IBM Developer
SKILLS NETWORK

Winning Space Race with Data Science

Albert Tan
11 Apr 2025

Outline

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

Executive Summary

- Summary of methodologies
  - Data Collection
  - Data Wrangling
  - EDA with data visualization
  - EDA with SQL
  - Build an interactive map with Folium
  - Build a dashboard with Plotly Dash
  - Predictive Analysis (Classification)
- Summary of all results
  - EDA Results
  - Interactive Analytics
  - Predictive Analysis

Introduction

- Project background

  This is the applied capstone project under the IBM Data Science learning course. This project is on delivering data driven insights on SpaceX being the only private company ever to return a spacecraft from low-earth orbit with its Falcon 9 rocket launches which cost 62 million dollars as advertised on its website compared to other providers with cost upward of 165 million dollars. Much of this savings from Space X is because Space X can reuse the first stage of the rocket.

  For any alternate company who wants to bid against SpaceX for a rocket launch, using Data Science methodology helps these competitors to make choices when sending rockets into space and potentially outbid Space X in the rocket launches.

- Business Problem

  Space X has been able to price their Falcon 9 launches at 62 million dollars by reusing the first stage of the rocket provided it lands successfully. However, there were unexpected attempted landings that failed to land and were unplanned. We are on a mission to find out if the first stage of the rocket launch will land successfully based on the historical data set features like payload, landing site of the Falcon 9.

Section 1

Methodology

Methodology

Executive Summary

- Data collection methodology
  - Data was collected using Space X REST API and using web scraping on Wikipedia about Space X Falcon 9.
- Perform data wrangling
  - Determine which feature has missing value and perform one-hot encoding on the first stage landing Outcome label with 1 being successful and 0 being unsuccessful
- Perform exploratory data analysis (EDA) using visualization and SQL
- Perform interactive visual analytics using Folium and Plotly Dash
- Perform predictive analysis using classification model
  - Train Logistic Regression, K-Nearest Neighbors, Support Vector Machines, and Decision Tree models using GridSearchCV to find the best hyperparameters for each of the model and determine the best classification model.

Data Collection

The data was collected using 2 methods:

1. Request to Space X REST API:
    - Using the http GET method to retrieve past Space X rocket launches data via API
    - Parsed and normalized return json result into pandas dataframe
    - Filter dataframe result to include only Falcon 9 launches.

2. Using Web Scraping to Wikipedia Page on Space X Falcon 9
    - Use BeautifulSoap library to web scrape html table containing Falcon 9 launches via url link to the Wikipedia page.
    - Extract all columns from the 3rd html table in the return scraped html page
    - Create pandas dataframe from the parsed html table for further analysis

Data Collection – SpaceX API

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=requests.get(static_json_url)
```

```
response.status_code
```

```
200
```

Task1: REST API request to retrieve and parse Space X launch data into a dataframe

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
data.head()
```

static_fire_date_utc	static_fire_date_unix	tbd	net	window	rocket	success	details	crew	ships	capsules	payloads	launchpad	auto_update	f

Finally lets construct our dataset using the data we have obtained. We we combine the columns into a dictionary.

```
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}
```

Then, we need to create a Pandas data frame from the dictionary launch_dict.

```
# Create a data from launch_dict
data = pd.DataFrame.from_dict(launch_dict)
```

As the initial dataframe contains lots of IDs, a new dataframe is created using these IDs to retrieve useful information for the columns: rocket, payloads, launchpad and cores. The data is also restricted up till the launch date of 13 Nov 2020 (inclusive).

Show the summary of the dataframe

```
# Show the head of the dataframe
data.head()
```

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

Filter the dataframe to include only Falcon 9 launches.

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 dataframe using the BoosterVersion column to only keep the Falcon 9 launches. Save the filtered data to a new dataframe called data_falcon9.

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

	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins	Reused	Legs	LandingPad	Block	ReusedCount	Serial	Longitude	Latitude
4	6	2010-06-04	Falcon 9	NaN	LEO	CCSFS SLC 40	None None	1	False	False	False	None	1.0	0	B0003	-80.577366	28.561857
5	8	2012-05-22	Falcon 9	525.0	LEO	CCSFS SLC 40	None None	1	False	False	False	None	1.0	0	B0005	-80.577366	28.561857
6	10	2013-03-01	Falcon 9	677.0	ISS	CCSFS SLC 40	None None	1	False	False	False	None	1.0	0	B0007	-80.577366	28.561857
7	11	2013-09-29	Falcon 9	500.0	PO	VAFB SLC 4E	False Ocean	1	False	False	False	None	1.0	0	B1003	-120.610829	34.632093
8	12	2013-12-03	Falcon 9	3170.0	GTO	CCSFS SLC 40	None None	1	False	False	False	None	1.0	0	B1004	-80.577366	28.561857

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
```

Task 3: Dealing with Missing Values

Calculate below the mean for the PayloadMass using the .mean(). Then use the mean and the .replace() function to replace np.nan values in the data with the mean you calculated.

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

# Replace the np.nan values with its mean value
data_falcon9['PayloadMass'].fillna(value=mean_PayLoadMass, inplace=True)
```

```
<bound method NDFrame._add_numeric_operations.<locals>.mean of 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
```

As PayloadMass column contains missing value, we Impute these missing value with the mean of PayloadMass for falcon 9 launches.

GitHub link: 1. Hands-on Lab - Complete the Data Collection API Lab.ipynb

Data Collection - Scraping

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)
```

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

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

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

```
# Use soup.title attribute
soup.title
```

```
<title>List of Falcon 9 and Falcon Heavy launches - Wikipedia</title>
```

Using HTTP GET request to retrieve Wikipedia HTML page of Falcon 9 and load response into BeautifulSoap object.

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 `BeautifulSoup`, please check the external reference link towards the end of this lab

```
# 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)
```

Python

```
<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 Universal Time">UTC</a>)
</th>
<th scope="col"><a href="/wiki/List_of_Falcon_9_first-stage_boosters" title="List of Falcon 9 first-stage boosters">Version,<br/>Booster</a> <sup class="reference" id="cite_ref-booster_11-0"><a href
</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"><span class="cite-bracket">[</span>c<span class="cite-bracket">]</span></a></sup>
</th>
<th scope="col">Payload mass
```

Extract all column/variable names from the HTML table header

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

We will create an empty dictionary with keys from the extracted column names in the previous task. Later, this dictionary will be converted into a Pandas dataframe

```
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']=[]
```

Parsed the html table and extract the relevant columns and append into a dictionary object

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

Create a dataframe from the parsed dictionary object.

```
df
```

	Flight No.	Launch site	Payload	Payload mass	Orbit	Customer	Launch outcome	Version Booster	Booster landing	Date	Time
0	1	CCAFS	Dragon Spacecraft Qualification Unit	0	LEO	SpaceX	Success\n	F9 v1.07B0003.18	Failure	4 June 2010	18:45
1	2	CCAFS	Dragon	0	LEO	NASA	Success	F9 v1.07B0004.18	Failure	8 December 2010	15:43
2	3	CCAFS	Dragon	525 kg	LEO	NASA	Success	F9 v1.07B0005.18	No attempt\n	22 May 2012	07:44
3	4	CCAFS	SpaceX CRS-1	4,700 kg	LEO	NASA	Success\n	F9 v1.07B0006.18	No attempt	8 October 2012	00:35

GitHub link: [2. Hands-on Lab - Complete the Data Collection with Web Scraping lab.ipynb]

Data Wrangling

TASK 1: Calculate the number of launches on each site

The data contains several Space X launch facilities: Cape Canaveral Space Launch Complex 40 **VAFB SLC 4E** , Vandenberg Air Force Base Space Launch Complex 4E **(SLC-4E)**, Kennedy Space Center Launch Complex 39A **KSC LC 39A** .The location of each Launch Is placed in the column `LaunchSite`

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

Use the method `value_counts()` on the column `LaunchSite` to determine the number of launches on each site:

```
# Apply value_counts() on column LaunchSite
df['LaunchSite'].value_counts()
```

Python

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

Calculate number of launches on each site

TASK 2: Calculate the number and occurrence of each orbit

Use the method `.value_counts()` to determine the number and occurrence of each orbit in the column `Orbit`

```
# Apply value_counts on Orbit column
df['Orbit'].value_counts()
```

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

Calculate number & occurrence of each orbit

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_outcomes`.Then assign it to a variable landing_outcomes.

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

```
Outcome
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: count, dtype: int64
```

Calculate number & occurrence of mission outcome for each orbit

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 in `Outcome` is in the set `bad_outcome`; otherwise, it's one. Then assign it to the variable `landing_class`:

```
# landing_class = 0 if bad_outcome
# landing_class = 1 otherwise
landing_class = []
for outcome in df['Outcome']:
    if outcome in bad_outcomes:
        landing_class.append(0)
    else:
        landing_class.append(1)
```

Pytho

Create landing label (0:unsuccessful, 1:successful) from outcome column

This variable will represent the classification variable that represents the outcome of each launch. If the value is zero, the first stage did not land successfully; one means the first stage landed Successfully

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

Pytho

	Class
0	0
1	0
2	0
3	0
4	0

We can use the following line of code to determine the success rate:

```
df["Class"].mean()
```

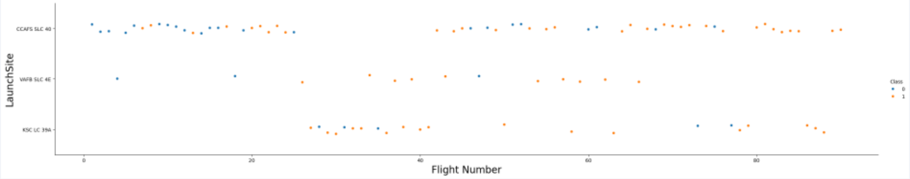
```
np.float64(0.6666666666666666)
```

Calculate average success rate

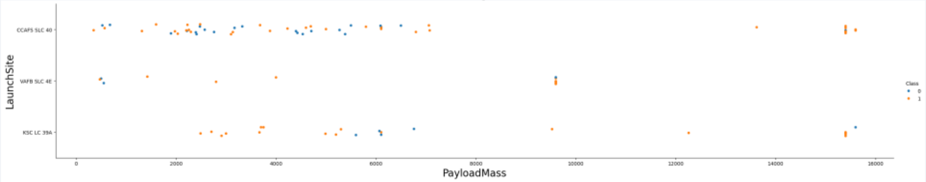
GitHub Link: 3. Hands-on Lab - Data Wrangling.ipynb

EDA with Data Visualization

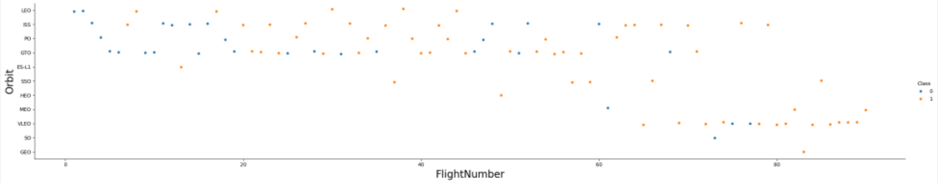
Scatter Plot used to visualize relationship between 2 independent variables:



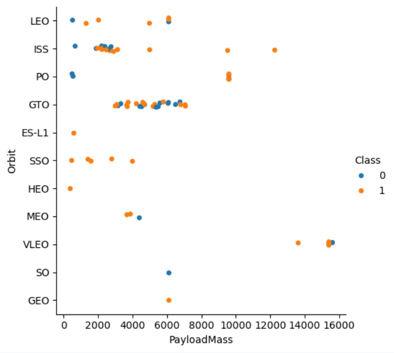
LaunchSite vs Flight Number



LaunchSite vs PayloadMass

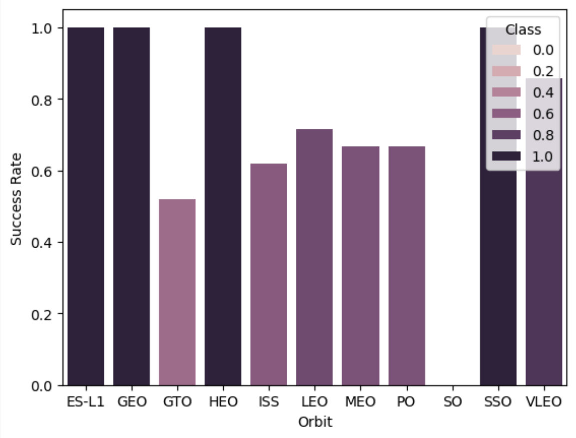


Orbit vs FlightNumber

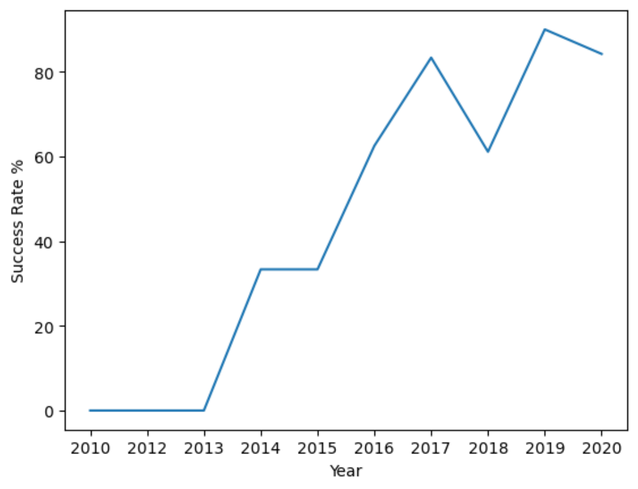


Orbit vs PayloadMass

Bar chart used to visualize relationship between success rate & Orbit type:



Bar chart to visualize relationship between Success Rate % & Year:



GitHub Link: [5. EDA with Visualization Lab.ipynb]

EDA with SQL

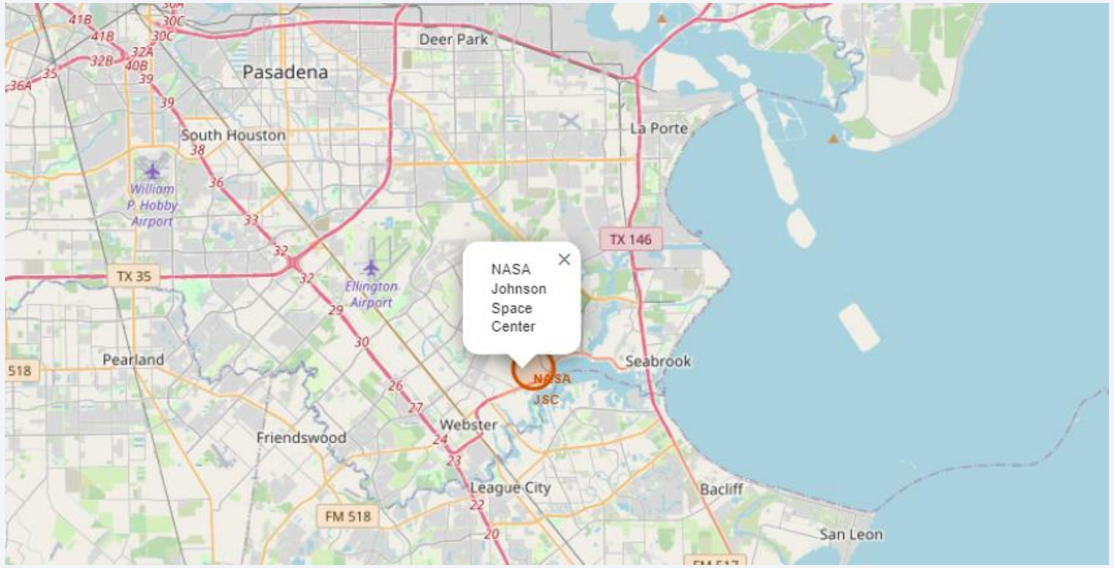
SQL queries are used to perform the following:

- **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 succesful 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 by 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 landing outcomes (such as Failure (drone ship) or Success (ground pad)) between the date 2010-06-04 and 2017-03-20, in descending order.**

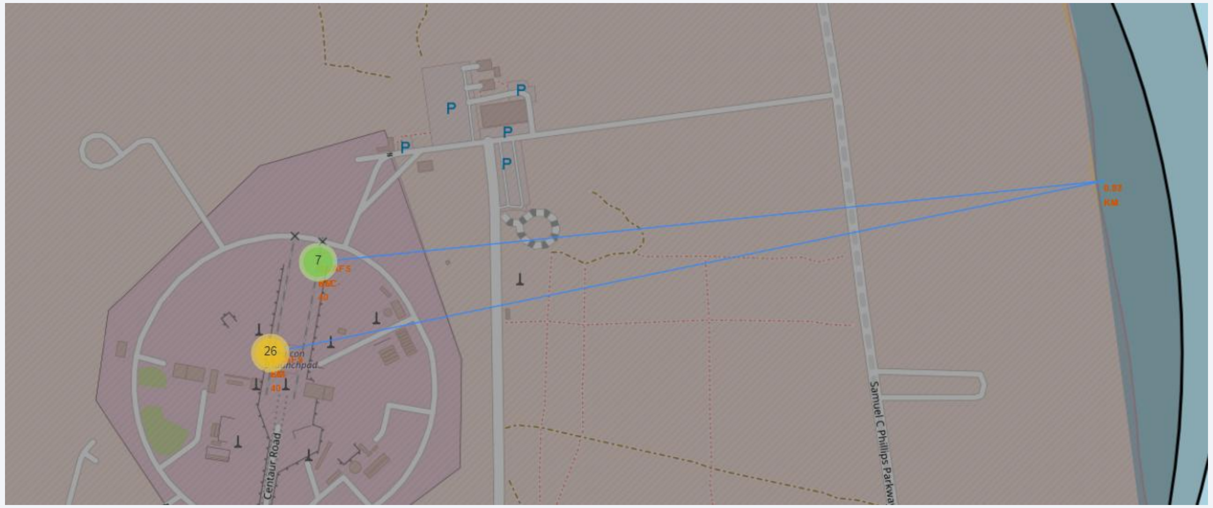
GitHub link: [4. Hands-on Lab - Complete the EDA with SQL.ipynb]

Build an Interactive Map with Folium

Circle marker used to show NASA Johnson Space Center



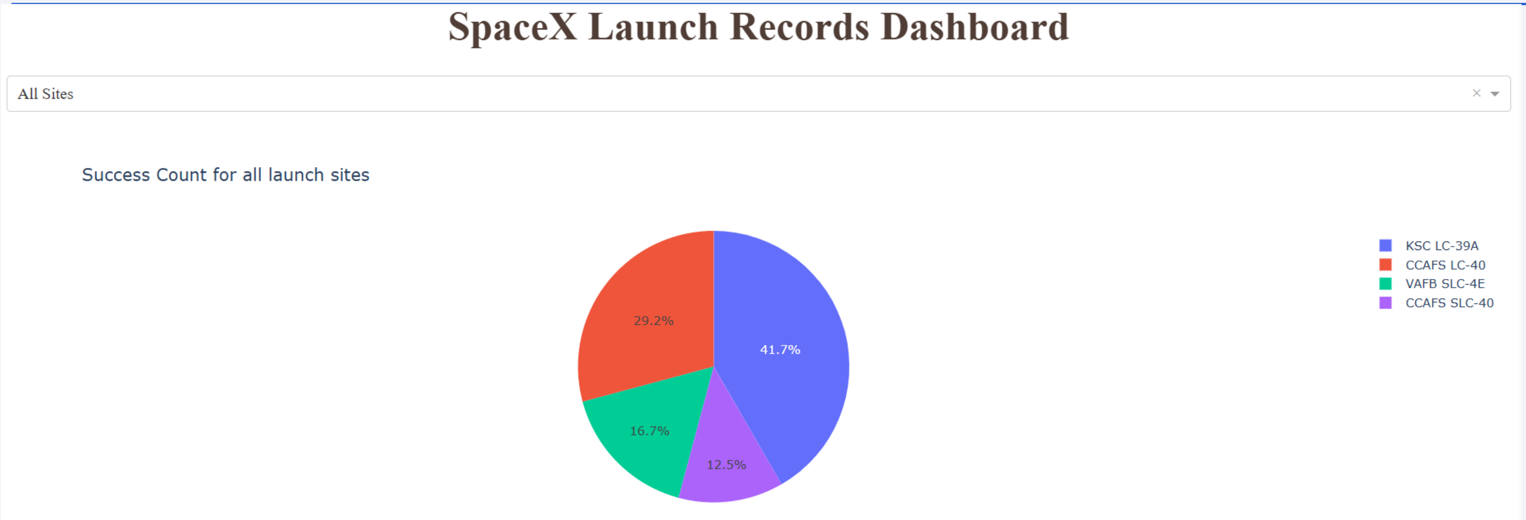
Distance Marker using lines to show proximity of a launch site. E.g. Distance to coastline



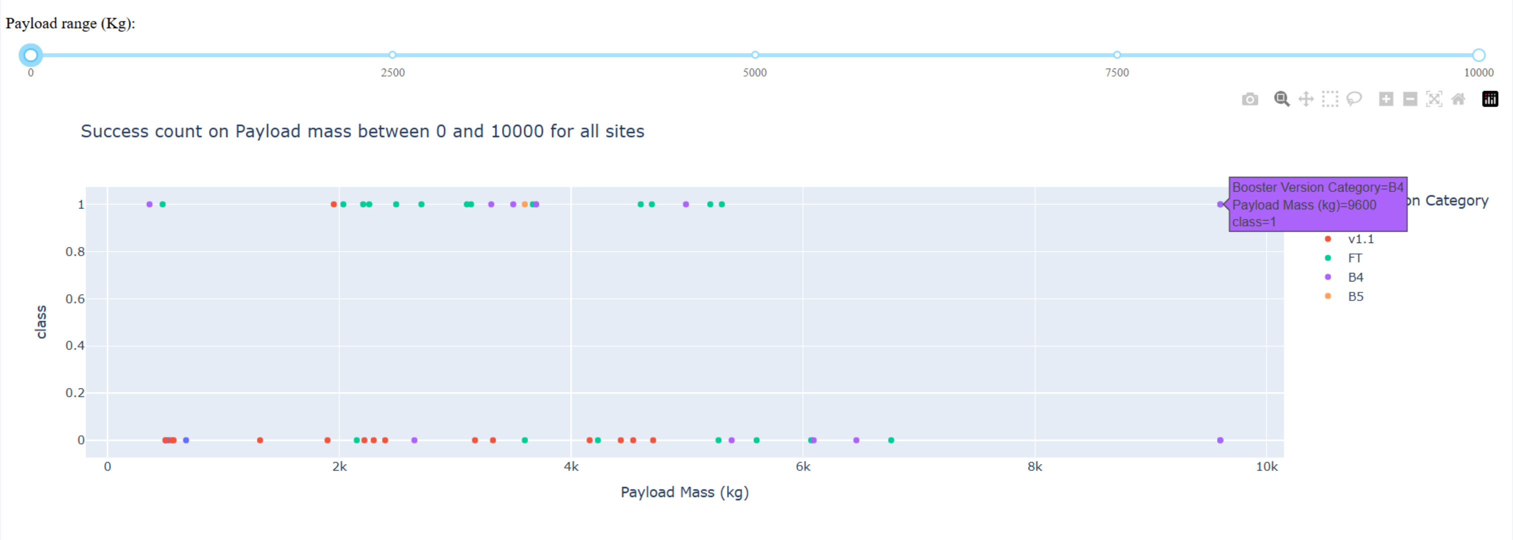
Circle marker used to show various launch sites.

GitHub Link: [6. Hands-on Lab - Interactive Visual Analytics with Folium lab.ipynb]

Build a Dashboard with Plotly Dash



Used a drop-down option to select either a launch site/All sites and show the success count on a pie chart.



Used a payload range slider to show the success launches of all / each site by payload mass on a scatter plot.

GitHub Link: [7. Hands-on Lab - Build an Interactive Dashboard with Ploty Dash.py]

Predictive Analysis (Classification)

- The Space X launch data were split into training and testing datasets. (20% of the data were used for the test set.)
- Models using Logistic Regression, SVM, Decision Tree and KNN were created and trained using the training dataset and fitted into GridSearchCV to find the best hyper-parameters for each of the classification model.
- Each model accuracy were evaluated using testing datasets. All models exhibit good accuracy of above 83% but the Decision Tree model has the best accuracy at 94.44%.

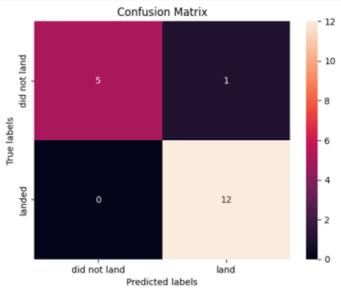
TASK 12

Find the method performs best:

```
print('LR Accuracy:', '{:.2%}'.format(lr_accuracy_test))
print( 'SVM Accuracy:', '{:.2%}'.format(svm_accuracy_test))
print('Decision Tree Accuracy:', '{:.2%}'.format(tree_accuracy_test))
print('KNN Accuracy:', '{:.2%}'.format(knn_accuracy_test))
```

```
LR Accuracy: 83.33%
SVM Accuracy: 83.33%
Decision Tree Accuracy: 94.44%
KNN Accuracy: 83.33%
```

- Confusion Matrix were also created for each of the models in LR, SVM, Decision Tree and KNN.

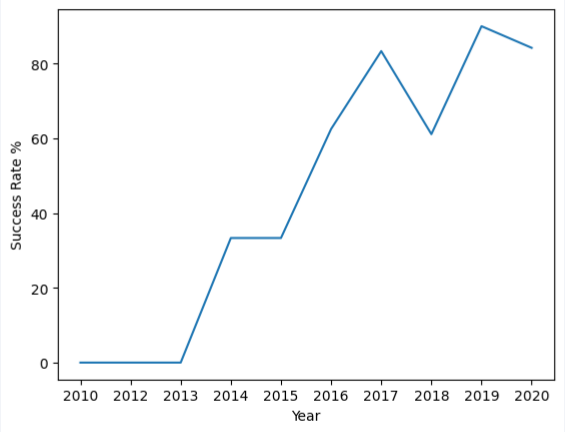


Confusion Matrix of Decision Tree Model

GitHub Link: [8. Hands-on Lab - Complete the Machine Learning Prediction lab.ipynb]

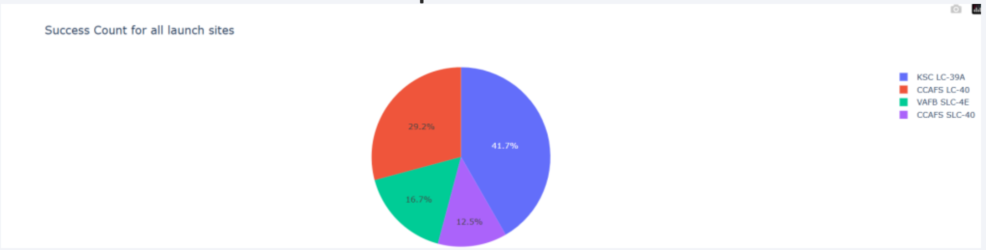
Results

- Decision Tree model is the best performing model for forecasting outcomes in this data.
- Lighter payloads have a higher success rate compared to heavier ones.

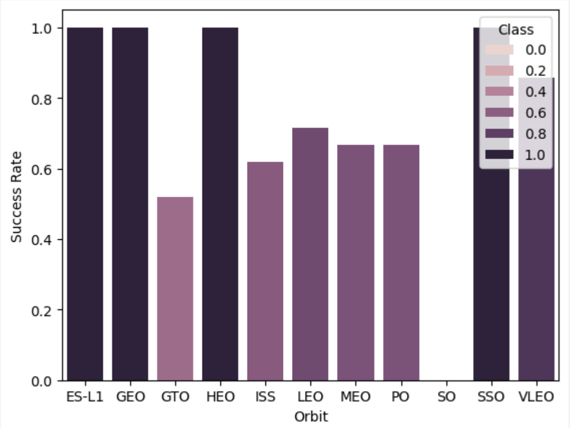


- Success rate seems to increase from year 2013, suggesting a trend in improvement over the years.

- 39A at Kennedy Space Center (KSC LC-39A) has the highest number of successful launches compared to other launch sites.



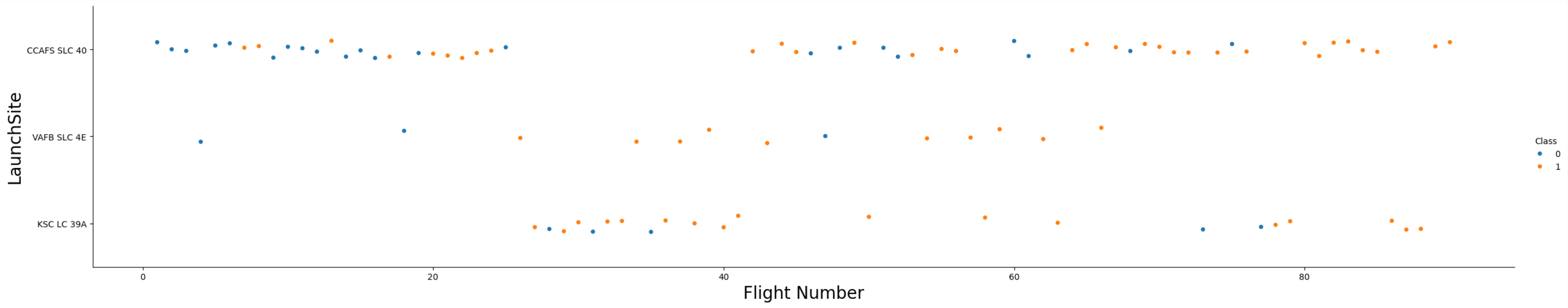
- GEO,HEO,SSO,ES L1 orbit types shows the highest rates of successful launches.



Section 2

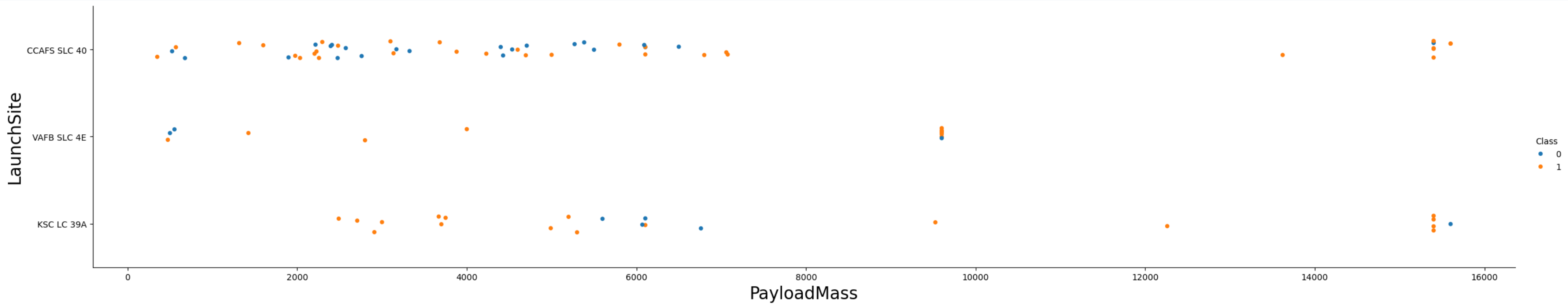
Insights drawn from EDA

Flight Number vs. Launch Site



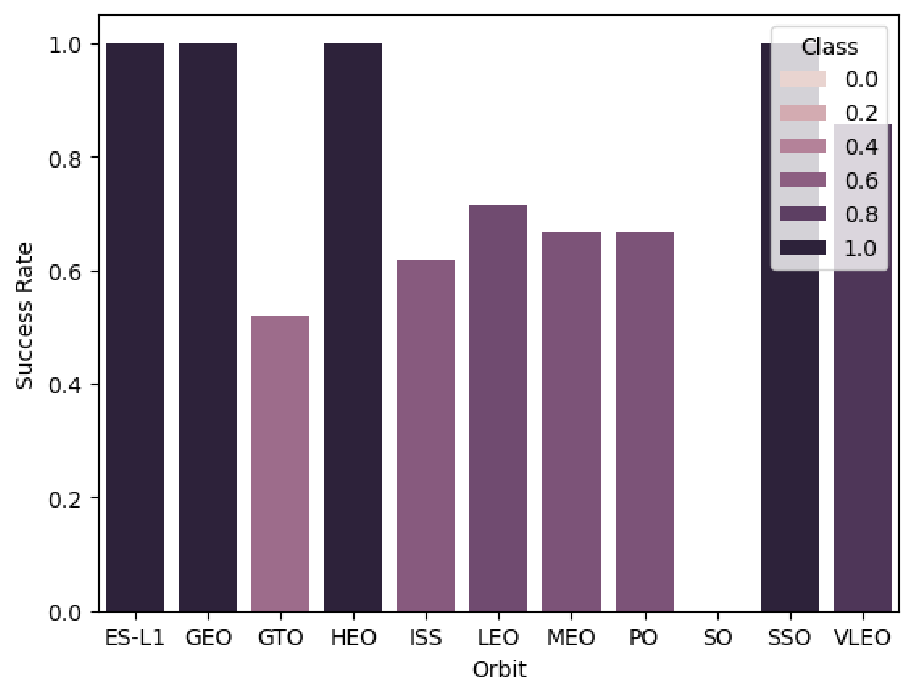
- The CCAFS SLC 40 launch site has the highest number of launches compared to other launch sites.

Payload vs. Launch Site



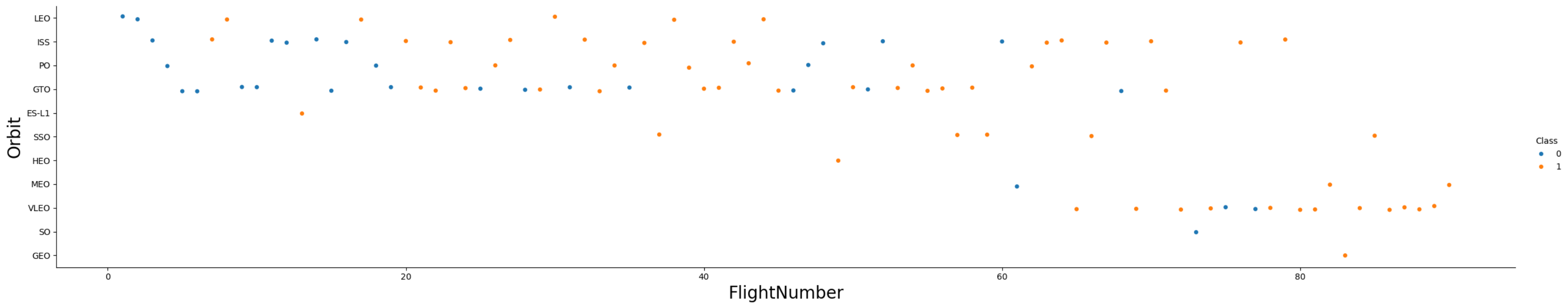
- Payloads with lower mass below 8000 have more launches compared to higher payloads of more than 8000.
- CCAFS SLC 40 and KSC LC 39A launch sites were used for heavier payload mass (> 14000) launches.

Success Rate vs. Orbit Type



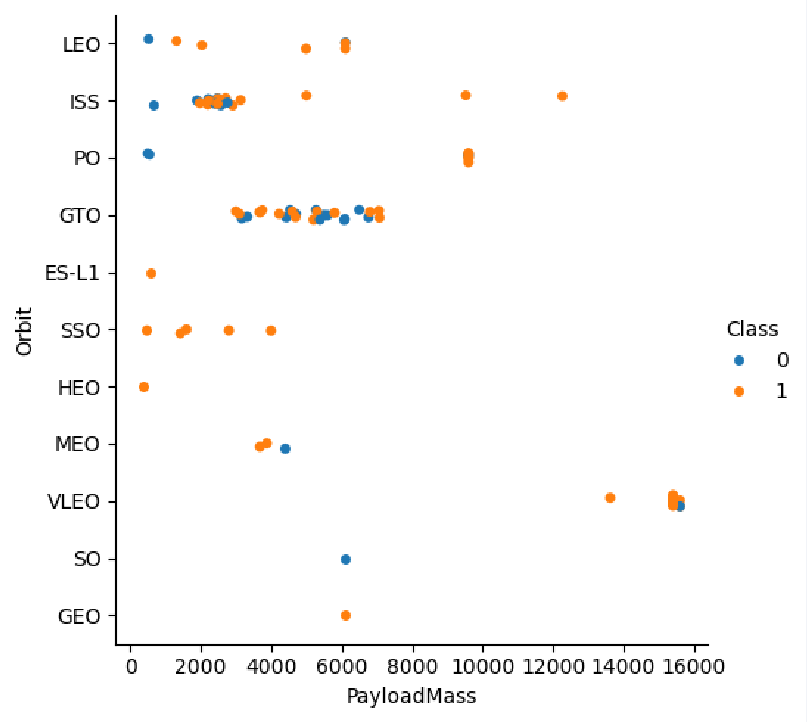
- ES-L1, GEO, HEO, SSO orbits have the most successful rate compared to other orbits.

Flight Number vs. Orbit Type



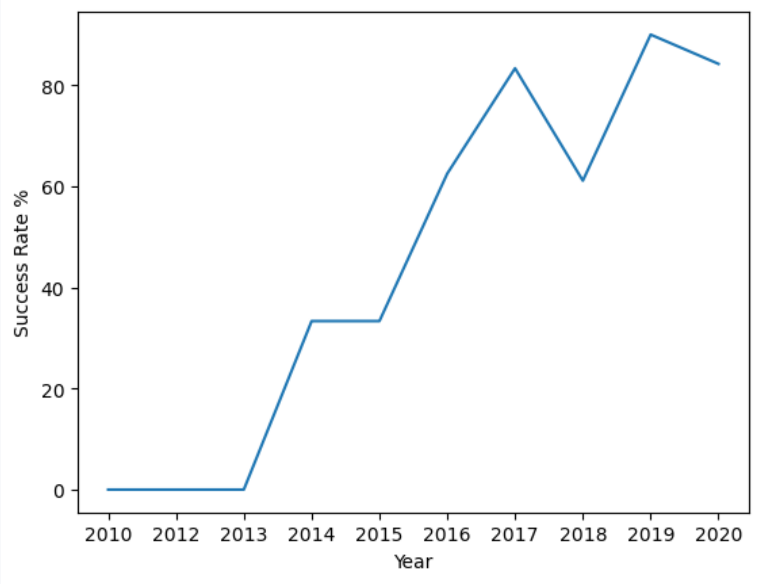
- The LEO orbit seems to exhibit success related to the number of flights. Conversely, in the GTO orbit, there appears to be no relationship between flight number and success.
- VLEO orbit seems to exhibit higher success in the later years based on the later flight numbers.

Payload vs. Orbit Type



- PO, LEO, and ISS orbits exhibit higher success landing rate with heavy payloads but for GTO orbit, it is difficult to differentiate the success rate as both outcomes are available.

Launch Success Yearly Trend



- The chart shows success rate increasing from 2013 till 2020. This could be due to experiences learned from past launches as well as advancement in technology improvement.

All Launch Site Names

Task 1

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

```
%%sql
SELECT DISTINCT LAUNCH_SITE
FROM SPACEXTABLE;
```

* sqlite:///my_data1.db
Done.

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

- SQL SELECT query was used to retrieve the various distinct launch sites.

Launch Site Names Begin with 'CCA'

Task 2

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

```
%%sql
SELECT *
FROM SPACEXTABLE WHERE launch_site like 'CCA%' LIMIT 5;
```

Python

* 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

- Using SQL SELECT query to list 5 records with starting launch site of ‘CCA’.

Total Payload Mass

Task 3

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

```
%%sql
SELECT SUM(PAYLOAD_MASS__KG_)
FROM SPACEXTABLE
WHERE Customer LIKE 'NASA (CRS)'
```

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

SUM(PAYLOAD_MASS__KG_)
45596

- Using SQL SELECT query and the SUM aggregate function to return the total payload mass by boosters launched by NASA (CRS).

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 SPACEXTABLE
where "Booster_Version" = 'F9 v1.1';
```

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

AVG(PAYLOAD_MASS__KG_)
2928.4

- Using SQL SELECT query with the AVG function to return the average payload mass carried by booster version F9 V1.1

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 SPACEXTABLE
WHERE Landing_Outcome LIKE '%Success (ground pad)%'
```

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

MIN(Date)
2015-12-22

- Using SQL SELECT with the MIN function to return the earliest date of a success landing outcome with a LIKE predicate in the query condition to match any string with “Success” in the column Landing_Outcome column.

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 SPACEXTABLE
WHERE "Landing_Outcome" like '%Success (drone ship)%'
AND PAYLOAD_MASS__KG_ > 4000
AND PAYLOAD_MASS__KG_ < 6000
```

[28]

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

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

- Using SQL SELECT query to return the booster versions with successful drone ship landing that has payload between 4000 and 6000 condition specified in the SQL query condition.

Total Number of Successful and Failure Mission Outcomes

Task 7

List the total number of successful and failure mission outcomes

```
%%sql

SELECT DISTINCT Mission_Outcome, COUNT(*) FROM SPACEXTABLE Group By Mission_Outcome;
```

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

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

- Using SQL SELECT query with COUNT function to return the total number of successful and failure mission outcomes group by Mission_Outcome.

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 SPACEXTABLE
WHERE PAYLOAD_MASS__KG_ in (
    SELECT MAX(PAYLOAD_MASS__KG_)
    FROM SPACEXTABLE
    )
ORDER BY "Booster_Version";
```

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

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

- Using SQL SELECT query to list the booster versions where the query involves a subquery to retrieve the maximum payload mass in the sql condition.

2015 Launch Records

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  substr(Date, 6,2) as MTH, Landing_Outcome, Booster_Version, Launch_Site
FROM SPACEXTABLE
WHERE Landing_Outcome = 'Failure (drone ship)' AND substr(Date,1,4) = '2015';
```

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

MTH	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

- Using SQL SELECT statement to retrieve the list of month, failed landing outcome, booster version and launch site for those failed landing in drone ship in the year 2015.

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) AS Total_Count
FROM SPACEXTABLE
WHERE Date BETWEEN '2010-06-04' AND '2017-03-20'
GROUP BY Landing_Outcome
ORDER BY Total_Count DESC;
```

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

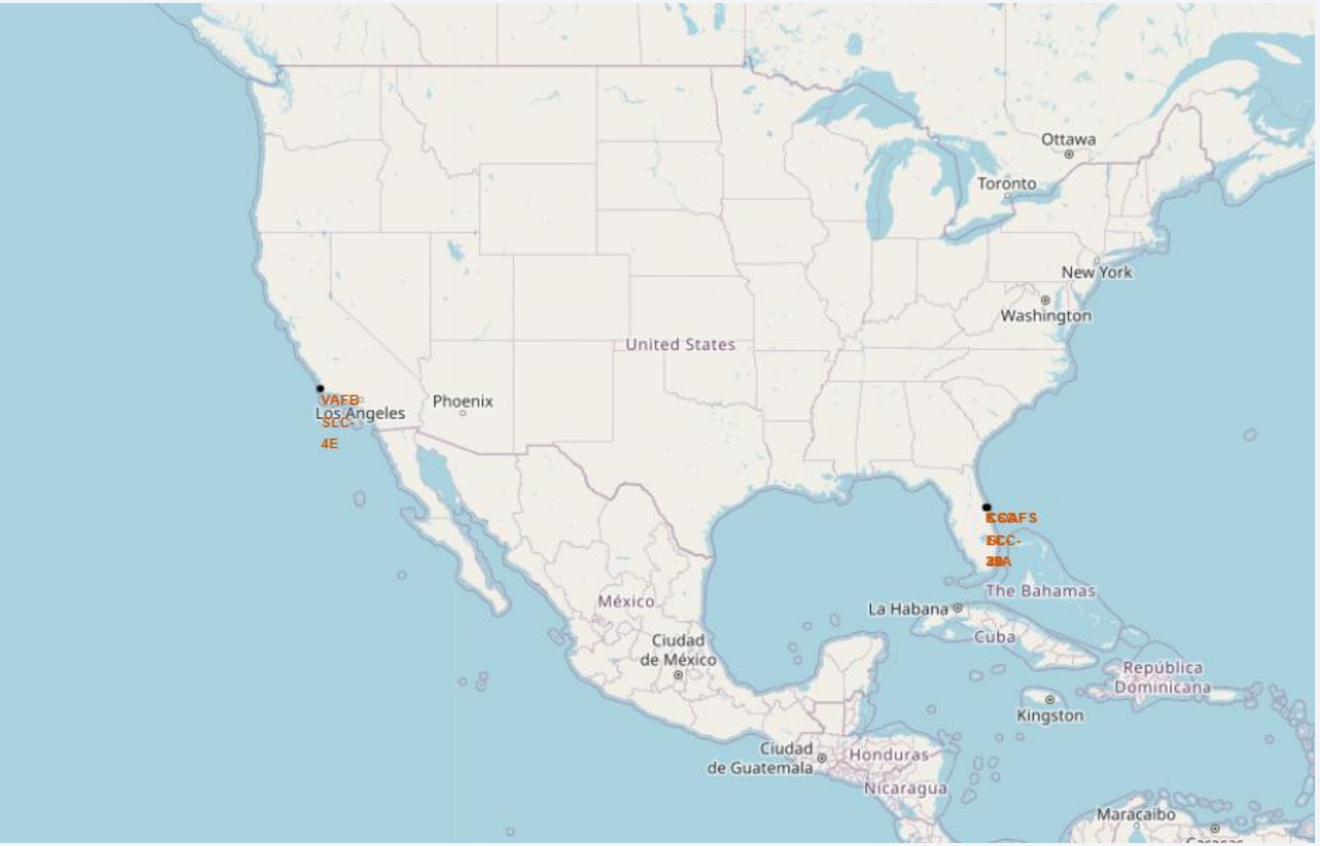
Landing_Outcome	Total_Count
No attempt	10
Success (drone ship)	5
Failure (drone ship)	5
Success (ground pad)	3
Controlled (ocean)	3
Uncontrolled (ocean)	2
Failure (parachute)	2
Precluded (drone ship)	1

- Using SQL SELECT query to 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 by total count.

Section 3

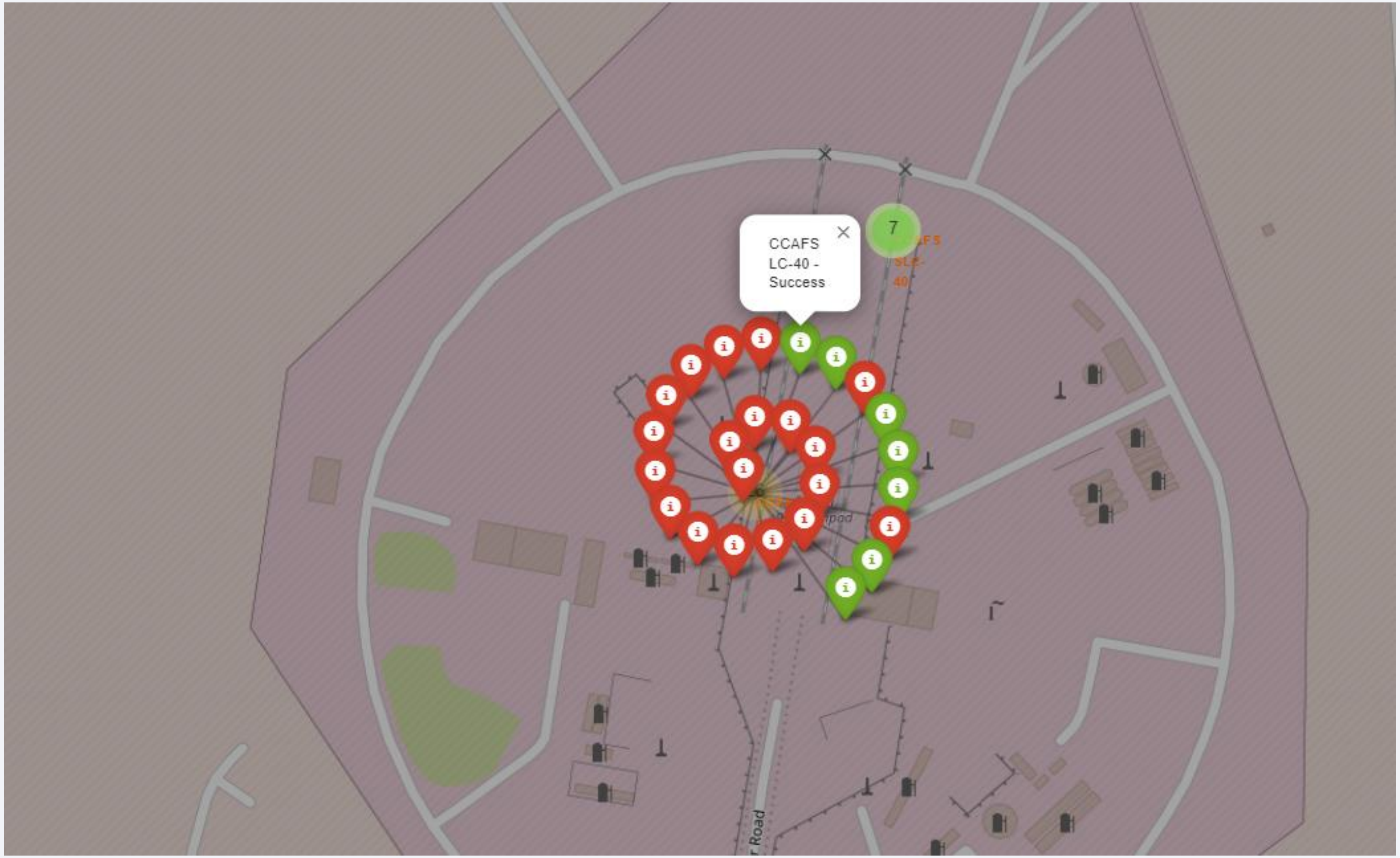
Launch Sites Proximities Analysis

All Launch Sites on Map



This image shows all the launch sites labeled by markers and name of the site on the map.

Map showing launch outcomes (success/fail) on a site



- This map shows a cluster of launches for a site where a green marker denotes a successful launch and a red marker shows a failed launched landing.

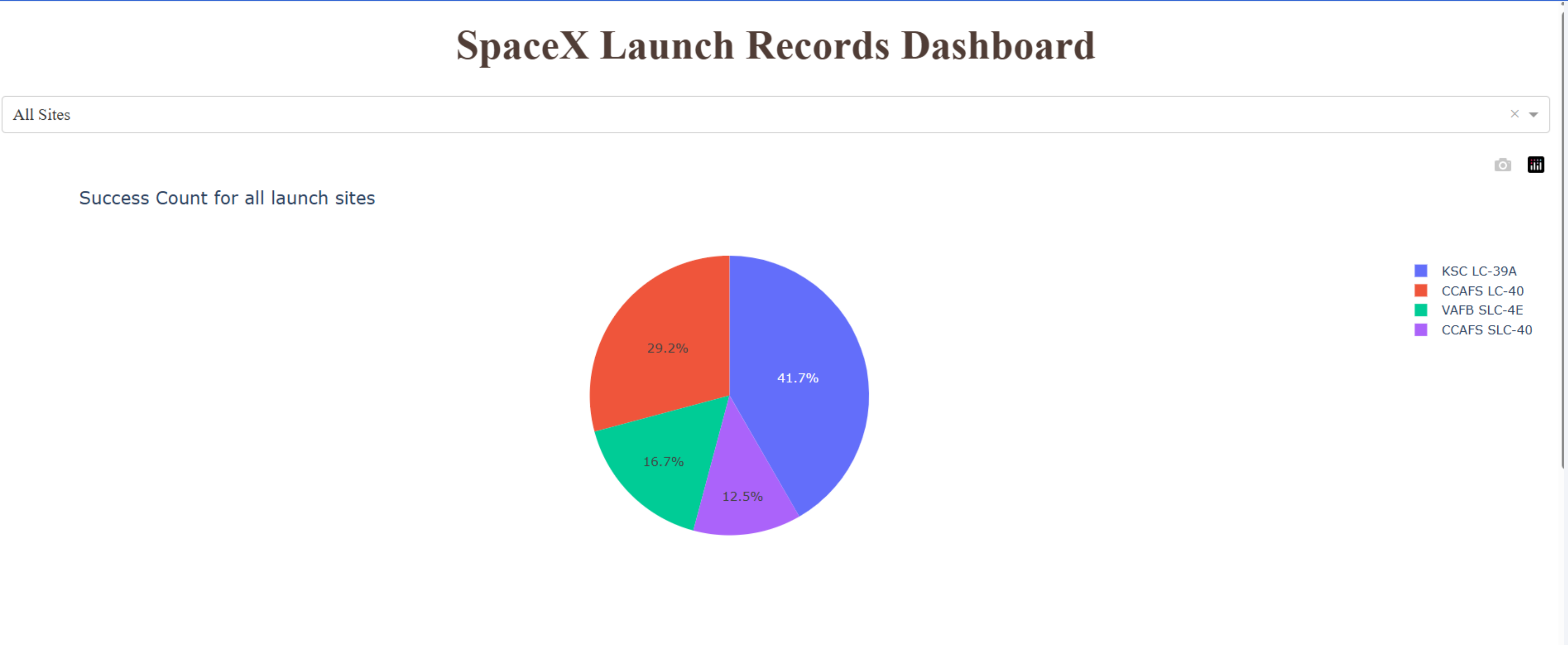
Distance between a launch site to a proximity

- Map showing distance of proximity between launch site CCAFS SLC-40 to the nearest coastline. This distance is approximately 0.83km.
- One possible reason for a launch site to be near coastline is to factor in the risk of launch landing failure and the possibility of the stage 1 rocket falling into the sea to minimize damage to other areas.

Section 4

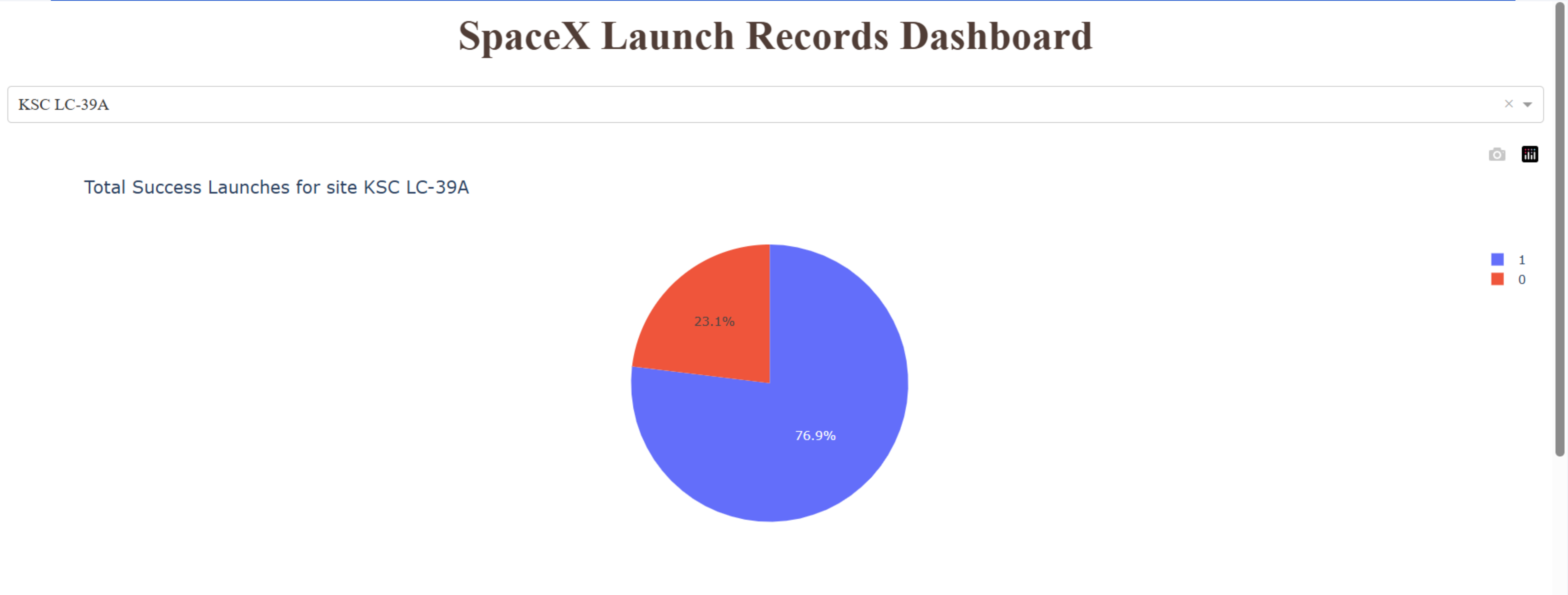
Build a Dashboard with Plotly Dash

Launch success rate for All Sites



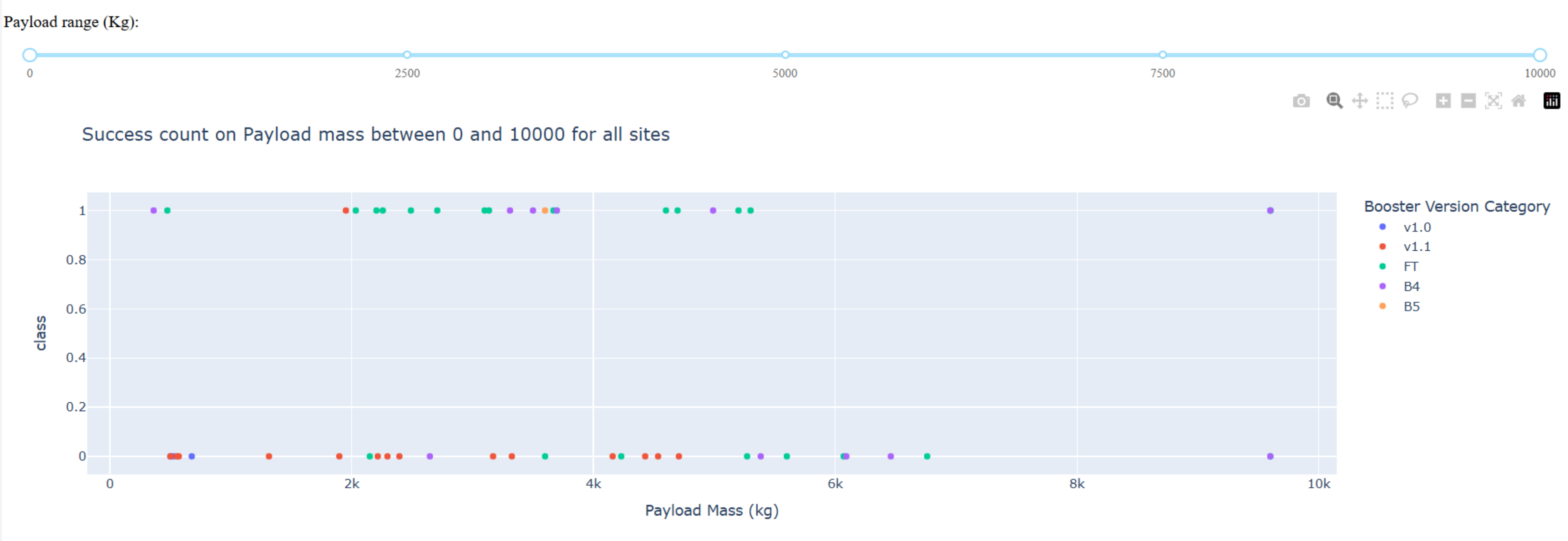
- Pie chart showing the launch success rate for all sites.
- This chart shows that launch site KSC LC-39A has the highest success at 41.7% while site CCAFS SLC-40 has the lowest success launches.

Launch site with highest success rate



- This pie chart shows the launch site of KSC LC-39A that has the highest successful launches. 76.9% of the launches were successful with a failure rate of 23.1% compared to other launch sites.

Payload vs Launch Outcome



- The payload range that has the highest success launches is between 2,000 to 4,000 kg, denoted by the most number of plots in that range, followed by the payload range of 4,000 to 6,000 kg with these 2 ranges having class (success) value 1.
- Booster version FT (green spot) has the highest successful launches followed by B4 (purple spot) version.
- Version v1.1 (red spot) has the most failure rate with class 0.

Section 5

Predictive Analysis (Classification)

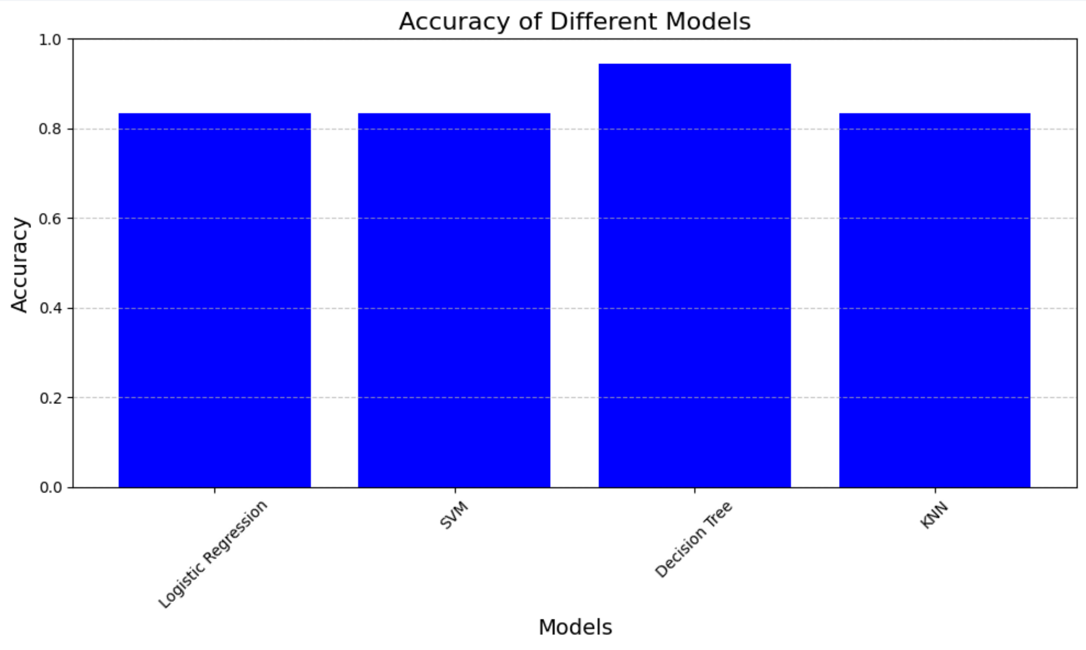
Classification Accuracy

TASK 12

Find the method performs best:

```
print('LR Accuracy:', '{:.2%}'.format(lr_accuracy_test))
print( 'SVM Accuracy:', '{:.2%}'.format(svm_accuracy_test))
print('Decision Tree Accuracy:', '{:.2%}'.format(tree_accuracy_test))
print('KNN Accuracy:', '{:.2%}'.format(knn_accuracy_test))
```

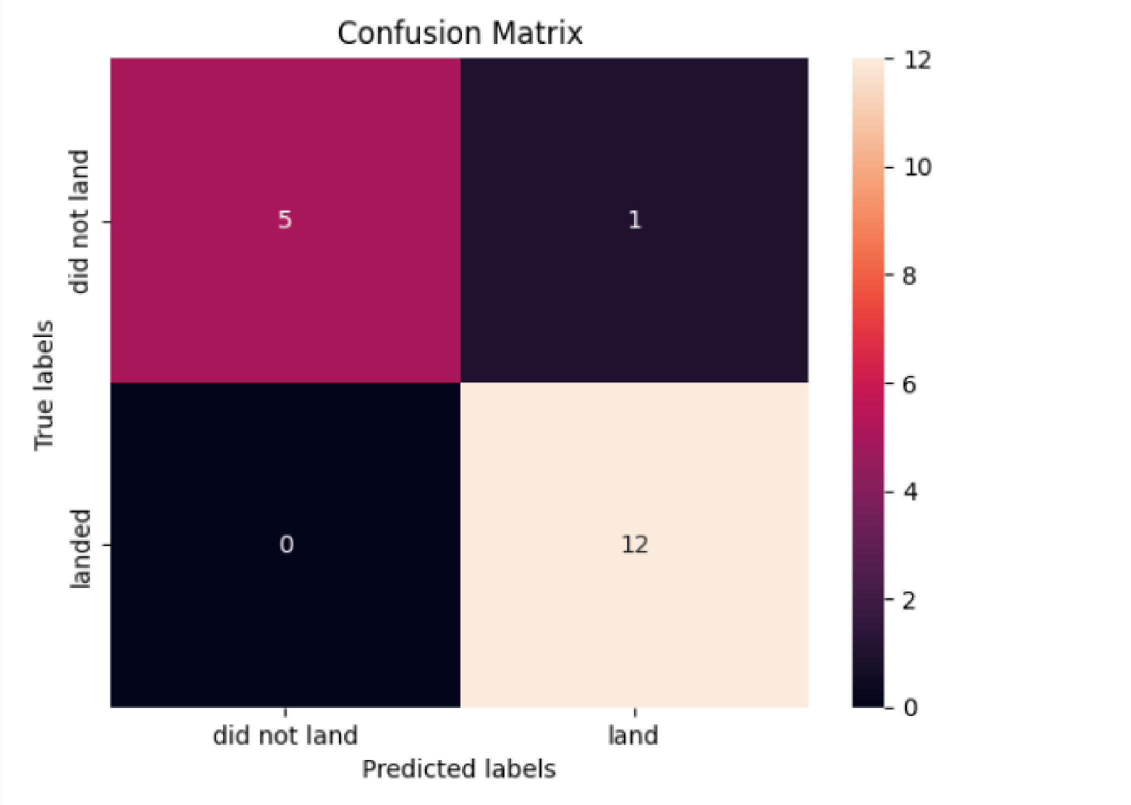
```
LR Accuracy: 83.33%
SVM Accuracy: 83.33%
Decision Tree Accuracy: 94.44%
KNN Accuracy: 83.33%
```



- The Decision Tree model has the highest accuracy of 94.44%.

Confusion Matrix

- Accuracy score is given by the formula: (TP + TN) / (TP + TN +FP + FN).
- In the case of Decision Tree model, the Accuracy score is (12 + 5) / (12 + 5 + 1 + 0) = 0.9444.
- As Accuracy is the proportion of all classifications that were correct to the total number of input samples, it can be used as one of the metric to determine a model performance.



Conclusions

- The Decision Tree model is the best performing model for predicting Space X launch outcomes.
- The lighter payloads mass have higher success in stage 1 landing compare to heavier payload mass.
- The success rate of stage 1 landing increased from 2013 till 2020, suggesting an upward trend.
- The launch site KSC LC-39A has the highest successful landing rate.
- GEO,HEO,SSO,ES L1 orbit types exhibit the highest rates of successful launches.

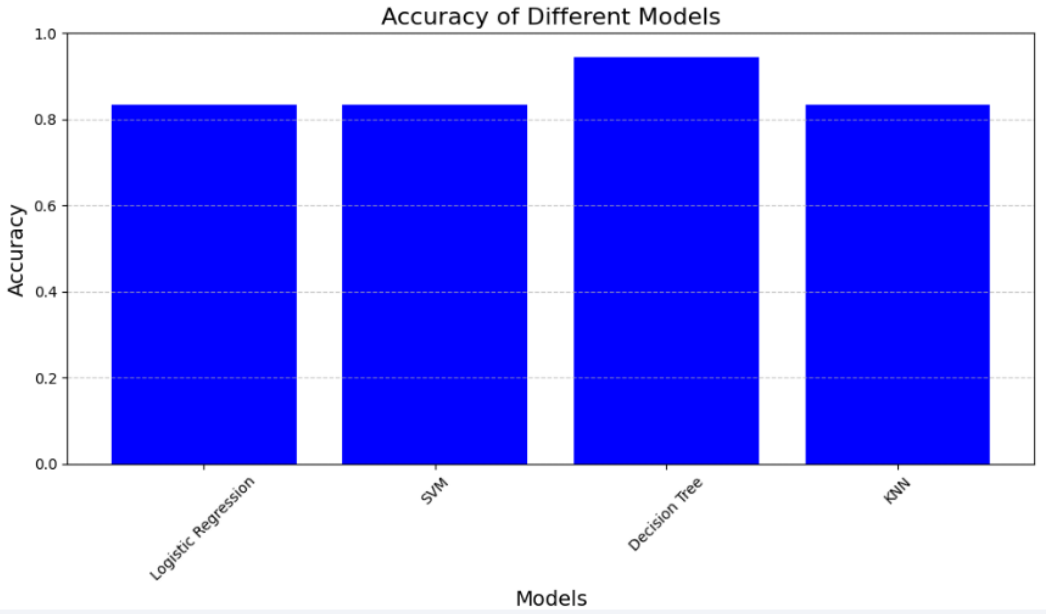
Appendix

```python
import matplotlib.pyplot as plt

# Accuracy values
accuracies = [lr_accuracy_test, svm_accuracy_test, tree_accuracy_test, knn_accuracy_test]
models = ['Logistic Regression', 'SVM', 'Decision Tree', 'KNN']

# Plotting
plt.figure(figsize=(10, 6))
plt.bar(models, accuracies, color='blue')
plt.xlabel('Models', fontsize=14)
plt.ylabel('Accuracy', fontsize=14)
plt.title('Accuracy of Different Models', fontsize=16)
plt.ylim(0, 1)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

0.1s



Sample Python codes to plot bar chart to compare
The accuracy scores for each classification model.

Thank you!