IBM Developer
SKILLS NETWORK

Winning Space Race with Data Science

Arthur Nowak
12th Sep 2022

Outline

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

Executive Summary: Introduction

In the following slides the price of a rocket launch will be estimated.

To do this account data from similar launches will be analyzed.

Drivers for launch success will be identified and factored into the prediction models to produce the best estimate.

Executive Summary; Methodologies

To do this these techniques will be used:

- web scraping
- data wrangling
- imputations
- SQL-queries
- exploratory data analysis
- visualization
- machine learning prediction

Executive Summary: Results I

Launches in general get more successful with time (flight number).

VAFB-SLC launch site doesn't launch heavy payloads.

Launch success to some orbits(LEO) improve with time while success rates to other orbits (GTO, ISS) are not strongly correlated with time.

Launches with heavy payloads have a higher success rate in orbits Polar, LEO, ISS and don't show a pattern for other orbits.

Executive Summary: Results II

Proximity to a city, railway, highway, coastline is important for a launch site. Also beeing close to the equator helps.

"CCAFS LC-40" *(Cape Canaveral Space Launch Complex 40)* has most launches.

The most successful launch site is "KSC LC-39A" *(Kennedy Space Center Launch Complex 39A)* with a 76,9% success rate.

"VAFB SLC-4E" *(Vandenberg Air Force Base Space Launch Complex 4E)* has good success for booster version with heavy payload.

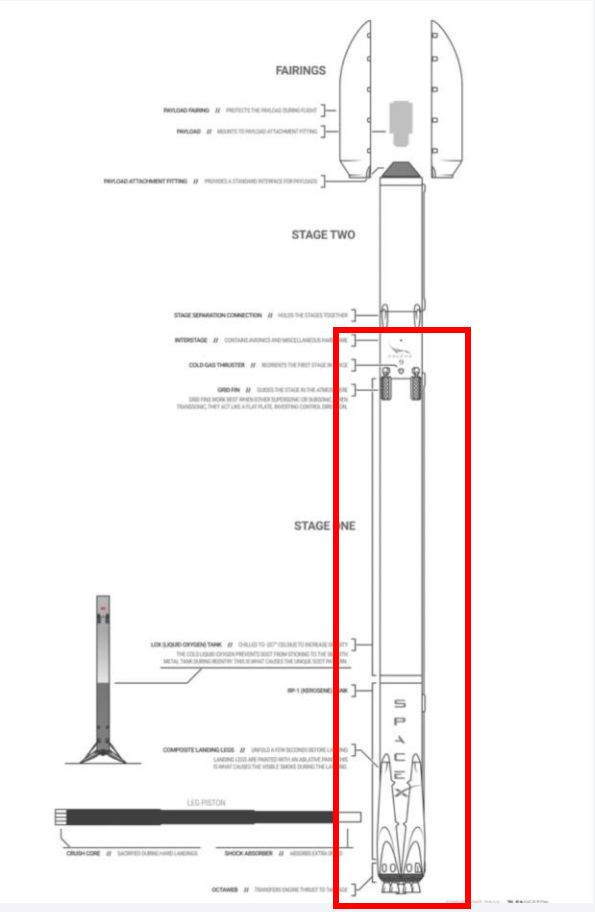
Booster version category "B4" is best for heavy payloads.

Booster version category "FT" has a solid success rate for medium to light payloads.

The launch success can be predicted with an accuracy of 83% with a decision tree classification.

Introduction

- As a data scientist the goal is to help a rocket company "Space Y" estimate the price of each launch by analyzing Space X results.
- Also it will be determine if the first stage will be reused.



Section 1

Methodology

Methodology

Executive Summary

- Data collection methodology I:
  - The requests-library was used to obtain launch data from the spacexdata.api.
  - The data was enriched by requesting additional data, and putting it into the correct format.
  - With the pandas library the data was subset to extract the most relevant columns.
  - The data was filtered for Falcon 9 records.
  - Incomplete entries were populated by the column mean to not distort the results.

Methodology

Executive Summary

- Data collection methodology II (web scraping):
  - Also the html-code from the wikipedia-website was loaded.
  - With the BeautifulSoup-library and helper functions the table data was extracted
  - And parsed into a pandas-dataframe.
  - Perform data wrangling:
  - Missing values were identified.
  - Samples sizes were counted.
  - Outcome data was analyzed and translated into a better processable format.
  - Categorical variables were expanded into columns with dummy variables for each category.

Methodology

Executive Summary

- Perform exploratory data analysis (EDA) using SQL:
  - Spacex-dataset was loaded into a database table.
  - Using sqlalchemy it was queried to identify
    - unique values
    - Subsets
    - Summaries
    - Averages
    - corner cases
    - rankings

Methodology

Executive Summary

- Perform exploratory data analysis (EDA) using visualization:
  - Matplotlib and Seaborn-libraries were used to create scatter plots. Three dimensions of information were plotted to identify patterns in for launch success outcomes.
  - For this the effects of these columns were plotted against each other:
    - payload mass
    - flight number
    - time
    - different launch sites
    - orbit types

Methodology

Executive Summary

- Perform interactive visual analytics using Folium and Plotly Dash:
  - Dash-library and plotly.express was used to plot interactive pie-charts and scatterplots.
  - The controls were a dropdown and a slider.
  - Different combinations of launch sites and payload-mass ranges were examined.
  - It was identified which launch sites and booster versions were most successful for which payload-weights.

Methodology

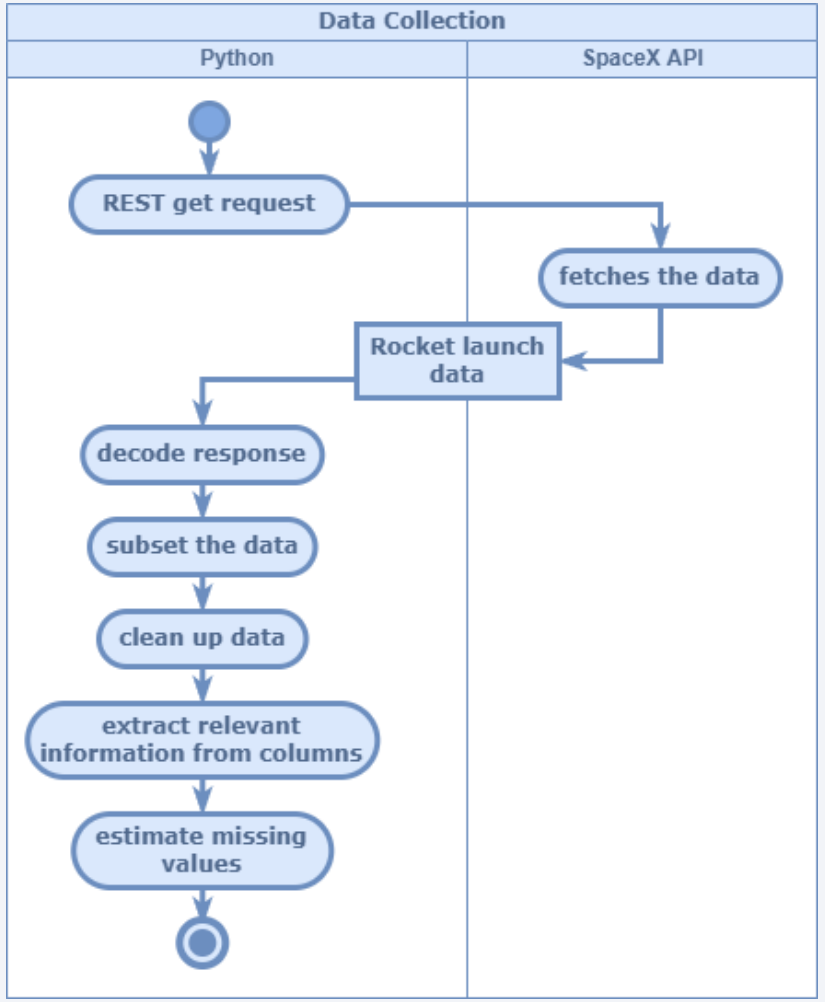
Executive Summary

- Perform predictive analysis using classification models:
  - Sklearn was used to perform GridSearch training for these classification Models:
    - Logistic regression
    - Support Vector Machines
    - Decision Tree
    - K Nearest Neighbors
  - The models were evaluated by calculating accuracy-scores and examining confusion matrices.

Data Collection – SpaceX API

- The GitHub URL of the completed SpaceX API calls notebook:

  https://github.com/ArthurNow/DataScienceMaLeCapstone/blob/eb9bf73b195e38c69b790dbf351d9dc0007efa77/jupyter-labs-spacex-data-collection-api%20(3).ipynb

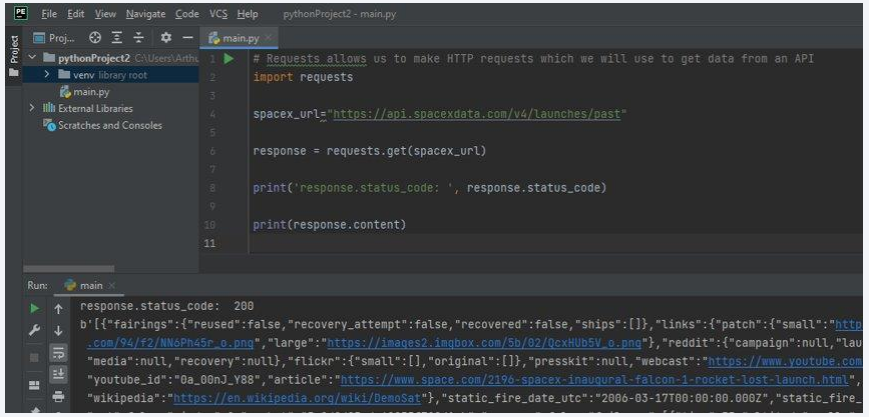


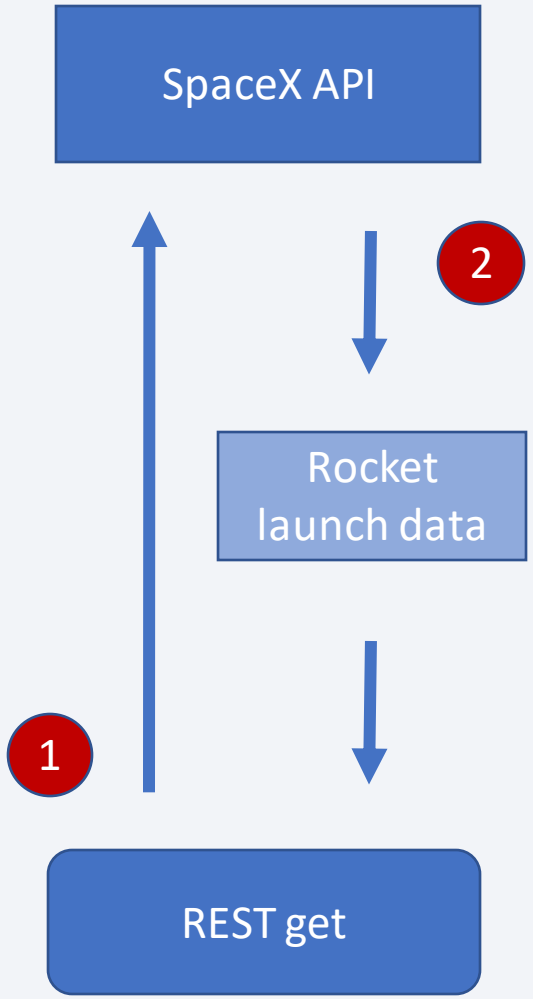
Data Collection

- A REST-get was made by the python requests library.

  https://de.wikipedia.org/wiki/Representational_State_Transfer

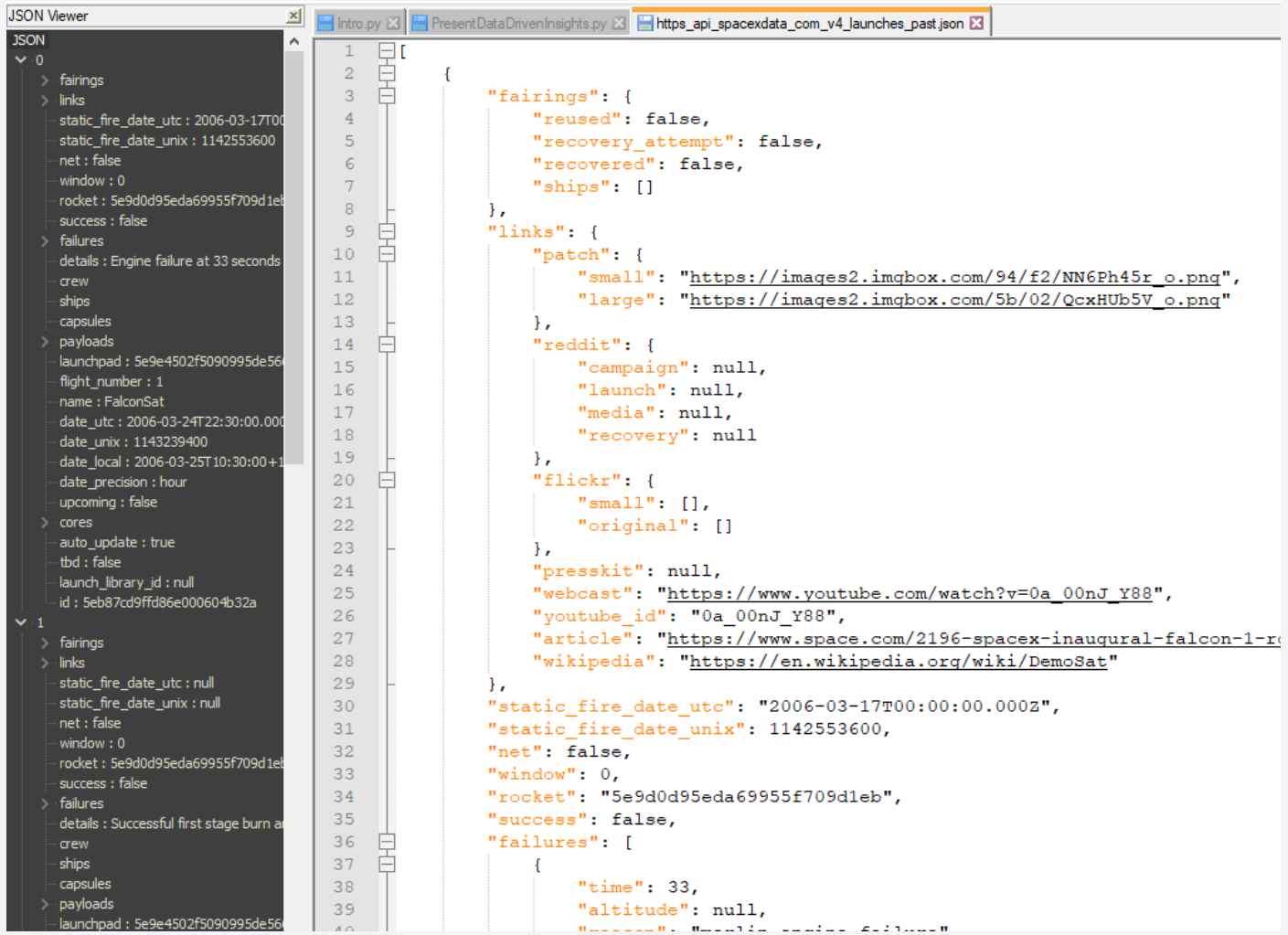
  The result was a json, which was parsed into a pandas dataframe.





Data Collection - content of the response (json)

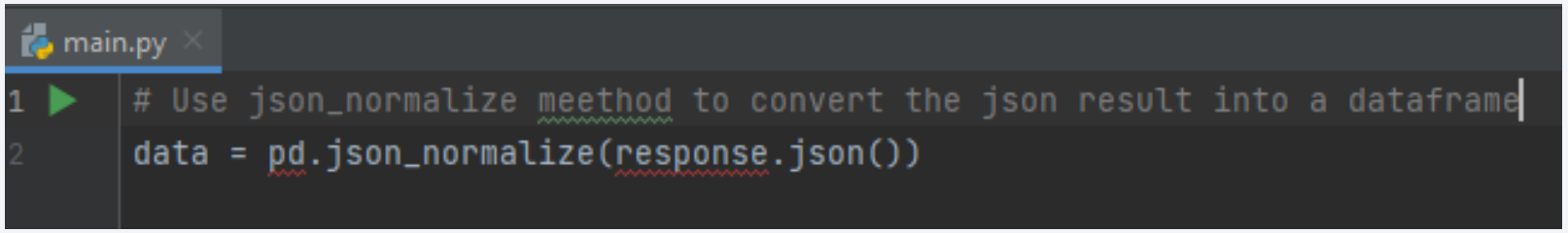
Rocket launch data.json



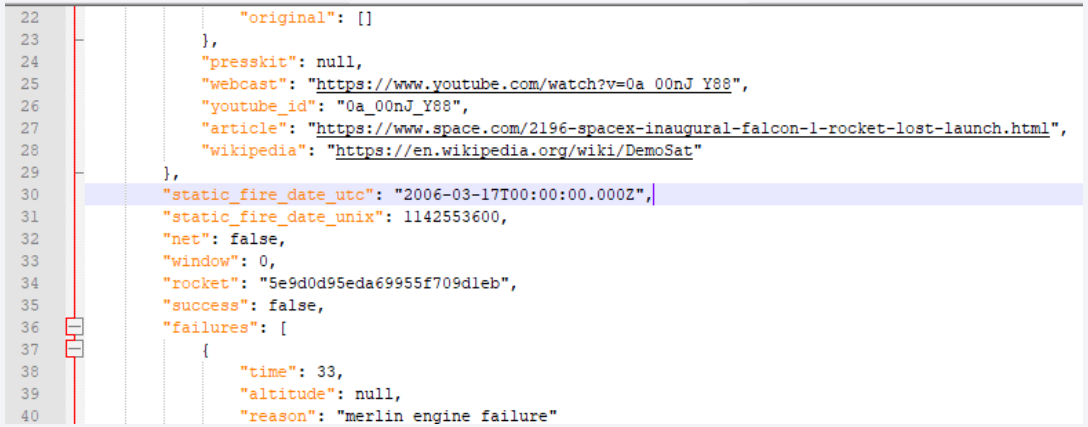
Data Wrangling I: - decoding the response

The data will be transformed from the raw-format into one that is easier processable:

1. Normalize semi-structured JSON data into a flat table.



Data Wrangling I: - decoding the response



```
            "original": []
        },
        "presskit": null,
        "webcast": "https://www.youtube.com/watch?v=0a_00nJ_Y88",
        "youtube_id": "0a_00nJ_Y88",
        "article": "https://www.space.com/2196-spacex-inaugural-falcon-1-rocket-lost-launch.html",
        "wikipedia": "https://en.wikipedia.org/wiki/DemoSat"
    },
    "static_fire_date_utc": "2006-03-17T00:00:00.000Z",
    "static_fire_date_unix": 1142553600,
    "net": false,
    "window": 0,
    "rocket": "5e9d0d95eda69955f709d1eb",
    "success": false,
    "failures": [
        {
            "time": 33,
            "altitude": null,
            "reason": "merlin engine failure"
```

Multidimensional JSON

Two-dimensional, normalized dataframe

static_fire_date_utc	static_fire_date_unix	net	window	rocket	success	failures	details
0	2006-03-17T00:00:00.000Z	1.142554e+09	False	0.0	5e9d0d95eda69955f709d1eb	False	[{'time': 33, 'altitude': Nc
1	None	NaN	False	0.0	5e9d0d95eda69955f709d1eb	False	[{'time': 301, 'altitude': 2
2	None	NaN	False	0.0	5e9d0d95eda69955f709d1eb	False	[{'time': 140, 'altitude': 3
3	2008-09-20T00:00:00.000Z	1.221869e+09	False	0.0	5e9d0d95eda69955f709d1eb	True	[]
4	None	NaN	False	0.0	5e9d0d95eda69955f709d1eb	True	[]

Data Wrangling I: - subset only relevant data

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

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

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

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

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

Data Wrangling I: - enrich and transform records

```
#Global variables
BoosterVersion = []
PayloadMass = []
Orbit = []
LaunchSite = []
Outcome = []
Flights = []
GridFins = []
Reused = []
Legs = []
LandingPad = []
Block = []
ReusedCount = []
Serial = []
Longitude = []
Latitude = []

# These functions will apply the outputs globally to the above
variables. Let's take a looks at BoosterVersion variable. Before
we apply getBoosterVersion the list is empty:

print('\nBoosterVersion',BoosterVersion)
# Now, let's apply  getBoosterVersion function method to get the
booster version


# Call getBoosterVersion
getBoosterVersion(data)
# the list has now been update

print('\n')
print(BoosterVersion[0:5])
# we can apply the rest of the functions here:


# Call getLaunchSite
getLaunchSite(data)

# Call getPayloadData
getPayloadData(data)

# Call getCoreData
getCoreData(data)
```

```
# Takes the dataset and uses the rocket column to call the API
and append the data to the list
def getBoosterVersion(data):
    for x in data['rocket']:
        response = requests.get(
        "https://api.spacexdata.com/v4/rockets/"+str(x)).json()
        BoosterVersion.append(response['name'])
```

```
# Takes the dataset and uses the launchpad column to call the API
and append the data to the list
def getLaunchSite(data):
    for x in data['launchpad']:
        response = requests.get(
        "https://api.spacexdata.com/v4/launchpads/"+str(x)).json()
        Longitude.append(response['longitude'])
        Latitude.append(response['latitude'])
        LaunchSite.append(response['name'])
```

```
# Takes the dataset and uses the payloads column to call the API
and append the data to the lists
def getPayloadData(data):
    for load in data['payloads']:
        response = requests.get(
        "https://api.spacexdata.com/v4/payloads/"+load).json()
        PayloadMass.append(response['mass_kg'])
        Orbit.append(response['orbit'])
```

```
# Takes the dataset and uses the cores column to call the API and append the data to the lists
def getCoreData(data):
    for core in data['cores']:
            if core['core'] != None:
                response = requests.get("https://api.spacexdata.com/v4/cores/"+core['core']).json()
                Block.append(response['block'])
                ReusedCount.append(response['reuse_count'])
                Serial.append(response['serial'])
            else:
                Block.append(None)
                ReusedCount.append(None)
                Serial.append(None)
            Outcome.append(str(core['landing_success'])+' '+str(core['landing_type']))
            Flights.append(core['flight'])
            GridFins.append(core['gridfins'])
            Reused.append(core['reused'])
            Legs.append(core['legs'])
            LandingPad.append(core['landpad'])
```

Data Wrangling I: SpaceX - remove unwanted data

```
# Get names of indexes for which column Age has value 30
indexNames_falcon1 = df_launch[df_launch['BoosterVersion']!='Falcon 1'].index
# Delete these row indexes from dataFrame
data_falcon9 = df_launch.drop(indexNames_falcon1)


# Now that we have removed some values we should reset the FlgihtNumber column
data_falcon9.loc[:,'FlightNumber'] = list(range(1, data_falcon9.shape[0]+1))
print('\n data_falcon9 ',data_falcon9)
```

Only Falcon 9 data gets analyzed.
All Falcon 1 rows get removed.

Data Wrangling I: SpaceX - extrapolate missing data

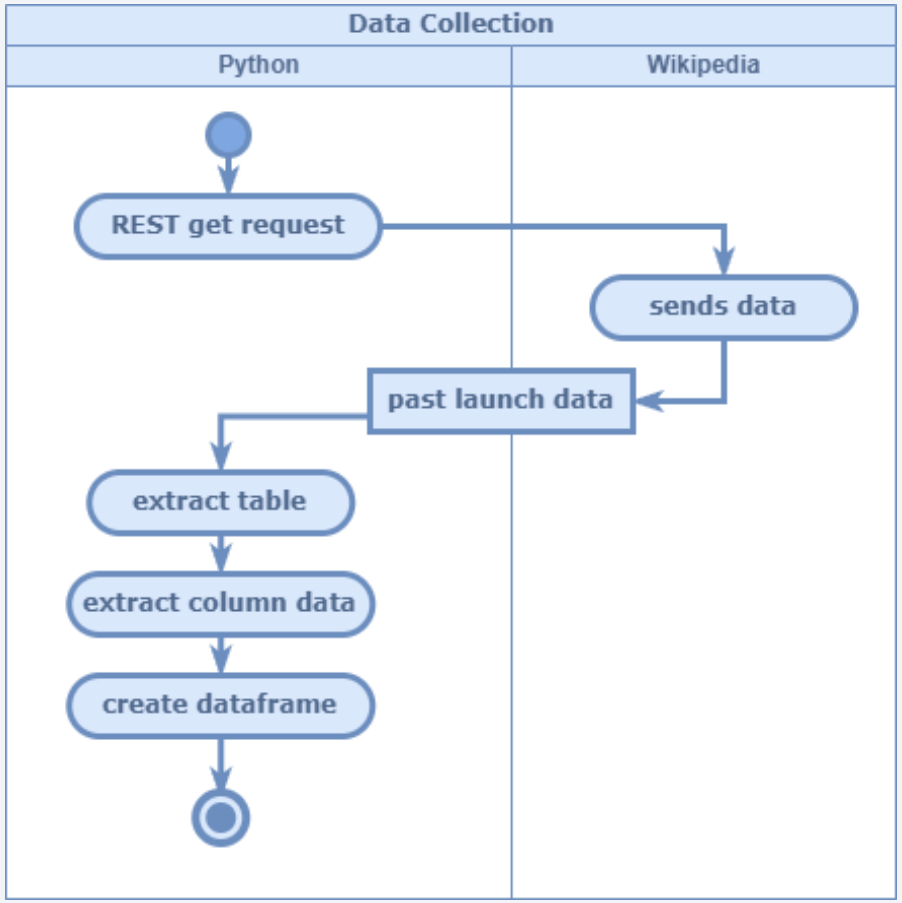
```
# Calculate the mean value of PayloadMass column
data_falcon9_PayloadMass_mean = data_falcon9['PayloadMass'].mean()

# Replace the np.nan values with its mean value
data_falcon9['PayloadMass'].replace(np.nan,data_falcon9_PayloadMass_mean, inplace=True)
```

Missing values get populated by the column-mean to not distort any future results

Data Collection – Web Scraping

- 1. Obtain Falcon 9 and Falcon Heavy Launches Records from Wikipedia
- The Data Collection with Web Scraping Notebook:

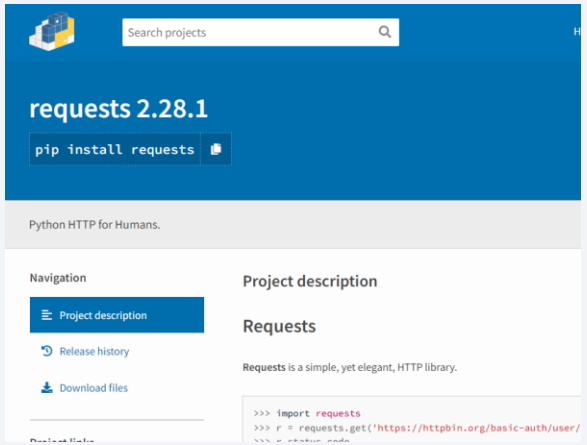
  https://github.com/ArthurNow/DataScienceMaLeCapstone/blob/main/jupyter-labs-webscraping%20(2).ipynb



Data Collection – Load the Falcon9 Launch Wiki page from its URL

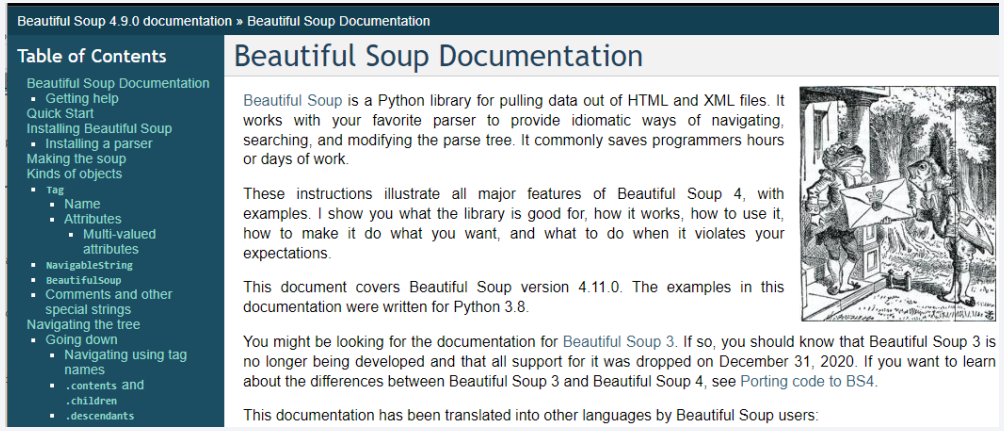
- 2. Use the Requests and BeautifulSoup-library:

https://pypi.org/project/requests/



Requests allows you to send HTTP/1.1 requests

https://www.crummy.com/software/BeautifulSoup/bs4/doc/



… for pulling data out of HTML and XML files

Data Collection

- 3. Load the Falcon9 Launch Wiki page from its URL

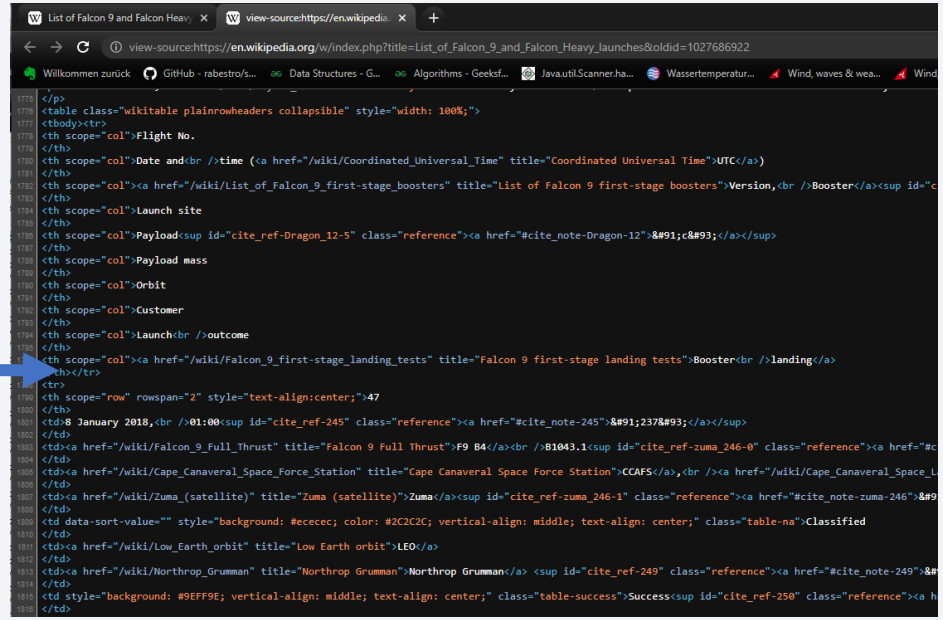
```
"""
TASK 1: Request the Falcon9 Launch Wiki page from its URL
First, let's perform an HTTP GET method to request the Falcon9 Launch HTML page, as an HTTP response.
"""

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

# Create a BeautifulSoup object from the HTML response
# Use BeautifulSoup() to create a BeautifulSoup object from a response text content

# soup = BeautifulSoup(html_data, 'html5lib')
# https://www.crummy.com/software/BeautifulSoup/bs4/doc/
soup = BeautifulSoup(html_data, 'html.parser')

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

Data Collection – extract table-data from html-page

Past launches

2010 to 2013

[hide] Flight No.	Date and time (UTC)	Version, Booster [b]	Launch site	Payload[c]	Payload mass	Orbit	Customer	Launch outcome	Booster landing
1	4 June 2010, 18:45	F9 v1.0[7] B0003.1[8]	CCAFS, SLC-40	Dragon Spacecraft Qualification Unit		LEO	SpaceX	Success	Failure[9][10] (parachute)
	First flight of Falcon 9 v1.0.[11] Used a boilerplate version of Dragon capsule which was not designed to separate from the second stage. (more details below) Attempted to recover the first stage by parachuting it into the ocean, but it burned up on reentry, before the parachutes even deployed.[12]								
2	8 December 2010, 15:43[13]	F9 v1.0[7] B0004.1[8]	CCAFS, SLC-40	Dragon demo flight C1 (Dragon C101)		LEO (ISS)	NASA (COTS) NRO	Success[9]	Failure[9][14] (parachute)
	Maiden flight of Dragon capsule, consisting of over 3 hours of testing thruster maneuvering and reentry.[15] Attempted to recover the first stage by parachuting it into the ocean, but it disintegrated upon reentry, before the parachutes were deployed.[12] (more details below) It also included two CubeSats,[16] and a wheel of Brouère cheese.								
3	22 May 2012, 07:44[17]	F9 v1.0[7] B0005.1[8]	CCAFS, SLC-40	Dragon demo flight C2+[18] (Dragon C102)	525 kg (1,157 lb)[19]	LEO (ISS)	NASA (COTS)	Success[20]	No attempt
	Dragon spacecraft demonstrated a series of tests before it was allowed to approach the International Space Station. Two days later, it became the first commercial spacecraft to board the ISS.[17] (more details below)								
4	8 October 2012, 00:35[21]	F9 v1.0[7] B0006.1[8]	CCAFS, SLC-40	SpaceX CRS-1[22] (Dragon C103)	4,700 kg (10,400 lb)	LEO (ISS)	NASA (CRS)	Success	No attempt
				Orbcomm-OG2[23]	172 kg (379 lb)[24]	LEO	Orbcomm	Partial failure[25]	
	CRS-1 was successful, but the secondary payload was inserted into an abnormally low orbit and subsequently lost. This was due to one of the nine Merlin engines shutting down during the launch, and NASA declining a second reignition, as per ISS visiting vehicle safety rules, the primary payload owner is contractually allowed to decline a second reignition. NASA stated that this was because SpaceX could not guarantee a high enough likelihood of the second stage completing the second burn successfully which was required to avoid any risk of secondary payload's collision with the ISS.[26][27][28]								
5	1 March 2013, 15:10	F9 v1.0[7] B0007.1[8]	CCAFS, SLC-40	SpaceX CRS-2[22] (Dragon C104)	4,877 kg (10,752 lb)	LEO (ISS)	NASA (CRS)	Success	No attempt
	Last launch of the original Falcon 9 v1.0 launch vehicle, first use of the unpressurized trunk section of Dragon.[29]								
6	29 September 2013, 16:00[30]	F9 v1.1[7] B1003[8]	VAFB, SLC-4E	CASSIOPE[22][31]	500 kg (1,100 lb)	Polar orbit LEO	MDA	Success[30]	Uncontrolled (ocean)[d]
	First commercial mission with a private customer, first launch from Vandenberg, and demonstration flight of Falcon 9 v1.1 with an improved 13-tonne to LEO capacity.[29] After separation from the second stage carrying Canadian commercial and scientific satellites, the first stage booster performed a controlled reentry,[32] and an ocean touchdown test for the first time. This provided good test data, even though the booster started rolling as it neared the ocean, leading to the shutdown of the central engine as the roll depleted it of fuel, resulting in a hard impact with the ocean.[30] This was the first known attempt of a rocket engine being lit to perform a supersonic retro propulsion, and allowed SpaceX to enter a public-private partnership with NASA and its Mars entry, descent, and								



Data Wrangling II: – extract table-data ...

```
# Next, we just need to iterate through the <th> elements and apply the provided extract_column_from_header() to extract
column name one by one
column_names = []

# Apply find_all() function with `th` element on first_launch_table
# Iterate each th element and apply the provided extract_column_from_header() to get a column name
# Append the Non-empty column name (`if name is not None and len(name) > 0`) into a list called column_names
for header in first_launch_table.findAll("th"):
    # print(header)
    name = extract_column_from_header(header)
    # print(name)
    if name is not None and len(name) > 0:
        column_names.append(name)
# 493  Version,<br />booster  is not extracted


# Check the extracted column names
print(column_names)
```

```
launch_dict= dict.fromkeys(column_names)

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

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

```
extracted_row = 0
#Extract each table
for table_number,table in enumerate(soup.find_all('table',"wikitable plainrowheaders collapsible")):
   # get table row
    for rows in table.find_all("tr"):
        #check to see if first table heading is as number corresponding to launch a number
        if rows.th:
            if rows.th.string:
                flight_number=rows.th.string.strip()
                flag=flight_number.isdigit()
        else:
            flag=False
        #get table element
        row=rows.find_all('td')
        #if it is number save cells in a dictonary
        if flag:
            extracted_row += 1
            # Flight Number value
            # TODO: Append the flight_number into launch_dict with key `Flight No.`
            launch_dict['Flight No.'].append(flight_number)
            print('Flight No. ', flight_number)
            datatimelist=date_time(row[0])

            # Date value
            # TODO: Append the date into launch_dict with key `Date`
            date = datatimelist[0].strip(',')
            launch_dict['Date'].append(date)
            print('Date ',date)

            # Time value
            # TODO: Append the time into launch_dict with key `Time`
            time = datatimelist[1]
```

Data Wrangling II: – … and store it into a dataframe

```
[16]: df=pd.DataFrame(launch_dict)
      df
```

[16]:

	Flight No.	Launch site	Payload	Payload mass	Orbit	Customer	Launch outcome	Version Booster	Booster landing	Date	Time
0	1	CCAFS	Dragon Spacecraft Qualification Unit	0	LEO	SpaceX	Success\n	F9 v1.0B0003.1	Failure	4 June 2010	18:45
1	2	CCAFS	Dragon	0	LEO	NASA	Success	F9 v1.0B0004.1	Failure	8 December 2010	15:43
2	3	CCAFS	Dragon	525 kg	LEO	NASA	Success	F9 v1.0B0005.1	No attempt\n	22 May 2012	07:44
3	4	CCAFS	SpaceX CRS-1	4,700 kg	LEO	NASA	Success\n	F9 v1.0B0006.1	No attempt	8 October 2012	00:35
4	5	CCAFS	SpaceX CRS-2	4,877 kg	LEO	NASA	Success\n	F9 v1.0B0007.1	No attempt\n	1 March 2013	15:10
...	...	...	...	...	...	...	...	...	...	...	...
116	117	CCSFS	Starlink	15,600 kg	LEO	SpaceX	Success\n	F9 B5B1051.10	Success	9 May 2021	06:42
117	118	KSC	Starlink	~14,000 kg	LEO	SpaceX	Success\n	F9 B5B1058.8	Success	15 May 2021	22:56
118	119	CCSFS	Starlink	15,600 kg	LEO	SpaceX	Success\n	F9 B5B1063.2	Success	26 May 2021	18:59
119	120	KSC	SpaceX CRS-22	3,328 kg	LEO	NASA	Success\n	F9 B5B1067.1	Success	3 June 2021	17:29
120	121	CCSFS	SXM-8	7,000 kg	GTO	Sirius XM	Success\n	F9 B5	Success	6 June 2021	04:26

Data Wrangling III

- The GitHub URL of the completed SpaceX API calls notebook:

  https://github.com/ArthurNow/DataScienceMaLeCapstone/blob/ea2bc377a3b16e2855f222c9fecccbbdd62f37f2/labs-jupyter-spacex-Data%20wrangling%20(1).ipynb

Data Wrangling: get overview of sample sizes

Unique records for each ...

Launch site

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

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

Orbit

```
# Apply value_counts on Orbit column
df.columns
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
```

Landing outcomes

```
# landing_outcomes = values on Outcome column
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
```

Data Wrangling: identify and encode unsuccessful landings

Create a set with identifiers for unsuccessful outcomes

```
for i,outcome in enumerate(landing_outcomes.keys()):
    print(i,outcome)
```

```
0 True ASDS
1 None None
2 True RTLS
3 False ASDS
4 True Ocean
5 False Ocean
6 None ASDS
7 False RTLS
```

We create a 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'}
```

encode bad landing outcomes into a dummy value "0"

```
# landing_class = 0 if bad_outcome
# landing_class = 1 otherwise

landingOutcomeLabel = []

for index, row in df.iterrows():
    outcome = row['Outcome']
    # print(outcome)
    #print(row)
    if outcome in bad_outcomes:
        landingOutcomeLabel.append(0)
    else:
        landingOutcomeLabel.append(1)

landing_class = landingOutcomeLabel
```

Data Wrangling: analyze landing outcomes

New column with the landing outcome dummy integer

```
df.head(5)
```

ıber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins	Reused	Legs	LandingPad	Block	ReusedCount	Serial	Longitude	Latitude	Class
1	2010-06-04	Falcon 9	6104.959412	LEO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B0003	-80.577366	28.561857	0
2	2012-05-22	Falcon 9	525.000000	LEO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B0005	-80.577366	28.561857	0
3	2013-03-01	Falcon 9	677.000000	ISS	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B0007	-80.577366	28.561857	0
4	2013-09-29	Falcon 9	500.000000	PO	VAFB SLC 4E	False Ocean	1	False	False	False	NaN	1.0	0	B1003	-120.610829	34.632093	0
5	2013-12-03	Falcon 9	3170.000000	GTO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B1004	-80.577366	28.561857	0

Calculate the overall success rate

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

```
0.6666666666666666
```

EDA with Data Visualization

- For an exploratory "ocular" analysis the launch success (class = 1) was plotted against various success-drivers like
    - payload-mass
    - Time
    - launch site
    - intended orbit.

= > If patterns beyond a uniform distribution of yellow and blue dots appears, then a relationship between the driver and the success of the launch might exist.

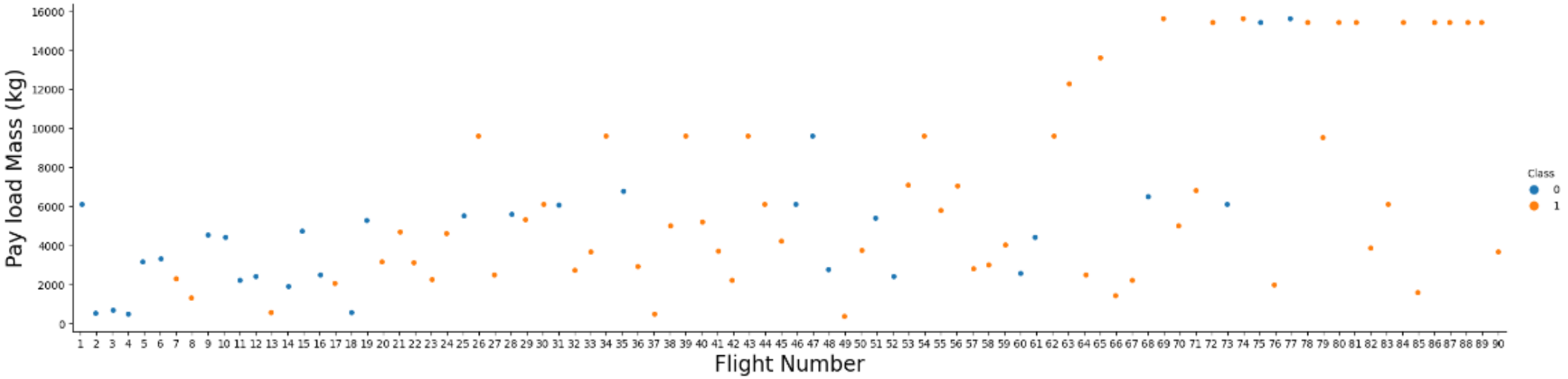
EDA with Data Visualization

- The GitHub URL of the completed EDA with data visualization notebook: https://github.com/ArthurNow/DataScienceMaLeCapstone/blob/2b0199ba6e11ccb01a861c3d76ed88ae53e78925/jupyter-labs-eda-dataviz%20(5).ipynb

EDA with Data Visualization: Flight Number - Payload

How does flight number and payload-weight affect the success of the launch?

```
[6]: sns.catplot(y="PayloadMass", x="FlightNumber", hue="Class", data=df, aspect = 4)
     plt.xlabel("Flight Number",fontsize=20)
     plt.ylabel("Pay load Mass (kg)",fontsize=20)
     plt.show()
```



EDA with SQL; performed queries

- The GitHub URL of the completed EDA with SQL notebook: https://github.com/ArthurNow/DataScienceMaLeCapstone/blob/395d0ad60e32efb5d15c6661d2e4293653289f9d/jupyter-labs-eda-sql-edx%20(7).ipynb

Build an Interactive Map with Folium

- Markers for each launch site were created on an interactive map.
- Pop-Ups with the title as well as a collapsable visualization of the success rates were added to the markers.
- Lines as well as distance measurements to the nearest coast, city, highway, railway were added.

This way there we have orientation on the map and a visual representation of important success drivers as well as success outcomes for each launch site.

Build an Interactive Map with Folium

- the GitHub URL of the completed interactive map with Folium map: https://github.com/ArthurNow/DataScienceMaLeCapstone/blob/f42195607144f486de7596df2ade0e308ae2abae/lab_jupyter_launch_site_location%20(5).ipynb

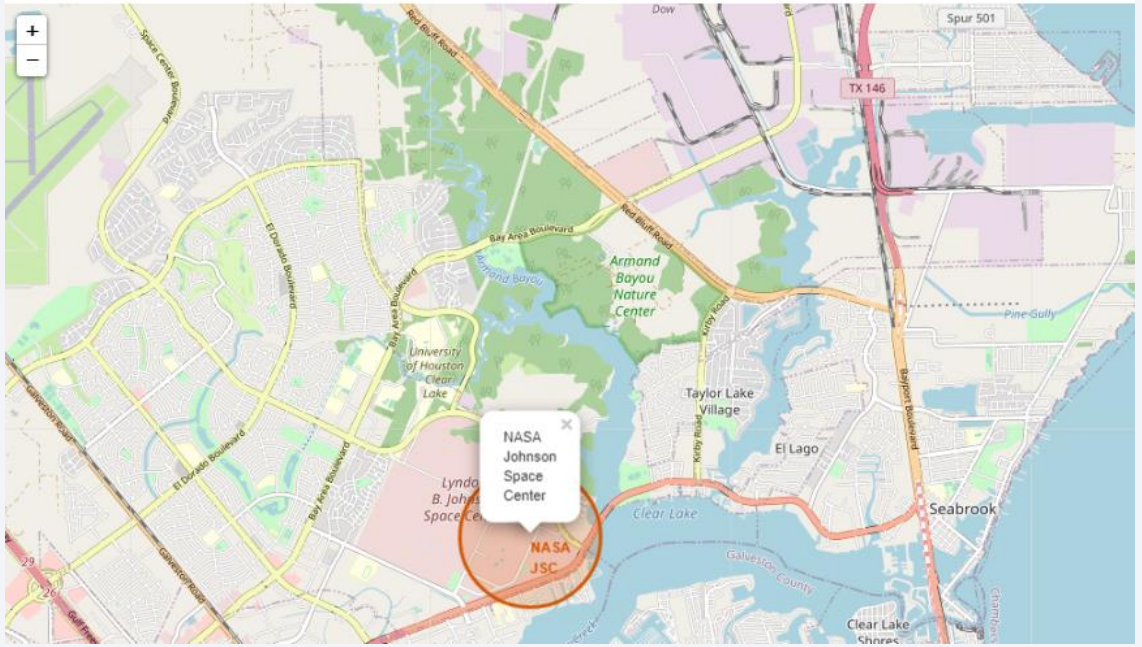
Interactive Map with Folium with Popup

Now, you can take a look at what are the coordinates for each site.

```
# 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
```

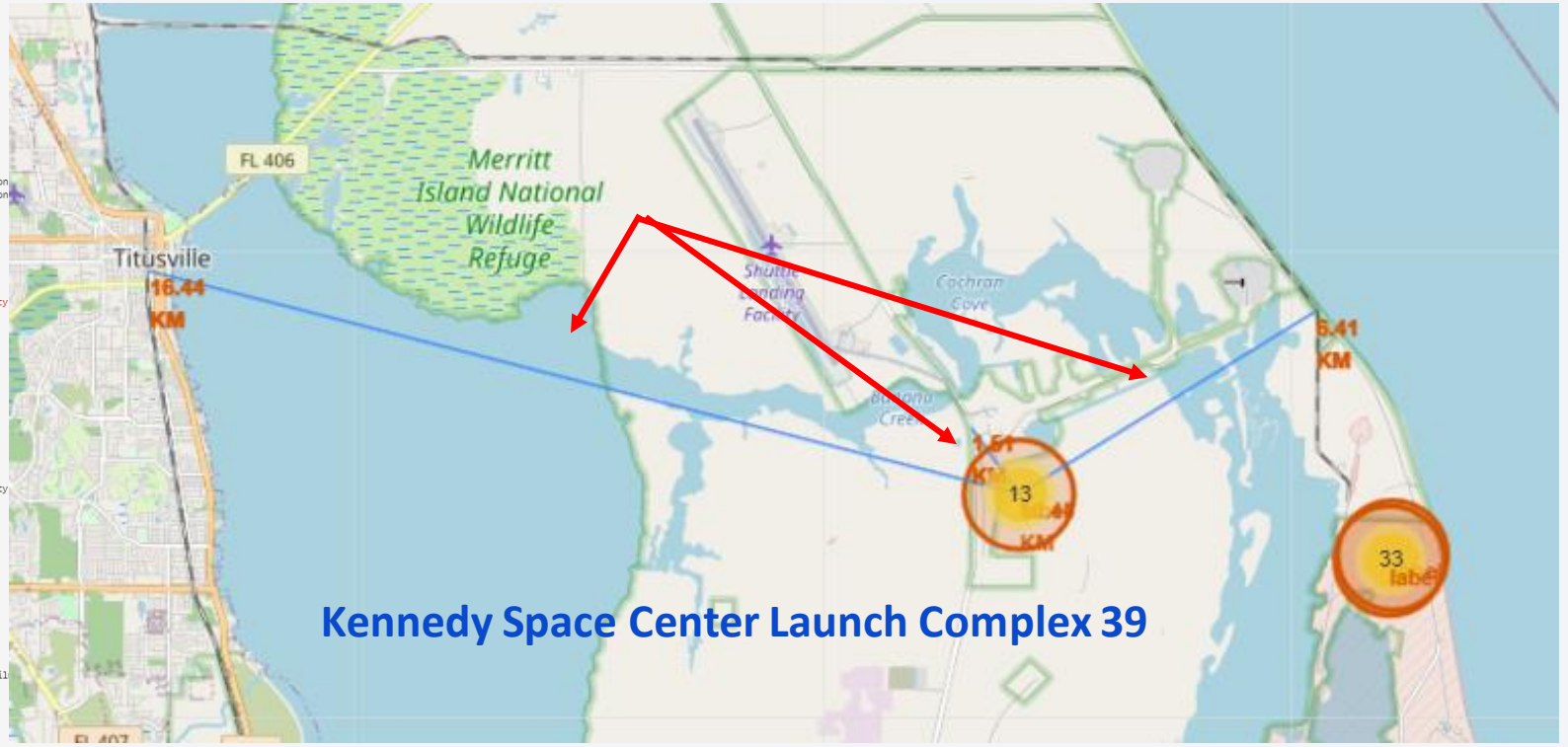
	Launch Site	Lat	Long
0	CCAFS LC-40	28.562302	-80.577356
1	CCAFS SLC-40	28.563197	-80.576820
2	KSC LC-39A	28.573255	-80.646895
3	VAFB SLC-4E	34.632834	-120.610745

```
# Create a blue circle at NASA Johnson Space Center's coordinate with a popup label showing its name
circle = folium.Circle(nasa_coordinate, radius=1000, color='#d35400', fill=True).add_child(folium.Popup('NASA Johnson Space Center'))
# Create a blue circle at NASA Johnson Space Center's coordinate with a icon showing its name
marker = folium.map.Marker(
    nasa_coordinate,
    # Create an icon as a text label
    icon=DivIcon(
        icon_size=(20,20),
        icon_anchor=(0,0),
        html='<div style="font-size: 12; color:#d35400;"><b>%s</b></div>' % 'NASA JSC',
        )
    )
site_map.add_child(circle)
site_map.add_child(marker)
```



Closest city, railway, highway from KSC LC-39A

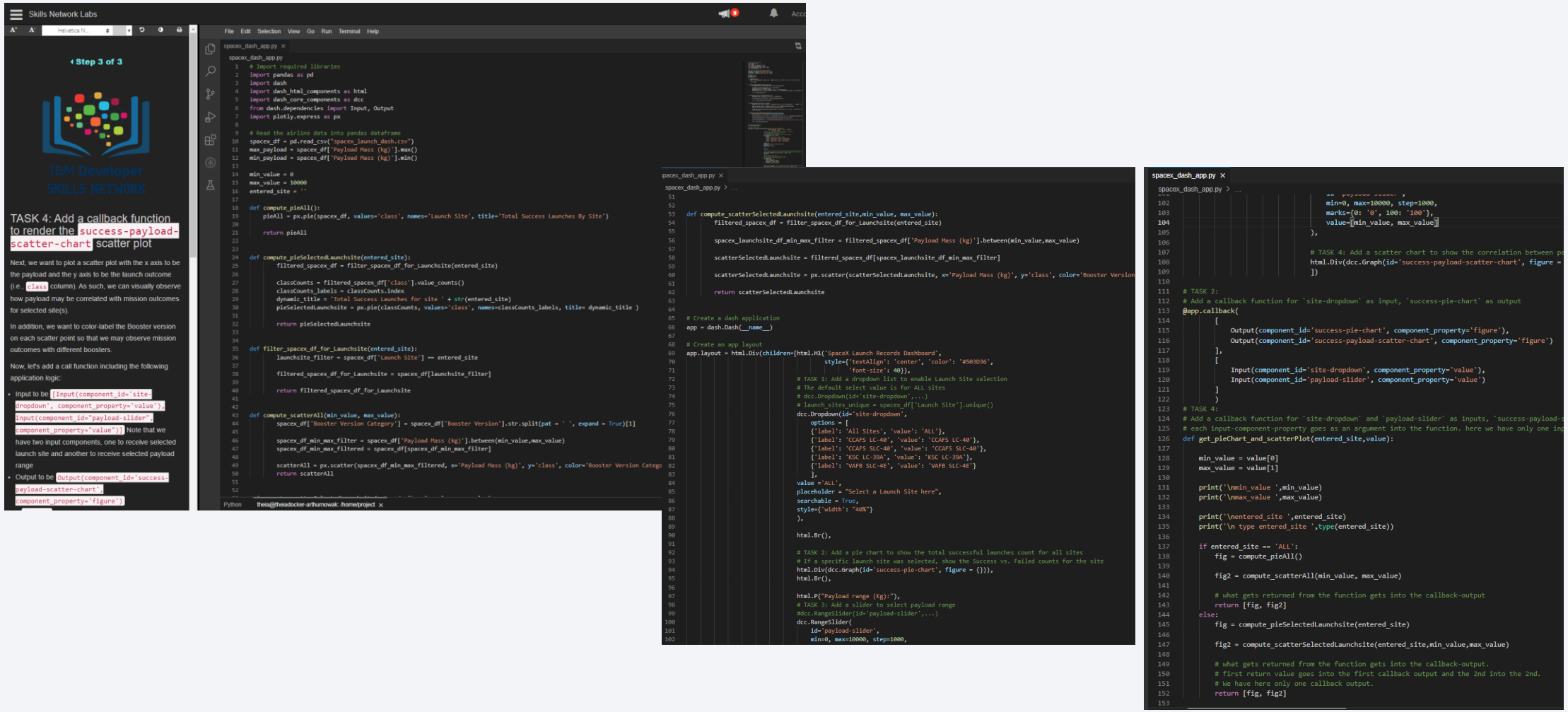
```
origin_city_lat = 28.61017
origin_city_lon = -80.80988
origin_city = [origin_city_lat, origin_city_lon]

origin_railway_lat = 28.60359
origin_railway_lon = -80.59107
origin_railway = [origin_railway_lat, origin_railway_lon]

origin_highway_lat = 28.58431
origin_highway_lon = -80.65585
origin_highway = [origin_highway_lat, origin_highway_lon]

launch_site_lat = 28.573255
launch_site_lon = -80.646895
dest_KSC_LC_39A_location_coords = [launch_site_lat, launch_site_lon]

# distance_coast = calculate_distance(launch_site_lat, launch_site_lon, origin_coast_lat, origin_coast_lon)
distance_city = calculate_distance(launch_site_lat, launch_site_lon, origin_city_lat, origin_city_lon)
distance_railway = calculate_distance(launch_site_lat, launch_site_lon, origin_railway_lat, origin_railway_lon
distance_highway = calculate_distance(launch_site_lat, launch_site_lon, origin_highway_lat, origin_highway_lon

"""
distance_marker_coast = folium.Marker(
    origin_coast,
    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_city
        )
    )
site_map.add_child(distance_marker_coast)

lines_coast = folium.PolyLine(locations=[origin_coast, dest_KSC_LC_39A_location_coords], weight=1)
site_map.add_child(lines_coast)
"""

distance_marker_city = folium.Marker(
    origin_city,
    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_city
        )
    )
site_map.add_child(distance_marker_city)

lines_city = folium.PolyLine(locations=[origin_city, dest_KSC_LC_39A_location_coords], weight=1)
site_map.add_child(lines_city)


distance_marker_railway = folium.Marker(
    origin_railway,
    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_rail
        )
    )
site_map.add_child(distance_marker_railway)

lines_railway = folium.PolyLine(locations=[origin_railway, dest_KSC_LC_39A_location_coords], weight=1)
site_map.add_child(lines_railway)
```



Build a Dashboard with Plotly Dash

- In an interactive dashboard with pie charts were created to get an overview of the launch success rate for each launch site.
- For each launch site a dynamic scatterplot was set up. It visualizes the launch success for each payload mass and booster version category
- GitHub URL of the completed Plotly Dash lab code: https://github.com/ArthurNow/DataScienceMaLeCapstone/blob/057fd914ef4990c293b52e2659a1855ebd9ae52e/07_BuildAnInteractiveDashboardWithPlotyDash

Build a Dashboard with Plotly Dash



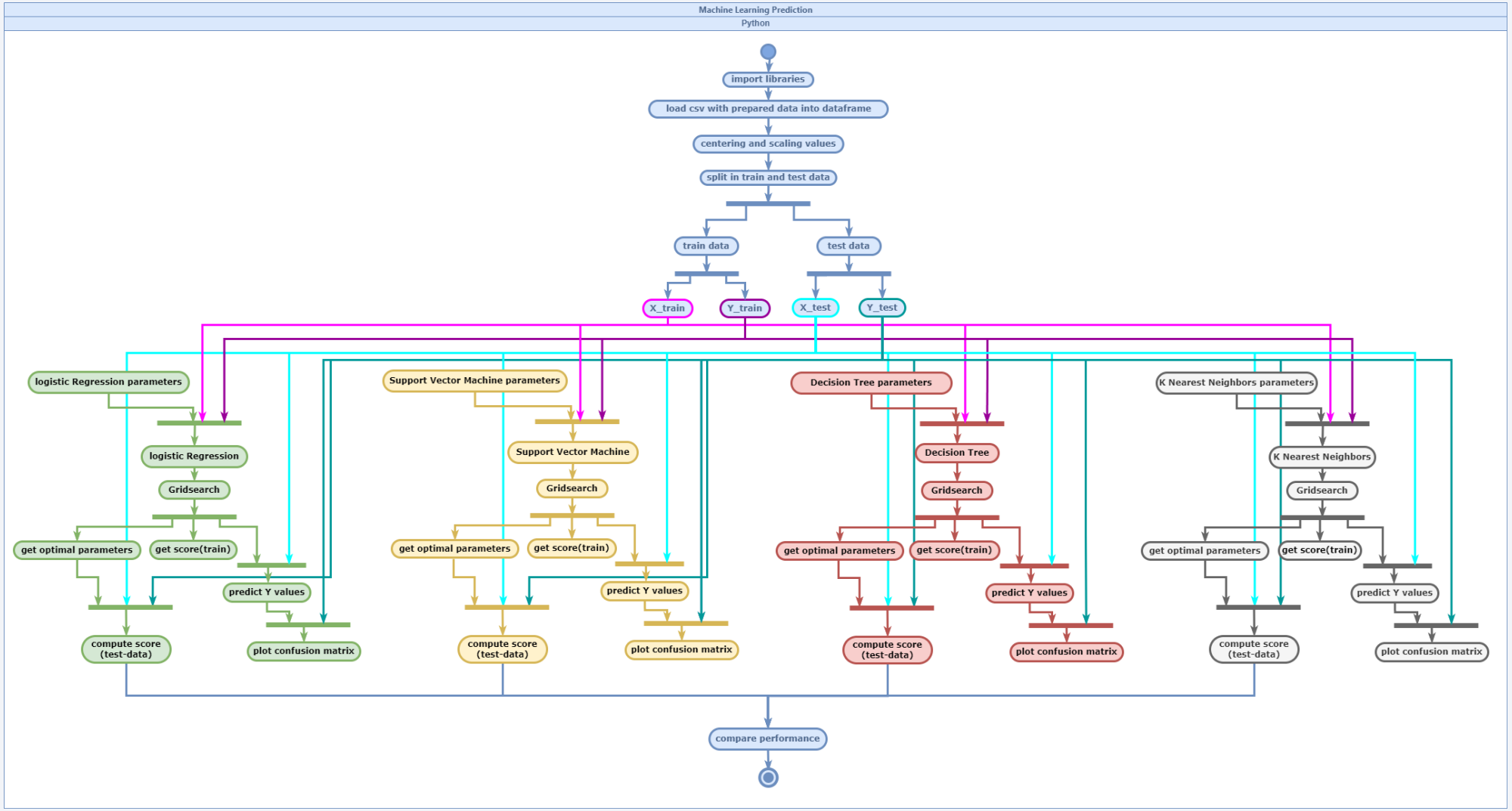
Predictive Analysis (Classification)

- Sklearn was imported for its classification functions.
- The already prepared spacex-data was loaded into a dataframe.
- Standardscaler was used to center the data and scale it. The mean was removed and it was scaled to unit variance.
- The data was split into a training and a testing set.
  20% of the data was held back for testing and evaluation of the model.
- For each classifier an object was created.
- The Gridsearch function was fed with:
  - The classifier-specific tuning parameters
  - the training data (observed data and outcome)
  - the classification object
  - a number for the desired cross-validation-folds

Predictive Analysis (Classification)

- For each classifier:
  - The parameters that produced the score (that best fit the observed train data across all folds) were printed.
  - The score for the training data was printed.
  - A prediction was made for the observed test data
  - A confusion matrix was build to compare predicted test data and actual test data.
  - The score for the test data was calculated.
- The models were evaluated by comparing the scores.

Predictive Analysis (Classification)- flowchart

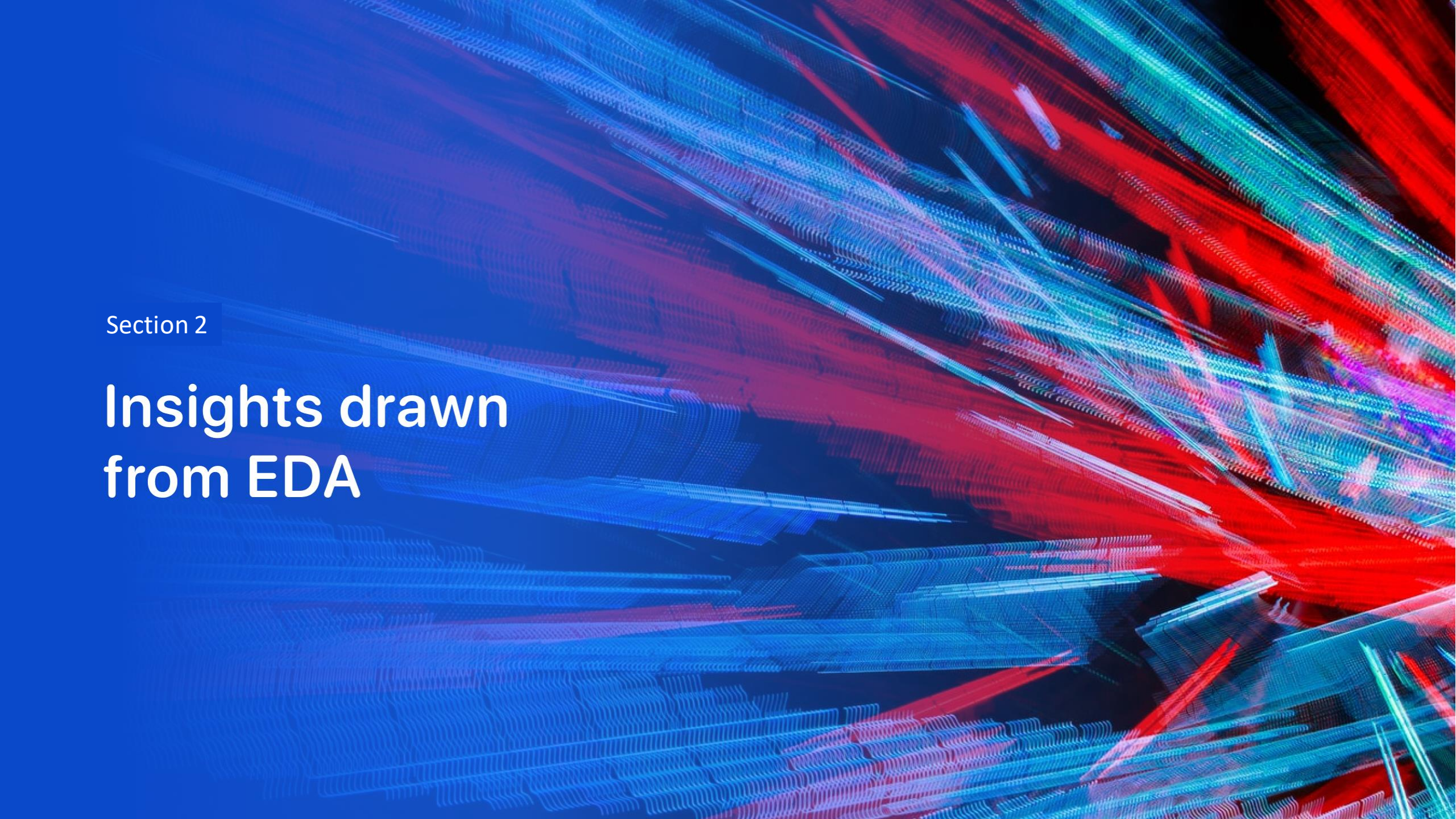


Predictive Analysis (Classification)

GitHub URL of the completed predictive analysis lab:
https://github.com/ArthurNow/DataScienceMaLeCapstone/blob/057fd914ef4990c293b52e2659a1855ebd9ae52e/SpaceX_Machine%20Learning%20Prediction_Part_5%20(9).ipynb

Section 2

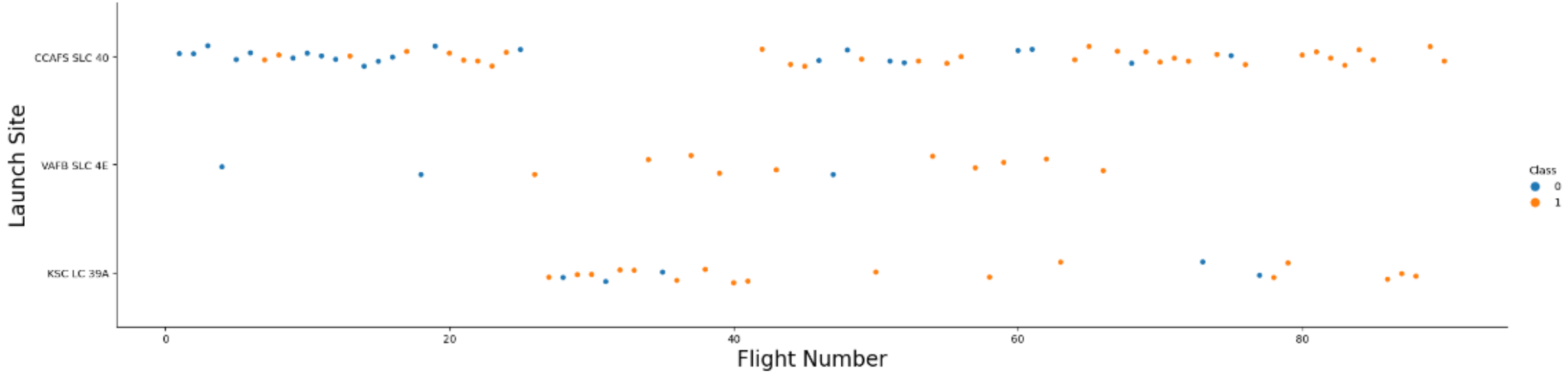
Insights drawn from EDA



Flight Number vs. Launch Site

- Cape Canaveral Space Launch Complex 40 (CCAFS LC-40) was used for the lowest flight numbers most often.
- Up to flight number 20 most launches were unsuccessful.
- The launch success improved with flight numbers.

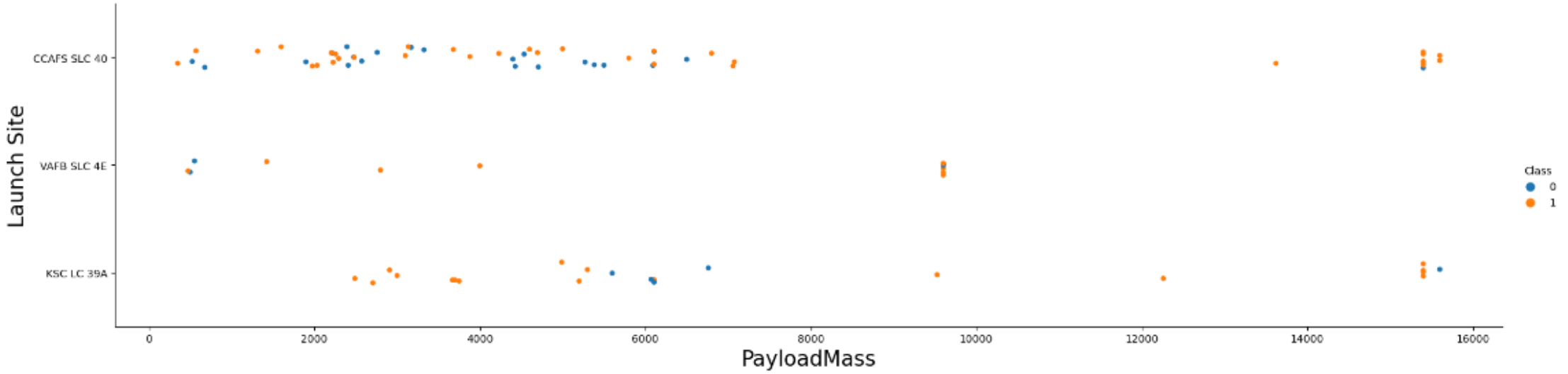
```
# Plot a scatter point chart with x axis to be Flight Number and y axis to be the launch site, and hue to be the class value
sns.catplot(y="LaunchSite", x="FlightNumber", hue="Class", data=df, aspect = 4)
plt.xlabel("Flight Number",fontsize=20)
plt.ylabel("Launch Site",fontsize=20)
plt.show()
```



Payload vs. Launch Site

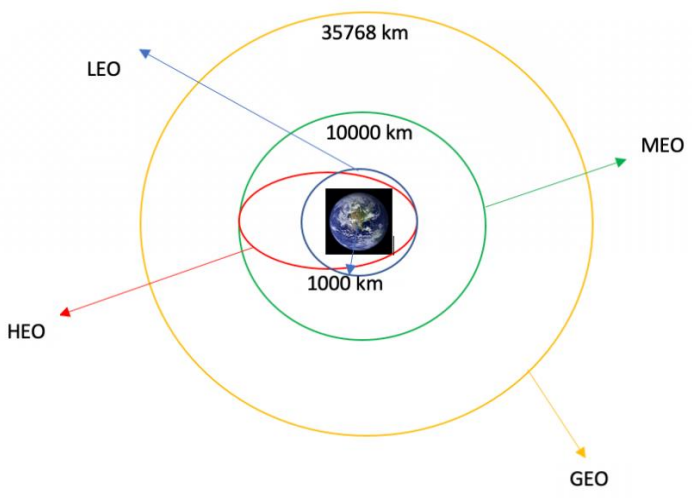
- Vandenberg Space Launch Complex 4 (VAFB SLC-4E) was not used for payload masses larder than 10000 kg.
- Payload masses up to 8000 kg had no impact on launch success in Cape Canaveral Space Launch Complex 40.
- The launch success improved with flight numbers.

```
# Plot a scatter point chart with x axis to be Pay Load Mass (kg) and y axis to be the launch site, and hue to be the class value
sns.catplot(y="LaunchSite", x="PayloadMass", hue="Class", data=df, aspect = 4)
plt.xlabel("PayloadMass",fontsize=20)
plt.ylabel("Launch Site",fontsize=20)
plt.show()
```

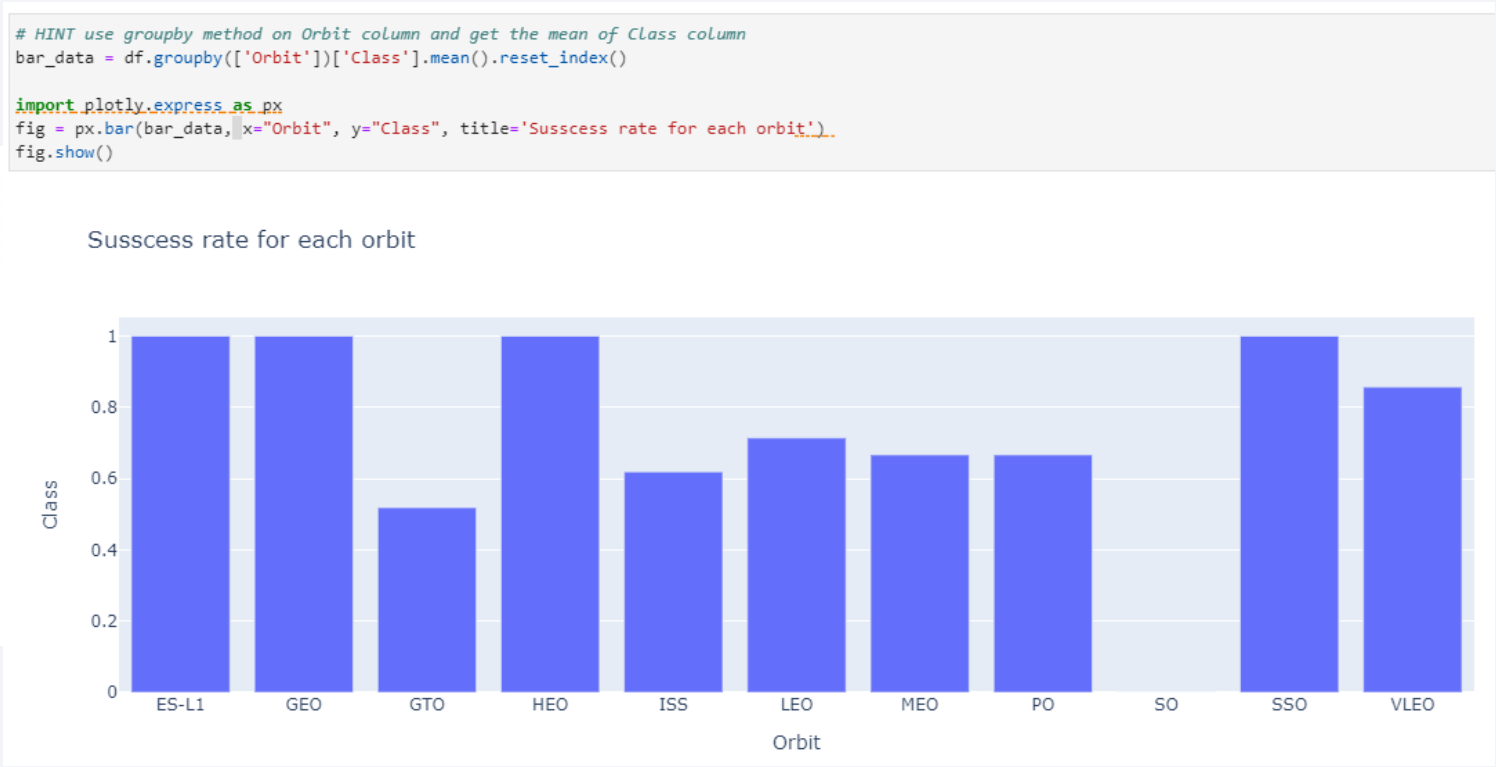


Success Rate vs. Orbit Type

- The data for SSO (or SO)-orbit (Sun-synchronous orbit) needs further inspection, because both synonyms are present.
- Launches to orbits ES-L1, GEO, HEO, SSO seem to have a 100% success rate.



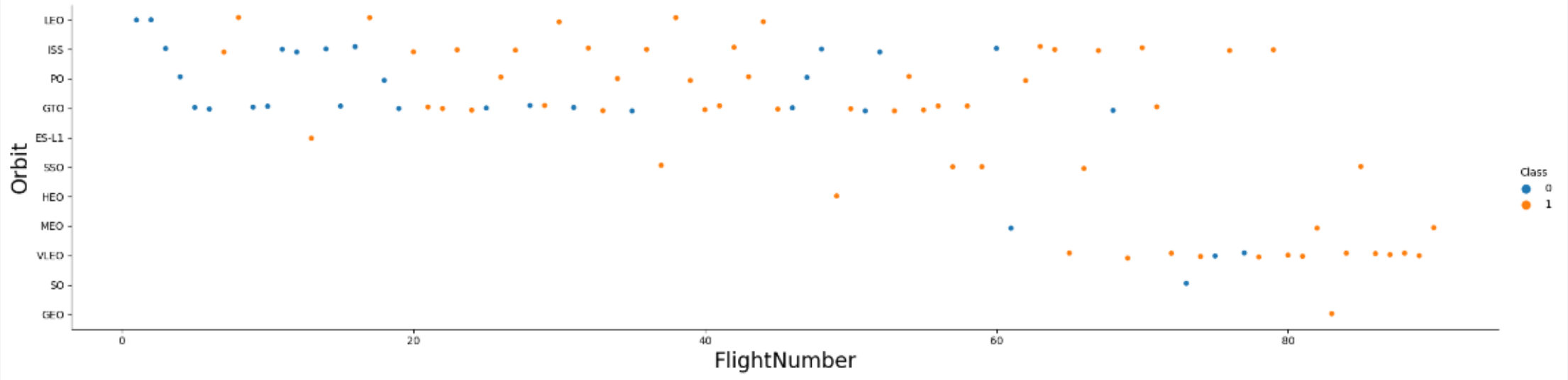

```
# HINT use groupby method on Orbit column and get the mean of Class column
bar_data = df.groupby(['Orbit'])['Class'].mean().reset_index()

import plotly.express as px
fig = px.bar(bar_data, x="Orbit", y="Class", title='Susscess rate for each orbit')
fig.show()
```



Flight Number vs. Orbit Type

- Launch success for the GEO-orbit seems not correlated to the flight number.
- For other orbits launch success improves with increasing flight number.

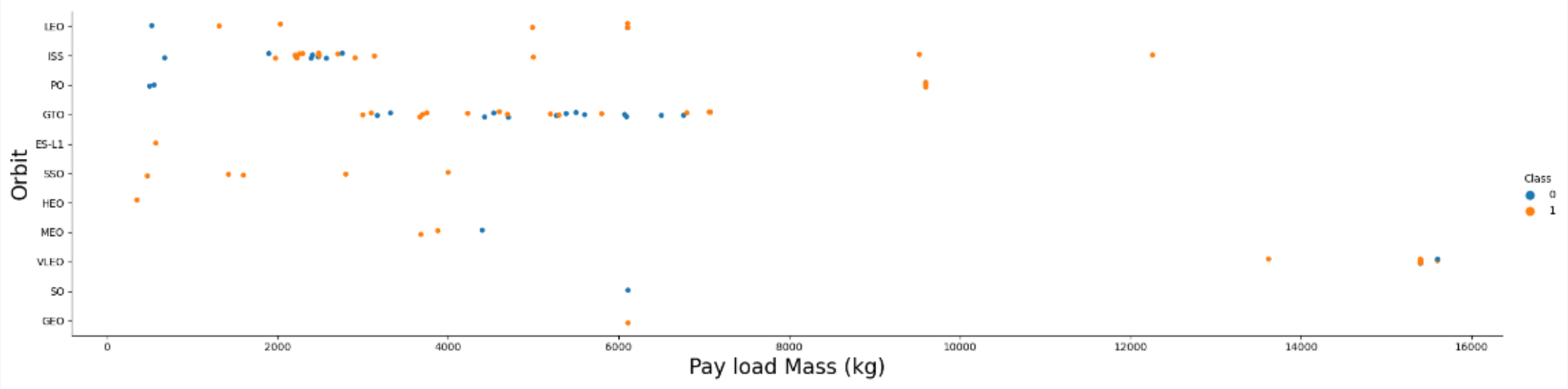
```
# Plot a scatter point chart with x axis to be FlightNumber and y axis to be the Orbit, and hue to be the class value
sns.catplot(y="Orbit", x="FlightNumber", hue="Class", data=df, aspect = 4)
plt.xlabel("FlightNumber",fontsize=20)
plt.ylabel("Orbit",fontsize=20)
plt.show()
```



Payload vs. Orbit Type

- Orbits Polar, LEO and ISS have better success rates with heavy payloads.
- Launches to GTO seem to have no correlations with payload mass.

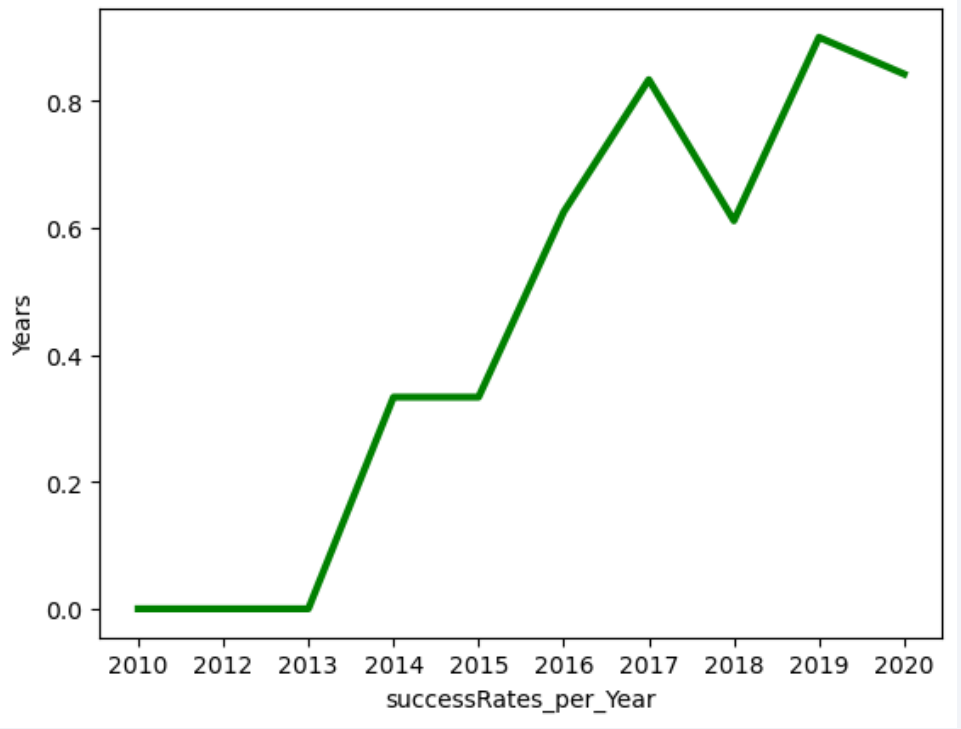
```
# Plot a scatter point chart with x axis to be Payload and y axis to be the Orbit, and hue to be the class value
sns.catplot(y="Orbit", x="PayloadMass", hue="Class", data=df, aspect = 4)
plt.xlabel("Pay load Mass (kg)",fontsize=20)
plt.ylabel("Orbit",fontsize=20)
plt.show()
```



Launch Success Yearly Trend

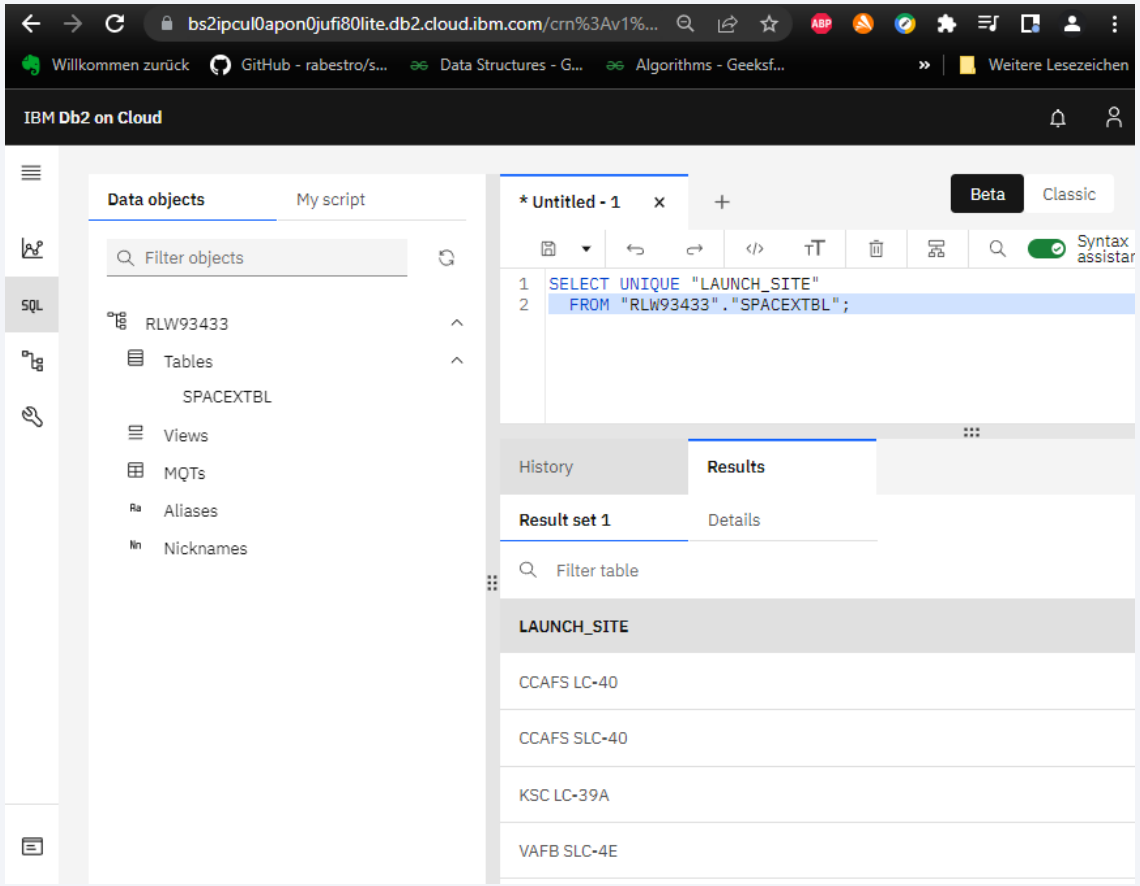
The overall launch success rate kept increasing since 2013.

```
x_data = list(sorted(Extract_unique_years()))

successRates_per_Year = calc_successRates_per_Year()
y_data = list(successRates_per_Year.values())

plt.plot(x_data, y_data, color = 'green', linewidth=3, linestyle='solid')
plt.xlabel("successRates_per_Year")
plt.ylabel("Years")

plt.show()
```



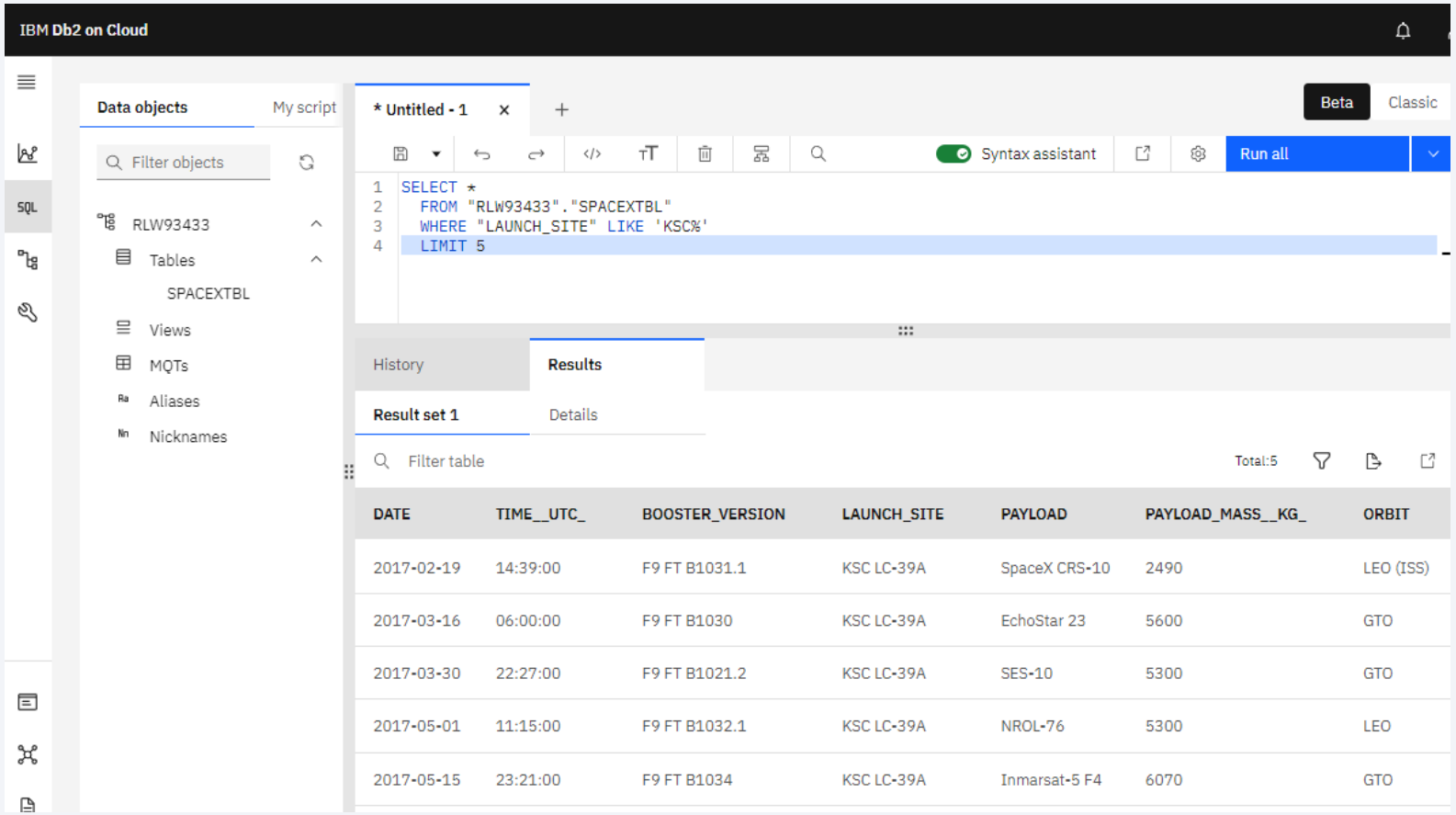
All Launch Site Names

These are the unique launch sites in the spacex data.



Launch Site Names Begin with 'KSC'

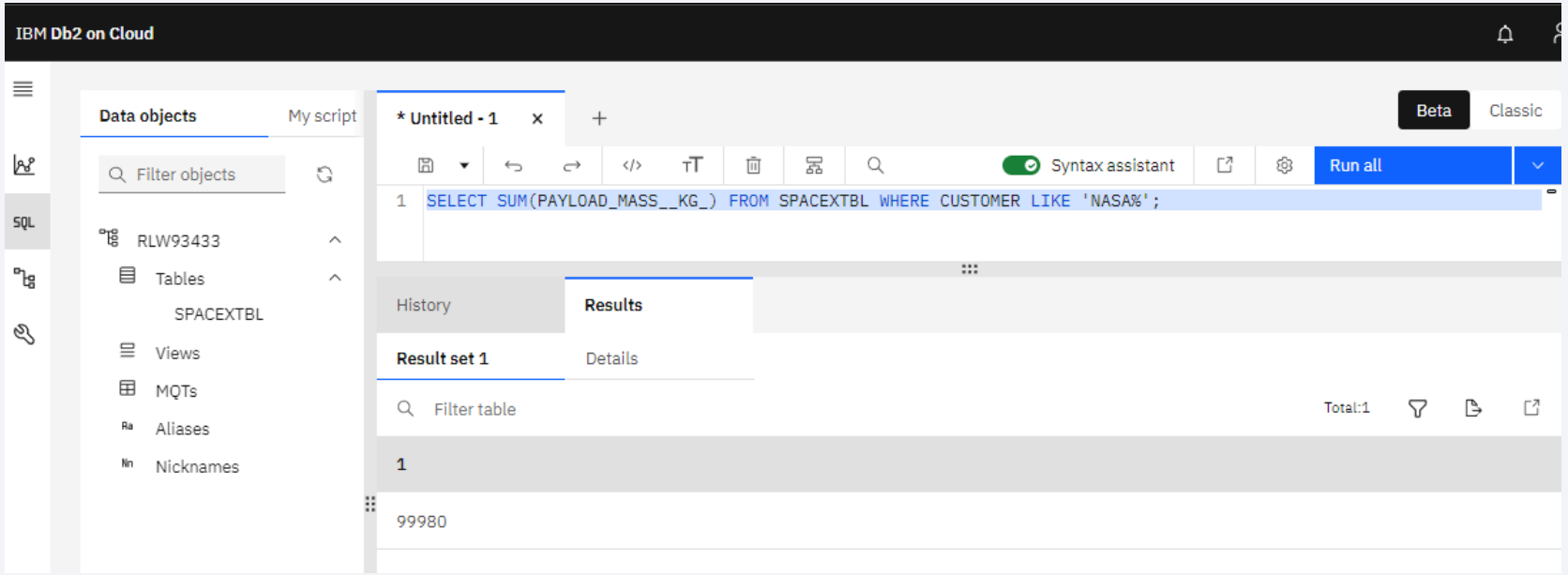
The spacex-data has at least 5 records for the Kennedy Space Center.



DATE	TIME__UTC_	BOOSTER_VERSION	LAUNCH_SITE	PAYLOAD	PAYLOAD_MASS__KG_	ORBIT
2017-02-19	14:39:00	F9 FT B1031.1	KSC LC-39A	SpaceX CRS-10	2490	LEO (ISS)
2017-03-16	06:00:00	F9 FT B1030	KSC LC-39A	EchoStar 23	5600	GTO
2017-03-30	22:27:00	F9 FT B1021.2	KSC LC-39A	SES-10	5300	GTO
2017-05-01	11:15:00	F9 FT B1032.1	KSC LC-39A	NROL-76	5300	LEO
2017-05-15	23:21:00	F9 FT B1034	KSC LC-39A	Inmarsat-5 F4	6070	GTO

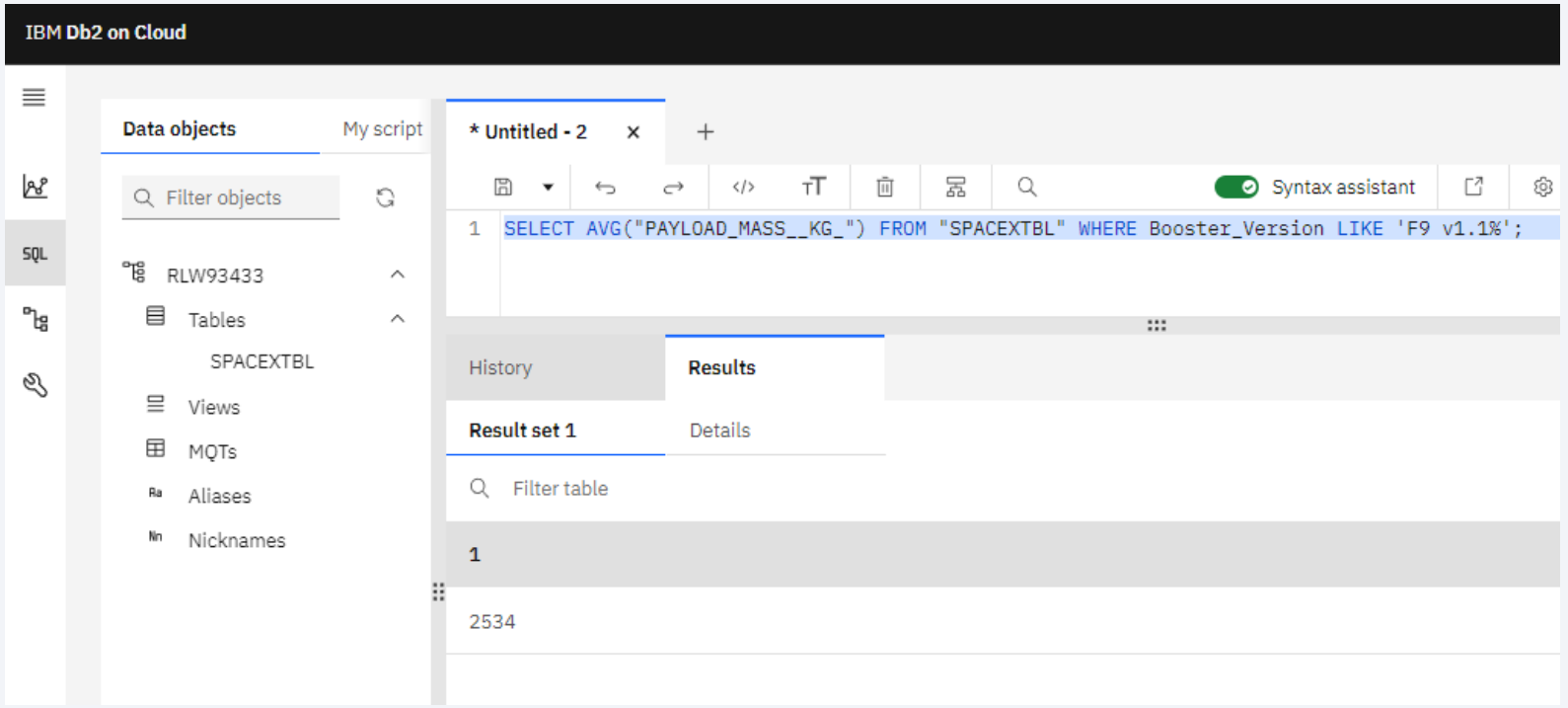
Total Payload Mass

The total payload carried by boosters from NASA is 99980 kg.



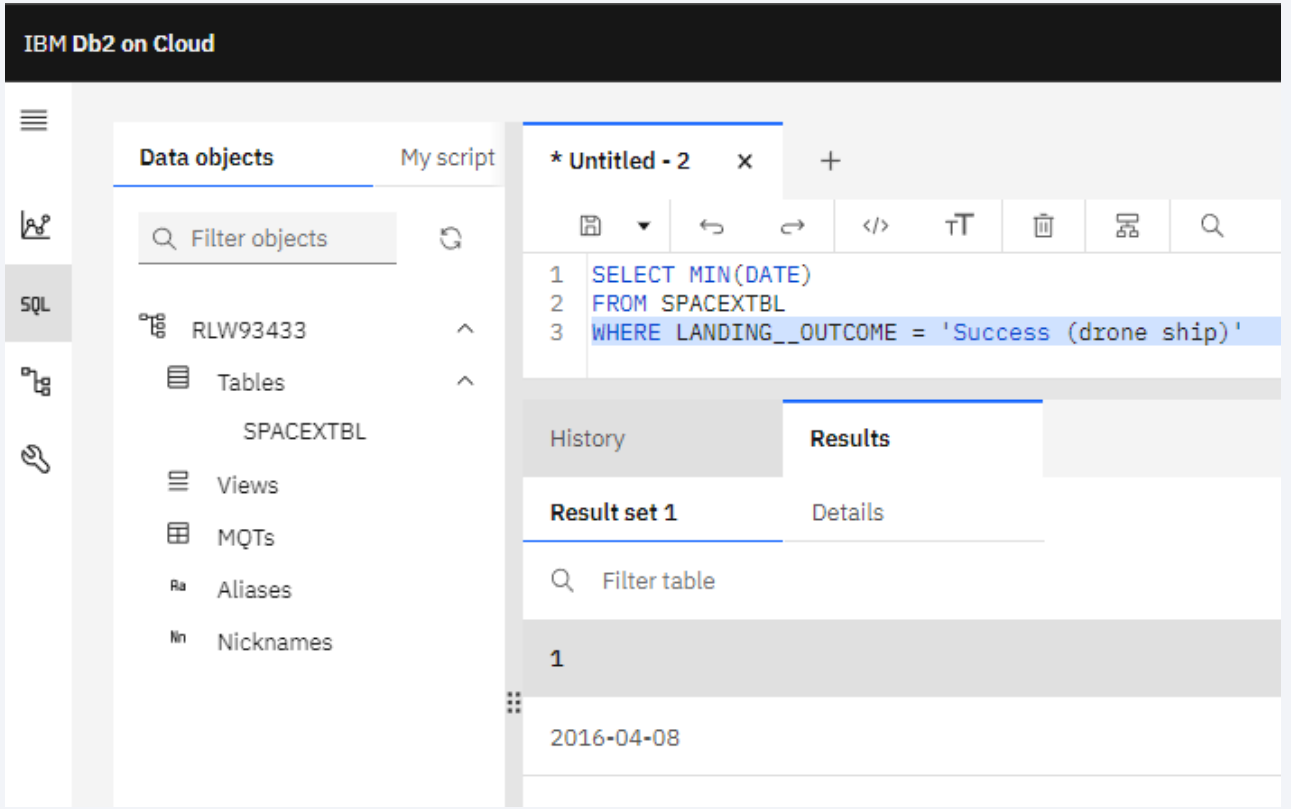
Average Payload Mass by F9 v1.1

The average payload mass carried by booster version F9 v1.1 is 2534 kg.



First Successful Ground Landing Date

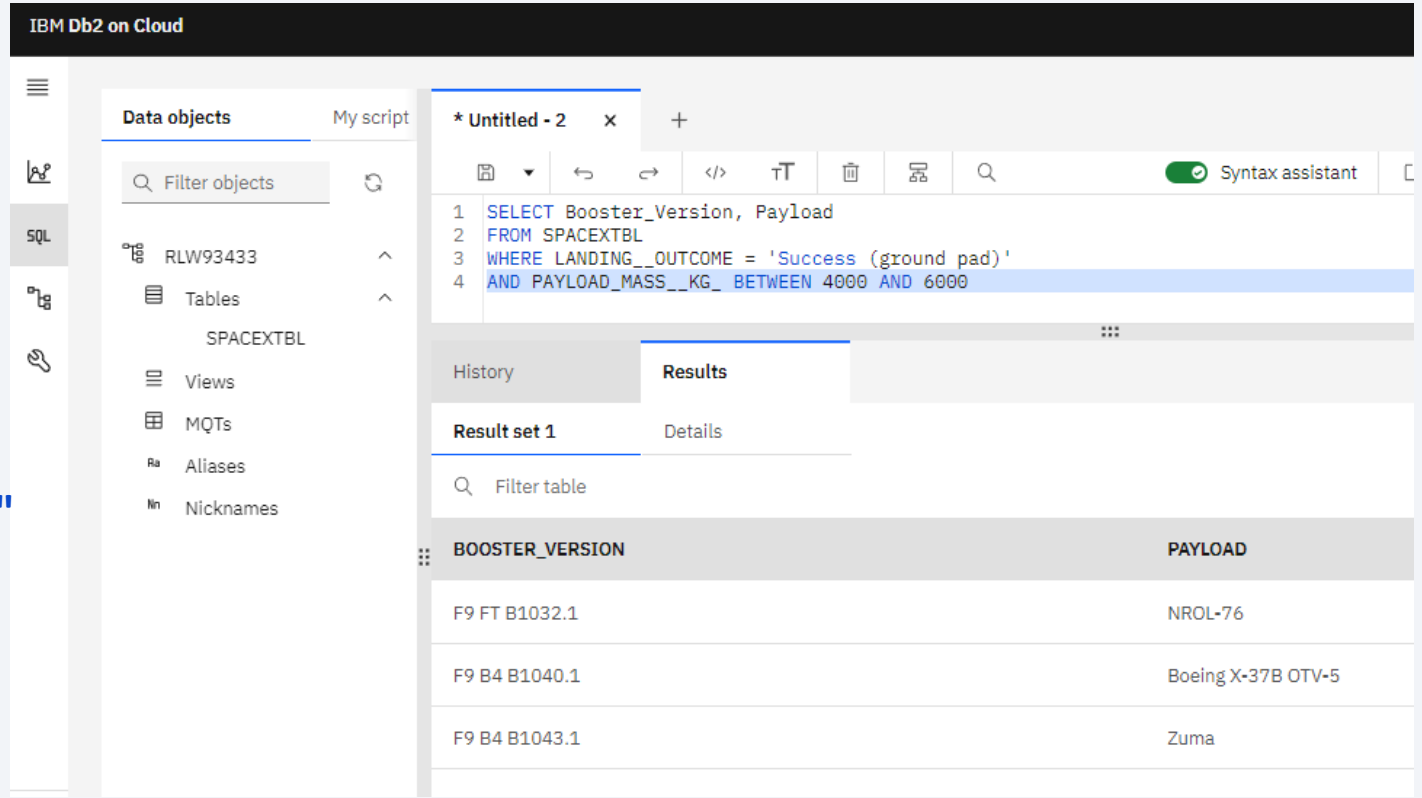
The first successful landing outcome on drone ship was 08th April 2014.



Successful Drone Ship Landing with Payload between 4000 and 6000

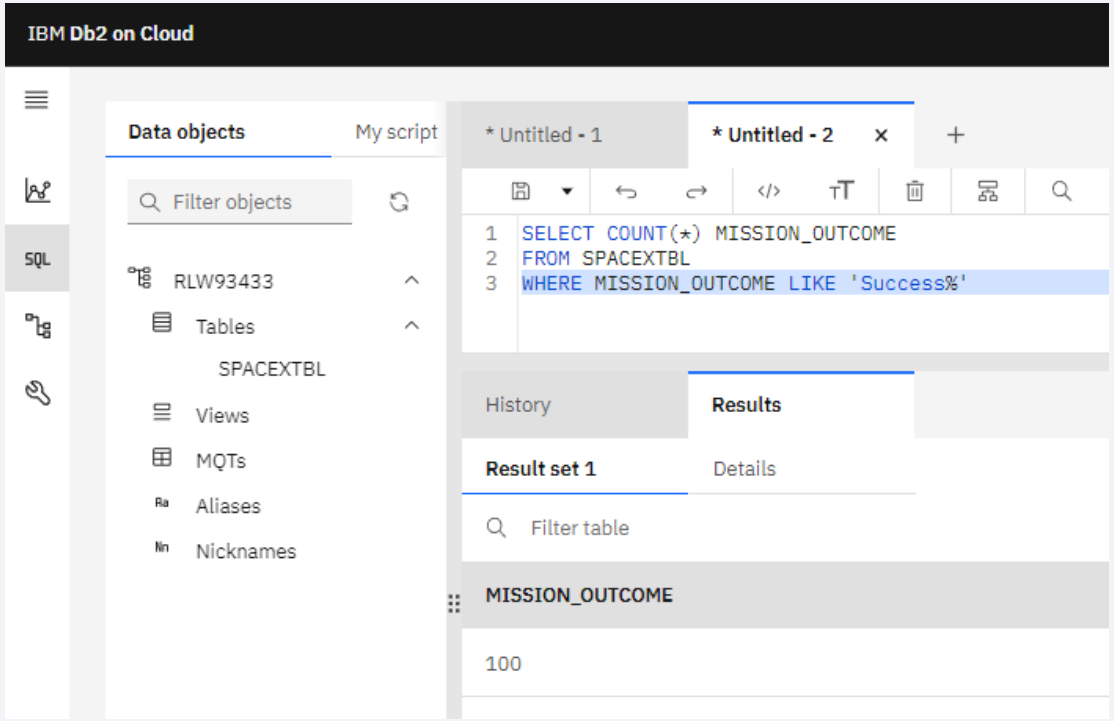
The boosters which have successfully landed on drone ship and had payload mass greater than 4000 but less than 6000 are:

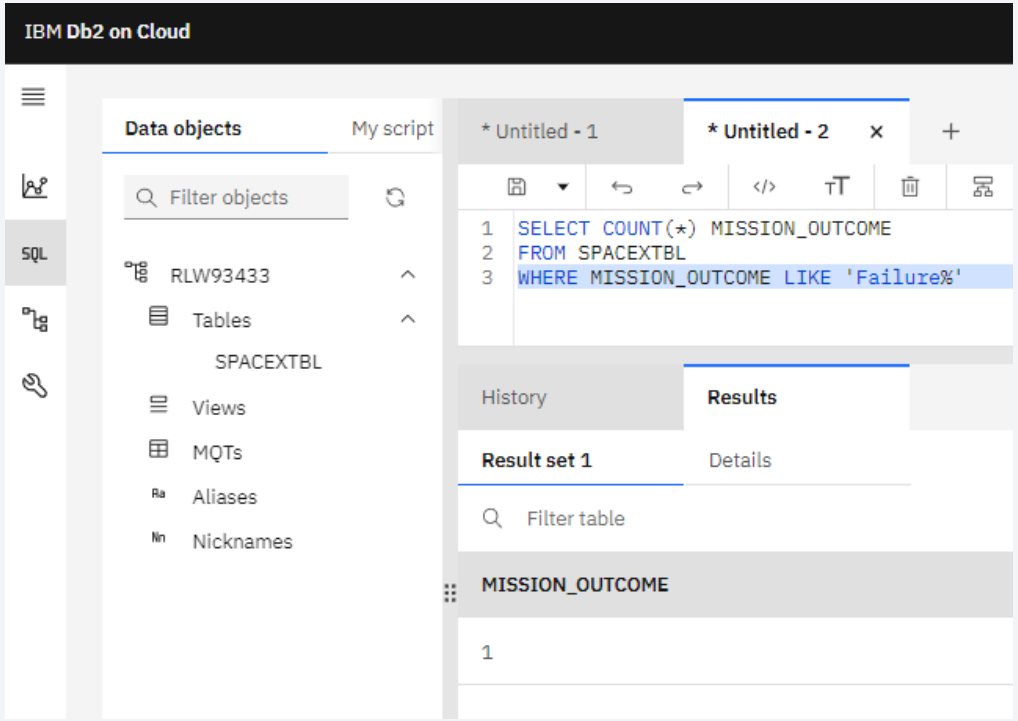
- **"F9 FT B1032.1 NROL-76"**
- **"F9 B4 B1040.1 Boeing X-37B OTV-5"**
- **"F9 B4 B1043.1 Zuma"**



Total Number of Successful and Failure Mission Outcomes

There were 100 successful mission outcomes.
There was 1 failure mission outcome.

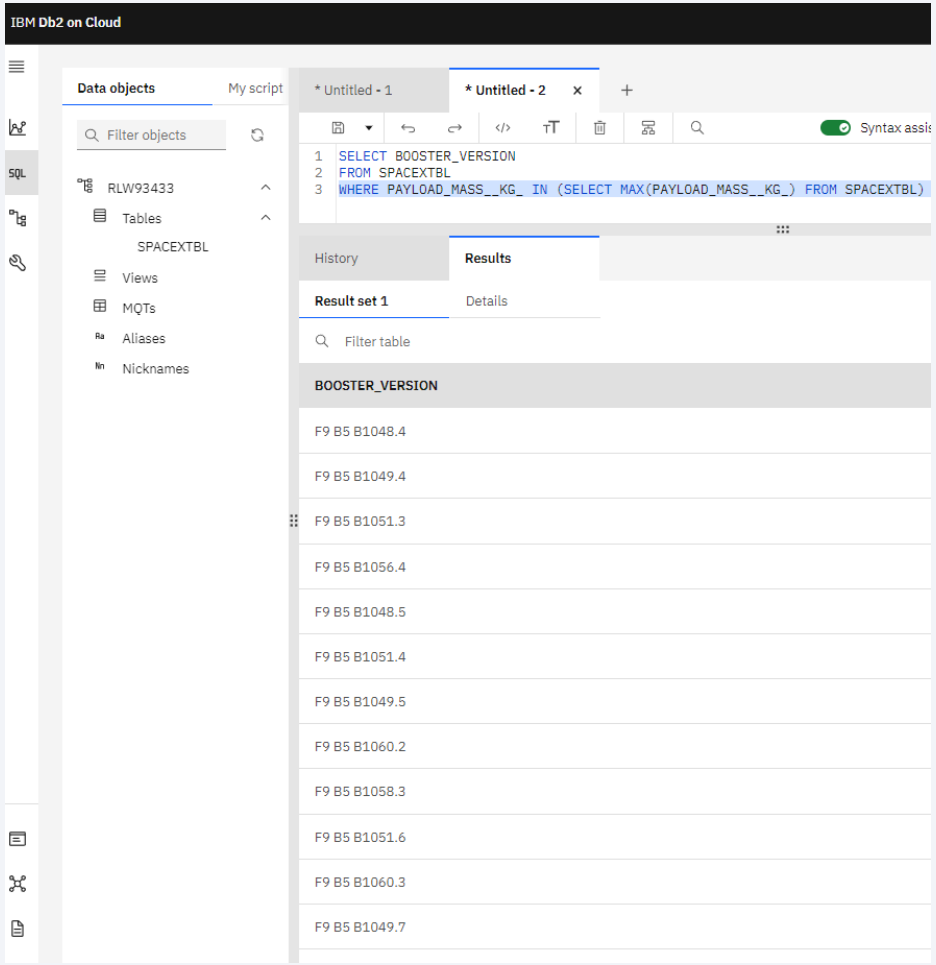




Boosters Carried Maximum Payload

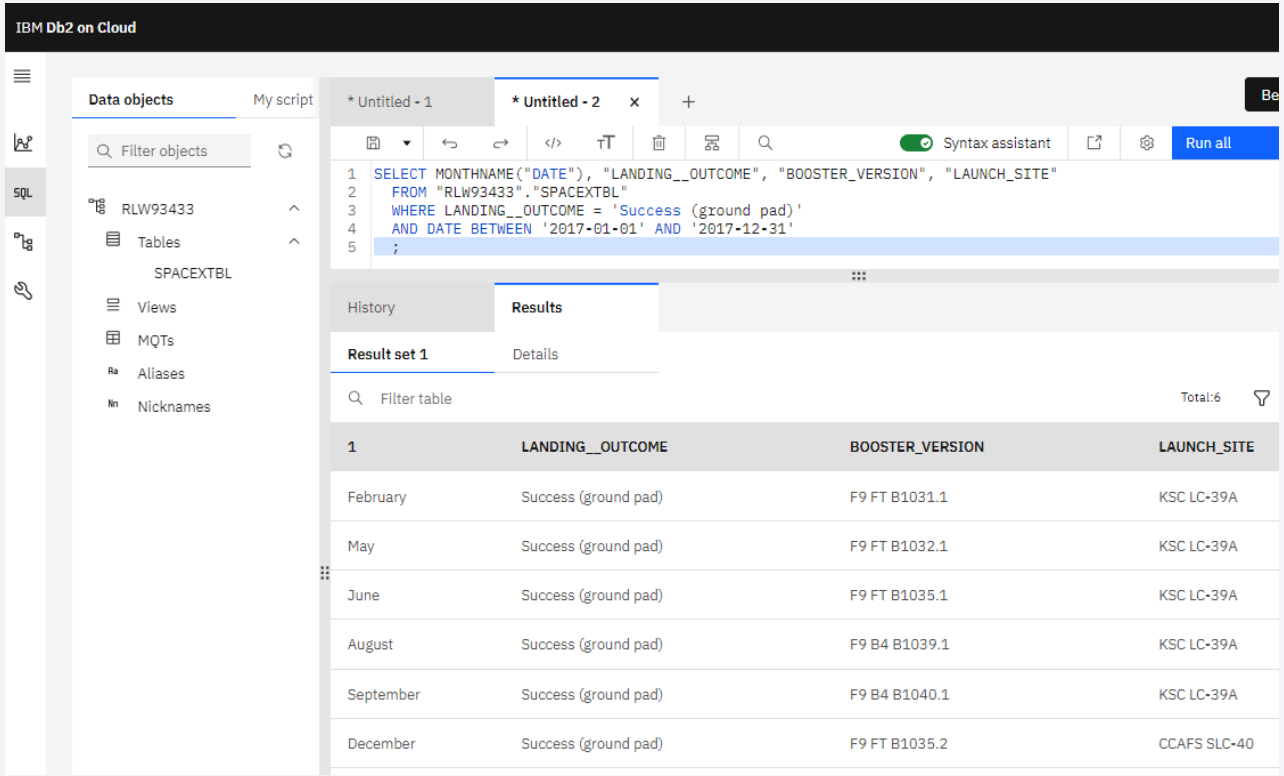
These boosters have carried the maximum payload mass:

- 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
- 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



2015 Launch Records

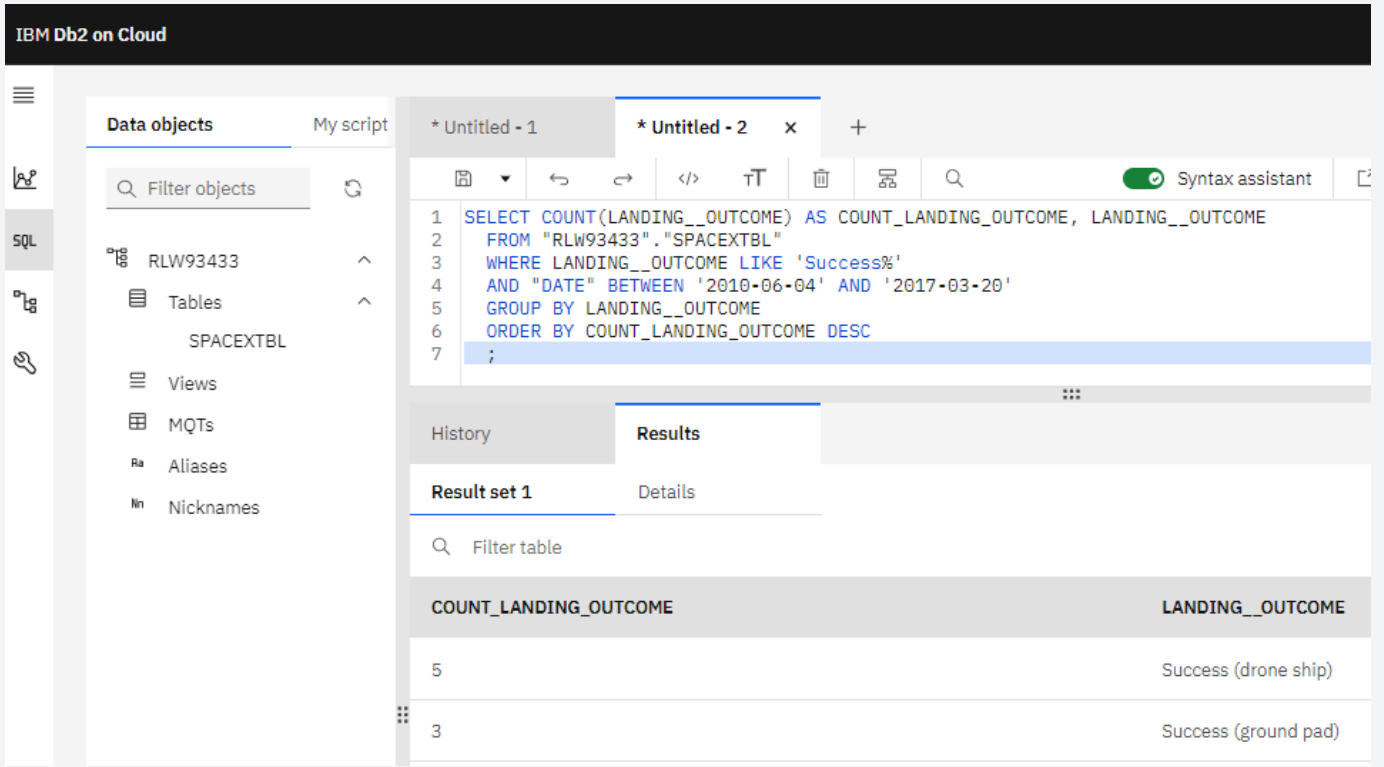
The records for the months in year 2017:



1	LANDING__OUTCOME	BOOSTER_VERSION	LAUNCH_SITE
February	Success (ground pad)	F9 FT B1031.1	KSC LC-39A
May	Success (ground pad)	F9 FT B1032.1	KSC LC-39A
June	Success (ground pad)	F9 FT B1035.1	KSC LC-39A
August	Success (ground pad)	F9 B4 B1039.1	KSC LC-39A
September	Success (ground pad)	F9 B4 B1040.1	KSC LC-39A
December	Success (ground pad)	F9 FT B1035.2	CCAFS SLC-40

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

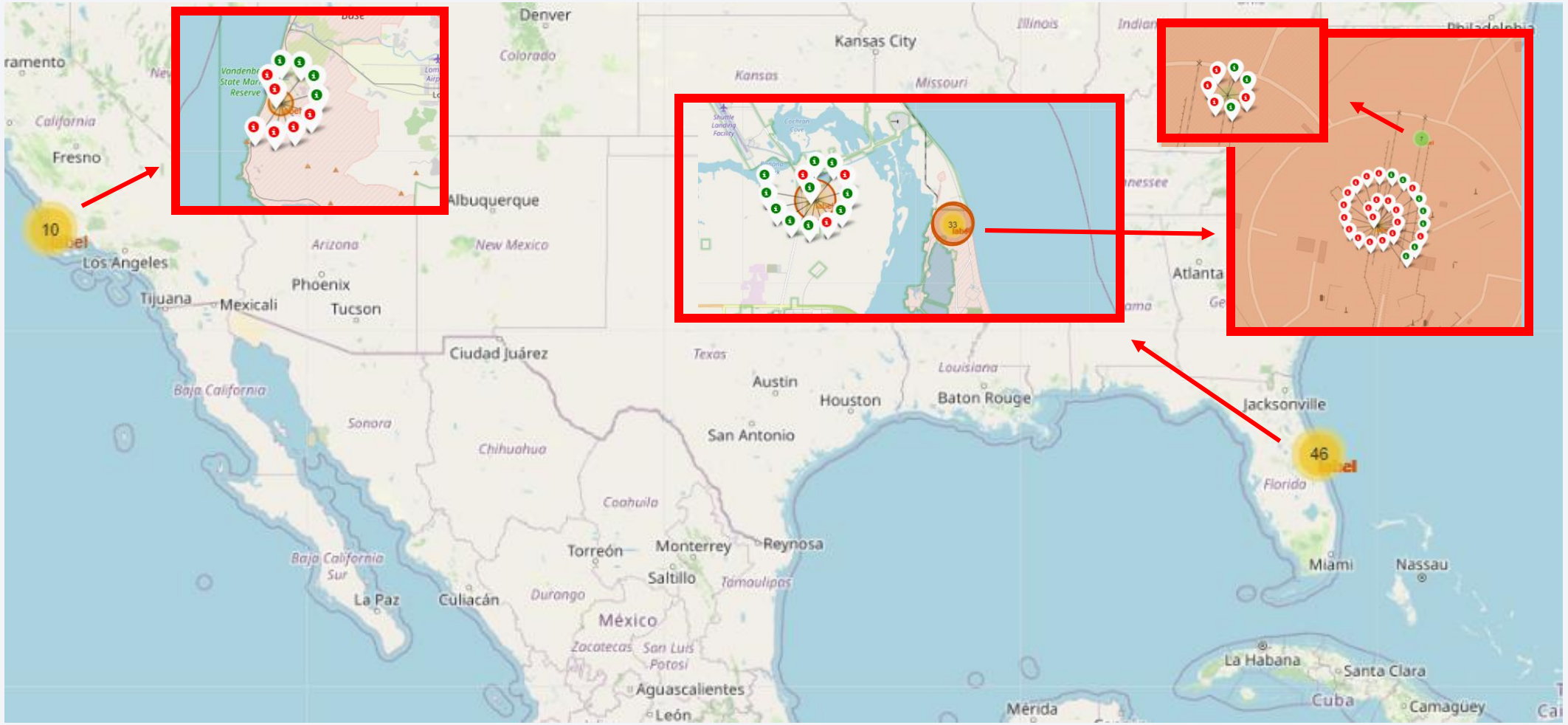
The count of successful landing outcomes between the date 04th June 2010 and 20th March 2017 in descending order:




Section 3
Launch Sites
Proximities Analysis

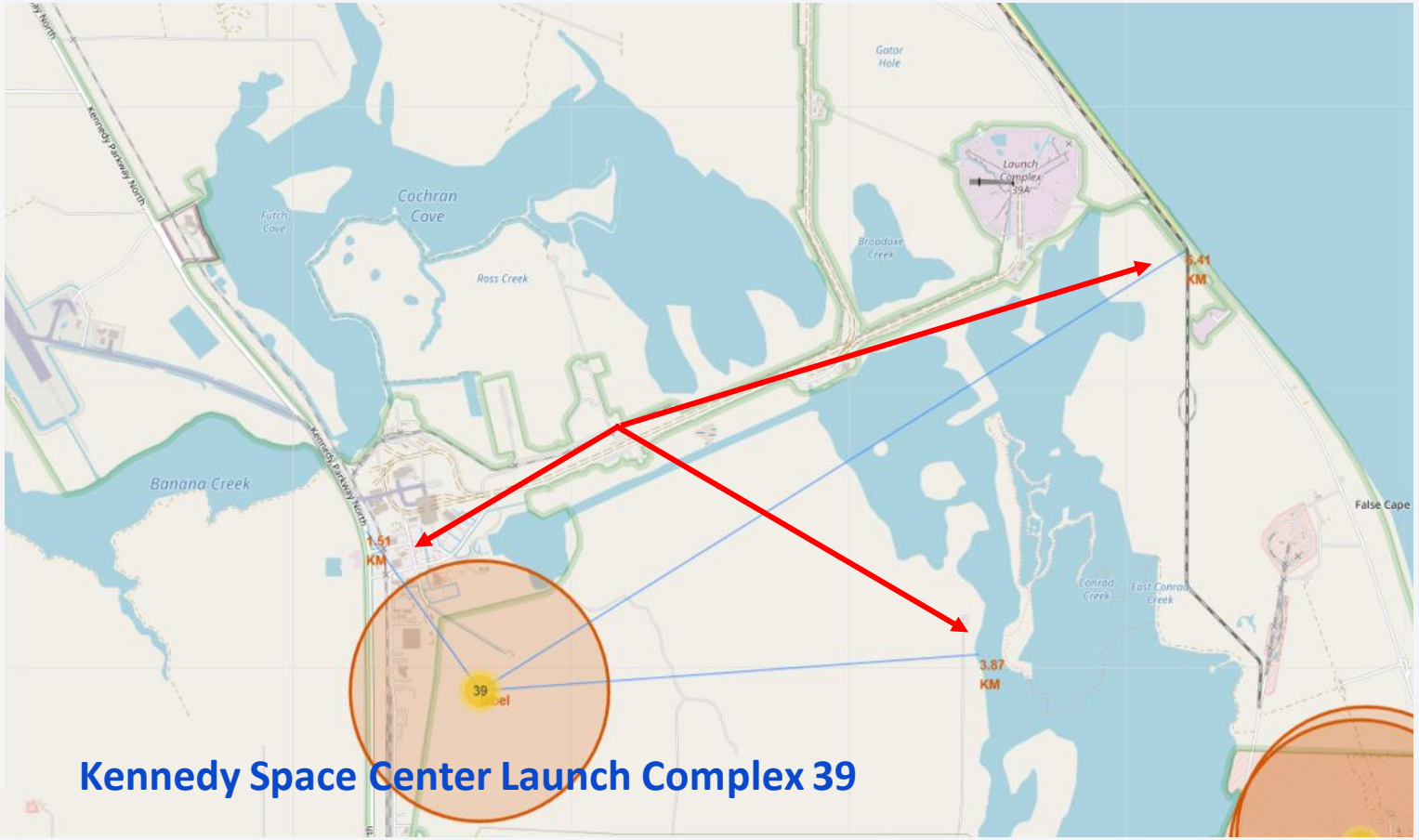
launch sites' location markers

color-labeled launch outcomes on the map

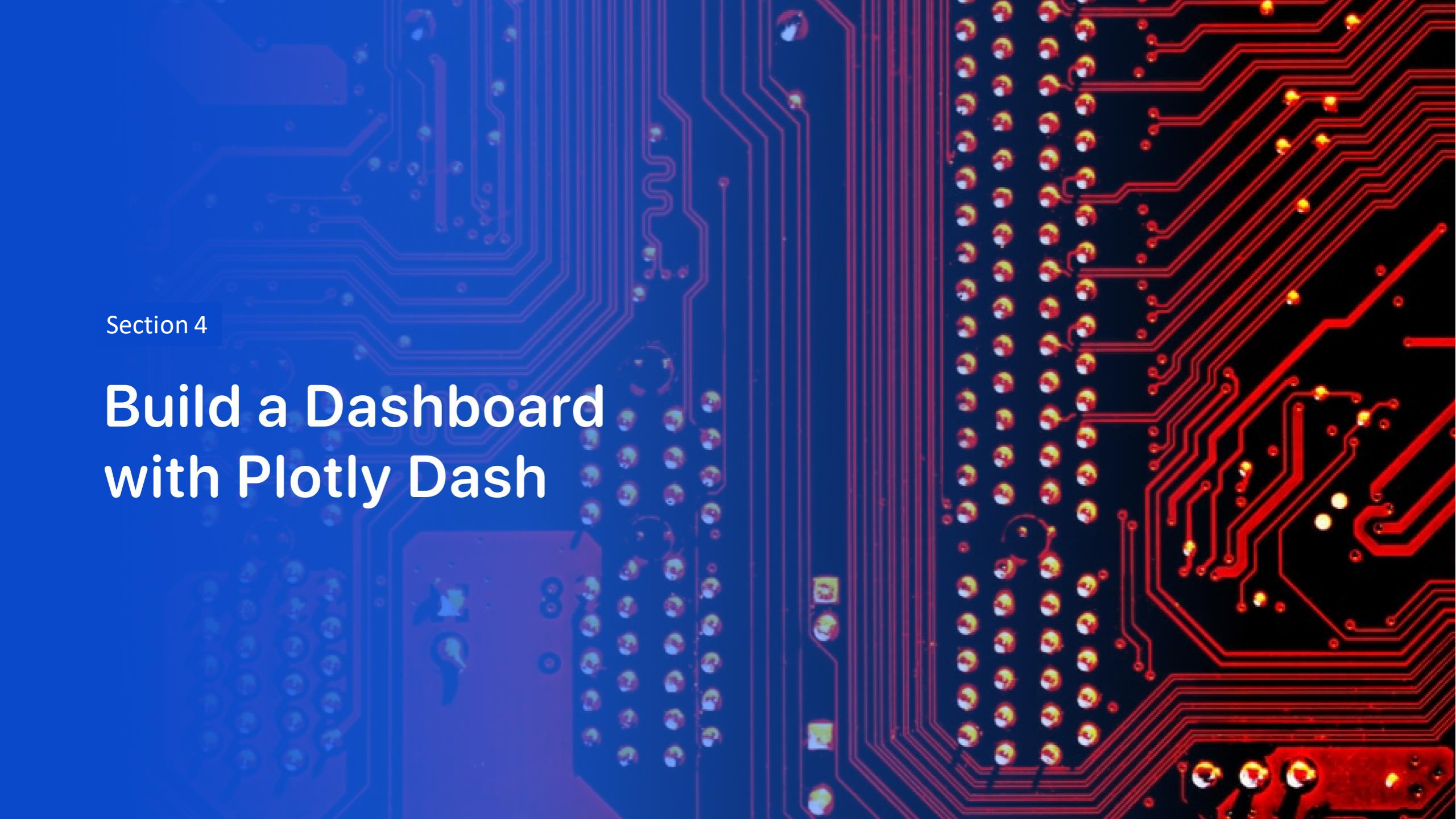


Closest railway, highway, coastline from KSC LC-39A

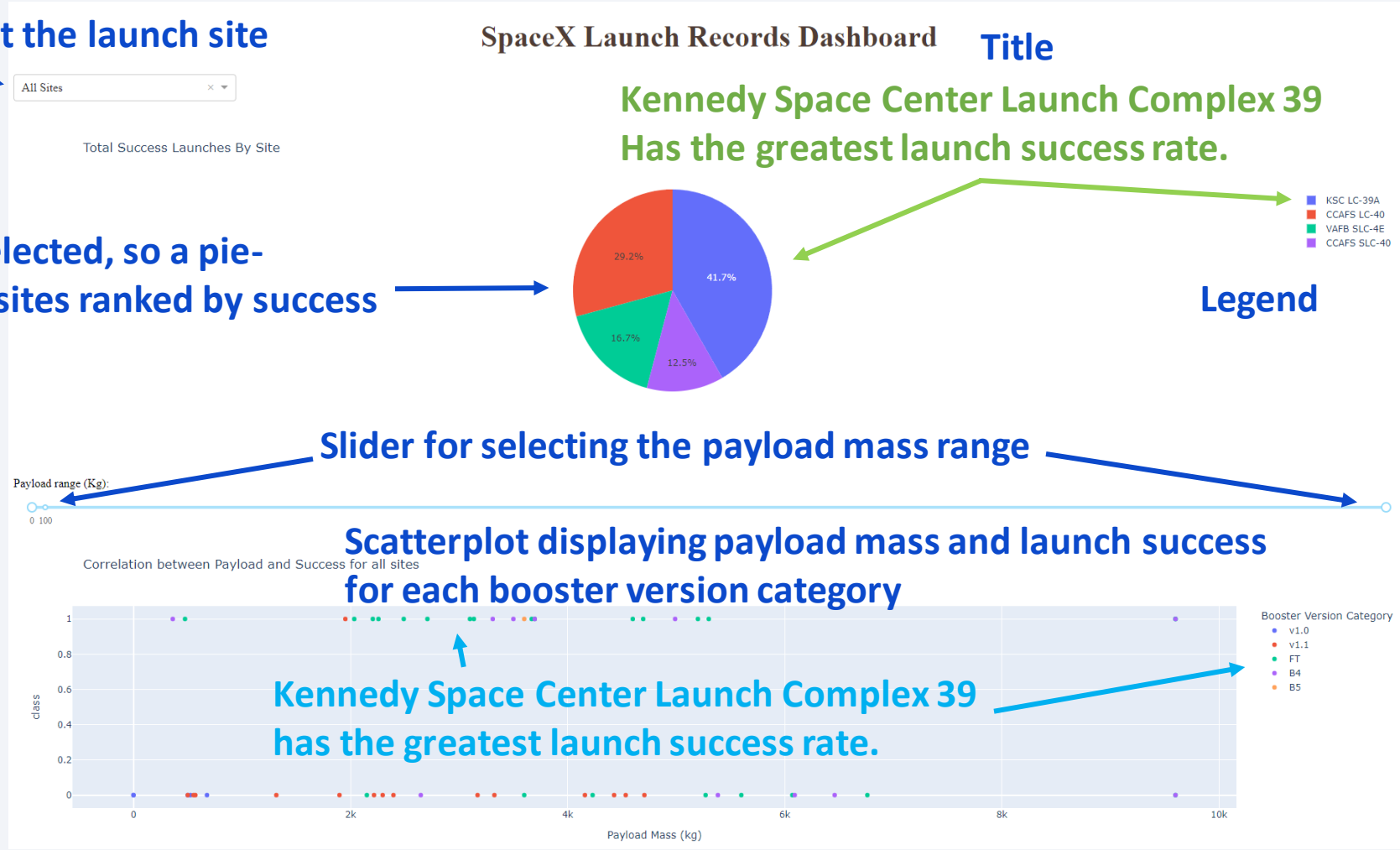
```
origin_city_lat = 28.61017
origin_city_lon = -80.80988
origin_city = [origin_city_lat, origin_city_lon]

origin_railway_lat = 28.60359
origin_railway_lon = -80.59107
origin_railway = [origin_railway_lat, origin_railway_lon]

origin_highway_lat = 28.58431
origin_highway_lon = -80.65585
origin_highway = [origin_highway_lat, origin_highway_lon]

launch_site_lat = 28.573255
launch_site_lon = -80.646895
dest_KSC_LC_39A_location_coords = [launch_site_lat, launch_site_lon]

# distance_coast = calculate_distance(launch_site_lat, launch_site_lon, origin_coast_lat, origin_coast_lon)
distance_city = calculate_distance(launch_site_lat, launch_site_lon, origin_city_lat, origin_city_lon)
distance_railway = calculate_distance(launch_site_lat, launch_site_lon, origin_railway_lat, origin_railway_lon)
distance_highway = calculate_distance(launch_site_lat, launch_site_lon, origin_highway_lat, origin_highway_lon)

"""
distance_marker_coast = folium.Marker(
    origin_coast,
    icon=DivIcon(
        icon_size=(20,20),
        icon_anchor=(0,0),
        html='<div style="font-size: 12; color:#d35400;"><b>%s</b></div>' % "{:.2f} KM".format(distance_city),
        )
    )
site_map.add_child(distance_marker_coast)

lines_coast = folium.PolyLine(locations=[origin_coast, dest_KSC_LC_39A_location_coords], weight=1)
site_map.add_child(lines_coast)
"""

distance_marker_city = folium.Marker(
    origin_city,
    icon=DivIcon(
        icon_size=(20,20),
        icon_anchor=(0,0),
        html='<div style="font-size: 12; color:#d35400;"><b>%s</b></div>' % "{:.2f} KM".format(distance_city),
        )
    )
site_map.add_child(distance_marker_city)

lines_city = folium.PolyLine(locations=[origin_city, dest_KSC_LC_39A_location_coords], weight=1)
site_map.add_child(lines_city)


distance_marker_railway = folium.Marker(
    origin_railway,
    icon=DivIcon(
        icon_size=(20,20),
        icon_anchor=(0,0),
        html='<div style="font-size: 12; color:#d35400;"><b>%s</b></div>' % "{:.2f} KM".format(distance_railway),
        )
    )
site_map.add_child(distance_marker_railway)

lines_railway = folium.PolyLine(locations=[origin_railway, dest_KSC_LC_39A_location_coords], weight=1)
site_map.add_child(lines_railway)
```



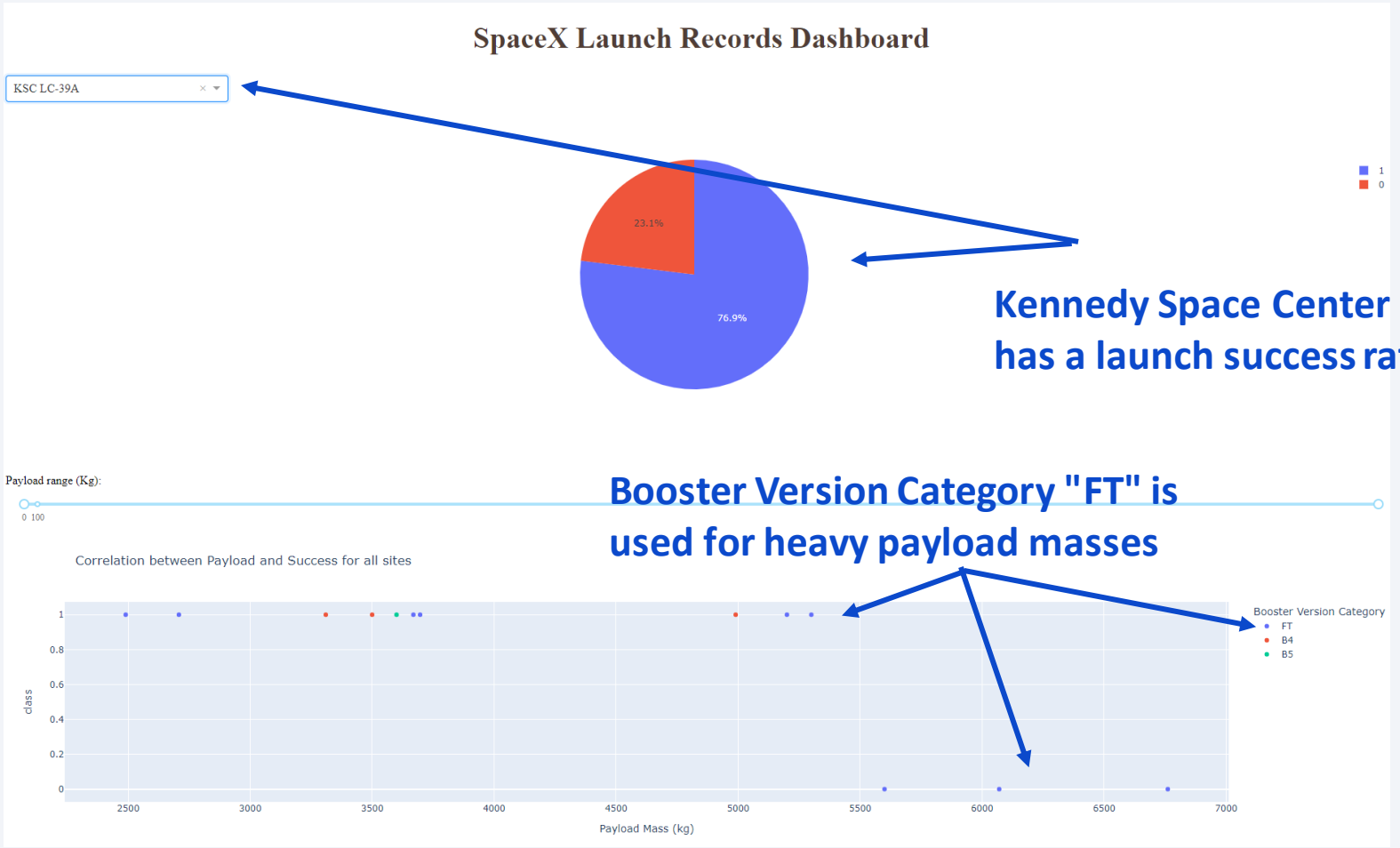
Kennedy Space Center Launch Complex 39


Section 4
Build a Dashboard with Plotly Dash

Launch success count for all sites



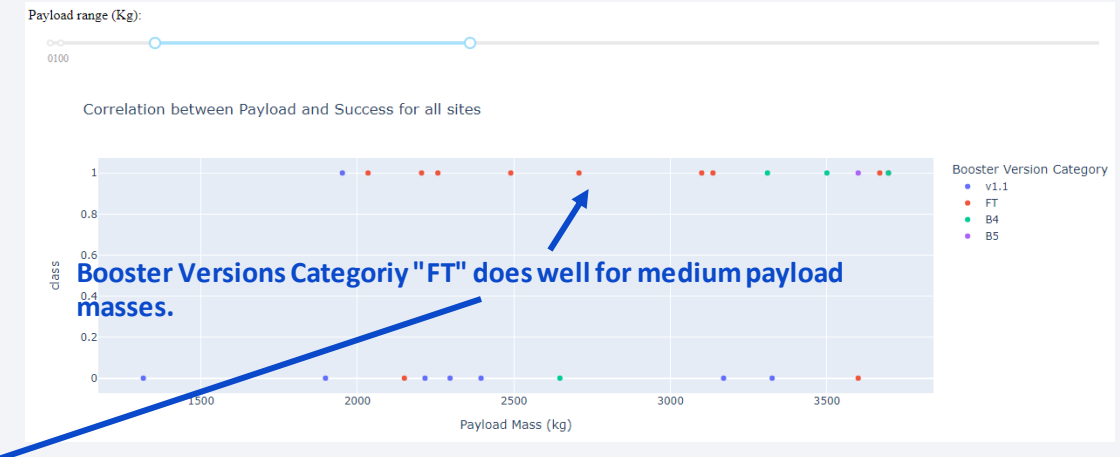
launch site with highest launch success ratio



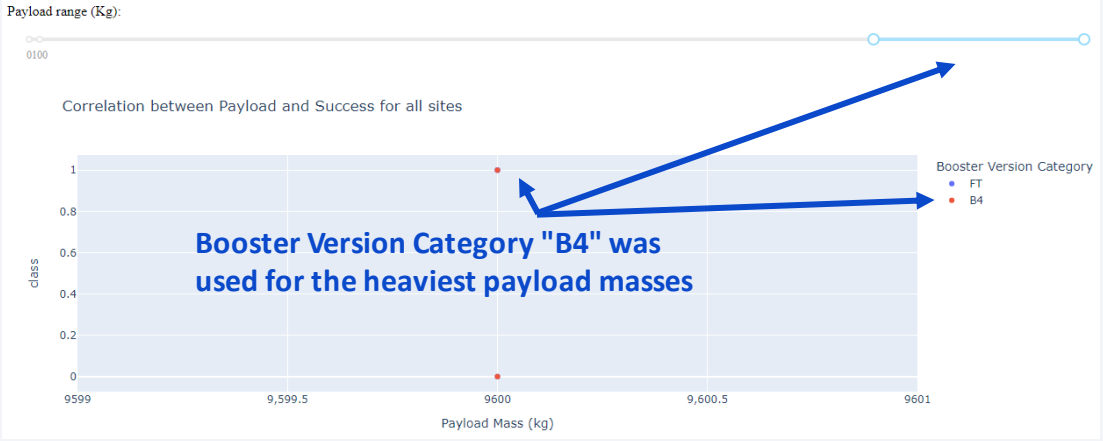
Payload vs. Launch Outcome scatter plot for all sites, different payload selected

All Booster Versions Categories are used for small payload masses.

v1.1 seems to not have done very well.



Booster Versions Categoriy "FT" does well for medium payload masses.

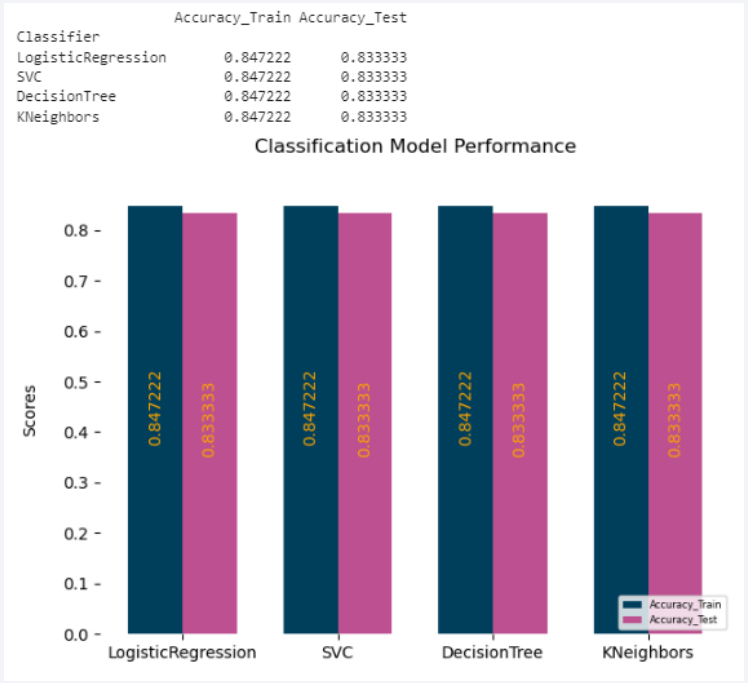


Booster Version Category "B4" was used for the heaviest payload masses

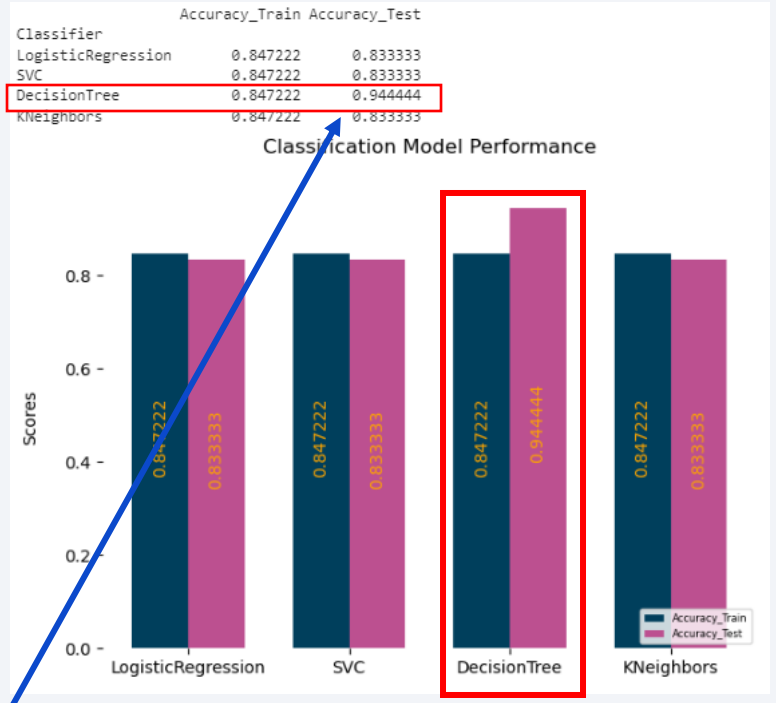
Section 5

Predictive Analysis (Classification)

Classification Accuracy



tree = DecisionTreeClassifier(**random_state=0**)



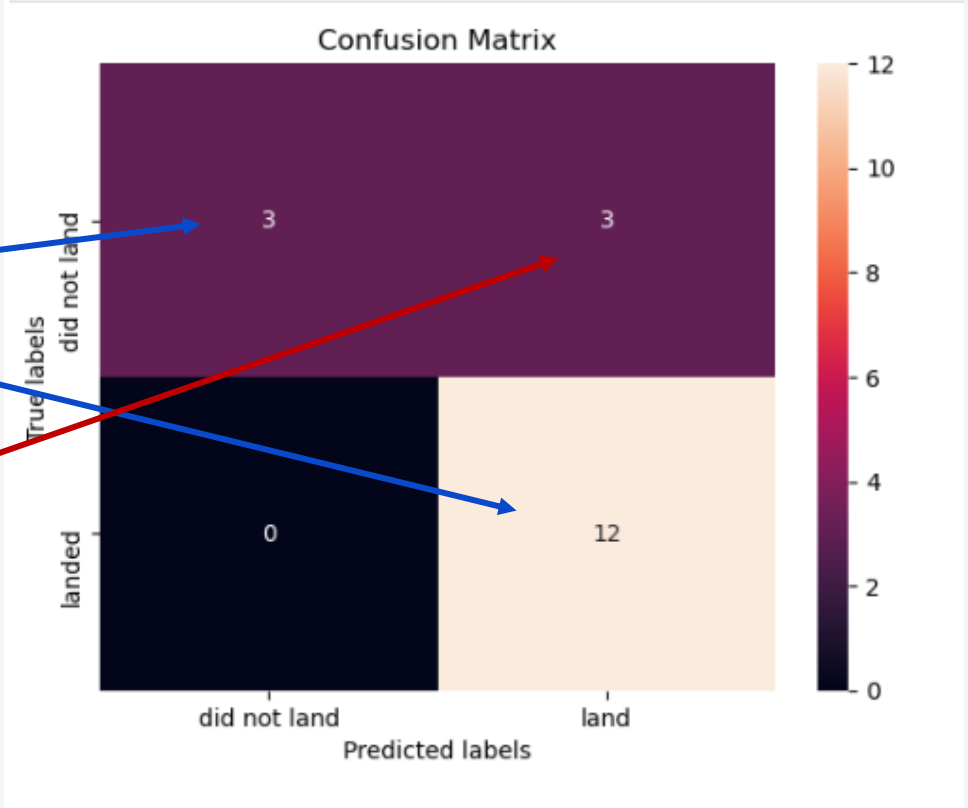
tree = DecisionTreeClassifier()

Without "random_state" the results for DecisionTreeClassifier can vary.
So the decision tree classifier can perform best of all the classifiers.

Confusion Matrix for decision tree classifier

15 outcomes were correctly predicted by the decision tree classifier.

The decision tree classifier slightly overestimated the launch success.



Conclusions

- The success of a rocket launch can be predicted with a 83% success rate by taking into account the launch site, booster version and payload mass and using a decision tree classifier.
- The Kennedy Space Center is a good choice for a successful landing with an overall chance of 76,9% to reuse the first stage of the rocket.
- Depending on the payload mass the correct booster version category improves launch success.

Thank you!