IBM Developer
SKILLS NETWORK

Winning the Space Race with Data Science

OLIVER LINDNER

2022-01-22

Presentation Outline

Section 1

Executive Summary

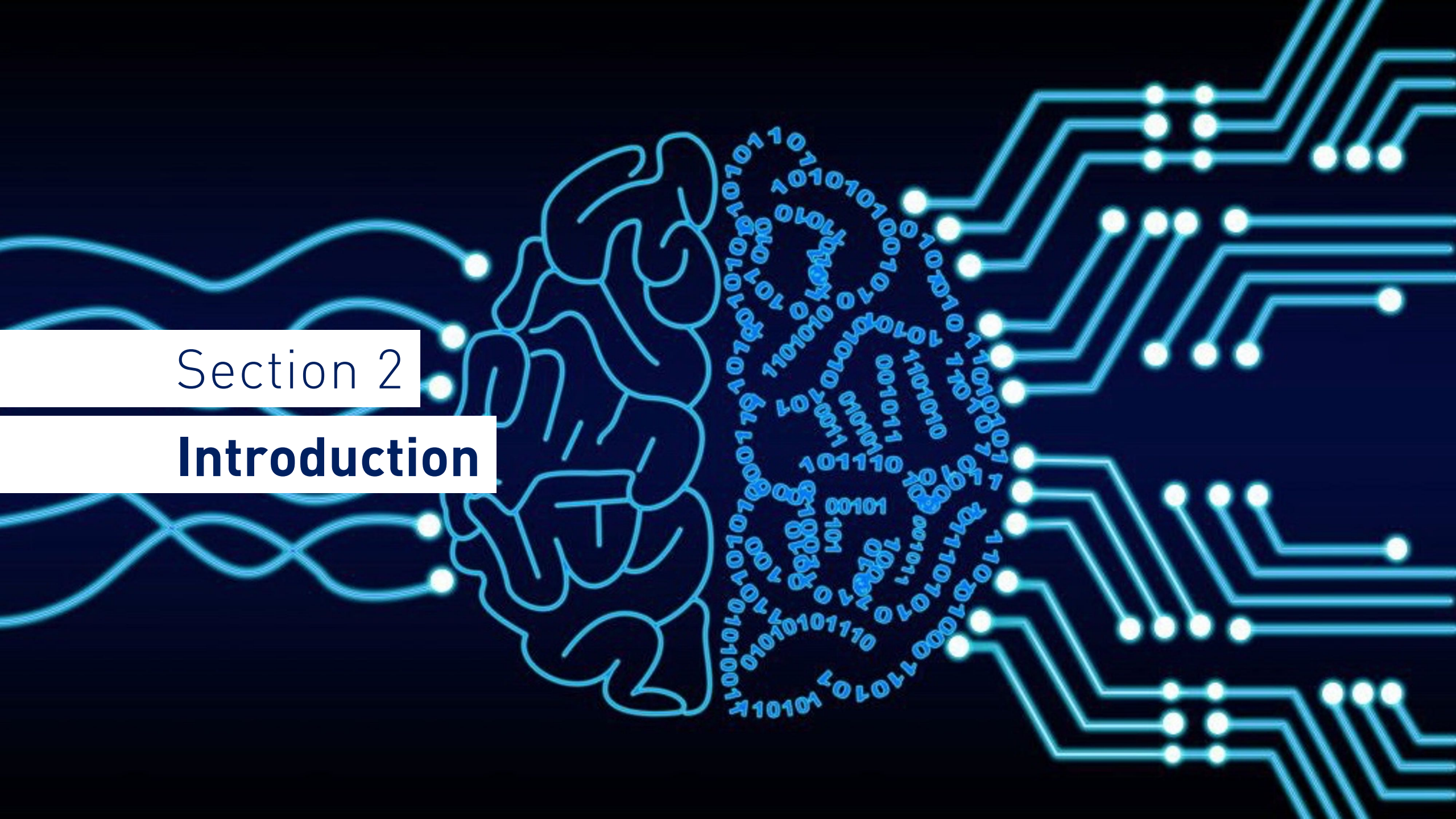
Section 1

Executive Summary

To maintain SpaceYs' competitive advantage, cost of launch missions should be further decreased. As reusability of the first stage is a major factor in mission cost, understanding the correlation of factors resulting in success or failure, and a reliable prediction of the outcome are needed.

Data analysis of Falcon9 launch data from 2010 until 2020 shows a steep learning curve after 2013 as the quota of successful landings of the first stage increased. It was obvious that heavier payloads and certain orbits deliver higher success rates than other options.

From the data, we could generate a "Decision Tree" ML model with a prediction accuracy of 89% that can be used to estimate the true cost of a planned flight based on the likelihood of landing mission outcome and reusability of the first stage component.



Section 2

Introduction

Section 2

Introduction: Project Background and Context

The price for a Falcon9 launch that customers are charged is roughly 62 Mio. US$ while a similar space mission offered by other providers will typically cost at least 165 Mio. US$. The huge gap in pricing is mainly based on the reusability of the first stage of the Falcon9 rocket design. A successful landing of the first stage and the ability to reuse this component in a follow up mission reduces cost enormously.

In the past, not all missions had a successful landing after successfully deploying the payload in its destination orbit. In order to further reduce the cost, the quota of successful landings must be maximised. The factors that influence success of a landing need to be analysed in order to optimize parameters for future missions and guarantee cost reduction that will allow SpaceY to maintain its competitive advantage.

Section 2

Introduction: Problems We Want to Find Answers

Factors for Successful Landing

As the main variable component for mission cost is reusability of the first stage after a successful landing, we need to identify the factors or combination of factors that influence whether a landing attempt of the first stage will be successful.

If correlations can be identified, we can than tackle the question of how to leverage this knowledge for future missions. We need to avoid unsafe combinations and adjust parameters for a intended mission. This will improve overall success rate.

Reliable Prediction

To calculate the true cost of a mission including the cost risk from the quota of reusability of the first stage component, we want to create an algorithm to predict if first stage will land successfully based on anticipated mission parameters such as launch site, payload, destination orbit, and booster type.

The classification model needs to be reliable enough and easy to use so that it can be employed in the offer creation and mission preparation process by laymen with confidence.

Section 3

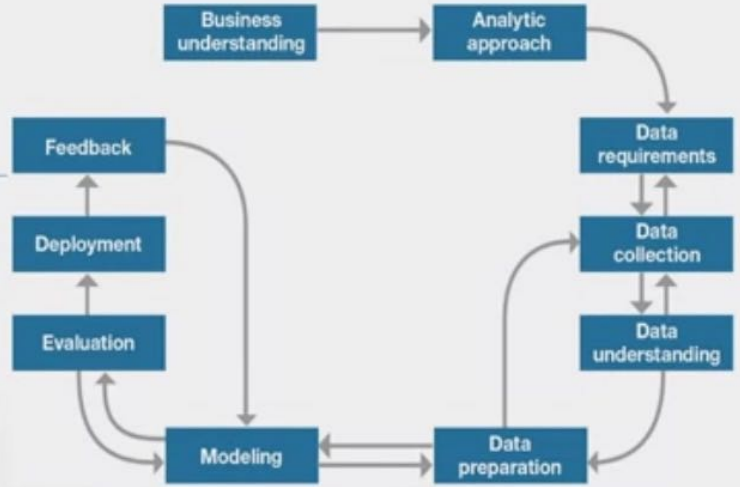
Methodology

Section 3

Methodology

COLLECTION	As a first step, data was collected from two sources. SpaxeX provides a Rest API endpoint to access launch records, while a process called Web Scraping was used to read Falcon9 data from the Wikipedia website.
WRANGLING	The data was aggregated, validated, and cleansed for further analysis steps. One-hot encoding was applied for categorical attributes and values normalized to prepare the data set for machine learning.
EXPLORATION	The data sets were explored using various visualization techniques (scatterplots, charts) to find and illustrates relationsships between variables and show patterns. SQL queries were run to obtain quantitave and qualitative insights into the data set. Advanced graphics libraries such as Folium and Plotly Dash were employed to obtain additional graphical evaluations.
PREDICTION	Suitable classification models were selected, trained, tested, tuned, and evaluated. The best performing model was identified.

The methodology in a nutshell...



The **Data Science Methodology** aims to answer the following 10 questions in this prescribed sequence:

From problem to approach:

1. *What is the problem that you are trying to solve?*
2. *How can you use data to answer the question?*

Working with the data:

3. *What data do you need to answer the question?*
4. *Where is the data coming from (identify all sources) and how will you get it?*
5. *Is the data that you collected representative of the problem to be solved?*
6. *What additional work is required to manipulate and work with the data?*

Deriving the answer:

7. *In what way can the data be visualized to get to the answer that is required?*
8. *Does the model used really answer the initial question or does it need to be adjusted?*
9. *Can you put the model into practice?*
10. *Can you get constructive feedback into answering the question?*

Source: IBM Developer Skills Network

Analytic Approach to Machine Learning

If the question is to determine probabilities of an action
- Use a Predictive model

If the question is to show relationships
- Use a descriptive model

If the question requires a yes/no answer
- Use a classification model

Analytic approach
- *How can you use data to answer the question?*

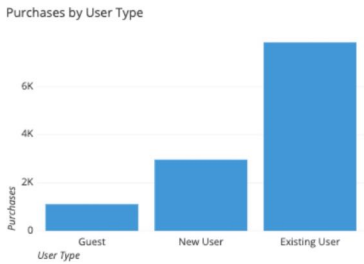
- The correct approach depends on business requirements for the model

Source: IBM Developer Skills Network

Examples of Basic Chart Types

What is a bar chart?

A bar chart (aka bar graph, column chart) plots numeric values for levels of a categorical feature as bars. Levels are plotted on one chart axis, and values are plotted on the other axis. Each categorical value claims one bar, and the length of each bar corresponds to the bar's value. Bars are plotted on a common baseline to allow for easy comparison of values.

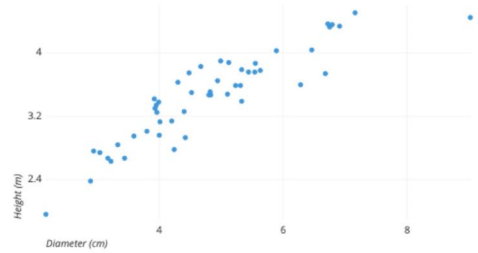


What is a line chart?

A line chart (aka line plot, line graph) uses points connected by line segments from left to right to demonstrate changes in value. The horizontal axis depicts a continuous progression, often that of time, while the vertical axis reports values for a metric of interest across that progression.

What is a scatter plot?

A scatter plot (aka scatter chart, scatter graph) uses dots to represent values for two different numeric variables. The position of each dot on the horizontal and vertical axis indicates values for an individual data point. Scatter plots are used to observe relationships between variables.



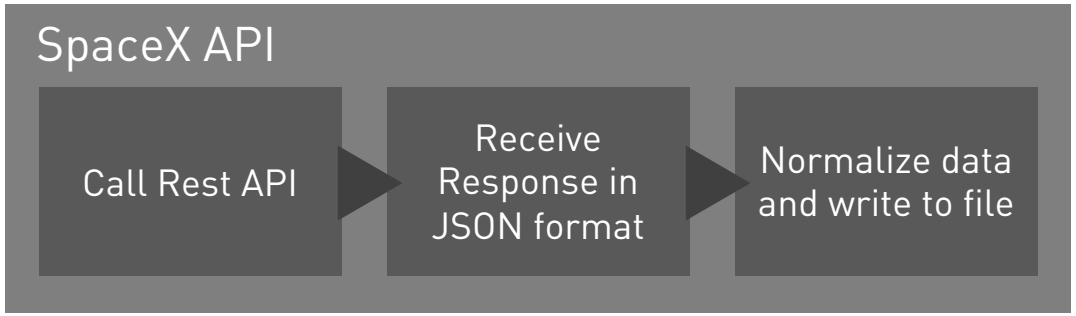
Source: https://chartio.com

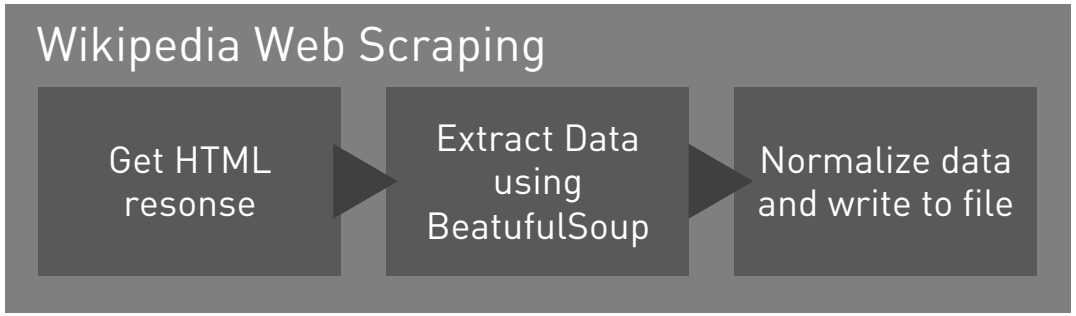
Data Collection

Two data sources were identified for data collection: (1) SpaceX Launch Data and (2) Wikipedia page on Falcon9 launch events.

SpaceX provides an API endpoint, so RESTful API calls were used to obtain data from this source. This data contains information on launches, booster used, payload mass, target orbit, additional information on the launch event, landing specifics, and the landing outcome.

The information from the Wikipedia website was obtained using a method called web scraping where the HTML web page is processes using the BeautifulSoup library. We were parsing the page data for information on Falcon9 launch event data in table format.





Data Collection – SpaceX API

1 Query SpaceX Web API Endpoint

```
In [6]: spacex_url="https://api.spacexdata.com/v4/launches/past"

In [7]: response = requests.get(spacex_url)
```

2 Convert to JSON and assign to Data Frame

```
In [11]: # Use json_normalize meethod to convert the json result into a dataframe
response_json = response.json()
df = pd.json_normalize(response_json)
```

3 Apply Custom Functions to Cleanse Data Frame

```
In [16]: # Call getBoosterVersion
getBoosterVersion(data)

In [18]: # Call getLaunchSite
getLaunchSite(data)

In [19]: # Call getPayloadData
getPayloadData(data)

In [20]: # Call getCoreData
getCoreData(data)

In [21]: launch_dict = {'FlightNumber': list(data['flight_number']),
'Date': list(data['date']),
'BoosterVersion':BoosterVersion,
'PayloadMass':PayloadMass,
'Orbit':Orbit,
'LaunchSite':LaunchSite,
'Outcome':Outcome,
'Flights':Flights,
'GridFins':GridFins,
'Reused':Reused,
'Legs':Legs,
'LandingPad':LandingPad,
'Block':Block,
'ReusedCount':ReusedCount,
'Serial':Serial,
'Longitude': Longitude,
'Latitude': Latitude}
```

4 Assign to (new) DF, Filter for “Falcon 9”, Renumber

```
In [22]: # Create a data from launch_dict
data_falcon9 = pd.DataFrame.from_dict(launch_dict)

In [24]: # Hint data['BoosterVersion']!='Falcon 1'
data_falcon9.drop(data_falcon9[ data_falcon9['BoosterVersion'] == 'Falcon 1' ].index, inplace=True)

In [25]: data_falcon9.loc[:,'FlightNumber'] = list(range(1, data_falcon9.shape[0]+1))
```

5 Replace empty Fields with Average Values

```
In [27]: # Calculate the mean value of PayloadMass column
mean_PayloadMass = data_falcon9['PayloadMass'].mean()

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

CSV dataset_part_1.csv

GitHub https://github.com/github-linaugs/data-science/blob/8f8d3d2f86d3fcf48cc1f4d7b22e302bd0cb86da/1%20-%20Data%20Collection%20API%20(Notebook).ipynb

Data Collection – Web Scraping

1 Query Wikipedia Website & Process Response

```
In [4]: static_url = "https://en.wikipedia.org/w/index.php?title=List_of_Falcon_9_and_Falcon_Heavy_launches&oldid=1027686922"
```

```
In [5]: # use requests.get() method with the provided static_url
        # assign the response to a object

        data = requests.get(static_url).text
```

2 Use BeautifulSoup for HTML Processing

```
In [6]: # Use BeautifulSoup() to create a BeautifulSoup object from a response text content

        soup = BeautifulSoup(data, 'html5lib')
```

```
In [8]: # Use the find_all function in the BeautifulSoup object, with element type `table`
        # Assign the result to a list called `html_tables`

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

```
In [9]: # Let's print the third table and check its content
        first_launch_table = html_tables[2]
        print(first_launch_table)
```

3 Find Data in Table and Append List/Dict Object

```
In [12]: 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
```

```
In [13]: 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
```

4 Convert to Pandas Data Frame

```
In [14]: df=pd.DataFrame(launch_dict)
```

CSV spacex_web_scraped.csv

GitHub https://github.com/github-linaugs/data-science/blob/8f8d3d2f86d3fcf48cc1f4d7b22e302bd0cb86da/2%20-%20Data%20Collection%20with%20Web%20Scraping%20(Notebook).ipynb

Data Wrangling

dataset_part_1.csv

1 Read CSV File into Pandas Data Frame

```
In [2]: df=pd.read_csv("https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/datasets/dataset_part_1.csv")
        df.head(10)
```

2 Extract Mission Outcome & Classify Failure Events

```
In [8]: 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:

```
In [9]: bad_outcomes=set(landing_outcomes.keys()[[1,3,5,6,7]])
        bad_outcomes

Out[9]: {'False ASDS', 'False Ocean', 'False RTLS', 'None ASDS', 'None None'}
```

3 Create Landing Outcome & Add to Data Frame

```
In [10]: # landing_class = 0 if bad_outcome
         # landing_class = 1 otherwise

         a = list(df['Outcome'])
         for i in range(len(a)):
             if a[i] in bad_outcomes:
                 a[i] = 0
             else:
                 a[i] = 1

         landing_class = a
```

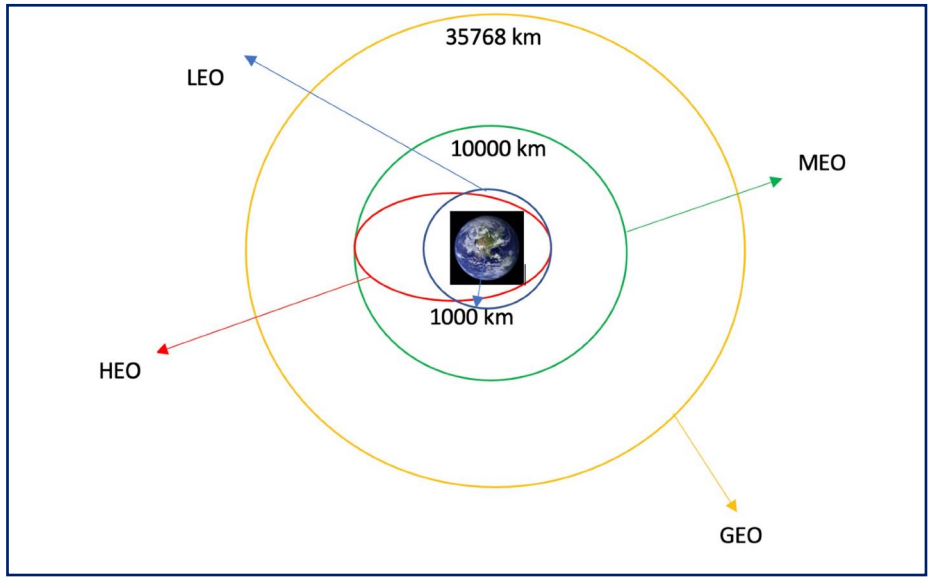
This variable will represent the classification variable that represents the outcome of each launch. If the value is zero, the first stage did not land successfully; one means the first stage landed Successfully

```
In [11]: df['Class']=landing_class
         df[['Class']].head(8)
```

dataset_part_2.csv

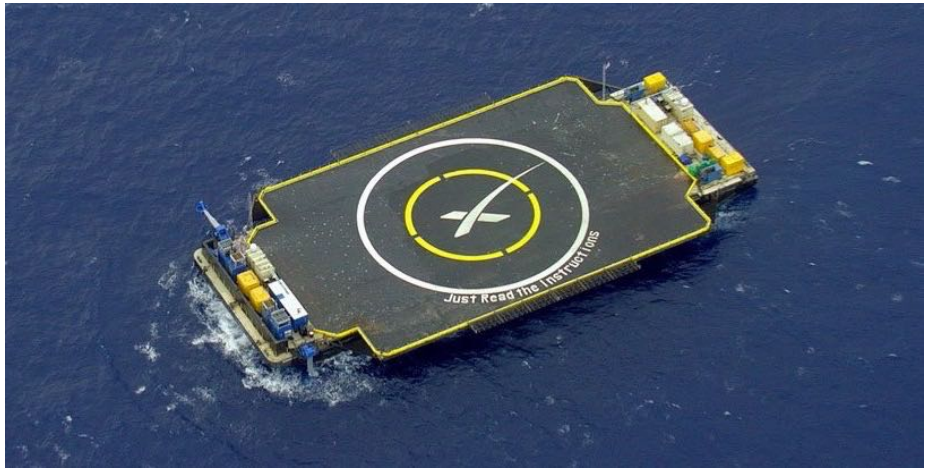
GitHub https://github.com/github-linaugs/data-science/blob/8f8d3d2f86d3fcf48cc1f4d7b22e302bd0cb86da/3%20-%20Data%20Wrangling%20(Notebook).ipynb

Data Wrangling: Attribute Details



These orbits may be targets for flight missions (in alphabetical order):

- ES-L1: Equilibrium relative to the center of mass between the sun and the earth
- GEO: Circular geosynchronous orbit at 35,786 kilometers
- GTO: Geosynchronous Orbit – altitude 35,786 kilometers
- HEO: Highly elliptical orbit
- ISS: A modular space station (habitable artificial satellite) in low Earth orbit
- LEO: Low Earth orbit - altitude of 2,000 km or less
- MEO: Geocentric orbits ranging in altitude from 2,000 km to just below geosynchronous orbit
- PO: Above or nearly above both poles of the body being orbited
- SSO (or SO): Sun-synchronous orbit
- VLEO: Very Low Earth Orbits - altitude below 450 km



The following values are logged in launch records for landing attempts:

- True Ocean: the mission outcome was successfully landed to a specific region of the ocean
- False Ocean: the mission outcome was unsuccessfully landed to a specific region of ocean.
- True RTLS: the mission outcome was successfully landed to a ground pad
- False RTLS: the mission outcome was unsuccessfully landed to a ground pad.
- True ASDS: the mission outcome was successfully landed on a drone ship
- False ASDS: the mission outcome was unsuccessfully landed on a drone ship.

Exploratory Data Analysis (EDA) with SQL

The following queries were run against *DB2 on Cloud* instance using SQL statements to analyze the data loaded into the SPACEXTBL table:

- Display the names of the unique launch sites in the space mission
- Display 5 records where launch sites begin with the string 'CCA'
- Display the total payload mass carried by boosters launched by NASA (CRS)
- Display average payload mass carried by booster version F9 v1.1
- List the date when the first successful landing outcome in ground pad was achieved
- List the names of the boosters which have success in drone ship and have payload mass greater than 4000 but less than 6000
- List the total number of successful and failure mission outcomes
- List the names of the booster_versions which have carried the maximum payload mass
- List the failed landing_outcomes in drone ship, their booster versions, and launch site names for in year 2015
- Rank the count of landing outcomes (such as 'Failure (drone ship)' or 'Success (ground pad)') between the date 2010-06-04 and 2017-03-20, in descending order

GitHub https://github.com/github-linaugs/data-science/blob/dcb937568bb5aa49cc1636e29a259b4b9fad6d45/4%20-%20Data%20Wrangling%20using%20SQL%20(Notebook).ipynb

Exploratory Data Analysis (EDA) with Data Visualization

To analyze the existing data, correlations were depicted in graphical representations. The type of graph was selected to provide the most insight and ease of understanding.

Scatter plots visualize correlations between two variables, including non-linear relationships:

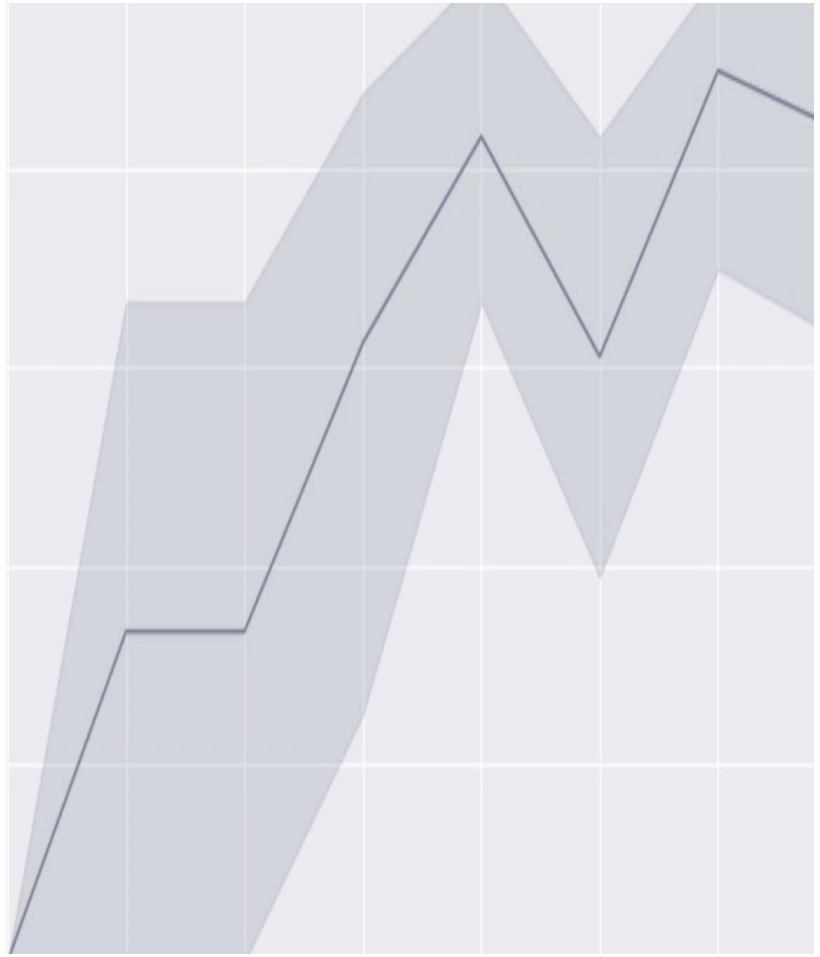
- Flight Number vs. Payload
- Flight Number vs. Launch Site
- Orbit vs. Flight Number
- Payload vs. Launch Site
- Flight Number vs. Orbit Type
- Payload vs. Orbit Type

Bar graphs allow direct comparison between categories:

- Success Rate of Orbit Type

Line graphs indicate performance over time and show trends:

- (Average) Success Rate per Year



GitHub https://github.com/github-linaugs/data-science/blob/83e2629c3e99c4655dff32b673d8372461915803/5%20-%20Visualization%20lab%20(Notebook).ipynb

Build an Interactive Map with Folium

To answer questions regarding location of existing launch sites across the globe, visualize their success rate, and show distance to landmarks, a map display was used for analysis and demonstration of findings.

Circles were added to this zoomable map for all launch locations; for each location all launch events were also marked with color-coded labels showing success (green flag) or failure (red flag) for missions for easy visual analysis of site success rate. To measure distance from landmarks, point markers based on their latitude/longitude were placed, guiding lines and measured distance values added.

About Folium

Folium is a powerful Python library that helps create several types of Leaflet maps. It builds on the data wrangling strengths of the Python ecosystem and the mapping strengths of the Leaflet.js library. Using Folium, data in Python can be manipulated, then visualize it in a Leaflet map.
Folium enables generation of a base map of specified width and height with either default tilesets (i.e., map styles) or a custom tileset URL.

The following tilesets are available by default:

- OpenStreetMap
- Mapbox Bright
- Mapbox Control Room
- Stamen (incl. Terrain, Toner, and Watercolor)
- Cloudmade
- Mapbox
- CartoDB (incl. positron and dark_matter)

https://python-visualization.github.io/folium/

GitHub https://github.com/github-linaugs/data-science/blob/2eb7b834dff42bc9bfcfecf71249599099943519/6%20-%20Interactive%20Visual%20Analytics%20(Notebook).ipynb

Build a Dashboard with Plotly Dash

An interactive dashboard application was created using the Plotly Dash library functions to show success rates for sites and the correlation between payload mass and success.
To show the relative distribution over all sites and individual sites, a pie chart type display was selected, whereas the relationship between payload and location is visualized using a scatter plot which is the best method to visualize non-linear patterns.

The interaction allows interested users to perform their own analytics and to validate the findings presented in this presentation.

About Plotly Dash

Dash is a low-code framework for rapidly building data apps. Written on top of Plotly.js and React.js, Dash is ideal for building and deploying data apps with customized user interfaces. It's particularly suited for anyone who works with data.
Through a couple of simple patterns, Dash abstracts away all of the technologies and protocols that are required to build a full-stack web app with interactive data visualization.
Dash is simple enough that you can bind a user interface to your code in less than 10 minutes.
Dash apps are rendered in the web browser. You can deploy your apps to VMs or Kubernetes clusters and then share them through URLs. Since Dash apps are viewed in the web browser, Dash is inherently cross-platform and mobile ready.
Dash is an open source library released under the permissive MIT license. Plotly develops Dash and also offers a platform for writing and deploying Dash apps in an enterprise environment.

https://dash.plotly.com

GitHub https://github.com/github-linaugs/data-science/blob/97a6ec63066748cc4a834c4a593f9a0c8ec08dca/spacex_dash_app.py

Predictive Analytics (Classification)

1 Build

Build Classification Models

- Load Dataset from CSV File using NumPy and Pandas library functions
- Transform Data (Standardize)
- Split Data into Training and Test Sets
- Check Number of Records in Test Set (Sample Size)
- Select suitable machine learning algorithms
- Train each selected machine learning algorithm (model) with training set data

2 Evaluate

Evaluate the Models

- Use GridSearchCV functions best_score_ and best_params_ to find the optimal hyperparameters for each model
- Use Confusion Matrix to quickly validate reliability of model

3 Improve

Improve the Models

- Tune hyperparameters of each model according to suggestions from GridSearchCV (feature engineering and algorithm tuning)
- Check accuracy of each model with optimized parameters using test set data by calculating score value using test data

4 Select

Select the Best Performing Classification Model

- Compare all score values that were calculated for the models

GitHub https://github.com/github-linaugs/data-science/blob/dcb937568bb5aa49cc1636e29a259b4b9fad6d45/7%20-%20Machine%20Learning%20Prediction%20(Notebook).ipynb

Section 4

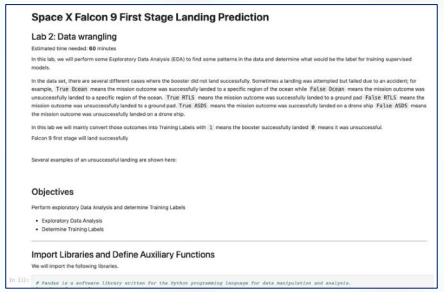
Results

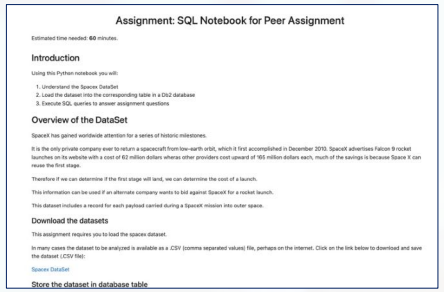
Section 4

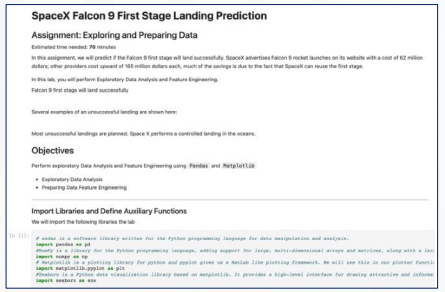
Results

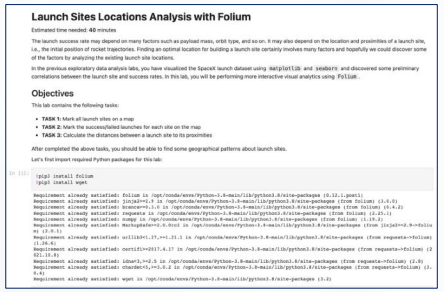
The results presented as screenshots on the followind slides in this section are taken from the various exploratory data analysis steps, graphical evaluations of data including map views, an interactive dashboard application created for this project, and the results of machine learning with various models.

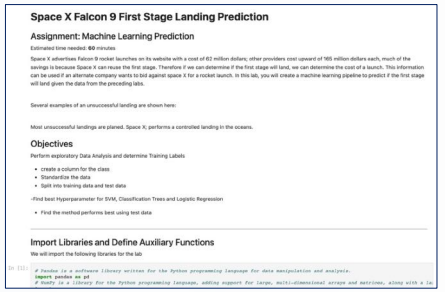
Links to the underlying Jupyter notebooks were provided in Section 3: Methodology for readers to check on the details of the process steps taken in the analysis performed.







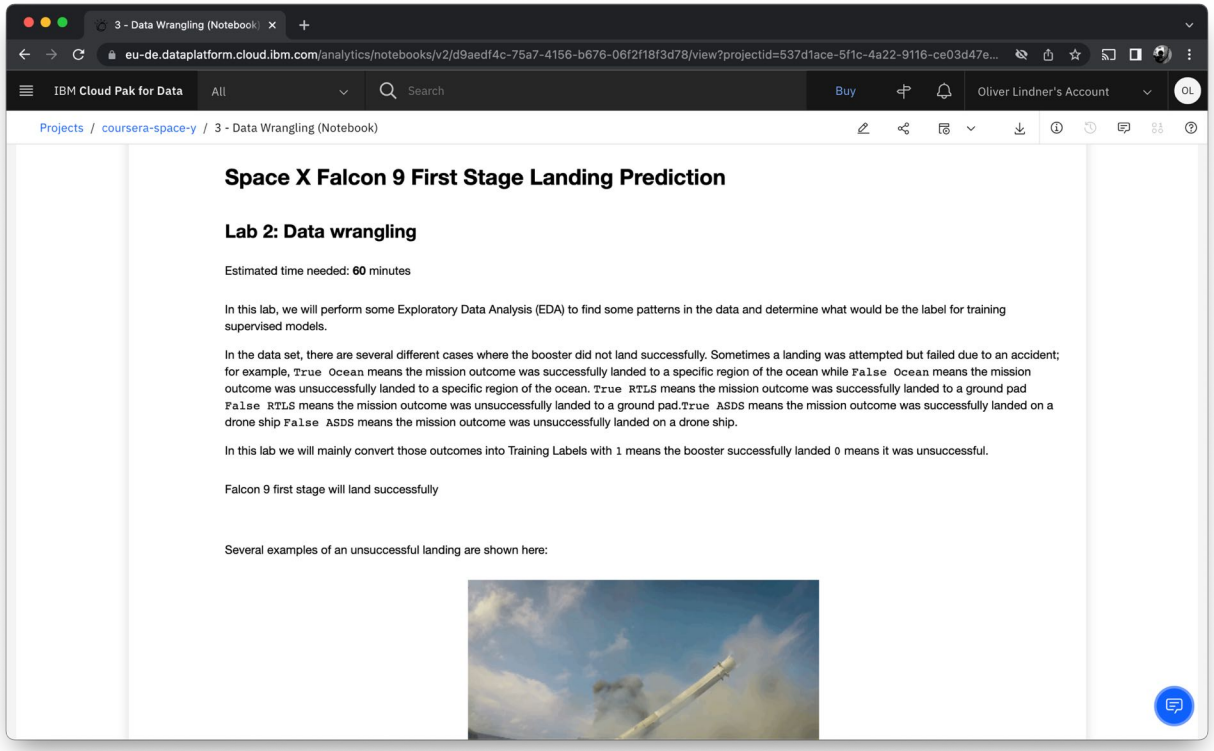




Insights drawn from Exploratory Data Analysis (EDA)

To explore the existing data, various Python libraries, such as Pandas and Numpy, was used on CSV files as well SQL queries run against a DB2 on Cloud database. All steps were performed and documented using Jupyter notebooks.

The individual queries and results are presented in the following slides including conclusions on the results of the analysis.



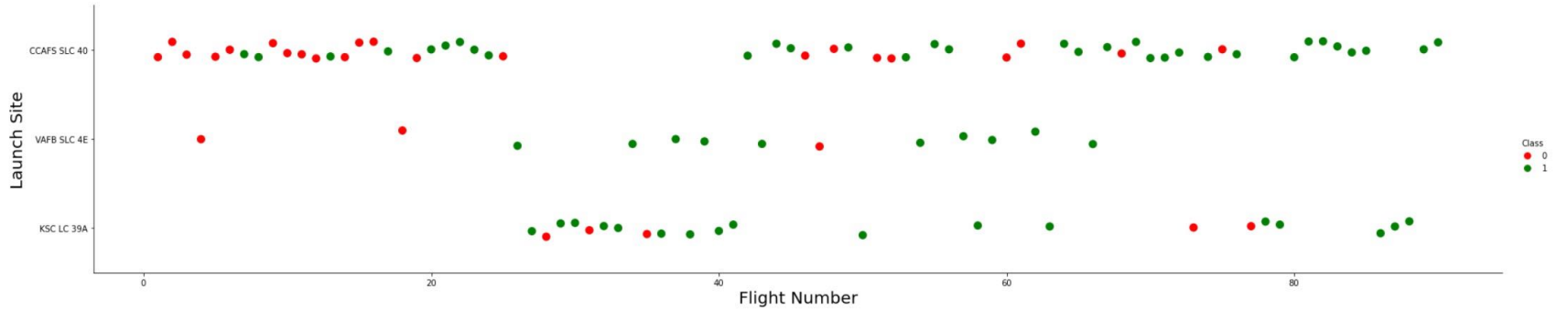
Insights drawn from EDA

Scatterplot of **Flight Number vs. Launch Site** Correlation

Did success quota improve over time?

Green dots (Class 1) show successful launch events, while red (Class 0) indicates unsuccessful mission.

The success rate has increased over time, with a significant improvement obvious after flight 20 indicating a steep learning curve.



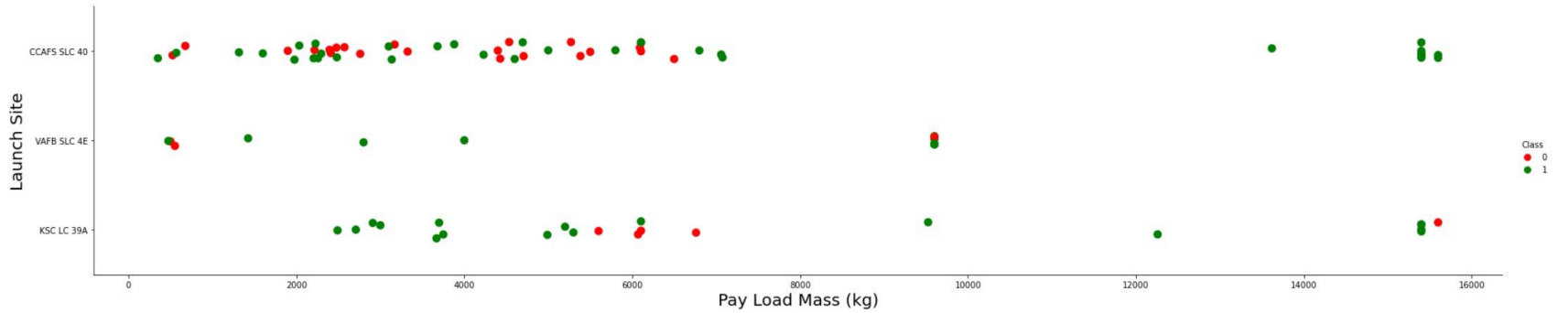
Insights drawn from EDA

Scatterplot of **Payload vs. Launch Site** Correlation

Do payload and/or launch site impact success rate?

Green dots (Class 1) show successful launch events, while red (Class 0) indicates unsuccessful mission.

Although there is no clear relation obvious, it seems that higher payload perform better and result in a higher rate of success.



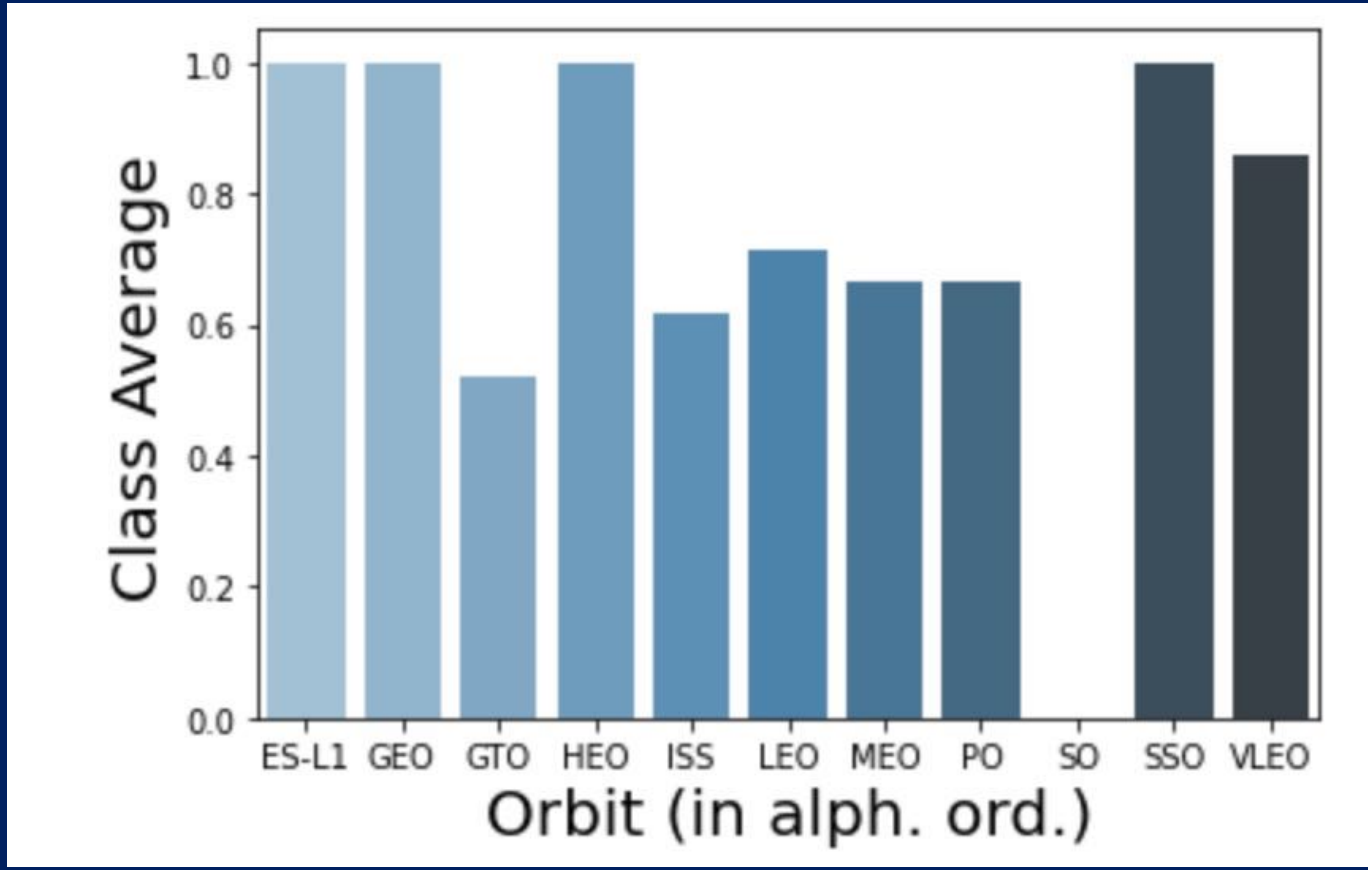
Insights drawn from EDA

Bar Chart of **Success Rate vs. Orbit Type** Correlation

Are certain orbits more successful than others?

The individual bars indicate the success rate average for all missions to that orbit.

We can see that ES-L1, GEO,HEO, and SSO have success rates of 100%.



Section 4: Results

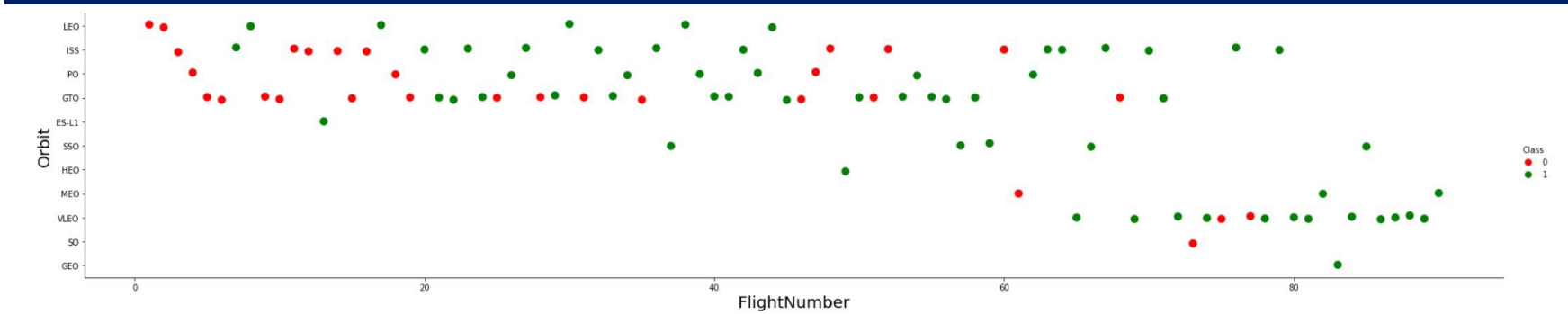
Insights drawn from EDA

Scatterplot of **Flight number vs. Orbit type** Correlation

Did success rate for individual orbits improve over time?

Green dots (Class 1) show successful launch events, while red (Class 0) indicates unsuccessful mission.

We can observe a learning curve for orbit LEO but cannot establish a similar observation for other orbits.



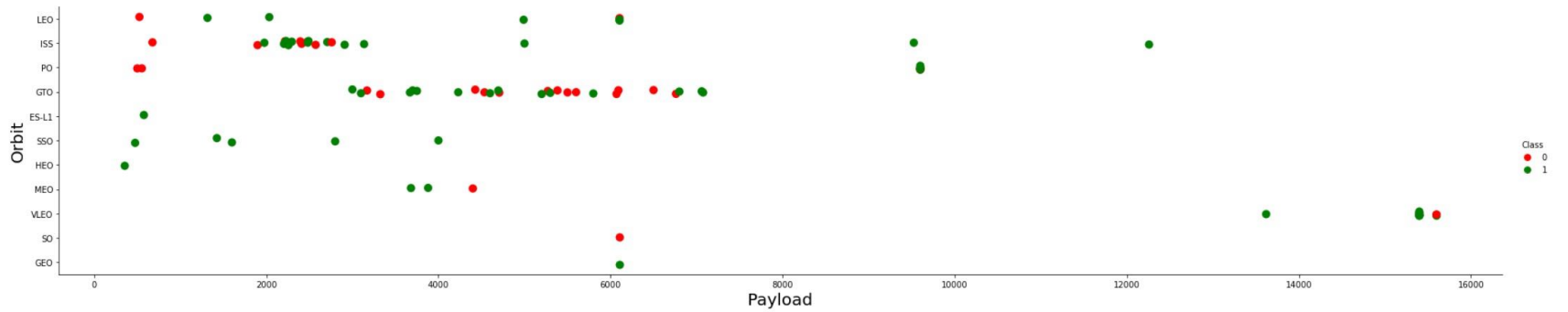
Insights drawn from EDA

Scatterplot of **Payload vs. Orbit Type** Correlation

Do payload and orbit type influence success rate?

Green dots (Class 1) show successful launch events, while red (Class 0) indicates unsuccessful mission.

There is no clear relationship between these variables and success visible as orbits do not show uniform results.

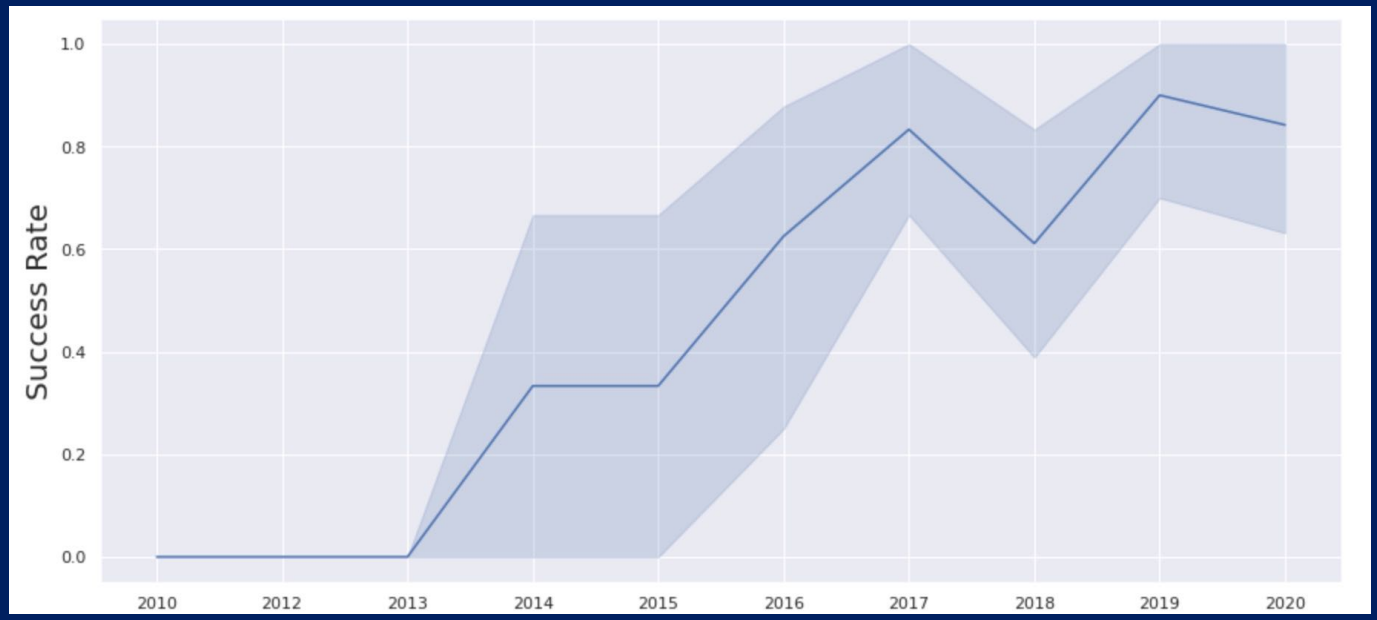


Insights drawn from EDA

Line Chart **Launch Success Yearly Trend** showing yearly average success rate

Are we showing progress regarding success rate from year to year?

The success rate has improved considerably since 2013, with a slight “dent” in 2018.



Insights drawn from EDA

Tabular list of **All Launch Site Names** (unique sites)

How many sites are in use and what is their name?

The SQL query is using the DISTINCT keyword to find unique items in column LAUNCH_SITES.

Currently, there are four launch sites.

```
In [7]: %sql select unique(LAUNCH_SITE) from SPACEXTBL;
 * ibm_db_sa://xwx41301:***@3883e7e4-18f5-4afe-be8c-fa31c41761d2.bs2io90l08kqb1od8lcg.databases.appdomain.cloud:31498/bludb
Done.
Out[7]:
```

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

Insights drawn from EDA

Tabular list **Launch Site Names Begin with 'CCA'** showing 5 records where data in launch_site attribute field starts with characters CCA

The SQL query is using a WHERE clause with the LIKE operator and a percent sign in the search token while the LIMIT clause restricts the number of results displayed.

```
In [9]: %sql select * from SPACEXTBL where LAUNCH_SITE like 'CCA%' limit 5;
```

```
 * ibm_db_sa://xwx41301:***@3883e7e4-18f5-4afe-be8c-fa31c41761d2.bs2io90l08kqb1od8lcg.databases.appdomain.cloud:31498/bludb
Done.
```

Out[9]:

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

Insights drawn from EDA

Query Result for **Total Payload Mass** of boosters from NASA

What is the sum of all payloads for customer NASA so far?

The SQL query is using the SUM() function and a WHERE clause to restrict results to the desired customer.

The total is shown as 45.596 kg.

```
In [15]: %sql select sum(PAYLOAD_MASS__KG_) as "total payload for NASA (CRS)" from SPACEXTBL where CUSTOMER = 'NASA (CRS)';
 * ibm_db_sa://xwx41301:***@3883e7e4-18f5-4afe-be8c-fa31c41761d2.bs2io90l08kqb1od8lcg.databases.appdomain.cloud:31498/bludb
Done.
Out[15]: total payload for NASA (CRS)
                                  45596
```

Insights drawn from EDA

Query Result for **Average Payload Mass by F9 v1.1**

What is the average payload transported by mentioned booster version?

The SQL query is using the AVG() function and a WHERE clause to restrict results to the desired booster type.

The result is shown as 2.928 kg.

```
In [16]: %sql select avg(PAYLOAD_MASS__KG_) as "average payload for booster F9 v1.1" from SPACEXTBL where BOOSTER_VERSION = 'F9 v1.1';
 * ibm_db_sa://xwx41301:***@3883e7e4-18f5-4afe-be8c-fa31c41761d2.bs2io90l08kqb1od8lcg.databases.appdomain.cloud:31498/bludb
Done.
Out[16]: average payload for booster F9 v1.1
         2928
```

Insights drawn from EDA

Query Result for **First Successful Ground Landing Date** showing the first successful landing outcome on ground pad

When did we successfully perform a ground landing for the first time?

The SQL query is using the MIN() function on the results restricted by a WHERE for the mentioned landing outcome.

The result is shown as December 22nd, 2015.

```
In [22]: %sql select min(DATE) as "first successful landing" from SPACEXTBL where LANDING__OUTCOME = 'Success (ground pad)';
 * ibm_db_sa://xwx41301:***@3883e7e4-18f5-4afe-be8c-fa31c41761d2.bs2io90l08kqb1od8lcg.databases.appdomain.cloud:31498/bludb
Done.
Out[22]: first successful landing
         2015-12-22
```

Insights drawn from EDA

Query Results for **Successful Drone Ship Landing with Payload between 4000 and 6000** showing the names of boosters which have successfully landed on drone ship and had payload mass greater than 4000 but less than 6000

The SQL query is using two WHERE clauses combined with AND keyword, and a value range using BETWEEN...AND... qualifiers.

```
In [23]: %sql select BOOSTER_VERSION from SPACEXTBL where ( LANDING__OUTCOME = 'Success (drone ship)' ) and ( PAYLOAD_MASS__KG_ between 4000 and 60(
 * ibm_db_sa://xwx41301:***@3883e7e4-18f5-4afe-be8c-fa31c41761d2.bs2io90l08kqb1od8lcg.databases.appdomain.cloud:31498/bludb
Done.
Out[23]:
```

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

Insights drawn from EDA

Query Result for **Total Number of Successful and Failure Mission Outcomes** showing the total number of successful and failure mission outcomes

The SQL query is using the COUNT() function on the results restricted by a WHERE clause for wildcard search using percent sign.

99 of the recorded missions were successful.

In [28]:
```
%sql select MISSION_OUTCOME as "Mission Outcome", count(*) as "Count" from SPACEXTBL group by MISSION_OUTCOME order by MISSION_OUTCOME;
```

```
 * ibm_db_sa://xwx41301:***@3883e7e4-18f5-4afe-be8c-fa31c41761d2.bs2io90l08kqb1od8lcg.databases.appdomain.cloud:31498/bludb
Done.
```

Out[28]:

Mission Outcome	Count
Failure (in flight)	1
Success	99
Success (payload status unclear)	1

Insights drawn from EDA

Query Results for **Boosters Carried Maximum Payload** listing the names of the booster which have carried the maximum payload mass

The SQL query is using a subquery which employs the MAX() function.

The highest payloads were carried by F9 B5 B10xx type boosters.

```
In [31]: %sql select distinct BOOSTER_VERSION as "Booster Version" from SPACEXTBL where PAYLOAD_MASS__KG_=( select max(PAYLOAD_MASS__KG_) from SPACE
```

```
 * ibm_db_sa://xwx41301:***@3883e7e4-18f5-4afe-be8c-fa31c41761d2.bs2io90l08kqb1od8lcg.databases.appdomain.cloud:31498/bludb
Done.
```

Out[31]:

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

Insights drawn from EDA

Query Results for **2015 Launch Records** showing the failed landing outcomes in drone ship, corresponding booster versions, and launch site names for the year 2015

The SQL query is using a WHERE clause with two search restrictions and the DATE() function to filter date column on year.

Two failure events were returned, both for launch site CCAFS LC-40.

```
In [35]: %sql select DATE, LANDING__OUTCOME, BOOSTER_VERSION, LAUNCH_SITE from SPACEXTBL where ( extract(YEAR FROM DATE)='2015' ) and ( LANDING__OUT
 * ibm_db_sa://xwx41301:***@3883e7e4-18f5-4afe-be8c-fa31c41761d2.bs2io90l08kqb1od8lcg.databases.appdomain.cloud:31498/bludb
Done.
```

Out[35]:

DATE	landing__outcome	booster_version	launch_site
2015-01-10	Failure (drone ship)	F9 v1.1 B1012	CCAFS LC-40
2015-04-14	Failure (drone ship)	F9 v1.1 B1015	CCAFS LC-40

Insights drawn from EDA

Query Results for **Rank Landing Outcomes Between 2010-06-04 and 2017-03-20** showing the number of landings per outcome type between June 4th, 2010, and March 20th, 2017, in descending order

The SQL query is using a WHERE clause, the GROUP BY function and the ORDER BY DESC option.

As a result, we can see that in most cases a landing was not attempted.

```
In [41]: %sql select LANDING__OUTCOME as "Outcome", count(LANDING__OUTCOME) as "Count (descending)" from SPACEXTBL where DATE between '2010-06-04' a
 * ibm_db_sa://xwx41301:***@3883e7e4-18f5-4afe-be8c-fa31c41761d2.bs2io90l08kqb1od8lcg.databases.appdomain.cloud:31498/bludb
Done.
```

Out[41]:

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

Launch Sites Proximities Analysis

A zoomable map view with markers for sites, launch events for each site, and various landmarks including measurements and lines showing the distance was created for analysis.

The visualization answers questions regarding location, distance to landmarks, and provides visual representation of site success rates.

All sites are located in the US, they are close to coastlines but maintain safe distance to larger cities and airports.

```
# Function to assign color to launch outcome
def assign_marker_color(launch_outcome):
    if launch_outcome == 1:
        return 'green'
    else:
        return 'red'

spacex_df['marker_color'] = spacex_df['class'].apply(assign_marker_color)
spacex_df.tail(10)
```

	Launch Site	Lat	Long	class	marker_color
46	KSC LC-39A	28.573255	-80.646895	1	green
47	KSC LC-39A	28.573255	-80.646895	1	green
48	KSC LC-39A	28.573255	-80.646895	1	green
49	CCAFS SLC-40	28.563197	-80.576820	1	green
50	CCAFS SLC-40	28.563197	-80.576820	1	green

```
# find coordinate of the closet coastline
# e.g.,: Lat: 28.56367  Lon: -80.57163
# distance_coastline = calculate_distance(launch_site_lat, launch_site_lon, coastline_lat, coastline_lon)

launch_site_lat = 28.563197
launch_site_lon = -80.576820
coastline_lat = 28.56334
coastline_lon = -80.56799
distance_coastline = calculate_distance(launch_site_lat, launch_site_lon, coastline_lat, coastline_lon)
print(distance_coastline,' km')
```

```
0.8627671182499878  km
```

```
distance_highway = calculate_distance(launch_site_lat, launch_site_lon, closest_highway[0], closest_highway[1
print('distance_highway =',distance_highway, ' km')
distance_railroad = calculate_distance(launch_site_lat, launch_site_lon, closest_railroad[0], closest_railroa
print('distance_railroad =',distance_railroad, ' km')
distance_city = calculate_distance(launch_site_lat, launch_site_lon, closest_city[0], closest_city[1])
print('distance_city =',distance_city, ' km')
```

```
distance_highway = 0.5834695366934144  km
distance_railroad = 1.2845344718142522  km
distance_city = 51.43416999517233  km
```

Launch Sites Proximities

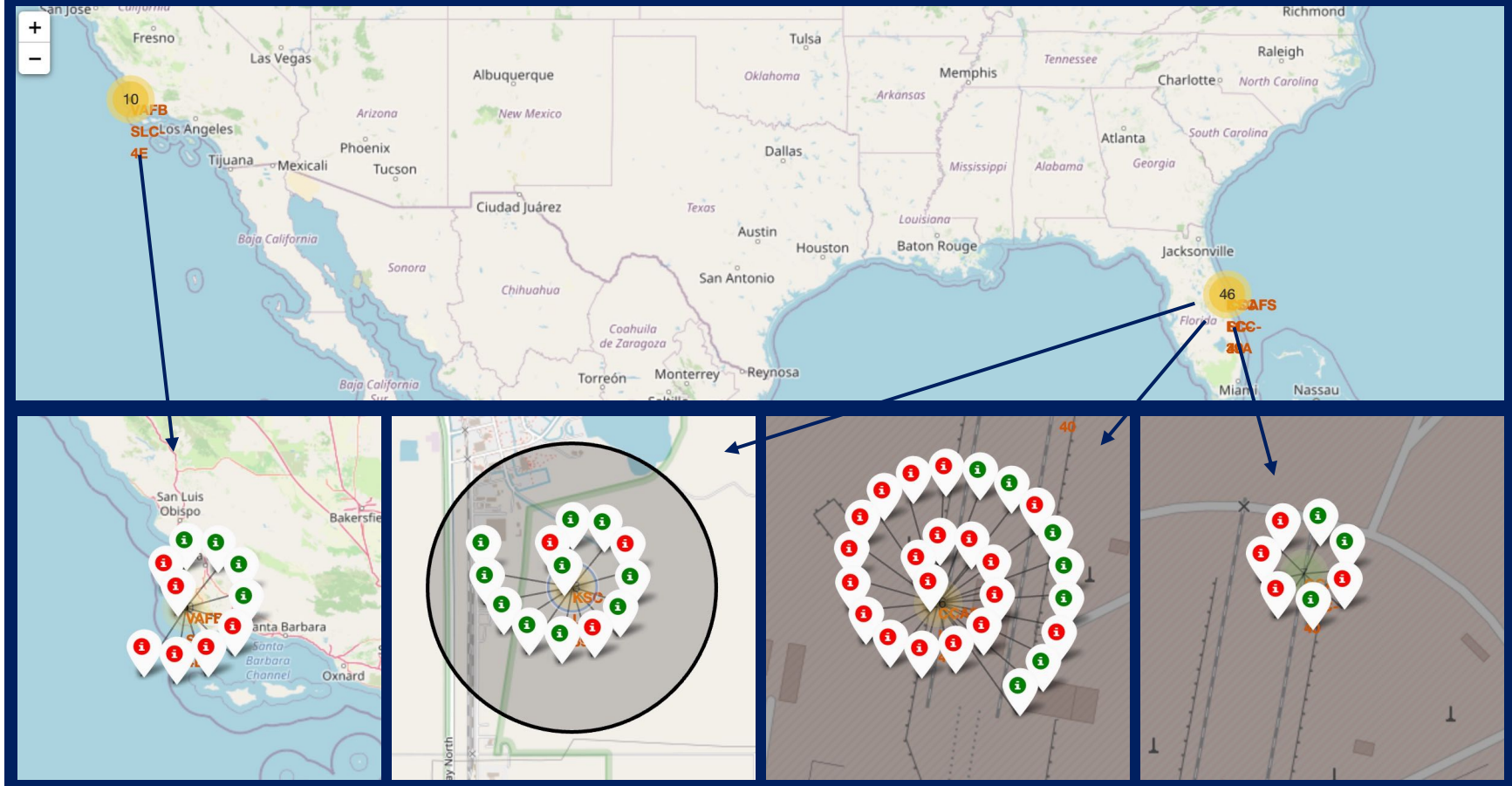
Map display **Launch Site Locations** showing all current sites

Currently, there are four launch sites, all in the US and all close to the coast; one site is in California north of Los Angeles, and three sites are in Florida east of Orlando.

Launch Sites Proximities

Map details for **Launch Outcome Indicators** showing status of launch events per site using color-labeled markers

Green indicates successful launch, red indicates failure. The markers are sorted in the order of flight number for corresponding site (from center showing first launch to outside showing last launch).

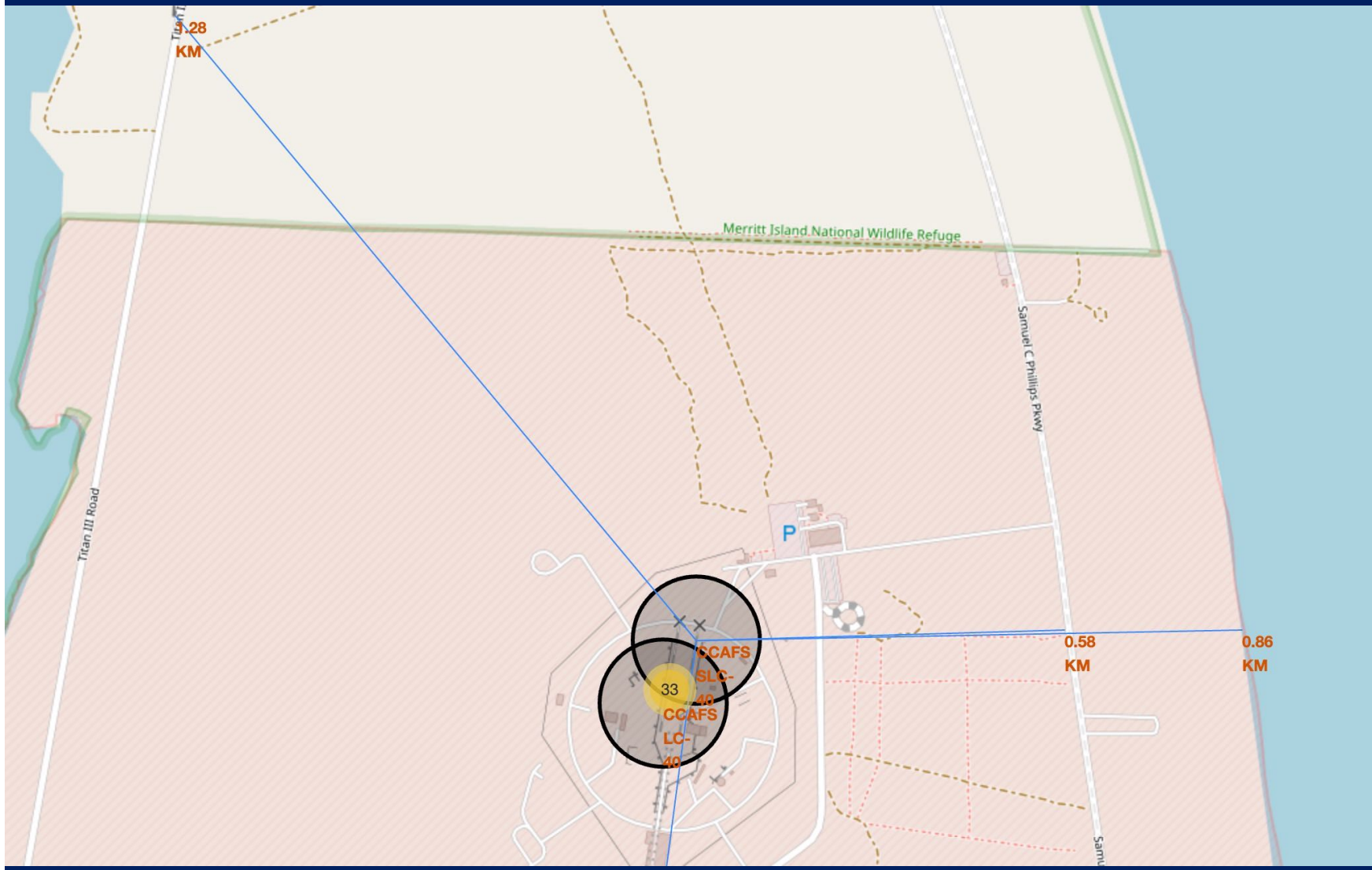


Launch Sites Proximities

Map details showing **Launch Site Distance to Landmarks**

To answer the question whether existing sites are close to railway lines, highways, coastlines, or cities, such points were marked on map, distancse measured, and added to the map view.

Site CAFS SLC-40 shown here is close to coastline (0,86 km) and highway (0,58 km) but keeps safe distance to nearest city and airport (51,43 km to Orlando Melbourne Airport).

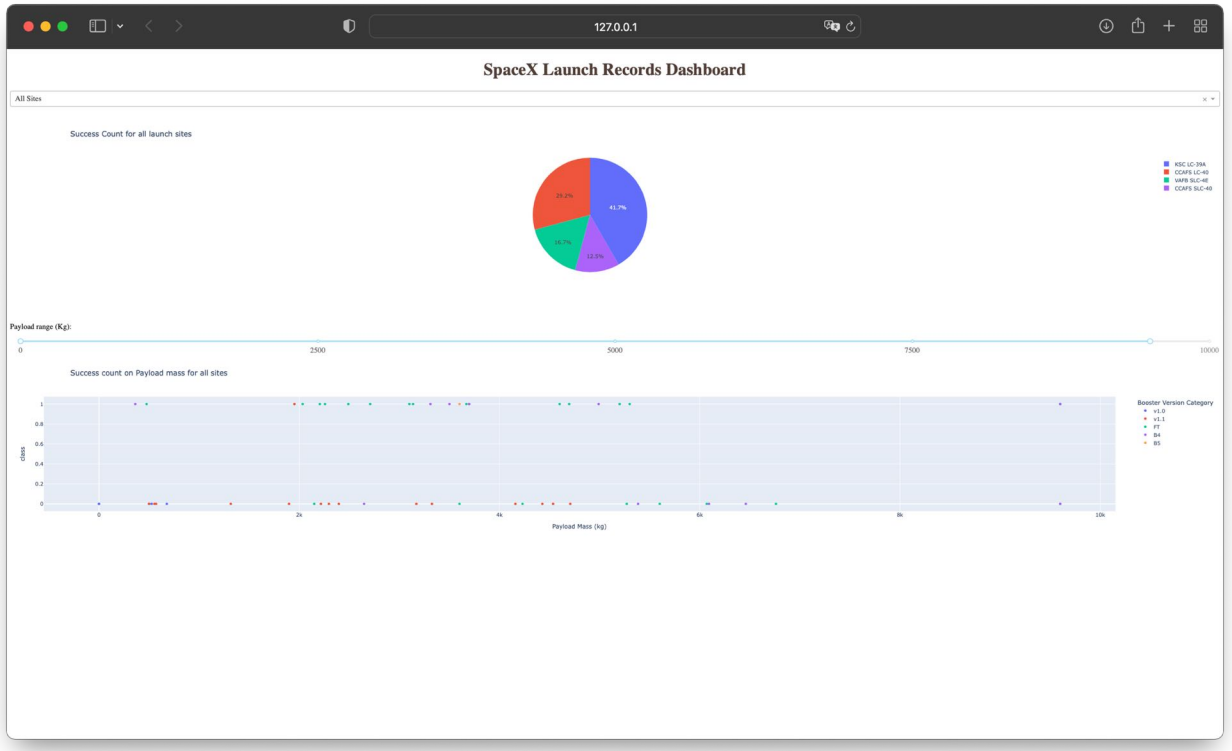


Build a Dashboard with Plotly Dash

As an additional step to explore the data in addition to the method presented in the previous slides, a stand-alone dashboard app was built using the Plotly Dash libraries.

The graphs provide insight into the success rates for specific payloads and the distribution of successful launches for a site and overall.

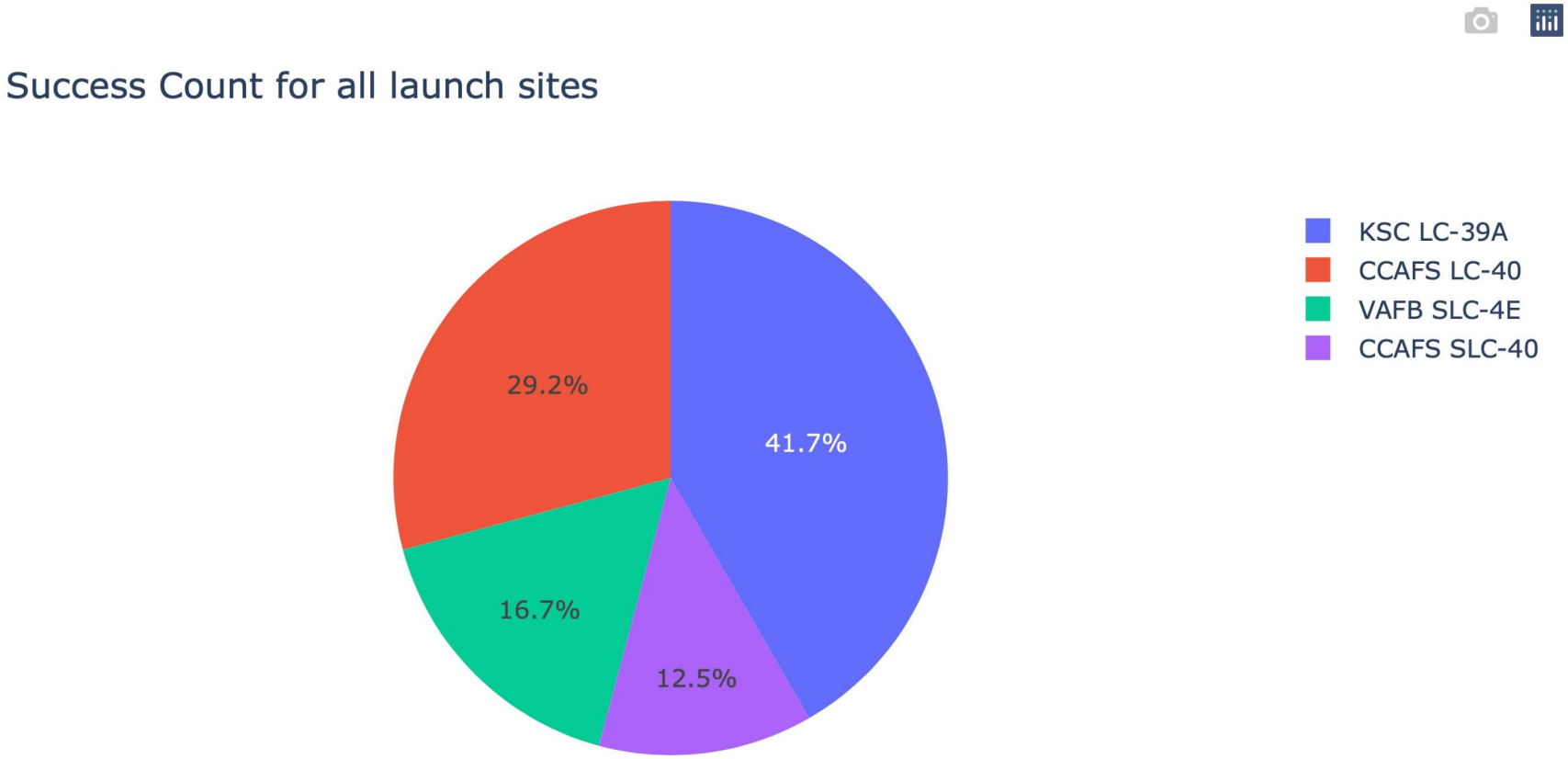
The app is available for other users to perform their own explorations and to validate the steps taken in this project.



Build a Dashboard with Dash

Pie chart for **Success Count for All Launch Sites** shows the percentage of each site in respect to all successful launch events

We can observe that site KSC LC-39A has the highest number of successful launches with a share of 41,7% of the total of successful missions, site VAFB SLC-40 having the lowest share with 12,5%.

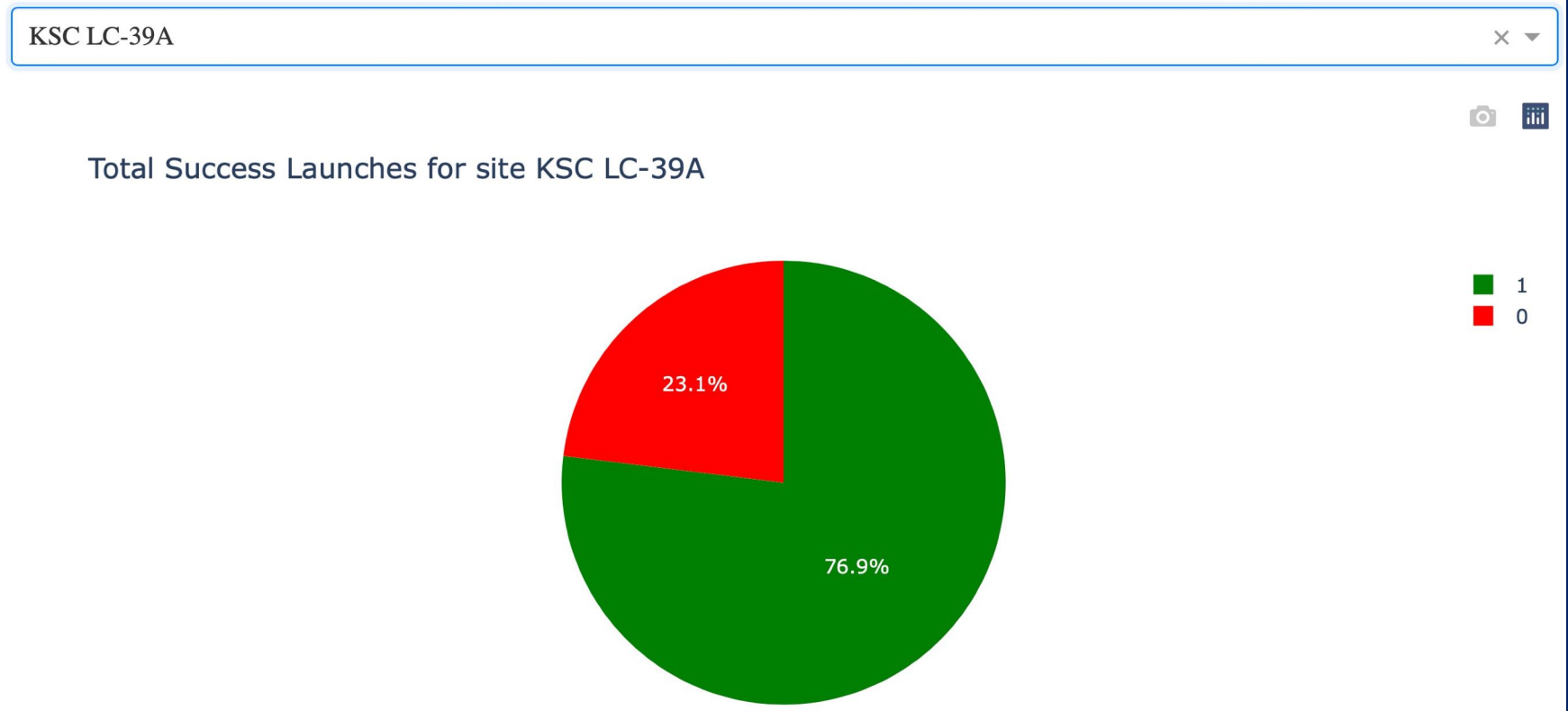


Build a Dashboard with Dash

Pie chart for **Total Succes for site KSC LC-39A** shows the quota of successful vs. failed launch events for this location

This chart can be used to identify the launch site with highest launch success ratio.

The success rate of 76,9% observed for KSC LC-39A is the highest of all sites.



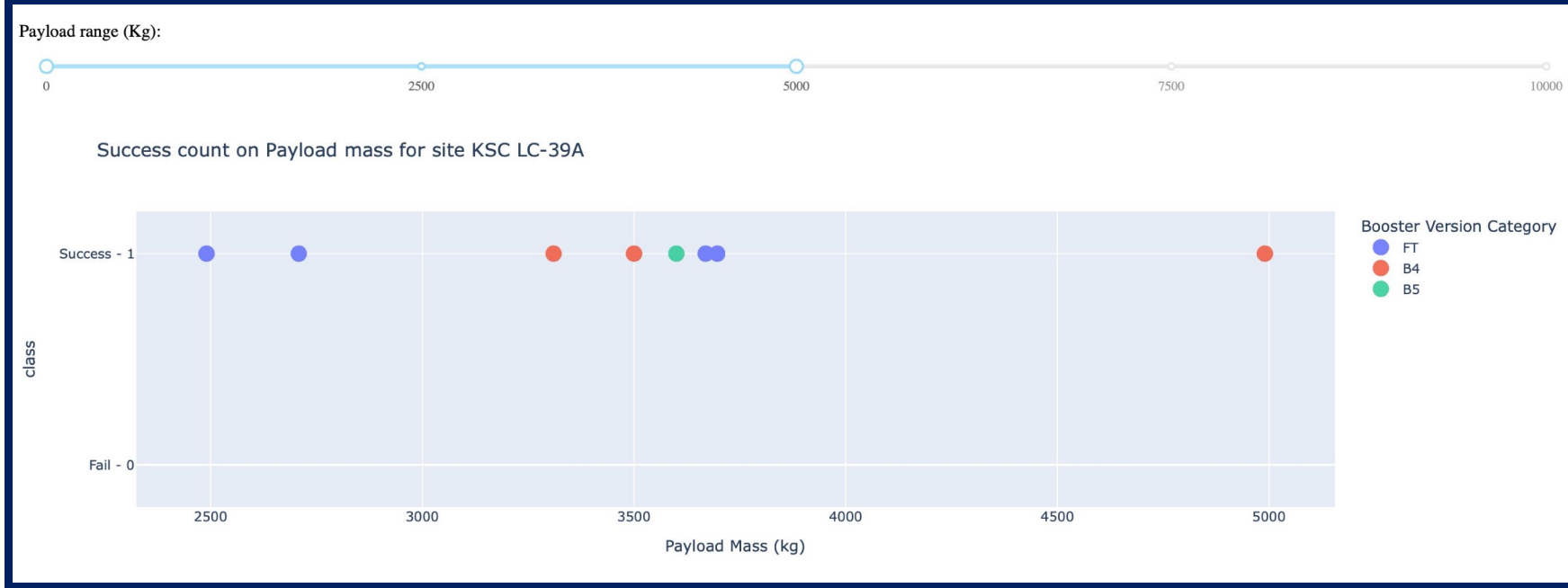
Build a Dashboard with Dash

Scatterplot for **Payload vs. Launch**

The slider for this graphic allows to analyze the correlation of payload on launch outcome.

The example shows results for site KSC LC-39A for payloads mass 0 to 5000kg (top) and 5000 to 6800kg (bottom).

We can see that this site does better with payloads of less than 5400kg.

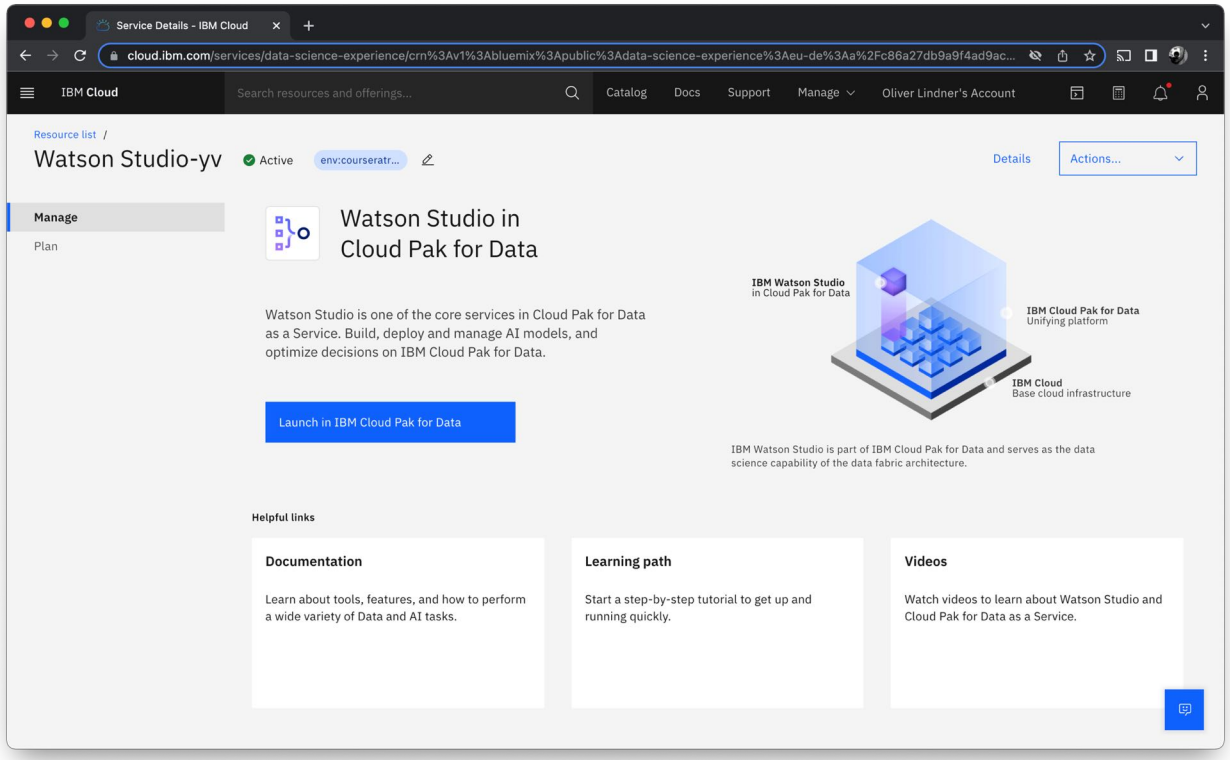


Predictive Analysis (Classification)

Several suitable machine learning models were trained and tested using the available data set:

- Decision Tree
- Logistic Regression
- K-nearest Neighbors
- Support Vector Machine

The task was to tune all models for optimum results and find the best performing model with the highest accuracy.

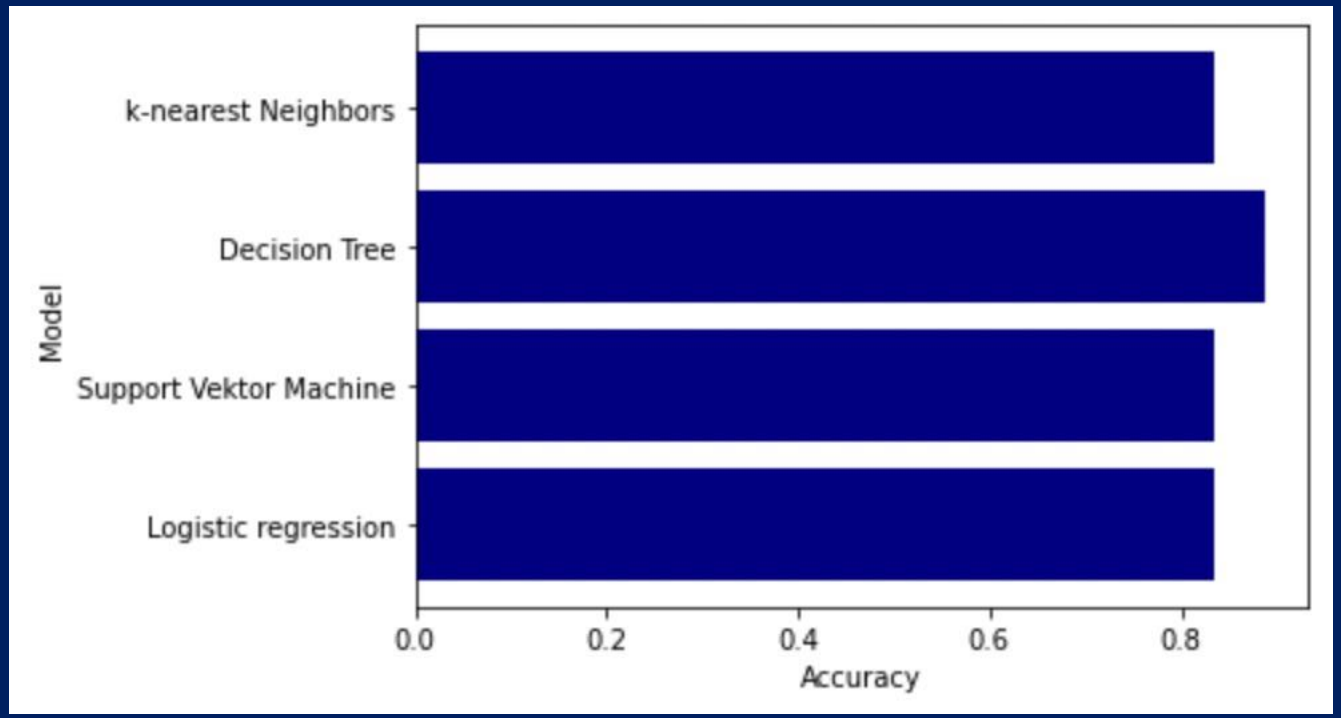


Predictive Analysis

Overall Results for Model **Classification**

The outcome of the comparison of the prediction accuracy for the four models trained and tested is shown in horizontal bar graph here.

The model with highest prediction accuracy was “Decision Tree” with 88,88%.

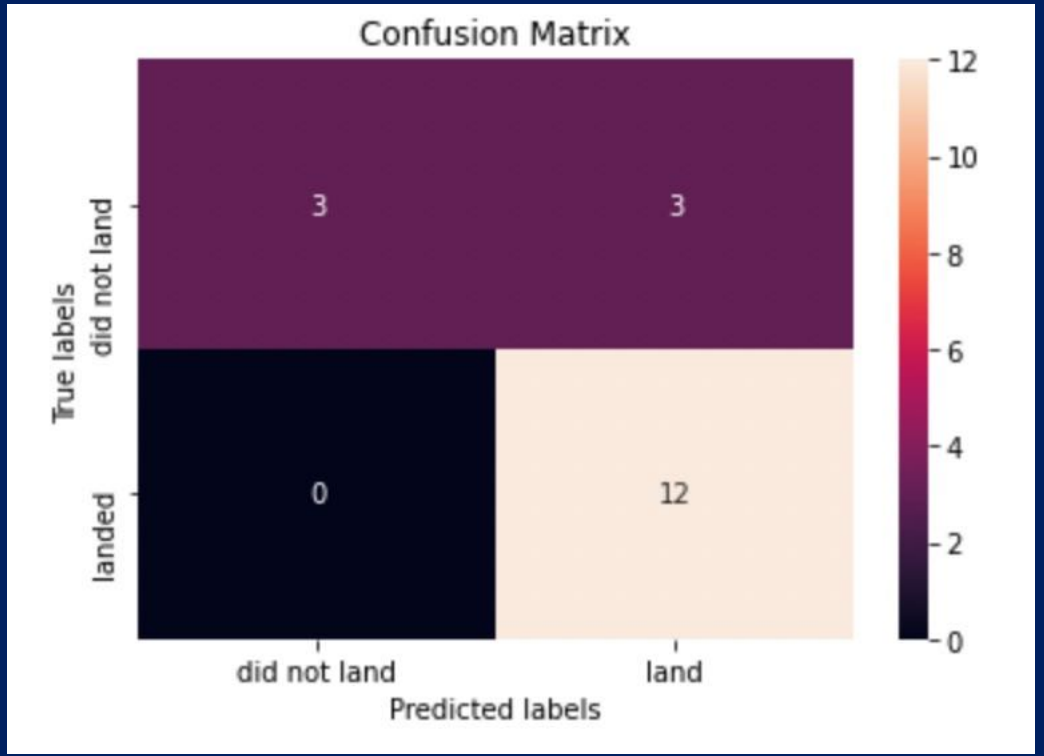


GitHub

Predictive Analysis

Confusion Matrix for the best performing model, Decision Tree

The graphic illustrated that the model, using the data set of 18 records, correctly predicted 12 successful landings but erred in properly predicting 3 failures of landing (false positive).



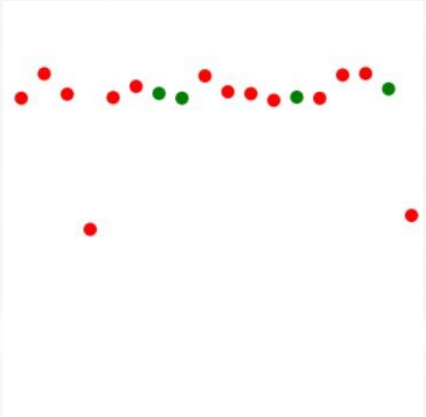
GitHub

Section 5

Conclusion

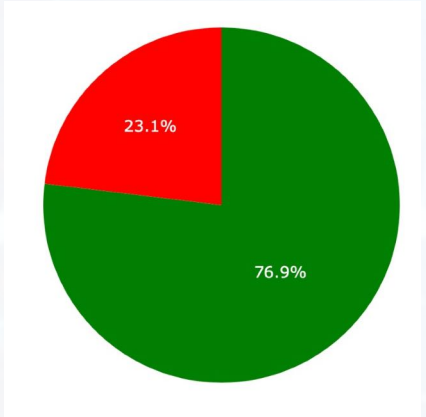
Section 5

Conclusion



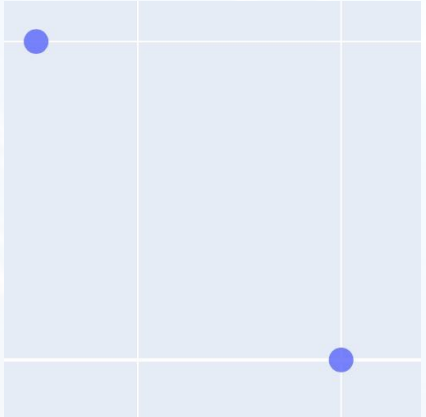
Flight Number

After 2013, the quota of successful launches increased consistently.



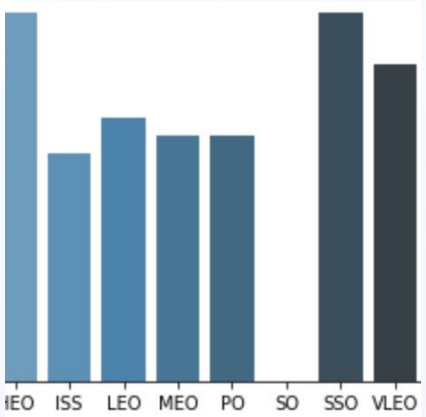
Launch Sites

Site KSLC-39A has the highest number of successful launches and the highest success rate overall.



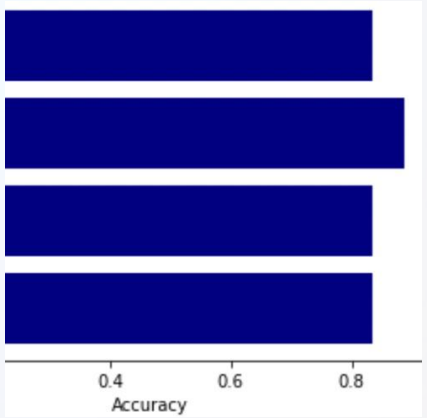
Payload

In general, higher payloads perform better than lighter payloads.



Orbits

Orbits SSO, HEO, GEO, and ES-L1 have the highest success rates (100%).



Outcome Prediction

Using the existing dataset, all models show prediction accuracy >80% with “Decision Tree” doing best with almost 89%.

Section 5

Conclusion: Caveat

The number of launch records that could be used for this analysis is limited. Segmenting the data set for training and testing the machine learning algorithm as per best practices results in a training set of only 18 records. For machine learning, this is a very low number. We do have to expect some model drift and a lower performance ratio in real life application than observed in lab environment.

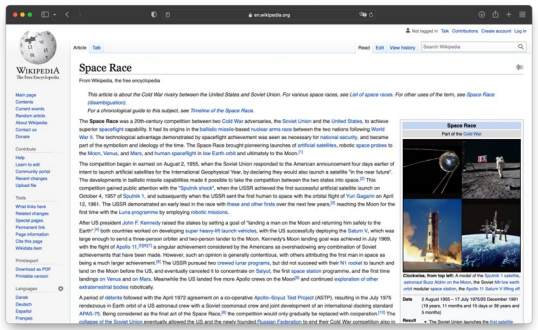
To optimize the model and maintain prediction accuracy, additional data is needed. The model should be retrained regularly when new records are available.

Section 6

Appendices

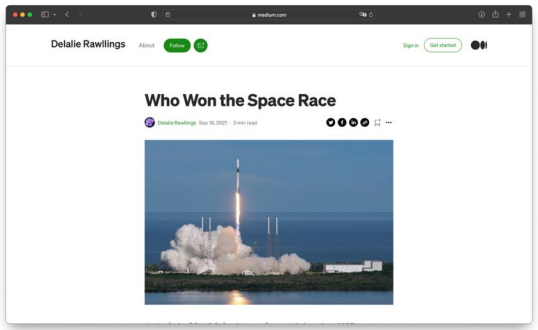
A: **Background Information**

Suggested internet reading on Space Race and Big Data in the Quest to Succeed



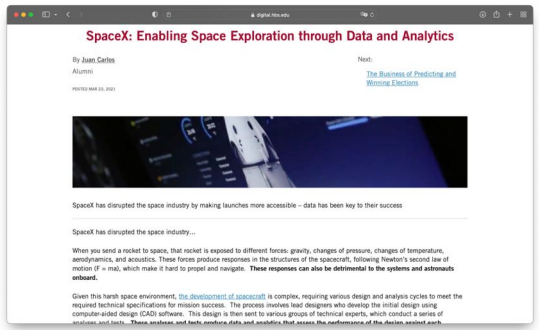
Space Race
Wikipedia

https://en.wikipedia.org/wiki/Space_Race



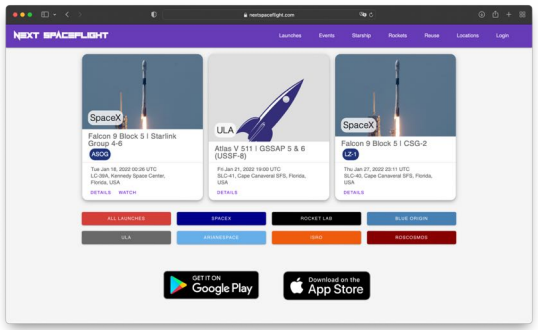
Who Won the Space Race
Delalie Rawllings

https://medium.com/@fredrawllings/who-won-the-space-race-fe6d0562b6b1



SpaceX: Enabling Space Exploration through Data and Analytics
Juan Carlos

https://digital.hbs.edu/platform-digit/submission/spacex-enabling-space-exploration-through-data-and-analytics/



Next Spaceflight
nextspaceflight.com

https://nextspaceflight.com

Section 6: Appendices

B: **Data Sources**: dataset_part_2.csv

```
In [2]: df=pd.read_csv("https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/datasets/dataset_part_2.csv")
```

```
In [3]: df.head(5)
```

Out[3]:

	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins	Reused	Legs	LandingPad	Block	ReusedCount	Serial	Longitude	Latitude	Class
0	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
1	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
2	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
3	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
4	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

```
In [6]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 90 entries, 0 to 89
Data columns (total 18 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   FlightNumber    90 non-null     int64
 1   Date            90 non-null     object
 2   BoosterVersion  90 non-null     object
 3   PayloadMass     90 non-null     float64
 4   Orbit           90 non-null     object
 5   LaunchSite      90 non-null     object
 6   Outcome         90 non-null     object
 7   Flights         90 non-null     int64
 8   GridFins        90 non-null     bool
 9   Reused          90 non-null     bool
 10  Legs            90 non-null     bool
 11  LandingPad      64 non-null     object
 12  Block           90 non-null     float64
 13  ReusedCount     90 non-null     int64
 14  Serial          90 non-null     object
 15  Longitude       90 non-null     float64
 16  Latitude        90 non-null     float64
 17  Class           90 non-null     int64
dtypes: bool(3), float64(4), int64(4), object(7)
```

```
In [5]: df.describe()
```

Out[5]:

	FlightNumber	PayloadMass	Flights	Block	ReusedCount	Longitude	Latitude	Class
count	90.000000	90.000000	90.000000	90.000000	90.000000	90.000000	90.000000	90.000000
mean	45.500000	6104.959412	1.788889	3.500000	1.655556	-86.366477	29.449963	0.666667
std	26.124701	4694.671720	1.213172	1.595288	1.710254	14.149518	2.141306	0.474045
min	1.000000	350.000000	1.000000	1.000000	0.000000	-120.610829	28.561857	0.000000
25%	23.250000	2510.750000	1.000000	2.000000	0.000000	-80.603956	28.561857	0.000000
50%	45.500000	4701.500000	1.000000	4.000000	1.000000	-80.577366	28.561857	1.000000
75%	67.750000	8912.750000	2.000000	5.000000	3.000000	-80.577366	28.608058	1.000000
max	90.000000	15600.000000	6.000000	5.000000	5.000000	-80.577366	34.632093	1.000000

Section 6: Appendices

B: **Data Sources**: Spacex.csv

```
In [12]: df_data_1.head(5)
```

Out[12]:

	Date	Time (UTC)	Booster_Version	Launch_Site	Payload	PAYLOAD_MASS__KG_	Orbit	Customer	Mission_Outcome	Landing _Outcome
0	04-06-2010	18:45:00	F9 v1.0 B0003	CCAFS LC-40	Dragon Spacecraft Qualification Unit	0	LEO	SpaceX	Success	Failure (parachute)
1	08-12-2010	15:43:00	F9 v1.0 B0004	CCAFS LC-40	Dragon demo flight C1, two CubeSats, barrel of...	0	LEO (ISS)	NASA (COTS) NRO	Success	Failure (parachute)
2	22-05-2012	07:44:00	F9 v1.0 B0005	CCAFS LC-40	Dragon demo flight C2	525	LEO (ISS)	NASA (COTS)	Success	No attempt
3	08-10-2012	00:35:00	F9 v1.0 B0006	CCAFS LC-40	SpaceX CRS-1	500	LEO (ISS)	NASA (CRS)	Success	No attempt
4	01-03-2013	15:10:00	F9 v1.0 B0007	CCAFS LC-40	SpaceX CRS-2	677	LEO (ISS)	NASA (CRS)	Success	No attempt

```
In [14]: df_data_1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 101 entries, 0 to 100
Data columns (total 10 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Date               101 non-null    object
 1   Time (UTC)         101 non-null    object
 2   Booster_Version    101 non-null    object
 3   Launch_Site        101 non-null    object
 4   Payload            101 non-null    object
 5   PAYLOAD_MASS__KG_  101 non-null    int64
 6   Orbit              101 non-null    object
 7   Customer           101 non-null    object
 8   Mission_Outcome    101 non-null    object
 9   Landing _Outcome   101 non-null    object
dtypes: int64(1), object(9)
```

```
In [13]: df_data_1.describe()
```

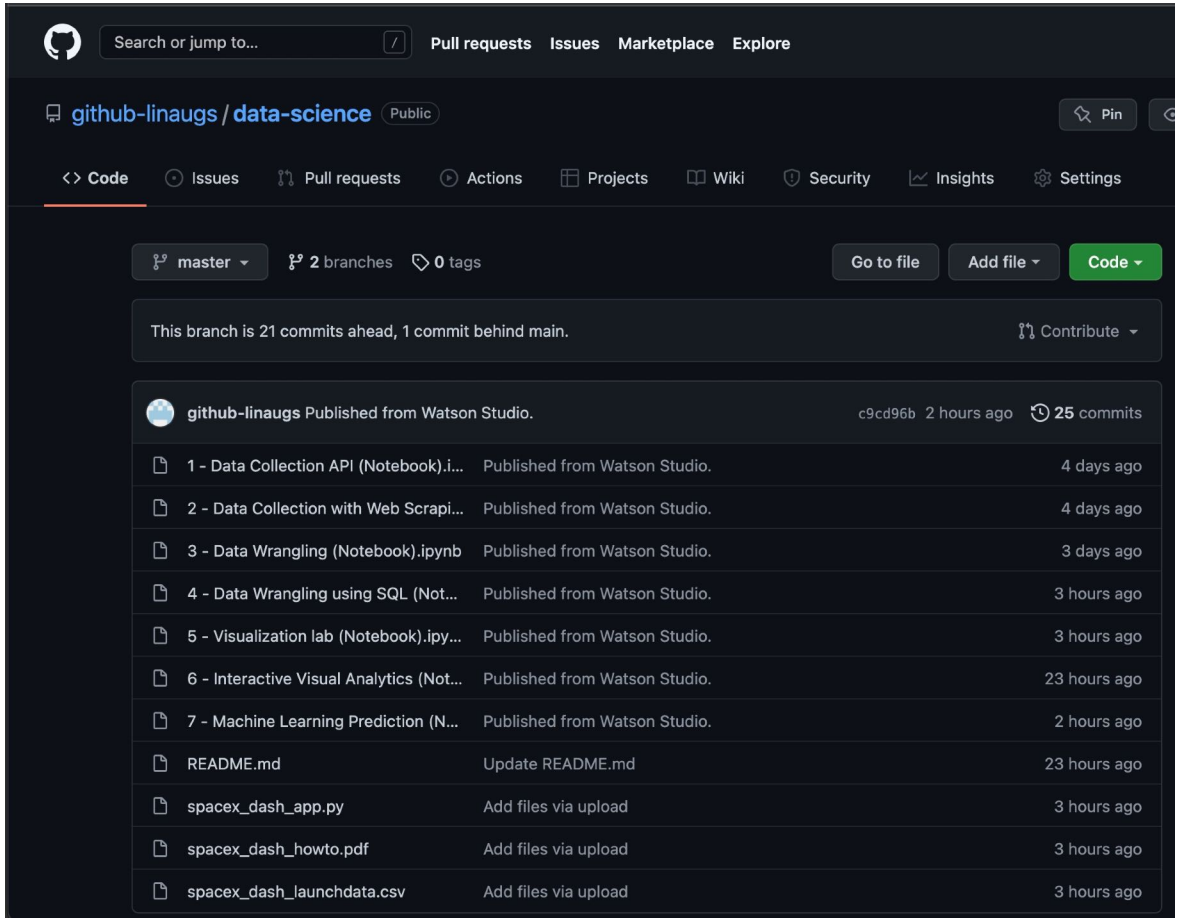
Out[13]:

	PAYLOAD_MASS__KG_
count	101.000000
mean	6138.287129
std	4900.998607
min	0.000000
25%	2500.000000
50%	4535.000000
75%	9600.000000
max	15600.000000

C: **Github Repository for this Project**

All Jupyter-Notebooks used to prepare this presentation, the Plotly Dash application (interactive dashboard) code including corresponding launch data file, and the PDF-version of this presentation are available for download on Github:

https://github.com/github-linaugs/data-science/tree/master



Thank you!