Winning Space Race with Data Science

Muhamad Ihsan Fauzi

21-12-2022

Outline

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

Executive Summary

Summary of Methodologies

In this project, we use two methods for data collection: using SpaceX REST API and web scraping. Then we proceed with data wrangling in order to prepare required information for modeling such as number of launch on each site, number of occurrence and mission outcome on each orbit, and landing outcome label. After that, we do exploratory data analysis using visualization and SQL. We also utilized Folium and Plotly Dash in order to create interactive visualization. Finally, we create the predictive analysis by using several classification models such as Logistic Regression, Support Vector Machine, Decision Tree, and K Nearest Neighbor.

Summary of All Results

There are many interesting insights we got from the exploratory data analysis such as correlation between flight number and launch site, success rate and orbit type, and launch success rate yearly trend. We also can see from the visualization the proximity of launch sites to surrounding environment including railway, highway, and coastline. Then on the interactive dashboard, we can see the success rate of each sites and the relation between payload and booster version with launch outcome. On the predictive analysis part, we can see which model have the highest accuracy along with it's confusion matrix.

Introduction

Project Background and Context

The commercial space age is here, companies are making space travel affordable for everyone. Perhaps the most successful is SpaceX. One reason SpaceX can do this is the rocket launches are relatively inexpensive. 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. If an alternate company wants to bid against SpaceX for a rocket launch, then it needs to determine the price of each launch. One way to do this is by gathering information about Space X and creating dashboards. We will also determine if SpaceX will reuse the first stage.

Problems to Answers

Sometimes the first stage does not land. Sometimes it will crash. Other times, Space X will sacrifice the first stage due to the mission parameters like payload, orbit, and customer. Instead of using rocket science to determine if the first stage will land successfully, we will train a machine learning model and use public information to predict if SpaceX will reuse the first stage. Therefore, if we can determine if the first stage will land, we can determine the cost of a launch.

Section 1

Methodology

Methodology

Executive Summary

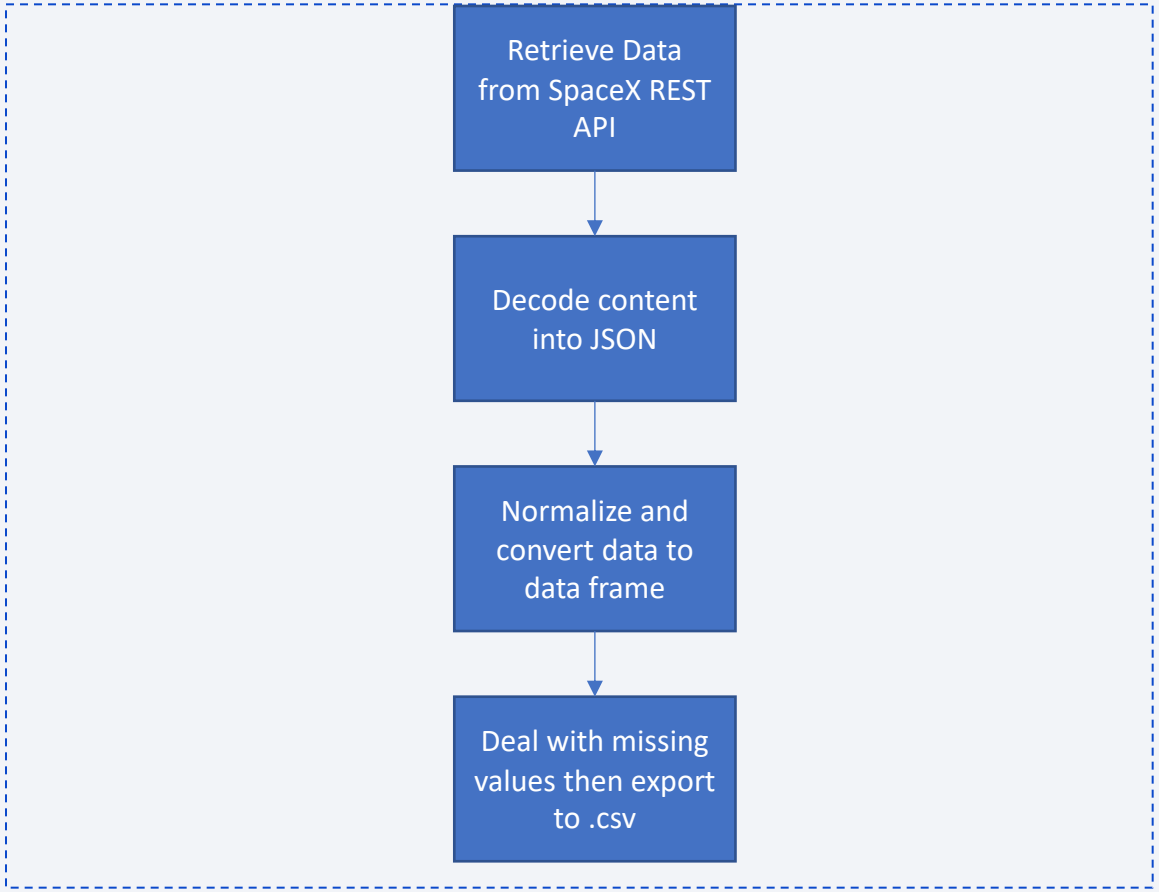
- Data collection methodology:
  - The data was collected using two methods: 1. With an API from SpaceX REST API 2. Using web scraping from Falcon 9 related Wiki pages
- Perform data wrangling
  - There are some important data attributes including launch sites, outcome, payload, booster version, etc. Then we calculate number of launches and occurrence as well as creating landing outcome label
- Perform exploratory data analysis (EDA) using visualization and SQL
- Perform interactive visual analytics using Folium and Plotly Dash
- Perform predictive analysis using classification models
  - The models built by learning from the training data which was prepared in the previous steps. The data transformed before used for learning. Then we use GridSearch to find the best parameters for each models. After that, we evaluate the score accuracy for each models using the test data

Data Collection

- The data was collected using two methods: 1. With an API from SpaceX REST API 2. Using web scraping from Falcon 9 related Wiki pages
- For API method, we target specific API endpoints to get data about launches. We perform a GET request using requests library and retrieve the result in JSON format. Then we convert the result into Pandas data frame.
- For web scraping method, we use BeautifulSoup package to web scrape some HTML tables from Falcon 9 related Wiki pages. Then we parse the data and convert them into Pandas data frame.

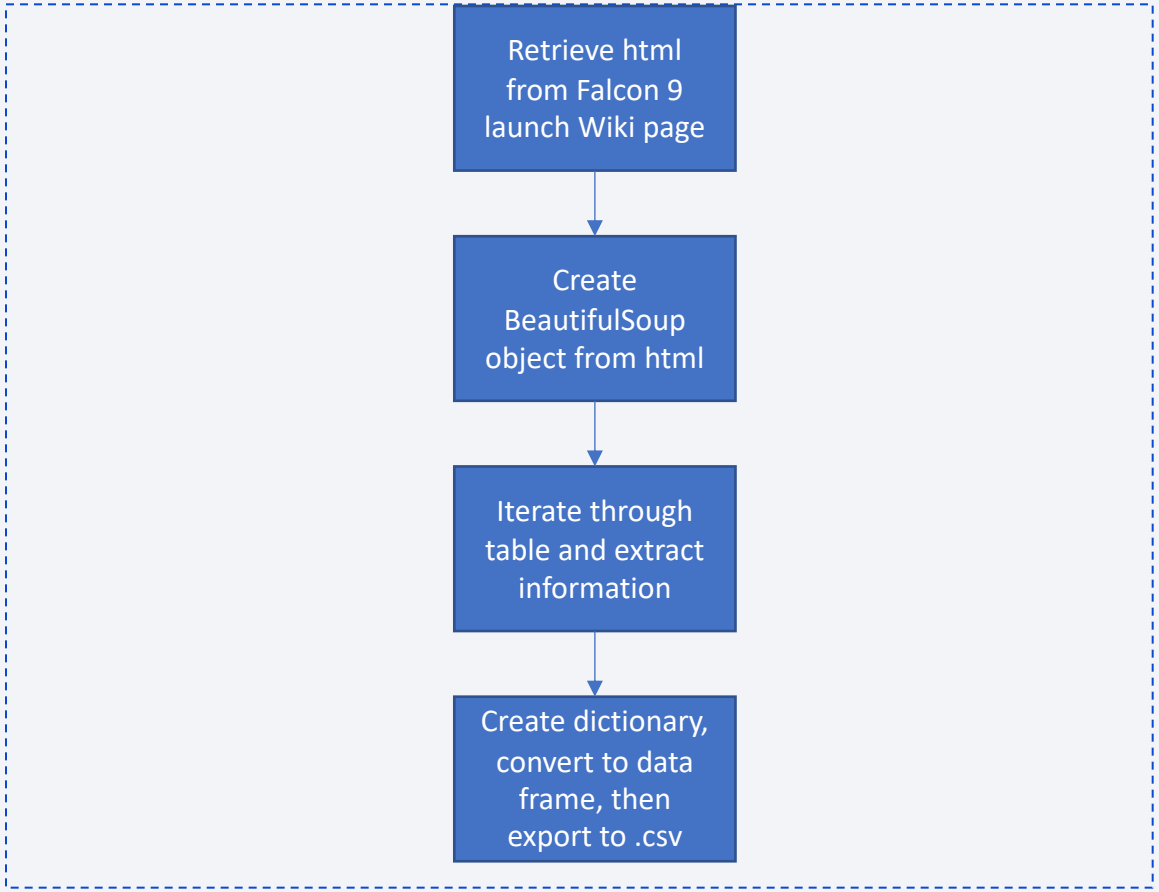
Data Collection – SpaceX API

- The SpaceX REST API endpoints starts with api.spacexdata.com/v4/ and we retrieve required data from different endpoints e.g., /capsules, /cores, etc.
- We retrieve the required data using the requests library. We decode the response content into JSON using .json() function and turn it into Pandas data frame using .json_normalize().
- Since lots of data from past launches is using ID, we need to retrieve information from each related endpoints then convert the result into data frame.
- We filter the data frame to only include Falcon 9 launches then we deal with missing values using .mean() and .replace() method.
- The complete work for data collection using API can be seen in the github repository below: https://github.com/mifauzi85/applied_ds_capstone/blob/main/jupyter-labs-spacex-data-collection-api.ipynb



Data Collection - Scraping

- We do web scraping to collect Falcon 9 historical launch records from a Wikipedia page titled List of Falcon 9 and Falcon Heavy launches.
- We retrieve the static page with HTTP GET method using requests library. Then we create BeautifulSoup object from the HTML response.
- We extract all columns/variables names from the HTML table header. First, we use .find_all() function to get required table, then we iterate through the <th> elements and apply extract_column_from_header() function to extract column name one by one.
- We create an empty dictionary and then iterate through the tables to fill required information. The resulted dictionary then converted into data frame.
- The complete work for data collection using web scraping can be seen in the github repository below: https://github.com/mifauzi85/applied_ds_capstone/blob/main/jupyter-labs-webscraping.ipynb



Data Wrangling

- In the data set there are different cases where the booster did not land successfully. In this step we mainly convert those outcomes into training labels 1 = successfully landed, and 0 = unsuccessful.
- We start by calculating percentage of missing values and identify numerical or categorical features. We then calculate number of launches on each site, number of occurrences of each orbit, and number of landing outcomes on each outcome type.
- We continue by identifying bad outcomes and create new feature landing_class to separate successful outcome (class = 1) from the unsuccessful (class = 0). We then append the new feature landing_class into the data set.
- The complete work for data wrangling can be seen in the github repository below: https://github.com/mifauzi85/applied_ds_capstone/blob/main/labs-jupyter-spacex-Data%20wrangling.ipynb

EDA with Data Visualization

- The first step in the EDA is to see the affect of flight number and payload to launch outcome. We use catplot to visualize this and from the chart created we can see that as the flight number increases, the first stage is more likely to land successfully. The more massive the payload, the less likely the first stage will return.
- Then we visualize the relationship between flight number and launch site using catplot as well. It seems like there is no correlation between flight number and launch site because even the flight number increase there is still failed launches.
- We then visualize the relationship between Payload and Launch Site using catplot. We can observe that for the VAFB-SLC launchsite there are no rockets launched for heavypayload mass(greater than 10000)
- Next, we check if there is relationship between success rate and orbit type. We create bar chart to visualize the relationship. We can see that ES-L1, GEO, HEO, and SSO has the highest mean success rate among orbit types.
- Next, we check relationship between flight number and orbit type. We use catplot again to visualize this and we can 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.
- We then check relationship between payload and orbit type by using catplot. 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.
- We then visualize the launch success rate yearly trend using line chart and we can observe that the sucess rate since 2013 kept increasing till 2020
- The last step here is to to one-hot-encoding to categorical variables and cast numerical variables into float
- The complete work for EDA with data visualization can be seen in the github repository below: https://github.com/mifauzi85/applied_ds_capstone/blob/main/jupyter-labs-eda-dataviz.ipynb

EDA with SQL

- For the EDA with SQL, we are using SQLite, so the first step is loading the SQL extension and establish connection with the database.
- Next, we are displaying unique launch sites in the space mission by using SELECT DISTINCT() query. Then we check 5 launch sites begin with string 'CCA' by using LIKE 'CCA%' in the WHERE clause and limit the result using LIMIT 5
- We then calculate the total payload carried by boosters launched by NASA by using SELECT SUM() query and adding LIKE '%NASA (CRS)%' in the WHERE clause. We also calculate average payload by booster version F9 v1.1 by using SELECT AVG() and LIKE '%F9 v1.1%' in WHERE clause.
- We then find out the date when the first successful landing outcome in ground pad achieved. Here we must use subquery to perform SELECT MIN() function and SUBSTR() function on Date column as well as filtering the data using WHERE clause.
- Next, we list the names of the boosters which have success in drone ship and have payload mass greater than 4000 but less than 6000. We do this by adding BETWEEN .. AND .. in the WHERE clause.
- We continue with checking total number of successful and failure mission outcomes. We do this by using SELECT COUNT() function and GROUP BY the mission outcome. After that we list the name of boosters which have carried the max payload, we do this using subquery to use SELECT MAX() function on payload column.
- After that, we list a records which display the month names, failure landing_outcomes in drone ship ,booster versions, launch_site for the months in year 2015. Since SQLite does not support month names, we used SUBSTR(Date, 4, 2) to get the months and SUBSTR(Date,7,4)='2015' for the year.
- Lastly, we use SUBSTR() combined with '| |' and 'BETWEEN .. AND ..' to find the records of successful landing outcome between the date 04-06-2010 and 20-03-2017. We then use COUNT() function to count the result and then 'ORDER BY .. DESC' to rank the result in descending order.

- The complete work for EDA with data visualization can be seen in the github repository below: https://github.com/mifauzi85/applied_ds_capstone/blob/main/jupyter-labs-eda-sql-coursera_sqllite.ipynb

Build an Interactive Map with Folium

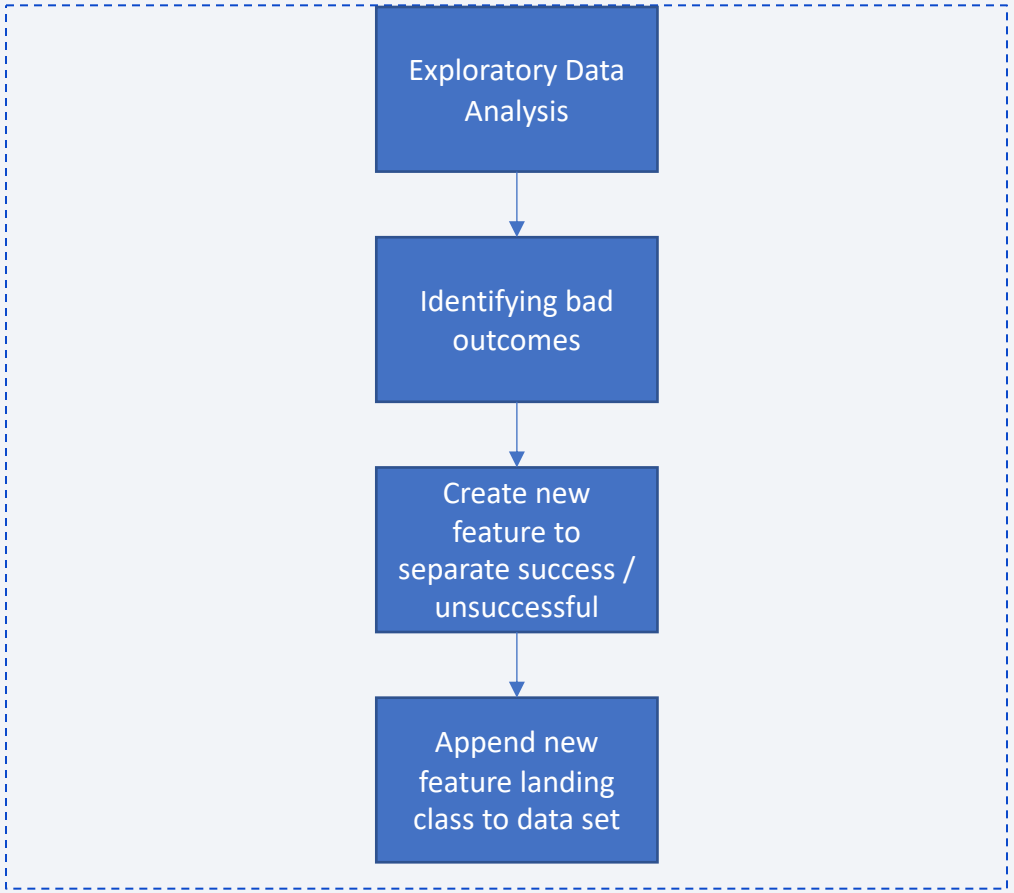
- The first object we created is folium.Circle to add highlighted circle area on a specific coordinate. Then we add folium.Popup object to show a pop-up label showing the name of the area. We then add folium.map.Marker object to show an icon as text label on the coordinate
- Next, we want to show launch outcomes for each site, and see which site has the highest success rates. But since many launch records will have same launch sites, we use MarkerCluster object to contain many markers at each launch sites coordinate. We do this by iterating through the data set and adding folium.Marker object with folium.Icon to indicate launch outcome into the MarkerCluster object.
- We then create folium.PolyLine objects to show the distance between launch site coastline, highway, railway, and a city. That way we can illustrate the proximities between each launch site to the surroundings.
- The complete work of interactive map with folium can be seen in the github repository below: https://github.com/mifauzi85/applied_ds_capstone/blob/main/lab_jupyter_launch_site_location.ipynb

Build a Dashboard with Plotly Dash

- The first step we did was adding a Drop-down input component to list all falcon 9 launch sites.
- After that, we created a callback function to generate pie chart based on selection on the drop-down input. The data shown in the pie chart is the success rate of the launches. If the selection is all, then it will show total success launches of all sites. If the selection is specific site, then the data will show the success and failed count of that site. This way we can observe which site has the largest successful launches and which site has the highest launch success rate.
- Next, we want to check correlation between variable payload to mission outcome. We want to be able to easily select different payload range and see if we can identify some visual patterns. To achieve this, we use Range Slider input component to select the payload.
- After that, we created a callback function to generate scatter plot between payload and launch outcome so we can see the correlation between these two variables. We also color-label the Booster version on each scatter point so that we may observe mission outcomes with different boosters. This way we can observe which payload range(s) has the highest launch success rate, which payload range(s) has the lowest launch success rate, and which F9 Booster version (v1.0, v1.1, FT, B4, B5, etc.) has the highest launch success rate.
- The python code for creating dashboard with Plotly Dash can be seen in the github repository below: https://github.com/mifauzi85/applied_ds_capstone/blob/main/spacex_dash_app.py
- The dashboard created can be downloaded from the github repository below: https://github.com/mifauzi85/applied_ds_capstone/blob/main/spacex_launch_dashboard.webarchive

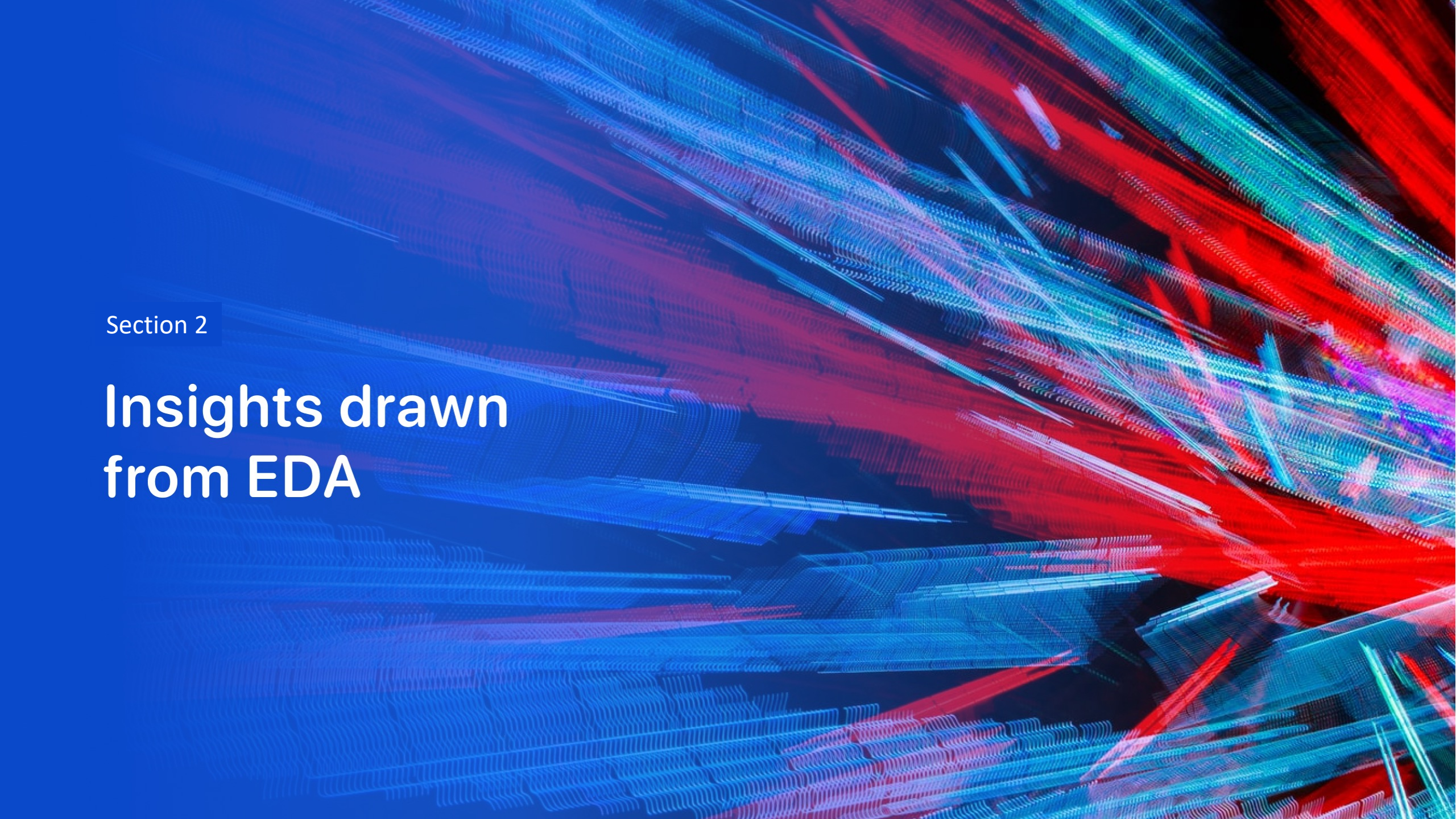
Predictive Analysis (Classification)

- First step is transforming the data using StandardScaler and then we split the data into training and test data using train_test_split.
- We then find best hyperparameters for Logistic Regression, SVM, Decision Tree, and KNN using GridSearchCV. Then we train the model using training data and check the best parameters and accuracy.
- After that, we use the best_estimator_ of each models to calculate the accuracy on test data using the method score(). We also create prediction using test data and plot the result into confusion matrix.
- Finally, we compare the performance of each model by calculating the jaccard score, F1 score, and log loss.
- The complete work for predictive analytics can be seen in the github repository below: https://github.com/mifauzi85/applied_ds_capstone/blob/main/SpaceX_Machine%20Learning%20Prediction_Part_5.ipynb

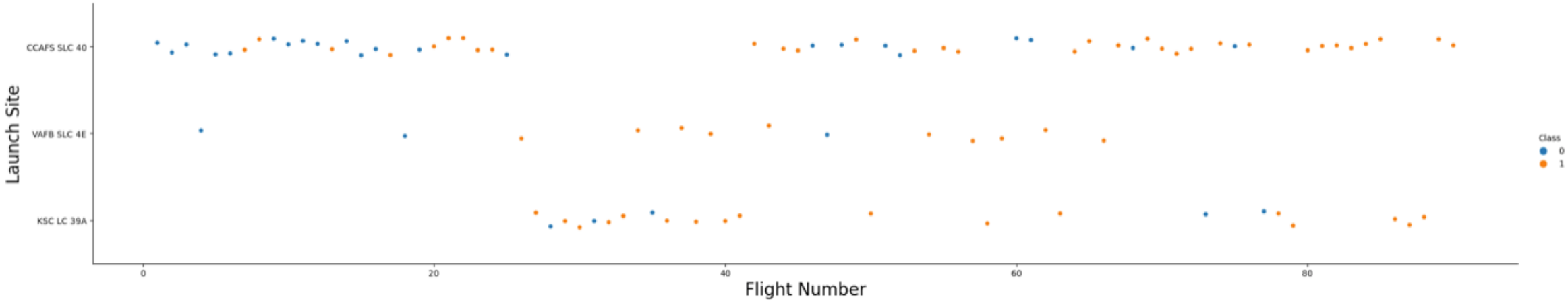


Results

- Exploratory data analysis results:
  - The insights drawn from EDA can be seen in section 2. This is including results from EDA with visualization and SQL
- Interactive analytics demo in screenshots:
  - You can see the screenshots from interactive analytics including interactive map created with Folium and Dashboard created with Plotly Dash in section 3 and 4.
- Predictive analysis results:
  - The predictive analysis results from each model built can be seen in section 5 including the model accuracy and its confusion matrix.

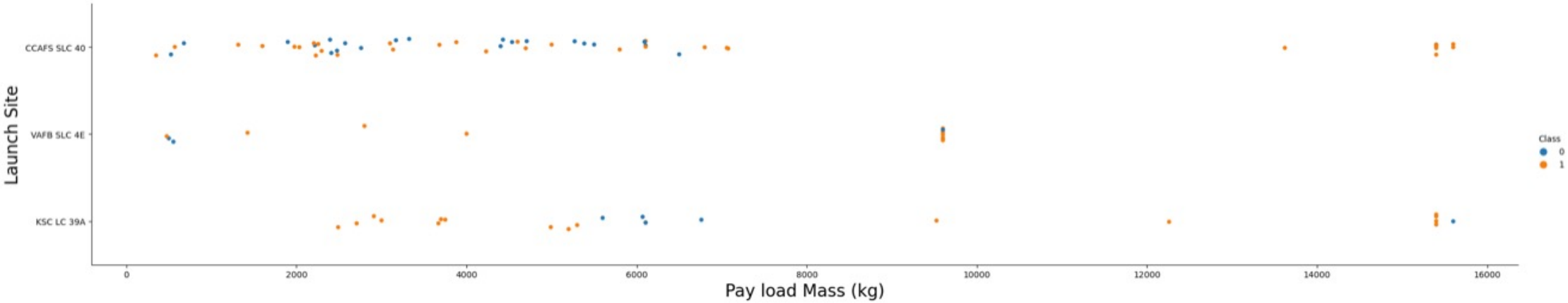

Section 2
Insights drawn from EDA

Flight Number vs. Launch Site



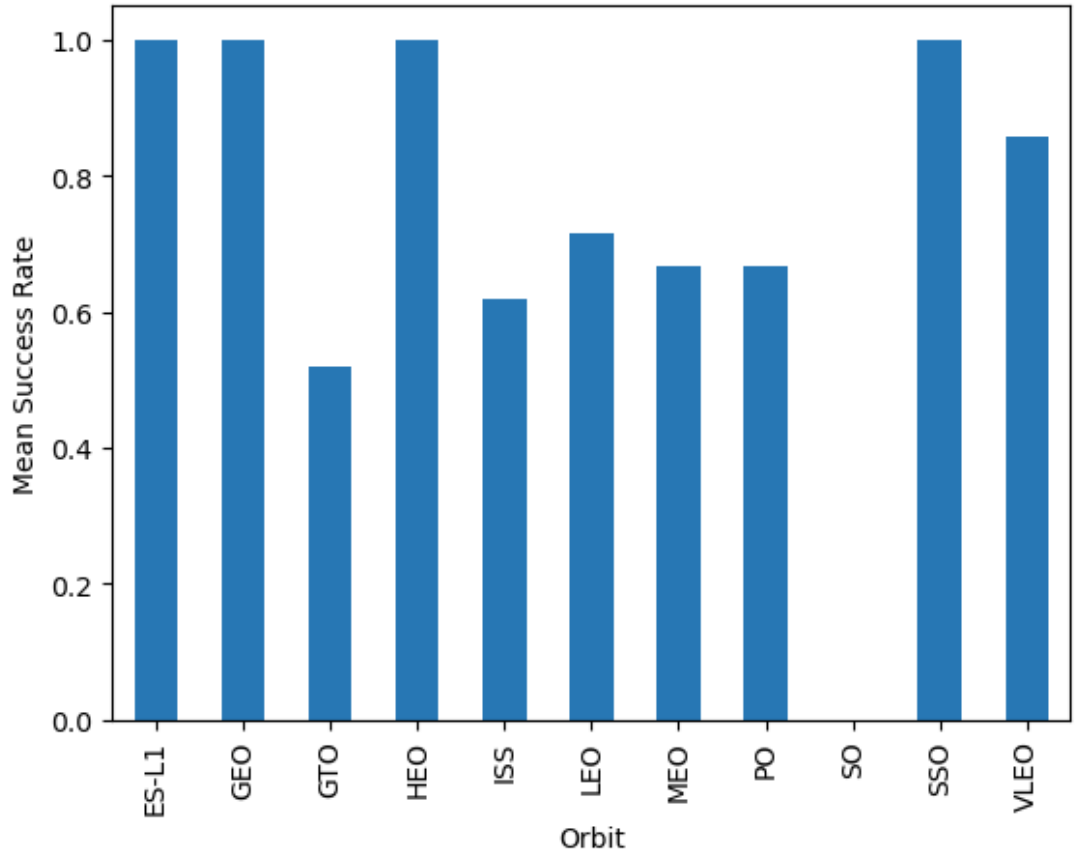
As we can see from the scatter plot, the number of successful launches increased with increasing flight number. Most of the initial flights happened at CCAFS SLC 40 and mostly failed. But after several flights, the number of successful launches increased. This means the team at SpaceX fine tuned the launches until reached higher success rate before they try it at other launch sites. As we can see that between flight number 25th and 40th they did launches on VAFB SLC 4E and KSC LC 39A instead of CCAFS SLC 40. After both of them reach high success rate then they continue launches on all sites.

Payload vs. Launch Site



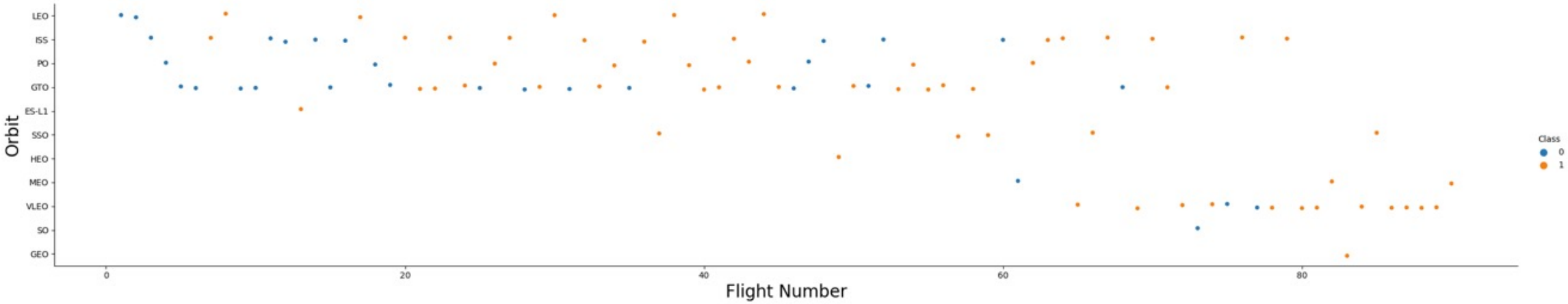
Now if you observe Payload Vs. Launch Site scatter point chart you will find for the VAFB-SLC launchsite there are no rockets launched for heavypayload mass(greater than 10000).

Success Rate vs. Orbit Type



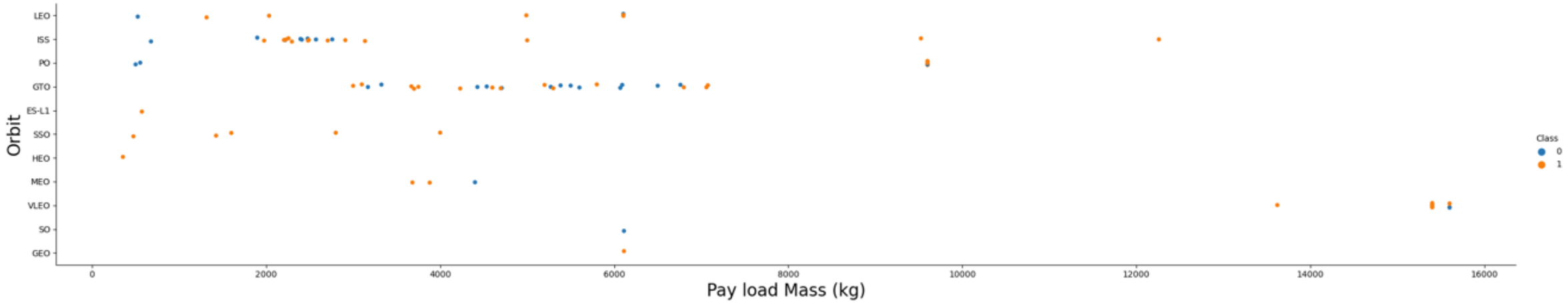
As we can see from the chart above, the ES-L1, GEO, HEO, and SSO have the highest success rate compared to other orbit type with mean success rate 1.0

Flight Number vs. Orbit Type



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.

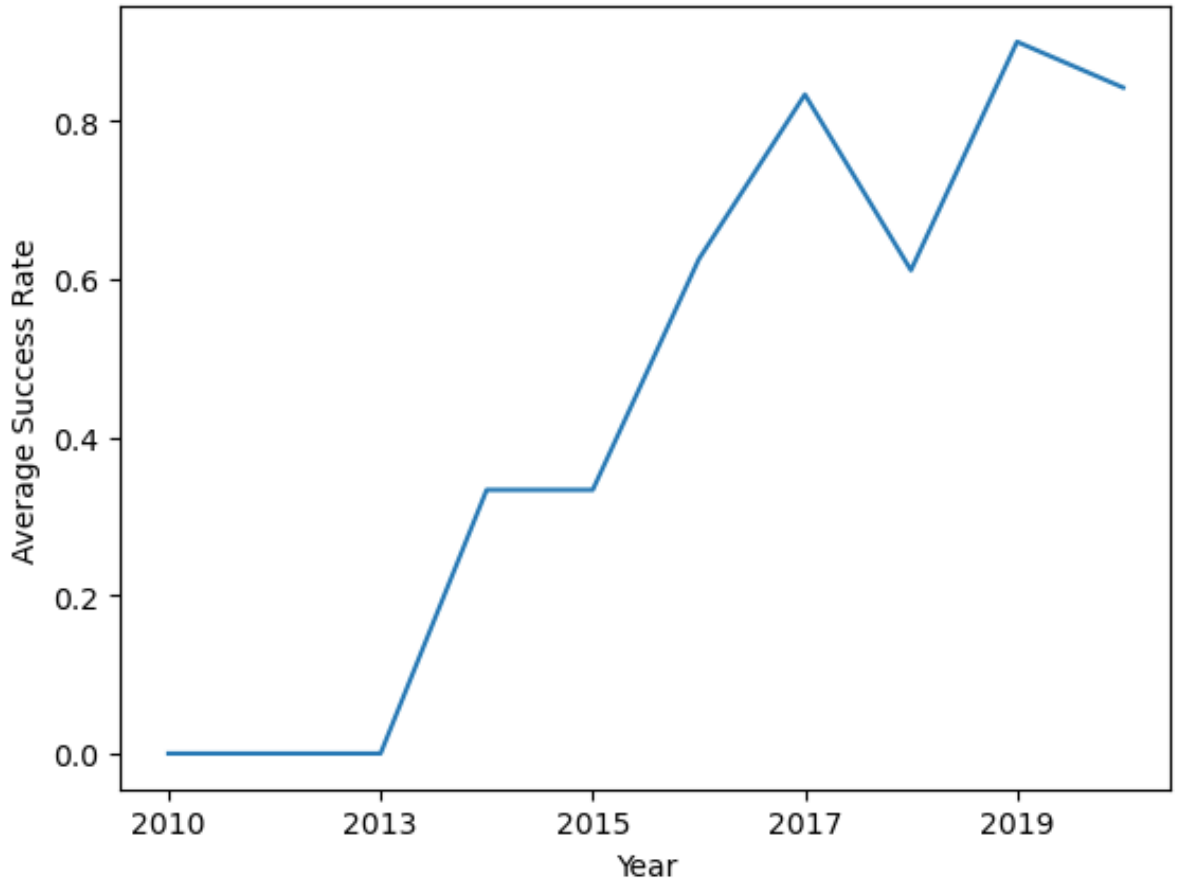
Payload vs. Orbit Type



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



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

All Launch Site Names

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

```
%sql SELECT DISTINCT("Launch_Site") FROM SPACEXTBL
```

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

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

We use SELECT DISTINCT from column Launch_Site to show the distinct launch sites.

Launch Site Names Begin with 'CCA'

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

```
%sql SELECT * FROM SPACEXTBL WHERE "Launch_Site" LIKE 'CCA%' LIMIT 5
```

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

Date	Time (UTC)	Booster_Version	Launch_Site	Payload	PAYLOAD_MASS__KG_	Orbit	Customer	Mission_Outcome	Landing _Outcome
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

On the query, we use LIKE ‘CCA%’ in the where clause to get launch sites beginning with string ‘CCA’.

We also use LIMIT 5 to limit the result to just 5 records.

Total Payload Mass

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

```
%sql SELECT SUM(PAYLOAD_MASS__KG_) FROM SPACEXTBL WHERE Customer like '%NASA (CRS)%'
```

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

SUM(PAYLOAD_MASS__KG_)
48213

First, we use LIKE '%NASA (CRS)%' in the where clause to get launches by NASA.

Then we calculate the total payload by using SUM(PAYLOAD_MASS__KG_).

Average Payload Mass by F9 v1.1

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, we use LIKE '%F9 v1.1%' in the where clause to get launches using booster version F9 v1.1.

Then we calculate the average payload by using AVG(PAYLOAD_MASS__KG_).

First Successful Ground Landing Date

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

Hint:Use min function

```
%%sql SELECT DATE FROM SPACEXTBL
WHERE SUBSTR(Date,7,4) = (SELECT MIN(SUBSTR(Date,7,4)) FROM SPACEXTBL WHERE [Landing _Outcome] = 'Success (ground pad)')
AND [Landing _Outcome] = 'Success (ground pad)'
```

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

Date
22-12-2015

Since SQLite doesn't support date related functions, we need to use SUBSTR to get the year from the Date column. We use subquery and use MIN() function to get the lowest year where the landing outcome is 'Success (ground pad)'. And then we select the date where the SUBSTR(Date, 7, 4) equals to the result of the subquery.

Successful Drone Ship Landing with Payload between 4000 and 6000

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

```
%sql SELECT "Booster_Version" FROM SPACEXTBL WHERE [Landing _Outcome] = 'Success (drone ship)' AND PAYLOAD_MASS__KG_ BETWEEN 4000 AND 6000
```

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

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

We search the records where landing outcome equals to ‘Success (drone ship)’ then use range BETWEEN 4000 AND 6000 for the payload mass.

Total Number of Successful and Failure Mission Outcomes

List the total number of successful and failure mission outcomes

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

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

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

We GROUP BY the records based on Mission_Outcome column and then we calculate the COUNT() of each mission outcome.

Boosters Carried Maximum Payload

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

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

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

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

We used subquery to get the max payload using MAX(PAYLOAD_MASS__KG_). Then we display the Booster_Version where they PAYLOAD_MASS__KG_ equals to the subquery result.

2015 Launch Records

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(Date, 4, 2) AS Month, [Landing _Outcome], "Booster_Version", "Launch_Site"
FROM SPACEXTBL
WHERE SUBSTR(Date, 7, 4) = '2015' AND
[Landing _Outcome] = 'Failure (drone ship)'
```

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

Month	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

Since SQLite doesn't support date related functions, we need to use SUBSTR to get the year and month from the Date column. We filter the result based on landing outcome equal to 'Failure (drone ship)'.

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

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 SUCCESS
FROM SPACEXTBL
WHERE [Landing _Outcome] LIKE '%Success%'
AND SUBSTR(DATE,7,4)||SUBSTR(DATE,4,2)||SUBSTR(DATE,1,2)
BETWEEN '20100604' and '20170320'
GROUP BY [Landing _Outcome]
ORDER BY SUCCESS DESC
```

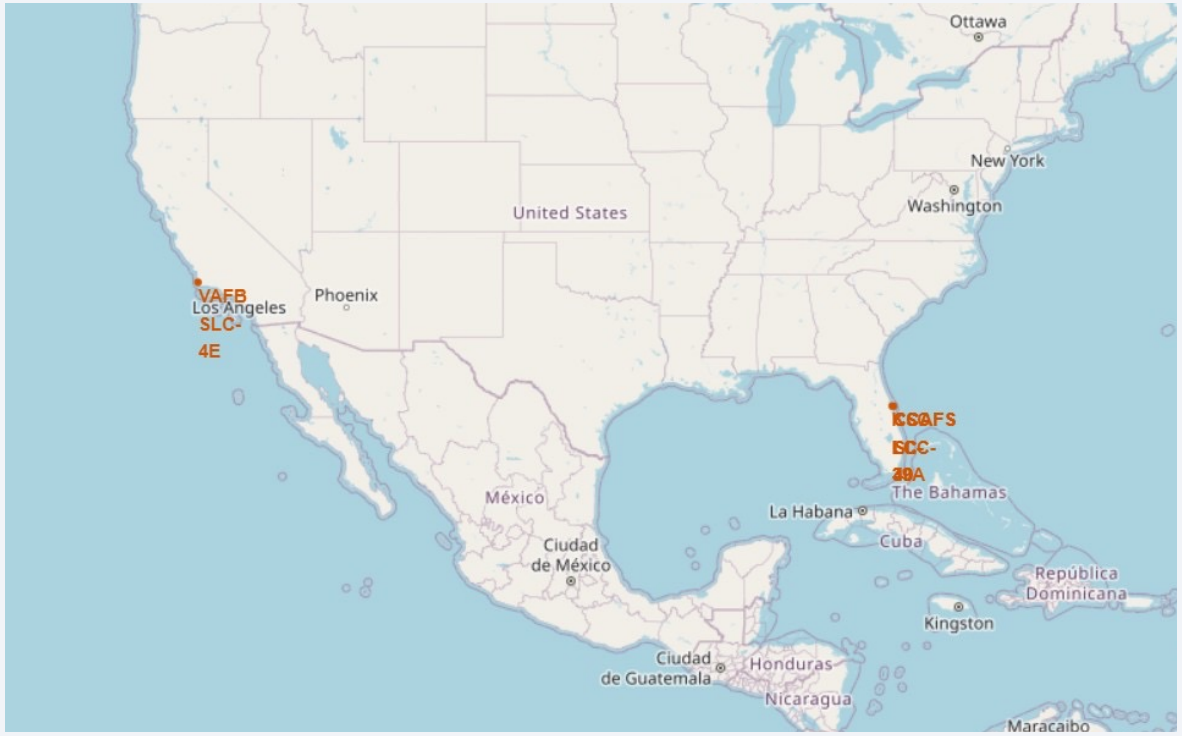
```
 * sqlite:///my_data1.db
Done.
```

Landing _Outcome	SUCCESS
Success (drone ship)	5
Success (ground pad)	3

Since SQLite doesn't support date related functions, we need to use SUBSTR and concatenate (| |) to construct the full date string. And then we use BETWEEN '20100604' AND '20170320' to filter records between those dates. We then GROUP BY the landing outcome so we can COUNT() the landing outcome. Lastly, we ORDER the result BY the count result in DESC order.

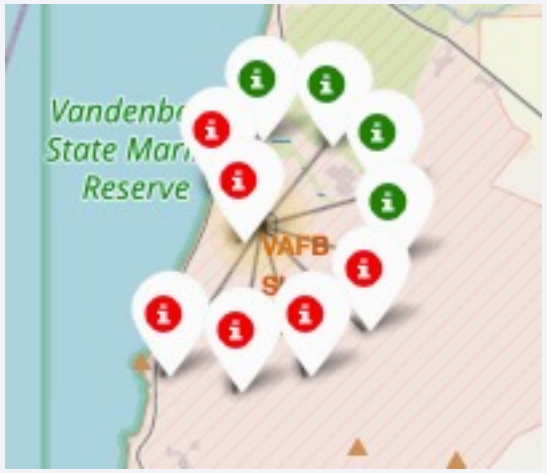

Section 3
Launch Sites
Proximities Analysis

Launch Sites Location



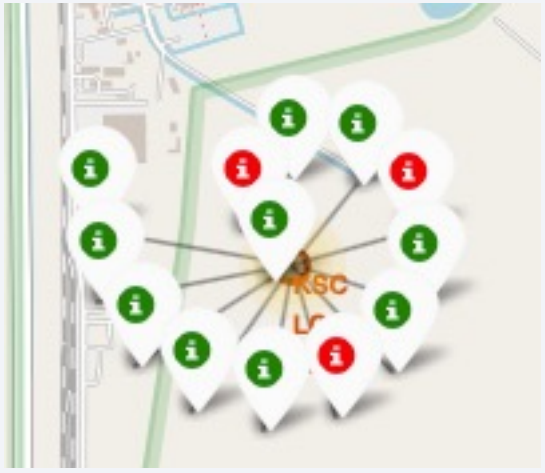
Based on the map, all launch sites are in proximity to the Equator line and very close proximity to the coast. I think it's to get the shortest distance to earth's orbit and the lesser dense area to minimize impact if there is an accident.

Launch Outcomes



VAFB SLC 4E

CCAFS LC-40



KSC LC-39A

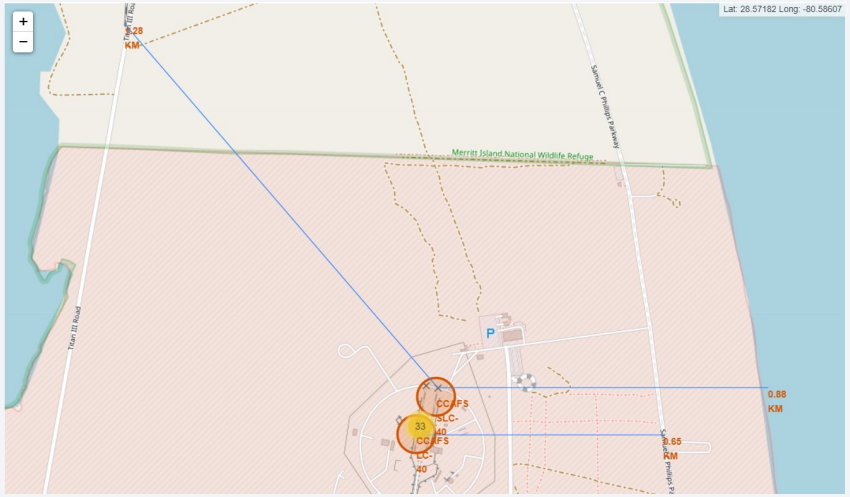
CCAFS SLC-40

The pictures beside shows the launch outcomes of different launch sites (green = success, red = fail).

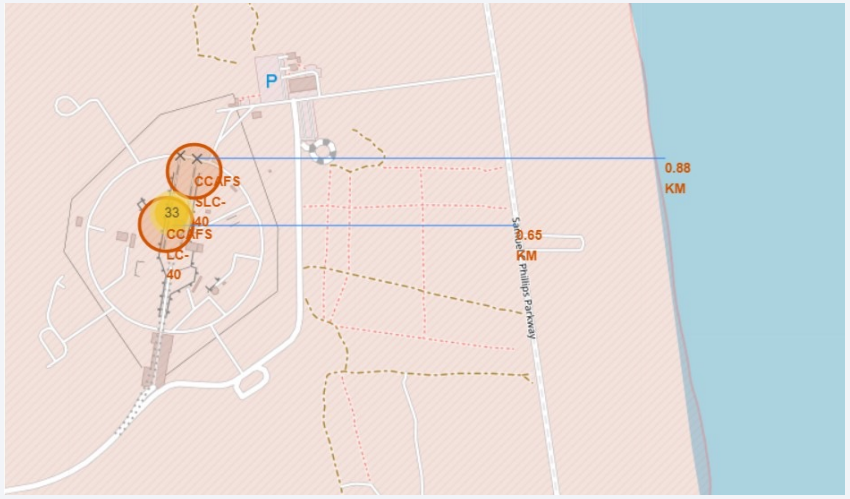
As we can see, the highest success rate was happened at KSC LC-39A. The lowest success rate is happened at CCAFS LC-40. But the number of launches is also mostly happened at CCAFS LC-40.

We also can see that CCAFS SLC-40 have higher success rate even though it's located at the same coordinate with CCAFS LC-40. But the number of launches and successful launches also smaller.

Launch Site Proximities



Distance to railway



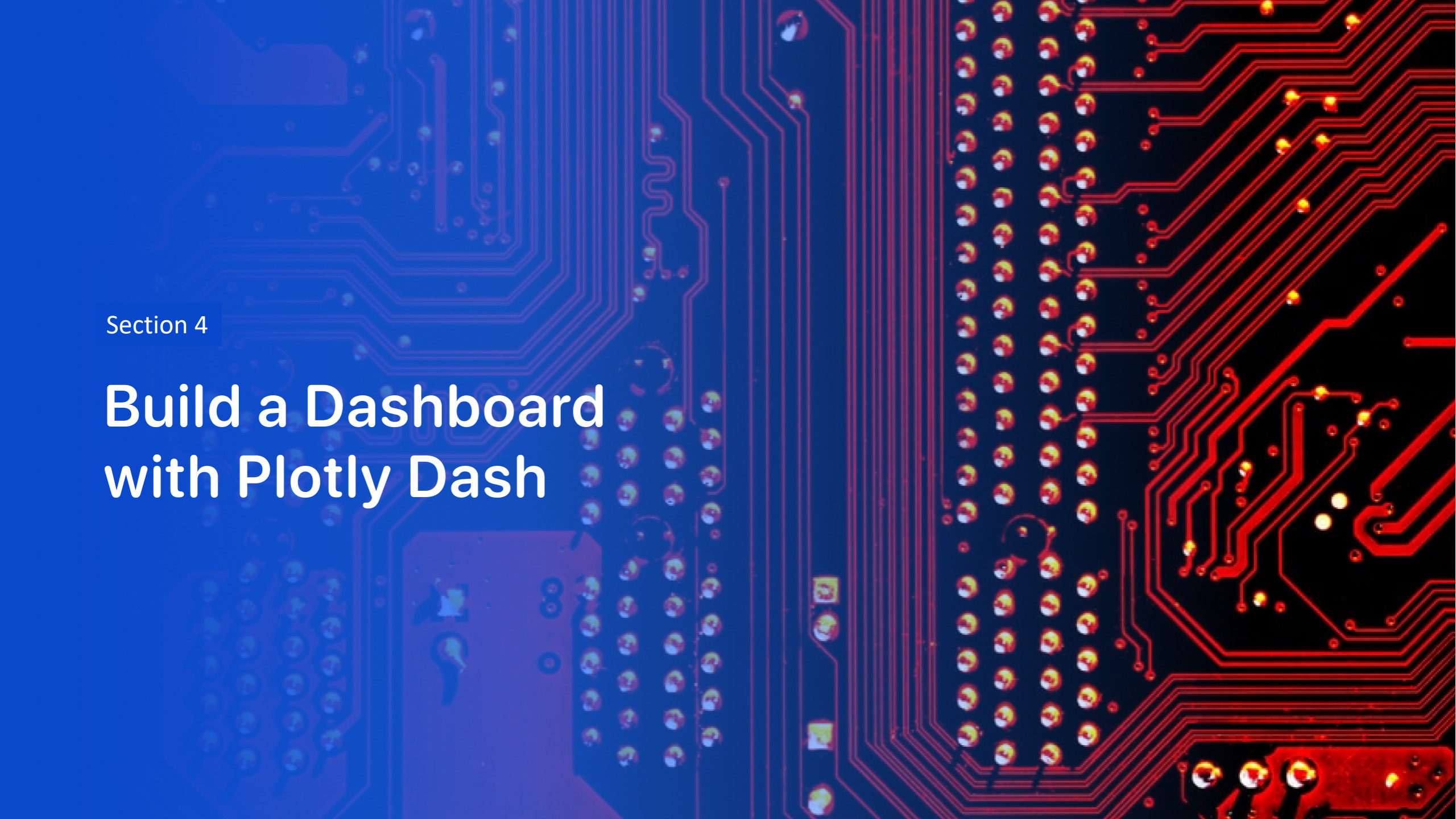
Distance to highway and coastline

Distance to cities

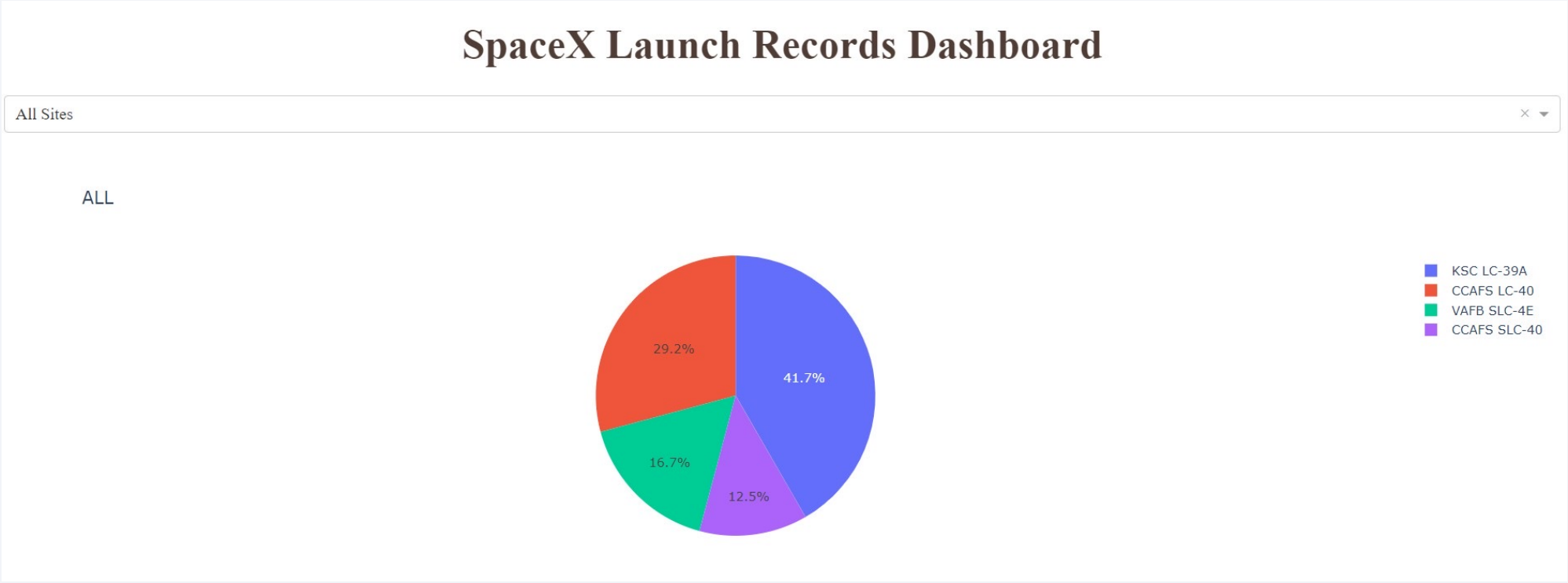
As you can see from the pictures besides, the launch sites are in very close proximity to railway, highway, and coastline.

They are also in quite significant distance to cities as you can see from the distance between CCAFS LC-40 and Cape Canaveral.

The launch sites located near railways and highways to make it easy for transporting logistics for the launches. They are in close proximity to coastline and keep certain distance away from cities for safety and minimize risk if accidents happen.

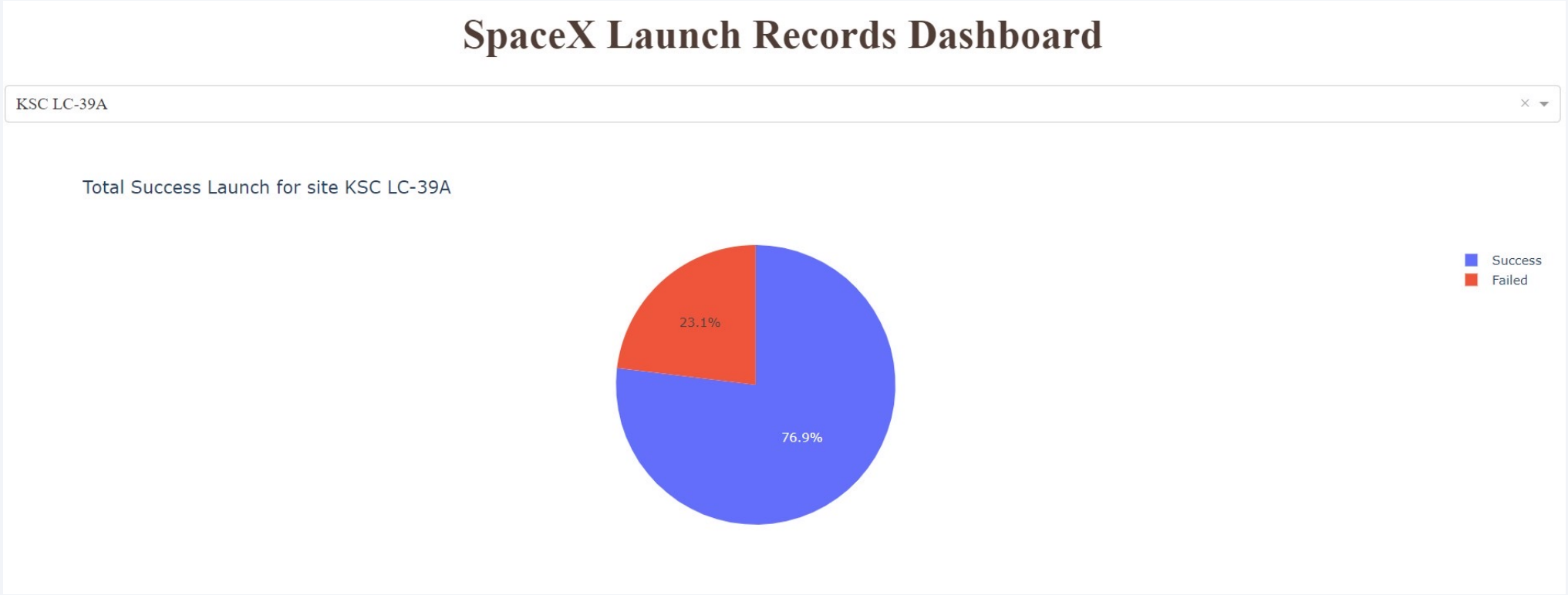

Section 4
Build a Dashboard with Plotly Dash

Success Launch Count All Sites



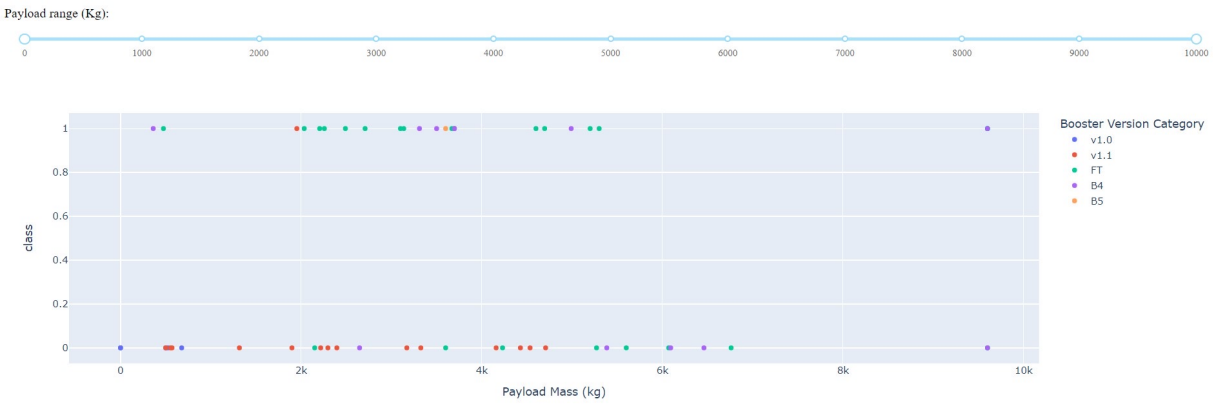
The total successful launches from all sites is 24. As we can see from the chart above, KSC LC-39A has the highest count with 10 launches or 41.7%. Followed by CCAFS LC-40 with 7 launches or 29.2%. The third place is VAFB SLC-4E with 6 launches or 16.7% and the last one is CCAFS SLC-40 with 3 launches or 12.5%. One thing to note here, this count doesn't represent the success ratio. For example, although CCAFS LC-40 is the second in terms of successful launches count, its success rate is only 26.9% because it has 19 failed launches. In contrast, CCAFS SLC-40 has success rate 42.9% because it only has 4 failed launches.

Highest Success Rate Site

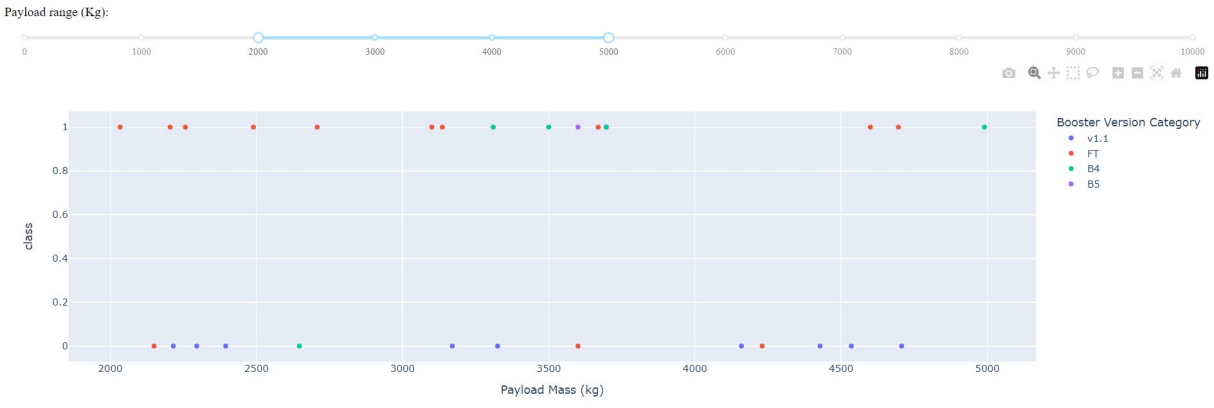


KSC LC-39A consistently shown that it has the highest launch success. Not only it has the highest count of success launches, but it is also having the highest success rate with 76.9%. It is followed by CCAFS SLC-40 with 42.9% even though it has the lowest count of success launches. The third place is VAFB SLC-4E with 40% and the last one is CCAFS LC-40 with only 26.9%. With this dashboard we can understand that the number of successful launches doesn't guarantee the success rate.

Success Rate Based On Payload and Booster Version



As you can see from the dashboard, most successful launches happened for payload in range 2000 – 5000 Kg.

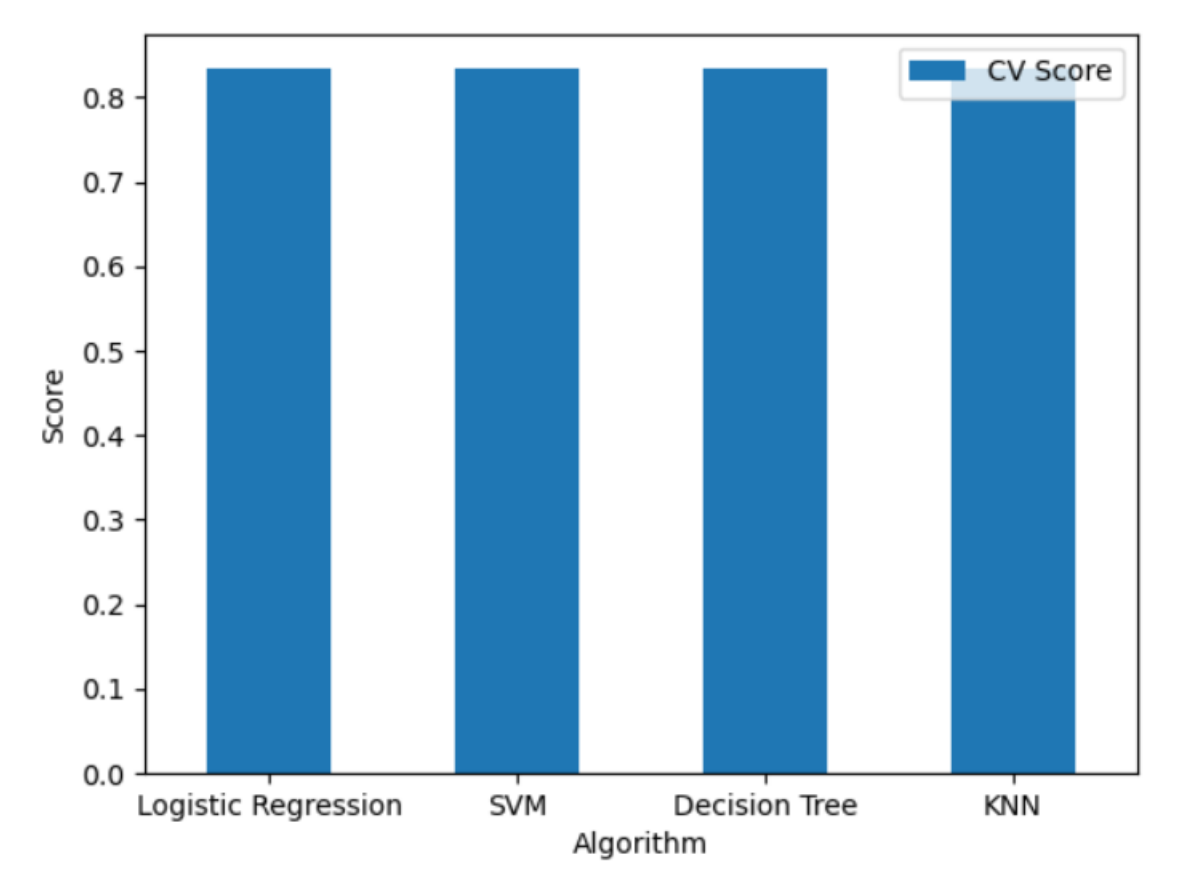


If we dive deeper for payload in that range, we can see that FT is the booster version with highest success rate. While v1.1 didn't have any successful launches within this payload range.

Section 5

Predictive Analysis (Classification)

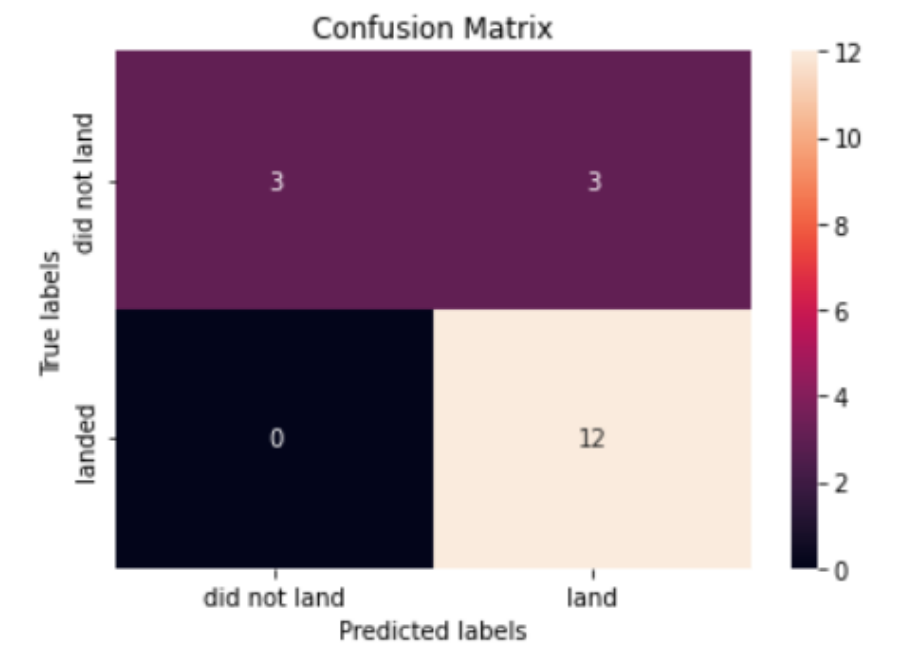
Classification Accuracy



Algorithm	CV Score	Jaccard Score	F1-score
Logistic Regression	0.833333	0.8	0.888889
SVM	0.833333	0.8	0.888889
Decision Tree	0.833333	0.8	0.888889
KNN	0.833333	0.8	0.888889

As we can see, all algorithms / models give good performance with same accuracy score among GridSearchCV score, Jaccard Score, and F-1 score.

Confusion Matrix



The picture besides is the confusion matrix for Logistic Regression model. As you can see from the confusion matrix, we see that logistic regression can distinguish between the different classes. The major problem is false positives.

Conclusions

- It is possible to create launch success predictions based on Falcon 9 launch data
- All algorithms / models give good prediction performance and score accuracy
- For Logistic Regressions the best parameters are 'C': 0.01, 'penalty': 'l2', 'solver': 'lbfgs'
- For SVM the best parameters are 'C': 1.0, 'gamma': 0.03162277660168379, 'kernel': 'sigmoid'
- For Decision Tree the best parameters are 'criterion': 'gini', 'max_depth': 12, 'max_features': 'sqrt', 'min_samples_leaf': 4, 'min_samples_split': 2, 'splitter': 'best'
- For KNN the best parameters are 'algorithm': 'auto', 'n_neighbors': 10, 'p': 1

Appendix – Data Collection with API

- Python code for collecting SpaceX launches data via API:

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

response = requests.get(static_json_url)

data = pd.json_normalize(response.json())
```

- After some data cleansing, we create new data frame consist of the data that we need from dictionary launch_dict:

```
df = pd.DataFrame(launch_dict)

df.head() #showing the head of the dataframe
```

	FlightNumber	Date	BoosterVersion	Payload Mass	Orbit	LaunchSite	Outcome	Flights	GridFins	Reused	Legs	LandingPad	Block	Reused Count	Serial	Longitude	Latitude
0	1	2006-03-24	Falcon 1	20.0	LEO	Kwajalein Atoll	None None	1	False	False	False	None	NaN	0	Merlin1A	167.743129	9.047721
1	2	2007-03-21	Falcon 1	NaN	LEO	Kwajalein Atoll	None None	1	False	False	False	None	NaN	0	Merlin2A	167.743129	9.047721
2	4	2008-09-28	Falcon 1	165.0	LEO	Kwajalein Atoll	None None	1	False	False	False	None	NaN	0	Merlin2C	167.743129	9.047721
3	5	2009-07-13	Falcon 1	200.0	LEO	Kwajalein Atoll	None None	1	False	False	False	None	NaN	0	Merlin3C	167.743129	9.047721
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

Appendix – Data Collection with API (1)

- Filtering data frame to only include Falcon 9 launches and resetting the FlightNumber column, followed by checking missing values:

```
data_falcon9 = df[df['BoosterVersion'] != 'Falcon 1’]

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

data_falcon9.isnull().sum()
```

```
FlightNumber      0
Date              0
BoosterVersion    0
PayloadMass       5
Orbit             0
LaunchSite        0
Outcome           0
Flights           0
GridFins          0
Reused            0
Legs              0
LandingPad       26
Block             0
ReusedCount       0
Serial            0
Longitude         0
Latitude          0
dtype: int64
```


Before we can continue, we must deal with these missing values. The LandingPad column will retain None values to represent when landing pads were not used.

Appendix – Data Collection with API (2)

- Dealing with missing values:

```
# Calculate the mean value of PayloadMass column

plmean = data_falcon9['PayloadMass'].mean()

# Replace the np.nan values with its mean value

data_falcon9['PayloadMass'] = data_falcon9['PayloadMass'].replace(np.nan, plmean)

data_falcon9.isnull().sum()
```

```
FlightNumber      0
Date              0
BoosterVersion    0
PayloadMass       0
Orbit             0
LaunchSite        0
Outcome           0
Flights           0
GridFins          0
Reused            0
Legs              0
LandingPad       26
Block             0
ReusedCount       0
Serial            0
Longitude         0
Latitude          0
dtype: int64
```

Appendix – Data Collection with web scraping

- Python code for collecting SpaceX launches data via web scraping:

```
static_url = "https://en.wikipedia.org/w/index.php?title=List_of_Falcon_9_and_Falcon_Heavy_launches&oldid=1027686922"

response = requests.get(static_url)

soup = BeautifulSoup(response.text, 'html.parser')

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

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

- Extracting all column/variable names from HTML table header:

```
html_tables = soup.find_all('table')

first_launch_table = html_tables[2] #the third table is our target table contains the actual launch records

column_names = []

for row in first_launch_table.find_all('th'):

    name = extract_column_from_header(row)

    if (name != None and len(name) > 0):

        column_names.append(name)

print(column_names)
```

```
['Flight No.', 'Date and time ( )', 'Launch site', 'Payload', 'Payload mass', 'Orbit', 'Customer', 'Launch outcome']
```

Appendix – Data Collection with web scraping (1)

- Create empty dictionary with keys from the extracted column names:

```
launch_dict= dict.fromkeys(column_names)

del launch_dict['Date and time ( )’] #remove unnecessary column

launch_dict['Flight No.'] = [] #initiate launch_dict with empty list

launch_dict['Launch site'] = []

launch_dict['Payload'] = []

launch_dict['Payload mass'] = []

launch_dict['Orbit'] = []

launch_dict['Customer'] = []

launch_dict['Launch outcome'] = []

launch_dict['Version Booster']=[] #adding new columns

launch_dict['Booster landing']=[]

launch_dict['Date']=[]

launch_dict['Time']=[]

Launch_dict
```

```
{'Flight No.': [],
 'Launch site': [],
 'Payload': [],
 'Payload mass': [],
 'Orbit': [],
 'Customer': [],
 'Launch outcome': [],
 'Version Booster': [],
 'Booster landing': [],
 'Date': [],
 'Time': []}
```

Appendix – Data Collection with web scraping (2)

- Fill-up the launch_dict with records extracted from table rows (include removing some noises):

```
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')
```

Continue next slide…

Appendix – Data Collection with web scraping (3)

```
#if it is number save cells in a dictonary
if flag:
    extracted_row += 1
    # Flight Number value
    launch_dict['Flight No.'].append(flight_number)
    datatimelist=date_time(row[0])
    # Date value
    date = datatimelist[0].strip(',')
    launch_dict['Date'].append(date)
    # Time value
    time = datatimelist[1]
    launch_dict['Time'].append(time)
```

Continue next slide…

Appendix – Data Collection with web scraping (4)

```
 # Booster version
bv=booster_version(row[1])
if not(bv):
    bv=row[1].a.string
launch_dict['Version Booster'].append(bv)
# Launch Site
launch_site = row[2].a.string
launch_dict['Launch site'].append(launch_site)
# Payload
payload = row[3].a.string
launch_dict['Payload'].append(payload)
# Payload Mass
payload_mass = get_mass(row[4])
launch_dict['Payload mass'].append(payload_mass)
```

Continue next slide…

Appendix – Data Collection with web scraping (5)

```
            # Orbit
        orbit = row[5].a.string
        launch_dict['Orbit'].append(orbit)
        # Customer
        if (row[6].a != None):
            customer = row[6].a.string
        else:
            customer = row[6].text
        launch_dict['Customer'].append(customer)
        # Launch outcome
        launch_outcome = list(row[7].strings)[0]
        launch_dict['Launch outcome'].append(launch_outcome)
        # Booster landing
        booster_landing = landing_status(row[8])
        launch_dict['Booster landing'].append(booster_landing)
df=pd.DataFrame(launch_dict) #create data frame from the filled up launch_dict
df.to_csv('spacex_web_scraped.csv', index=False) #save the data frame to csv
```

Appendix – Data Wrangling

- Loading dataset from previous section and calculate number of launches on each site:

```
df=pd.read_csv("https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/datasets/dataset_part_1.csv")

df['LaunchSite'].value_counts()
```

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

- Calculate the number and occurrences of each orbit

```
df['Orbit'].value_counts()
```

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

Appendix – Data Wrangling (1)

- Calculate the number of landing outcomes per outcome type:

```
landing_outcomes = df['Outcome'].value_counts()

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
```

- Creating set of outcomes where the second stage did not land successfully:

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

bad_outcomes
```

```
{'False ASDS', 'False Ocean', 'False RTLS', 'None ASDS', 'None None'}
```

Appendix – Data Wrangling (2)

- Create landing outcome label from outcome column:

```
landing_class = []
for outcome in df['Outcome']:
        if outcome in bad_outcomes:
                landing_class.append(0)
        else:
                landing_class.append(1)
df['Class']=landing_class
df.head(5)
```

BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins	Reused	Legs	LandingPad	Block	ReusedCount	Serial	Longitude	Latitude	Class
Falcon 9	6104.959412	LEO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B0003	-80.577366	28.561857	0
Falcon 9	525.000000	LEO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B0005	-80.577366	28.561857	0
Falcon 9	677.000000	ISS	CCAFS SLC	None	1	False	False	False	NaN	1.0	0	B0007	-80.577366	28.561857	0

- Calculate the success rate:

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

```
0.6666666666666666
```

Appendix – EDA with SQL

- Displaying the name of unique launch sites:

```
%sql SELECT DISTINCT("Launch_Site") FROM SPACEXTBL
```

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

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

```
%sql SELECT * FROM SPACEXTBL WHERE "Launch_Site" LIKE 'CCA%' LIMIT 5
```

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

Appendix – EDA with SQL (1)

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

```
%sql SELECT SUM(PAYLOAD_MASS__KG_) FROM SPACEXTBL WHERE Customer like '%NASA (CRS)%'
```

SUM(PAYLOAD_MASS__KG_)
48213

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

AVG(PAYLOAD_MASS__KG_)
2534.6666666666665

- The date when the first succesful landing outcome in ground pad was acheived

```
%%sql SELECT DATE FROM SPACEXTBL WHERE SUBSTR(Date,7,4) = (SELECT MIN(SUBSTR(Date,7,4)) FROM SPACEXTBL WHERE [Landing _Outcome] = 'Success (ground pad)') AND [Landing _Outcome] = 'Success (ground pad)'
```

Date
22-12-2015

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

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

Appendix – EDA with SQL (2)

- The total number of successful and failure mission outcomes:

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

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

- The names of the booster_versions which have carried the maximum payload mass

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

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

Appendix – EDA with SQL (3)

- Display the month names, failure landing_outcomes in drone ship ,booster versions, launch_site for the months in year 2015:

```
%%sql SELECT SUBSTR(Date, 4, 2) AS Month, [Landing _Outcome], "Booster_Version", "Launch_Site"
FROM SPACEXTBL
WHERE SUBSTR(Date, 7, 4) = '2015' AND [Landing _Outcome] = 'Failure (drone ship)'
```

Month	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

- 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 SUCCESS
FROM SPACEXTBL
WHERE [Landing _Outcome] LIKE '%Success%'
AND SUBSTR(DATE,7,4)||SUBSTR(DATE,4,2)||SUBSTR(DATE,1,2) BETWEEN '20100604' and '20170320'
GROUP BY [Landing _Outcome] ORDER BY SUCCESS DESC
```

Landing _Outcome	SUCCESS
Success (drone ship)	5
Success (ground pad)	3

Appendix – EDA with Visualization

- Visualize the relationship between Flight Number and Launch Site:

```
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()
```

- Visualize the relationship between Payload and Launch Site:

```
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()
```

- Visualize the relationship between success rate of each orbit type

```
mean = df.groupby(['Orbit'])['Class'].mean()
mean.plot(kind = 'bar', ylabel = 'Mean Success Rate')
```

- Visualize the relationship between FlightNumber and Orbit type

```
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()
```

Appendix – EDA with Visualization (1)

- Visualize the relationship between Payload and Orbit type:

```
sns.catplot(y = 'Orbit', x = 'PayloadMass', hue = 'Class', data = df, aspect = 5)
plt.xlabel('Pay load Mass (kg)', fontsize = 20)
plt.ylabel('Orbit', fontsize = 20)
plt.show()
```

- Visualize the launch success yearly trend:

```
year=[]
def Extract_year(date):
    for i in df["Date"]:
        year.append(i.split("-")[0])
    return year
year = Extract_year(df['Date']) df['Year'] = year

# Plot a line chart with x axis to be the extracted year and y axis to be the success rate
avg = df.groupby(['Year'])['Class'].mean()
avg.plot(kind = 'line', ylabel = 'Average Success Rate')
```

- Features engineering (select features, create dummy variables for categorical columns, cast numeric columns to float64):

```
features = df[['FlightNumber', 'PayloadMass', 'Orbit', 'LaunchSite', 'Flights', 'GridFins', 'Reused', 'Legs', 'LandingPad', 'Block', 'ReusedCount', 'Serial']]
dm1 = pd.get_dummies(df['Orbit’])
dm2 = pd.get_dummies(df['LaunchSite’])
dm3 = pd.get_dummies(df['LandingPad’])
dm4 = pd.get_dummies(df['Serial’])
frames = [features, dm1, dm2, dm3, dm4]
features_one_hot = pd.concat(frames)
features_one_hot.drop(columns=['Orbit','LaunchSite','LandingPad','Serial'], axis=1,inplace=True)
features_one_hot = features_one_hot.fillna(0)
features_one_hot = features_one_hot.astype(float)
```

Appendix – Interactive Map with Folium

- Marking all launch sites on a map:

```
#download and read data source
from js import fetch
import io
URL = 'https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/datasets/spacex_launch_geo.csv'
resp = await fetch(URL)
spacex_csv_file = io.BytesIO((await resp.arrayBuffer()).to_py())
spacex_df=pd.read_csv(spacex_csv_file)

# Select relevant sub-columns: `Launch Site`, `Lat(Latitude)`, `Long(Longitude)`, `class`
spacex_df = spacex_df[['Launch Site', 'Lat', 'Long', 'class']]
launch_sites_df = spacex_df.groupby(['Launch Site'], as_index=False).first()
launch_sites_df = launch_sites_df[['Launch Site', 'Lat', 'Long']]
launch_sites_df
```

	Launch Site	Lat	Long
0	CCAFS LC-40	28.562302	-80.577356
1	CCAFS SLC-40	28.563197	-80.576820
2	KSC LC-39A	28.573255	-80.646895
3	VAFB SLC-4E	34.632834	-120.610745

Appendix – Interactive Map with Folium (1)

- Create and add folium.Circle and folium.Marker for each launch site on the map:

```
nasa_coordinate = [29.559684888503615, -95.0830971930759]

# Initial the map
site_map = folium.Map(location=nasa_coordinate, zoom_start=4)

# For each launch site, add a Circle object based on its coordinate (Lat, Long) values. In addition, add Launch site name as a popup label
for index, site in launch_sites_df.iterrows():
   folium.Circle(
      [site['Lat'], site['Long']],
      radius = 50,
      color = '#d35400',
      fill = True
   ).add_child(folium.Popup(site['Launch Site'])).add_to(site_map)
   folium.map.Marker(
      [site['Lat'], site['Long']],
      icon = DivIcon(
         icon_size = (20,20),
         icon_anchor = (0,0),
         html = '<div style="font-size: 12; color:#d35400;"><b>%s</b></div>' % site['Launch Site']
      )
   ).add_to(site_map)
```

Appendix – Interactive Map with Folium (2)

- Marking the success/failed launches for each site on a map:

```
# Create a marker cluster object
marker_cluster = MarkerCluster()

# Function to assign color to launch outcome
def assign_marker_color(launch_outcome):
    if launch_outcome == 1:
        return 'green'
    else:
        return 'red'
spacex_df['marker_color'] = spacex_df['class'].apply(assign_marker_color)

# Add marker_cluster to current site_map
site_map.add_child(marker_cluster)

# for each row in spacex_df data frame, create a Marker object with its coordinate & customize the Marker's icon property to indicate success/failed
for index, record in spacex_df.iterrows():
    marker = folium.Marker(
        [record['Lat'], record['Long']],
        icon = folium.Icon(color='white', icon_color=record['marker_color'])
    )
    marker_cluster.add_child(marker)
```

Appendix – Interactive Map with Folium (3)

- Adding mouse position to get the coordinate (Lat, Long) for a mouse over on the map :

```
formatter = "function(num) {return L.Util.formatNum(num, 5);};”
mouse_position = MousePosition(
   position='topright’,  separator=' Long: ‘, empty_string='NaN’, lng_first=False, num_digits=20, prefix='Lat:’,
   lat_formatter=formatter, lng_formatter=formatter,)
site_map.add_child(mouse_position)
```

- Function to calculate distance between two points on the map:

```
from math import sin, cos, sqrt, atan2, radians
def calculate_distance(lat1, lon1, lat2, lon2):
   # approximate radius of earth in km
   R = 6373.0
   lat1 = radians(lat1)
   lon1 = radians(lon1)
   lat2 = radians(lat2)
   lon2 = radians(lon2)

   dlon = lon2 - lon1
   dlat = lat2 - lat1

   a = sin(dlat / 2)**2 + cos(lat1) * cos(lat2) * sin(dlon / 2)**2
   c = 2 * atan2(sqrt(a), sqrt(1 - a))

   distance = R * c
   return distance
```

Appendix – Interactive Map with Folium (4)

- Draw a line to the nearest coastline :

```
# calculating distance from CCAFS-SLC-40 to the closest coastline
distance = calculate_distance(28.56341, -80.57677, 28.56341, -80.56778)

# Create and add a folium.Marker to display the distance between coastline point and launch site using the icon property
distance_marker = folium.Marker(
   [28.56341, -80.56778], icon=DivIcon(
      icon_size=(20,20), icon_anchor=(0,0), html='<div style="font-size: 12; color:#d35400;"><b>%s</b></div>' % "{:10.2f} KM".format(distance),))

# Create a `folium.PolyLine` object using the coastline coordinates and launch site coordinate
lines=folium.PolyLine(locations=[[28.56341, -80.57677], [28.56341, -80.56778]], weight=1)
site_map.add_child(lines)
site_map.add_child(distance_marker)
```

- Draw a line to the nearest highway:

```
# calculating distance from CCAFS LC-40 to nearest highway
distance_hw = calculate_distance(28.56229, -80.57726, 28.56229, -80.57063)

distance_hw_marker = folium.Marker(
   [28.56229, -80.57063], icon=DivIcon(
      icon_size=(20,20), icon_anchor=(0,0), html='<div style="font-size: 12; color:#d35400;"><b>%s</b></div>' % "{:10.2f} KM".format(distance_hw),))

lines_hw=folium.PolyLine(locations=[[28.56229, -80.57726], [28.56229, -80.57063]], weight=1)

site_map.add_child(lines_hw)
site_map.add_child(distance_hw_marker)
```

Appendix – Interactive Map with Folium (5)

- Draw a line to the nearest railway :

```
# calculating distance from CCAFS-SLC-40 to the nearest railway
distance_rw = calculate_distance(28.56341, -80.57677, 28.57213, -80.58529)

distance_rw_marker = folium.Marker(

   [28.57213, -80.58529], icon=DivIcon(
      icon_size=(20,20), icon_anchor=(0,0), html='<div style="font-size: 12; color:#d35400;"><b>%s</b></div>' % "{:10.2f} KM".format(distance_rw),))

lines_rw=folium.PolyLine(locations=[[28.56341, -80.57677], [28.57213, -80.58529]], weight=1)

site_map.add_child(lines_rw)
site_map.add_child(distance_rw_marker)
```

- Draw a line to the nearest city:

```
# calculating distance from CCAFS LC-40 to Cape Canaveral
distance_ct = calculate_distance(28.56229, -80.57726, 28.40193, -80.60472)

distance_ct_marker = folium.Marker(
   [28.40193, -80.60472], icon=DivIcon(
      icon_size=(20,20), icon_anchor=(0,0), html='<div style="font-size: 12; color:#d35400;"><b>%s</b></div>' % "{:10.2f} KM".format(distance_ct),))

lines_ct=folium.PolyLine(locations=[[28.56229, -80.57726], [28.40193, -80.60472]], weight=1)

site_map.add_child(lines_ct)
site_map.add_child(distance_ct_marker)
```

Thank you!