IBM Developer
SKILLS NETWORK

Winning Space Race with Data Science

Vjačeslav Usmanov
2023-05-01

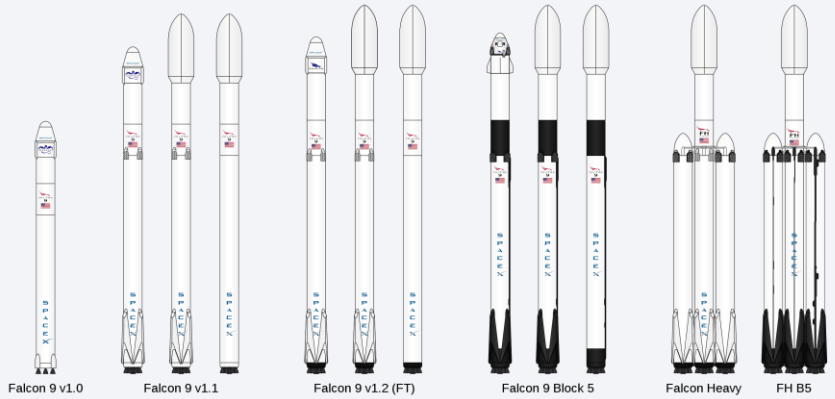
Outline

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

Executive Summary

- Summary of methodologies
  - Data Collection using SpaceX API
  - Data Collection with Web Scraping
  - Data Wrangling
  - Exploratory Data Analysis using SQL
  - EDA DataViz Using Python Pandas and Matplotlib
  - Machine Learning Landing Prediction
  - Launch Sites Analysis with Folium-Interactive Visual Analytics and PlotyDash
- Summary of all results
  - EDA results
  - Interactive Visual Analytics and Dashboards
  - Predictive Analysis (Classification)

Introduction



Project background and context

SpaceX advertises Falcon 9 rocket launches on its website with a cost of 62 million dollars; other providers cost upward of 165 million dollars each, much of the savings is because SpaceX can reuse the first stage. Therefore, if we can determine if the first stage will land, we can determine the cost of a launch. This information can be used if an alternate company wants to bid against SpaceX for a rocket launch.

Problems you want to find answers

In this capstone, we will predict if the Falcon 9 first stage will land successfully using data from Falcon 9 rocket launches advertised on its website.

Section 1

Methodology

Methodology

Executive Summary

- Data collection methodology:
  - Description of how SpaceX Falcon9 data was collected
- Perform data wrangling
  - Description of how data was wrangled
- Perform exploratory data analysis (EDA) using visualization and SQL
- Perform interactive visual analytics using Folium and Plotly Dash
- Perform predictive analysis using classification models
  - Summary of how I built, evaluated, improved, and found the best performing classification model

Data Collection

- Describe how data sets were collected

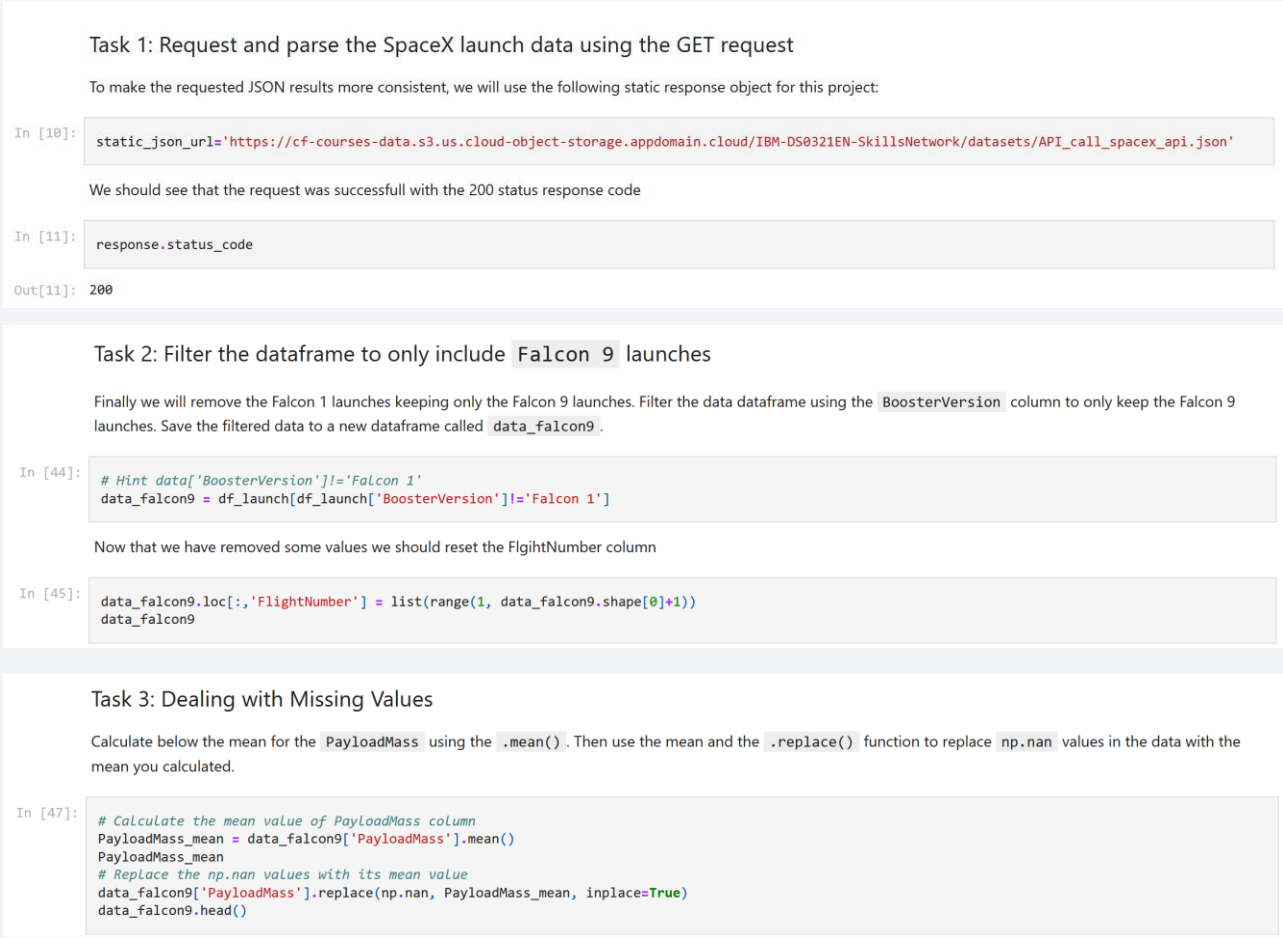
Data was first collected using SpaceX API (a RESTful API) by making a get request to the SpaceX API. This was done by first defining a series helper functions that would help in the use of the API to extract information using identification numbers in the launch data and then requesting rocket launch data from the SpaceX API url.

Finally, to make the requested JSON results more consistent, the SpaceX launch data was requested and parsed using the GET request and then decoded the response content as a Json result which was then converted into a Pandas data frame.

Also performed web scraping to collect Falcon 9 historical launch records from a Wikipedia page titled List of Falcon 9 and Falcon Heavy launches of the launch records are stored in a HTML. Using BeautifulSoup and request Libraries, I extract the Falcon 9 launch HTML table records from the Wikipedia page, Parsed the table and converted it into a Pandas data frame

Data Collection – SpaceX API

- Data collected using SpaceX API (a RESTful API) by making a get request to the SpaceX API then requested and parsed the SpaceX launch data using the GET request and decoded the response content as a Jsonresult which was then converted into a Pandas data frame
- Here is the GitHub URL of the completed SpaceX API calls notebook: https://github.com/UsmanovSla/SpaceX_Falcon_9/blob/main/001_jupyter_labs_spacex_data_collection_api.ipynb



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:

In [10]:
```
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

In [11]:
```
response.status_code
```

Out[11]: 200

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`.

In [44]:
```
# Hint data['BoosterVersion']!='Falcon 1'
data_falcon9 = df_launch[df_launch['BoosterVersion']!='Falcon 1']
```

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

In [45]:
```
data_falcon9.loc[:,'FlightNumber'] = list(range(1, data_falcon9.shape[0]+1))
data_falcon9
```

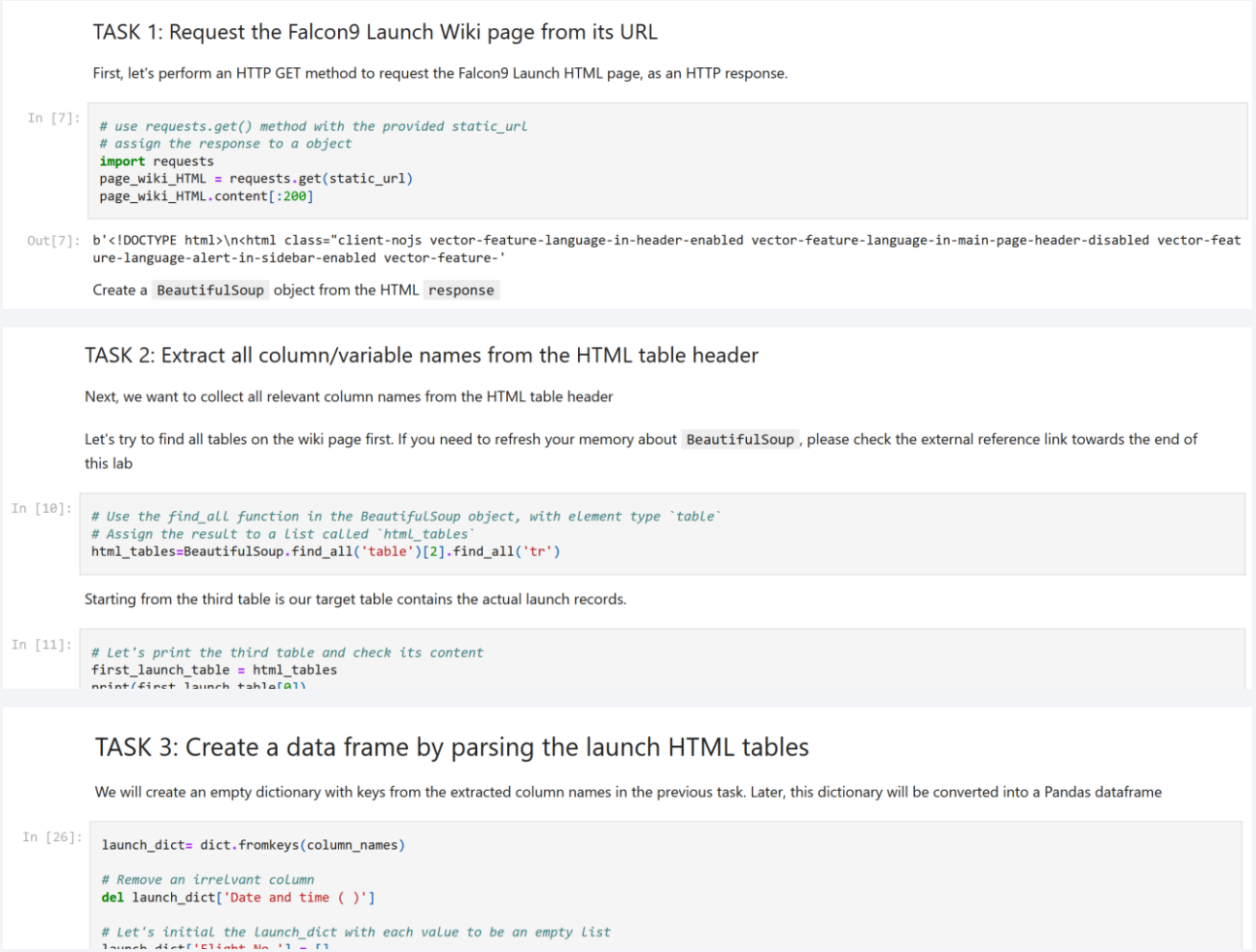
Task 3: Dealing with Missing Values

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.

In [47]:
```
# Calculate the mean value of PayloadMass column
PayloadMass_mean = data_falcon9['PayloadMass'].mean()
PayloadMass_mean
# Replace the np.nan values with its mean value
data_falcon9['PayloadMass'].replace(np.nan, PayloadMass_mean, inplace=True)
data_falcon9.head()
```

Data Collection - Scraping

- Performed web scraping to collect Falcon 9 historical launch records from a Wikipedia using BeautifulSoup and request, to extract the Falcon 9 launch records from HTML table of the Wikipedia page, then created a data frame by parsing the launch HTML.
- Here is the GitHub URL of the completed web scraping notebook: https://github.com/UsmanovSla/Space_X_Falcon_9/blob/main/002_jupyter_labs_spacex_webscraping.ipynb



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.

In [7]:
```
# use requests.get() method with the provided static_url
# assign the response to a object
import requests
page_wiki_HTML = requests.get(static_url)
page_wiki_HTML.content[:200]
```

Out[7]:
```
b'<!DOCTYPE html>\n<html class="client-nojs vector-feature-language-in-header-enabled vector-feature-language-in-main-page-header-disabled vector-feature-language-alert-in-sidebar-enabled vector-feature-'
```

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

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

In [10]:
```
# Use the find_all function in the BeautifulSoup object, with element type `table`
# Assign the result to a list called `html_tables`
html_tables=BeautifulSoup.find_all('table')[2].find_all('tr')
```

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

In [11]:
```
# Let's print the third table and check its content
first_launch_table = html_tables
```

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

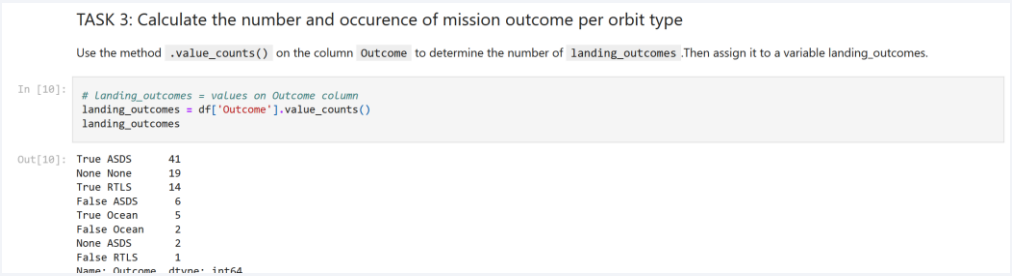
In [26]:
```
launch_dict= dict.fromkeys(column_names)

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

# Let's initial the launch_dict with each value to be an empty list
```

Data Wrangling

- After obtaining and creating a Pandas DF from the collected data, data was filtered using the BoosterVersion column to only keep the Falcon 9 launches, then dealt with the missing data values in the LandingPad and PayloadMass columns. For the PayloadMass, missing data values were replaced using mean value of column.
- Also performed some Exploratory Data Analysis (EDA) to find some patterns in the data and determine what would be the label for training supervised models
- Here is the GitHub URL of completed data wrangling related notebooks: https://github.com/UsmanovSla/Space_X_Falcon_9/blob/main/003_jupyter_labs_spacex_data_wrangling.ipynb

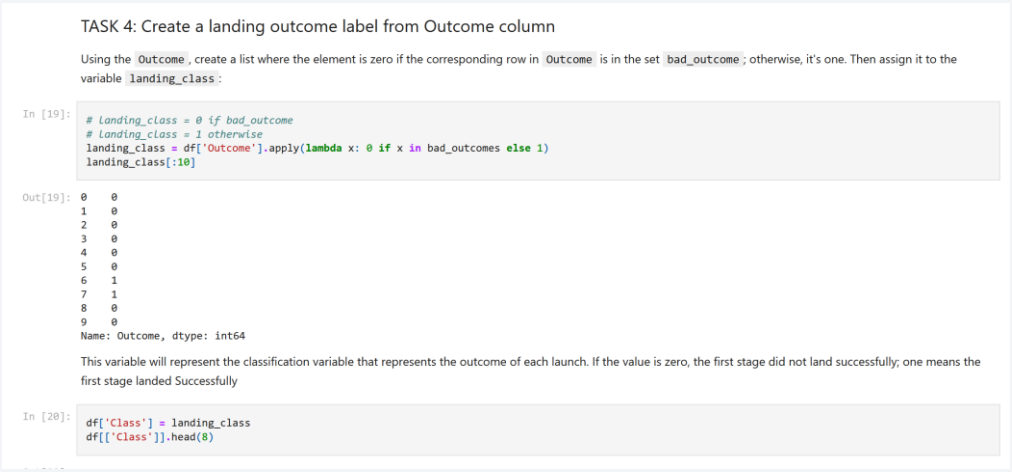


TASK 3: Calculate the number and occurence of mission outcome per orbit type

Use the method `.value_counts()` on the column `Outcome` to determine the number of `landing_outcomes`.Then assign it to a variable landing_outcomes.

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

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



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`:

```
In [19]:
# landing_class = 0 if bad_outcome
# landing_class = 1 otherwise
landing_class = df['Outcome'].apply(lambda x: 0 if x in bad_outcomes else 1)
landing_class[:10]
```

```
Out[19]: 0    0
1    0
2    0
3    0
4    0
5    0
6    1
7    1
8    0
9    0
Name: Outcome, dtype: int64
```

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

```
In [20]:
df['Class'] = landing_class
df[['Class']].head(8)
```

EDA with Data Visualization

- Performed data Analysis and Feature Engineering using Pandas and Matplotlib
  - Exploratory Data Analysis
  - Preparing Data Feature Engineering
- Used scatter plots to Visualize the relationship between Flight Number and Launch Site, Payload and Launch Site, FlightNumber and Orbit type, Payload and Orbit type.
- Used Bar chart to Visualize the relationship between success rate of each orbit type
- Line plot to Visualize the launch success yearly trend. Add the GitHub URL of your completed EDA with data visualization notebook, as an external reference and peer-review purpose
- Here is the GitHub URL of completed EDA with data visualization notebook: https://github.com/UsmanovSla/Space_X_Falcon_9/blob/main/005_jupyter_labs_EDA_dataviz.ipynb

EDA with SQL

- The following SQL queries were performed for EDA
- Here is the GitHub URL of completed EDA with SQL notebook: https://github.com/UsmanovSla/Space_X_Falcon_9/blob/main/004_jupyter_labs_EDA_sql_sqllite.ipynb

Task 1

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

```
[ ]: %sql SELECT DISTINCT Launch_Site FROM SPACEXTBL;
```

Task 2

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

```
[ ]: %sql SELECT * FROM SPACEXTBL WHERE Launch_Site LIKE 'CCA%' LIMIT 5;
```

Task 3

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

```
[ ]: %sql SELECT SUM(PAYLOAD_MASS__KG_) AS  Payload_Mass FROM SPACEXTBL WHERE Customer = 'NASA (CRS)';
```

Task 4

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

```
[ ]: %%sql SELECT AVG(PAYLOAD_MASS__KG_) AS Avg_Payload_Mass FROM SPACEXTBL WHERE Booster_Version LIKE 'F9 v1.1%';

SELECT * FROM PRAGMA_TABLE_INFO('SPACEXTBL');

UPDATE SPACEXTBL SET date_ = substr(date, 7, 4) || '-' || substr(date, 4, 2) || '-' || substr(date, 1, 2);
```

EDA with SQL

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)';
```

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 DISTINCT(Booster_Version) FROM SPACEXTBL WHERE PAYLOAD_MASS__KG_ >= 4000 AND PAYLOAD_MASS__KG_ <= 6000 AND Landing_Outcome LIKE 'Success
```

Task 7

List the total number of successful and failure mission outcomes

```
[ ]: %sql SELECT MISSION_OUTCOME, COUNT(MISSION_OUTCOME) AS Mission_Outcomes FROM SPACEXTBL GROUP BY MISSION_OUTCOME;
```

Task 8

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

```
[ ]: %sql SELECT DISTINCT(Booster_Version) FROM SPACEXTBL \
WHERE PAYLOAD_MASS__KG_ = (SELECT MAX(PAYLOAD_MASS__KG_) FROM SPACEXTBL);
```

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, 4, 2) as month to get the months and substr(Date,7,4)='2015' for year.

[ ]:
```
%sql SELECT substr ("--JanFebMarAprMayJunJulAugSepOctNovDec", strftime ("%m", date_) * 3, 3) AS "Month/2015", BOOSTER_VERSION, LAUNCH_SITE, LANDING_O
    WHERE (LANDING_OUTCOME = 'Failure (drone ship)') AND date_ >= '2015-01-01' AND date_ <= '2015-12-31';
```

Task 10

Rank the count of successful landing_outcomes between the date 04-06-2010 and 20-03-2017 in descending order.

[ ]:
```
%sql SELECT Landing_Outcome, COUNT(Landing_Outcome) AS Successful_Landing FROM SPACEXTBL \
    WHERE date_ BETWEEN '2010-06-04' AND '2017-03-20' \
    GROUP BY Landing_Outcome \
    ORDER BY Successful_Landing DESC;
```

Build an Interactive Map with Folium

- Created folium map to marked all the launch sites, and created map objects such as markers, circles, lines to mark the success or failure of launches for each launch site.
- Created a launch set outcomes (failure=0 or success=1).
- Here is the GitHub URL of completed interactive map with Folium map: https://github.com/UsmanovSla/Space_X_Falcon_9/blob/main/006_jupyter_labs_launch_site_location.ipynb

Build a Dashboard with Plotly Dash

- Built an interactive dashboard application with Plotly dash by:
  - Adding a Launch Site Drop-down Input Component
  - Adding a callback function to render success-pie-chart based on selected site dropdown
  - Adding a Range Slider to Select Payload
  - Adding a callback function to render the success-payload-scatter-chart scatter plot
- Here is the GitHub URL of completed Plotly Dash lab: https://github.com/UsmanovSla/Space_X_Falcon_9/blob/main/008_python_spacex_dash_app.py

```
                                        min=0,
                                        max=10000,
                                        step=1000,
                                        value=[min_payload, max_payload]),
                                # 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 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='Success Count for all launch sites')
        return fig
    else:
        filtered_df = spacex_df[spacex_df['Launch Site'] == entered_site]
        filtered_df = filtered_df.groupby(['Launch Site', 'class']).size().reset_index(name='class count')
        fig = px.pie(filtered_df, values='class count', names='class', title=f"Total Success Launches for site {entered_site}")
        return fig

# 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 scatter(entered_site, payload):
    filtered_df = spacex_df[spacex_df['Payload Mass (kg)'].between(payload[0], payload[1])]
    # thought reusing filtered_df may cause issues, but tried it out of curiosity and it seems to be working fine
```

Predictive Analysis (Classification)

- Summary of how I built, evaluated, improved, and found the best performing classification model
- After loading the data as a Pandas Dataframe, I set out to perform exploratory Data Analysis and determine Training Labels by:
  - creating a NumPy array from the column Class in data, by applying the method to_numpy() then assigned it to the variable Y as the outcome variable.
  - Then standardized the feature dataset (x) by transforming it using preprocessing. StandardScaler() function from Sklearn.
  - After which the data was split into training and testing sets using the function train_test_split from sklearn.model_selection with the test_size parameter set to 0.2 and random_state to 2.
- Here is the GitHub URL of completed predictive analysis lab: https://github.com/UsmanovSla/Space_X_Falcon_9/blob/main/007_jupyter_labs_Machine_Learning_Prediction.ipynb

Predictive Analysis (Classification)

- In order to find the best ML model/ method that would performs best using the test data between SVM, Classification Trees, k nearest neighbors and Logistic Regression:
  - First created an object for each of the algorithms then created a GridSearchCV object and assigned them a set of parameters for each model.
  - For each of the models under evaluation, the GridsearchCV object was created with cv=10, then fit the training data into the GridSearch object for each to Find best Hyperparameter.
  - After fitting the training set, we output GridSearchCV object for each of the models, then displayed the best parameters using the data attribute best_params_ and the accuracy on the validation data using the data attribute best_score_.
  - Finally using the method score to calculate the accuracy on the test data for each model and plotted a confussion matrix for each using the test and predicted outcomes.

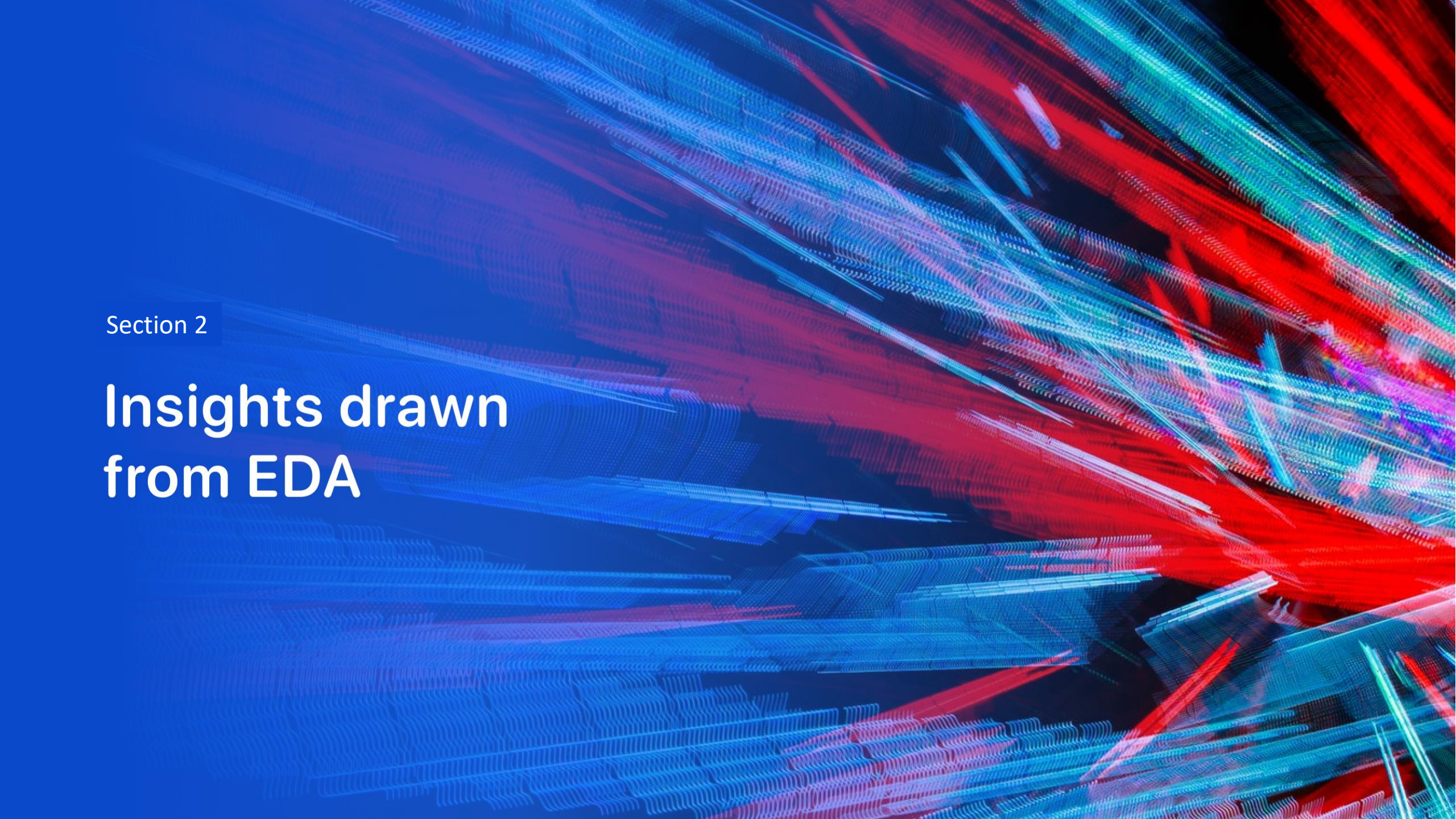
Results

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

The table below shows the test data accuracy score for each of the methods comparing them to show which performed best using the test data between SVM, Classification Trees, k nearest neighbors and Logistic Regression

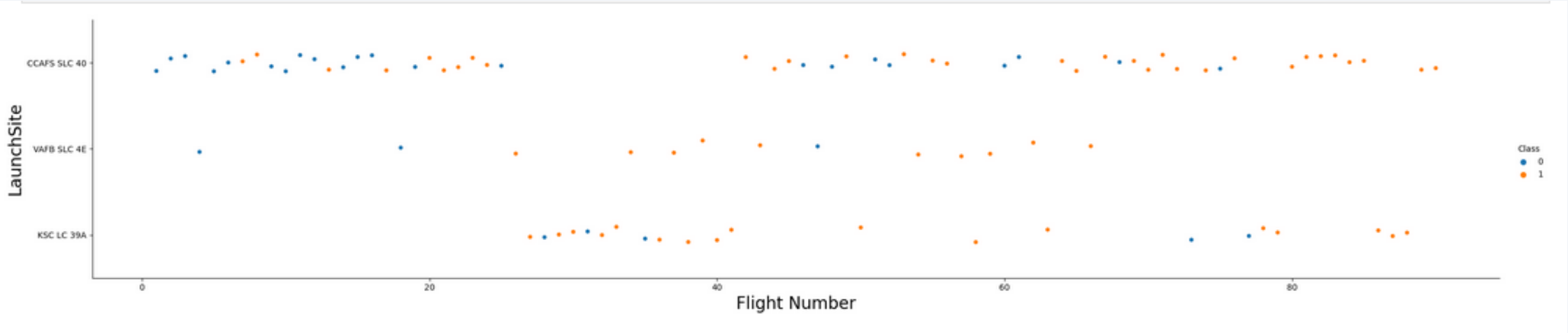
Out[68]:

	0
Method	Test Data Accuracy
Logistic_Reg	0.833333
SVM	0.833333
Decision Tree	0.833333
KNN	0.833333


Section 2
Insights drawn from EDA

Flight Number vs. Launch Site

- Show a scatter plot of Flight Number vs. Launch Site



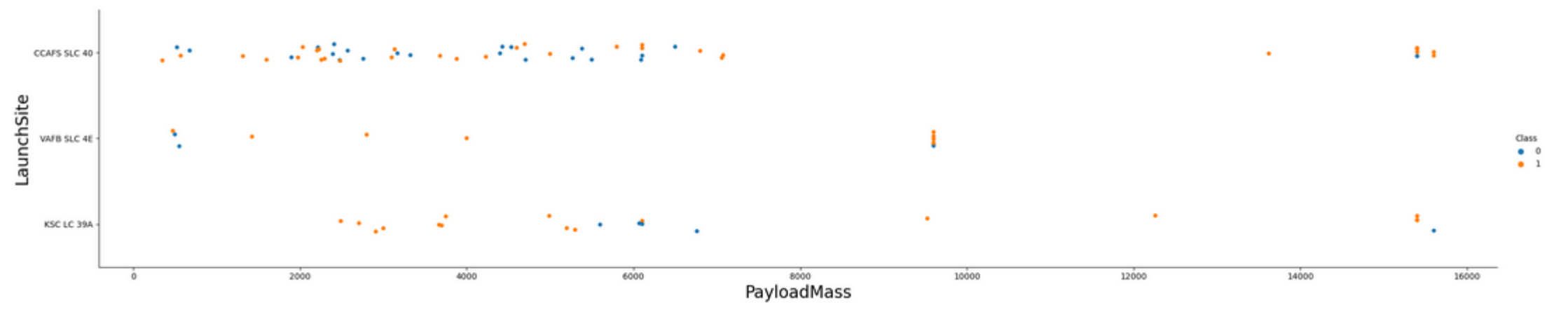
- Show the screenshot of the scatter plot with explanations

Now try to explain the patterns you found in the Flight Number vs. Launch Site scatter point plots.

We can deduce that, as the flight number increases in each of the 3 launcg sites, so does the success rate. For instance, the success rate for the VAFB SLC 4E launch site is 100% after the Flight number 50. Both KSC LC 39A and CCAFS SLC 40 have a 100% success rates after 80th flight.

Payload vs. Launch Site

- Show a scatter plot of Payload vs. Launch Site



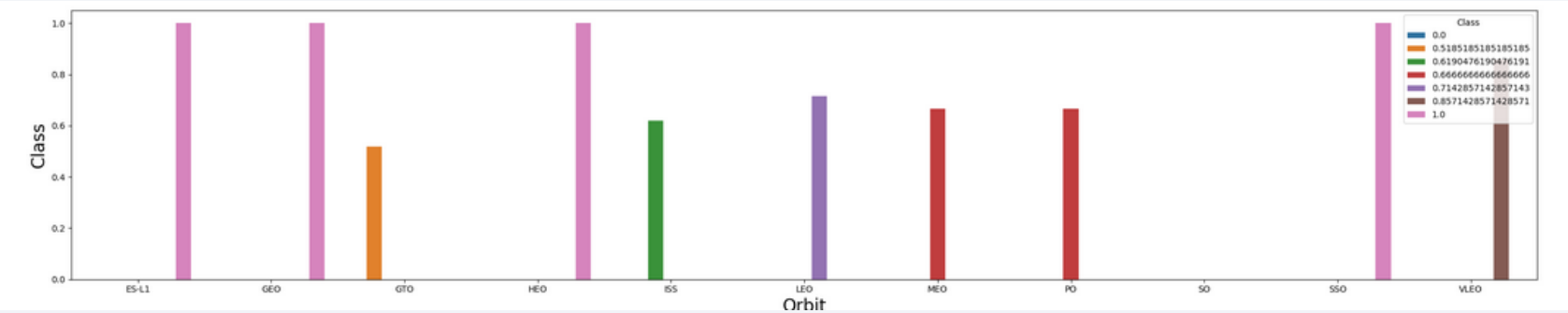
- Show the screenshot of the scatter plot with explanations

```
Index(['FlightNumber', 'Date', 'BoosterVersion', 'PayloadMass', 'Orbit',
       'LaunchSite', 'Outcome', 'Flights', 'GridFins', 'Reused', 'Legs',
       'LandingPad', 'Block', 'ReusedCount', 'Serial', 'Longitude', 'Latitude',
       'Class'],
      dtype='object')
```

Now if you observe Payload Vs. Launch Site scatter point chart you will find for the VAFB-SLC launchsite there are no rockets launched for heavypayload mass(greater than 10000).

Success Rate vs. Orbit Type

- Show a bar chart for the success rate of each orbit type



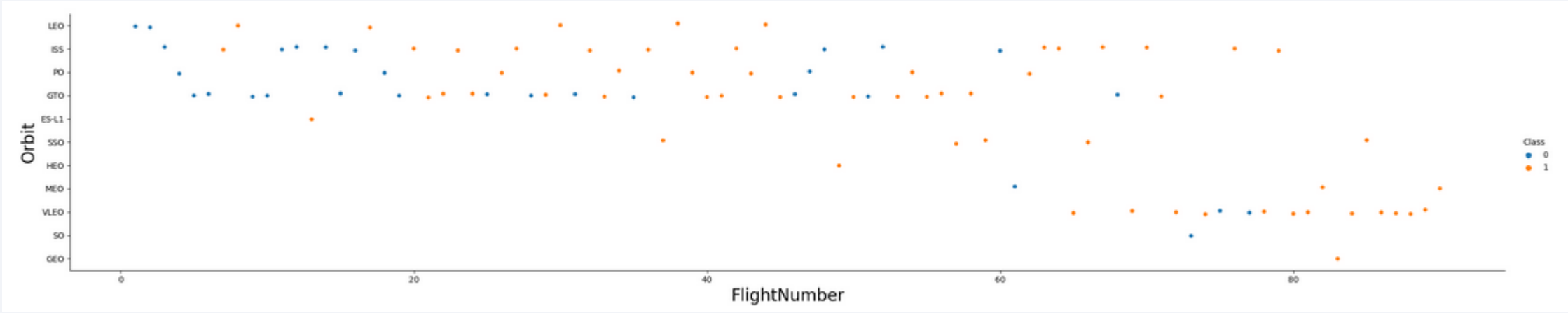
- Show the screenshot of the scatter plot with explanations

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

Orbits ES-L1, GEO, HEO & SSO have the highest success rates at 100%, with SO orbit having the lowest success rate at ~50%. Orbit SO has 0% success rate.

Flight Number vs. Orbit Type

- Show a scatter point of Flight number vs. Orbit type

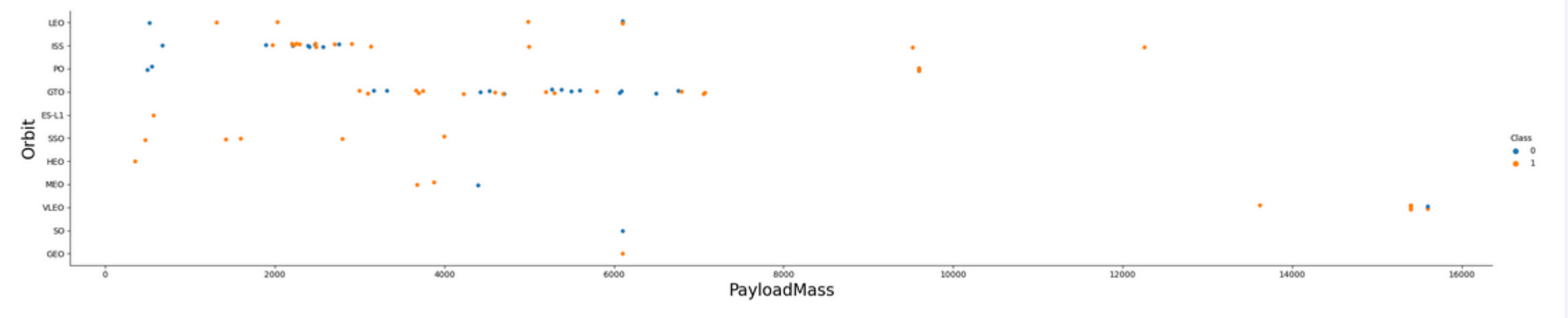


- Show the screenshot of the scatter plot with explanations

You should see that 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.

Payload vs. Orbit Type

- Show a scatter point of payload vs. orbit type



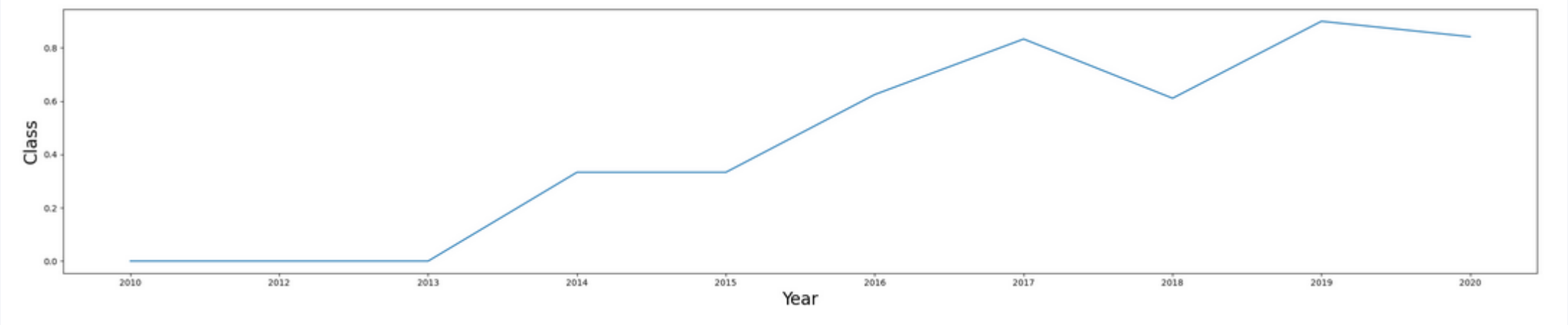
- Show the screenshot of the scatter plot with explanations

With heavy payloads the successful landing or positive landing rate are more for Polar,LEO and ISS.

However for GTO we cannot distinguish this well as both positive landing rate and negative landing(unsuccessful mission) are both there here.

Launch Success Yearly Trend

- Show a line chart of yearly average success rate



- Show the screenshot of the scatter plot with explanations

you can observe that the sucess rate since 2013 kept increasing till 2020

All Launch Site Names

- Find the names of the unique launch sites

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

- Used ‘SELECT DISTINCT’ statement to return only the unique launch sites from the ‘LAUNCH_SITE’ column of the SPACEXTBL table:

```
%sql SELECT DISTINCT Launch_Site FROM SPACEXTBL;
```

Launch Site Names Begin with 'CCA'

- Find 5 records where launch sites begin with `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
04-06-2010	18:45:00	F9 v1.0 B0003	CCAFS LC-40	Dragon Spacecraft Qualification Unit	0	LEO	SpaceX	Success	Failure (parachute)
08-12-2010	15:43:00	F9 v1.0 B0004	CCAFS LC-40	Dragon demo flight C1, two CubeSats, barrel of Brouere cheese	0	LEO (ISS)	NASA (COTS) NRO	Success	Failure (parachute)
22-05-2012	07:44:00	F9 v1.0 B0005	CCAFS LC-40	Dragon demo flight C2	525	LEO (ISS)	NASA (COTS)	Success	No attempt
08-10-2012	00:35:00	F9 v1.0 B0006	CCAFS LC-40	SpaceX CRS-1	500	LEO (ISS)	NASA (CRS)	Success	No attempt
01-03-2013	15:10:00	F9 v1.0 B0007	CCAFS LC-40	SpaceX CRS-2	677	LEO (ISS)	NASA (CRS)	Success	No attempt

- Used ‘LIKE’ command with ‘%’ wildcard in ‘WHERE’ clause to select and dispay a table of all records where launch sites begin with the string 'CCA'

Total Payload Mass

- Calculate the total payload carried by boosters from NASA

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

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

Payload_Mass
45596

- Used the ‘SUM()’ function to return and dispaly the total sum of ‘PAYLOAD_MASS_KG’ column for Customer ‘NASA(CRS’

Average Payload Mass by F9 v1.1

- Calculate the average payload mass carried by booster version F9 v1.1

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

Payload Mass Kgs	Customer	Booster_Version
2534.6666666666665	MDA	F9 v1.1 B1003

- Used the 'AVG()' function to return and dispaly the average payload mass carried by booster version F9 v1.1

First Successful Ground Landing Date

- Find the dates of the first successful landing outcome on ground pad

```
UPDATE SPACEXTBL SET date_ = substr(date, 7, 4) || '-' || substr(date, 4, 2) || '-' || substr(date, 1, 2);
```

```
%sql SELECT MIN(date_) FROM SPACEXTBL WHERE LANDING_OUTCOME = 'Success (ground pad)';
```

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

MIN(date_)
2015-12-22

- Used the 'MIN()' function to return and dispaly the first (oldest) date when first successful landing outcome on ground pad 'Success (ground pad)'happened.

Successful Drone Ship Landing with Payload between 4000 and 6000

- List the names of boosters which have successfully landed on drone ship and had payload mass greater than 4000 but less than 6000

```
%sql SELECT DISTINCT(Booster_Version) FROM SPACEXTBL WHERE PAYLOAD_MASS__KG_ >= 4000 AND PAYLOAD_MASS__KG_ <= 6000 AND Landing_Outcome LIKE 'Success (dro
 * sqlite:///my_data1.db
Done.
```

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

- Used ‘Select Distinct’ statement to return and list the ‘unique’ names of boosters with operators >4000 and <6000 to only list booster with payloads btween 4000-6000 with landing outcome of ‘Success (drone ship)’.

Total Number of Successful and Failure Mission Outcomes

- Calculate the total number of successful and failure mission outcomes

```
%sql SELECT MISSION_OUTCOME, COUNT(MISSION_OUTCOME) AS Mission_Outcomes FROM SPACEXTBL GROUP BY MISSION_OUTCOME;
```

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

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

- Used the ‘COUNT()’ together with the ‘GROUP BY’ statement to return total number of missions outcomes

Boosters Carried Maximum Payload

- List the names of the booster which have carried the maximum payload mass

```
%sql SELECT DISTINCT(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
F9 B5 B1060.2
F9 B5 B1058.3
F9 B5 B1051.6
F9 B5 B1060.3
F9 B5 B1049.7

- Using a Subquerry to return and pass the Max payload and used it list all the boosters that have carried the Max payload of 15600kgs

2015 Launch Records

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

```
%sql SELECT substr ("--JanFebMarAprMayJunJulAugSepOctNovDec", strftime ("%m", date_) * 3, 3) AS "Month/2015", BOOSTER_VERSION, LAUNCH_SITE, LANDING_OUTCO
    WHERE (LANDING_OUTCOME = 'Failure (drone ship)') AND date_ >= '2015-01-01' AND date_ <= '2015-12-31';
```

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

Month/2015	Booster_Version	Launch_Site	Landing_Outcome
Jan	F9 v1.1 B1012	CCAFS LC-40	Failure (drone ship)
Apr	F9 v1.1 B1015	CCAFS LC-40	Failure (drone ship)

- Used the ‘subsrt()’ in the select statement to get the month and year from the date column where substr(Date,7,4)='2015' for year and Landing_outcome was ‘Failure (drone ship’) and return the records nmatching the filter.

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

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

```
%sql SELECT Landing_Outcome, COUNT(Landing_Outcome) AS Successful_Landing FROM SPACEXTBL \
    WHERE date_ BETWEEN '2010-06-04' AND '2017-03-20' \
    GROUP BY Landing_Outcome \
    ORDER BY Successful_Landing DESC;
```

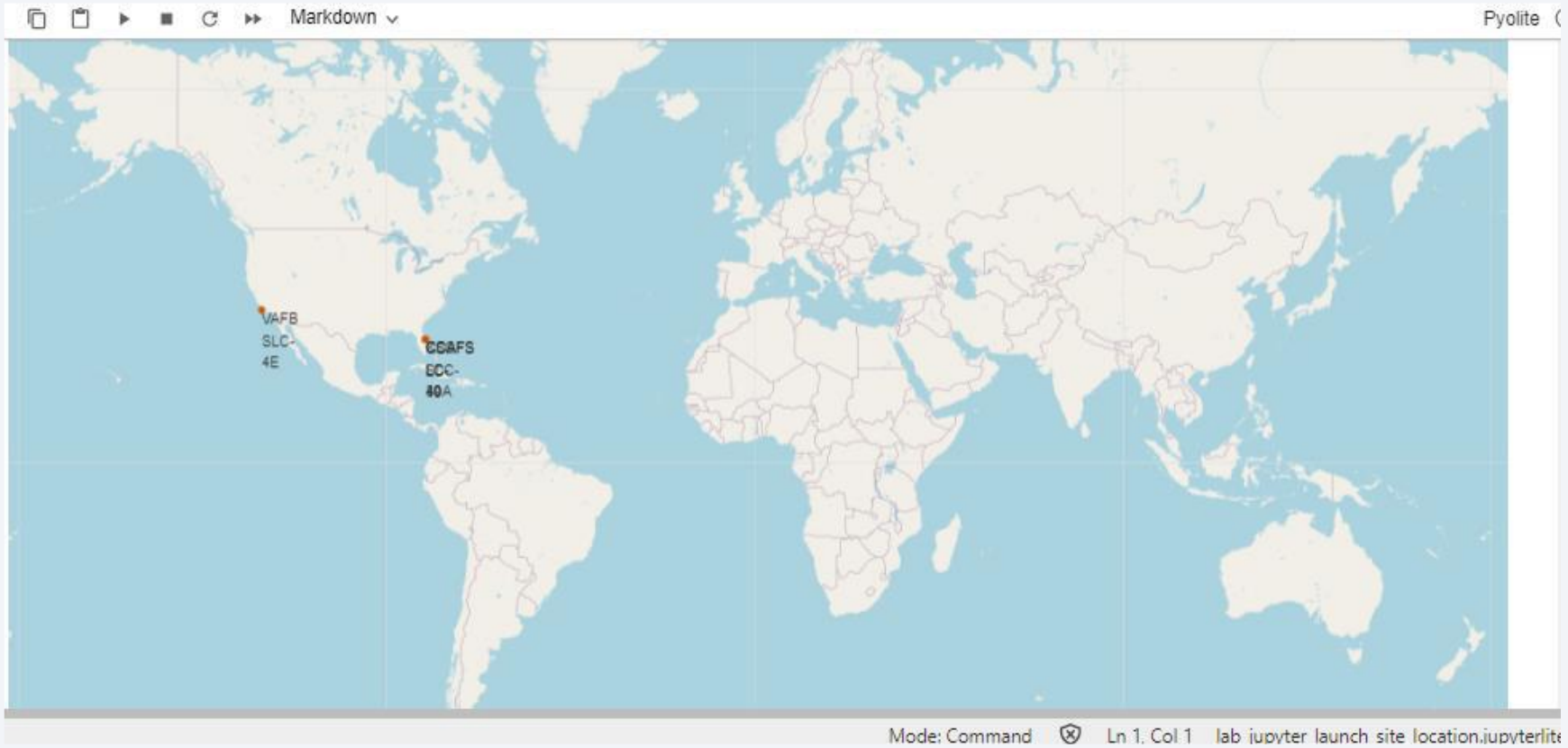
```
 * sqlite:///my_data1.db
Done.
```

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

Section 3

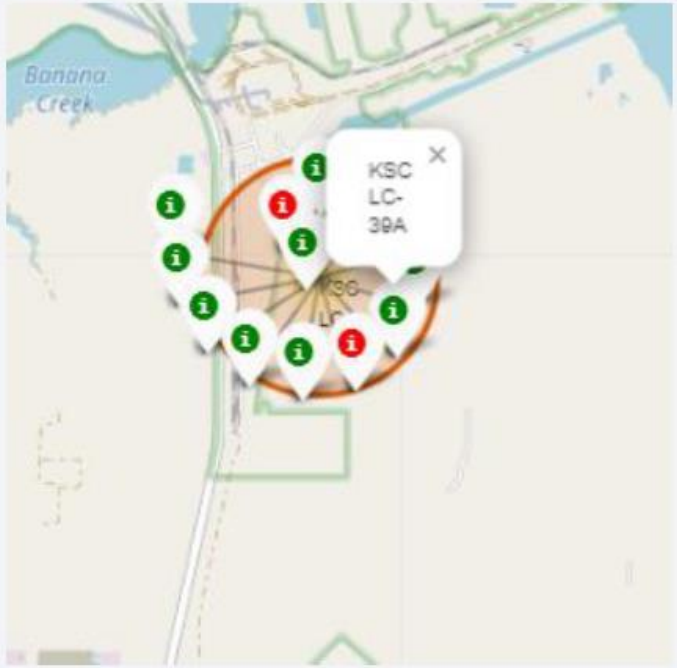
Launch Sites Proximities Analysis

Markers of all launch sites on global map



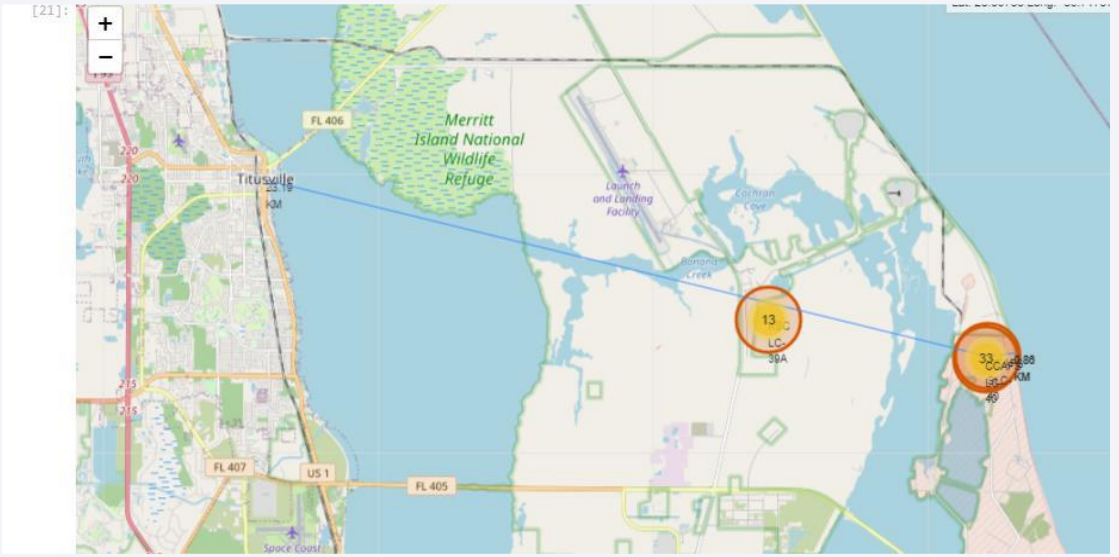
- All launch sites are in proximity to the Equator, (located southwards of the US map). Also all the launch sites are in very close proximity to the coast.

Launch outcomes for each site on the map With Color Markers

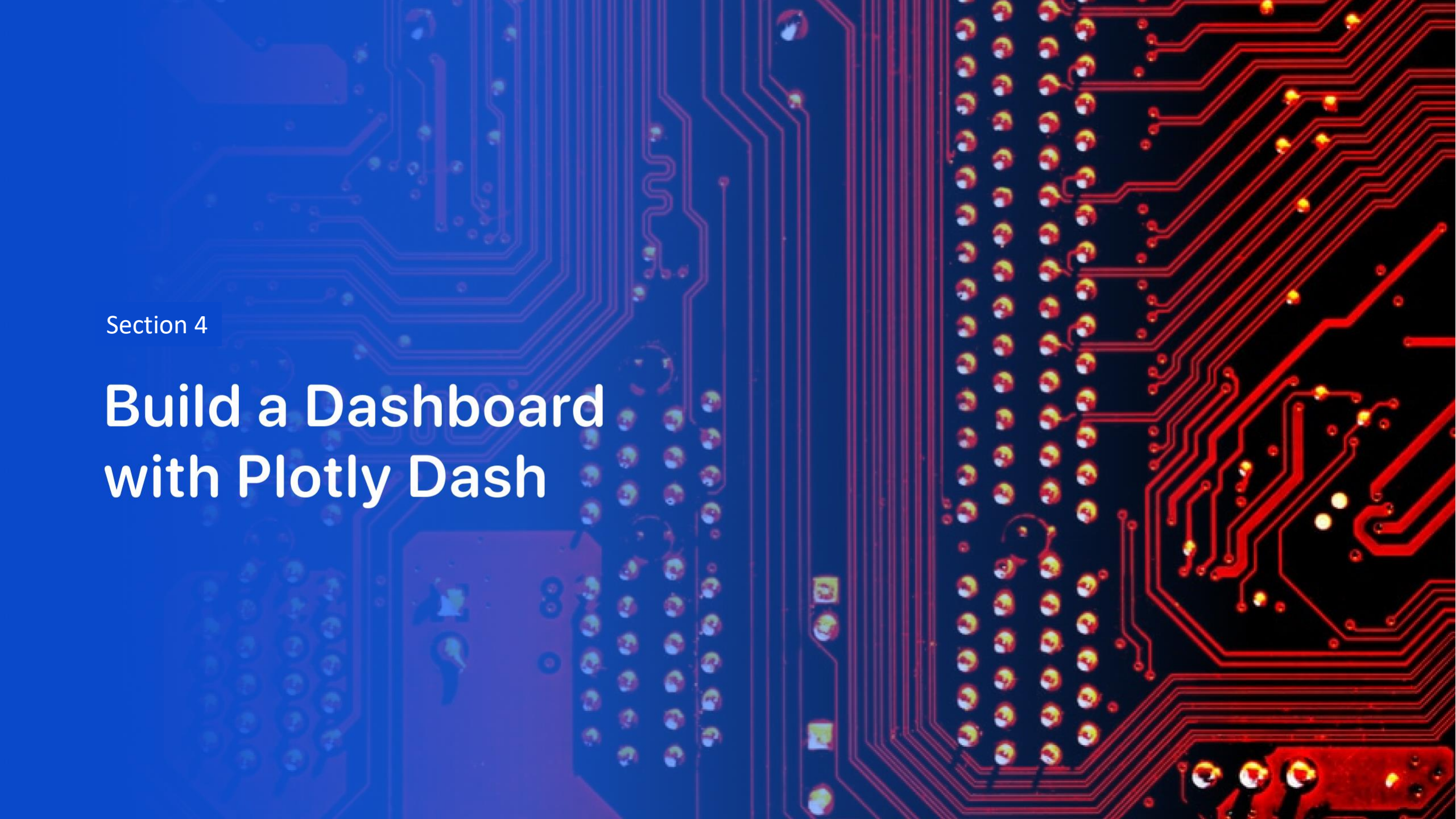


- In the Eastern coast (Florida) Launch site KSC LC-39A has relatively high success rates compared to CCAFS SLC-40 & CCAFS LC-40.

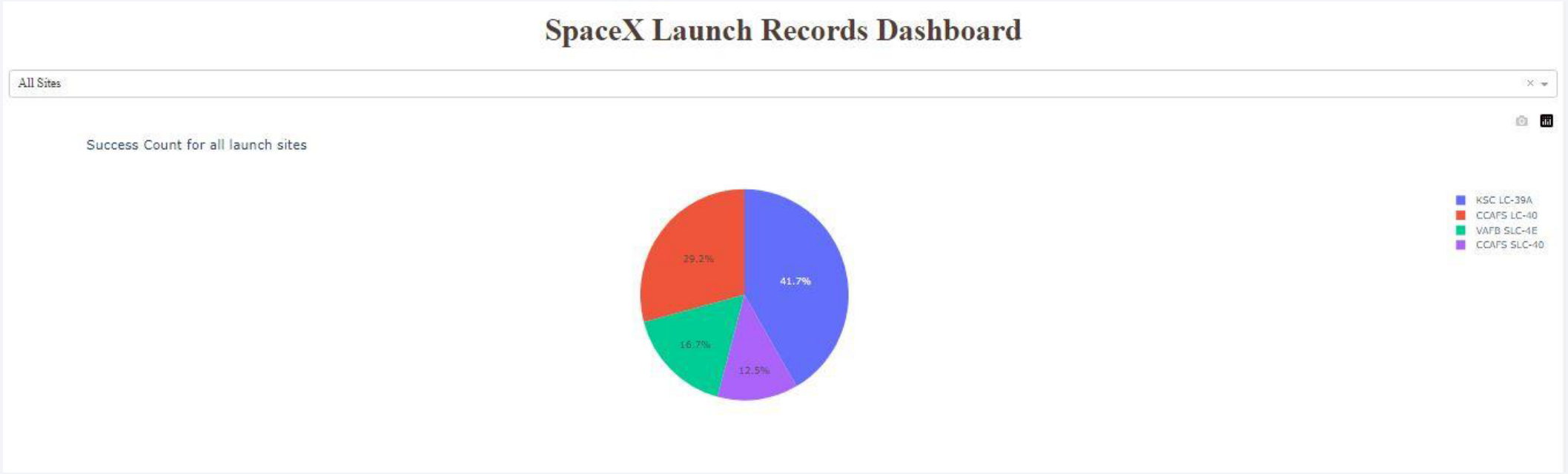
Launch outcomes for each site on the map With Color Markers



- Launch siteCCAFS SLC-40proximity to coastline is 0.86km
- Launch siteCCAFS SLC-40closest to highway (Washington Avenue) is 23.19km

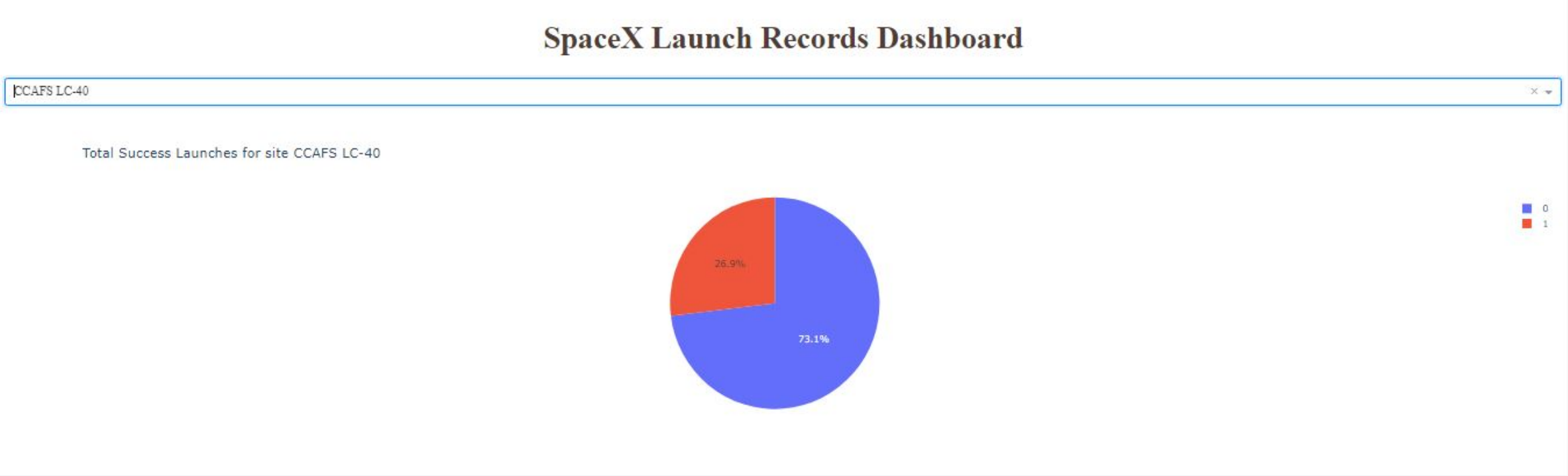

Section 4
Build a Dashboard with Plotly Dash

Pie-Chart for launch success count for all sites



- Launch site KSC LC-39A has the highest launch success rate at 42% followed by CCAFS LC-40 at 29%, VAFB SLC-4E at 17% and lastly launch site CCAFS SLC-40 with a success rate of 13%

Pie chart for the launch site with highest launch success ratio



- Launch site CCAFS LC-40 had the 2ndhighest success ratio of 73% success against 27% failed launches

<Dashboard Screenshot 3>

- For Launch site CCAFS LC-40 the booster version FT has the largest success ratefrom a payload mass of >2000kg

Section 5

Predictive Analysis (Classification)

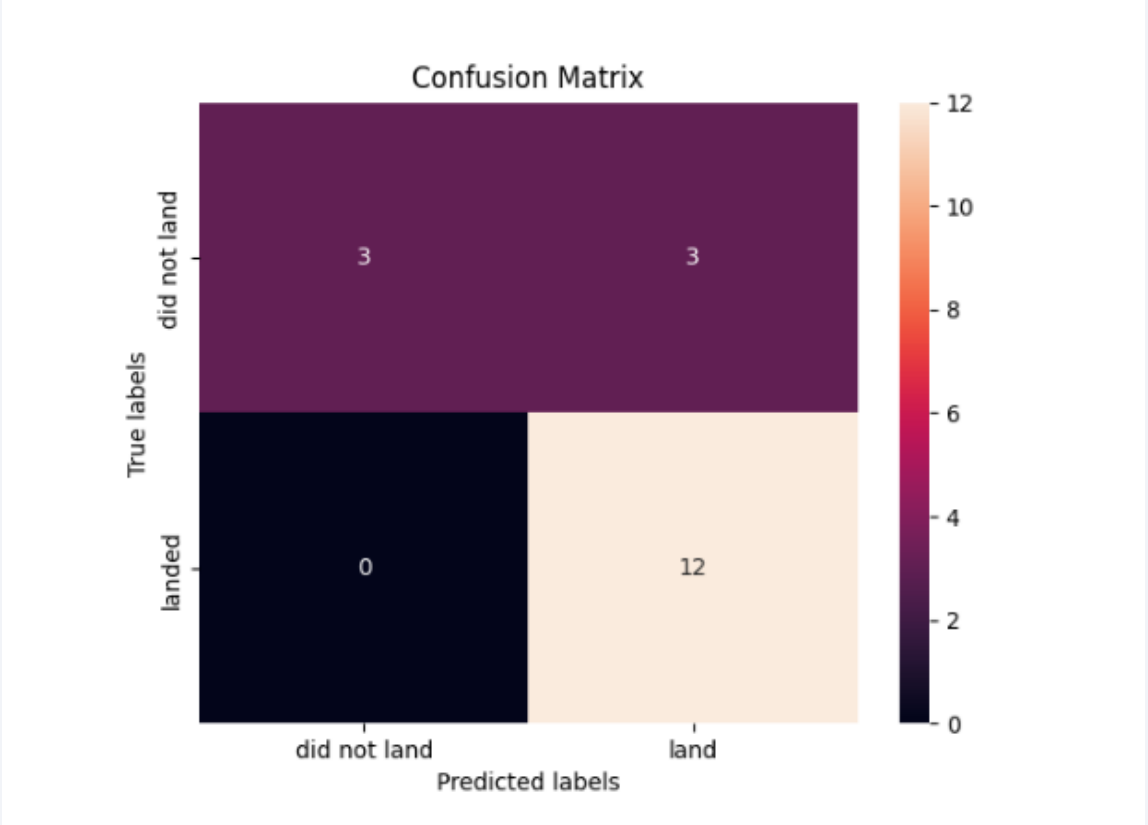
Classification Accuracy

	0
Method	Test Data Accuracy
Logistic_Reg	0.833333
SVM	0.833333
Decision Tree	0.833333
KNN	0.833333

All the methods perform equally on the test data: i.e. They all have the same accuracy of 0.833333 on the test Data

Confusion Matrix

- All the 4 classification model had the same confusion matrixes and were able equally distinguish between the different classes. The major problem is false positives for all the models.

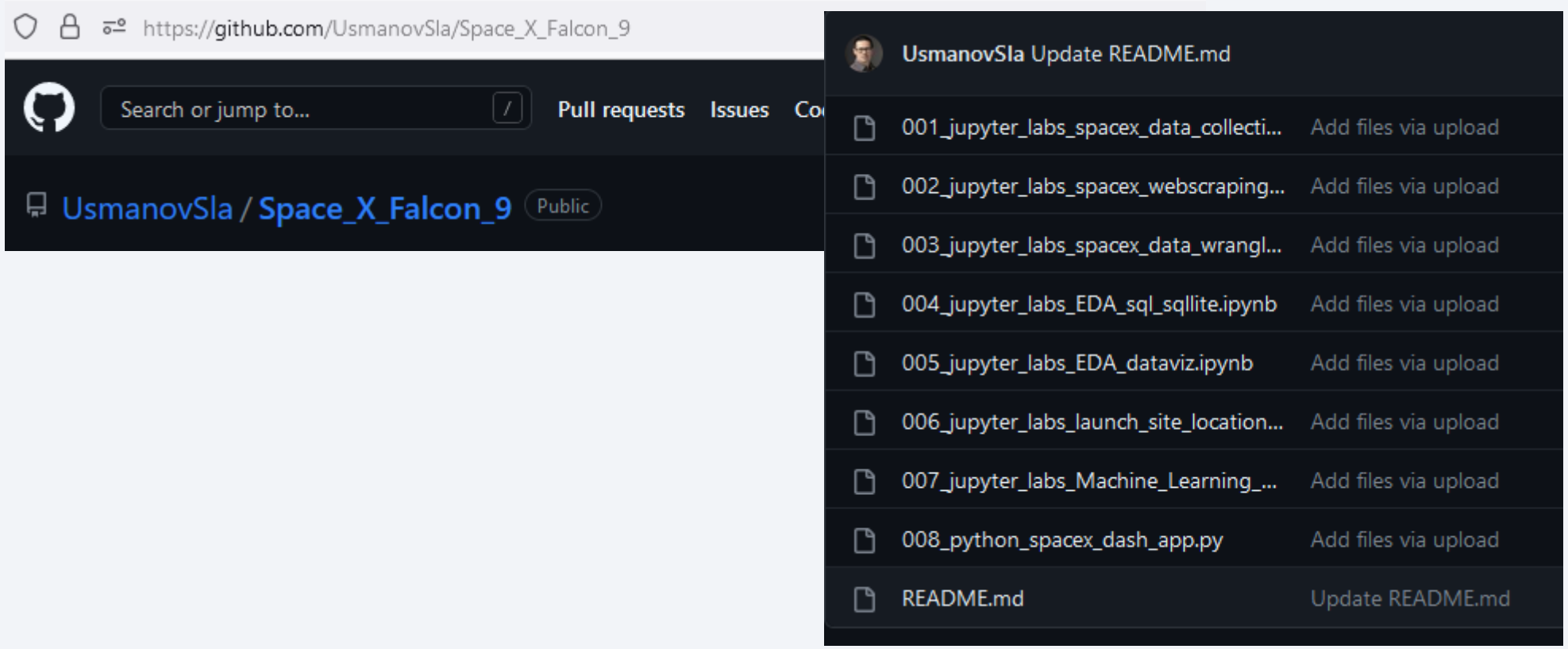


Conclusions

- Different launch sites have different success rates. CCAFS LC-40, has a success rate of 60 %, while KSC LC-39A and VAFB SLC 4E has a success rate of 77%.
- We can deduce that, as the flight number increases in each of the 3 launch sites, so does the success rate. For instance, the success rate for the VAFB SLC 4E launch site is 100% after the Flight number 50. Both KSC LC 39A and CCAFS SLC 40 have a 100% success rates after 80th flight
- If you observe Payload Vs. Launch Site scatter point chart you will find for the VAFB-SLC launchsite there are no rockets launched for heavy payload mass(greater than 10000).
- Orbits ES-L1, GEO, HEO & SSO have the highest success rates at 100%, with SO orbit having the lowest success rate at ~50%. Orbit SO has 0% success rate.
- 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
- With heavy payloads the successful landing or positive landing rate are more for Polar, LEO and ISS. However for GTO we cannot distinguish this well as both positive landing rate and negative landing(unsuccessful mission) are both there here
- •Anf finally the sucess rate since 2013 kept increasing till 2020.

Appendix

- Include any relevant assets like Python code snippets, SQL queries, charts, Notebook outputs, or data sets that you may have created during this project



Thank you!