IBM Developer
SKILLS NETWORK

Winning Space Race with Data Science

Syaheed B Jabar
4th September 2023

Outline

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

Executive Summary

- Summary of methodologies
  - Data collection via Space X API and web scraping
  - Various exploratory data analysis and data visualization to gather insights
  - Predictive analysis of launch success using machine learning

- Summary of all results
  - KSC LC-39A seem to be the most promising Space X launch site (highest success rate)
    - KSC LC-39A, CCAFS SLC-40, CCAFS LC-40 and VAFB SLC-4E are all **near the coast and near the equator**
  - Orbits with the best success rates are GEO, HEO, SSO, ES-L1
  - Low weighted payloads have a better success rate than the heavy weighted payloads (particularly with FT boosters)
  - FT boosters seem reliable, and particularly has a high success rate with drone ship rescues
  - With the available data, machine learning models can predict future outcomes with about 80% - 90% accuracy
    - Decision Trees seems to be the best model (highest accuracy, least false positives)
    - Between Logistic Regression, kNN or SVM, not much difference between model performance

Introduction

- Project background and context
  - Space X saves cost on its rocket launches because the first stage (e.g., of the Falcon 9) can be detached, make a successful landing and then be re-used
  - We want to know how to maximize the likelihood of this landing procedure
- Problems you want to find answers
  - What are the main characteristics of a successful or failed landing ?
    - Such as launch site, orbit type, payload weights, boosters used
  - How accurately can we predict a failed launch or a successful one given the above characteristics?

Section 1

Methodology

Methodology

Executive Summary

- Data collection methodology:
  - Wikipedia (via scraping)
  - Using Space X Rest API
- Perform data wrangling
  - Filter the data, remove/impute missing values/ one hot encoding to prep data for predictive modelling
- Perform exploratory data analysis (EDA) using visualization and SQL
- Perform interactive visual analytics using Folium and Plotly Dash
- Perform predictive analysis using classification models
  - Build, tune and evaluate models

Data Collection

- Data collection was done in two ways. This was because each offered some unique information not offered by the other

1) Space X Rest API (https://api.spacexdata.com/v4/)

Request information from API → JSON file from API → Pandas data.frame from JSON → Save / Export as .csv

Data Columns:

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

2) Webscrape Wikipedia (URL link)

Request data from Wikipedia URL → Fill a dictionary using data from Soup using tags → Pandas data.frame from dictionary → Save / Export as .csv

Data Columns:

Flight Number, Launch site, Payload, PayloadMass, Orbit, Customer, Launch outcome, Version Booster, Booster landing, Date, Time

Data Collection – SpaceX API

Jupyter notebook with code for API calls can be found at:

https://github.com/syaheed/public-code/blob/master/IBM%20Data%20Science/Capstone/data-collection-api.ipynb

- To get data form the SpaceX API:

```
response = requests.get(spacex_url)
```

1) Get response from API

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

2) Save response to JSON and normalize

```
# Lets take a subset of our dataframe keeping only the features we want and the flight number, and date_utc.
data = data[['rocket', 'payloads', 'launchpad', 'cores', 'flight_number', 'date_utc']]

# We will remove rows with multiple cores because those are falcon rockets with 2 extra rocket boosters and rows that have
data = data[data['cores'].map(len)==1]
data = data[data['payloads'].map(len)==1]

# Since payloads and cores are lists of size 1 we will also extract the single value in the list and replace the feature.
data['cores'] = data['cores'].map(lambda x : x[0])
data['payloads'] = data['payloads'].map(lambda x : x[0])

# We also want to convert the date_utc to a datetime datatype and then extracting the date leaving the time
data['date'] = pd.to_datetime(data['date_utc']).dt.date

# Using the date we will restrict the dates of the launches
data = data[data['date'] <= datetime.date(2020, 11, 13)]
```

3) Transform the data

```
# Call getBoosterVersion
getBoosterVersion(data)

# Call getLaunchSite
getLaunchSite(data)

# Call getPayloadData
getPayloadData(data)

# Call getCoreData
getCoreData(data)
```

4) Use prepared functions to obtain additional data

```
launch_dict = {'FlightNumber': list(data['flight_number']),
'Date': list(data['date']),
'BoosterVersion':BoosterVersion,
'PayloadMass':PayloadMass,
'Orbit':Orbit,
'LaunchSite':LaunchSite,
'Outcome':Outcome,
'Flights':Flights,
'GridFins':GridFins,
'Reused':Reused,
'Legs':Legs,
'LandingPad':LandingPad,
'Block':Block,
'ReusedCount':ReusedCount,
'Serial':Serial,
'Longitude': Longitude,
'Latitude': Latitude}
```

5) Prepare empty dictionary with the expected columns

```
# Create a data from launch_dict
data = pd.DataFrame({key:pd.Series(value) for key, value in launch_dict.items()})
```

6) Dictionary to pandas dataframe

```
# Hint data['BoosterVersion']!='Falcon 1'
data_falcon9 = data[data['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))
```

7) Remove unwanted data, re-index

```
# Calculate the mean value of PayloadMass column
mean_payload_mass = data_falcon9['PayloadMass'].mean()

# Replace the np.nan values with its mean value
data_falcon9['PayloadMass'] = data_falcon9['PayloadMass'].replace(np.nan, mean_payload_mass)
data_falcon9.isnull().sum()
```

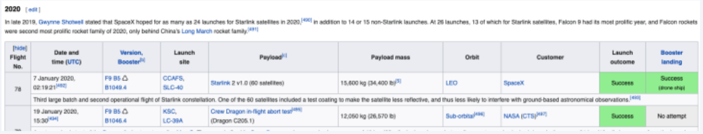
8) Data imputation using means

```
data_falcon9.to_csv('dataset_part_1.csv', index=False)
```

9) Data export

Data Collection - Scraping

- We want to get the data from tables in the URL: ["https://en.wikipedia.org/w/index.php?title=List_of_Falcon_9_and_Falcon_Heavy_launches&oldid=1027686922"](https://en.wikipedia.org/w/index.php?title=List_of_Falcon_9_and_Falcon_Heavy_launches&oldid=1027686922)



1.) Request data from URL

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

2) Create a Beautiful Soup object

```
# Use BeautifulSoup() to create a BeautifulSoup object from a response text content
soup = BeautifulSoup(response.text, "html5lib")
```

3) Find tables in soup obj

```
# 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.findAll('table')
```

4) Select table with relevant data

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

5) Each column has a table header element <th>. Use to get all column names

```
column_names = []

# Apply find_all() function with `th` element on first_launch_table
# Iterate each th element and apply the provided extract_column_from_header() to get a column name
# Append the Non-empty column name (`if name is not None and len(name) > 0`) into a list called column_names

for th in first_launch_table.find_all('th'):
    name = extract_column_from_header(th)
    if name is not None and len(name) > 0 :
        column_names.append(name)
```

6) Create an empty dictionary with the columns that we want to fill in

```
launch_dict= dict.fromkeys(column_names)

# Remove an irrelvant column
del launch_dict['Date and time ( )']

# Let's initial the launch_dict with each value to be an empty list
launch_dict['Flight No.'] = []
launch_dict['Launch site'] = []
launch_dict['Payload'] = []
launch_dict['Payload mass'] = []
launch_dict['Orbit'] = []
launch_dict['Customer'] = []
launch_dict['Launch outcome'] = []
# Added some new columns
launch_dict['Version Booster']=[]
launch_dict['Booster landing']=[]
launch_dict['Date']=[]
launch_dict['Time']=[]
```

7) Iterate though each row, fill in each column per row

```
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)
```

8) Do this for all columns, see code below

9) Convert dictionary to a pandas data.frame

```
df=pd.DataFrame(launch_dict)
```

10) Export

```
df.to_csv('spacex_web_scraped.csv', index=False)
```

- Jupyter notebook with code for scraping can be found at:
  - https://github.com/syaheed/public-code/blob/master/IBM%20Data%20Science/Capstone/data-webscraping.ipynb

Data Wrangling

- In dataset_part1.csv, there are several cases where the booster did not land successfully. In the "Outcomes" column:
  - Strings: 'True Ocean', 'True RTLS', 'True ASDS' means the mission has been successful.
  - Strings: 'False ASDS', 'False Ocean', 'False RTLS', 'None ASDS', 'None None' means the mission was a failure.
- To help with later analysis, we need to transform the above string variables into categorical variables (0 or 1)

In [9]:

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

```
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
```

In [10]:

```
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
```

These are the distinct cases in 'outcome'

In [11]:

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

Out[11]: {'False ASDS', 'False Ocean', 'False RTLS', 'None ASDS', 'None None'}

These are the 'bad outcomes'

We assign "Class" to 0 to the 'bad outcomes', '1' otherwise

```
landing_class = []
for key,value in df["Outcome"].items():
    if value in bad_outcomes:
        landing_class.append(0)
    else:
        landing_class.append(1)
```

In [13]:

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

Out[13]:

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

Append new column to imported data.frame

```
df.to_csv("dataset_part_2.csv", index=False)
```

- Export the data with the new class column for later use
- Link to dataset_part_2.csv

- Jupyter notebook with python code for data wrangling can be found at:
  - https://github.com/syaheed/public-code/blob/master/IBM%20Data%20Science/Capstone/data-wrangling.ipynb

EDA with Data Visualization

- Data visualization was done on the SpaceX dataset:
- This was done to be able to begin developing insights abut the data and the relation between the different variables that could relate to a successful/failed launch.
    - The appropriate plot to use depends on the nature of the data and what insights one wishes to gain. E.g., bar plots are good to look at summary stats (e.g., means), scatter plots and lines are good to look at trends across time (or flight number in this case)
- Charts plotted:
    - Flight Number vs. Payload Mass (scatterplot / catplot)
    - Flight Number vs. Launch Site (scatterplot / catplot)
    - Payload Mass vs. Launch Site (scatterplot / catplot)
    - Orbit Type vs. Success Rate (bar chart)
    - Flight Number vs. Orbit Type (scatterplot / catplot)
    - Payload Mass vs Orbit Type (scatterplot / catplot)
    - Success Rate Yearly Trend (line chart)

- Jupyter notebook with EDA/SQL analysis can be found at:
    - https://github.com/syaheed/public-code/blob/master/IBM%20Data%20Science/Capstone/eda-sql.ipynb

EDA with SQL

- SQL queries were done on to gather and understand data from [SpaceX dataset]:
  - Display the names of the unique launch sites in the space mission
  - Display 5 records where launch sites begin with the string 'CCA'
  - Display the total payload mass carried by boosters launched by NASA (CRS).
  - Display average payload mass carried by booster version F9 v1.1
  - 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 (using a subquery)
  - 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 successful landing outcomes between the date 2010-06-04 and 2017-03-20, in descending order

- Jupyter notebook with EDA/SQL analysis can be found at:
  - https://github.com/syaheed/public-code/blob/master/IBM%20Data%20Science/Capstone/eda-sql.ipynb

Build an Interactive Map with Folium

- Using the Folium python library, we can map markers and objects like lines to locations on interest onto an interactive map. These can help us gather more insights into identifying the features that what make a good launch site
  - Created a Folium map object centered on NASA Johnson Space Center at Houson, Texas (blue circle, with name)
  - Marked each of the four launch sites (red circle, with name)
    - Interactive markers to show proportion of successful (green) and unsuccessful landings (red) for site.
  - Used a sample site to show distance between launch site to key locations (railway, highway, coast, city) and plot a line between them

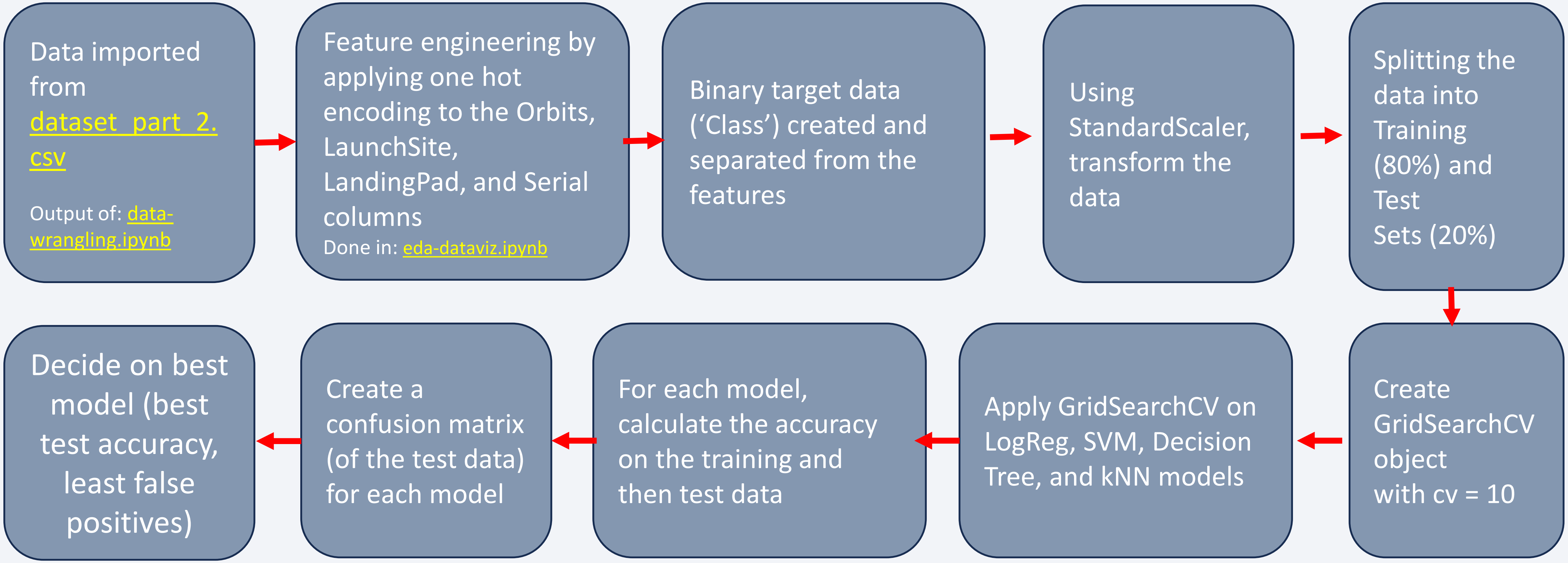
- Jupyter notebook with interactive maps using Folium analysis can be found at:
  - https://github.com/syaheed/public-code/blob/master/IBM%20Data%20Science/Capstone/launch_site_location.ipynb

Build a Dashboard with Plotly Dash

- Dashboards provide an opportunity for the user to focus in on details they want to look at by introducing interactivity
- Using Plotly/Dash (python libraries), created an app with:
  - User choices :
    - A dropdown list for the user to choose the specific launch site to look at (or all sites)
    - A slider so users can restrict the data to some payload weight range
  - Visualizations that update based on the user choices above :
    - Pie chart showing Success Launches (All Sites/Certain Site)
    - Scatter plot of payload mass, success/failure and booster versions:

- Dashboard code can be found at:
  - https://github.com/syaheed/public-code/blob/master/IBM%20Data%20Science/Capstone/spacex_dash_app.py

Predictive Analysis (Classification)



- Machine learning code and analysis can be found at:
  - https://github.com/syaheed/public-code/blob/master/IBM%20Data%20Science/Capstone/machine_learning.ipynb

Results

- Exploratory data analysis (EDA) results
  - Insights from EDA based on visualizations (Section 2)
  - Insights from EDA based on simple SQL queries (Section 2)
- Interactive analytics demo in screenshots
  - Launch site proximity analysis using Folium maps (Section3)
  - Dashboard analysis using Dash/Plotly (Section3)
- Predictive analysis results (Section 5)

Section 2

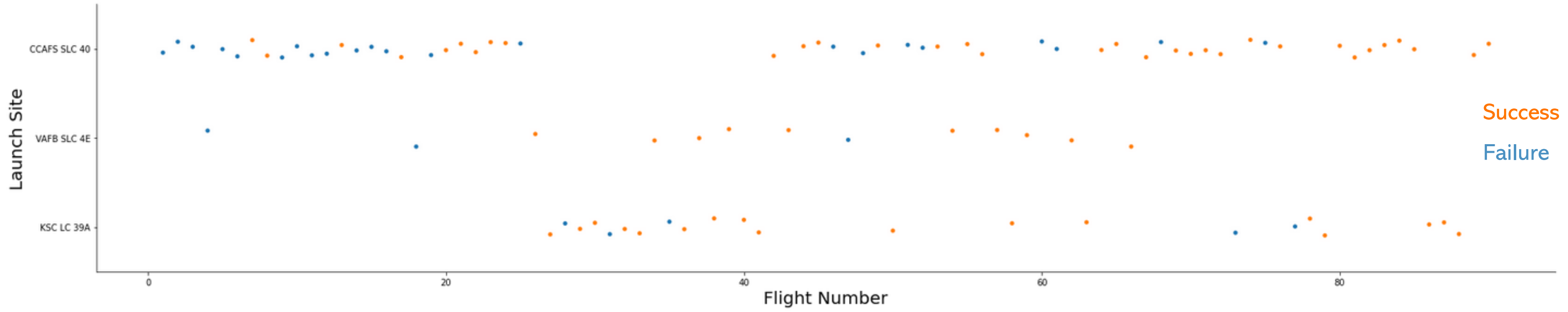
Insights drawn from EDA

Flight Number vs. Launch Site

[Click Here for the code]() for EDA

- Analysis done using dataset_part_2.csv processed previously, loaded into a Pandas data.frame in Python, and a catplot visualized, where each dot represents either a successful (orange) launch or a failures (blue).
- Initial flights were from CCAFS SLC 40, which started off mostly failing, but then began having a string of successful launches around flight 20
- Launches then shifted to KSC LC 39A, which has a reasonable proportion of successes.
- After 40-ish flights, majority of launches shifted to CCAFS SLC 40 again, now with more successes than before.

In [4]:
```
# 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="LaunchSite", x="FlightNumber", hue="Class", data=df, aspect = 5)
plt.xlabel("Flight Number",fontsize=20)
plt.ylabel("Launch Site",fontsize=20)
plt.show()
```

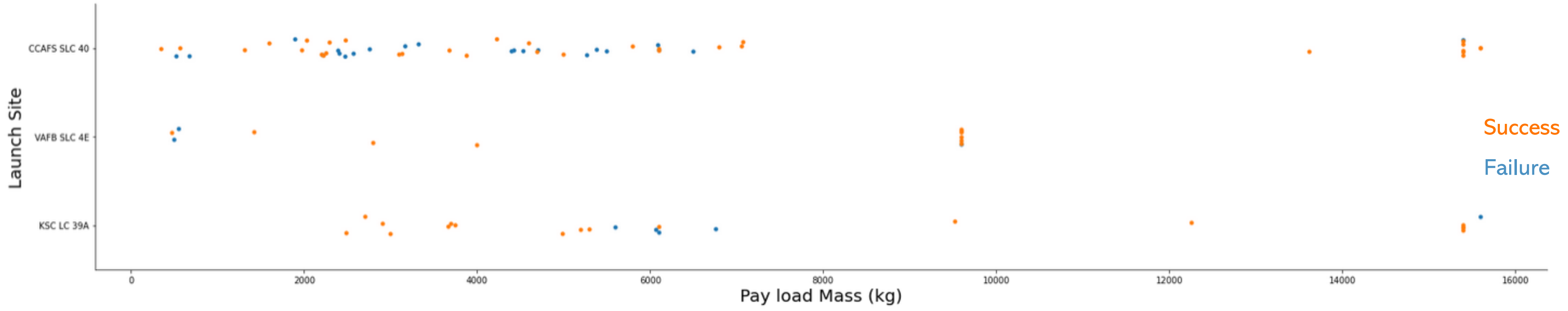


Payload vs. Launch Site

[Click Here for the code](#) for EDA

- A similar catplot was done for payload vs launch site
- Very high pay loads (>9000 kgs) seem to have a high rate of success, but there is not much data to say this concretely
  - For example, VAFB-SLC launch site there are no rockets launched for payload mass greater than 10000
- Lower pay loads (<4000 kgs) tended to have more success than mid-weight ones (between 4000-9000 kgs), particularly so for launch site KSC LC 39A (we will revisit this in Section 4.
- Successes across different pay loads is more haphazard for CCAFs SLC 40

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



Success Rate vs. Orbit Type

Click Here for the code for EDA

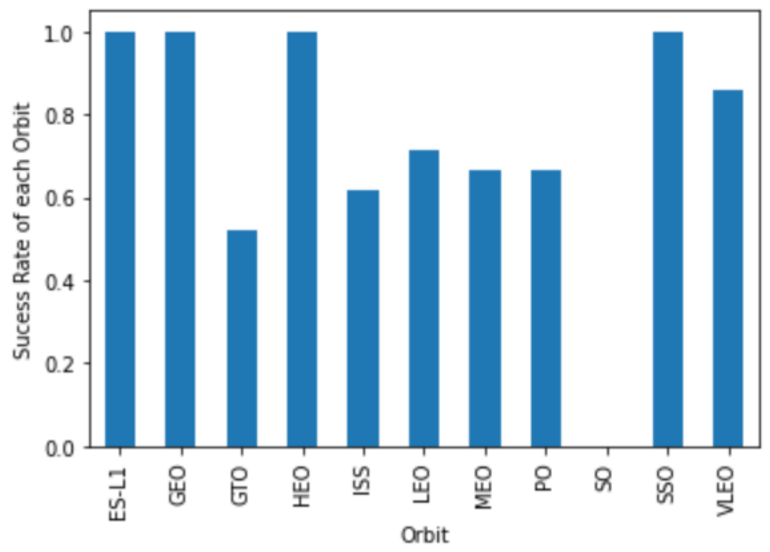
- Took the success/failure data for orbit type and calculated the **mean** for each

- 4 orbit types have a **100% success rate**
  - **Geostationary orbit (GEO)**
  - **Highly elliptical orbit (HEO)**
  - **Sun-synchronous orbit (SSO)**
  - **Earth-Sun-Lagrange point 1 (ES-L1)**

- Recommended to use those orbits in the future
  - Keep in mind though, there are unequal numbers of attempts for each orbit type

```python
# HINT use groupby method on Orbit column and get the mean of Class column
pl = df.groupby('Orbit')['Class'].mean()
ax = pl.plot(kind='bar')
ax.set_xlabel("Orbit")
ax.set_ylabel("Sucess Rate of each Orbit")
```

```
Text(0, 0.5, 'Sucess Rate of each Orbit')
```

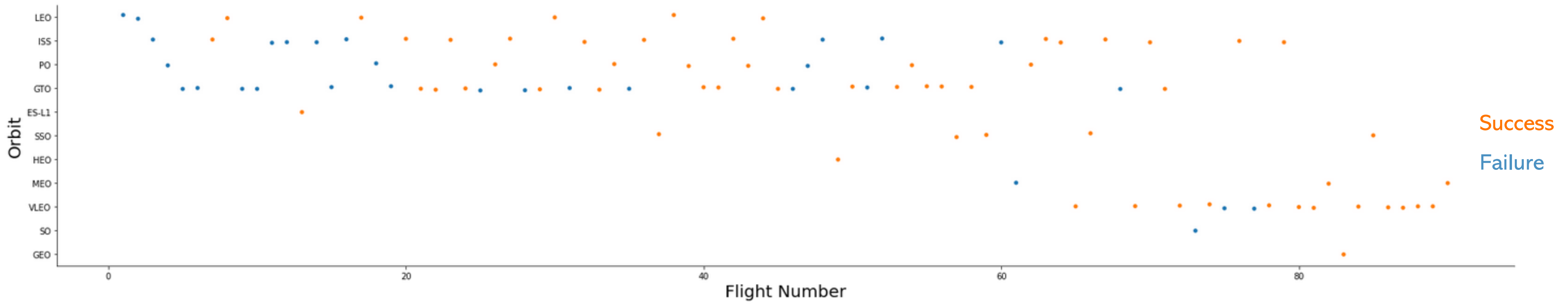


Flight Number vs. Orbit Type

[Click Here for the code](#) for EDA

- A catplot done again, this type on flight number vs. orbit type.
- We see a shift from LEO/ISS/PO/GTO to MEO/VLEO, SO, GEO orbits, around flight number 60
- Seems like launch success rate also increases over flight numbers, presumably reflecting better choices for launch conditions
    - In the LEO orbit the Success appears related to the number of flights; on the other hand, there seems to be no relationship between flight number when in GTO orbit

```python
# 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()
```

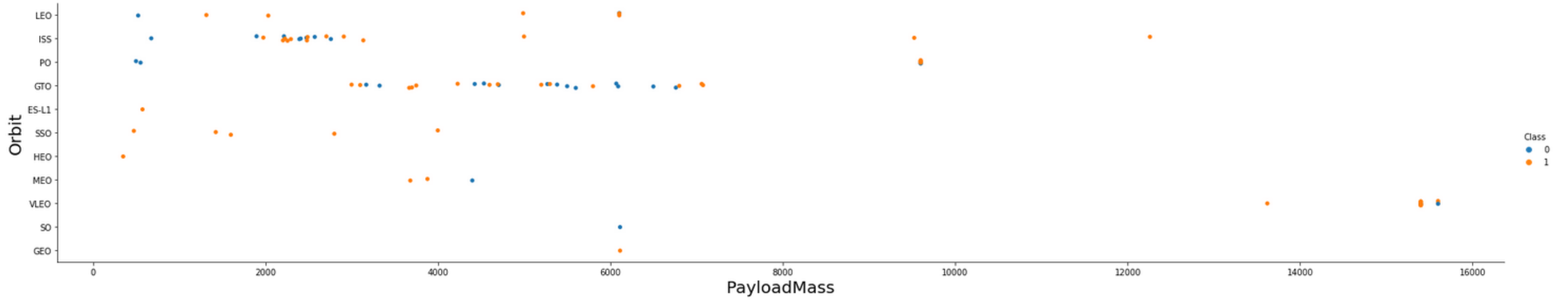


Payload vs. Orbit Type

Click Here for the code for EDA

- Another catplot, this time payload vs orbit type
- Generally, there are more launches with lower payload (< 7000 kgs) across different orbits. With payloads above 4000kgs the successful landing or positive landing rate are more for Polar, LEO and ISS.
  - However for GTO we cannot distinguish this well as there are both success and failures in this payload range

```
# 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("PayloadMass",fontsize=20)
plt.ylabel("Orbit",fontsize=20)
plt.show()
```



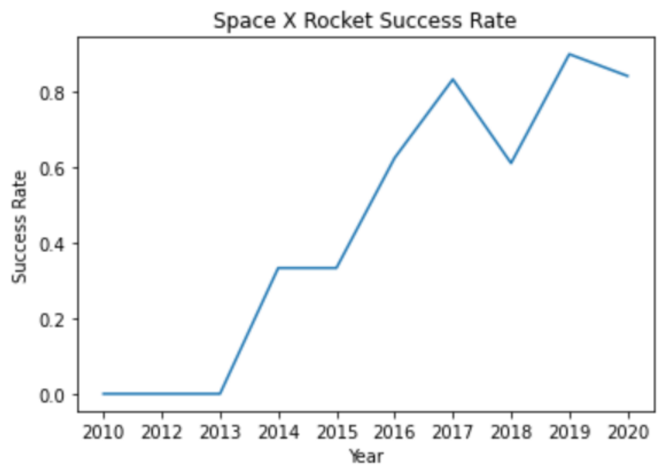
Launch Success Yearly Trend

Click Here for the code for EDA

- The year was extracted from the date data
- Launches were grouped by year and the mean across success/failures was taken
- **Success rate has generally increased**
    - From 0% in 2013 to a high of 90% in 2019
    - There was a dip in success rate in 2018 and a minor dip in 2020

```python
# A function to Extract years from the date
year=[]
def Extract_year(date):
    for i in df["Date"]:
        year.append(i.split("-")[0])
    return year
```

```python
# Plot a line chart with x axis to be the extracted year and y axis to be the success rate
df['Year'] = Extract_year(df["Date"])
df_groupby_year = df.groupby("Year",as_index=False)["Class"].mean()
sns.lineplot(data = df_groupby_year, x="Year", y="Class")
plt.xlabel("Year")
plt.title('Space X Rocket Success Rate')
plt.ylabel("Success Rate")
plt.show()
```



All Launch Site Names

Click Here for the code for SQL Codes

- Data was:
  - Taken from the SpaceX dataset
  - Imported into Python (Pandas dataframe) and converted to a SQL table using sqlite
  - Blank rows were removed prior to analysis

```
import pandas as pd
df = pd.read_csv("https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/labs/module_2/data/Spacex.csv")
df.to_sql("SPACEXTBL", con, if_exists='replace', index=False,method="multi")
```

```
%sql create table SPACEXTABLE as select * from SPACEXTBL where Date is not null
```

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

- Launch site names from SpaceX:

```
[87]: %sql SELECT DISTINCT LAUNCH_SITE FROM SPACEXTBL;
```

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

[87]:

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

- There are four sites in the dataset

Launch Site Names Begin with 'CCA'

[Click Here for the code](#) for SQL Codes

- Here are 5 records where launch sites begin with `CCA`
  - Note that there are actually 60 matching records
  - “Limit 5” used to just find five matching records
  - Pattern matching ‘%CCA%’ used so that the Launch Sites with strings occurring after ‘CCA’ are still found

```
%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
2010-04-06	18:45:00	F9 v1.0 B0003	CCAFS LC-40	Dragon Spacecraft Qualification Unit	0	LEO	SpaceX	Success	Failure (parachute)
2010-08-12	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	07:44:00	F9 v1.0 B0005	CCAFS LC-40	Dragon demo flight C2	525	LEO (ISS)	NASA (COTS)	Success	No attempt
2012-08-10	00:35:00	F9 v1.0 B0006	CCAFS LC-40	SpaceX CRS-1	500	LEO (ISS)	NASA (CRS)	Success	No attempt
2013-01-03	15:10:00	F9 v1.0 B0007	CCAFS LC-40	SpaceX CRS-2	677	LEO (ISS)	NASA (CRS)	Success	No attempt

Total Payload Mass

Click Here for the code for SQL Codes

- According to the dataset, NASA has taken a sum total of 45596 KG (across 20 launches)
  - Note: Lab Exercise specified 'NASA (CRS)'

```
%sql SELECT SUM(PAYLOAD_MASS__KG_) FROM SPACEXTBL WHERE CUSTOMER = 'NASA (CRS)';

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

SUM(PAYLOAD_MASS__KG_)
45596

- If including any CUSTOMER that has 'NASA' in it (e.g., NASA (COTS) NRO), they have taken a sum of 107010 KG (across 32 launches)

```
%sql SELECT SUM(PAYLOAD_MASS__KG_) FROM SPACEXTBL WHERE CUSTOMER LIKE '%NASA%';

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

SUM(PAYLOAD_MASS__KG_)
107010

Average Payload Mass by F9 v1.1

Click Here for the code for SQL Codes

- Average payload mass carried by booster version F9 v1.1 is 2535 kg
- Used pattern matching “%F9 v1.1%” to get the different variants
  - e.g., B1011, B1010, etc.

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

```
%sql SELECT AVG(PAYLOAD_MASS__KG_) FROM SPACEXTBL WHERE BOOSTER_VERSION LIKE '%F9 v1.1%';
```

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

AVG(PAYLOAD_MASS__KG_)
2534.6666666666665

First Successful Ground Landing Date

Click Here for the code for SQL Codes

- First successful landing outcome on ground pad occurred on **22nd Dec 2015**
- Selected Landing_Outcome specifically for "Success (ground pad case)"

```
%sql SELECT MIN(Date) FROM SPACEXTBL WHERE Landing_Outcome = "Success (ground pad)";
 * sqlite:///my_data1.db
Done.
```

MIN(Date)
2015-12-22

Successful Drone Ship Landing with Payload between 4000 and 6000

Click Here for the code for SQL Codes

- These are the names of boosters which have successfully landed on drone ship and had payload mass greater than 4000 but less than 6000 kgs
  - "Success (drone ship)" was specifically used to avoid cases like land-based landings
- Note that all drone ship success cases used F9 FT boosters

```
[29]: %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.
```

[29]:

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

Total Number of Successful and Failure Mission Outcomes

Click Here for the code for SQL Codes

- There are 100 successful mission outcomes and 1 failure
  - note that these are not LANDING outcomes

- Pattern matching used on MISSON_OUTCOME because of cases like
  - Success (payload status unclear)
  - Failure (in flight)

```
[83]: %sql SELECT COUNT(MISSION_OUTCOME) AS Successful FROM SPACEXTBL WHERE MISSION_OUTCOME LIKE '%Success%'

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

[83]:

Successful
100

```
[82]: %sql SELECT COUNT(MISSION_OUTCOME) AS Failures FROM SPACEXTBL WHERE MISSION_OUTCOME LIKE '%Failure%'

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

[82]:

Failures
1

Boosters Carried Maximum Payload

Click Here for the code for SQL Codes

- The following are boosters which have carried the maximum payload mass
  - Maximum payload mass is 15600 KG
  - Note that they are all F9 B5 boosters

```
[81]: %sql SELECT DISTINCT BOOSTER_VERSION, PAYLOAD_MASS__KG_ FROM SPACEXTBL WHERE PAYLOAD_MASS__KG_ = (SELECT max(PAYLOAD_MASS__KG_) FROM SPACEXTBL)

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

[81]:

Booster_Version	PAYLOAD_MASS__KG_
F9 B5 B1048.4	15600
F9 B5 B1049.4	15600
F9 B5 B1051.3	15600
F9 B5 B1056.4	15600
F9 B5 B1048.5	15600
F9 B5 B1051.4	15600
F9 B5 B1049.5	15600
F9 B5 B1060.2	15600
F9 B5 B1058.3	15600
F9 B5 B1051.6	15600
F9 B5 B1060.3	15600
F9 B5 B1049.7	15600

2015 Launch Records

Click Here for the code for SQL Codes

- The following are the failed landing_outcomes in drone ship, their booster versions, and launch site names in year 2015

```
# substring for year has to be substr(Date,1,4) for me (year month day), substr(DATE, 6, 2) for month

%sql SELECT substr(DATE, 6, 2) AS MONTH, LANDING_OUTCOME, BOOSTER_VERSION, LAUNCH_SITE FROM SPACEXTBL WHERE LANDING_OUTCOME = 'Failure (drone ship)' and substr(DATE,1,4) = '2015'
```

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

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

- There were 2 landing failures in 2015, both are drone ship cases, using Booster F9 v1.1 and from launch site CCAFS LC-40

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

Click Here for the code for SQL Codes

- The following are the counts 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
  - This was done using the following SQL commands:

```
[70]: %sql SELECT COUNT(*),LANDING_OUTCOME FROM SPACEXTBL WHERE DATE >= '2010-06-04' and DATE <= '2017-03-20'  \
GROUP BY LANDING_OUTCOME ORDER BY COUNT(LANDING_OUTCOME) DESC

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

[70]:

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

- For example, there were 10 instances of “Success” overall split into:
  - 5 “ground pad” cases
  - 5 “drone ship” cases

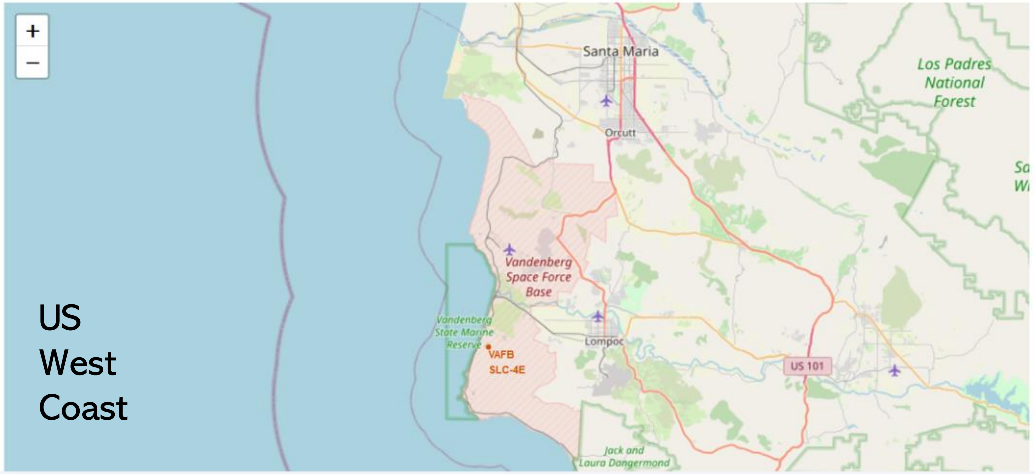
Section 3

Launch Sites Proximities Analysis

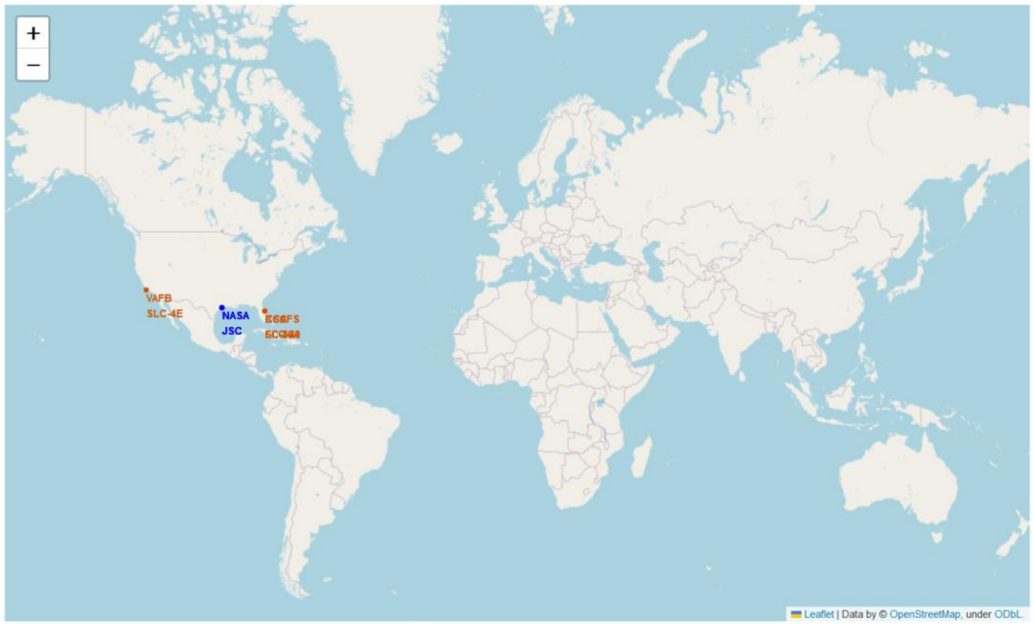
Launch Site Locations

[Click Here for the code](#) for Folium maps

- SpaceX's launch locations (marked in red) are all located on the US coasts, near the equator
  - US West Coast:
    - VAFB SLC-4E [34.632834 -120.610745]
  - US East Coast:
    - CCAFS LC-40 [28.562302, -80.57735]
    - CCAFS SLC-40 [28.563197 -80.576820]
    - KSC LC-39A [28.573255 -80.646895]



NASA marked in blue



US East Coast

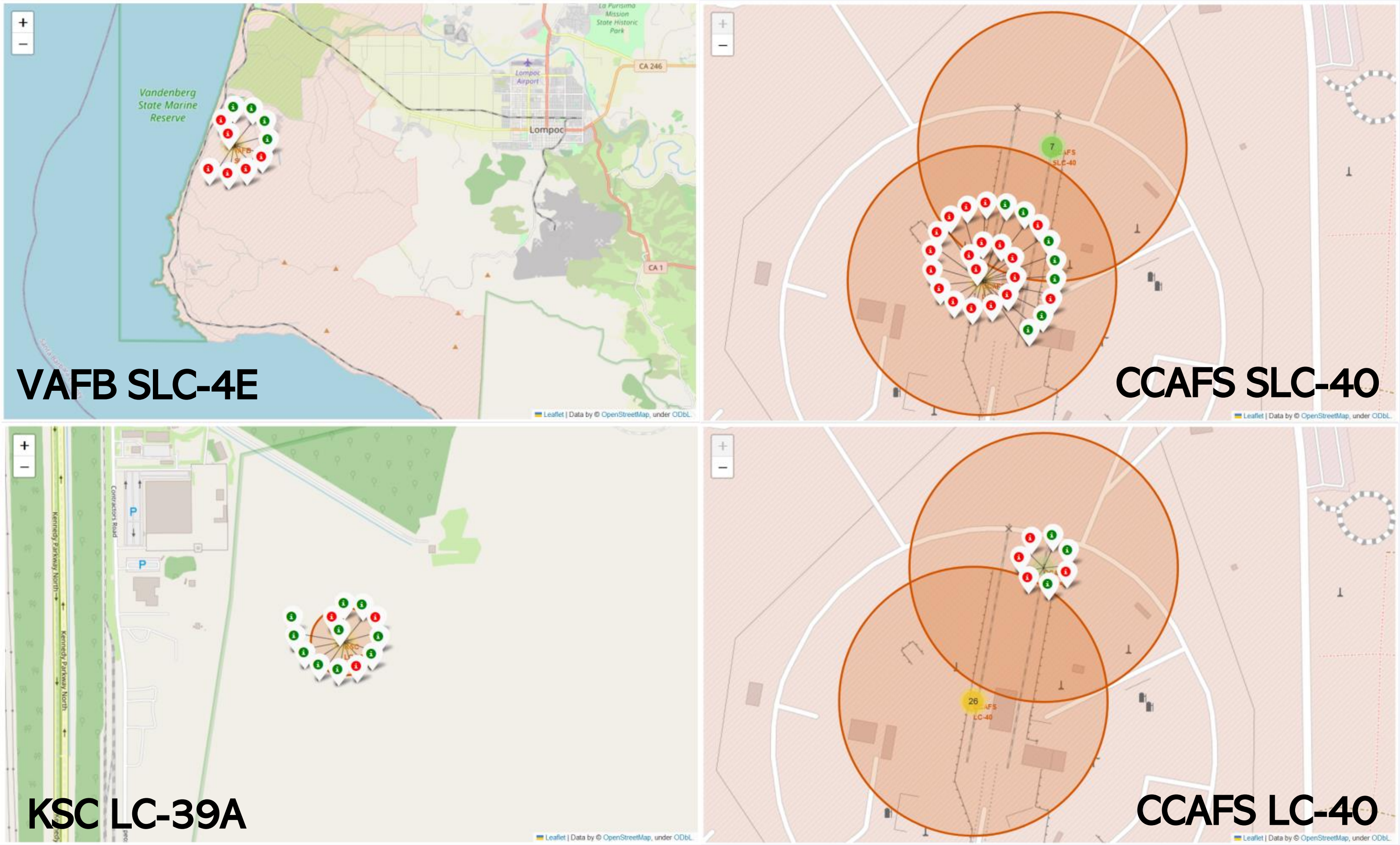
Success/Failed Launches for Each Site

[Click Here for the code](#) for Folium maps

Red = Failed

Green = Success

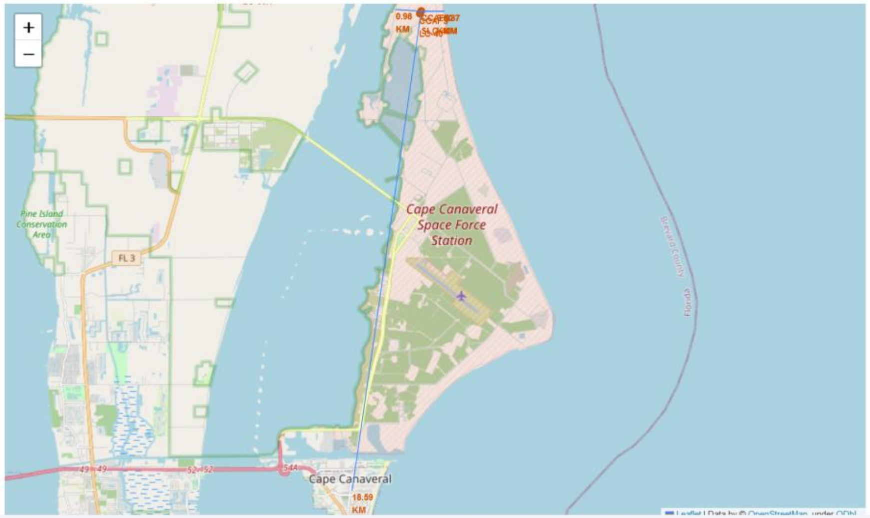
- KSC LC-39A seems to have the highest rate of successful launches
- Will look at the statistics in more detail in Section 4

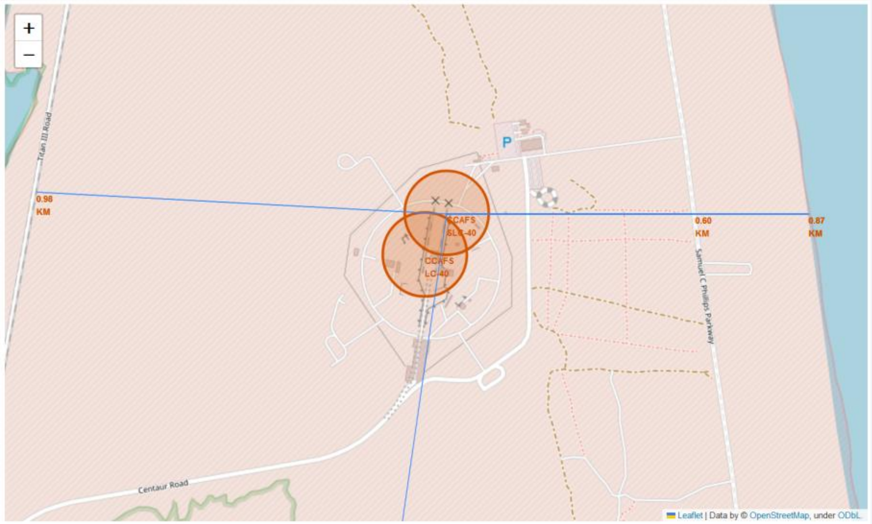


Launch Site and its Proximities

[Click Here for the code]() for Folium maps

- Sites also tend to be near certain features:
  - For example, CCAFS LC-40 (and by extension, CCAFS SLC-40) is :
    - **Near the coastline (**0.87 km) , which means water landings are possible in the event of faults / predicted failures
    - **Near the highway (**0.60 km), easier to transport materials
    - **Near the railway (**0.98 km) , easier to transport materials
  - And ideally **not so near to the city** that it poses danger, but also not so far that it's difficult to find people who want to work at the launch site
    - In this case, the closest city (Cape Canaveral) is18.59 km away (I'm assuming it is a city)





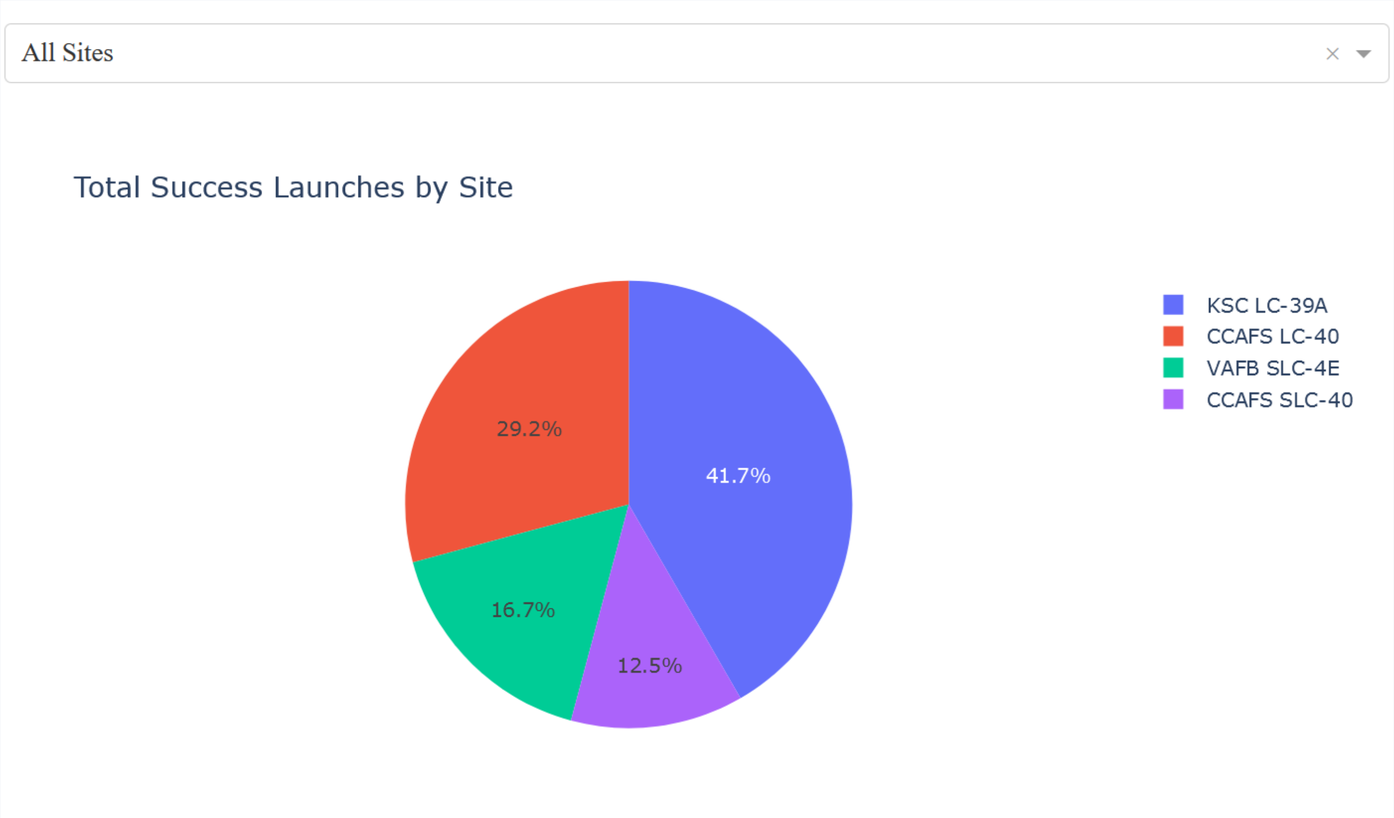
Section 4

Build a Dashboard with Plotly Dash

Launch Success

Click Here for the code
for interactive plot

- Most successes in landings (41.7%) were seen in the **KSC LC-39A** launch site
  - Although this was not controlled for the number of attempts at each site
  - We should look at success rate per site

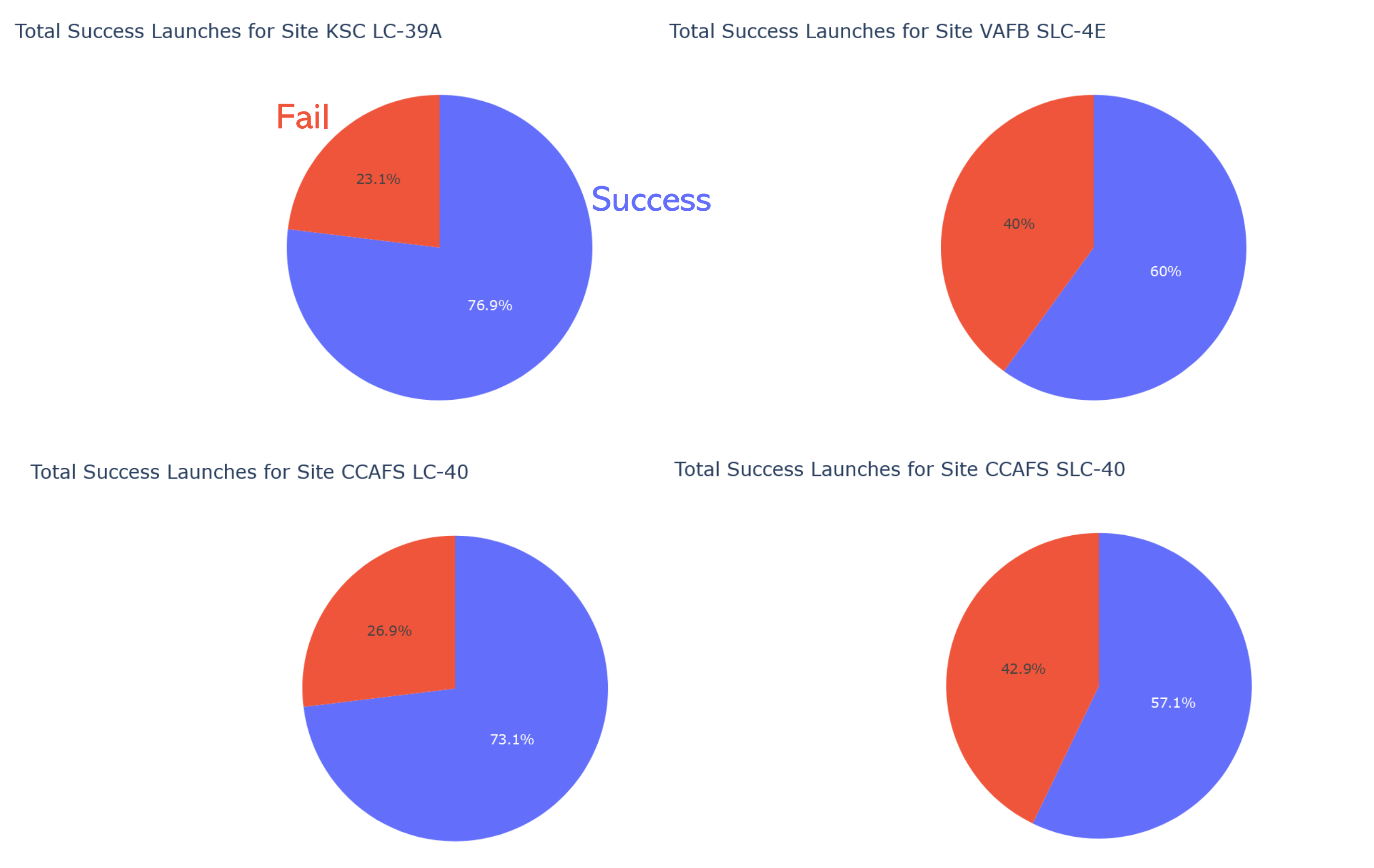


Success per Launch Site

Click Here for the code
for interactive plot

- Site **KSC LC-39A** had the highest rate of success (76.9% of launches landed)

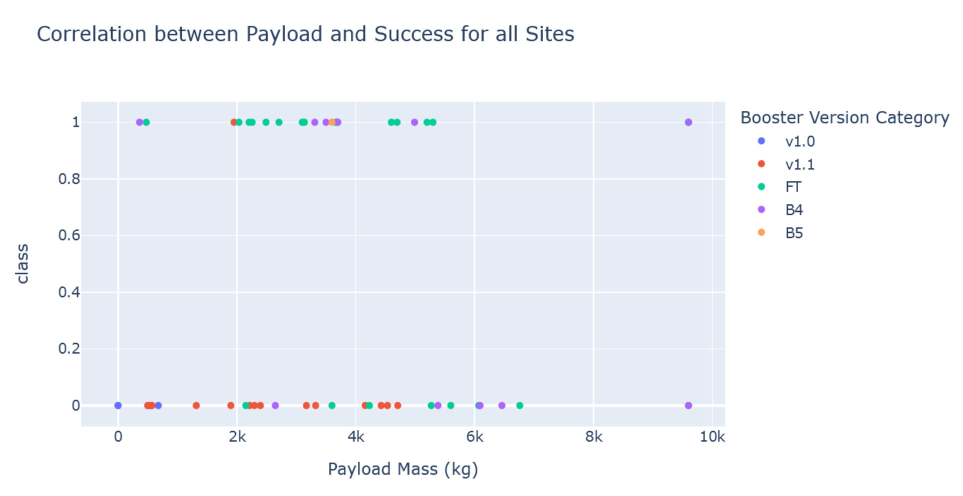
- Of the 4 sites, it has the best success rate, but note that different site had different number of attempts, payload mass, and booster versions

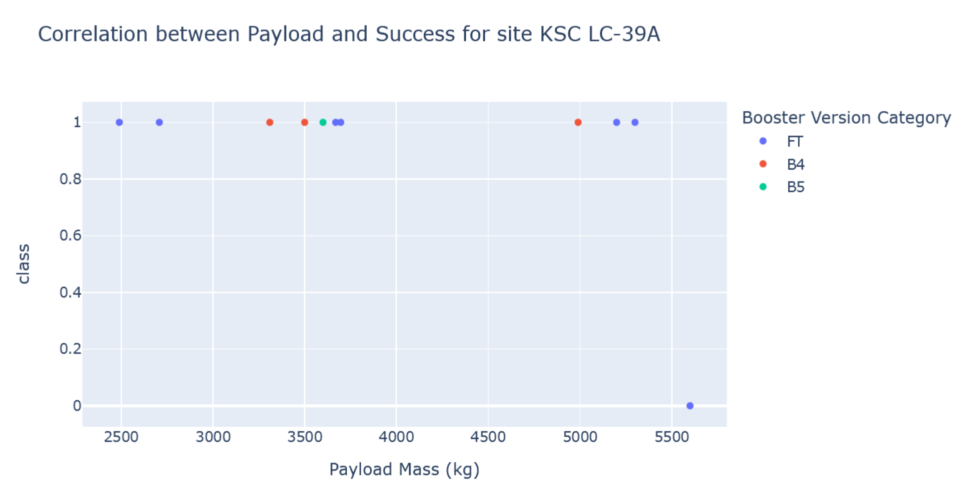


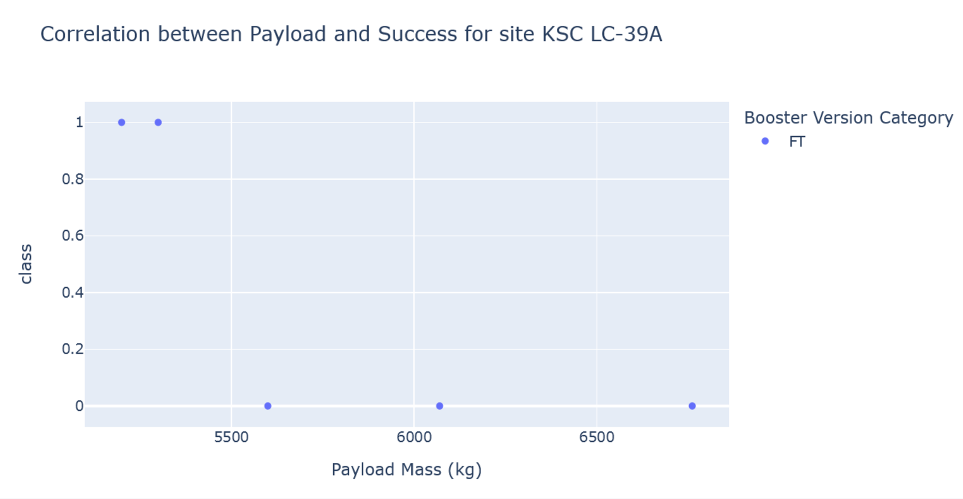
Payload, Boosters and Success

[Click Here for the code](#) for interactive plot

- Generally, across sites, the successful landings are from <u>**smaller payload**</u> (<6000kg)
- Launches with <u>**FT boosters**</u> seems to land successfully most often
- For the must successful site, <u>KSC LC-39A</u>, this seems particularly true. The combination of FT boosters below 5500kgs seems to land consistently there (compare bottom left and right panels)







Section 5

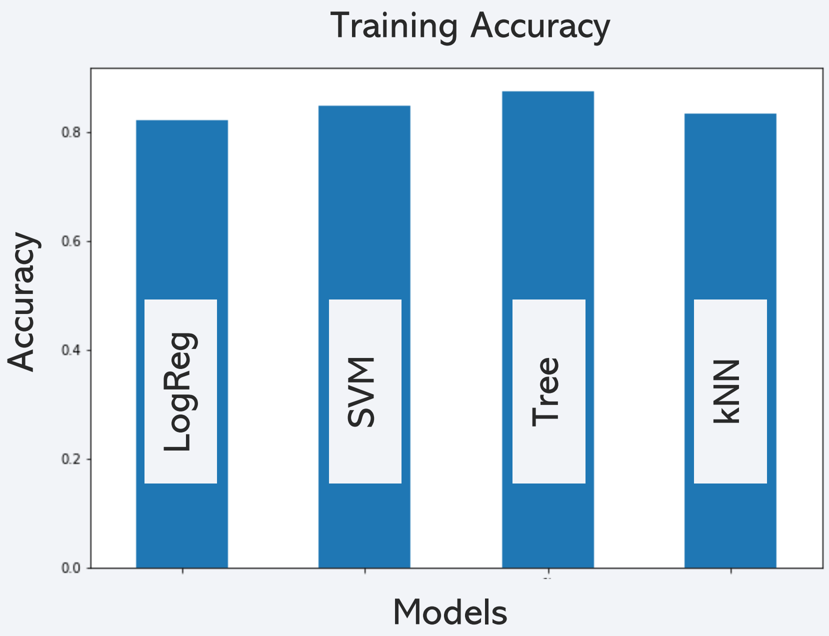
Predictive Analysis (Classification)

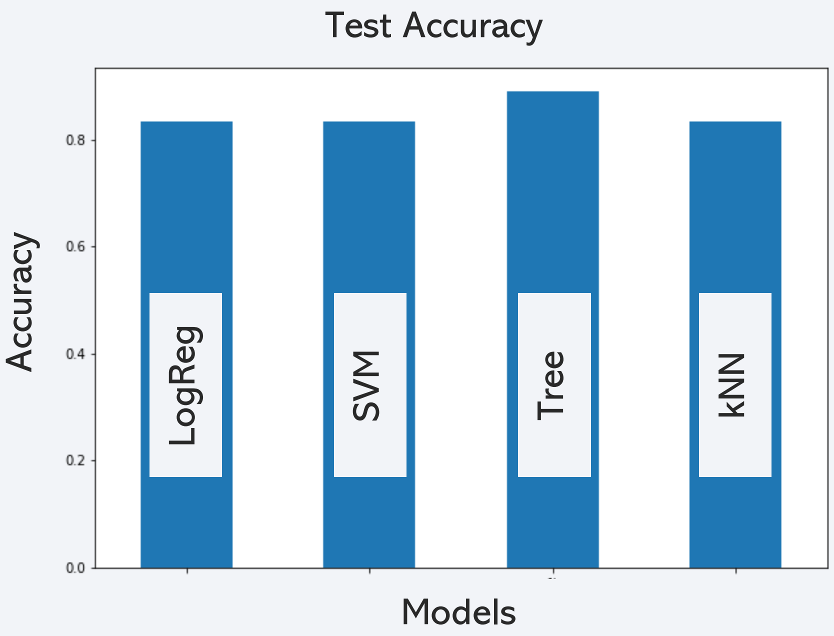
Classification Accuracy

[Click Here for the code](#) for classification

- 20% of data (18 cases) withheld from training, used as test cases
- **<u>Decision Trees</u>** gave the highest training **<u>and</u>** test accuracy *
  - Logistic Regression (LogReg), Support Vector Machines (SVM), k-Nearest Neighbours (kNN) all had the same test accuracies

	Accuracy Train	Accuracy Test
Logreg	0.821429	0.833333
Svm	0.848214	0.833333
Tree	0.875000	0.888889
Knn	0.833929	0.833333





* Caveat:

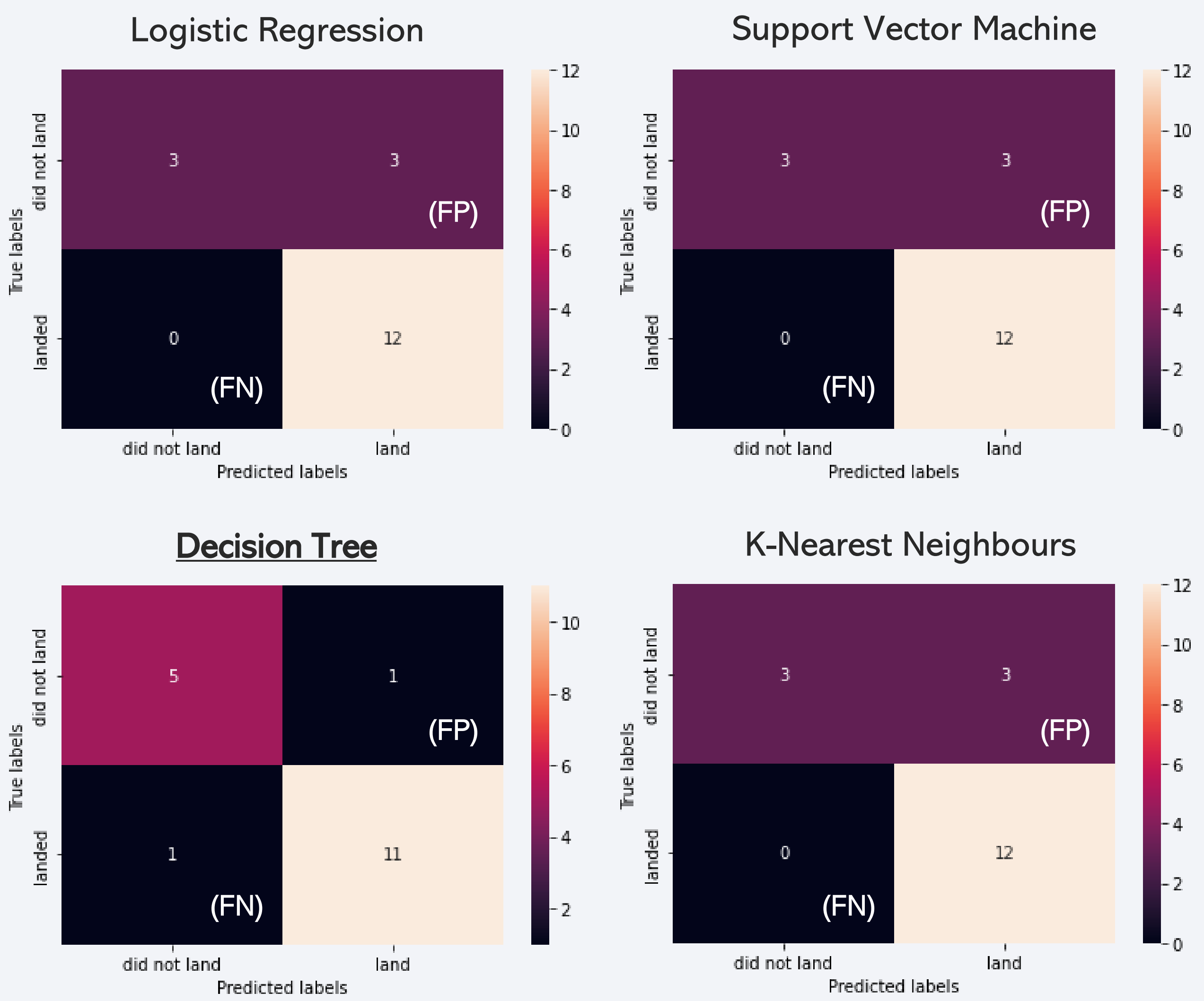
- Because of randomization, it is not guaranteed that one would obtain the same results every time

Confusion Matrix*

[Click Here for the code](#) for classification

* Matrices are for the 18 test cases

- Model with highest accuracy, <u>Decision Trees</u>, makes:
  - <u>3</u> cases of wrong prediction
    - <u>1</u> case of false negatives (FN) (predicted no land, actual land)
    - <u>1</u> case of false positives (FP) (predicted land, actual no land)
  - <u>16</u> cases of correct prediction
    - 11 predicted landings
    - 5 predicted no landings
- LogReg, SVM and kNN
  - same proportion of errors/FP/FN



Conclusions

- In order to maximize chances of successful landing (and therefore reduce costs):
  - Recommend **coastal launch** sites near the **equator** (using KSC LC-39A as an example)
    - Nearer equator probably makes more sense for geosynchronous orbits (which tends to give good success rates)
    - Failed landings have a better probability of recovery if the crash happens in the water (especially if planned for drone ship rescue, etc.)
  - **Low weighted payloads (<5500 – 6000kgs)** are recommended, particularly with **FT boosters**
    - FT boosters also seem to be good with drone ship rescues
- Machine learning models can predict landing successes reasonably well (80-90% accuracy), but false positives (predict success when there is failure) can be an issue
  - Decision Trees work best (compared to Logistic Regression, kNN, SVM; better accuracy + less FP)
  - **Could obtain better predictions with more data** (e.g., atmospheric conditions)

Appendix

- Code and data used in this project can be found at https://github.com/syaheed/public-code/tree/master/IBM%20Data%20Science/Capstone
- Thanks to IBM and Coursera
- Thanks to all peer-reviewers

Thank you!