IBM Developer
SKILLS NETWORK

Winning Space Race with Data Science

Ashutosh Agashe
21/11/2021

Outline

Executive Summary

- **Summary of methodologies**

➢ **Data Collection** : Collect necessary data from API and webscraping

➢ **Data Wrangling** : Remove unwanted and incompatible data, replace blank data with suitable values

➢ **Exploratory Data Analysis** : Using SQL and Pandas, find basic descriptive statistics of the data to gain an overall view

➢ **Data Visualization** : Using Interactive maps with folium and various types of plots to gain more insights

➢ **Dashboard** : Prepare a dashboard with plotly dash to view all necessary information at one single place

➢ **Predictive Analysis** : Classify the data using various algorithms and check which one works the best

- **Summary of all results**

➢ Exploratory Data Analysis results

➢ Interactive Maps and Dashboard results

➢ Predictive Analysis results

Link to github repository

Introduction

Project background and context

- SPACEX is a leading space exploration company having diverse success. Main reason of its dominance is the cost of space mission ($ 62 Mn vs > $165 Mn for competitors) .This is because first stage of its Falcon9 launches, which is largest and most expensive can be recovered and then reused. Thus, if we can predict if the first stage will land successfully, we can accurately calculate cost of each launch. This information is vital to the competitor company as well, because cost of the launch is key component for this business

Problems you want to find answers

- Rather than using complicated rocket science equations, we can use public data available to predict outcome of landing of first stage based on data analysis and machine learning. In this method, we should find :
- What parameters have impact on landing outcome?
- Can we accurately predict the landing outcome given all the necessary parameters ?
- How can SpaceX improve the landing success rate using optimum parameters and lower the launch cost further ?

Section 1

Methodology

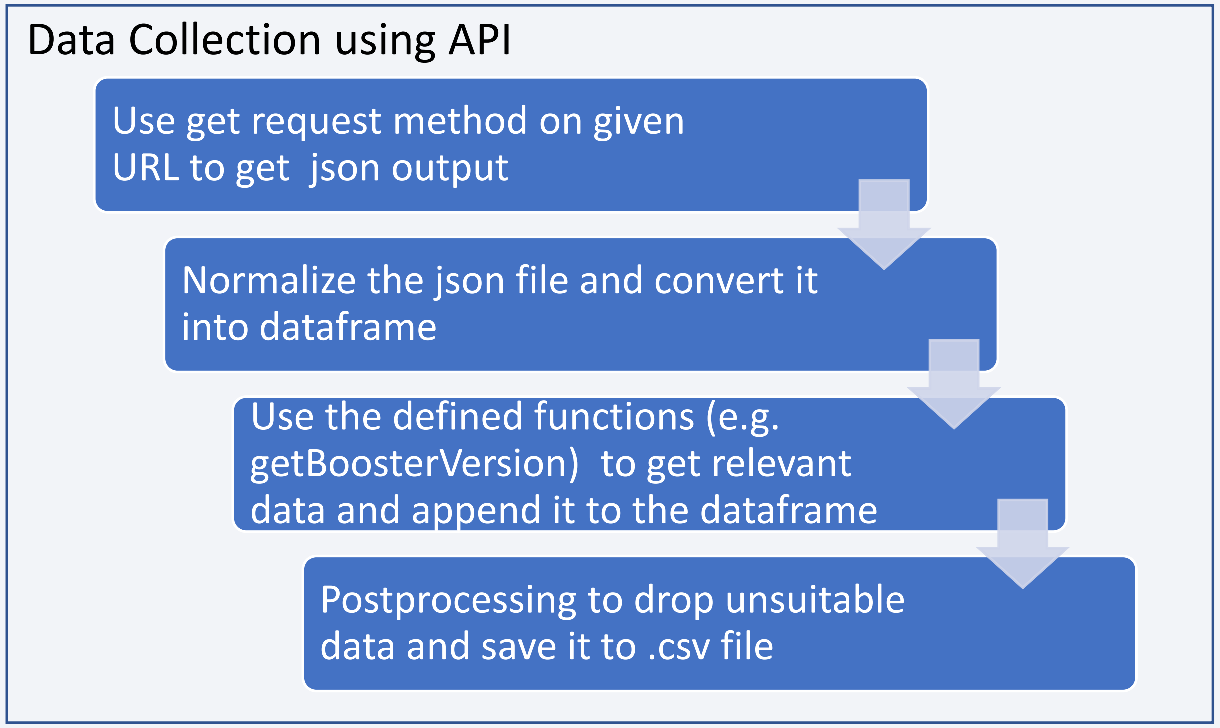
Methodology

- Executive Summary
- Data collection
  - Using SpaceX Rest API
  - Webscraping from wikipedia
- Perform data wrangling
  - Removing or replacing null values
  - One-hot-encoding for discrete variables to make data suitable for further analysis
- Perform exploratory data analysis (EDA) using visualization and SQL
  - Get descriptive statistics using SQL queries, visualize relationships between variables using barplot, catplot, scatterplot etc.
- Perform interactive visual analytics using Folium and Plotly Dash
  - Build interactive maps with folium to gain insights on launch site locations
  - Build interactive dashboard using plotly Dash to view yearly info. of launch success
- Perform predictive analysis using classification models
  - Evaluate various models on training data using different parameter sets and cross-validation
  - Find optimum parameter set as well as best model that predicts launch outcome accurately

Data Collection

- Data sets were collected by -

1. Using ‘SpaceX REST API’ : Get the data of past launches from specified URL
2. Webscraping from Wikipedia article



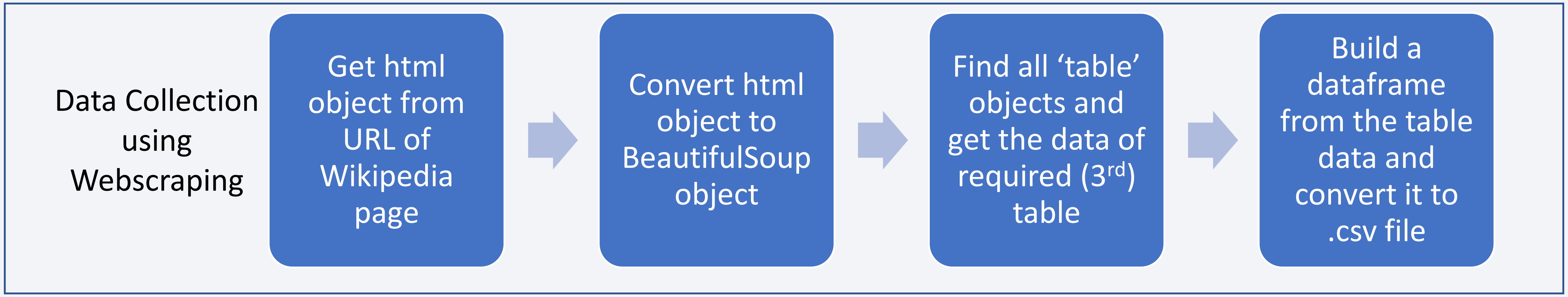
Data Collection using API

Use get request method on given URL to get json output

Normalize the json file and convert it into dataframe

Use the defined functions (e.g. getBoosterVersion) to get relevant data and append it to the dataframe

Postprocessing to drop unsuitable data and save it to .csv file



Data Collection using Webscraping

Get html object from URL of Wikipedia page

Convert html object to BeautifulSoup object

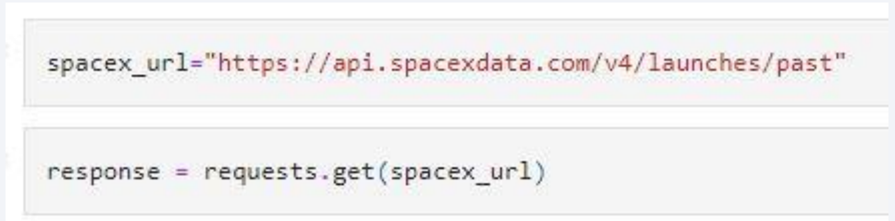
Find all ‘table’ objects and get the data of required (3rd) table

Build a dataframe from the table data and convert it to .csv file

Data Collection – SpaceX API

- Data collection with SpaceX REST API calls

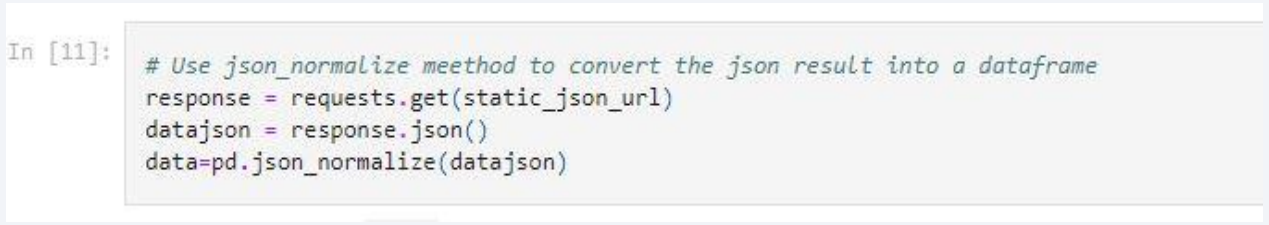
1) Obtain Data from URL using API call of requests.get



```
spacex_url="https://api.spacexdata.com/v4/launches/past"
```

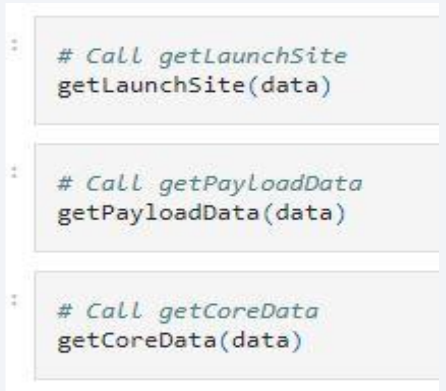
```
response = requests.get(spacex_url)
```

2) Convert the response to json format and normalize



```
In [11]:  # Use json_normalize meethod to convert the json result into a dataframe
          response = requests.get(static_json_url)
          datajson = response.json()
          data=pd.json_normalize(datajson)
```

4) From the flight number, get related data (about Booster version, launch site, payload) using API calls in the defined functions (e.g. getBoosterVersion) and save it in different columns

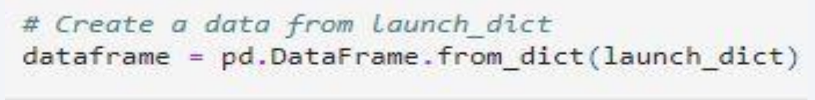


```
# Call getLaunchSite
getLaunchSite(data)
```

```
# Call getPayloadData
getPayloadData(data)
```

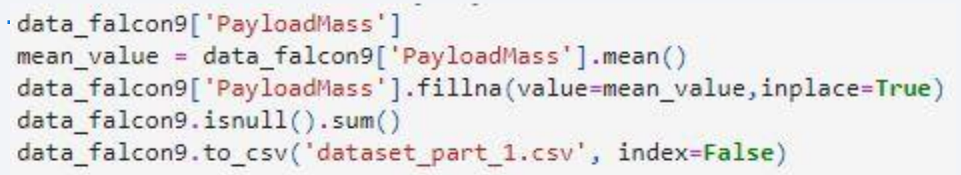
```
# Call getCoreData
getCoreData(data)
```

3) Prepare dictionary from the columns and convert it to dataframe



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

5) Perform wrangling and save it in .csv



```
data_falcon9['PayloadMass']
mean_value = data_falcon9['PayloadMass'].mean()
data_falcon9['PayloadMass'].fillna(value=mean_value,inplace=True)
data_falcon9.isnull().sum()
data_falcon9.to_csv('dataset_part_1.csv', index=False)
```

Link to Data Collection using API Notebook

Data Collection – Web Scraping

- Web scraping Process

1) Get html response from Wikipedia page URL

```
# use requests.get() method with the provided static_url
response = requests.get(static_url)
# assign the response to a object
Falcon9html = response
```

2) Convert html response to beautifulSoup object

```
# Use BeautifulSoup() to create a BeautifulSoup object
soup=BeautifulSoup(Falcon9html.text,'html.parser')
```

3) Extract all tables from object

```
tablex = soup.find_all('table')
table_headers=[]
for th in tablex:
    table_headers.append(th)
# Assign the result to a list called `html_tables`
html_tables = table_headers
html_tables
```

4) Get the data of required (3rd) table

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

5) Extract column names from table headers

```
column_names = []
launchtablecols= first_launch_table.find_all('th')
for colname in launchtablecols:
    colname1 = extract_column_from_header(colname)
    if not(pd.isnull(colname1)) and len(colname1) > 0 :
        column_names.append(colname1)
```

6) Create a dictionary with keys as column names and empty arrays as values, then append relevant values

```
launch_dict= dict.fromkeys(column_names)

if flag:
    extracted_row += 1
    # Flight Number value
    # TODO: Append the flight_number into launch_dict with key `Flight No.`
    launch_dict['Flight No.'].append(flight_number)
    #print(flight_number)
    datatimelist=date_time(row[0])
```

7) Convert dictionary to dataframe and save it to .csv file

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

Link to Data Collection by Webscraping Notebook

Data Wrangling

Two Objectives for this specific exercise :

1) Remove or replace null values present in the data (Done in data collection notebook, method explained in slide #8)
2) Assign training labels to the data (landing successful or not). This will be the output variable (to be predicted) in subsequent analysis

1) Get number of launches from each site, to each orbit and for each outcome type for better understanding

```
# Apply value_counts() on column LaunchSite
df['LaunchSite'].value_counts()
# Apply value_counts on Orbit column
df['Orbit'].value_counts()
landing_outcomes=df['Outcome'].value_counts()
landing_outcomes
```

2) Make a list of bad outcomes

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

3) Check if outcome is good or bad by checking if outcome belongs to list of bad outcomes, store it as a column of binary values (1 for success, 0 for failure)

```
landing_class=[]
OutcomeColumn=df['Outcome'].tolist()

for i in range(0,len(OutcomeColumn)):
    if OutcomeColumn[i] in bad_outcomes:
        Outcomevalue=0;
    else :
        Outcomevalue=1;
    landing_class.append(Outcomevalue)
```

4) Append the column to the dataframe

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

5) Check overall success rate by getting mean of column ‘Class’ and store the dataframe in .csv file

```
print(df["Class"].mean())
df.to_csv("dataset_part_2.csv", index=False)
```

Link to Data Wrangling Notebook

EDA with Data Visualization

Brief Summary of Plots

Sr. no.	Type of Plot	Y axis Variable	X axis Variable	Rationale of plot
1	Scatter Plot	Payload mass (Continuous)	Flight number (Discrete)	To check relationship between payload mass and landing success, also flight number and launch success. We can see that as flight number increases, success rate is more (improvement over time)
2	Scatter Plot	Launch site (Discrete)	Flight number (Discrete)	To check relationship between launch site and landing success. From plot, no significant difference is seen between different launch sites
3	Scatter plot	Launch site (Discrete)	Payload mass (Continuous)	To check relationship between launch site and payload mass. There is no significant relation seen in plot
4	Bar plot	Success Rate (Continuous)	Orbit type (Discrete)	To check success rates for different orbit types. For some orbits, it is 100% (e.g. ES-L1), and for SO, it is 0. Thus, there is strong relationship between success rate and orbit type
5	Scatter plot	Orbit Type (Discrete)	Flight number (Discrete)	To check relationship between orbit type and successful outcomes. Adding on to bar plot, it also displays number of flights. Similar insights can be drawn as that of bar plot
6	Scatter plot	Orbit Type (Discrete)	Payload Mass (Continuous)	To check relationship between orbit type and payload mass, if success rate depends on payload mass for each orbit type.
7	Line plot	Success Rate (Continuous)	Launch year (Discrete)	To check how the success rate changes over time. It is evident that over time, success rate is increasing

Link to EDA with Data Visualization Notebook

EDA with SQL

- SQL queries performed to get relevant information :

➢ Display names of unique launch sites

➢ Display 5 records where name of launch site begins with ‘CCA’

➢ Display total payload mass (in kgs.) carried by boosters launched by NASA (CRS)

➢ Display average payload mass (in kgs.) carried by booster version F9 v1.1

➢ Display the date at which first successful landing outcome in ground pad was achieved

➢ Display the list of boosters which have success in drone ship landing and payload mass between 4000 and 6000 kgs.

➢ Display the list of total successful and failed mission outcomes

➢ Display the list of booster versions which have carried maximum payload mass

➢ Display the list of launch site and booster version for failed landing outcomes in drone ship in year 2015

➢ Rank the landing outcomes according to frequency in descending order between 2010-06-04 and 2017-03-20

Link to EDA with SQL Notebook

Build an Interactive Map with Folium

- **Step 1**
  - ➢ Prepared a map showing location of all launch sites
  - ➢ Used circle object to specify location and marker object to display name of launch site. Circle object also displays a pop-up with launch site name on clicking
- **Step 2**
  - ➢ Showed successful and unsuccessful launches from each site by color-coding (Green for successful, i.e. class=1; and Red for unsuccessful, i.e. class=0)
  - ➢ This is done by adding a column of marker color to dataframe based on launch outcome, inclusion of that color to the marker object by assigning column value to icon color and then clustering the outcomes using MarkerCluster() object
- **Step 3**
  - ➢ Calculated distances of nearest coastline, railways, highways and cities from launch sites
  - ➢ First added mouse position child to map, then obtained co-ordinates of required place by surfing the map
  - ➢ Calculated the distance between launch site and required location (co-ordinates are known by mouse position) using defined function
  - ➢ Created distance marker object displaying line connecting site and location as well as distance between them and added to the map

Link to Interactive Visual Analytics with Folium Notebook

Build a Dashboard with Plotly Dash

- Motivation : Visualizing relationship between Launch site, Payload mass, Booster version category and Outcome
- Primary objective of the dashboard is to visualize these relationships at one place with required details. It also eliminates the complexity of analysis and shows necessary results in compact and crisp manner
- Dashboard has 2 interactive elements : Dropdown and Slider, and 2 plots : 1 Pie plot and 1 Scatter plot
- Interactive nature of dashboard enables us to view outcomes for specific launch site as well as for specific payload mass range
- **Dropdown** enables us to select specific launch site
- **Pie plot** helps us see proportion of total successful launches by launch site as well as proportion of success or failure for specific site if it is selected through dropdown. Thus, pie plot gives us overall as well as granular view based on our requirement
- Payload mass **slider** helps us select a specific range in which we want to see outcomes
- **Scatter plot** takes launch site and payload slider range as inputs and displays all successful and failure outcomes for the specific input. It also color-codes data points by booster version category, so we can see its impact on outcome as well

Link to Notebook generating Dashboard with Plotly Dash

Predictive Analysis (Classification)

- **Preparing the data-set for analysis**

1. Load the data with continuous variables, one-hot-encoded discrete variables and binary output variable into a dataframe
2. Normalize the data (using preprocessing.StandardScaler().fit_transform(Dataframe) function)
3. Split the data in training and testing sets using train_test_split() function, check size of testing data set (should be 18)

- **Building models for analysis**

1. Create an object of desired model (e.g. LogisticRegression() for logistic regression)
2. Create a dictionary of parameter sets on which analysis is to be performed (Parameters as keys, their values as values)
3. Create a GridSearchCV() object with CV value 10 for cross-validation for each model

- **Optimizing the model**

1. Train the model with GridSearchCV object (using .fit(X_train,Y_train) function, this evaluates the model over all parameter sets)
2. Get best parameter set for the model and its accuracy (using .best_params_ and .best_score_)
3. Check the accuracy (score) of model with best parameter set on testing data (using .score(X_test, Y_test))
4. Plot confusion matrix for testing data set for each model

- **Best model is the one having highest score on testing data set**

Link to Predictive Modelling Notebook

Results

Exploratory data analysis results

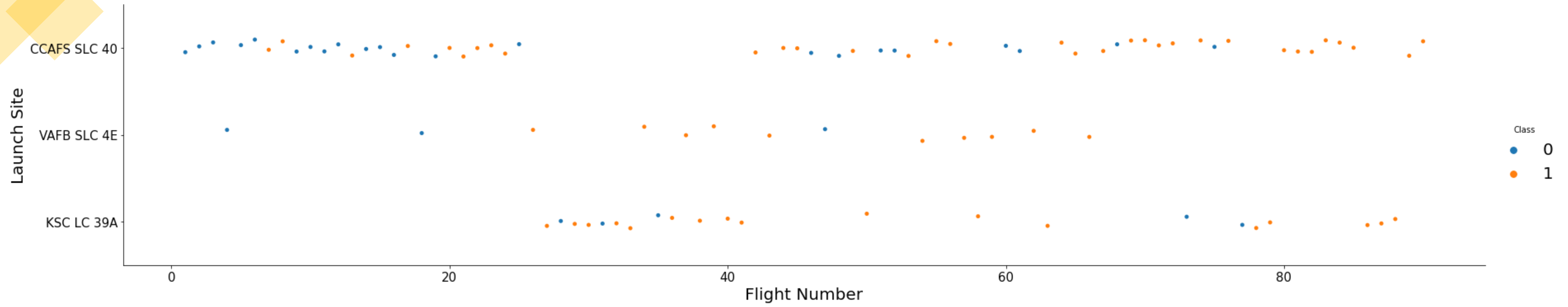
Interactive analytics demo in screenshots

Predictive analysis results

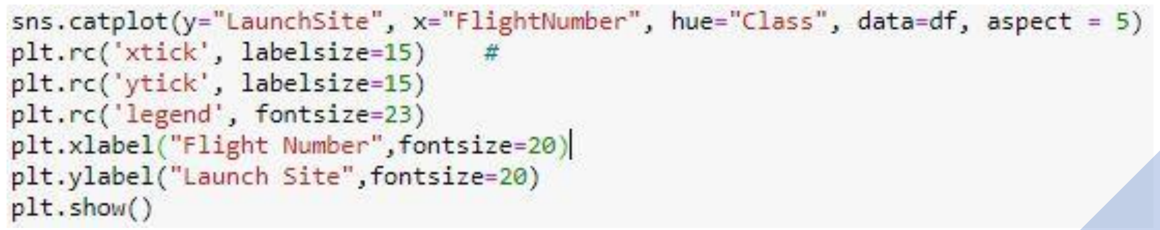
Section 2

Insights drawn from EDA

Launch Site vs. Flight Number

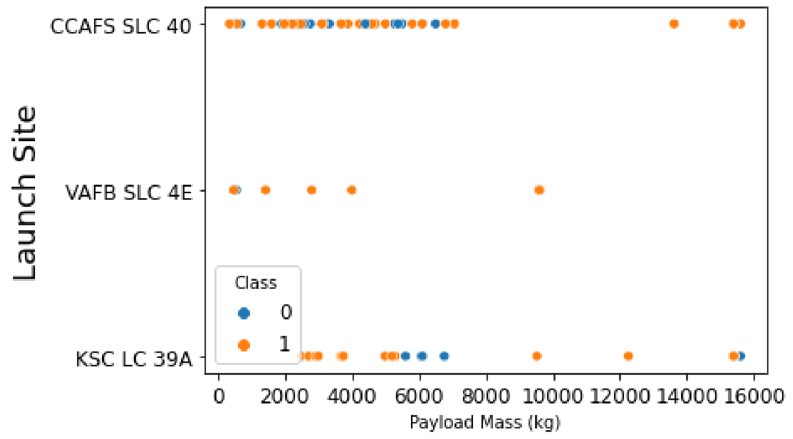


- This plot is used to visualize relationship between Launch site, flight number and landing success
- Success rate for larger flight number is more, thus we can say that over time, success rate has improved
- No significant difference is visible for success rates in different launch sites
- Earlier launches took place predominantly in launch site CCAFS SLC-40



```
sns.catplot(y="LaunchSite", x="FlightNumber", hue="Class", data=df, aspect = 5)
plt.rc('xtick', labelsize=15)    #
plt.rc('ytick', labelsize=15)
plt.rc('legend', fontsize=23)
plt.xlabel("Flight Number",fontsize=20)
plt.ylabel("Launch Site",fontsize=20)
plt.show()
```

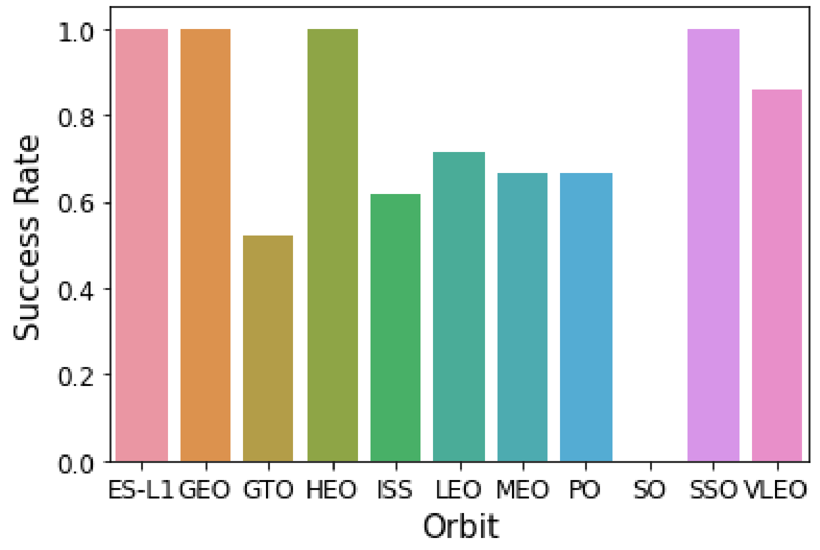
Launch Site vs. Payload

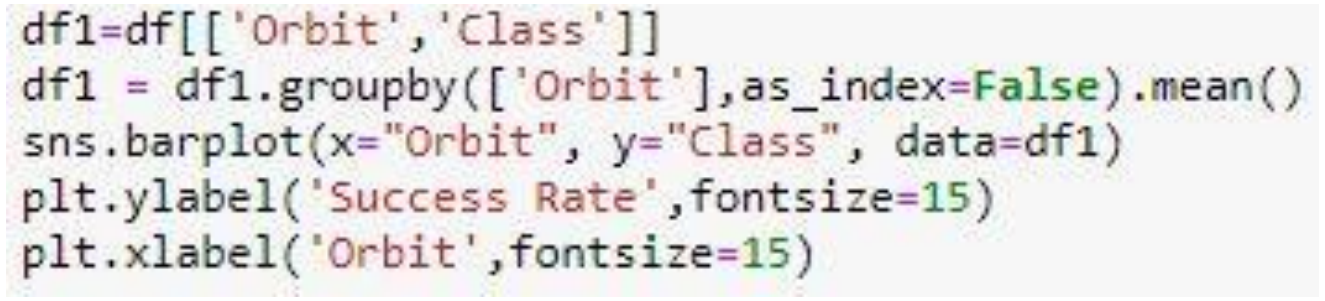


```
sns.scatterplot(y="LaunchSite", x="PayloadMass", hue="Class", data=df)
plt.rc('xtick', labelsize=12)    #
plt.rc('ytick', labelsize=12)
plt.rc('legend', fontsize=10)
plt.xlabel("Payload Mass (kg)",fontsize=10)
plt.ylabel("Launch Site",fontsize=18)
plt.show()
```

- This plot is used to visualize relationship between Launch site, payload mass and landing success
- Majority of launches are with lesser payload (< 8000 kg)
- However, success rate is very high for higher payloads i.e. > 8000 kg (just 1 failure in 8 launches)
- Between Launch site and payload mass, no significant relationship is visible

Success Rate vs. Orbit Type

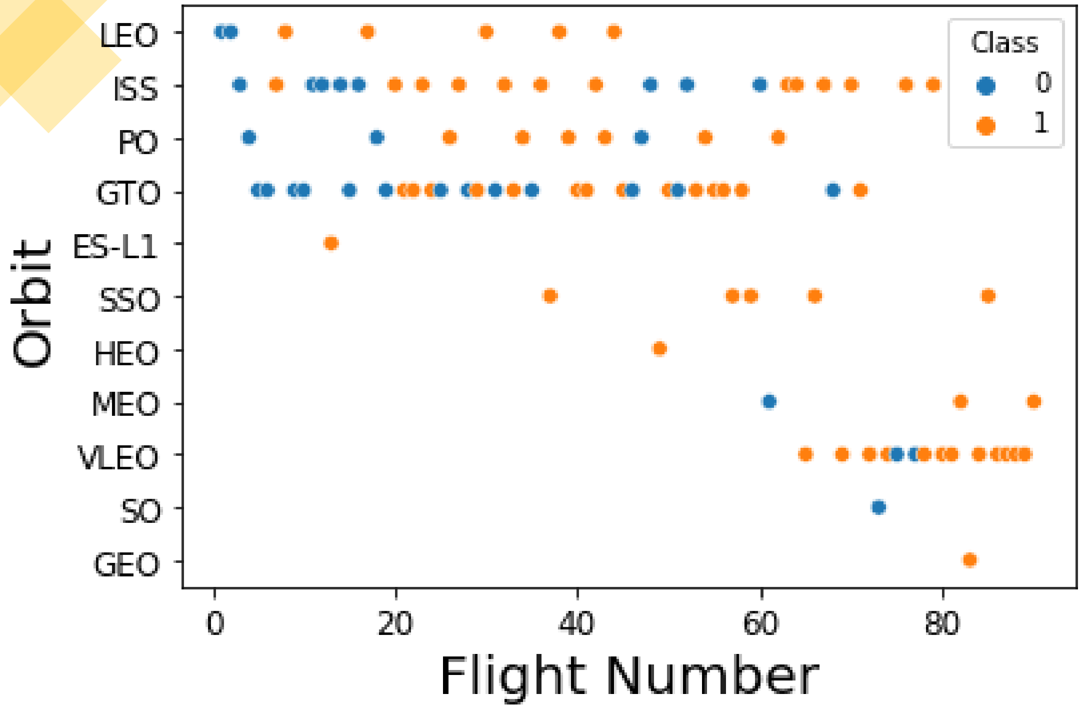


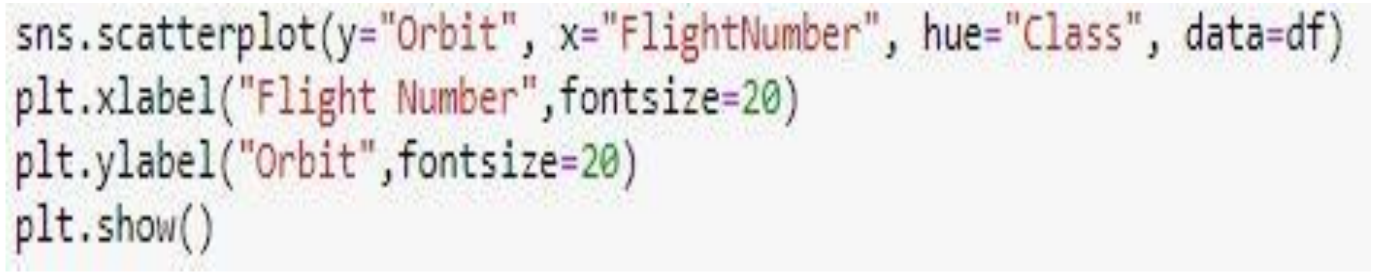


```
df1=df[['Orbit','Class']]
df1 = df1.groupby(['Orbit'],as_index=False).mean()
sns.barplot(x="Orbit", y="Class", data=df1)
plt.ylabel('Success Rate',fontsize=15)
plt.xlabel('Orbit',fontsize=15)
```

- This plot shows success rates for different orbits into which launches took place
- Orbit like ES-L1, GEO, SSO, ISS have 100% success rate, while SO orbit has 0% success rate
- Significant difference is visible for success rate for different orbits, so it will be an important parameter when predicting the outcome for new data

Orbit Type vs. Flight Number

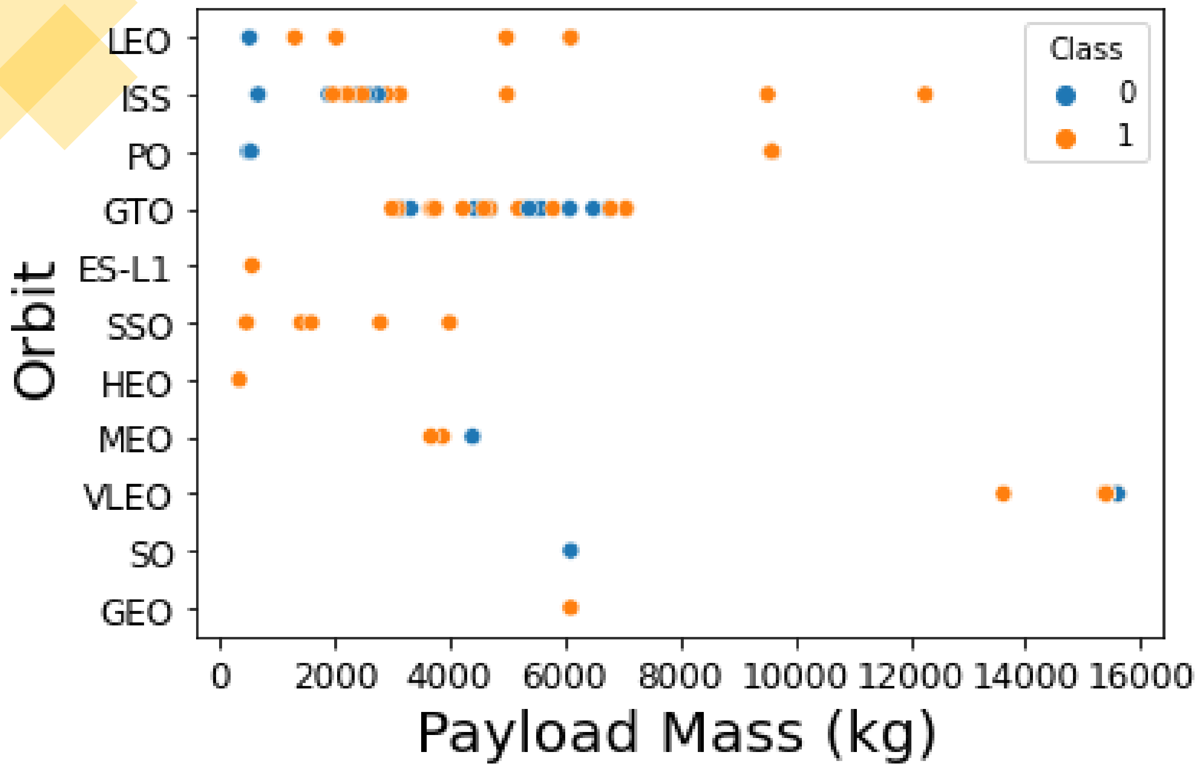




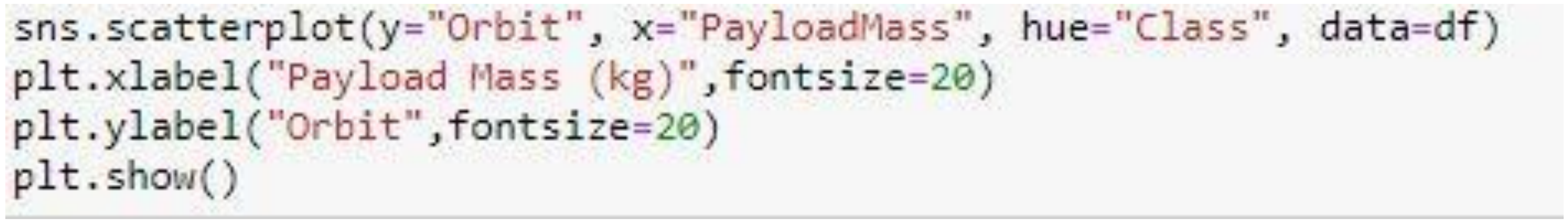
```
sns.scatterplot(y="Orbit", x="FlightNumber", hue="Class", data=df)
plt.xlabel("Flight Number",fontsize=20)
plt.ylabel("Orbit",fontsize=20)
plt.show()
```

- This plot shows relationship between Orbit type and flight number
- Majority of early launches are in orbit types of LEO, ISS, PO, GTO and ES-L1; orbit types of SSO, HEO, MEO, VLEO, SO and GEO have only later launches (i.e. larger flight number)
- LEO, ISS, PO and GTO have earlier as well as later launches

Orbit Type vs. Payload

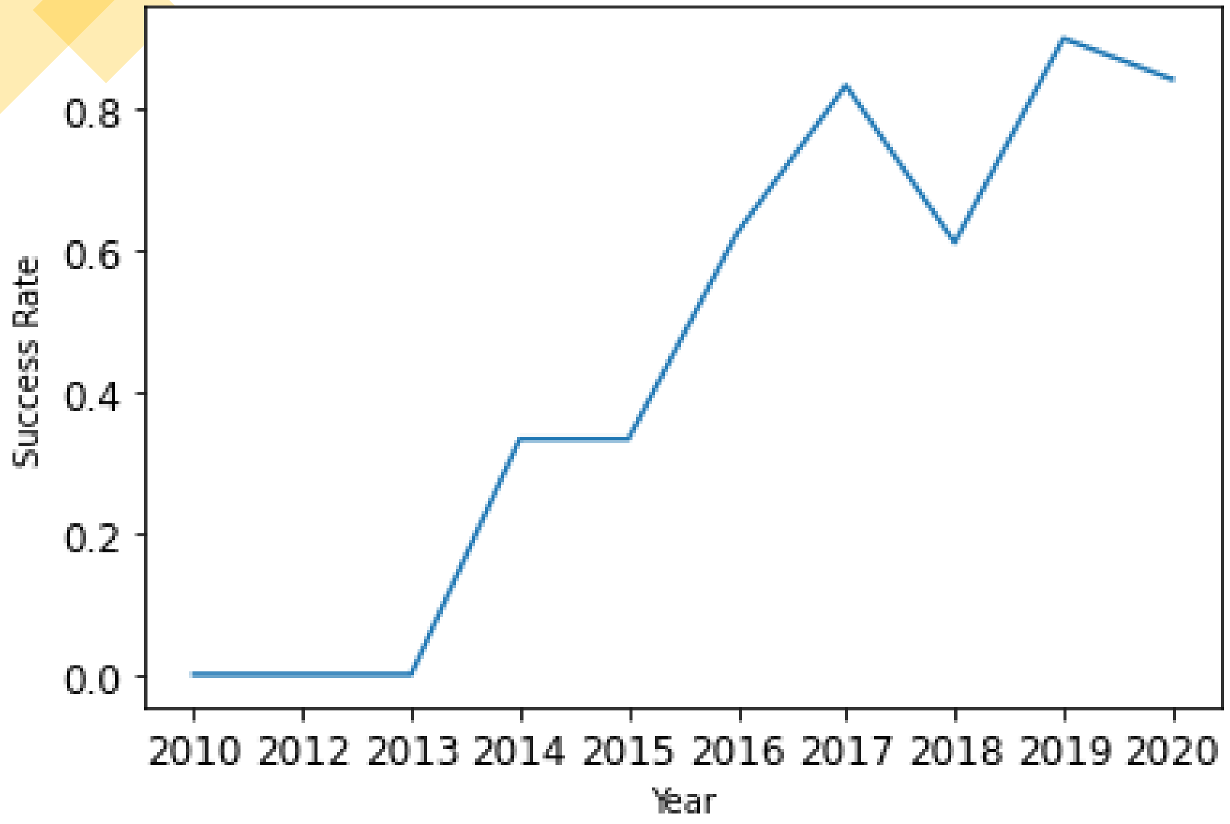


- For orbits LEO,GTO, ES-L1, SSO, HEO, SO, GEO and MEO; launches only with smaller payload took place (< 8000 kg)
- For VLEO orbit, all launches are with higher payload (>8000 kg)
- For ISS and PO orbits, both smaller and larger payload mass launches are present
- Thus, there is significant relationship in payload mass and orbit type

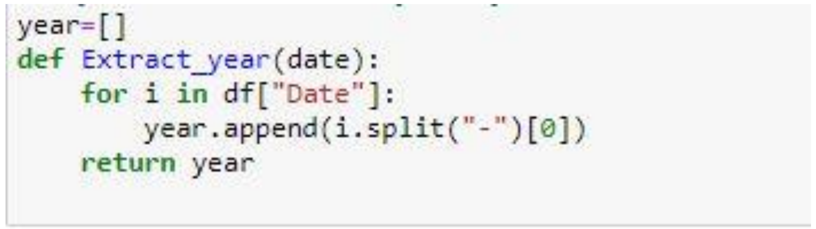


```
sns.scatterplot(y="Orbit", x="PayloadMass", hue="Class", data=df)
plt.xlabel("Payload Mass (kg)",fontsize=20)
plt.ylabel("Orbit",fontsize=20)
plt.show()
```

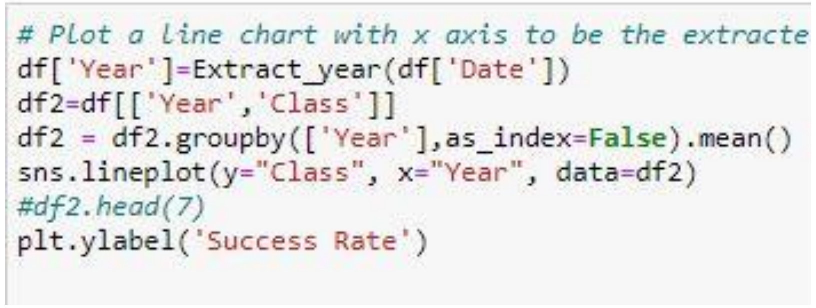
Launch Success Yearly Trend



- This chart explains trend of success rate over time
- Success rate has shown overall increasing trend with small dips in years 2018 and 2020
- Significant progress in success rate is visible from 0% in 2013 to > 80% in 2019

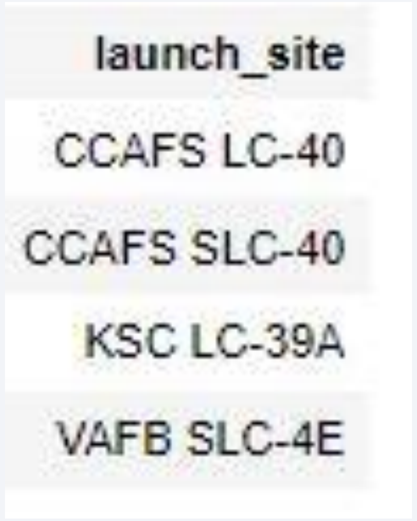


```
year=[]
def Extract_year(date):
    for i in df["Date"]:
        year.append(i.split("-")[0])
    return year
```

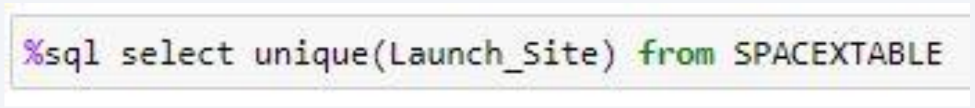


```
# Plot a line chart with x axis to be the extracte
df['Year']=Extract_year(df['Date'])
df2=df[['Year','Class']]
df2 = df2.groupby(['Year'],as_index=False).mean()
sns.lineplot(y="Class", x="Year", data=df2)
#df2.head(7)
plt.ylabel('Success Rate')
```

All Launch Site Names



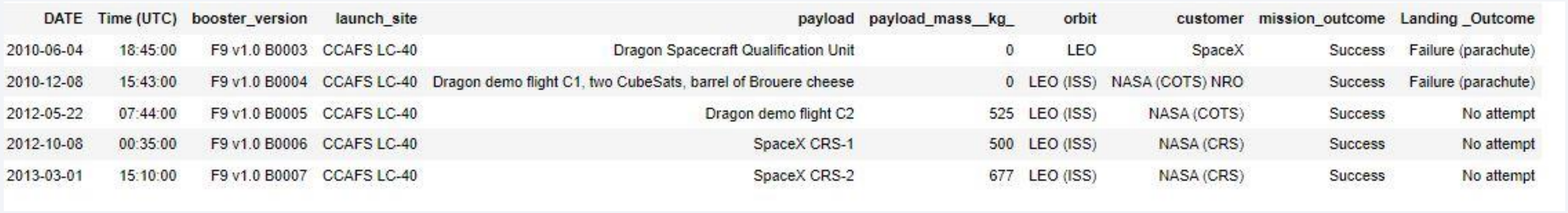
launch_site
CCAFS LC-40
CCAFS SLC-40
KSC LC-39A
VAFB SLC-4E

- SQL Query to get all launch site names : 

```
%sql select unique(Launch_Site) from SPACEXTABLE
```

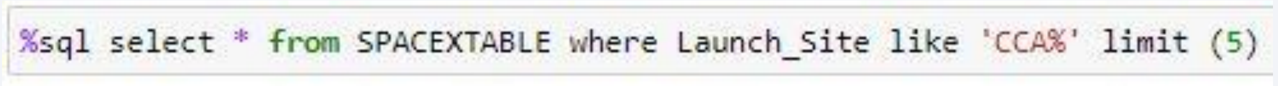
- unique() → gets unique values from input
- Column 'Launch_Site' is provided as in input from data array 'SPACEXTABLE'
- There are 4 different launch sites as seen in the output

Launch Site Names that Begin with 'CCA'



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	07:44:00	F9 v1.0 B0005	CCAFS LC-40	Dragon demo flight C2	525	LEO (ISS)	NASA (COTS)	Success	No attempt
2012-10-08	00:35:00	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

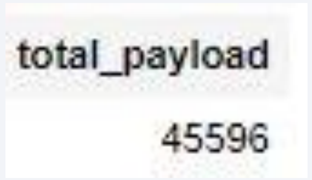
- SQL Queiry performed :



```
%sql select * from SPACEXTABLE where Launch_Site like 'CCA%' limit (5)
```

- Using * will display entire row in the data
- ‘where’ is used to include a logical test, if that test is satisfied, data will be included in output
- ‘X like Y’ check equality of X and Y. Whichever data in X matches with Y will satisfy the logical test (here, checking which strings in Launch_Site column match with ‘CCA’)
- % sign at the end signifies to look for only the start of string, i.e. if a string in input starts with required characters, logical test is satisfied. Input string does not have to be exact match
- limit() signifies number of data points to be shown

Total Payload Mass



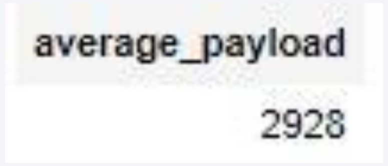
total_payload
45596

- SQL query performed :

```
%sql SELECT sum(payload_mass__kg_) as total_payload FROM SPACEXTABLE WHERE customer LIKE 'NASA (CRS)' Group By customer
```

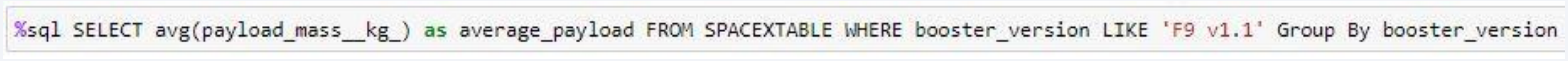
- Where customer like ‘NASA (CRS)’ checks exact matching of strings in customer column with ‘NASA (CRS)’
- Group By customer brings all datasets together which have same customer name (here, NASA(CRS) as we have already filtered the dataset). Applying this is not mandatory though
- Sum() performs summation of all entries in the input
- ‘as total_payload’ creates a new variable named ‘total_payload’ and assigns predefined value (here, sum(paload__mass_kg_)) to it
- Total mass of payloads carried for customer NASA (CRS) is 45596 kg

Average Payload Mass by F9 v1.1



average_payload
2928

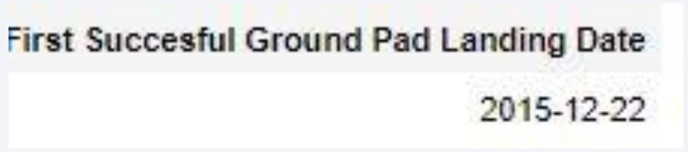
- SQL query performed :



```
%sql SELECT avg(payload_mass__kg_) as average_payload FROM SPACEXTABLE WHERE booster_version LIKE 'F9 v1.1' Group By booster_version
```

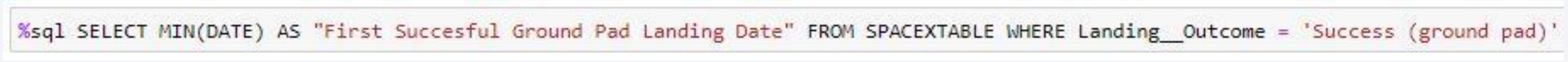
- {WHERE booster_version like ‘F9 v1.1’ } will filter the dataset and only those entries in booster_version that perfectly match with ‘F9 v1.1’ will remain
- avg(payload_mass__kg_) will calculate average of all entries in column ‘payload_mass__kg_’
- ‘as average_payload’ creates new variables called ‘average_payload’ and assigns value of the average to it
- Average payload mass carried by booster version F9 v1.1 is 2928 kg
- Using Group By is optional and does not change the result

First Successful Ground Landing Date



First Succesful Ground Pad Landing Date
2015-12-22

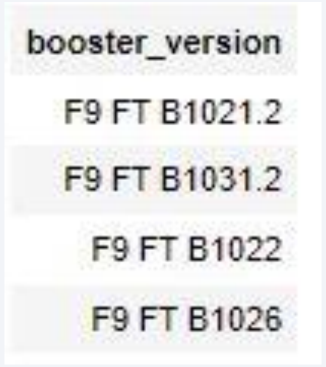
- SQL query performed :



```
%sql SELECT MIN(DATE) AS "First Succesful Ground Pad Landing Date" FROM SPACEXTABLE WHERE Landing__Outcome = 'Success (ground pad)'
```

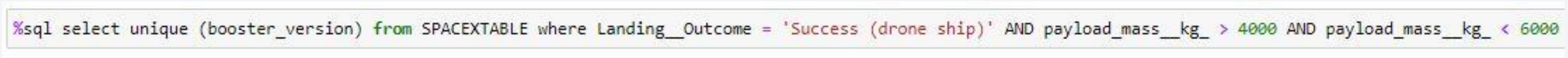
- {WHERE Landing__Outcome = ‘Success (ground pad)’} filters the dataset and only those entries in Landing__Outcome column have exact match with ‘Success (ground pad)’ will remain
- MIN(DATE) selects minimum value from ‘DATE’ column
- AS “First Successful Ground Pad Landing Date” creates a new variable and assigns the minimum value to it

Successful Drone Ship Landing with Payload between 4000 and 6000



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

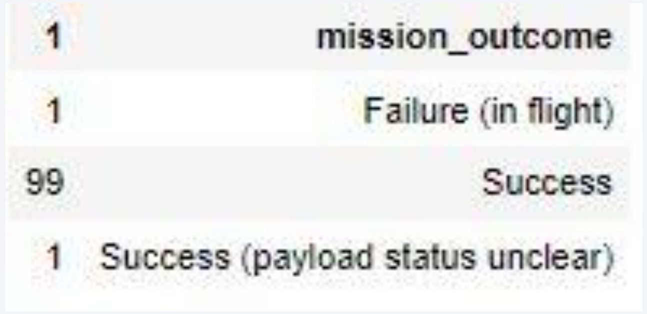
- SQL query performed :



```
%sql select unique (booster_version) from SPACEXTABLE where Landing__Outcome = 'Success (drone ship)' AND payload_mass__kg_ > 4000 AND payload_mass__kg_ < 6000
```

- {WHERE X AND Y AND Z} will filter the dataset and the entries which satisfy all 3 logical conditions X, Y and Z will remain. (AND operator is used as satisfying all conditions is necessary)
- Condition X : Entries in column 'Landing__Outcome' should exactly match with string 'Success (drone ship)'
- Condition Y : Entries in column 'payload_mass__kg_' should have value greater than 4000
- Condition Z : Entries in column 'payload_mass__kg_' should have value lesser than 6000
- Unique (booster_version) will select unique values in column 'booster_version' in the filtered datasets. This will avoid displaying repeat results

Total Number of Successful and Failure Mission Outcomes



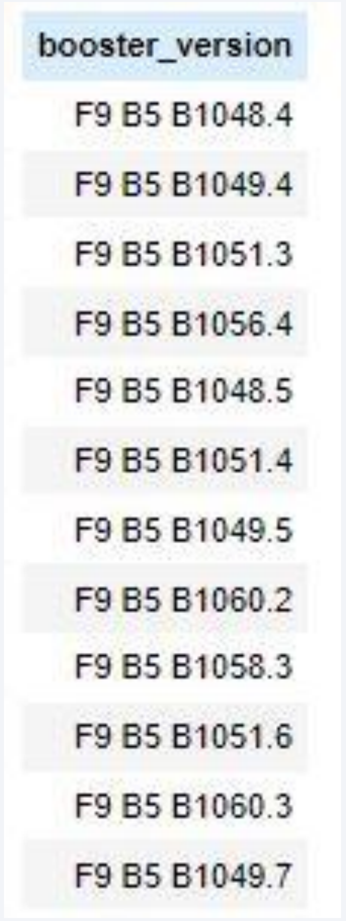
1	mission_outcome
1	Failure (in flight)
99	Success
1	Success (payload status unclear)

- SQL query performed :

```
%sql SELECT count(MISSION_OUTCOME),MISSION_OUTCOME FROM SPACEXTABLE Group By MISSION_OUTCOME
```

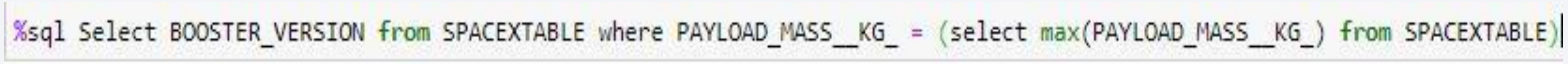
- Group By MISSION_OUTCOME will group the entries with same value in column 'MISSION_OUTCOME' together
- Count() will count the number of entries in the column
- Total successful outcomes are 100 (99 +1)
- Total failed outcomes is 1

Boosters that Carried Maximum Payload



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

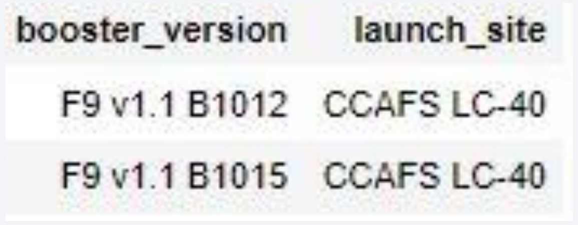
- SQL query performed :



```
%sql Select BOOSTER_VERSION from SPACEXTABLE where PAYLOAD_MASS__KG_ = (select max(PAYLOAD_MASS__KG_) from SPACEXTABLE)
```

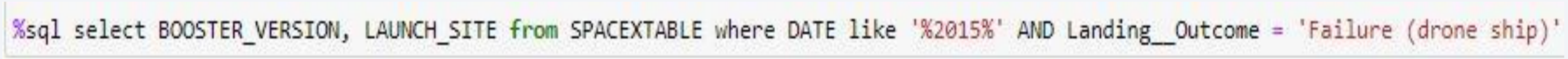
- (select max(PAYLOAD_MASS__KG_) from SPACEXTABLE) calculates maximum value in the column ‘PAYLOAD_MASS__KG_’
- {where PAYLOAD_MASS__KG_ = (max)} filters data and only entries which have same value as max. value in column ‘PAYLOAD_MASS__KG_’ remain
- select BOOSTER_VERSION displays entries in ‘BOOSTER_VERSION’ column in the filtered dataset

2015 Launch Records



booster_version	launch_site
F9 v1.1 B1012	CCAFS LC-40
F9 v1.1 B1015	CCAFS LC-40

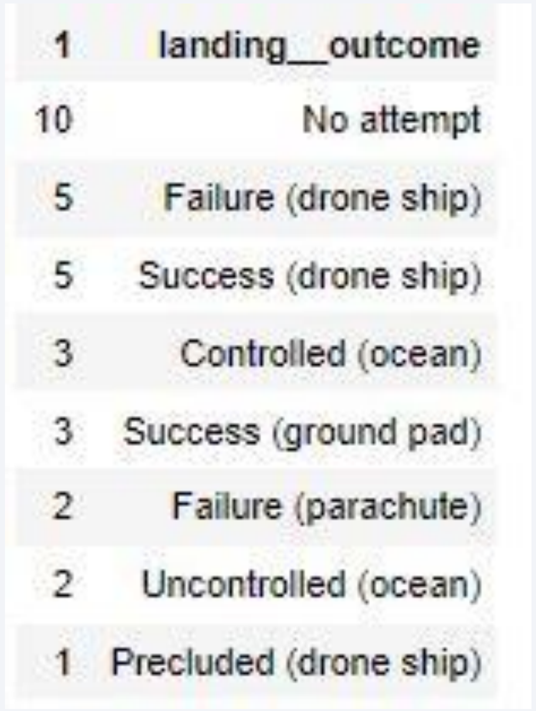
- SQL query performed :



```
%sql select BOOSTER_VERSION, LAUNCH_SITE from SPACEXTABLE where DATE like '%2015%' AND Landing__Outcome = 'Failure (drone ship)'
```

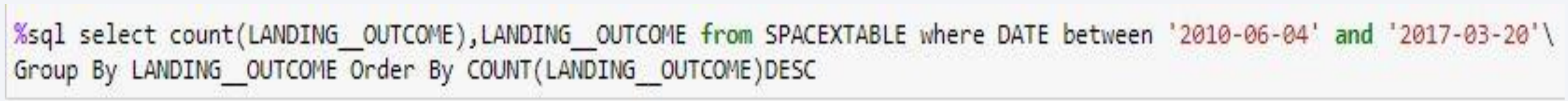
- {DATE like '%2015%'} checks if string '2015' appears anywhere in the entries in column 'DATE' (this may be at start, end or middle, all will satisfy)
- {Landing__Outcome = 'Failure (drone ship)'} checks if entries in column 'Landing_Outcome' match exactly with string 'Failure (drone ship)'
- {WHERE A AND B} will filter the data and only entries which satisfy both conditions A and B will remain. Here, condition A is for checking where Date is in year 2015, condition B is for checking whether landing outcome is failure in drone ship
- select BOOSTER_VERSION, LAUNCH_SITE will display these 2 columns of the filtered dataset
- There are 2 drone ship failures in 2015, for which booster version and launch site are listed above

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



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

- SQL query performed :



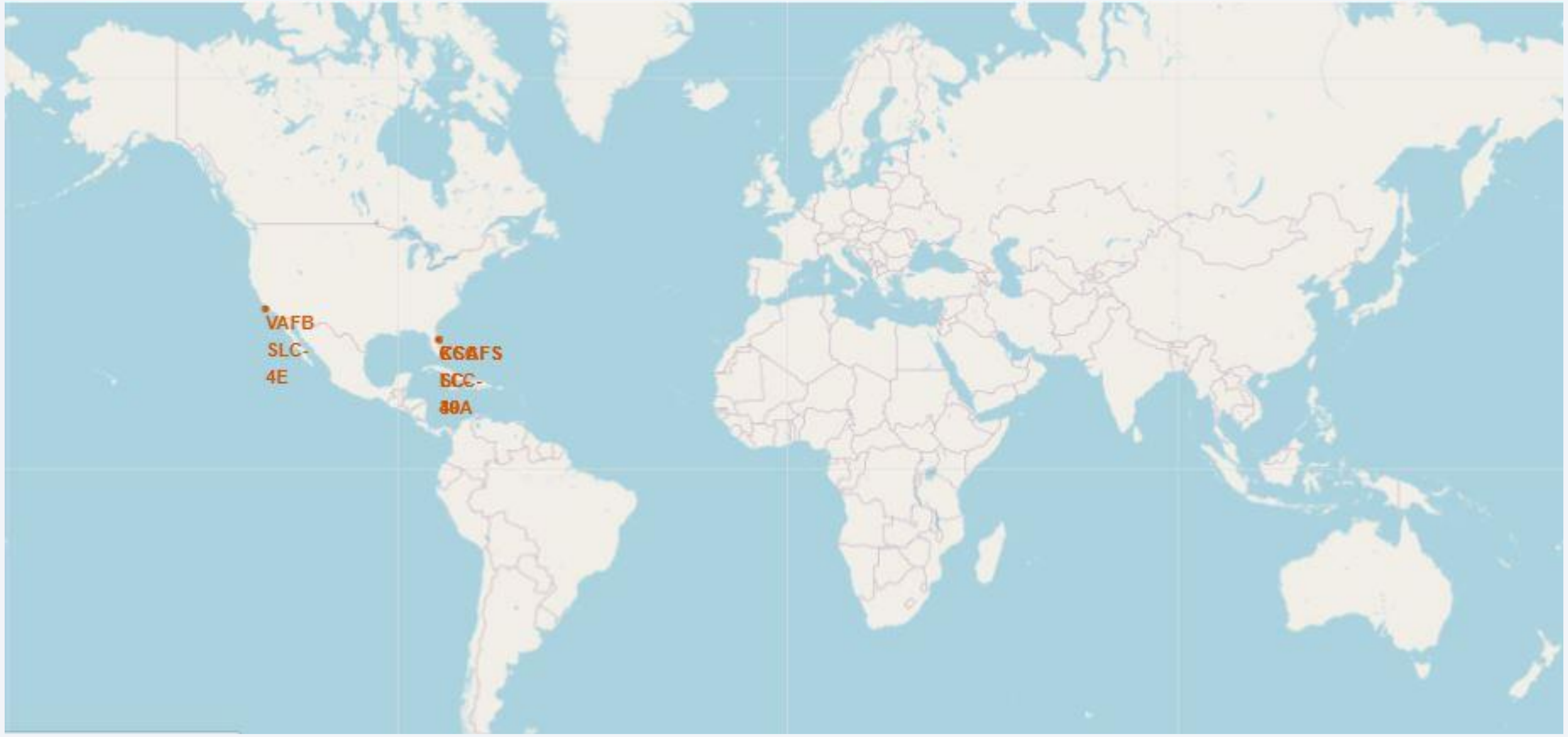
```
%sql select count(LANDING__OUTCOME),LANDING__OUTCOME from SPACEXTABLE where DATE between '2010-06-04' and '2017-03-20'\
Group By LANDING__OUTCOME Order By COUNT(LANDING__OUTCOME)DESC
```

- {where DATE between X and Y} will filter the dataset and only entries in column 'DATE' that fall between X and Y will remain
- {Group By LANDING__OUTCOME} will group the data having same values in column 'LANDING__OUTCOME'
- {Order By COUNT(LANDING__OUTCOME) DESC} will arrange the data in descending order based on values in column COUNT(LANDING__OUTCOME)
- {select count(LANDING__OUTCOME),LANDING__OUTCOME} will display the data in those 2 columns
- Count() function counts the frequency of value in column

Section 4

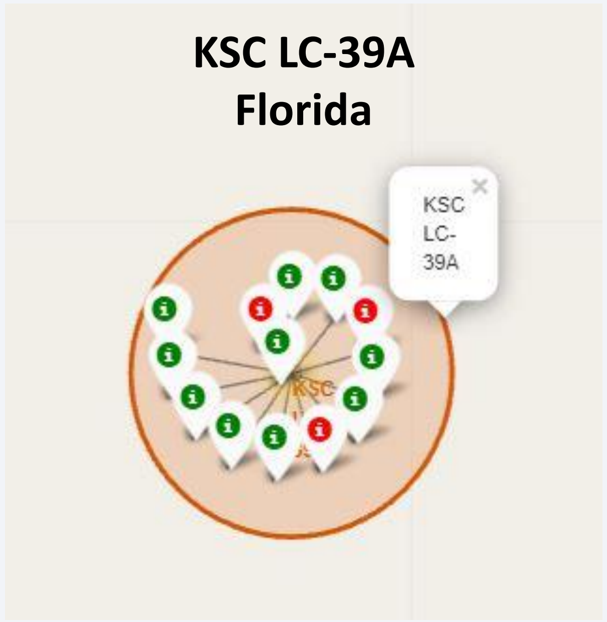
Launch Sites Proximities Analysis

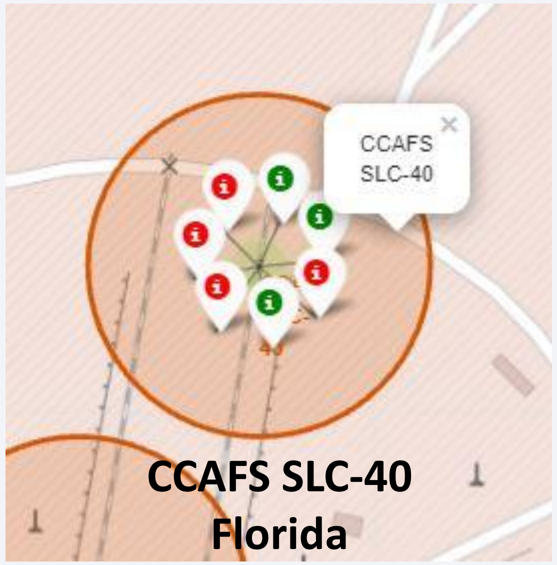
SPACEX Launch Site Locations

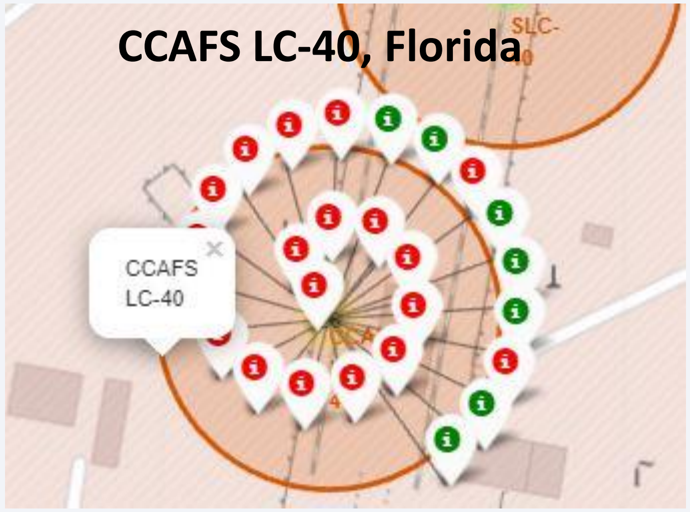


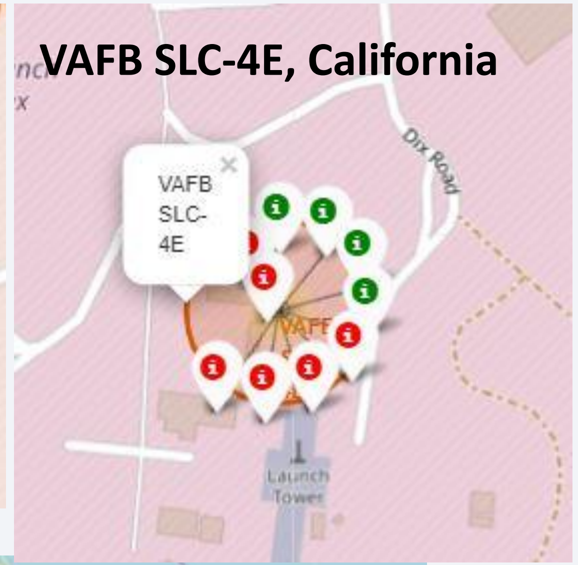
- All 4 launch sites are in USA : 3 on East coast (Florida) and 1 on West coast (California)
- All launch sites are in close proximity to coastline, but away from equator

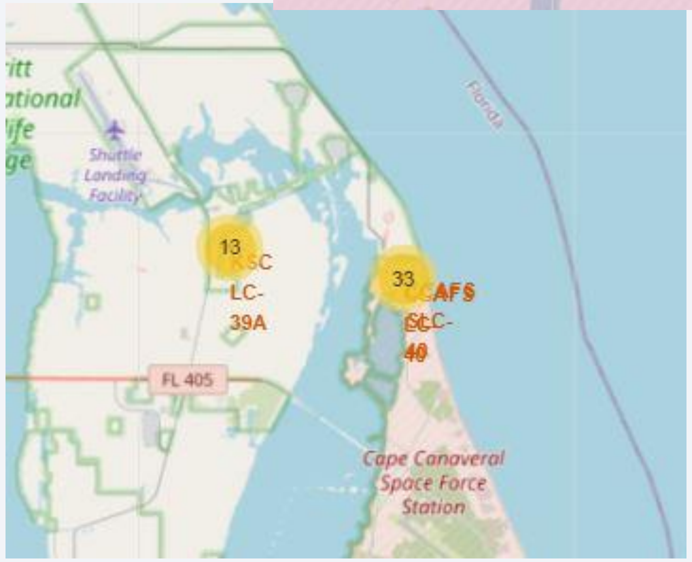
Successful and Failed Launches from each site











- Successful and failed launches from all sites shown above
- If zoomed out, markers cluster together as shown in figure on the right
- On looking at map images, success rate for KSC LC-39A is very high
- Number of launches from CCAFS LC-40 are highest and from CCAFS SLC-40 are lowest

Proximities of Launch Sites

CCAFS SLC-40 is selected as representative Launch site

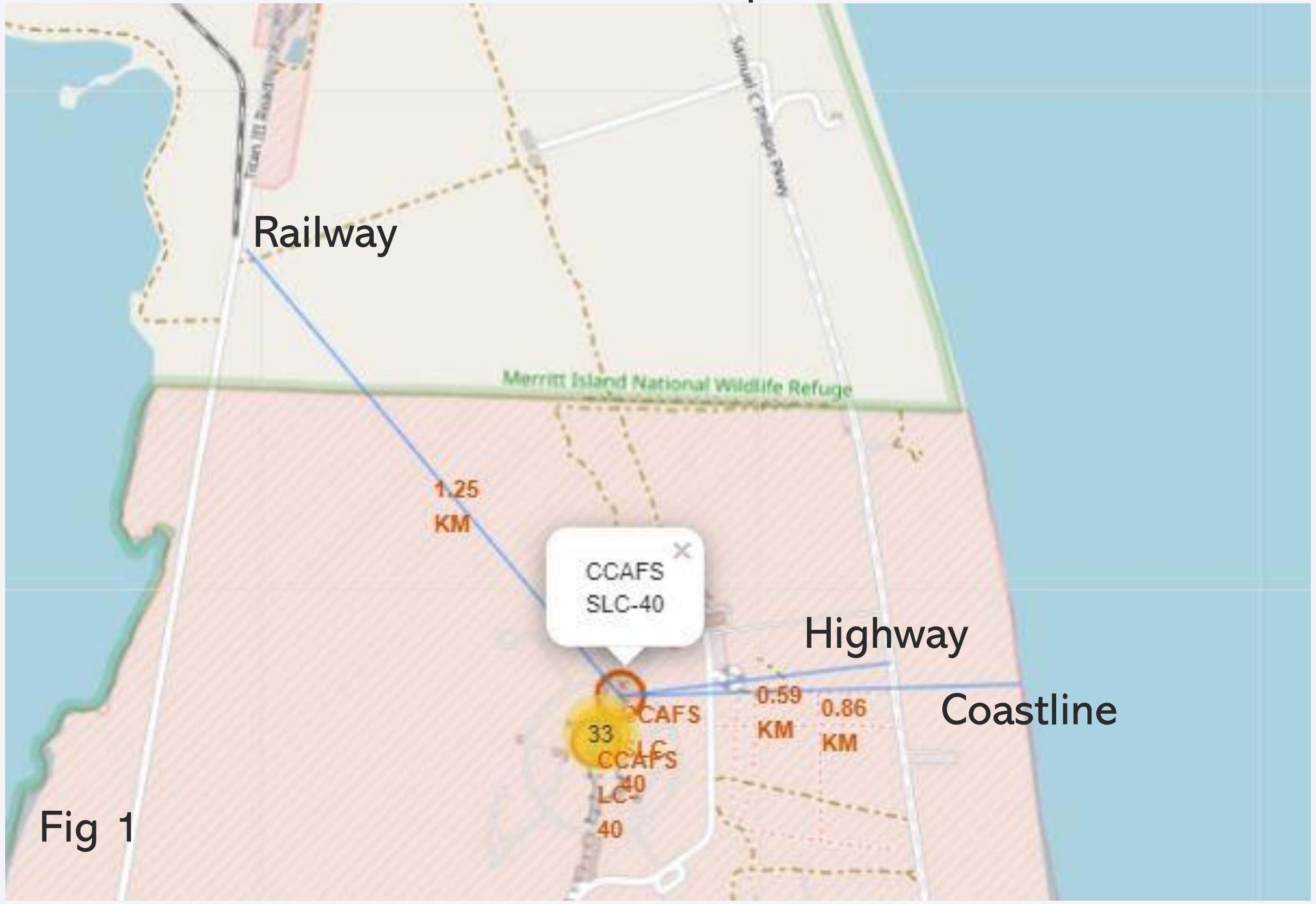


Fig 1

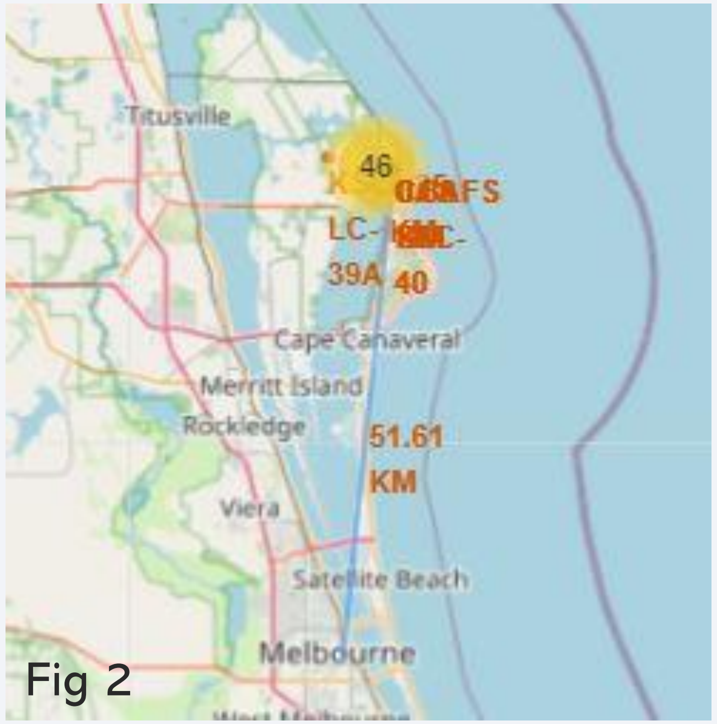
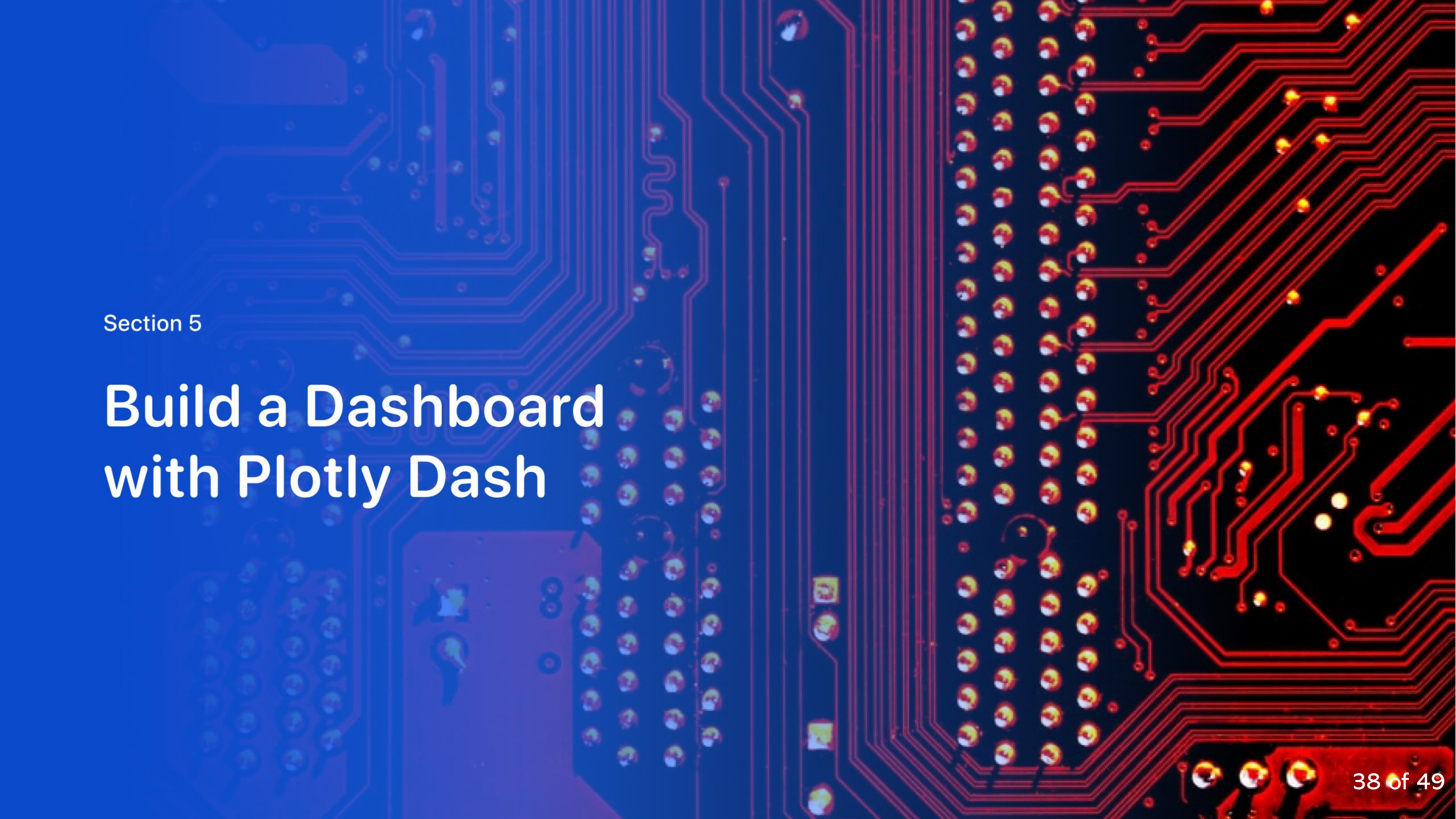


Fig 2

Fig 2 shows that Launch-site is certain distance away from nearest city (considered Melbourne here)

Fig 1 shows that Launch-site is in close proximity (~ 1km) with Coastline, Highway and Railway

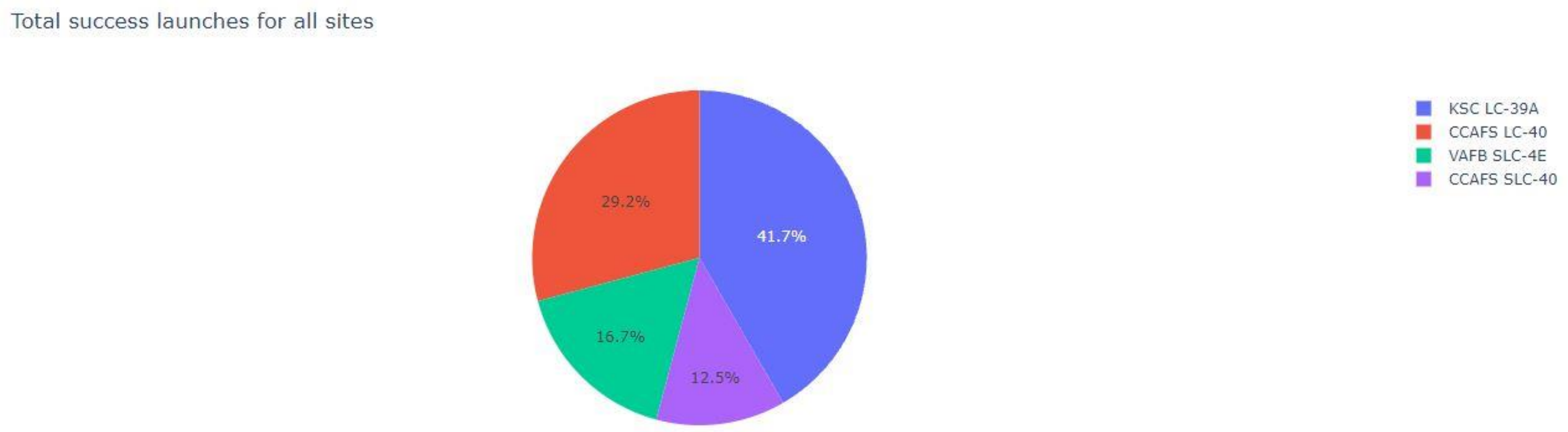
More data and analysis in appendix (slide #47)



Section 5

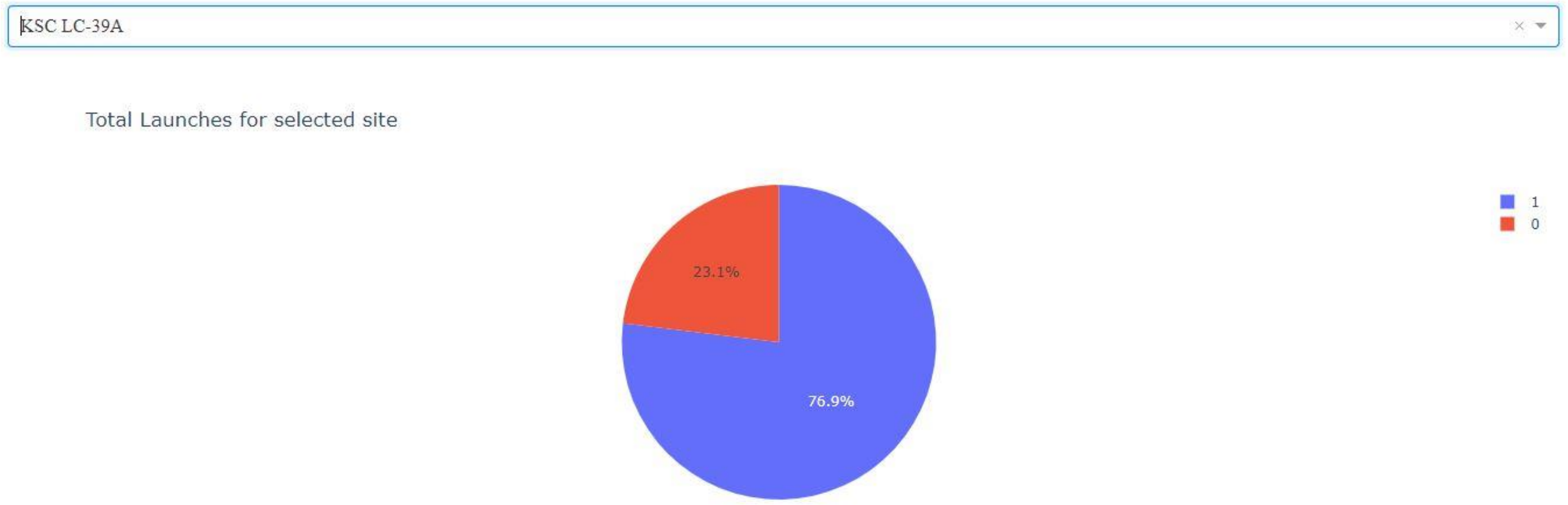
Build a Dashboard with Plotly Dash

Pie-chart for Successful Launches from all Launch Sites



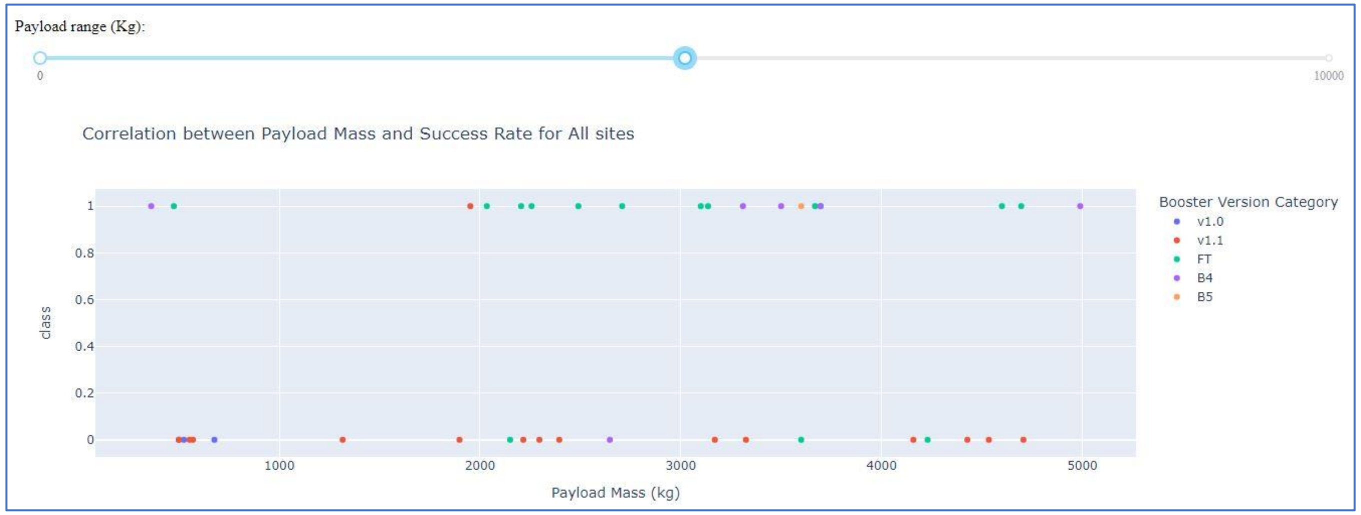
- KSC LC-39A has most number of successful launches
- CCAFS SLC-40 has least number of successful launches

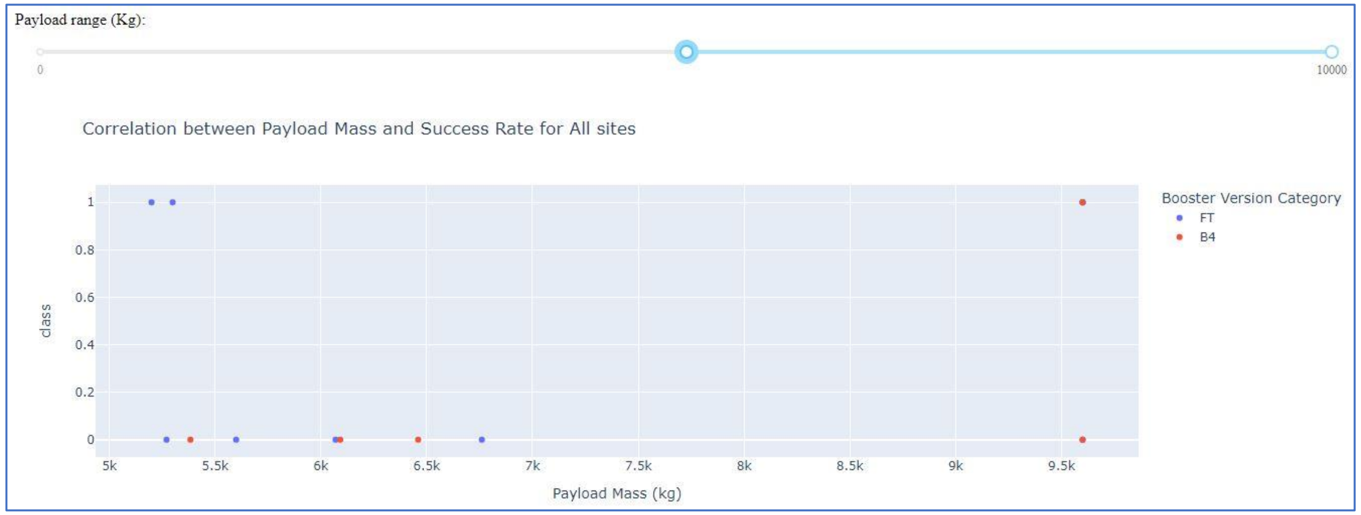
Pie chart of successes and failures for Launch Site with highest success ratio


KSC LC-39A

- On inspecting pie-chart for all individual sites, KSC LC-39A has highest ratio of success
- 76.9% launches from KSC-39A were successful, 23.1% launches were failed

Launch Outcome vs. Payload Mass for Launches from all sites





- For low payload, ratio of success and failure is similar, but at high payload, more failures can be seen
- All booster versions are used for low payload launches, however high payload launches are done only using 2 booster versions : B4 and FT
- Note : Low payload : < 5000 kg
  High payload : > 5000 kg

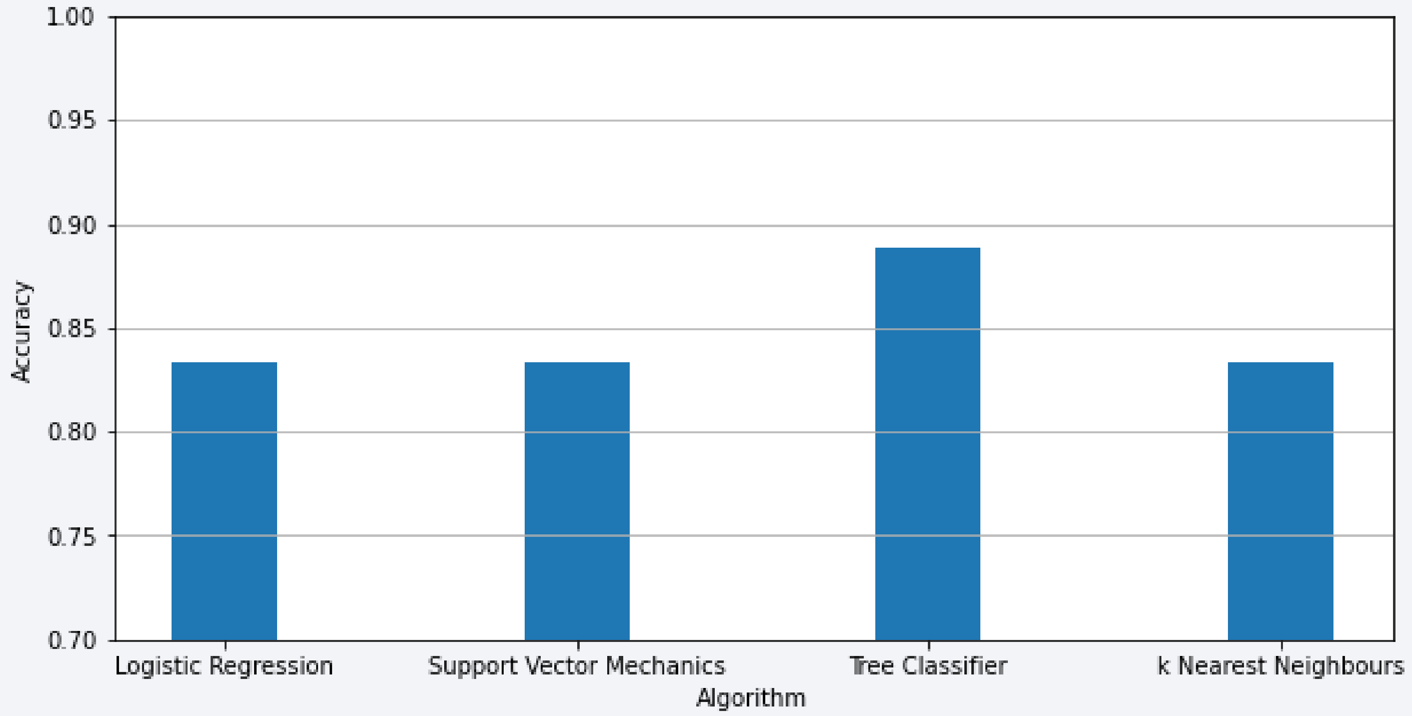
Please zoom to see figures in detail

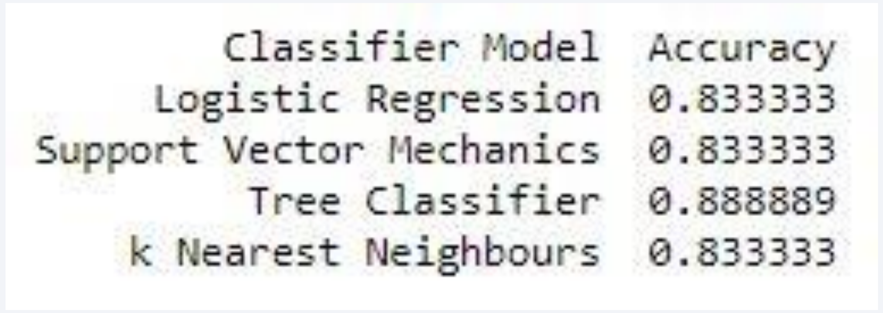
Section 6

Predictive Analysis (Classification)

Classification Accuracy

- To correctly evaluate model accuracy, we should look at accuracy for testing data
- Below is the bar-plot for model accuracies (on test data) for various algorithms



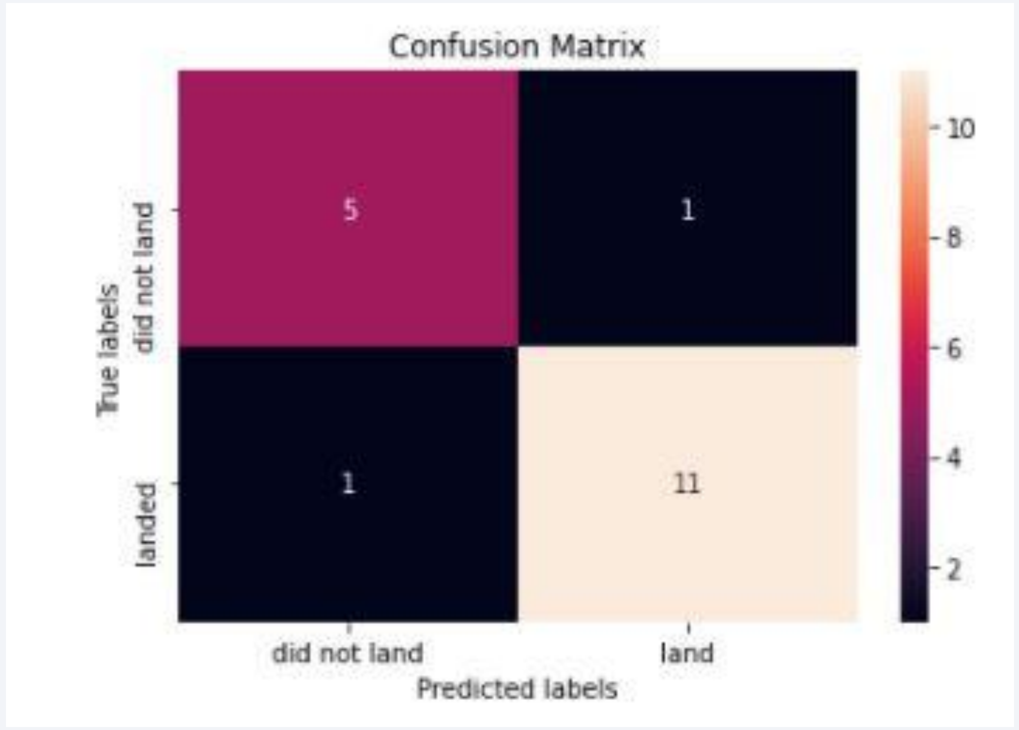


```
         Classifier Model  Accuracy
      Logistic Regression  0.833333
 Support Vector Mechanics  0.833333
          Tree Classifier  0.888889
      k Nearest Neighbours 0.833333
```

Best Parameter set for Tree Classifier is {'criterion': 'gini', 'max_depth': 8, 'max_features': 'auto', 'min_samples_leaf': 4, 'min_samples_split': 5, 'splitter': 'best'}

- For all 4 models, accuracy is considerably high (note that y axis starts from 0.7)
- Tree classifier has highest accuracy (0.88), all other algorithms have accuracy of 0.833
- More analysis in Appendix (slide #48)

Confusion Matrix



- Plotted on the left is confusion matrix for Tree Classifier which is best performing model
- It correctly predicts 16 out of 18 algorithm i.e. 11 outcomes predicted as 'land' which actually landed (True positive) and 5 outcomes predicted as 'did not land' which actually did not land (True negative)
- It incorrectly predicted one outcome as 'land' though it did not land. This is False positive. (top right)
- It incorrectly predicted one outcome as 'did not land' where it did actually land. This is False negative (bottom left)

Model Accuracy = (True positives + True negatives)/Total
Thus, accuracy for our model = (11+5)/18 = 16/18 = 0.889

Conclusions

- Tree Classifier model works best for predictive modeling. It can predict the outcome with sufficiently high accuracy (88 %). Other models have good accuracy as well (83 %). Thus, given the information of launch parameters, we can accurately predict landing success or failure
- Parameters having most impact on successful landing are Launch site, Payload mass, Launch orbit
- Launch site KSC LC-39A has both highest number and highest proportion for successful launches. All launch sites are near to coastline and away from cities, but are connected by highway and railway
- Launches will lesser payload have more proportion of success
- For few orbits, 100% success can be seen (e.g. GEO, SSO, ISS) while for orbit SO, success rate is 0%
- Success rate of launches has clearly improved over time from 0% in 2013 to >80% in 2019. With few efforts (like data modeling done in this project) success rate can be further improved

Appendix

- Appendix A – Launch Site Proximities
- Appendix B – Predictive Modeling Insights

Appendix A : Launch Site Proximities

Launch Site	Distance to nearest Landmark (in kms)				Nearest City Considered
	Coastline	Railway	Highway	City	
KSC LC-39A	6.55	0.70	0.85	52.47	Melbourne
CCAFS LC-40	0.93	1.30	0.65	51.50	Melbourne
CCAFS SLC-40	0.86	1.25	0.59	51.61	Melbourne
VAFB SLC-4E	1.34	1.26	5.57	39.05	Santa Maria

- All sites are close to coast-line. This might be due to safety reasons as failed rocket falling in ocean is much safer than falling on land
- All sites are significantly away from nearest city. This may have similar reasons as failed rocket away from densely populated area will be safer
- All sites are close to highway and railway. This might be due to easy movement of personnel, equipment, parts etc.
- Note : There is no fixed rule to get nearest city. One may choose different cities, however any city will generally be away from launch site

Back to slide #37

Appendix B : Predictive Modeling Insights

- There are a few parameters which were varied and their effects were found as an additional exercise. Summarizing the results for accuracies obtained below :

Algorithm	Default Condition		CV = 5		CV=15		Test Size 0.3		Test size = 1/6	
	Test	Train	Test	Train	Test	Train	Test	Train	Test	Train
Logistic Regression	0.833	0.846	0.833	0.834	0.833	0.846	0.815	0.859	0.866	0.826
Support Vector Mechanics	0.833	0.848	0.833	0.834	0.833	0.833	0.777	0.859	0.933	0.826
Tree Classifier	0.889	0.889	0.833	0.888	0.833	0.893	0.777	0.890	0.933	0.891
k Nearest Neighbours	0.833	0.848	0.833	0.861	0.777	0.846	0.815	0.874	0.866	0.841

- 1st parameter varied was cross validation (CV). All models when checked with a lower and higher value (5 and 15) did not give significantly different results, though a slight drop in testing accuracy was observed for higher value.
- This indicates that the default value is close to optimum
- 2nd parameter varied was Training and Testing split. All models were checked with a higher and lower proportion of testing data (0.3 and 0.166). Selected only slightly lower value as a very low value will result in loss of granularity (e.g. if only 10 datapoints are in test data, accuracy would be only in multiples of 10%, i.e. 80%, 90% etc.)
- For higher test data size, slight drop in accuracies is observed and for lower test data size, accuracies tend to increase

Back to slide #43

Thank you!