: the  element of a table data cell extracts extra row
    """
    out=[i for i in table_cells.strings][0]
    return out


def get_mass(table_cells):
    mass=unicodedata.normalize("NFKD", table_cells.text).strip()
    if mass:
        mass.find("kg")
        new_mass=mass[0:mass.find("kg")+2]
    else:
        new_mass=0
    return new_mass


def extract_column_from_header(row):
    """
    This function returns the landing status from the HTML table cell 
    Input: the  element of a table data cell extracts extra row
    """
    if (row.br):
        row.br.extract()
    if row.a:
        row.a.extract()
    if row.sup:
        row.sup.extract()

    colunm_name = ' '.join(row.contents)

    # Filter the digit and empty names
    if not(colunm_name.strip().isdigit()):
        colunm_name = colunm_name.strip()
        return colunm_name
```

```
[13]: extracted_row = 0
#Extract each table
for table_number,table in enumerate(soup.find_all('table',"wikitable plainrowheaders collapsible")):
   # get table row
    for rows in table.find_all("tr"):
        #check to see if first table heading is as number corresponding to launch a number
        if rows.th:
            if rows.th.string:
                flight_number=rows.th.string.strip()
                flag=flight_number.isdigit()
        else:
            flag=False
        #get table element
        row=rows.find_all('td')
        #if it is number save cells in a dictonary
        if flag:
            extracted_row += 1
            # Flight Number value
            # TODO: Append the flight_number into launch_dict with key 'Flight No.'
            #print(flight_number)
            datatimelist=date_time(row[0])
            launch_dict['Flight No.'].append(flight_number)

            # Date value
            # TODO: Append the date into launch_dict with key 'Date'
            date = datatimelist[0].strip(',')
            launch_dict['Date'].append(date)
            #print(date)

            # Time value
            # TODO: Append the time into launch_dict with key 'Time'
            time = datatimelist[1]
            launch_dict['Time'].append(time)
            #print(time)

            # Booster version
            # TODO: Append the bv into launch_dict with key 'Version Booster'
            bv=booster_version(row[1])
            if not(bv):
                bv=row[1].a.string
            launch_dict['Version Booster'].append(bv)
            #print(bv)

            # Launch Site
            # TODO: Append the bv into launch_dict with key 'Launch Site'
            launch_site = row[2].a.string
            launch_dict['Launch site'].append(launch_site)
            #print(launch_site)

            # Payload
            # TODO: Append the payload into launch_dict with key 'Payload'
            payload = row[3].a.string
            launch_dict['Payload'].append(payload)
            #print(payload)

            # Payload Mass
            # TODO: Append the payload_mass into launch_dict with key 'Payload mass'
            payload_mass = get_mass(row[4])
            launch_dict['Payload mass'].append(payload_mass)
            #print(payload)

            # Orbit
            # TODO: Append the orbit into launch_dict with key 'Orbit'
            orbit = row[5].a.string
            launch_dict['Orbit'].append(orbit)
            #print(orbit)

            # Customer
            # TODO: Append the customer into launch_dict with key 'Customer'
```


GITHUB: IBMDataScienceCapstone/jupyter-labs-webscraping.ipynb at main · SpongeSJT/IBMDataScienceCapstone (github.com)

Data Wrangling

- The data were explored, using value counts and date type investigations on LaunchSite, Orbit, and Outcome columns
- A dictionary was created for the Outcome column, to isolate each type of outcome
- The column converted the types of landing to either 1 for success (True), or 0 for failure (False)
- Feature engineering was also performed on the columns "Orbit", "LaunchSite", "LandingPad", "Serial", with "one hot encoding" used to prepare the data for Machine Learning

[23]:
```
for i in df['Outcome']:
    if i in set(bad_outcomes):
        landing_class.append(0)
    else:
        landing_class.append(1)
```

[24]:
```
df['Class']=landing_class
df[['Class']].head(8)
```

[24]:

	Class
0	0
1	0
2	0
3	0
4	0
5	0
6	1
7	1

[10]:
```
# landing_outcomes = values on Outcome column
landing_outcomes = df['Outcome'].value_counts()
landing_outcomes
```

[10]:
```
True ASDS      41
None None      19
True RTLS      14
False ASDS      6
True Ocean      5
False Ocean     2
None ASDS       2
False RTLS      1
Name: Outcome, dtype: int64
```

[11]:
```
for i,outcome in enumerate(landing_outcomes.keys()):
    print(i,outcome)
```

```
0 True ASDS
1 None None
2 True RTLS
3 False ASDS
4 True Ocean
5 False Ocean
6 None ASDS
7 False RTLS
```

We create a set of outcomes where the second stage did not land successfully:

[12]:
```
bad_outcomes=set(landing_outcomes.keys()[[1,3,5,6,7]])
```

GITHUB: [IBMDataScienceCapstone/labs-jupyter-spacex-data_wrangling_jupyterlite.jupyterlite.ipynb at main · SpongeSJT/IBMDataScienceCapstone (github.com)]

EDA with Data Visualization

- Charts plotted
  - Flight number vs Launch Site, Scatter plot. Allowed us to see trend of how many launches were performed per site and when they were done in the sequence of launches
  - Payload Mass vs Launch Site, Scatter plot. Allowed visualization of trend between payload mass and launch site
  - Orbit Type vs Success Rate, Bar plot. Allowed visualization of success rate of launches vs their desired orbit
  - Flight Number vs Orbit type, Scatter plot. Allowed visualization of changes in orbit type during the sequence of launches
  - Payload vs Orbit type, Scatter plot. Allowed visualization of different payload masses and their desired orbit
  - Yearly Success rate change, Line plot. Showing how success of launches changed over time

GITHUB: [IBMDataScienceCapstone/vizeda.ipynb at main · SpongeSJT/IBMDataScienceCapstone (github.com)]

EDA with SQL

- SQL Queries:
  - Display unique launch site names
  - Display sites containing string ‘CCA’
  - Display sum of payload masses launched in the table
  - Display average payload mass in the table
  - List the date when the first successful landing outcome in ground pad was achieved
  - List the names of the boosters which have success in drone ship and have payload mass greater than 4000 but less than 6000
  - List the total number of successful and failure mission outcomes
  - List the names of the booster_versions which have carried the maximum payload mass
  - List the records which will display the month names, failure landing_outcomes in drone ship ,booster versions, launch_site for the months 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: [IBMDataScienceCapstone/sqleda.ipynb at main · SpongeSJT/IBMDataScienceCapstone (github.com)]()

Build an Interactive Map with Folium

- Explain why you added those objects
- Add the GitHub URL of your completed interactive map with Folium map, as an external reference and peer-review purpose
- Created a map of the SpaceX launch sites
- Added map markers to show their locations
- Added circles to show the number of launches per site
- Added lines to show the nearest city, the nearest railroad, and the nearest highway

GITHUB: [IBMDataScienceCapstone/lab_jupyter_launch_site_location.jupyterlite.ipynb at main · SpongeSJT/IBMDataScienceCapstone (github.com)]

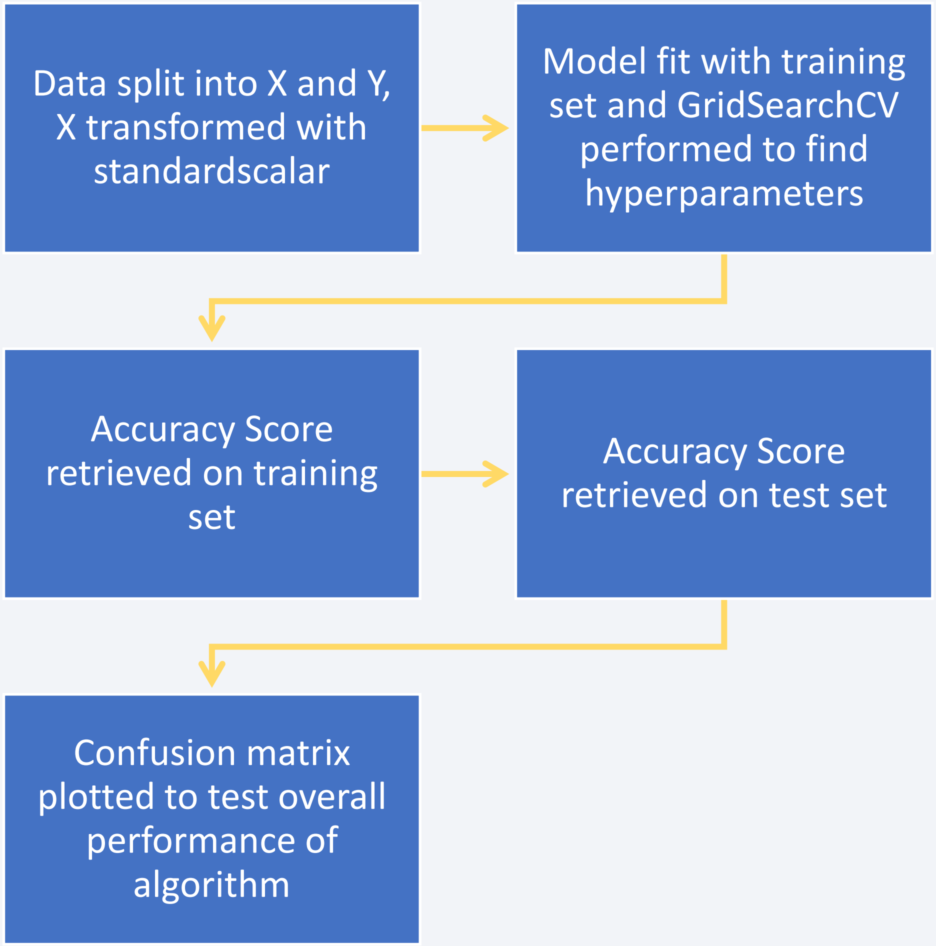
Build a Dashboard with Plotly Dash

- A Plotly Dashboard was created with pie charts and scatter plots
- The pie charts were plotted to show the number of successful landings across the different launch sites
- The pie chart could show all of the launch sites and their share of the total successes, or the success rate of a single site
- The scatter plot was made to show the relationship between payload mass, booster version, and whether or not the landing was successful
- The scatter plot was controllable by a payload mass slider to isolate regions of interest in the data space

GITHUB: [IBMDataScienceCapstone/dash.txt at main · SpongeSJT/IBMDataScienceCapstone (github.com)]

Predictive Analysis (Classification)

- Four machine learning algorithms were explored
  - K Nearest Neighbors
  - Logistic Regression
  - Support Vector Machine (SVM)
  - Decision Tree
- Data split into X and Y, and test train split. X was transformed with a standard scaler prior to test train splitting
- Model was trained on a training set of the data comprising of 80% of the original dataset
- GridSearchCV was used to tune the model hyperparameters
- Accuracy on training set was returned, then accuracy on the testing set. A confusion matrix was then plotted
- K Nearest Neighbors and SVM returned the highest accuracy



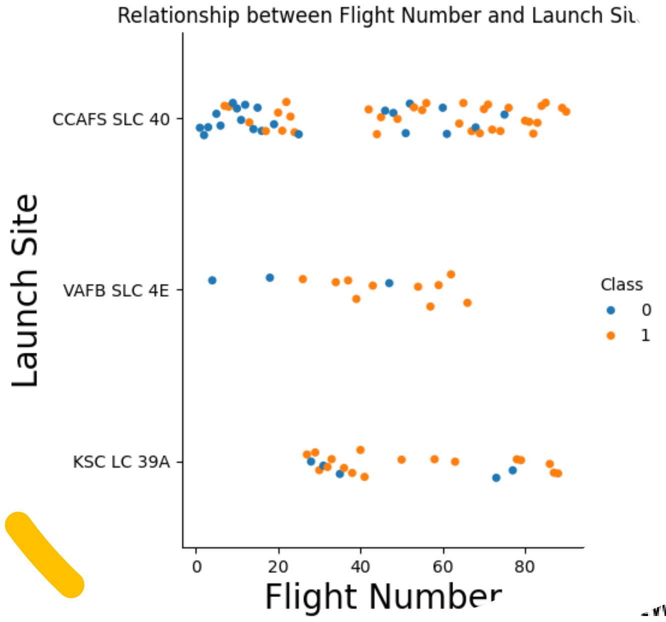
GITHUB: [IBMDataScienceCapstone/SpaceX_Machine_Learning_Prediction_Part_5.jupyterlite (1).ipynb at main · SpongeSJT/IBMDataScienceCapstone (github.com)]

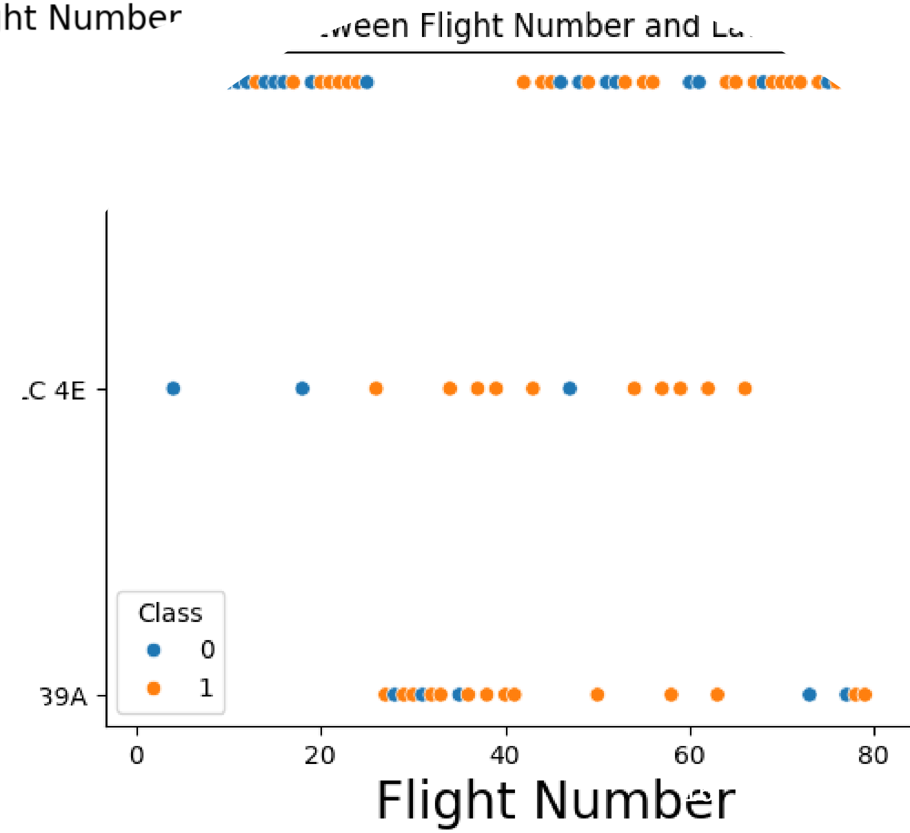
Section 2

Insights drawn from EDA

Flight Number vs. Launch Site

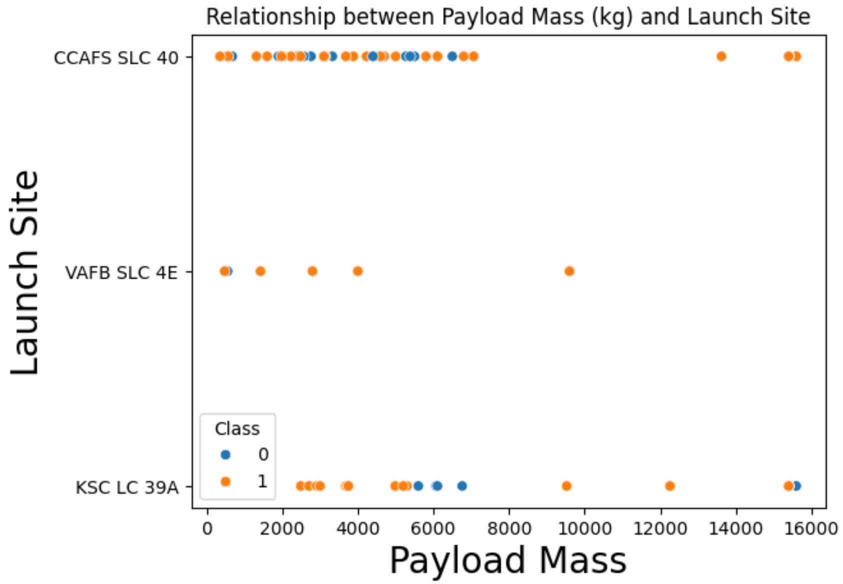
- These plots show us the number of launches per site over the span of the project
- We can see when in sequence sites were most active
- We can get a sense of success rate by class, 1= success, 0 = failure





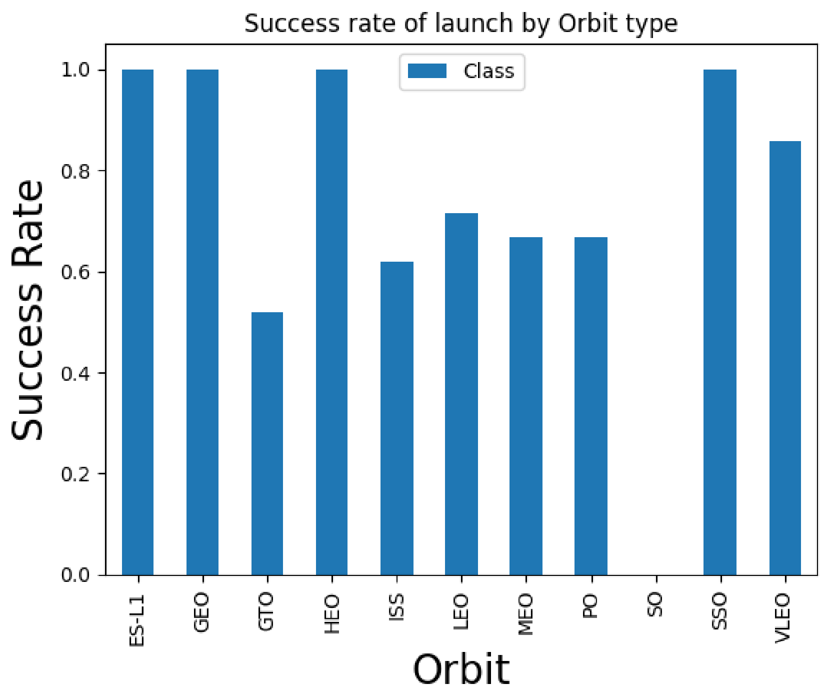
Payload vs. Launch Site

- This plot shows us the number of launches per site with a given payload mass
- We can see the distribution of weights launched and see any preference between site and payload mass. Highest masses are reserved for KSC and CCAFS sites.
- We can get a sense of success rate by class, 1= success, 0 = failure



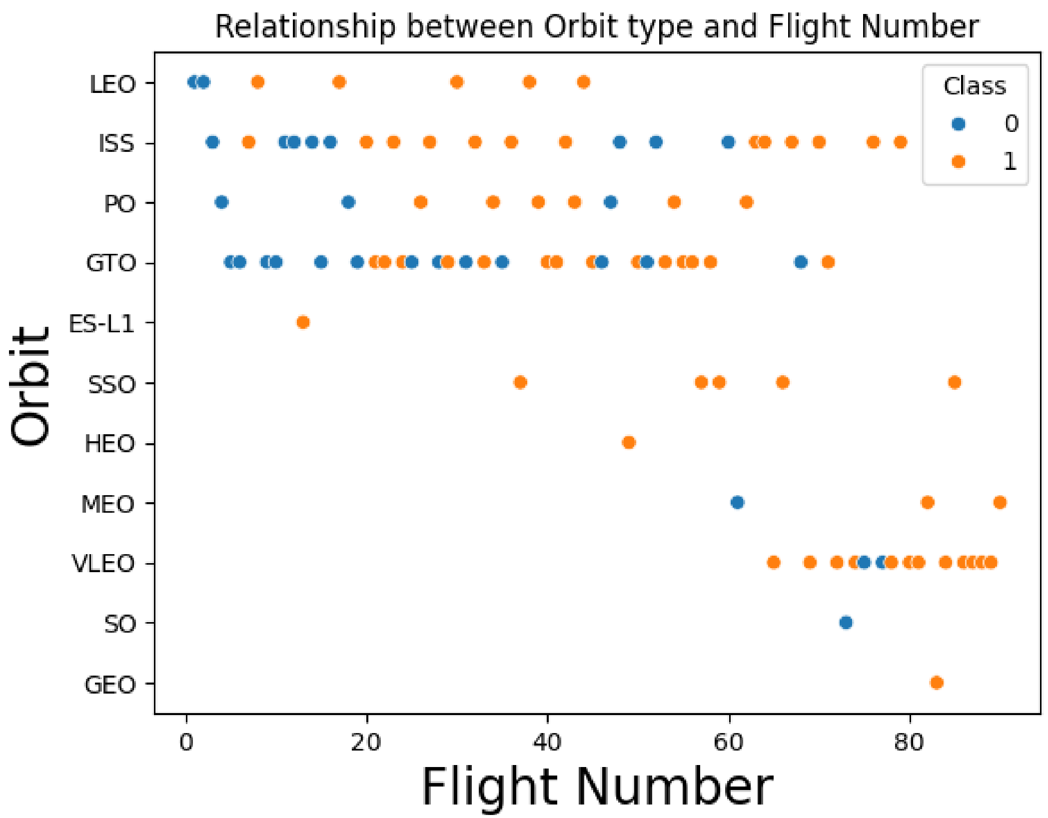
Success Rate vs. Orbit Type

- This plot shows us the success rate by Orbit type
- We can see that the most successful launch types are ES-L1, GEO, HEO, and SSO.
- Least successful being SO, GTO, and ISS
- No information about the distribution of sites and orbit type



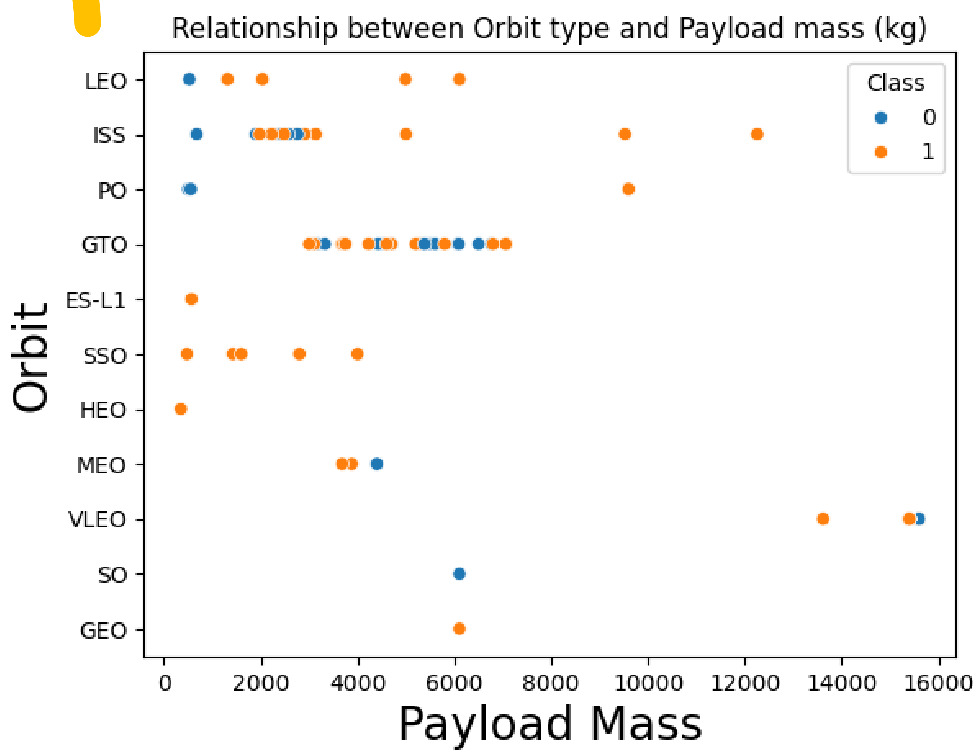
Flight Number vs. Orbit Type

- We can see the distribution of orbit types over the span of the data
- We see consistent launches in LEO, ISS, PO, and GTO
- GEO, SO, VLEO, MEO only occur later
- Success driven by more experience?



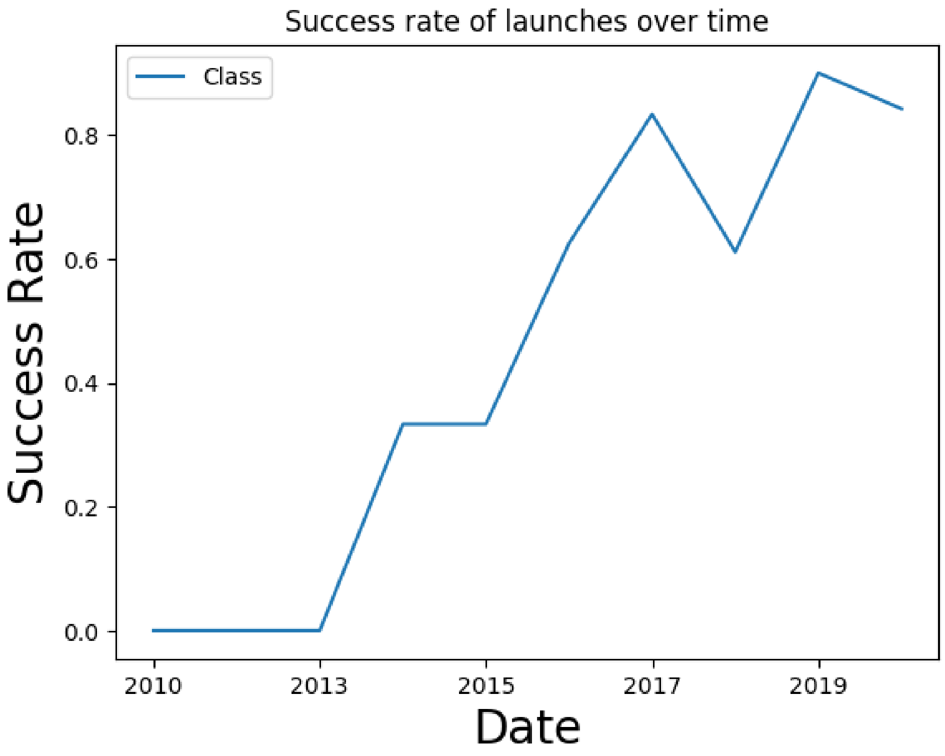
Payload vs. Orbit Type

- We see overall a preference of <7000kg Payload mass
- GTO has a narrow band of launches between 2000 and 8000 kg
- Smallest ranges are in MEO
- Largest range in ISS



Launch Success Yearly Trend

- We see a steady increase in success rate with a slight dip towards 2020
- This could be due to fewer launches in the latter years of the dataset



All Launch Site Names

- Find the names of the unique launch sites
- We see the list of 4 launch sites

```
[10]: %sql SELECT DISTINCT "Launch_Site" FROM SPACEXTBL

 * sqlite:///my_data1.db

Done.

[10]:
```

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

Launch Site Names Begin with 'CCA'

- Find 5 records where launch sites begin with `CCA`
- We see that there are 60 launches from sites beginning with CCA

```
[11]: %sql SELECT count(*) from SPACEXTBL Where "Launch_Site" LIKE '%CCA%'
       * sqlite:///my_data1.db
      Done.
[11]:
```

count(*)
60

Total Payload Mass

- Calculate the total payload carried by boosters from NASA
- The total sum of the payloads is displayed here as 619967 kg

Task 3

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

```
[12]: %sql SELECT SUM("PAYLOAD_MASS__KG_") FROM SPACEXTBL
 * sqlite:///my_data1.db
Done.
[12]:
```

SUM("PAYLOAD_MASS__KG_")
619967

Average Payload Mass by F9 v1.1

- Calculate the average payload mass carried by booster version F9 v1.1
- We see the average mass carried is 2928.4 kg

```
[13]: %sql SELECT AVG("PAYLOAD_MASS__KG_") FROM SPACEXTBL WHERE "Booster_Version" like 'F9 v1.1'

 * sqlite:///my_data1.db

Done.
```

[13]:

AVG("PAYLOAD_MASS__KG_")
2928.4

First Successful Ground Landing Date

- Find the dates of the first successful landing outcome on ground pad
- The earliest success in our data set was on the 22nd of December 2015

```
[14]: %sql SELECT MIN("Date") from SPACEXTBL WHERE "Landing_Outcome" like '%success%'

 * sqlite:///my_data1.db
Done.
[14]: 
```

MIN("Date")
2015-12-22

Successful Drone Ship Landing with Payload between 4000 and 6000

- List the names of boosters which have successfully landed on drone ship and had payload mass greater than 4000 but less than 6000
- We see that only F9 FT boosters have carried weights between 4000 and 6000 kg

```
[15]: %sql SELECT Booster_Version from SPACEXTBL WHERE "Landing_Outcome" like 'Success (drone ship)' and "PAYLOAD_MASS__KG_" between 4000 and 6000
 * sqlite:///my_data1.db
Done.
```

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

Total Number of Successful and Failure Mission Outcomes

- Calculate the total number of successful and failure mission outcomes
- We see here there have been 61 total successes

```
[16]: %sql SELECT count("Mission_Outcome") from SPACEXTBL WHERE "Landing_Outcome" like '%success%'

 * sqlite:///my_data1.db

Done.

[16]: ,,,,,,,,,,,
```

count("Mission_Outcome")
61

Boosters Carried Maximum Payload

- List the names of the booster which have carried the maximum payload mass
- We see only the F9 B5 boosters have carried the maximum payload

```
[21]: %sql SELECT DISTINCT("Booster_Version") From SPACEXTBL Where "PAYLOAD_MASS__KG_"=(SELECT MAX("PAYLOAD_MASS__KG_") from SPACEXTBL)
 * sqlite:///my_data1.db
Done.
```

[21]:

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

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

```
[20]: %sql SELECT "Date", "Landing_Outcome", "Booster_Version", "Launch_Site" from SPACEXTBL Where "Date" like '%2015%' and "Landing_Outcome" like '%failure%'
 * sqlite:///my_data1.db
Done.
[20]:
```

Date	Landing_Outcome	Booster_Version	Launch_Site
2015-10-01	Failure (drone ship)	F9 v1.1 B1012	CCAFS LC-40
2015-04-14	Failure (drone ship)	F9 v1.1 B1015	CCAFS LC-40

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

- Rank the count of landing outcomes (such as Failure (drone ship) or Success (ground pad)) between the date 2010-06-04 and 2017-03-20, in descending order
- The most prevalent result is no attempt, with a 3 way split on 2nd place for success (ground pad), success (drone ship), and failure (drone ship)

```
[23]: %sql SELECT "Landing_Outcome" ,COUNT(*) as COUNT_LAUNCHES FROM SPACEXTBL WHERE "Date" between '2010-06-04' and '2017-03-20' GROUP BY "Landing_Outcome" ORDER BY COUNT_LAUNCHES DESC
 * sqlite:///my_data1.db
Done.
[23]:
```

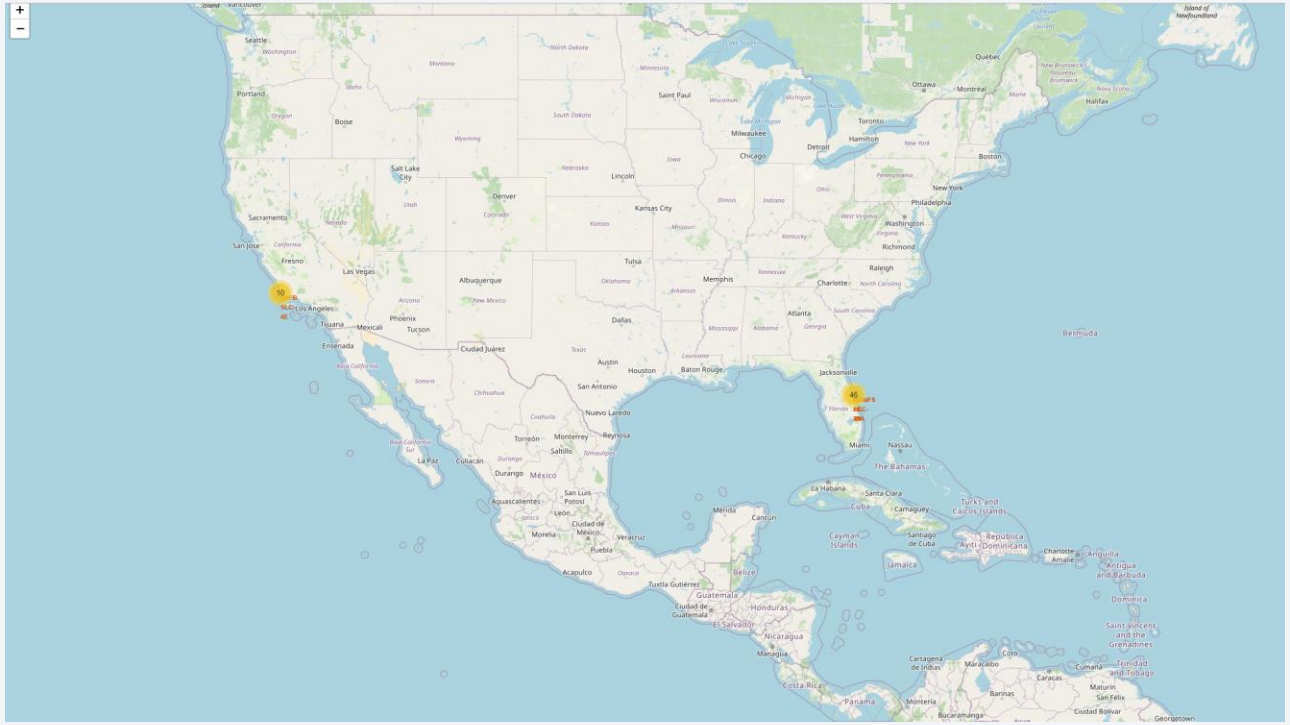
Landing_Outcome	COUNT_LAUNCHES
No attempt	10
Success (ground pad)	5
Success (drone ship)	5
Failure (drone ship)	5
Controlled (ocean)	3
Uncontrolled (ocean)	2
Precluded (drone ship)	1
Failure (parachute)	1

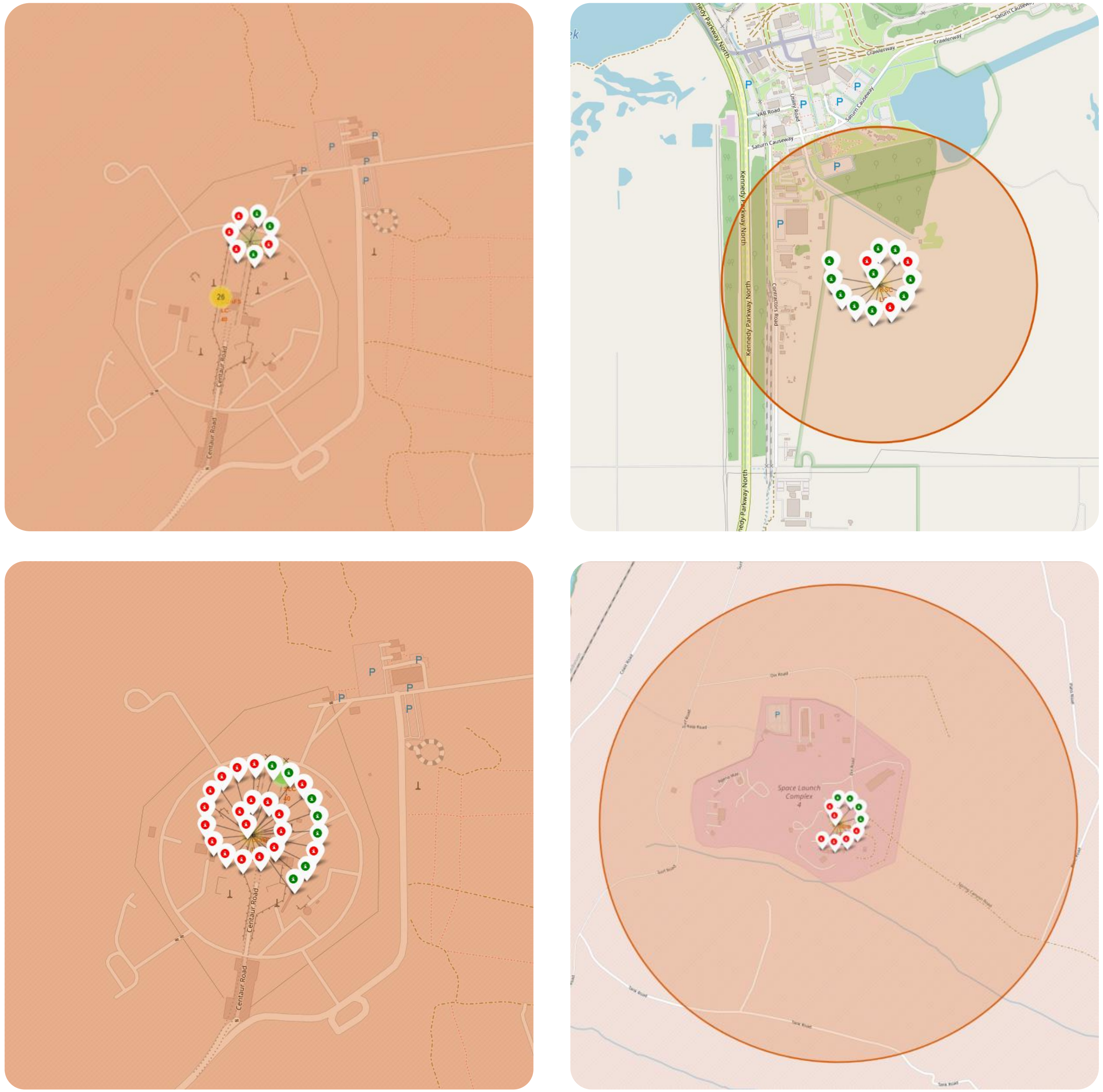
Section 3

Launch Sites Proximities Analysis

Launch site locations map

- We see the launch locations on the map
- They are based on the coast of Florida and California
- California was likely chosen for existing infrastructure and ease of hiring top engineering talent near the silicon valley area
- Florida likewise has a lot of existing infrastructure, and close ties for NASA. More launches were performed here than California



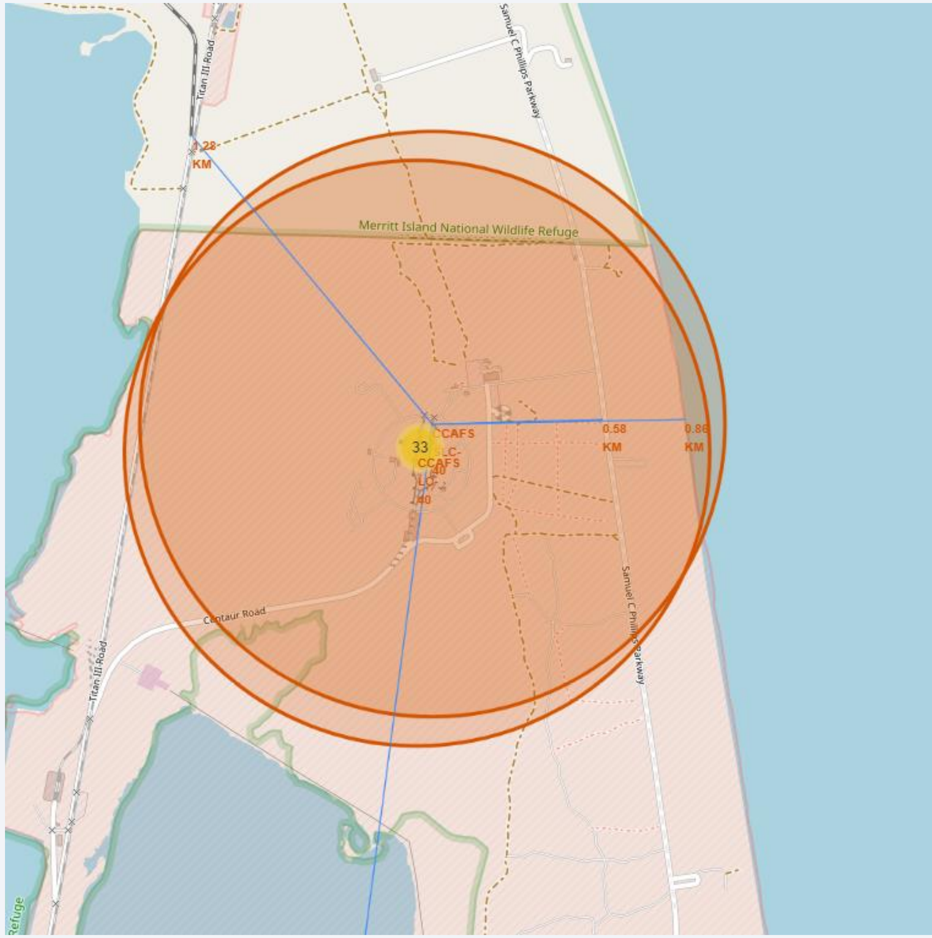


Success and number per site

- We see the number of launches per site as a spiral of markers per site
- Red is failure, green is success

Launch site proximity to landmarks

- Map showing proximity of nearest railway, highway, and coastline to the launch site

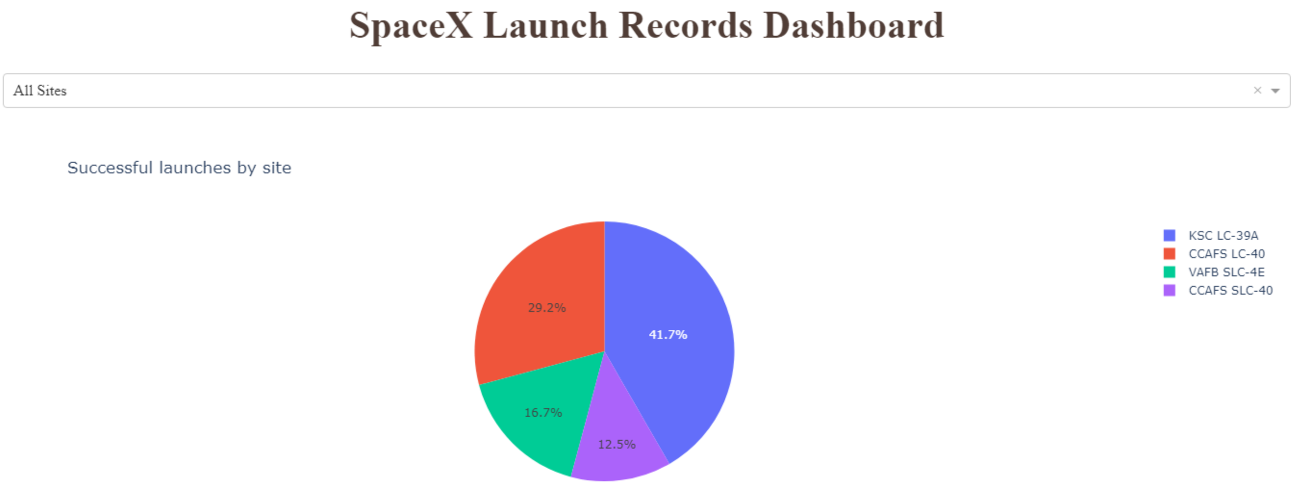


Section 4

Build a Dashboard with Plotly Dash

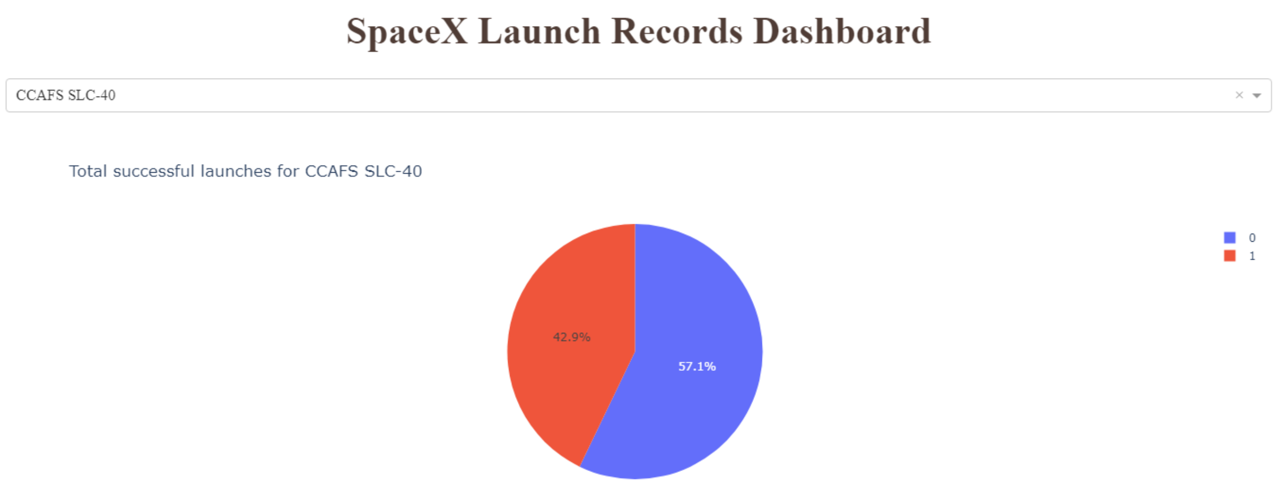
Total Successes by Launch site

- We see here the homepage of the dashboard
- The pie chart displays the percentage of successful launches broken down by launch site
- Greatest proportion of successes attributed to KSC LC-39A
- Lowest proportion attributed to CCAFS SLC-40



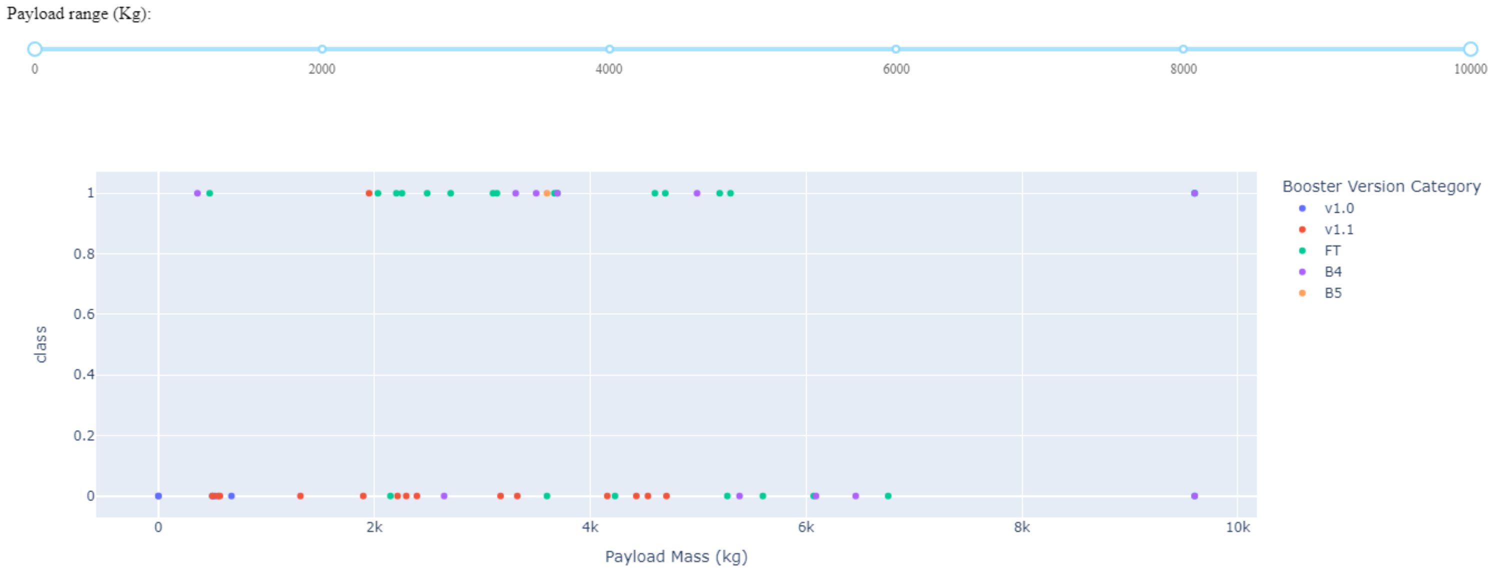
Most successful site

- Despite making up the smallest number of successes, CCAFS SLC-40 has the highest success rate
- Fewer launches at this site but they were on the whole more successful



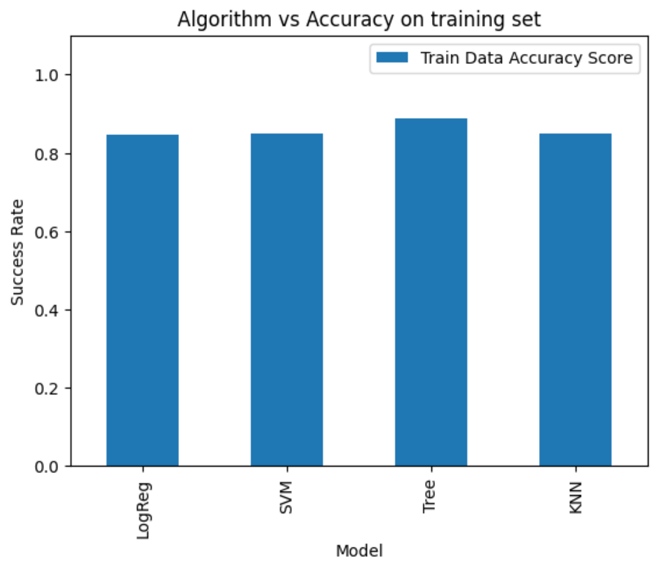
Payload Mass by Site, and success rate

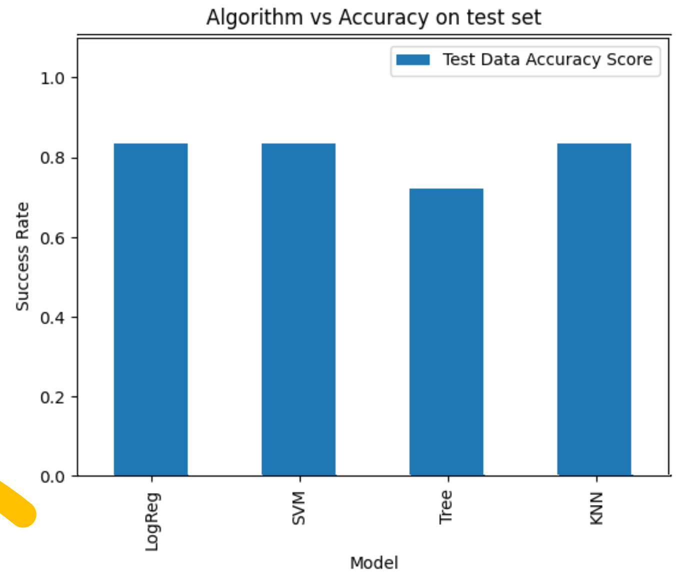
- Highest success rate is seen between masses of 2000 kg and 4000 kg
- The most successful boosters appear to be FT and B4



Section 5

Predictive Analysis (Classification)



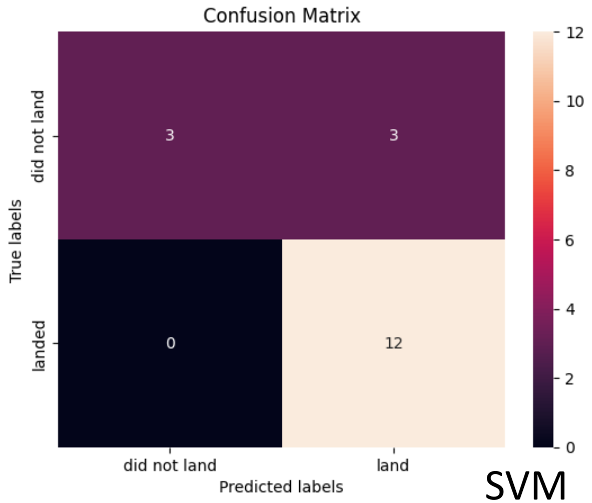


Classification Accuracy

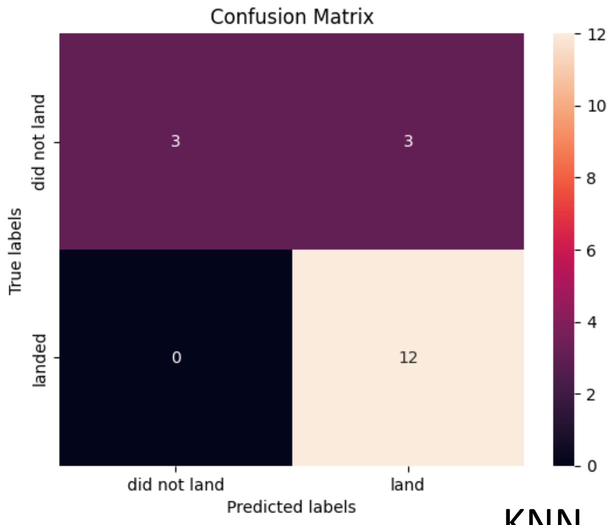
- We see that in the training data set the decision tree algorithm out-performs all others
- However, when applied to the test set it falls flat
- The overall best performance was produced by the KNN and SVM algorithms

Confusion Matrix

- We see the confusion matrices for the KNN and the SVM algorithms
- Both perform identically, with both producing the best accuracy scores on training and test sets
- We see no false negatives, but three false positives produced by both models



SVM



KNN

Conclusions

- SpaceX saw a steady increase in successes until 2020
- Payload masses are most successful between 2000 and 4000 kg. we should limit our launches to these masses until high success is achieved
- The most successful site is CCAFS SLC-40, we should replicate their set-up and operations
- The classification models broadly perform to a similar level, though SVM or KNN should be used for greatest accuracy

Thank you!