Winning Space Race with Data Science

<Name>
<Date>

The objective

Make sure you end up here …

… and not here

Outline

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

Executive Summary

- To analyze various aspects of booter recovery after a satellite launch, we collected the data made available by SpaceX and did various cross analysis to see what would be the best setup to use
- The conclusion is that
  - The models are good at predicting successes, but not as good at predicting failure (telling you what not to do, but not guaranteeing a success)
  - Using the FT and the B4 booster is probably the best choice
  - The launch site needs to be close to the shore, not that much for recovery success, but more for ease of bringing the booster back to the launch pad
  - Although the vast majority of the missions have brought the satellite on the correct orbit, it took several years to acquire the knowledge on how to recover the booster

Introduction

- You are looking to create a company offering space travel and need to find a path towards success

- We analyzed how your predecessor developed their expertise to success through analyzing data about their business development history:
  - Launch location
  - Technology development
  - Business development

Section 1

Methodology

Methodology

Executive Summary

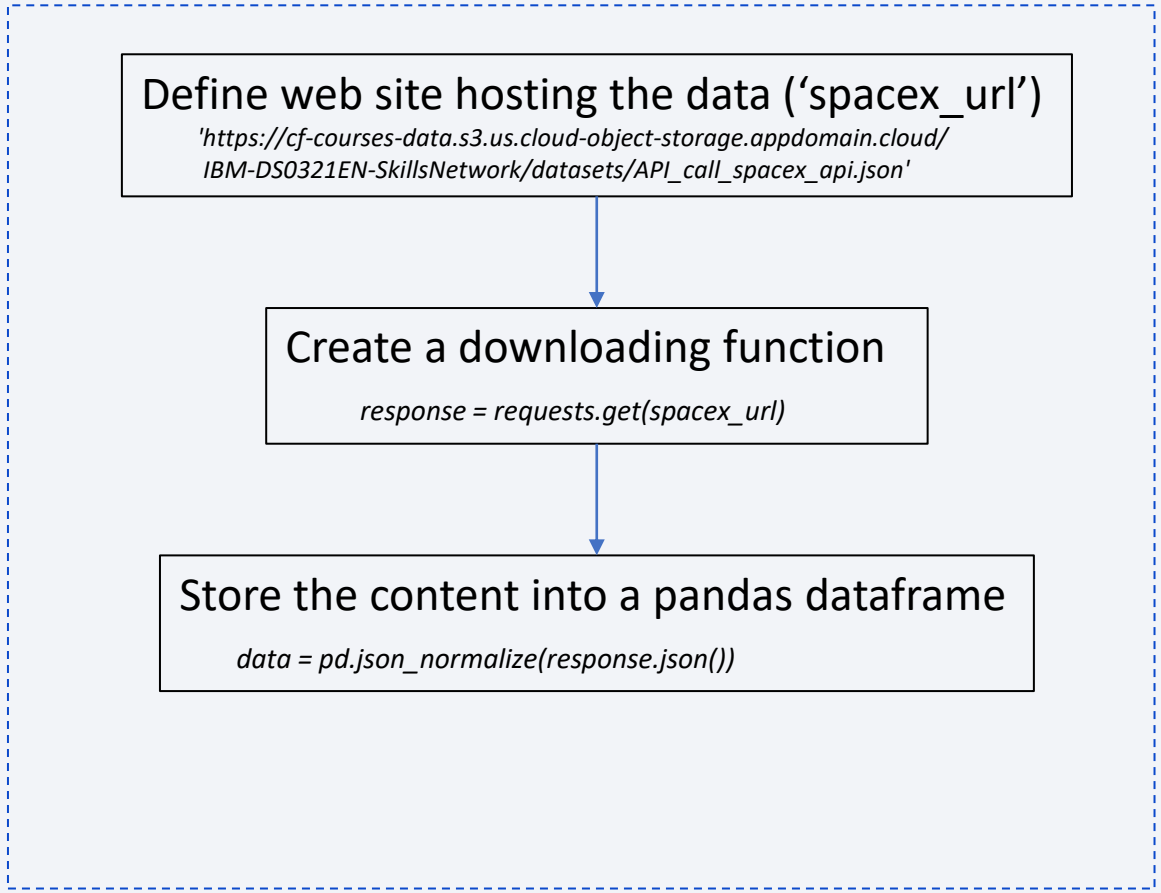
- Data collection methodology:
  - Describe how data was collected
- Perform data wrangling
  - Describe how data was processed
- Perform exploratory data analysis (EDA) using visualization and SQL
- Perform interactive visual analytics using Folium and Plotly Dash
- Perform predictive analysis using classification models
  - How to build, tune, evaluate classification models

Data Collection

Find Data Sources	Wikipedia, Alon Mask internet sites, etc.
Download Data	Using Python and BeautifulSoup technology
Data Treatment (missing values)	Convert qualitative and descriptive into numerical data Filling missing values with neutral values
Data harmonizaton	Make sure that all the data have the same quantum, so no one element outshines other aspects

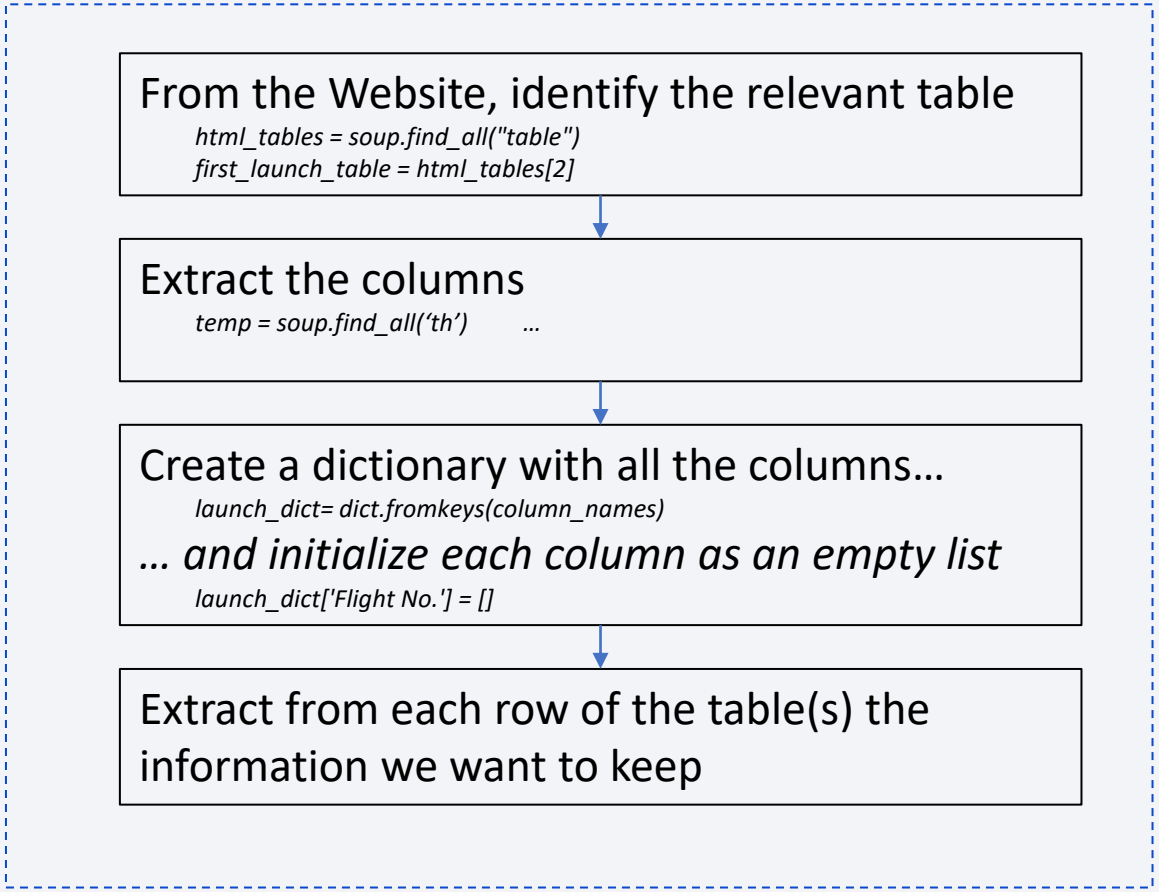
Data Collection – SpaceX API

- Github Location:
- https://github.com/albadhuyberechts/IBMDataScience/blob/main/IBM_Final/jupyter-labs-spacex-data-collection-api.ipynb



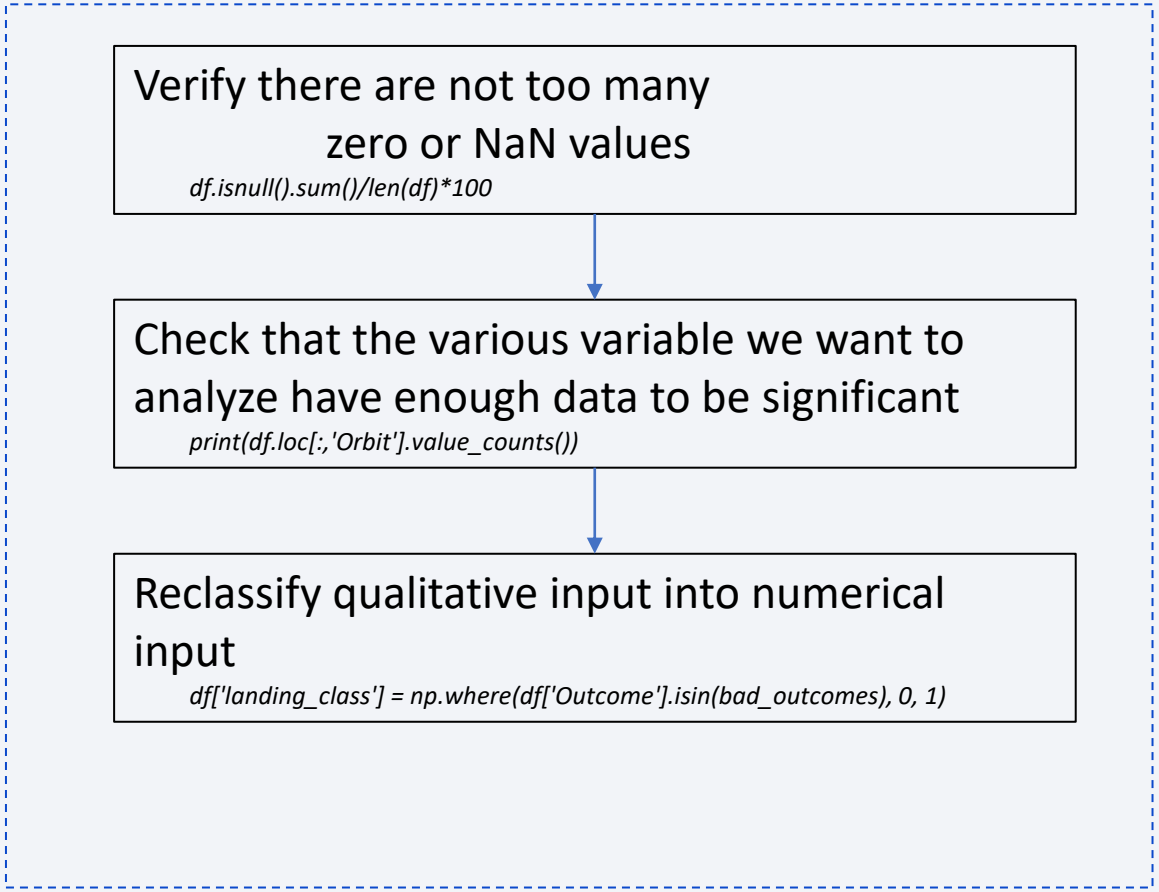
Data Collection - Scraping

- Github Location:
- https://github.com/albadhuyberechts/IBMDataScience/blob/main/IBM_Final/jupyter-labs-webscraping.ipynb



Data Wrangling

- Github Location:
- https://github.com/albadhuyberechts/IBMDataScience/blob/main/IBM_Final/labs-jupyter-spacex-Data%20wrangling.ipynb



EDA with Data Visualization

Graph used	Objective
Launch Site vs Flight number	Shows that SpaceX first used CC, then tried to use Houston, but came back to CC
Payload vs. Flight number	Shows that SpaceX initially started with relatively light pay load and increased the size overtime, while still servicing customers wanting to launch light payload in later flights.
Success rate vs. Orbit	The Bar chart does not seem to show a large correlation between these element. Perhaps is it more because the successful orbit when only used for later flights?
Orbit/Success vs. Flight number	Shows that the success rate improved with time for easy orbit, but remained a hit/miss for the GTO
Payload vs. Orbit type	It does not add much to the previous graph but rather shows what customers want
Success rate vs. time	Shows a clear learning period with 100% failure for two years then a gradual improvement until the technique was properly mastered

EDA with Data Visualization

- *Github File location*
- The below python file works properly
  - https://github.com/albadhuyberechts/IBMDataScience/blob/main/IBM_Final/EDA%20Visualisation.py
- While the Jupyter Notebook, even when reloaded from scratch from Coursera never showed the outcomes (5 attempts).
  - https://github.com/albadhuyberechts/IBMDataScience/blob/main/IBM_Final/jupyter-labs-eda-dataviz.ipynb.jupyterlite.ipynb
- Both files have the same code, but the Jupyter Notebook does not show the results.

EDA with SQL

- SELECT * FROM SPACEXTBL
  - *Discovery request to get a feeling of the table content*
- SELECT DISTINCT Launch_Site FROM SPACEXTBL
  - *Check how many launch size exist*
- SELECT TOP 5 Launch_Site FROM SPACEXTBL WHERE (LEFT(Launch_Site, 3) = "CCA")
  - *All site in Cap Carnaveral are within a few hudred meteres of oen another, we might want to regroup them under one site*
- SELECT SUM(PAYLOAD_MASS__KG_) FROM SPACEXTBL WHERE Customer = "NASA (CRS)“
  - *Check how much support they got from NASA in this project*
- SELECT AVG(PAYLOAD_MASS__KG_) FROM SPACEXTBL WHERE Booster_Version = "F9 v1.1“
  - *What is the target market in term of payload*
- SELECT MIN(Date) FROM SPACEXTBL WHERE (Landing_Outcome = "Success (ground pad)")
  - *See how long took to succeed with a medium size payload*
- SELECT Booster_Version FROM SPACEXTBL WHERE ((Landing_Outcome = "Success (drone ship)") AND (PAYLOAD_MASS__KG_>4000) AND (PAYLOAD_MASS__KG_<6000))
  - *Which booster version was used when they became successful (was it a big factor?)*
- SELECT COUNT(Mission_Outcome), Mission_Outcome FROM SPACEXTBL GROUP BY Mission_Outcome
  - *Check which were the main reason for a mission failure*
- SELECT Booster_Version FROM SPACEXTBL WHERE (PAYLOAD_MASS__KG_ = (SELECT MAX(PAYLOAD_MASS__KG_) FROM SPACEXTBL))
  - *See which booster was used for heavy payloads*
- SELECT SUBSTRING(SPACEXTBL.Date, 6,2), Landing_Outcome, Booster_Version,Launch_Site FROM SPACEXTBL
  WHERE ((SUBSTRING(Date, 0, 5) = 2015) AND (Landing_Outcome LIKE ''%drone ship%‘’))
  - *What was the factor change that made the mission successful, if any*
- SELECT Landing_Outcome, COUNT(Landing_Outcome) from SPACEXTBL
  WHERE (Date > "2010-06-04") GROUP BY Landing_Outcome ORDER BY (COUNT(Landing_Outcome)) DESC
  - *What were the most frequent reason for a landing failure*

EDA with SQL

- *Github File location*
- https://github.com/albadhuyberechts/IBMDataScience/blob/main/IBM_Final/jupyter-labs-eda-sql-coursera_sqllite.ipynb

Build an Interactive Map with Folium

- On each launch site we added
  - a marker to identify its name
  - A circle to show the number of launches and whether they were mainly successful (green circle) or a failure (red colour)
  - Within each circle, when zoomed in, we showed smaller cricles with the successful and failed launches and their details
- Additionally, we added lines showing how far each launch site is from the coast, as the boosters are recovered at sea (distance to bring them back)

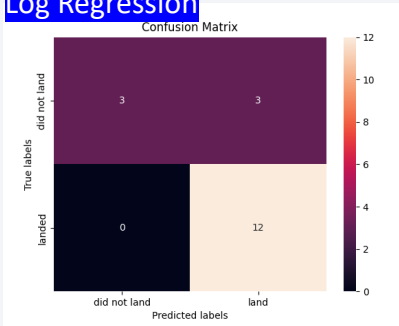
- *Github File location*
- https://github.com/albadhuyberechts/IBMDataScience/blob/main/IBM_Final/lab_jupyter_launch_site_location.jupyterlite.ipynb

Build a Dashboard with Plotly Dash

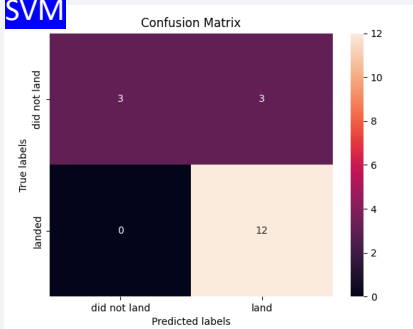
- Summarize what plots/graphs and interactions you have added to a dashboard
- Explain why you added those plots and interactions
- Add the GitHub URL of your completed Plotly Dash lab, as an external reference and peer-review purpose

Predictive Analysis (Classification)

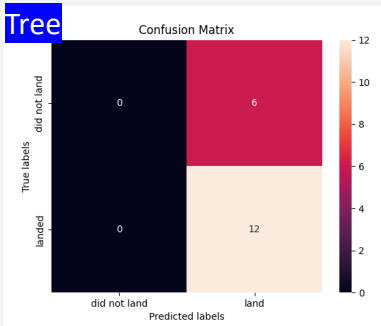
Log Regression



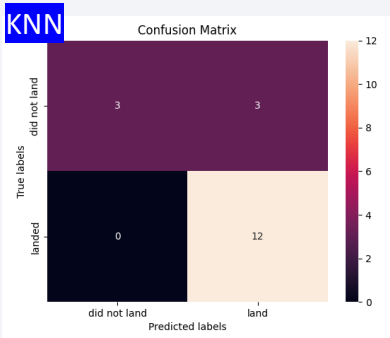
SVM

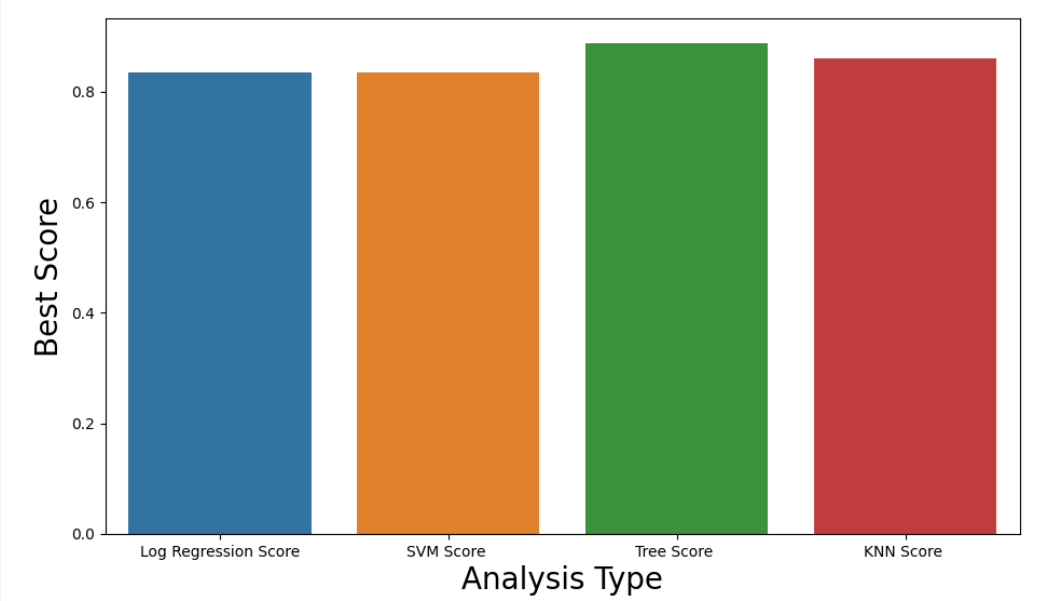


Tree



KNN



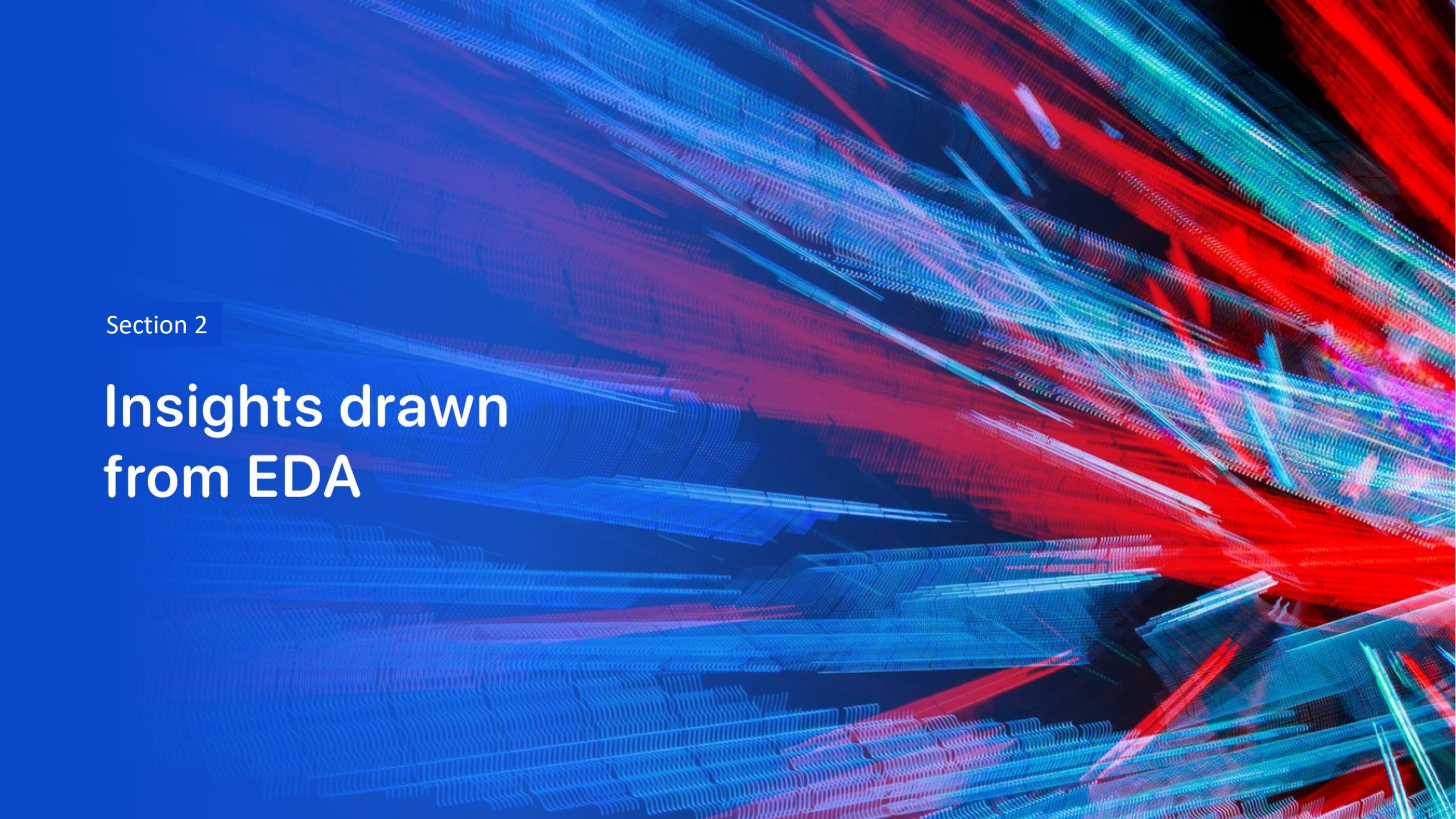


Github File location

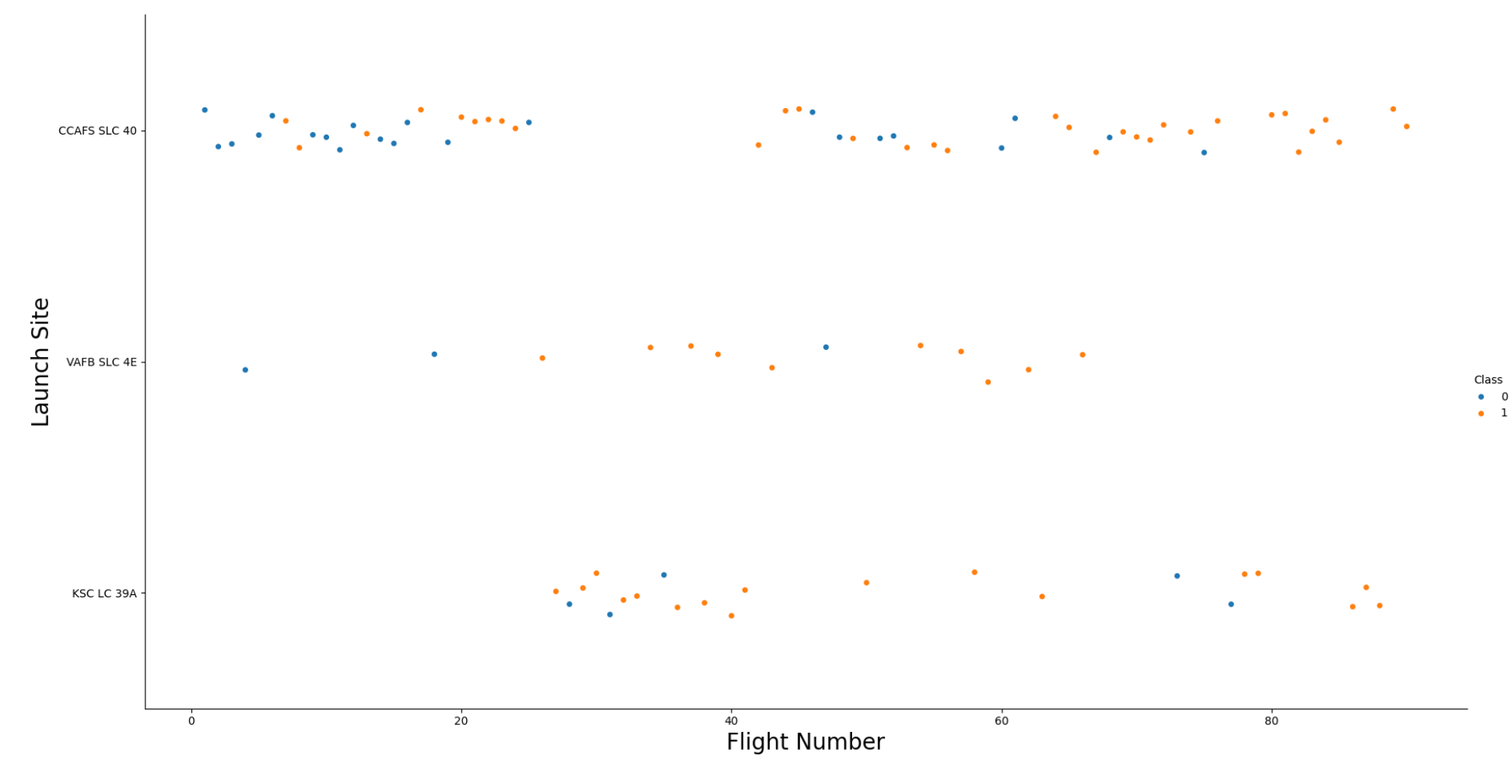
https://github.com/albadhuyberechts/IBMDataScience/blob/main/IBM_Final/SpaceX_Machine_Learning_Prediction_Part_5.jupyterlite.ipynb

Results

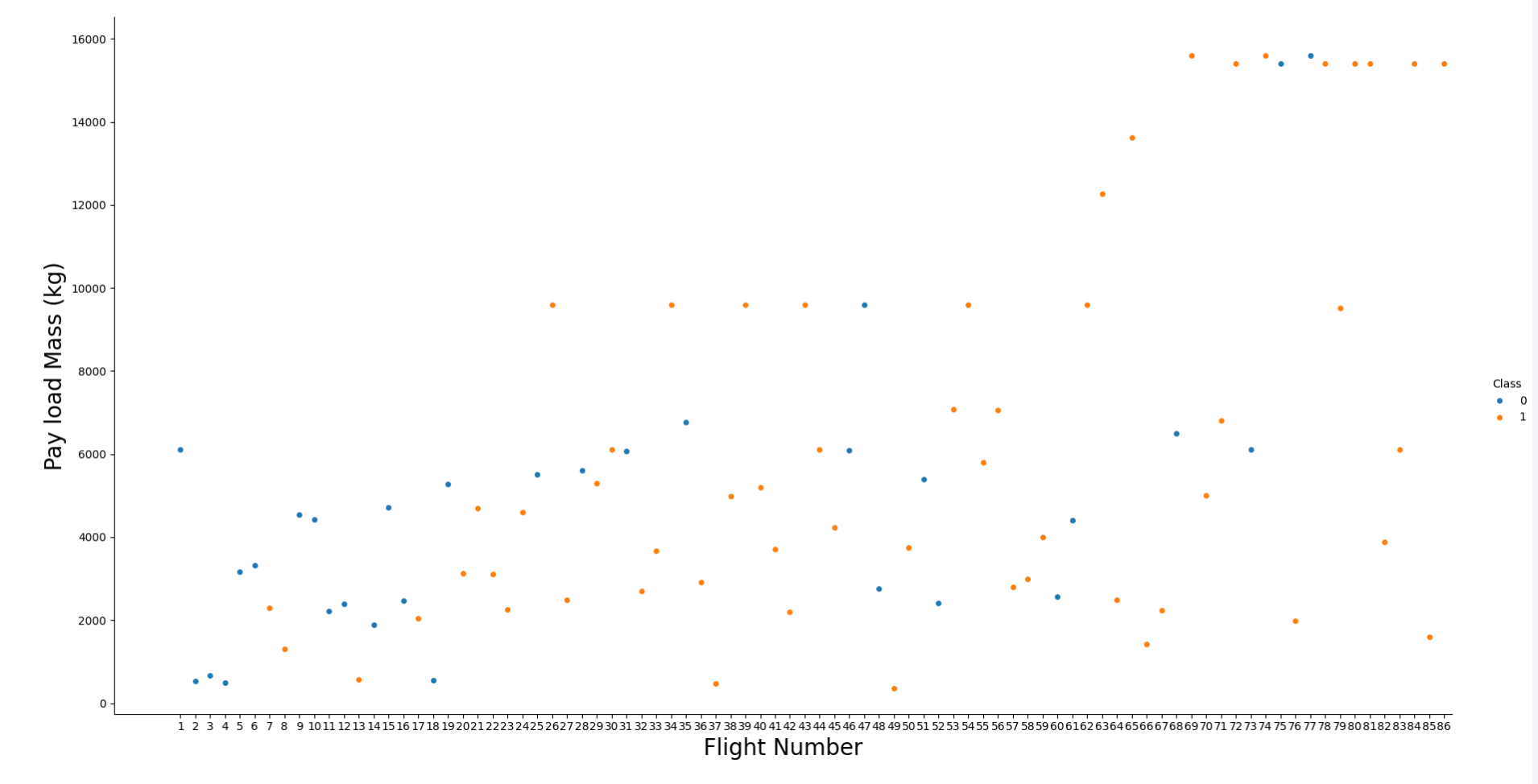
- The model with the best accuracy is officially the Tree based model, but on the test sample it only predicts success.
- The test sample is fairly small, so this might be a fluke, we will need to further analyze the outcomes, probably using more random split between the train and test data, and test on future data
- Each model are very good at predicting the success and very bod at predicting the failure. Probably because the failure were due to factor not captured, like :
  - What technical aspect created the failure and for how many more flights was that technical problem not yet fixed
  - The weather impact (wind, sea waves, etc.) at the landing site and their influence on the landing success
  - Others


Section 2
Insights drawn from EDA

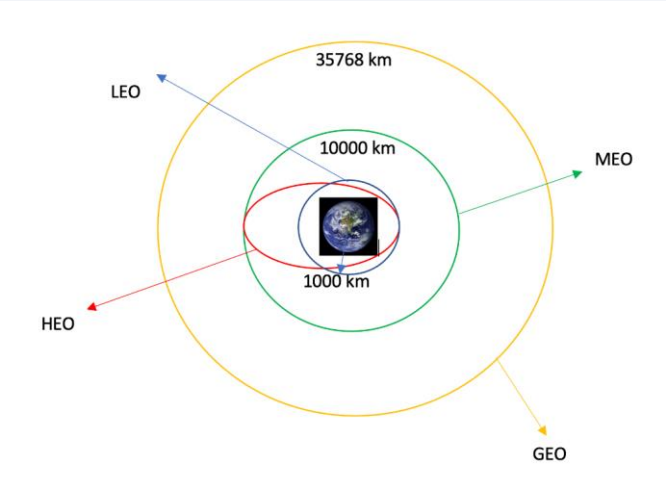
Flight Number vs. Launch Site

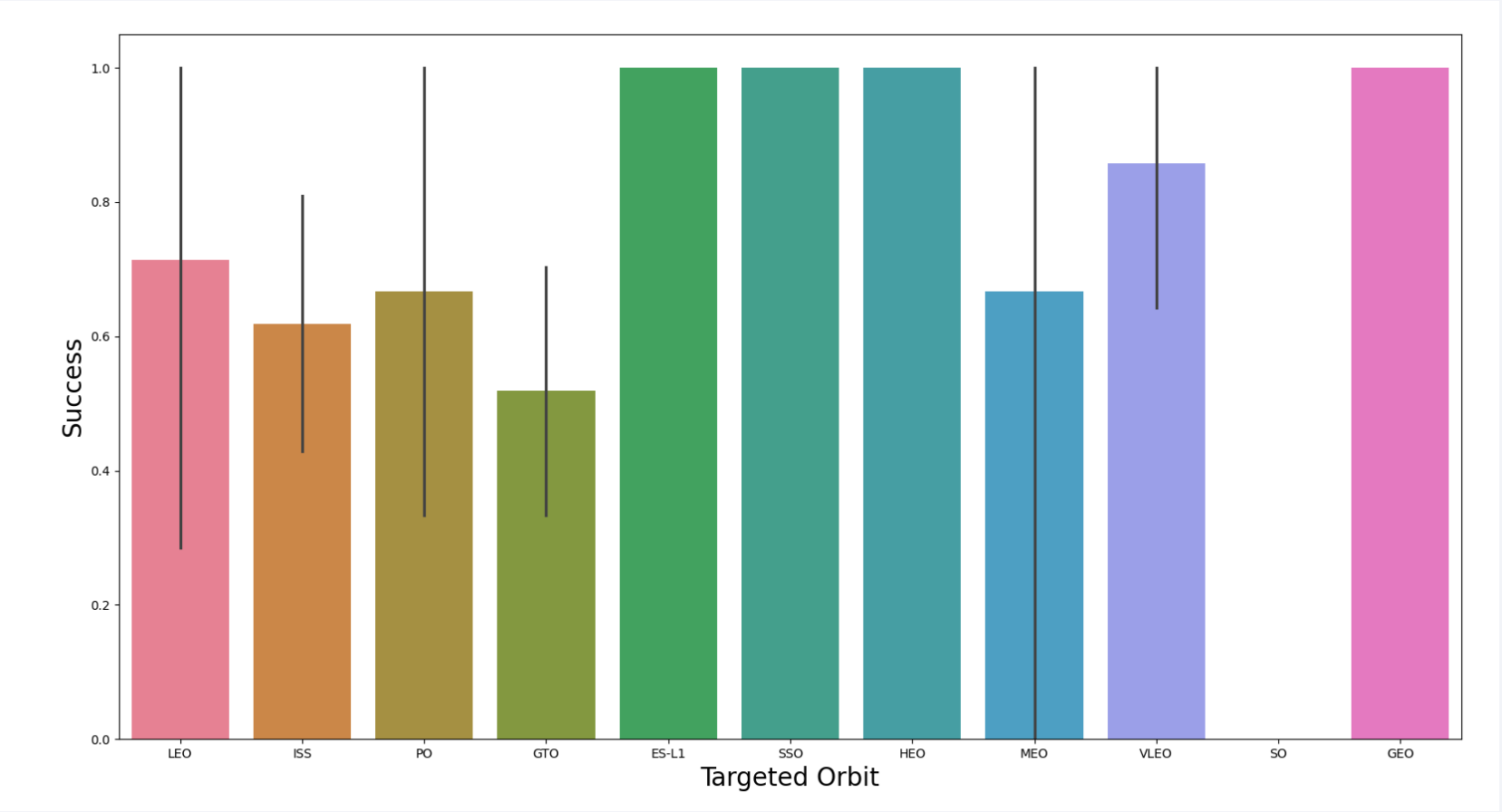


Payload vs. Launch Site

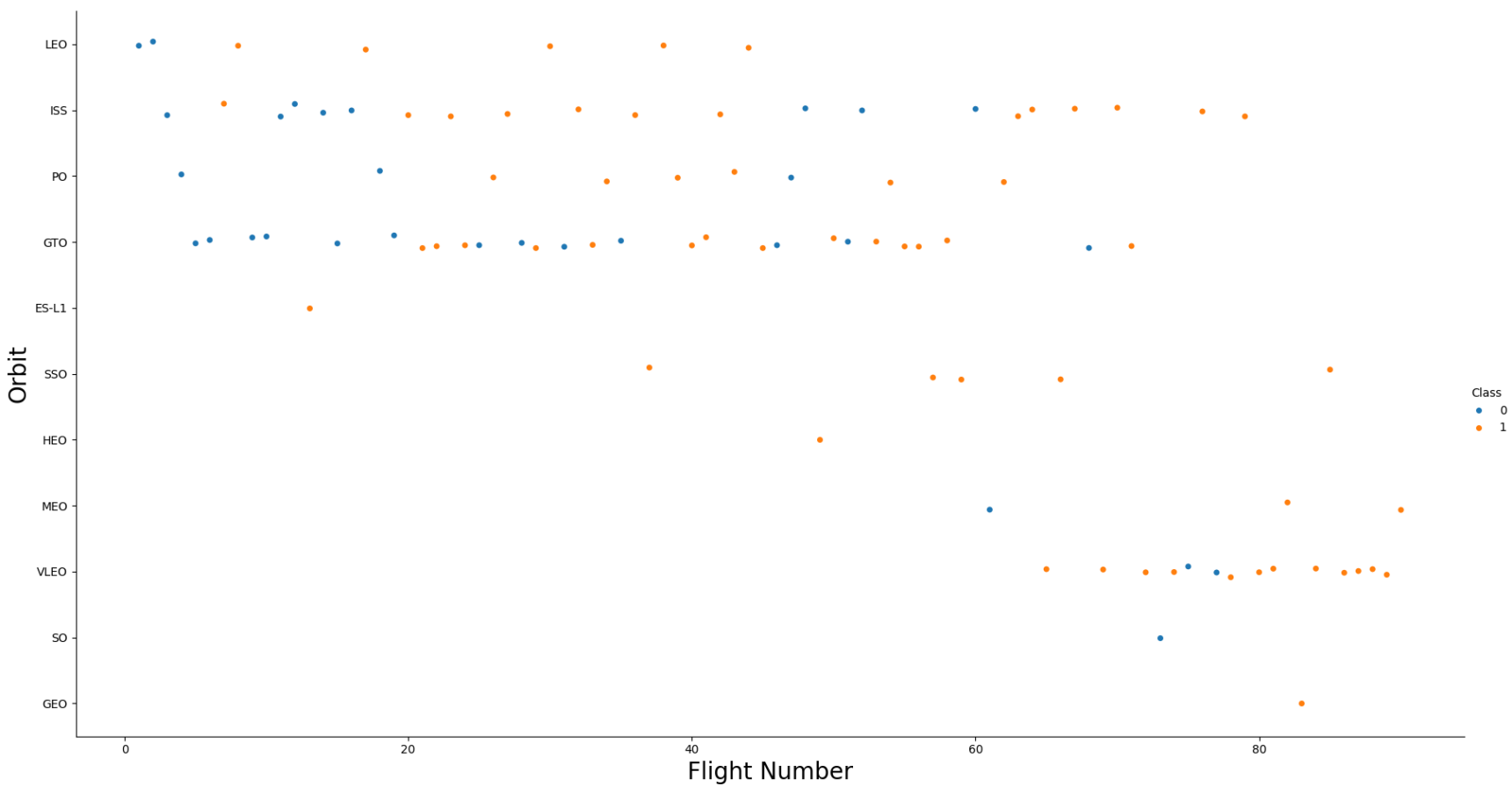


Success Rate vs. Orbit Type

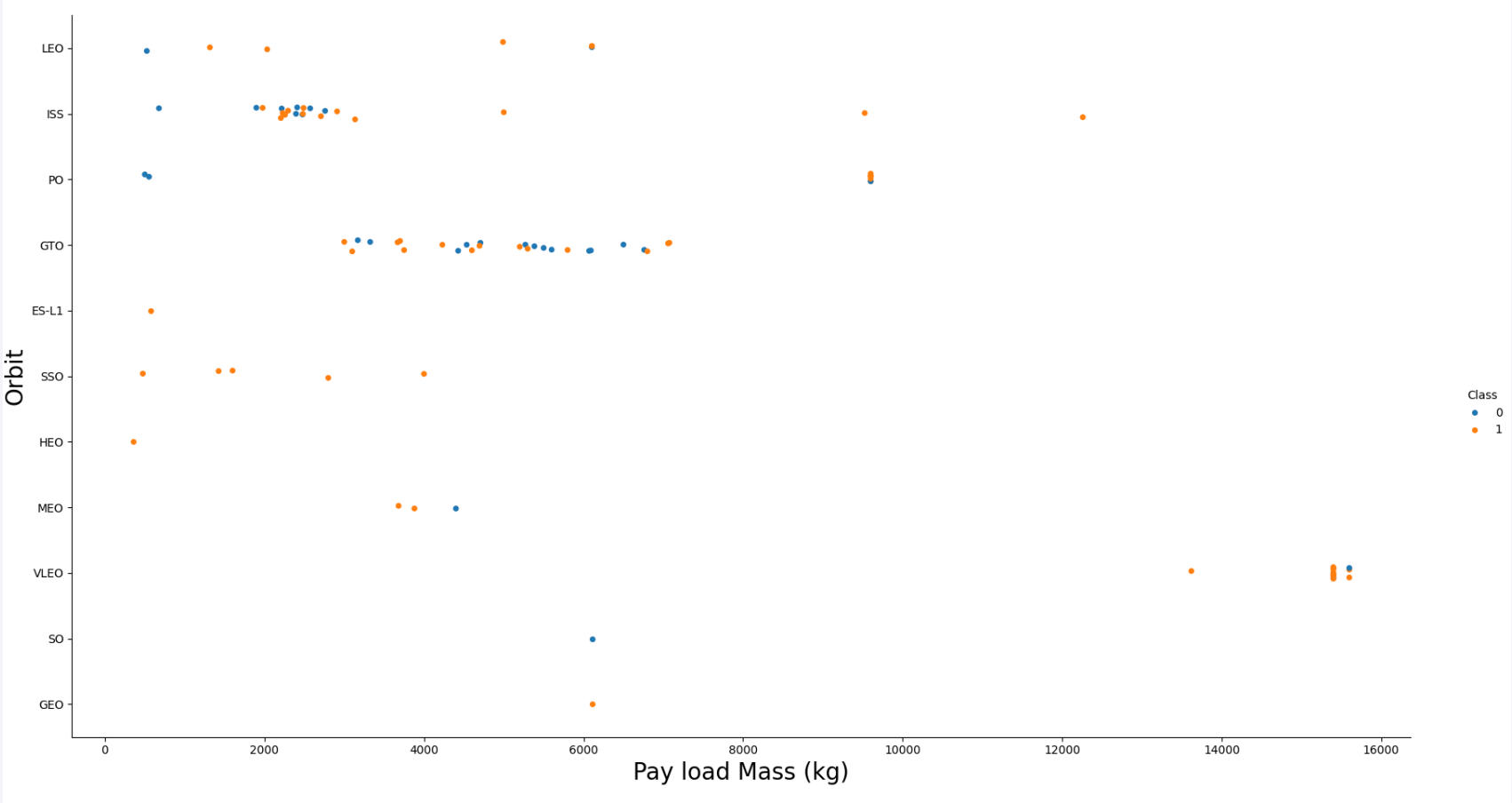




Flight Number vs. Orbit Type

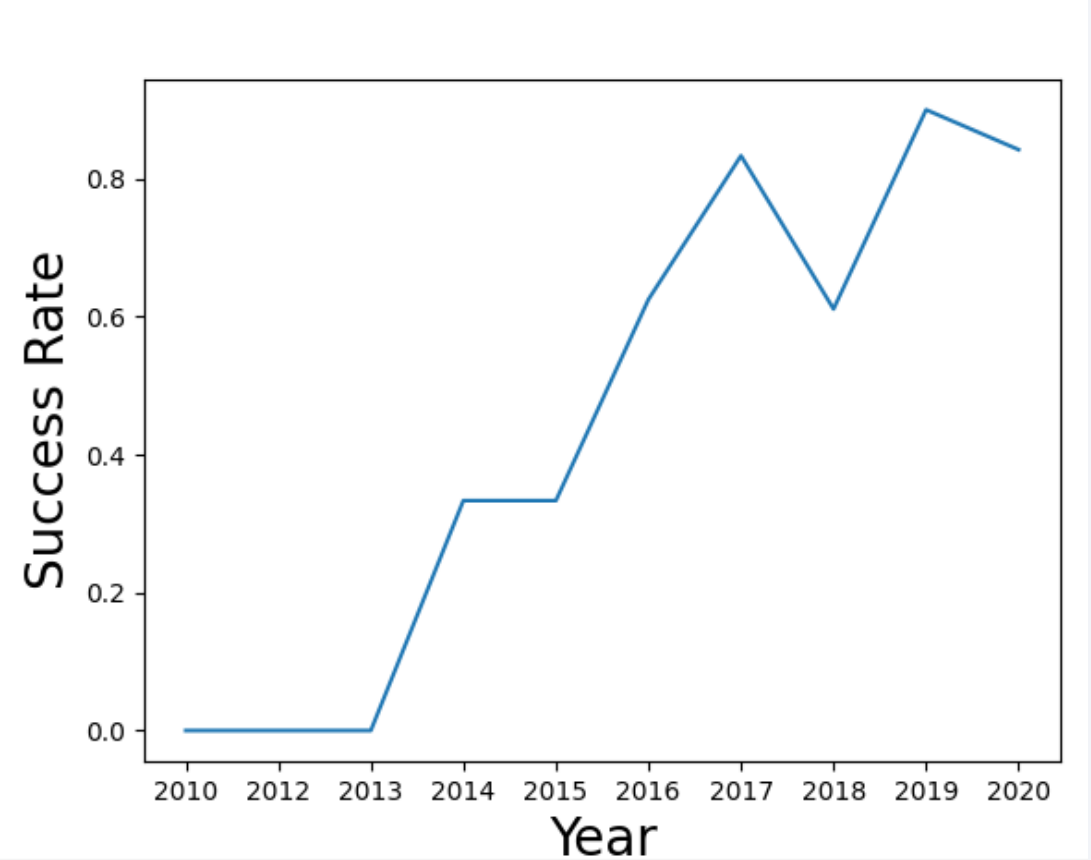


Payload vs. Orbit Type



Launch Success Yearly Trend

- Shows that the success rate was very low until the technique was well developed, and the it remained stable



All Launch Site Names

Task 1

Display the names of the unique launch sites in the space mission

```
%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

CCA FS stands for Cap Canaveral Launch Centre

KSC stands for Kennedy Space Centre

VAFB stands for Vandenberg Space Force Base

Launch Site Names Begin with 'CCA'

Task 2

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

```
%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	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

CCA… stands for Cap Canaveral sites, where NASA operates several launch pads

As they offer little difference with one another, we might want to regroup them as one single site in our data analysis

Total Payload Mass

Task 3

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

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

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

SUM(PAYLOAD_MASS__KG_)
45596

The total payload represents the size of the market captured by SpaceX and us a good indicator of what volume can be achieved, knowing that SpaceX does not have the capacity to satisfy all the possible customers

Average Payload Mass by F9 v1.1

Task 4

Display average payload mass carried by booster version F9 v1.1

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

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

AVG(PAYLOAD_MASS__KG_)
2928.4

This number represents the average weight of the payload taken by a rocket with F9 V1.1 booster

First Successful Ground Landing Date

Task 5

List the date when the first succesful landing outcome in ground pad was acheived.

Hint:Use min function

```
%sql SELECT MIN(Date) FROM SPACEXTBL WHERE (Landing_Outcome = "Success (ground pad)")
```

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

MIN(Date)
2015-12-22

This date is the first time when a booster was correctly recovered on a pad on the gound

Successful Drone Ship Landing with Payload between 4000 and 6000

Task 6

List the names of the boosters which have success in drone ship and have payload mass greater than 4000 but less than 6000

```
%sql SELECT Booster_Version FROM SPACEXTBL 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

It took several iterations to develop a booster which could bring a substantial payload in space and be recovered

The more power full the booster, the heavier and thus the more difficult it is to recover it successfully

Here are the versions that finally worked

Total Number of Successful and Failure Mission Outcomes

Task 7

List the total number of successful and failure mission outcomes

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

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

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

This graph shows that, although a large number of mission initially failed to lead to the recovery of the booster, most mission succeeded in putting their payload in orbit

Boosters Carried Maximum Payload

Task 8

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

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

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

Booster_Version
F9 B5 B1048.4
F9 B5 B1049.4
F9 B5 B1051.3
F9 B5 B1056.4
F9 B5 B1048.5
F9 B5 B1051.4
F9 B5 B1049.5

It took some time to develop a power booster which could be recovered, and when this was done, various versions had to be created to bring payloads at various orbits. Here is their list.

2015 Launch Records

	Landing_Outcome	Booster_Version	Launch_Site
01	Failure (drone ship)	F9 v1.1 B1012	CCAFS LC-40
04	Failure (drone ship)	F9 v1.1 B1015	CCAFS LC-40
06	Precluded (drone ship)	F9 v1.1 B1018	CCAFS LC-40

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

Task 10

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.

```
%sql SELECT Landing_Outcome, COUNT(Landing_Outcome) from SPACEXTBL WHERE (Date > "2010-06-04") GROUP BY Landing_Outcome ORDER BY (COUNT(Landing_Outcome)) DESC
```

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

Landing_Outcome	COUNT(Landing_Outcome)
Success	38
No attempt	21
Success (drone ship)	14
Success (ground pad)	9
Failure (drone ship)	5
Controlled (ocean)	5
Failure	3
Uncontrolled (ocean)	2

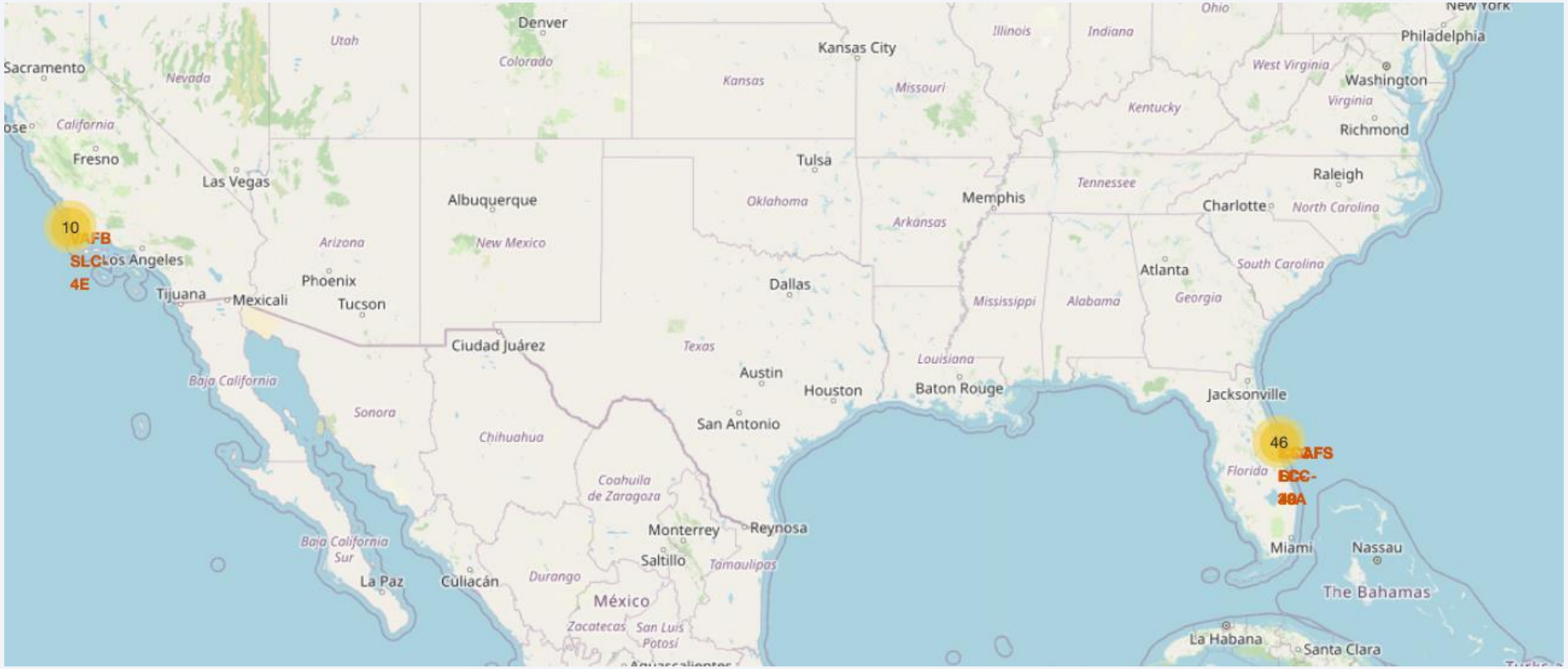
This represents the number of successes and the number of failures sorted by failure causes

Section 3

Launch Sites Proximities Analysis

Location of launch sites

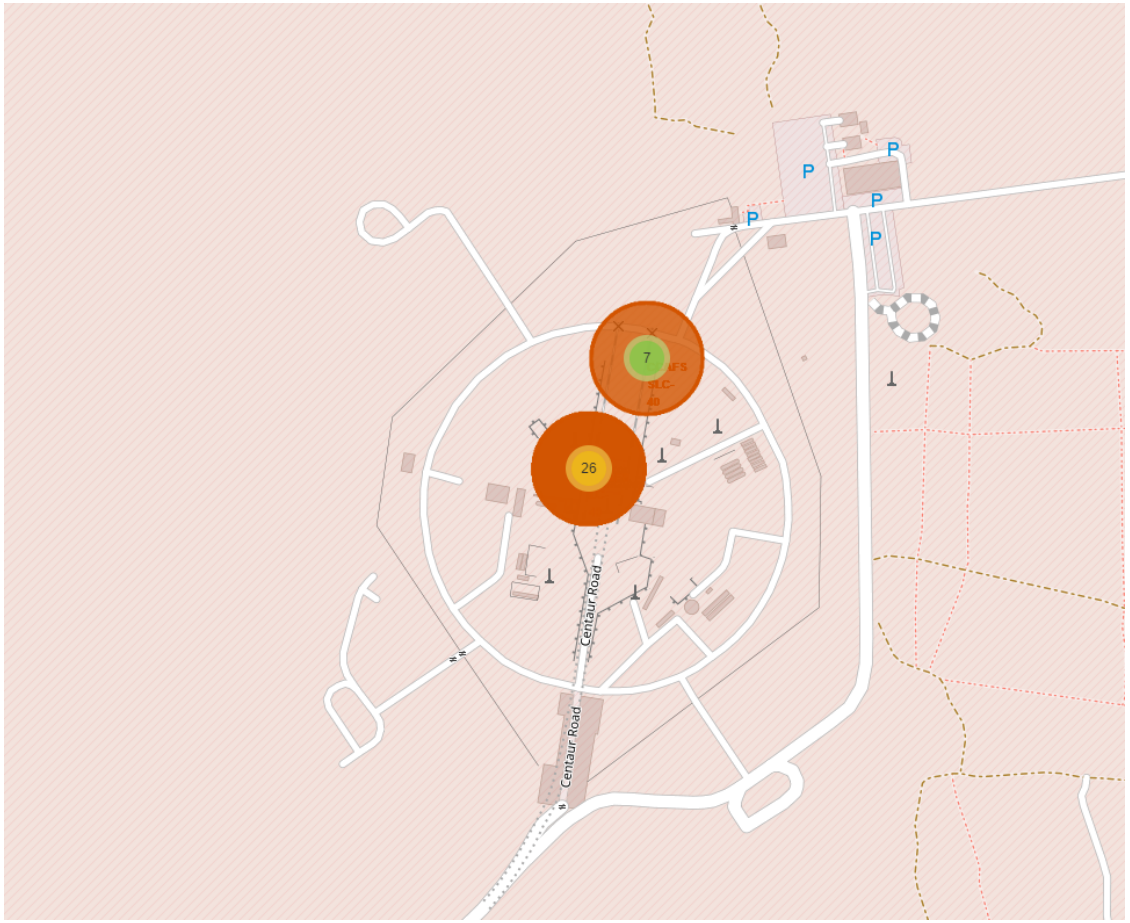
The launch sites used are al close the shore as the boosters are recovered at sea and need to be brought back to the launch site for reuse



Number of Launch per Launch Pad

This blown out map shows the launch pads used at Cap Canaveral, the number of time it was used and whether booster used for that launch were successfully recovered or not (majority recovered - green, unrecovered - red), since the start of the program

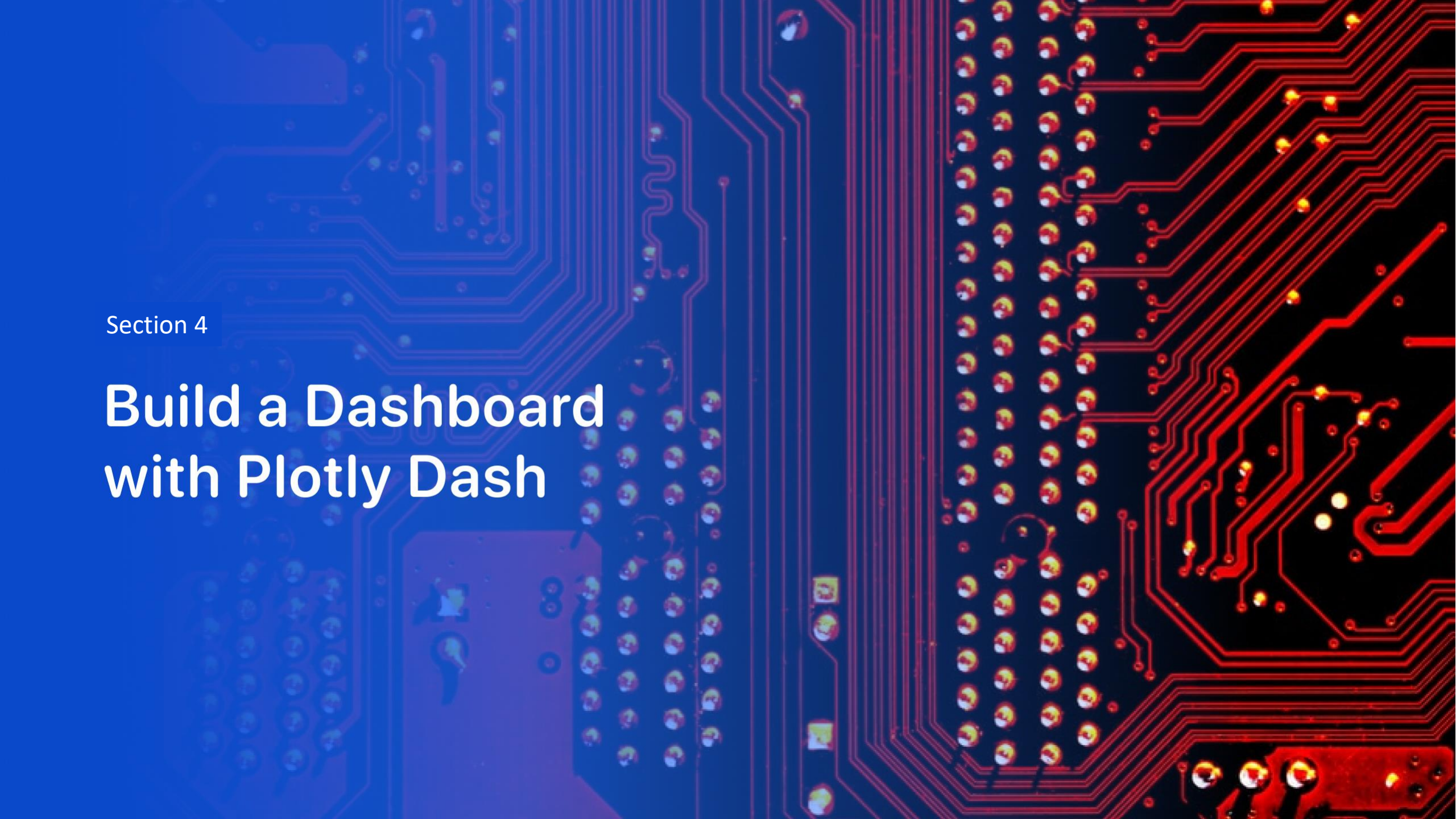
We need to keep in mind that a lot of the early flights where not very successful at recovering the booster and thus the recovery color might not represent the current recovery probability



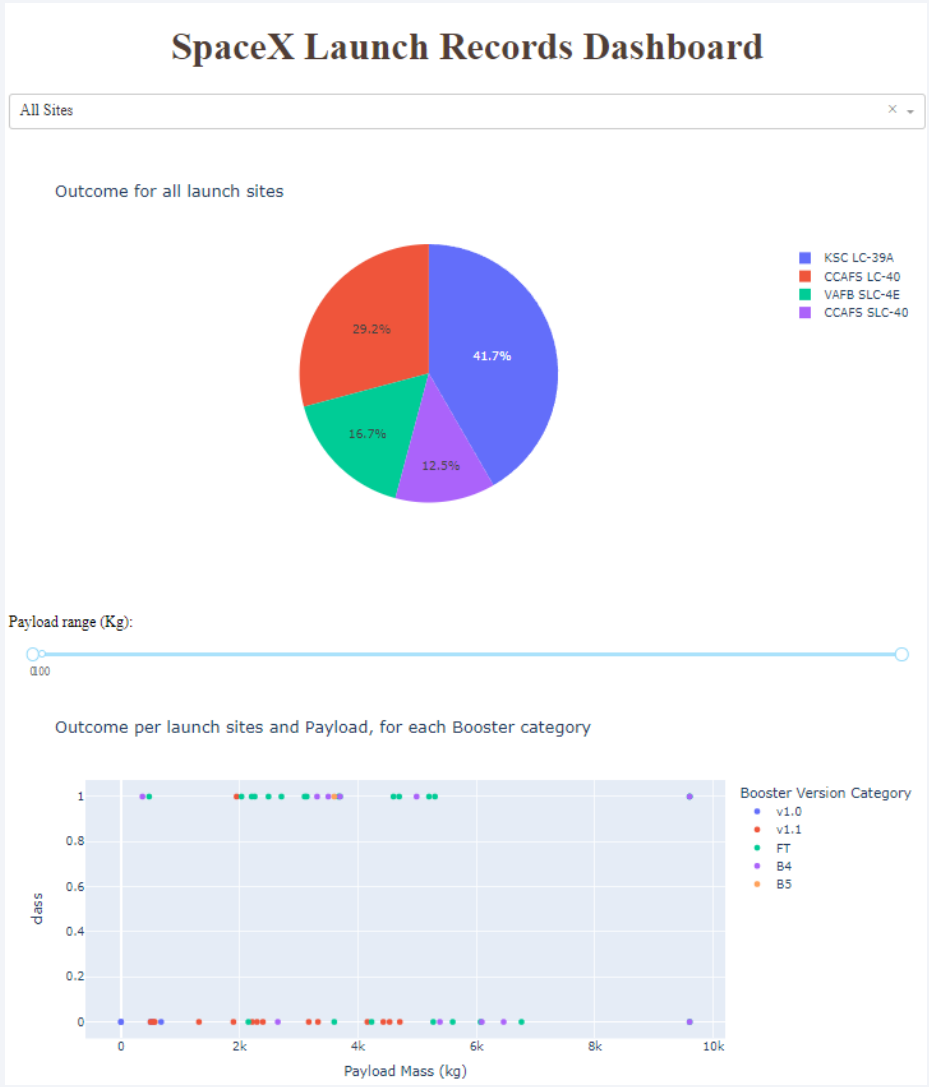
Recovery rate per launch pad

As the previous slide, this blown out map shows the launch pads used at Cap Canaveral, the number of time it was used and whether booster used for that launch were successfully recovered or not (majority recovered - green, unrecovered - red), since the start of the program

It shows that the early flights were indeed not very successful at leading to the recovery of the booster, but that it did improve substantially with time, confirming our suspicion on the previous slide


Section 4
Build a Dashboard with Plotly Dash

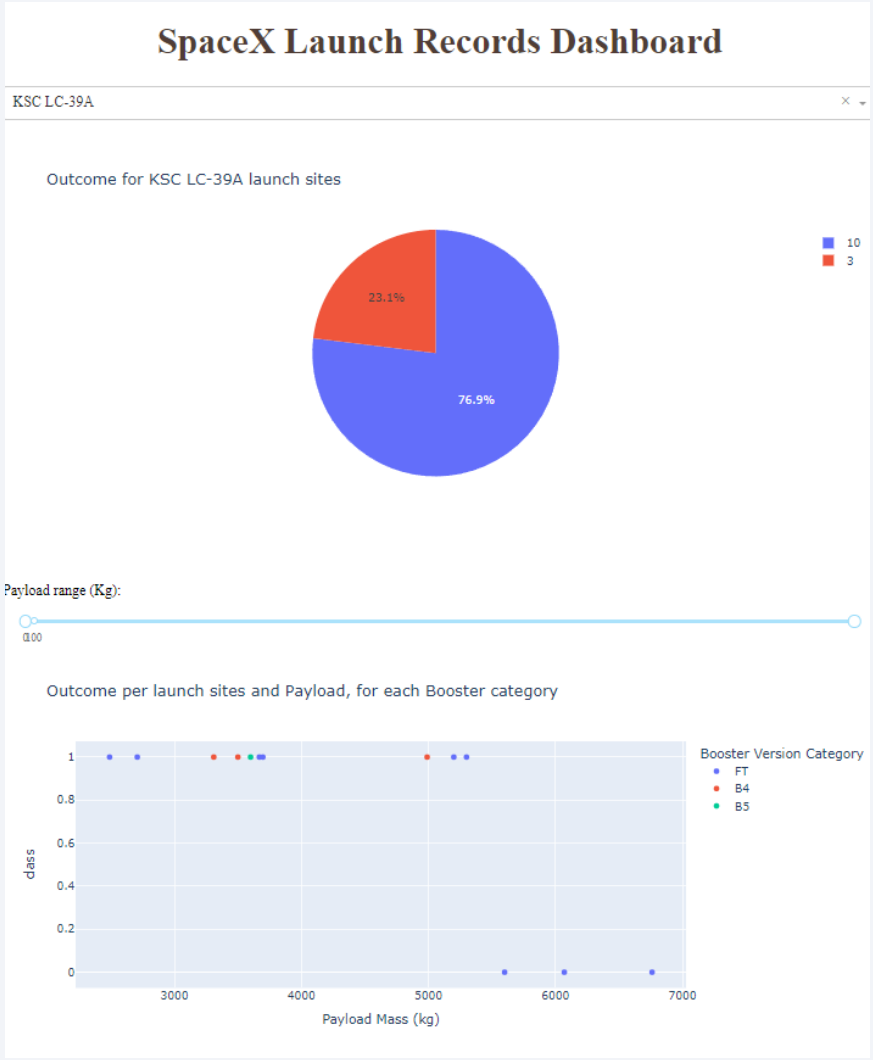
Number of successful recovery for all sites



The pie chart gives the percentage of mission with a successful recovery of the booster per launch pad compared to all the missions with a successful recovery of the booster

The bottom chart shows the successful and failed recovery per launch pad and payload mass

Number of Launch per Launch Pad

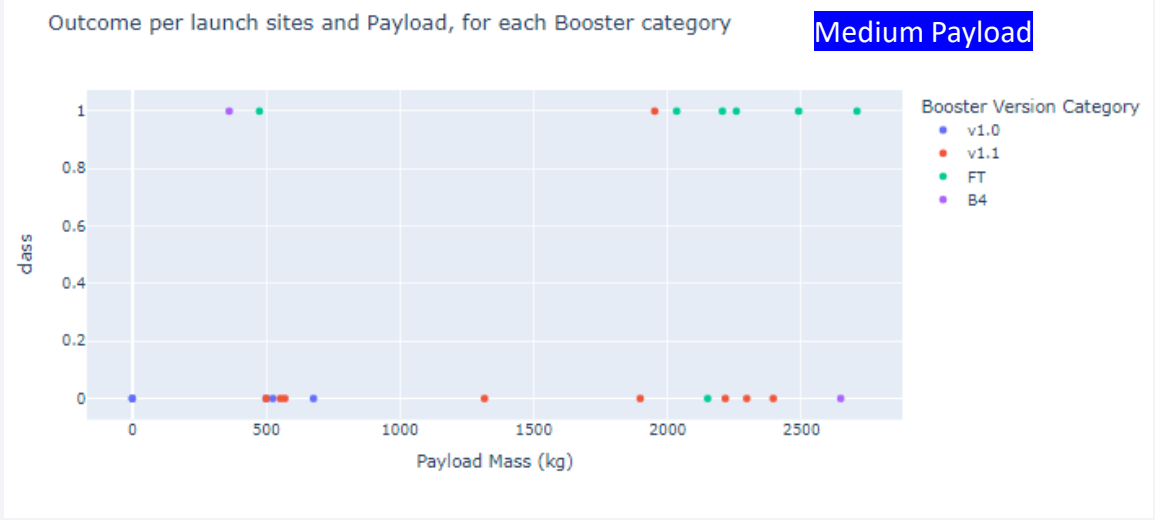


The launch pad with the highest recovery's success is the Houston site the Kennedy Space Center

This number is to be taken with a grain of salt, as KSC was only used when the technology had been perfected, and thus iyt does not in its history the early failures from the other sites

It was used only for a short while, probably because of its position more inland and the difficulty to bring a used booster from the seashore back to the Center

Recovery rate per Payload and Booster



For small payload, the booster used were the FT and B4, each with poor recovery rate, probably because small payloads were used during the early days

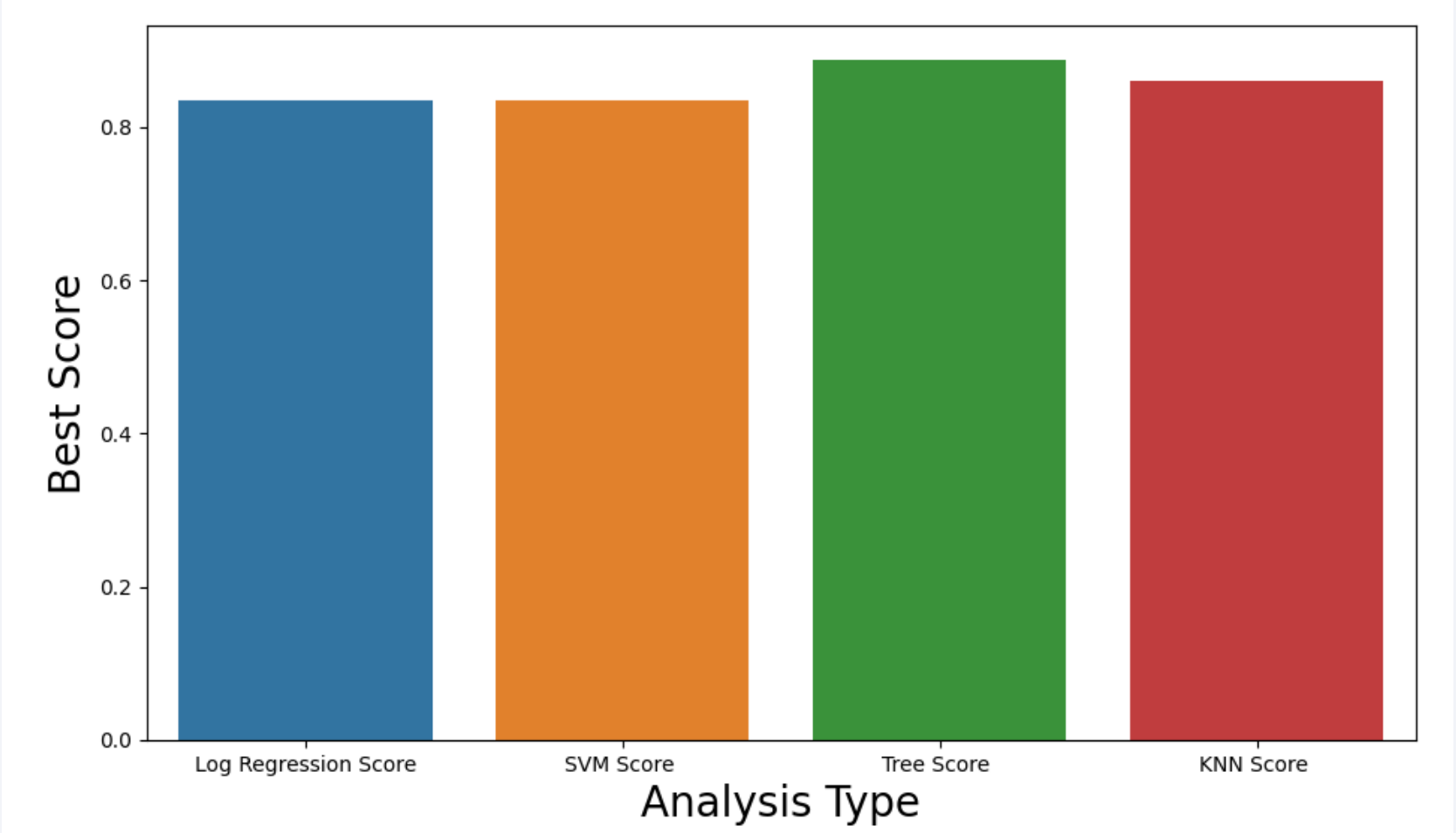
For medium and large payload, the FT and B4 show a very good recovery rate, confirming the point above

Section 5

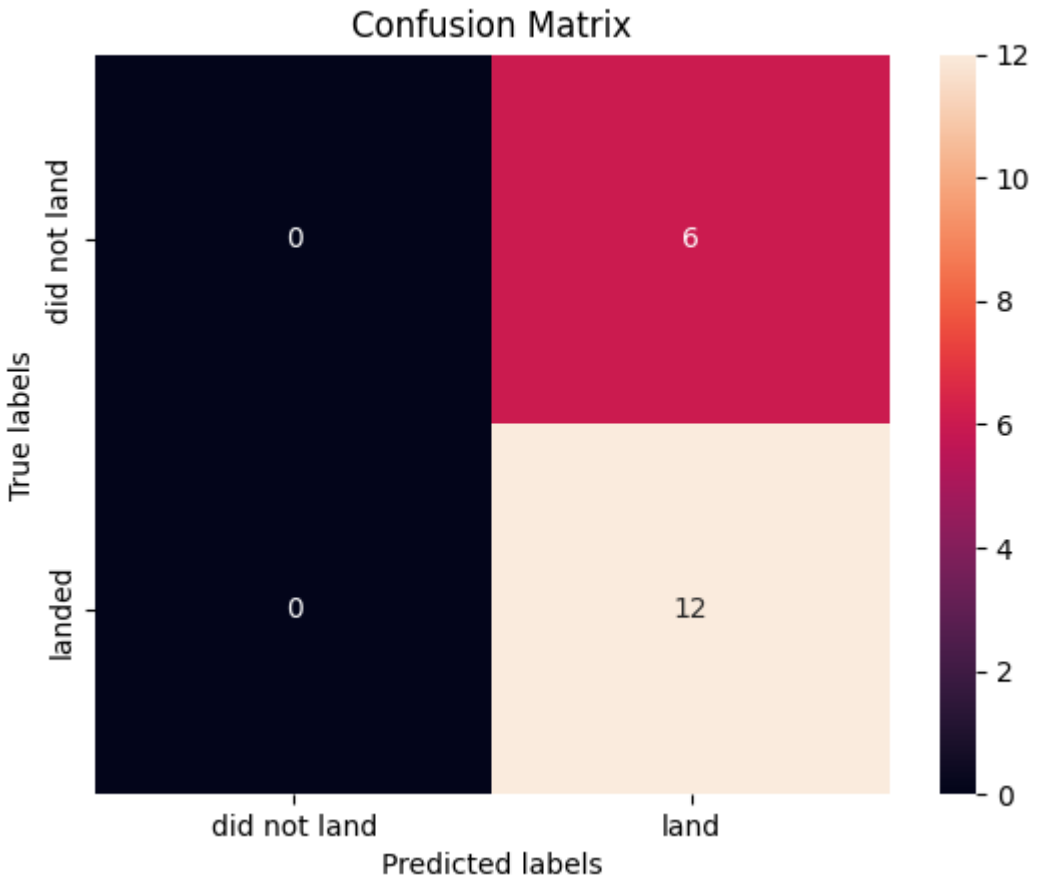
Predictive Analysis (Classification)

Classification Accuracy

The highest accuracy (labelled “Score”) is achieved with a Decision Tree Classifier model



Confusion Matrix



The confusion matrix shows that, although the model seems to show a great accuracy with the training sample, it also seem to predict always success, which is not very useful.

This being said, the other models predict only 50% of the time correctly a failure.

This result seems odd and requires further investigation.

Conclusions

- The existing boosters will nearly all the time bring their payload at destination, even at the beginning of the operations
- It could take up to two years for developing the knowledge on how to recover a booster systematically
- The location of the launch site does not really influence recovery rate, but influence the cost of bringing the booster back to the launch pad
- The average payload capacity required by the customers is about 3,000 kg and so the project should be built with that target in mind

Appendix: Data Collection

Task 1: Request and parse the SpaceX launch data using the GET request

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

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

We should see that the request was successfull with the 200 status response code

```
response.status_code
```

```
200
```

Now we decode the response content as a Json using `.json()` and turn it into a Pandas dataframe using `.json_normalize()`

```
# Use json_normalize meethod to convert the json result into a dataframe
data = pd.json_normalize(response.json())
```

Using the dataframe `data` print the first 5 rows

```
# Get the head of the dataframe
print(data.head)
```

```
<bound method NDFrame.head of          static_fire_date_utc  static_fire_date_unix    net  window  \
0    2006-03-17T00:00:00.000Z           1.142554e+09  False     0.0
1                        None                    NaN  False     0.0
2                        None                    NaN  False     0.0
3    2008-09-20T00:00:00.000Z           1.221869e+09  False     0.0
4                        None                    NaN  False     0.0
..                        ...                    ...    ...     ...
182                      None                    NaN  False     NaN
183                      None                    NaN  False     NaN
184                      None                    NaN  False     NaN
185                      None                    NaN  False     NaN
186                      None                    NaN  False     NaN

                     rocket success  \
0    5e9d0d95eda69955f709d1eb   False
1    5e9d0d95eda69955f709d1eb   False
```

Appendix: Data Collection

Task 2: Filter the dataframe to only include `Falcon 9` launches

Finally we will remove the Falcon 1 launches keeping only the Falcon 9 launches. Filter the data d
data to a new dataframe called `data_falcon9`.

```
# Hint data['BoosterVersion']!='Falcon 1'
data_falcon9 = newData.loc[newData['BoosterVersion']!='Falcon 1']
```

Now that we have removed some values we should reset the FlgihtNumber column

```
data_falcon9.loc[:,'FlightNumber'] = list(range(1, data_falcon9.shape[0]+1))
data_falcon9
```

```
/home/jupyterlab/conda/envs/python/lib/python3.7/site-packages/pandas/core/indexing.py:1773: Se
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/i
  self._setitem_single_column(ilocs[0], value, pi)
```

	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights
4	1	2010-06-04	Falcon 9	NaN	LEO	CCSFS SLC 40	None None	1
5	2	2012-05-22	Falcon 9	525.0	LEO	CCSFS SLC 40	None None	1
6	3	2013-03-01	Falcon 9	677.0	ISS	CCSFS SLC 40	None None	1
7	4	2013-09-29	Falcon 9	500.0	PO	VAFB SLC 4E	False Ocean	1
8	5	2013-12-03	Falcon 9	3170.0	GTO	CCSFS SLC 40	None None	1

Appendix: Data Collection

Task 3: Dealing with Missing Values

Calculate below the mean for the `PayloadMass` using the `.mean()`. Then u

```
# Calculate the mean value of PayloadMass column
meanPayLoad = data_falcon9.loc[:,'PayloadMass'].mean(skipna=True)
print(data_falcon9.loc[:,'PayloadMass'])
print('Average payload : ', meanPayLoad)
# Replace the np.nan values with its mean value
data_falcon9.loc[:,'PayloadMass'].fillna(meanPayLoad, inplace = True)
print(data_falcon9.loc[:,'PayloadMass'])
```

```
4         NaN
5       525.0
6       677.0
7       500.0
8      3170.0
       ...
89    15600.0
90    15600.0
91    15600.0
92    15600.0
93     3681.0
Name: PayloadMass, Length: 90, dtype: float64
Average payload :  6123.547647058824
4      6123.547647
5       525.000000
6       677.000000
7       500.000000
8      3170.000000
          ...
89    15600.000000
90    15600.000000
91    15600.000000
92    15600.000000
93     3681.000000
Name: PayloadMass, Length: 90, dtype: float64
```

Appendix: Web Scrapping

TASK 1: Request the Falcon9 Launch Wiki page from its URL

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

```
# use requests.get() method with the provided static_url
# assign the response to a object
response = requests.get(static_url)
#print(response.content)
```

Create a `BeautifulSoup` object from the HTML `response`

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

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

```
# Use soup.title attribute
print(soup.title.string)
```

```
List of Falcon 9 and Falcon Heavy launches - Wikipedia
```

Appendix: Web Scrapping

TASK 2: Extract all column/variable names from the HTML table header

Next, we want to collect all relevant column names from the HTML table header

Let's try to find all tables on the wiki page first. If you need to refresh your memory about `Beautiful`

```
# Use the find_all function in the BeautifulSoup object, with element type `table`
# Assign the result to a list called `html_tables`
html_tables = soup.find_all("table")
```

Starting from the third table is our target table contains the actual launch records.

```
# Let's print the third table and check its content
first_launch_table = html_tables[2]
print(first_launch_table)
```

```
<table class="wikitable plainrowheaders collapsible" style="width: 100%;">
<tbody><tr>
<th scope="col">Flight No.
</th>
<th scope="col">Date and<br/>time (<a href="/wiki/Coordinated_Universal_Time" title="Coordinated Univ
</th>
<th scope="col"><a href="/wiki/List_of_Falcon_9_first-stage_boosters" title="List of Falcon 9 first-s
-0"><a href="#cite_note-booster-11">[b]</a></sup>
</th>
<th scope="col">Launch site
</th>
<th scope="col">Payload<sup class="reference" id="cite_ref-Dragon_12-0"><a href="#cite_note-Dragon-12
```

Appendix: Web Scrapping

TASK 3: Create a data frame by parsing the launch HTML tables

```
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
            flightNumber = row[0]
            # TODO: Append the flight_number into launch_dict with key `Flight No.`
            launch_dict['Flight No.'].append(flightNumber)
            #print(flight_number)
----------
            datatimelist=date_time(row[0])
----------
            # 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
            #print(bv)
            launch_dict['Version Booster'].append(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`
  #print('Customer: ', row[6])
  customer = row[6].string
  if not(customer):
      customer=row[6].a.string
  launch_dict['Customer'].append(customer)
  #print(customer)
---
  # Launch outcome
  # TODO: Append the launch_outcome into launch_dict with key `Launch outcome`
  launch_outcome = list(row[7].strings)[0]
  launch_dict['Launch outcome'].append(launch_outcome)
  #print(launch_outcome)
---
  # Booster landing
  # TODO: Append the launch_outcome into launch_dict with key `Booster Landing`
  booster_landing = landing_status(row[8])
  launch_dict['Booster landing'].append(booster_landing)
  #print(booster_landing)
```

```
df= pd.DataFrame({ key:pd.Series(value) for key, value in launch_dict.items() })
print(df)
```

```
                                Flight No. Launch site  \
0                  [4 June 2010,, [], 18:45\n]       CCAFS
1    [8 December 2010,, [], 15:43, [[[13]]], \n]       CCAFS
2        [22 May 2012,, [], 07:44, [[[17]]], \n]       CCAFS
3     [8 October 2012,, [], 00:35, [[[21]]], \n]       CCAFS
4                [1 March 2013,, [], 15:10\n]       CCAFS
..                                          ...         ...
116       [9 May 2021, [], 06:42, [[[656]]], \n]       CCSFS
117      [15 May 2021, [], 22:56, [[[659]]], \n]         KSC
118      [26 May 2021, [], 18:59, [[[664]]], \n]       CCSFS
119      [3 June 2021, [], 17:29, [[[667]]], \n]         KSC
120      [6 June 2021, [], 04:26, [[[673]]], \n]       CCSFS
```

Appendix: Data Wrangling

TASK 1: Calculate the number of launches on each site

The data contains several Space X launch facilities: Cape Canaveral Spac
Launch Complex 39A KSC LC 39A .The location of each Launch Is placed

Next, let's see the number of launches for each site.

Use the method `value_counts()` on the column `LaunchSite` to det

```
# Apply value_counts() on column LaunchSite
print(df.loc[:,'LaunchSite'].value_counts())
```

```
CCAFS SLC 40    55
KSC LC 39A      22
VAFB SLC 4E     13
Name: LaunchSite, dtype: int64
```

Appendix: Data Wrangling

TASK 2: Calculate the number and occurrence of each orbit

Use the method `.value_counts()` to determine the number and occurrence c

```
# Apply value_counts on Orbit column
print(df.loc[:,'Orbit'].value_counts())
```

```
GTO      27
ISS      21
VLEO     14
PO        9
LEO       7
SSO       5
MEO       3
ES-L1     1
HEO       1
SO        1
GEO       1
Name: Orbit, dtype: int64
```

Appendix: Data Wrangling

TASK 3: Calculate the number and occurence of mission outcome of the orbits

Use the method `.value_counts()` on the column `Outcome` to determine the number of `landing_ou`

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

```
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
```

`True Ocean` means the mission outcome was successfully landed to a specific region of the ocean while
region of the ocean. `True RTLS` means the mission outcome was successfully landed to a ground pad
pad. `True ASDS` means the mission outcome was successfully landed to a drone ship `False ASDS` mea
`None None` these represent a failure to land.

```
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
```

Appendix: Data Wrangling

TASK 4: Create a landing outcome label from Outcome column

Using the `Outcome`, create a list where the element is zero if the corresponding row i

```
# landing_class = 0 if bad_outcome
# landing_class = 1 otherwise
df['landing_class'] = np.where(df['Outcome'].isin(bad_outcomes), 0, 1)
print(df[['Outcome', 'landing_class']])
print(df[['landing_class']].head(8))
print(df[['landing_class']].mean())
```

```
        Outcome  landing_class
0      None None              0
1      None None              0
2      None None              0
3    False Ocean              0
4      None None              0
..          ...            ...
85     True ASDS              1
86     True ASDS              1
87     True ASDS              1
88     True ASDS              1
89     True ASDS              1

[90 rows x 2 columns]
```

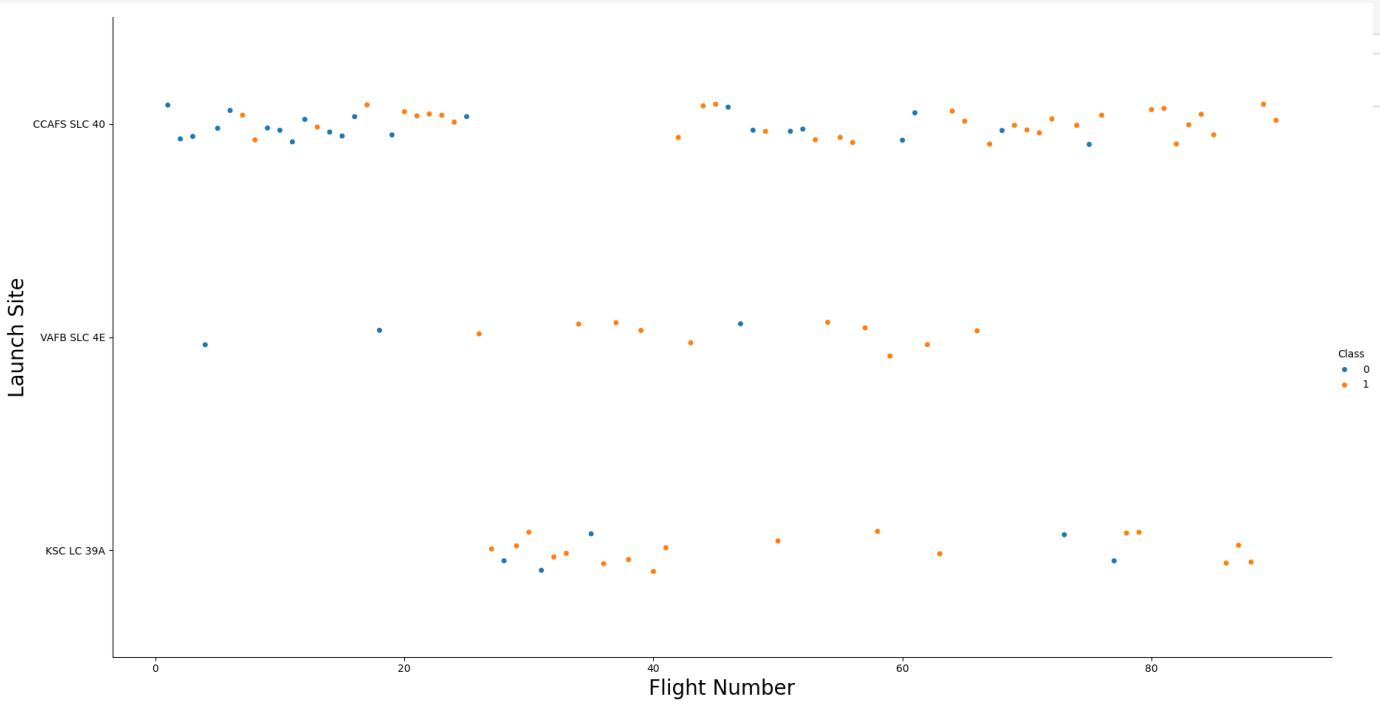
Appendix: EDA Visualization

```
### TASK 1: Visualize the relationship between Flight Number and Launch Site
```

Use the function `catplot` to plot `FlightNumber` vs `LaunchSite`, set the parameter `x` parameter to `FlightNumber`,set the `y` to `Launch Site` and set the parameter `hue` to `'class'`

```
# Plot a scatter point chart with x axis to be Flight Number and y axis to be the launch site, and hue to be the class value
sns.catplot(y="Launch Site", x="FlightNumber", hue="Class", data=df, aspect = 5)
plt.xlabel("Flight Number",fontsize=20)
plt.ylabel("Launch site",fontsize=20)
plt.show()
```

Now try to explain the patterns you found in the Flight Number vs.



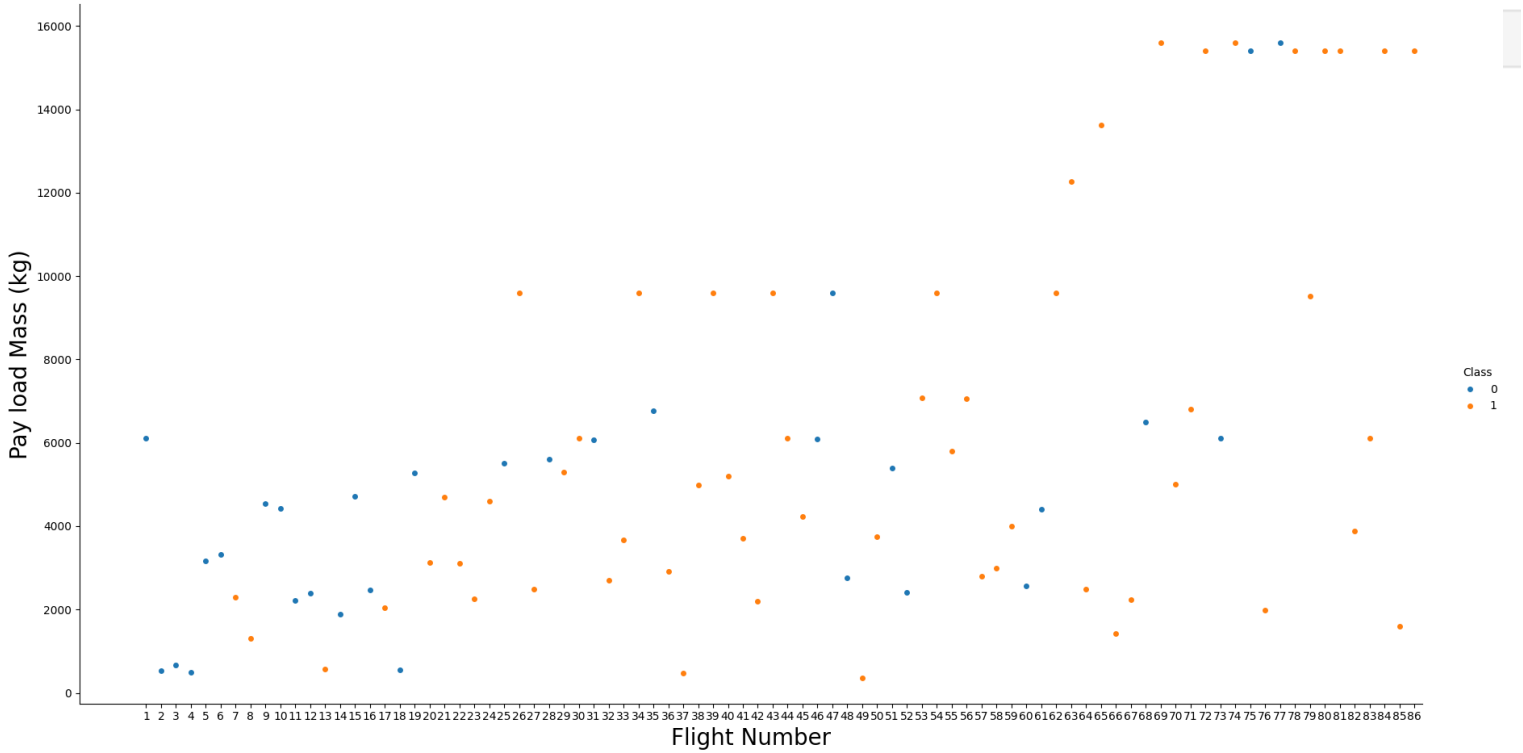
Appendix: EDA Visualization

```
### TASK 2: Visualize the relationship between Payload and Launch Site
sns.catplot(y="PayloadMass", x="Launch Site", hue="Class", data=df, aspect = 5)
plt.xlabel("Launch site",fontsize=20)
plt.ylabel("Pay load Mass (kg)",fontsize=20)
plt.show()
```

We also want to observe if there is any relationship between launch sites and their payload mass.

```
# Plot a scatter point chart with x axis to be
```

Now if you observe Payload Vs. Launch Site scatter poir



Appendix: EDA Visualization

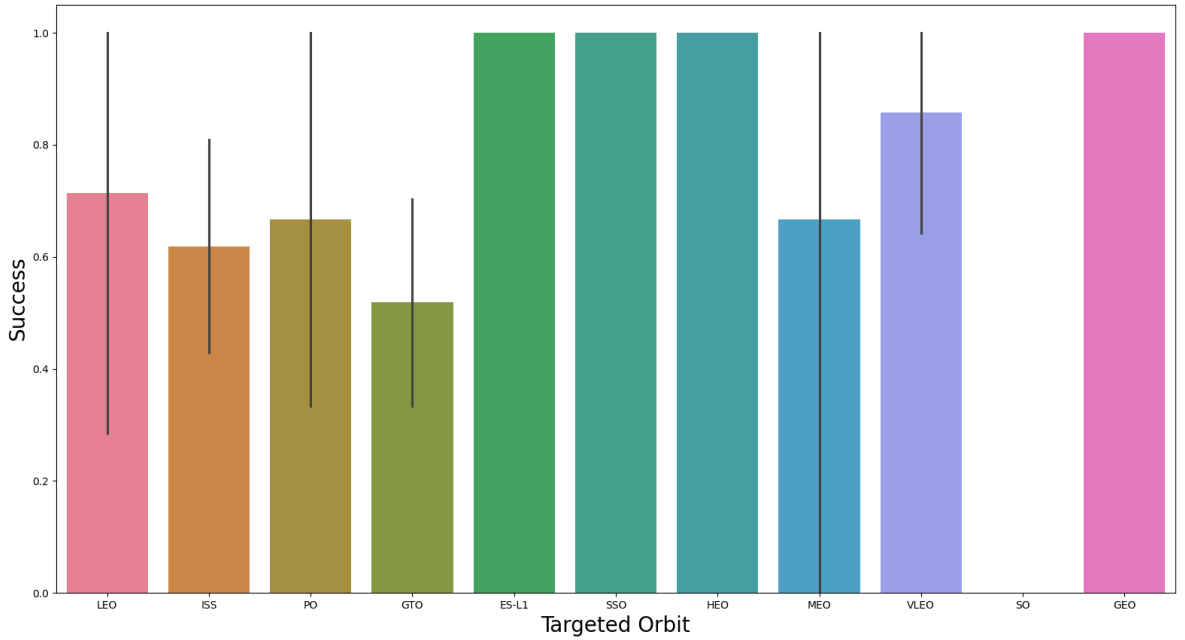
```
### TASK  3: Visualize the relationship between success rate of each orbit type
```

Next, we want to visually check if there are any relationship between success rate and orbit type.

Let's create a `bar chart` for the sucess rate of each orbit

```
# HINT use groupby method on Orbit column and get the mean of Class column
sns.barplot(x='Orbit', y='Class', hue='Orbit', data=df)
plt.xlabel("Targeted Orbit",fontsize=20)
plt.ylabel("Success Rate",fontsize=20)
plt.show()
```

Analyze the ploted bar chart try to find which orbits have high sucess rate.



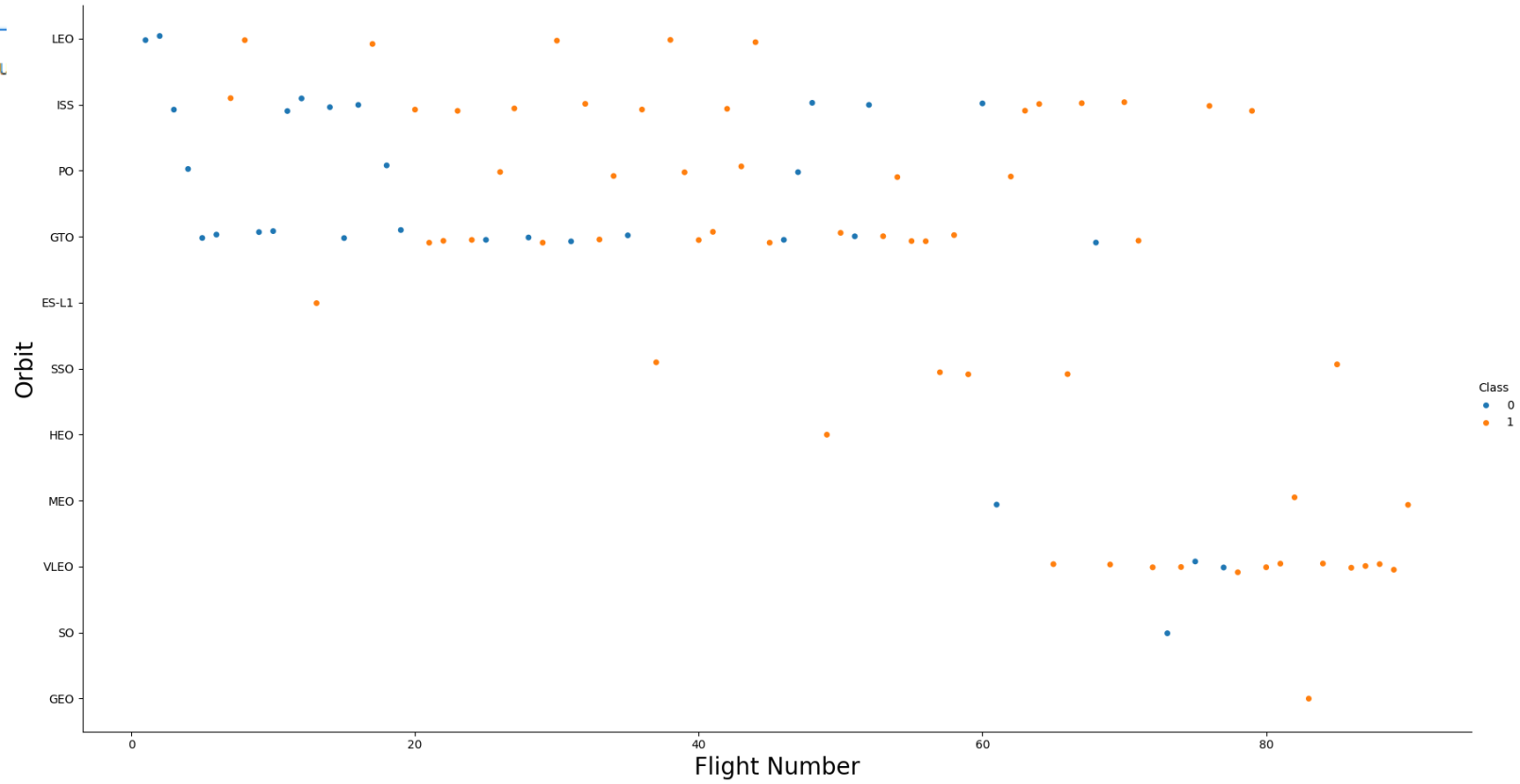
Appendix: EDA Visualization

```
### TASK  4: Visualize the relationship between FlightNumber and Orbit type
```

For each orbit, we want to see if there is any relationship between FlightNumber and Orbit type.

```
# Plot a scatter point chart with x axis to be FlightNumber and y axis to be the Orbit, and hue to be the class value
sns.catplot(y="Orbit", x="FlightNumber", hue="Class", data=df, aspect = 5)
plt.xlabel("Flight Number",fontsize=20)
plt.ylabel("Orbit",fontsize=20)
plt.show()
```

You should see that in the LEO orbit the Success appears related to the nu



Appendix: EDA Visualization

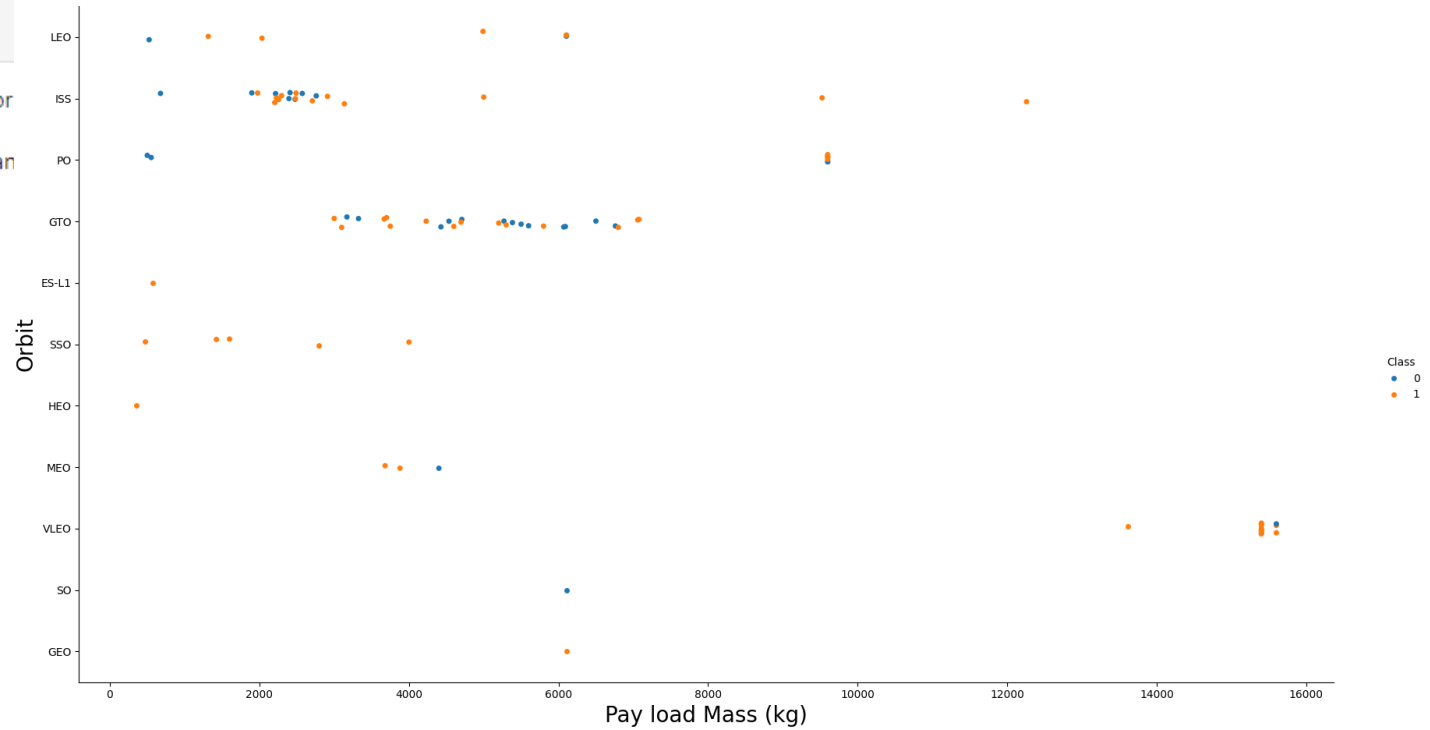
```
### TASK  5: Visualize the relationship between Payload and Orbit type
```

Similarly, we can plot the Payload vs. Orbit scatter point charts to reveal the relationship between Payload and Orbit type

```
# Plot a scatter point chart with x axis to be Payload and y axis to be the Orbit, and hue to be the class value
sns.catplot(y="Orbit", x="PayloadMass", hue="Class", data=df, aspect = 5)
plt.xlabel("Pay load Mass (kg)",fontsize=20)
plt.ylabel("Orbit",fontsize=20)
plt.show()
```

With heavy payloads the successful landing or positive landing rate are more for

However for GTO we cannot distinguish this well as both positive landing rate an



Appendix: EDA Visualization

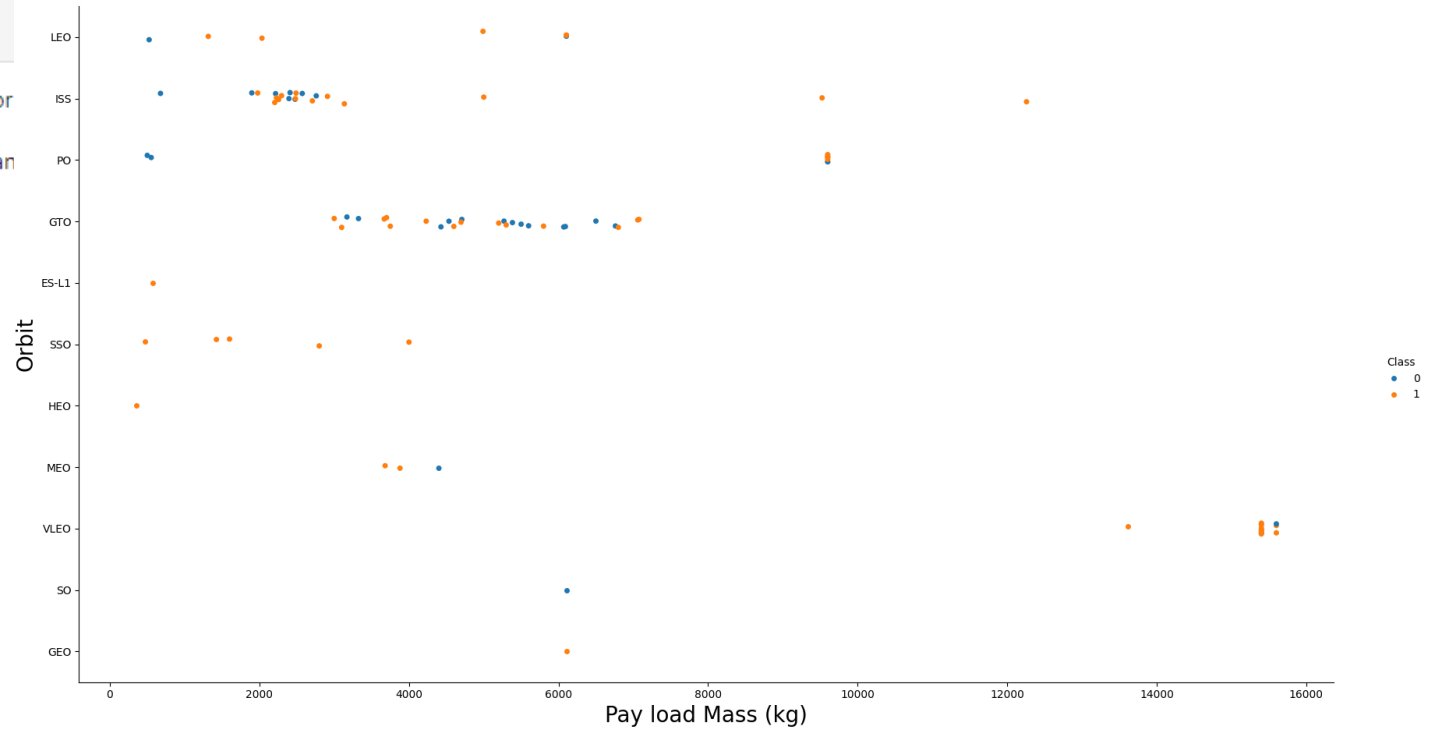
```
### TASK  5: Visualize the relationship between Payload and Orbit type
```

Similarly, we can plot the Payload vs. Orbit scatter point charts to reveal the relationship between Payload and Orbit type

```
# Plot a scatter point chart with x axis to be Payload and y axis to be the Orbit, and hue to be the class value
sns.catplot(y="Orbit", x="PayloadMass", hue="Class", data=df, aspect = 5)
plt.xlabel("Pay load Mass (kg)",fontsize=20)
plt.ylabel("Orbit",fontsize=20)
plt.show()
```

With heavy payloads the successful landing or positive landing rate are more for

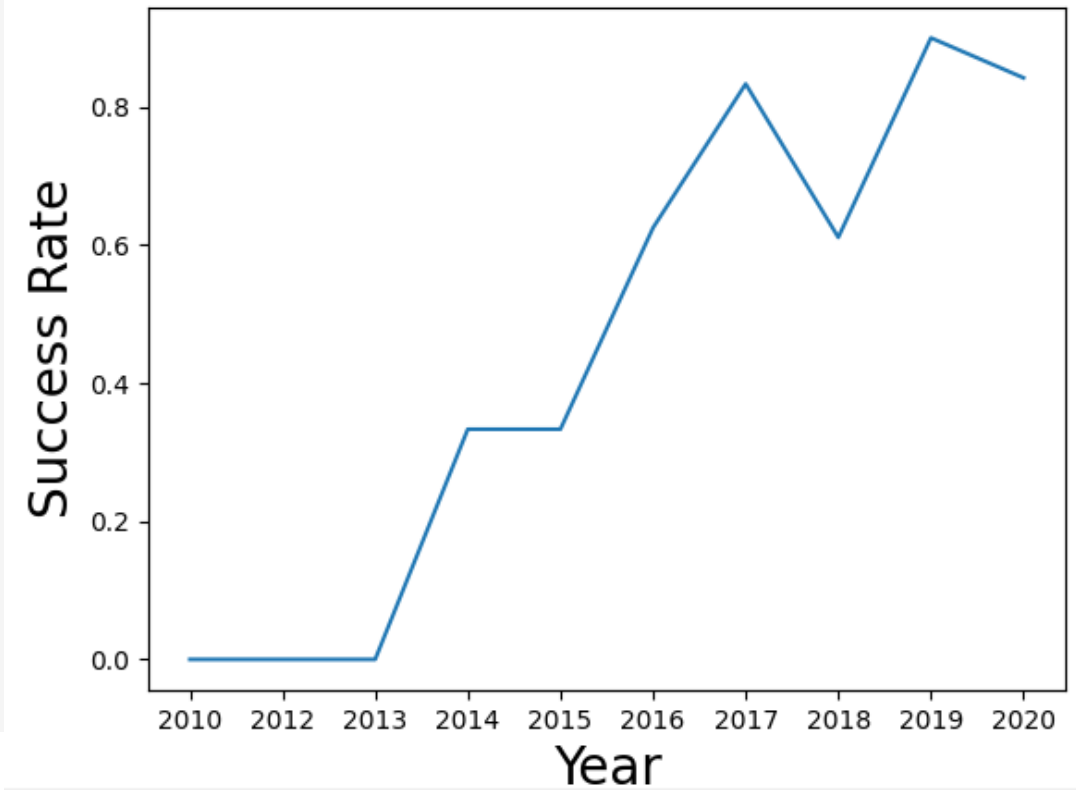
However for GTO we cannot distinguish this well as both positive landing rate an



Appendix: EDA Visualization

TASK 6: Visualize the launch success yearly trend

```
# Plot a line chart with x axis to be the extracted year and y axis to be the success rate
sRatio = []
SuccessRatio = pd.DataFrame()
SuccessRatio['Date'] = (df['Date'].unique())

def computeSuccessRatio():
    for i in SuccessRatio['Date']:
        #print('Date at work :', i)
        success = df.loc[df['Date'] == i, 'Class'].sum()
        #print('Success total : ', success)
        total = df.loc[df['Date'] == i, 'Class'].count()
        #print('Number of observations :', total)
        ratio = success / total
        #print('Ratio : ', ratio)
        sRatio.append(ratio)
    return sRatio

SuccessRatio['SRatio'] = computeSuccessRatio()

sns.lineplot(y="SRatio", x="Date", data=SuccessRatio)
plt.xlabel("Year",fontsize=20)
plt.ylabel("Success Rate",fontsize=20)
plt.show()
```



Appendix: EDA Visualization

```
### TASK  7: Create dummy variables to categorical columns
```

Use the function `get_dummies` and `features` dataframe to apply OneHotEncoder to the column `Orbits`, `LaunchSite`, `LandingPad`, and `Serial`. Assign the value to the variable `features_one_hot`, display the results using the method head. Your result dataframe must include all features including the encoded ones.

```
# HINT: Use get_dummies() function on the categorical columns
tempDF = df[['Orbit', 'LaunchSite', 'LandingPad', 'Serial']]
tempDF.head()
features_one_hot = pd.get_dummies(tempDF, prefix=['Orbit', 'LaunchSite', 'LandingPad', 'Serial'])
features_one_hot[['FlightNumber', 'PayloadMass', 'Flights', 'GridFins', 'Reused', 'Legs', 'Block', 'ReusedCount']] = features[['FlightNumber', 'PayloadMa

features_one_hot.head()
```

	Orbit_ES-L1	Orbit_GEO	Orbit_GTO	Orbit_HEO	Orbit_ISS	Orbit_LEO	Orbit_MEO	Orbit_PO	Orbit_SO	Orbit_SSO	...	Serial_B1060	Serial_B1062	FlightNumber	Payloa
0	False	False	False	False	False	True	False	False	False	False	...	False	False	1	6104.(
1	False	False	False	False	False	True	False	False	False	False	...	False	False	2	525.(
2	False	False	False	False	True	False	False	False	False	False	...	False	False	3	677.(
3	False	False	False	False	False	False	False	True	False	False	...	False	False	4	500.(
4	False	False	True	False	False	False	False	False	False	False	...	False	False	5	3170.(

5 rows × 80 columns

Appendix: EDA Visualization

```
### TASK  8: Cast all numeric columns to `float64`
```

Now that our `features_one_hot` dataframe only contains numbers cast the entire dataframe to variable type `float64`

```
# HINT: use astype function
features_one_hot = features_one_hot.astype(float)
features_one_hot.head()
```

	Orbit_ES-L1	Orbit_GEO	Orbit_GTO	Orbit_HEO	Orbit_ISS	Orbit_LEO	Orbit_MEO	Orbit_PO	Orbit_SO	Orbit_SSO	...	Ser
0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	...	
1	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	...	
2	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...	
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	...	
4	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	

5 rows × 80 columns

Appendix: EDA With SQL

Task 1

Display the names of the unique launch sites in the space mission

```
%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

Appendix: EDA With SQL

Task 2

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

```
%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	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

Task 3

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

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

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

SUM(PAYLOAD_MASS__KG_)
45596

Appendix: EDA With SQL

Task 4

Display average payload mass carried by booster version F9 v1.1

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

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

AVG(PAYLOAD_MASS__KG_)
2928.4

Task 5

List the date when the first succesful landing outcome in ground pad was acheived.

Hint:Use min function

```
%sql SELECT MIN(Date) FROM SPACEXTBL WHERE (Landing_Outcome = "Success (ground pad)")
```

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

MIN(Date)
2015-12-22

Task 6

List the names of the boosters which have success in drone ship and have payload mass greater than 4000 but less than 6000

```
%sql SELECT Booster_Version FROM SPACEXTBL 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

Appendix: EDA With SQL

Task 7

List the total number of successful and failure mission outcomes

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

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

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

Task 8

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

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

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

Booster_Version
F9 B5 B1048.4
F9 B5 B1049.4
F9 B5 B1051.3
F9 B5 B1056.4
F9 B5 B1048.5
F9 B5 B1051.4
F9 B5 B1049.5

Appendix: EDA With SQL

Task 9

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.

Note: SQLLite does not support monthnames. So you need to use substr(Date, 6,2) as month to get the months and substr(Date,0,5)='2015' for year.

```
%sql SELECT SUBSTRING( SPACEXTBL.Date, 6,2), Landing_Outcome, Booster_Version,Launch_Site FROM SPACEXTBL WHERE ((SUBSTRING(Date, 0, 5)=2015) AND(Landing_Outcome LIKE '%drone ship%'))
```

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

SUBSTRING(SPACEXTBL.Date, 6,2)	Landing_Outcome	Booster_Version	Launch_Site

Task 10

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.

```
%sql SELECT Landing_Outcome, COUNT(Landing_Outcome) from SPACEXTBL WHERE (Date > "2010-06-04") GROUP BY Landing_Outcome ORDER BY (COUNT(Landing_Outcome)) DESC
```

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

Landing_Outcome	COUNT(Landing_Outcome)
Success	38
No attempt	21
Success (drone ship)	14
Success (ground pad)	9
Failure (drone ship)	5
Controlled (ocean)	5
Failure	3
Uncontrolled (ocean)	2

Appendix: Locate Launch Sites

Task 1: Mark all launch sites on a map

```
folium.map.Marker(coordinate, icon=DivIcon(icon_size=(20,20),icon_anchor=(0,0), html='<div style="font-size: 12; color:#d35400;"><b>%s
```

```
# 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 name as a popup label
for index, launchSite in spacex_df.iterrows():
    site_map.add_child(
        folium.Circle(
            location = [launchSite['Lat'],launchSite['Long']],
            radius = 50,
            color = '#d35400',
            fill = True
            ))
    site_map.add_child(
        folium.map.Marker(
            location = [launchSite['Lat'],launchSite['Long']],
            icon = DivIcon(
                icon_size = (20,20),
                icon_anchor = (0, 0),
                html='<div style="font-size: 12; color:#d35400;"><b>%s</b></div>' % launchSite['Launch Site'],
                )
            ))
site_map.add_child(circle)
site_map.add_child(marker)
```

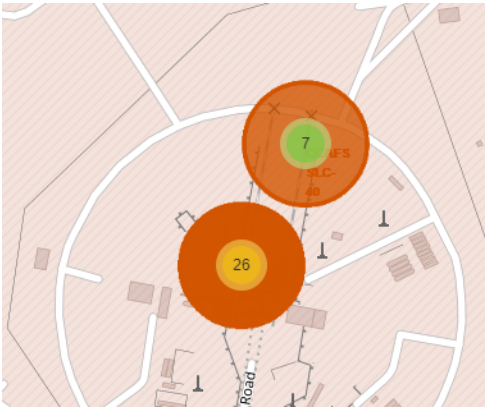
Appendix: Locate Launch Sites

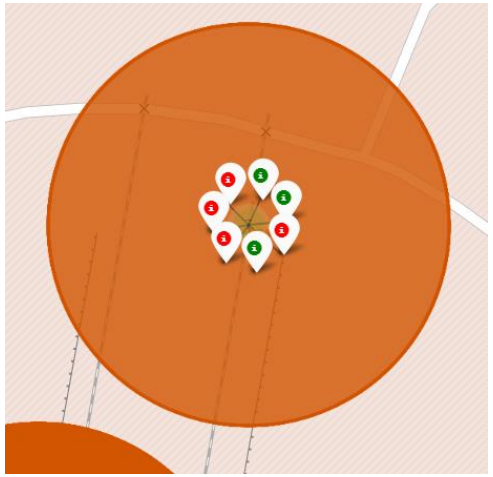
Task 2: Mark the success/failed launches for each site on the map

```
# Apply a function to check the value of `class` column
# If class=1, marker_color value will be green
# If class=0, marker_color value will be red

def getColour(classNbr):
    if classNbr == 1:
        return 'green'
    if classNbr == 0:
        return 'red'

spacex_df['marker_color'] = spacex_df['class'].apply(getColour)
```

TODO: For each launch result in `spacex_df` data frame, add a `folium.Marker` to `marker_cluster`

```
# 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
# and customize the Marker's icon property to indicate if this launch was successed or failed,
# e.g., icon=folium.Icon(color='white', icon_color=row['marker_color']
for index, record in spacex_df.iterrows():
    # TODO: Create and add a Marker cluster to the site map
    # marker = folium.Marker(...)
     marker = folium.Marker(
         location = [launchSite['Lat'],launchSite['Long']],
         icon=folium.Icon(color='white', icon_color=launchSite['marker_color'])
         )
    marker_cluster.add_child(marker)

site_map
```





Appendix: Locate Launch Sites

Task 1: Mark all launch sites on a map

```
folium.map.Marker(coordinate, icon=DivIcon(icon_size=(20,20),icon_anchor=(0,0), html='<div style="font-size: 12; color:#d35400;"><b>%s
```

```
# 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 name as a popup label
for index, launchSite in spacex_df.iterrows():
    site_map.add_child(
        folium.Circle(
            location = [launchSite['Lat'],launchSite['Long']],
            radius = 50,
            color = '#d35400',
            fill = True
            ))
    site_map.add_child(
        folium.map.Marker(
            location = [launchSite['Lat'],launchSite['Long']],
            icon = DivIcon(
                icon_size = (20,20),
                icon_anchor = (0, 0),
                html='<div style="font-size: 12; color:#d35400;"><b>%s</b></div>' % launchSite['Launch Site'],
                )
            ))
site_map.add_child(circle)
site_map.add_child(marker)
```

Appendix: Dash

```
# TASK 1: Add a dropdown list to enable Launch Site selection
# The default select value is for ALL sites
# dcc.Dropdown(id='site-dropdown',...)
dcc.Dropdown(id='site-dropdown',
    options=[
        {'label': 'All Sites', 'value': 'ALL'},
        {'label': 'CCAFS LC-40', 'value': 'CCAFS LC-40'},
        {'label': 'VAFB SLC-4E', 'value': 'VAFB SLC-4E'},
        {'label': 'KSC LC-39A', 'value': 'KSC LC-39A'},
        {'label': 'CCAFS SLC-40', 'value': 'CCAFS SLC-40'},
        ],
    value='ALL',
    placeholder="Select a launch site here",
    searchable=True
    ),
html.Br(),
```

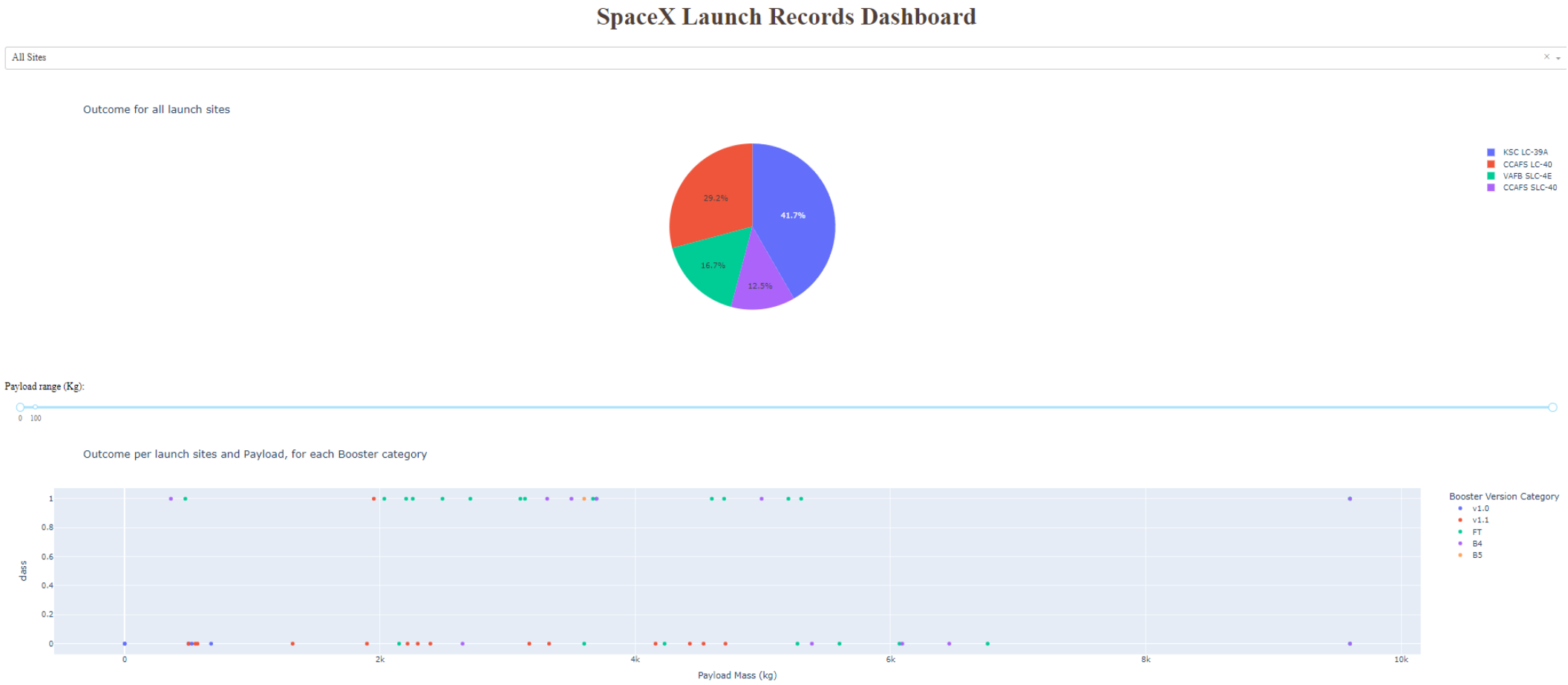
Appendix: Dash

```
# TASK 2: Add a pie chart to show the total successful launches count for all sites
# If a specific launch site was selected, show the Success vs. Failed counts for the site


html.Div(dcc.Graph(id='success-pie-chart')),
html.Br(),

html.P("Payload range (Kg):"),
# TASK 2:
# Add a callback function for `site-dropdown` as input, `success-pie-chart` as output
# Function decorator to specify function input and output

@app.callback(Output(component_id='success-pie-chart', component_property='figure'),
              Input(component_id='site-dropdown', component_property='value'))
def get_pie_chart(entered_site):
    filtered_df = spacex_df
    if entered_site == 'ALL':
        fig = px.pie(filtered_df, values='class',
        names='Launch Site',
        title='Outcome for all launch sites')
        return fig
    else:
        filtered_df = spacex_df.loc[spacex_df['Launch Site'] == entered_site]
        filtered_df = (filtered_df['class'].value_counts())
        print(filtered_df.dtypes)
        fig = px.pie(filtered_df, values='count',
                names='count',
                title=f'Outcome for {entered_site} launch sites')
        return fig
        # return the outcomes piechart for a selected site
```

Appendix: Dash

```
# TASK 3: Add a slider to select payload range
#dcc.RangeSlider(id='payload-slider',...)

dcc.RangeSlider(id = 'payload-slider',
                min=0, max=10000, step=1000,
                marks={ 0: '0',
                        100: '100'},
                value=[0, 10000]),

# TASK 4: Add a scatter chart to show the correlation between payload and launch success
html.Div(dcc.Graph(id='success-payload-scatter-chart')),
])

# TASK 4:
# Add a callback function for `site-dropdown` and `payload-slider` as inputs, `success-payload-scatter-chart` as output
@app.callback(Output(component_id='success-payload-scatter-chart', component_property='figure'),
              [Input(component_id='site-dropdown', component_property='value'), Input(component_id="payload-slider", component_property="value")])
def get_scatter_chart(entered_site, payload):
    print(payload[0])
    print(payload[1])
    filtered_df = spacex_df[(spacex_df['Payload Mass (kg)'] >= payload[0]) & (spacex_df['Payload Mass (kg)'] <= payload[1])]
    if entered_site == 'ALL':
        fig = px.scatter(filtered_df, 'Payload Mass (kg)', 'class',
                         color="Booster Version Category",
                         title='Outcome per launch sites and Payload, for each Booster category')
        return fig
    else:
        filtered_df = filtered_df.loc[spacex_df['Launch Site'] == entered_site]
        fig = px.scatter(filtered_df, 'Payload Mass (kg)', 'class',
                         color="Booster Version Category",
                         title='Outcome per launch sites and Payload, for each Booster category')
        return fig
```

Appendix: Dash



Appendix: Model Evaluation

```
# TASK 1
# Create a NumPy array from the column <code>Class</code> in <code>data</code>, by applying th
# assign it to the variable <code>Y</code>,make sure the output is a  Pandas series (only one

Y = data['Class']#
print(Y)
```

```
C:\Users\didie\AppData\Local
please provide us feedback at https://github.com/pandas-dev/pandas/issues/54466

  import pandas as pd
    FlightNumber        Date BoosterVersion  ...   Longitude   Latitude Class
0              1  2010-06-04       Falcon 9  ...  -80.577366  28.561857     0
1              2  2012-05-22       Falcon 9  ...  -80.577366  28.561857     0
2              3  2013-03-01       Falcon 9  ...  -80.577366  28.561857     0
3              4  2013-09-29       Falcon 9  ... -120.610829  34.632093     0
4              5  2013-12-03       Falcon 9  ...  -80.577366  28.561857     0
..           ...         ...            ...  ...         ...        ...    ...
85            86  2020-09-03       Falcon 9  ...  -80.603956  28.608058     1
86            87  2020-10-06       Falcon 9  ...  -80.603956  28.608058     1
87            88  2020-10-18       Falcon 9  ...  -80.603956  28.608058     1
88            89  2020-10-24       Falcon 9  ...  -80.577366  28.561857     1
89            90  2020-11-05       Falcon 9  ...  -80.577366  28.561857     1
```

Appendix: Model Evaluation

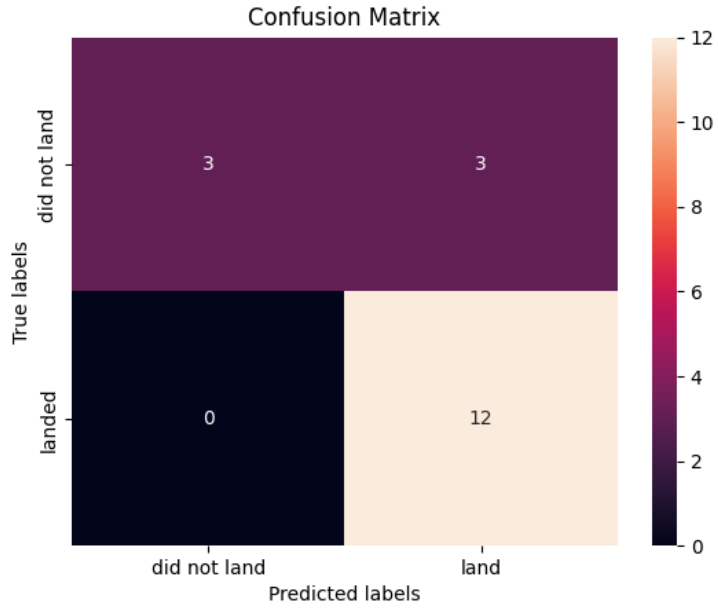
```
# TASK  2
# Standardize te data in <cod>X</code> then reassto the va<code>X</code>sing the transform pr

transformer = preprocessing.StandardScaler()
transformer.fit(X)
X = transformer.transform(X)
print(X)

# We split the data into training and testing data using the function  train_test_split.
# The training data is divided into validation data, a second set used for training  data;
# then the models are trained and hyperparameters are selected using the function GridSearchC

# TASK  3
# Use the function train_test_split to split the data X and Y into training and test data.
# Set the parameter test_size to  0.2 and random_state to 2. The training data and test data

# X_train, X_test, Y_train, Y_test
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.2, random_state = 2)

print('Y Test :', Y_test.count(), '\n', Y_test.shape)
```

```
C:\Users\didie\AppData\Loca
Name: Class, Length: 90, dtype: int64
[[-1.71291154e+00 -1.94814463e-16 -6.53912840e-01 ... -8.35531692e-01
   1.93309133e+00 -1.93309133e+00]
 [-1.67441914e+00 -1.19523159e+00 -6.53912840e-01 ... -8.35531692e-01
   1.93309133e+00 -1.93309133e+00]
 [-1.63592675e+00 -1.16267307e+00 -6.53912840e-01 ... -8.35531692e-01
   1.93309133e+00 -1.93309133e+00]
 ...
 [ 1.63592675e+00  1.99100483e+00  3.49060516e+00 ...  1.19684269e+00
  -5.17306132e-01  5.17306132e-01]
 [ 1.67441914e+00  1.99100483e+00  1.00389436e+00 ...  1.19684269e+00
  -5.17306132e-01  5.17306132e-01]
 [ 1.71291154e+00 -5.19213966e-01 -6.53912840e-01 ... -8.35531692e-01
  -5.17306132e-01  5.17306132e-01]]
Y Test : 18
 (18,)
```

Appendix: Model Evaluation – Log Regression

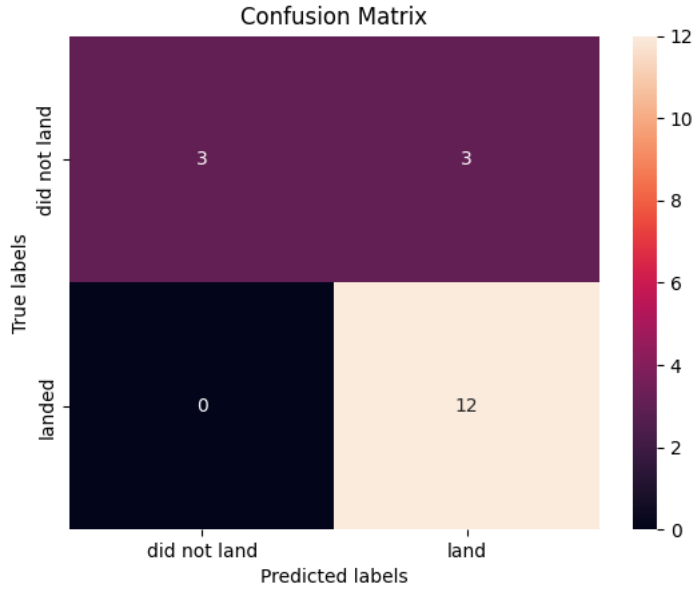
```
# TASK  4
# Create a logistic regression object then create a GridSearchCV object logreg_cv with cv = 10
# Fit the object to find the best parameters from the dictionary parameters.

parameters ={'C':[0.01,0.1,1],
             'penalty':['l2'],
             'solver':['lbfgs']}
parameters ={"C":[0.01,0.1,1],'penalty':['l2'], 'solver':['lbfgs']}     # l1 lasso l2 ridge
lr=LogisticRegression()
logreg_cv = GridSearchCV(lr, parameters)
logreg_cv.fit(X_train, Y_train)

# We output the GridSearchCV object for logistic regression.
# We display the best parameters using the data attribute best_params\_ and
# the accuracy on the validation data using the data attribute best_score\_.
print("LogRegression: tuned hpyerparameters :(best parameters) ",logreg_cv.best_params_)
print("Log Regression accuracy :",logreg_cv.best_score_)

# TASK  5
# Calculate the accuracy on the test data using the method score:
print('Log Regression Score : ',logreg_cv.score(X_test, Y_test))

# Lets look at the confusion matrix:
yhat=logreg_cv.predict(X_test)
plot_confusion_matrix(Y_test,yhat)

# Examining the confusion matrix, we see that logistic regression can distinguish between the
# We see that the major problem is false positives.
```

```
C:\Users\didie\AppData\Local
LogRegression: tuned hpyerparameters :(best parameters)  {'C': 0.1, 'penalty': 'l2', 'solver': 'lbfgs'}
Log Regression accuracy : 0.8342857142857143
Log Regression Score :  0.8333333333333334
```



Appendix: Model Evaluation - SVM

```
# TASK  6
# Create a support vector machine object then
# create a GridSearchCV object svm_cv with cv - 10.
# Fit the object to find the best parameters from the dictionary parameters.
parameters = {'kernel':('linear', 'rbf','poly','rbf', 'sigmoid'),
              'C': np.logspace(-3, 3, 5),
              'gamma':np.logspace(-3, 3, 5)}
svm = SVC()
svm_cv = GridSearchCV(svm, parameters)
svm_cv.fit(X_train, Y_train)

print("SVM tuned hyperparameters :(best parameters) ",svm_cv.best_params_)
print("SVM accuracy :",svm_cv.best_score_)
yhat=svm_cv.predict(X_test)
#plot_confusion_matrix(Y_test,yhat)

# TASK  7
# Calculate the accuracy on the test data using the method score:
print('SVM Score : ',svm_cv.score(X_test, Y_test))
```

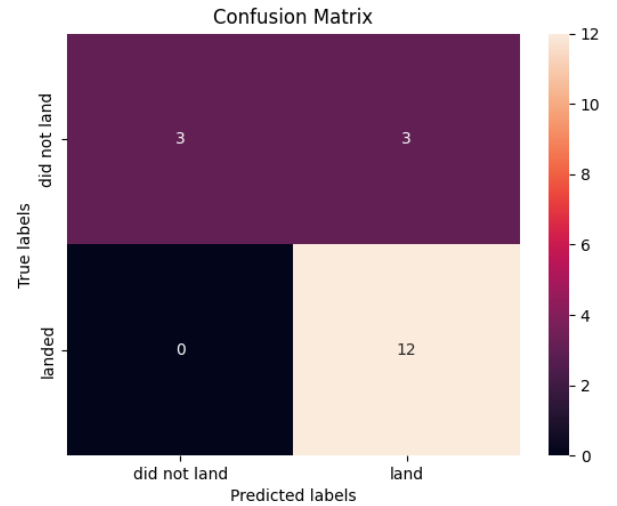



```
C:\Users\didie\AppData\Loca
SVM tuned hyperparameters :(best parameters)  {'C': 0.03162277660168379, 'gamma': 0.001, 'kernel': 'linear'}
SVM accuracy : 0.8342857142857142
SVM Score :  0.8333333333333334
```

Appendix: Model Evaluation – KNN and best

```
## Task 10
# Create a k nearest neighbors object then create a GridSearchCV object knn_cv with cv = 10.
# Fit the object to find the best parameters from the dictionary parameters.
parameters = {'n_neighbors': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
              'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute'],
              'p': [1,2]}

KNN = KNeighborsClassifier()

knn_cv = GridSearchCV(KNN, parameters)
knn_cv.fit(X_train, Y_train)

print("tuned hyperparameters :(best parameters) ",knn_cv.best_params_)
print("accuracy :",knn_cv.best_score_)

## TASK 11
print('KNN Score : ',knn_cv.score(X_test, Y_test))
yhat=knn_cv.predict(X_test)
plot_confusion_matrix(Y_test,yhat)

results = {'Log Regression Score' : logreg_cv.score(X_test, Y_test),
           'SVM Score ' : svm_cv.score(X_test, Y_test),
           'Tree Score ': tree_cv.score(X_test, Y_test),
           'KNN Score ': knn_cv.score(X_test, Y_test)
           }
max_value = max(results, key = results.get)
print(max_value)
```

```
C:\Users\didie\AppData\Local
tuned hyperparameters :(best parameters)  {'algorithm': 'auto', 'n_neighbors': 8, 'p': 1}
accuracy : 0.8609523809523811
KNN Score :  0.8333333333333334
Log Regression Score
```



Thank you!