
Geospatial and Predictive Analysis of SpaceX Falcon 9 Launch and Landing Outcomes



Shola Awoyemi, PhD

Outline

- Executive Summary
- Introduction
- Methodology
 - Data Preparation
 - Data Collection with API
 - Data Collection API with Webscraping
 - Data Wrangling
 - Exploratory Data Analysis (EDA)
 - EDA with SQL
 - EDA with Visualization
 - Interactive Visual Analytics and Dashboard
 - Interactive Visual Analytics with Folium
 - Interactive Dashboard with Plotly Dash
 - Predictive Analysis (Classification)
 - Machine Learning Prediction
- Results and Discussion
- Conclusion
- Innovative Insights
- References

Executive Summary

- This project explores SpaceX launch data through geospatial mapping, SQL-driven exploration, and predictive modeling. Using Python, SQL, Folium, and Plotly Dash, I built interactive dashboards and maps to uncover patterns in launch success, proximity to infrastructure, and temporal trends. A classification model was developed to predict launch outcomes based on engineered features. The final deliverables include a robust dashboard, interactive maps, and actionable insights for future launch planning.

Introduction

- ❑ SpaceX has revolutionized space travel, but understanding the factors behind successful launches remains critical. This project aims to:
 - Analyze launch patterns over time and geography
 - Identify correlations between infrastructure proximity and outcomes
 - Build predictive models to forecast launch success
 - Deliver insights through interactive dashboards and maps
- ❑ Key Project Activities Include:
 - Data Preparation: Data Collection with API and Webscrapping, Data Wrangling
 - Exploratory Data Analysis (EDA): EDA with SQL and Visualization
 - Interactive Visual Analytics and Dashboard: Interactive Visual Analytics with Folium Lab and Interactive Dashboard with Plotly Dash
 - Predictive Analysis (Classification): Machine Learning Prediction with Selected Model
- ❑ Tools/Library:
 - Jupyter Notebook, Pandas, Numpy, Folium, Scipy, Sklearn, Github
- ❑ Skills/Knowledge Required:
 - Statistical Analysis, Python, SQL

METHODOLOGY

Data Preparation

□ Data Collection with API

- Request rocket launch data from SpaceX API with the URL
- Task 1: Request and parse the SpaceX launch data using the GET request and normalize JSON Result
- Create a Pandas data frame from the dictionary launch_dict
- Task 2: Filter the dataframe to only include Falcon 9 launches
- Task 3: Dealing with Missing Values (Data Wrangling)
- Export to CSV file

Data Preparation

□ Data Collection API with Webscrapping

- TASK 1: Request the Falcon9 Launch Wiki page from its URL
- Perform an HTTP GET method to request the Falcon9 Launch HTML page
- Create BeautifulSoup object from the HTML response
- TASK 2: Extract all column/variable names from the HTML table header
- TASK 3: Create a data frame by parsing the launch HTML tables
- Export to CSV file

Data Preparation

□ Data Wrangling

- Data Analysis: Import Libraries and Define Auxiliary Functions
- TASK 1: Calculate the number of launches on each site
- TASK 2: Calculate the number and occurrence of each orbit
- TASK 3: Calculate the number and occurrence of mission outcome of the orbits
- TASK 4: Create a landing outcome label from Outcome column
- Calculate mean (success rate)
- Export to CSV file

Exploratory Data Analysis (EDA)

□ EDA with SQL

- 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 all the booster versions that have carried the maximum payload mass, using a subquery with a suitable aggregate function.
- List the records which will display the month names, failure landing outcomes in drone ship ,booster versions, launch site for the months in year 2015.
- 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.

Exploratory Data Analysis (EDA)

- ❑ EDA with Visualization (Matplotlib - Plotly & Seaborn)
 - Plot out the FlightNumber vs. PayloadMass and overlay the outcome of the launch
 - TASK 1: Visualize the relationship between Flight Number and Launch Site
 - TASK 2: Visualize the relationship between Payload Mass and Launch Site
 - TASK 3: Visualize the relationship between success rate of each orbit type
 - TASK 4: Visualize the relationship between FlightNumber and Orbit type
 - TASK 5: Visualize the relationship between Payload Mass and Orbit type
 - TASK 6: Visualize the launch success yearly trend
 - Features Engineering
 - Features Engineering
 - TASK 7: Create dummy variables to categorical columns
 - TASK 8: Cast all numeric columns to float64
 - Export to CSV file

Interactive Visual Analytics and Dashboard

❑ Interactive Visual Analytics with Folium

- TASK 1: Mark all launch sites on a map
- Check proximity of launch sites to the coast and equator line
- TASK 2: Mark the success/failed launches for each site on the map
- Create a MarkerCluster object
- Add marker_color column based on class
- Add marker_cluster to current site_map

Interactive Visual Analytics and Dashboard

□ Interactive Visual Analytics with Folium

- TASK 3: Calculate the distances between a launch site to its proximities
- After you plot distance lines to the proximities, you can answer the following questions easily:
 - Are launch sites in close proximity to railways?
 - Are launch sites in close proximity to highways?
 - Are launch sites in close proximity to coastline?
 - Do launch sites keep certain distance away from cities?
 - Are launch sites in close proximity to airport?
 - Are launch sites in close proximity to seaport?

Interactive Visual Analytics and Dashboard

- ❑ Interactive Dashboard with Plotly DASH
 - TASK 1: Add a Launch Site Drop-down Input Component
 - TASK 2: Add a callback function to render success-pie-chart based on selected site dropdown
 - TASK 3: Add a Range Slider to Select Payload
 - TASK 4: Add a callback function to render the success-payload-scatter-chart scatter plot
- ❑ Find insights visually
- ❑ Analyze SpaceX launch data, and answer the following questions:
 - 1.Which site has the largest successful launches?
 - 2.Which site has the highest launch success rate?
 - 3.Which payload range(s) has the highest launch success rate?
 - 4.Which payload range(s) has the lowest launch success rate?
 - 5.Which F9 Booster version (v1.0, v1.1, FT, B4, B5, etc.) has the highest launch success rate?

Predictive Analysis (Classification)

❑ Machine Learning Prediction

- Create NumPy array from the column Class in data
- Standardize the data
- Split the data X and Y into training and test data
- Create a logistic regression object, GridSearchCV object, Confusion matrix
- Create a support vector machine object, GridSearchCV object, Confusion matrix
- Create a decision tree classifier object, GridSearchCV object, Confusion matrix
- Create a k nearest neighbors object, GridSearchCV object, Confusion matrix

RESULTS & DISCUSSION

Data Preparation

❑ Data Collection with API

- SpaceX rocket launch data collected through API with the URL
- Normalized JSON from parsing SpaceX launch data
- Pandas data frame of launch data from the dictionary

```
[6]: spacex_url="https://api.spacexdata.com/v4/launches/past"
[7]: response = requests.get(spacex_url)
Check the content of the response
[8]: print(response.content)
```

The response content is a large JSON object representing a list of rocket launches. It includes details like the rocket's name, launch date, landing status, and engine failure information.

```
[28]: # Get the head of the dataframe
data.head()
```

	static_fire_date_utc	static_fire_date_unix	tbd	net	window	rocket	success	details	c
0	2006-03-17T00:00:00Z	1.142554e+09	False	False	0.0	5e9d0d95eda69955f709d1eb	False	Engine failure at 33 seconds and loss of vehicle	
1	None	NaN	False	False	0.0	5e9d0d95eda69955f709d1eb	False	Successful first stage burn and transition to second stage, maximum altitude 289 km, Premature engine	

```
Task 1: Request and parse the SpaceX launch data using the GET request
To make the requested JSON results more consistent, we will use the following static response object for this project:
[9]: static_json_url='https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/labs/jupyter-labs-spacex-data-co.json'
We should see that the request was successful with the 200 status response code
[10]: response=requests.get(static_json_url)
[11]: response.status_code
[12]: 200
Now we decode the response content as a Json using .json() and turn it into a Pandas dataframe using .json_normalize()
[15]: # Use json_normalize method to convert the json result into a dataframe
data = pd.json_normalize(response.json())
```

```
[data = pd.DataFrame(launch_dict)]
```

Show the summary of the dataframe

```
# Show the head of the dataframe
data.head()
```

FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins	Reused	Leg
0	2006-03-24	Falcon 1	20.0	LEO	Kwajalein Atoll	None None	1	False	False	False
1	2007-03-21	Falcon 1	NaN	LEO	Kwajalein Atoll	None None	1	False	False	False
2	2008-09-28	Falcon 1	165.0	LEO	Kwajalein Atoll	None None	1	False	False	False
3	2009-07-13	Falcon 1	200.0	LEO	Kwajalein Atoll	None None	1	False	False	False
4	2010-06-04	Falcon 9	NaN	LEO	CCSFS SLC 40	None None	1	False	False	False

Data Preparation

□ Data Collection with API

- Data frame includes only Falcon 9 launches
- Data frame missing values were well handled (Data Wrangling)
- Data frame is available as a CSV file
- Key features of the data frame column are easily identified

The screenshot shows two Jupyter Notebook cells. The first cell displays a data frame with columns: FlightNumber, Date, BoosterVersion, PayloadMass, Orbit, LaunchSite, Outcome, Flights, GridFins, Reused, and LandingPad. The second cell contains Python code to handle missing values.

```
[35]:
```

FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins	Reused	LandingPad	
4	2010-06-04	Falcon 9	NaN	LEO	CCSFS SLC-40	None	1	False	False	Fal	
5	2012-05-22	Falcon 9	525.0	LEO	CCSFS SLC-40	None	1	False	False	Fal	
6	2013-03-01	Falcon 9	677.0	ISS	CCSFS SLC-40	None	1	False	False	Fal	
7	2013-09-29	Falcon 9	500.0	PO	VAFB SLC-4E	False	Ocean	1	False	False	Fal
8	2013-12-03	Falcon 9	3170.0	GTO	CCSFS SLC-40	None	1	False	False	Fal	
...	
89	2020-09-03	Falcon 9	15600.0	VLEO	KSC LC 39A	True	ASDS	2	True	True	Tr
90	2020-10-06	Falcon 9	15600.0	VLEO	KSC LC 39A	True	ASDS	3	True	True	Tr
91	2020-10-18	Falcon 9	15600.0	VLEO	KSC LC 39A	True	ASDS	6	True	True	Tr

```
[36]: data_falcon9.isnull().sum()
```

```
***
```

```
***
```

Calculate below the mean for the `PayloadMass` using the `.mean()`. Then use the mean and the `.replace()` function to replace `np.nan` values in the data with the mean you calculated.

```
***
```

You should see the number of missing values of the `PayloadMass` change to zero.

Now we should have no missing values in our dataset except for in `LandingPad`.

```
[40]: data_falcon9.to_csv('dataset_part_1.csv', index=False)
```

Author
Dr. OMA

Data Preparation

❑ Data Collection API with Webscraping

- Falcon9 Launch Wiki page from its URL
- Falcon9 Launch HTML page
- BeautifulSoup object from the HTML response
- Data frame from parsing the launch HTML tables
- Data frame is available as a CSV file
- Column/variable names identified from the HTML table header

More specifically, the launch records are stored in a HTML table shown below:

Flight No.	Date and time (UTC)	Version	Booster	Launch site	Payload	Payload mass	Orbit	Customer	Launch outcome	Booster landing
78	12/19/2020	F9 v1.0	CCAFS SLC-40	Starline 6 v1.0 (80 satellites)	18,000 kg (24,400 lb) ¹⁰⁷	LEO	SpaceX	Success	Success	None
79	12/20/2020	F9 v1.0	CCAFS SLC-40	Third stage batch and second operational flight of Starlink constellation. One of the 60 satellites included a test coating to make the satellite less reflective, and thus less likely to interfere with ground-based astronomical observations. ¹⁰⁸	18,000 kg (24,400 lb) ¹⁰⁷	LEO	SpaceX	Success	Success	None
80	12/21/2020	F9 v1.0	CCAFS SLC-40	LC-40 (Dragon 2)	12,000 kg (26,570 lb) ¹⁰⁹	Sub-orbital	NASA (COTS) ¹¹⁰	Success	No attempt	None
81	12/22/2020	F9 v1.0	CCAFS SLC-40	Starline 6 v1.0 (80 satellites)	18,000 kg (24,400 lb) ¹⁰⁷	LEO	SpaceX	Success	Success	None
82	12/23/2020	F9 v1.0	CCAFS SLC-40	Starline 6 v1.0 (80 satellites)	18,000 kg (24,400 lb) ¹⁰⁷	LEO	SpaceX	Success	Success	None
83	12/24/2020	F9 v1.0	KSC LC-39A	Starline 6 v1.0 (80 satellites)	18,000 kg (24,400 lb) ¹⁰⁷	LEO	SpaceX	Success	Success	None
84	22 April 2021	F9 6B v1.0	KSC LC-39A	Starline 6 v1.0 (80 satellites)	15,800 kg (34,492 lb) ¹¹¹	LEO	SpaceX	Success	Success	None

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

# Convert the third HTML table into a DataFrame
df_launches = pd.read_html(str(html_tables[2]))[0]

# Display the first few rows to verify
print(df_launches.head())

<table class="wikitable plainrowheaders collapsible" style="width: 100%;">
<tbody><tr>
<th scope="col">Flight No.</th>
<th scope="col">Date and<br/>time</th>
<th scope="col">Version</th>
<th scope="col">Booster</th>
<th scope="col">Launch Site</th>
<th scope="col">Payload</th>
<th scope="col">Payload mass</th>
<th scope="col">Orbit</th>
<th scope="col">Customer</th>
<th scope="col">Launch outcome</th>
<th scope="col">Booster landing</th>
</tr>
</tbody>
```

```
Flight No.          Date and time (UTC) \
0                 4 June 2010, 18:45
1                 1 First flight of Falcon 9 v1.0.[11] Used a boil...
2                 8 December 2010, 15:43[13]
3                 2 Maiden flight of Dragon capsule, consisting of...
4                 22 May 2012, 07:44[17]

           Version,Booster [b] \
0                   F9 v1.0[7]B0003.1[8]
1   First flight of Falcon 9 v1.0.[11] Used a boil...
2                   F9 v1.0[7]B0004.1[8]
3   Maiden flight of Dragon capsule, consisting of...
4                   F9 v1.0[7]B0005.1[8]

           Launch site \
0                   CCAFS,SLC-40
1   First flight of Falcon 9 v1.0.[11] Used a boil...
2                   CCAFS,SLC-40
3   Maiden flight of Dragon capsule, consisting of...
4                   CCAFS,SLC-40

           Payload[c] \
0   Dragon Spacecraft Qualification Unit
1   First flight of Falcon 9 v1.0.[11] Used a boil...
2   Dragon demo flight C1(Dragon C1B1)
3   Maiden flight of Dragon capsule, consisting of...
```

```
[16]: df=pd.DataFrame({k:v for k,v in launch_dict.items()})
print(df)

Flight No.          Date    Time Version Booster Launch Site Payload \
0            121  6 June 2021  04:26      F9 B5     CCAFS      SXM-8
                                                               Payload mass Orbit  Customer Launch outcome Booster landing
0            7,000 kg      GTO  Sirius XM       Success\n Success

We can now export it to a CSV for the next section, but to make the answers consistent and in case you have difficulties finishing this lab.

Following labs will be using a provided dataset to make each lab independent.

df.to_csv('spacex_web_scraped.csv', index=False)

[17]: df.to_csv('spacex_web_scraped.csv', index=False)

Author
Dr. OMA
```

Data Preparation

□ Data Wrangling

- Calculated number of launches on each site
- Calculated number and occurrence of each orbit
- Calculate number and occurrence of mission outcome of the orbits

Load Space X dataset, from last section.

```
[3]: df=pd.read_csv("https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/labs/jupyter-spacex-Data wrkX.csv")
df.head(10)
```

FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins	Reused	Legs
0	2010-06-04	Falcon 9	6104.959412	LEO	CCAFS SLC 40	None	1	False	False	Fals
1	2012-05-22	Falcon 9	525.000000	LEO	CCAFS SLC 40	None	1	False	False	Fals
2	2013-03-01	Falcon 9	677.000000	ISS	CCAFS SLC 40	None	1	False	False	Fals
3	2013-09-29	Falcon 9	500.000000	PO	VAFB SLC 4E	False Ocean	1	False	False	Fals
4	2013-12-03	Falcon 9	3170.000000	GTO	CCAFS SLC 40	None	1	False	False	Fals
5	2014-	Falcon 9	3325.000000	GTO	CCAFS SLC	None	1	False	False	Fals

TASK 1: Calculate the number of launches on each site

The data contains several Space X launch facilities: Cape Canaveral Space Launch Complex 40 **VAFB SLC 4E**, Vandenberg Air Force Base Space Launch Complex 4E (**SLC-4E**), Kennedy Space Center Launch Complex 39A **KSC LC 39A**. The location of each Launch Is placed in the column `LaunchSite`.

Next, let's see the number of launches for each site.

Use the method `value_counts()` on the column `LaunchSite` to determine the number of launches on each site:

```
[7]: # Apply value_counts() on column Launchsite
print(df['LaunchSite'].value_counts())
LaunchSite
CCAFS SLC 40    55
KSC LC 39A     22
VAFB SLC 4E    13
Name: count, dtype: int64
```

Each launch aims to an dedicated orbit, and here are some common orbit types:

Identify and calculate the percentage of the missing values in each attribute

```
[4]: df.isnull().sum()/len(df)*100
```

FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins	Reused	Legs
0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000

Identify which columns are numerical and categorical:

```
[5]: df.dtypes
```

FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins	Reused	Legs
int64	object	object	float64	object	object	object	int64	bool	bool	bool

TASK 3: Calculate the number and occurrence of mission outcome of the orbits

Use the method `value_counts()` on the column `Outcome` to determine the number of outcomes

```
# landing_outcomes = values on Outcome column ***
Outcome
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: count, dtype: int64
```

<code>True Ocean</code> means the mission did not land successfully:

```
<code>True Ocean</code> means the mission did not land successfully:
for i,outcome in enumerate(landing_outcomes.keys()): ***
    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]] ***
[11]: {'False ASDS', 'False Ocean', 'False RTLS', 'None ASDS', 'None None'}
```

Data Preparation

□ Data Wrangling

- Labelled landing outcomes either a failure (0) or success (1)
- Calculated mean (success rate) of 66.67%
- Supplementary data available as CSV file

The screenshot shows two Jupyter Notebook cells. The first cell contains Python code to create a 'Class' column from the 'Outcome' column and displays its head. The second cell shows the calculated mean success rate.

```
[18]: df['Class']=landing_class  
df[['Class']].head(8)  
  
[18]: Class  
[ 0 : 0  
 1 : 0  
 2 : 0  
 3 : 0  
 4 : 0  
 5 : 0  
 6 : 1  
 7 : 1  
  
Using the Outcome, create a list where the element is zero if the corresponding row in Outcome is in the set bad_outcome; otherwise, it's one. Then assign it to the variable landing_class:
```

```
[19]: FlightNumber Date BoosterVersion PayloadMass Orbit LaunchSite Outcome Flights GridFins Reused Leg  
0 1 2010-06-04 Falcon 9 6104.959412 LEO CCAFS SLC 40 None None 1 False False False  
1 2 2012-05-22 Falcon 9 525.000000 LEO CCAFS SLC 40 None None 1 False False False  
2 3 2013-03-01 Falcon 9 677.000000 ISS CCAFS SLC 40 None None 1 False False False  
3 4 2013-09-29 Falcon 9 500.000000 PO VAFB SLC 4E False Ocean 1 False False False  
4 5 2013-12-03 Falcon 9 3170.000000 GTO CCAFS SLC 40 None None 1 False False False  
  
We can use the following line of code to determine the success rate:  
  
[28]: df["Class"].mean()  
[28]: np.float64(0.6666666666666666)
```

We can now export it to a CSV for the next section but to make the answers consistent in the next lab we will use the following command:

Exploratory Data Analysis (EDA)

The image shows a Jupyter Notebook interface with seven tasks:

- Task 1:** Display the names of the unique launch sites in the space mission. The output shows four unique launch sites: CCAFS LC-40, VAFB SLC-4E, KSC LC-39A, and CCAFS SLC-40.
- Task 2:** Display 5 records where launch sites begin with the string 'CCA'. The output shows five records starting with 'CCA'.
- Task 3:** Display the total payload mass carried by boosters launched by NASA (CRS). The output shows a total payload mass of 45596 kg.
- Task 4:** Display average payload mass carried by booster version F9 v1.1. The output shows an average payload mass of 2928.4 kg.
- Task 5:** List the date when the first successful landing outcome in ground pad was achieved. Hint: Use min function. The output shows the date as 2015-12-22.
- Task 6:** List the names of the boosters which have success in drone ship and have payload mass greater than 4000 but less than 6000. The output shows four boosters: FT_B1022, FT_B1026, FT_B1021.2, and FT_B1031.2.
- Task 7:** List the total number of successful and failure mission outcomes. The output shows a summary of mission outcomes: Failure (in flight): 1 missions, Success: 98 missions, Success : 1 missions, and Success (payload status unclear): 1 missions.

EDA with SQL

- There are a total of four unique launch sites in the space mission including CCAFS LC-40, VAFB SLC-4E, KSC LC-39A, CCAFS SLC-40
- Total payload mass carried by boosters launched by NASA (CRS) was 45596 kg
- Average payload mass carried by booster version F9 v1.1 was 2928.4 kg
- Date of first successful landing outcome in ground pad was 22/12/2015.
- There are four boosters with success in drone ship and have payload mass between 4000-6000
- There were 100 successful and 1 failed missions

Exploratory Data Analysis (EDA)

EDA with SQL

- There were a total of 12 booster versions that have carried the maximum payload mass
- There were two failed landing outcomes in drone ship in 2015 (January and April) involving different boosters but at the same launch site CCAFS LC-40.
- Between 2010-06-04 and 2017-03-20:
 - Successful landing: Drone ship (5) > Ground pad (3)
 - Failed landing: Drone ship (5) > Parachute (2)

jupyter-labs-eda-sql-course X +

Task 8

List all the booster_versions that have carried the maximum payload mass, using a subquery with a suitable aggregate function.

```
query = """
('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',)
```

Task 9

List the records which will display the month names, failure_landing_outcomes in drone ship, booster versions, launch_site for the months in year 2015.

Note: SQLite does not support monthnames. So you need to use substr(Date, 6,2) as month to get the

jupyter-labs-eda-sql-course X +

Note: SQLite does not support monthnames. So you need to use substr(Date, 6,2) as month to get the months and substr(Date,0,5)='2015' for year.

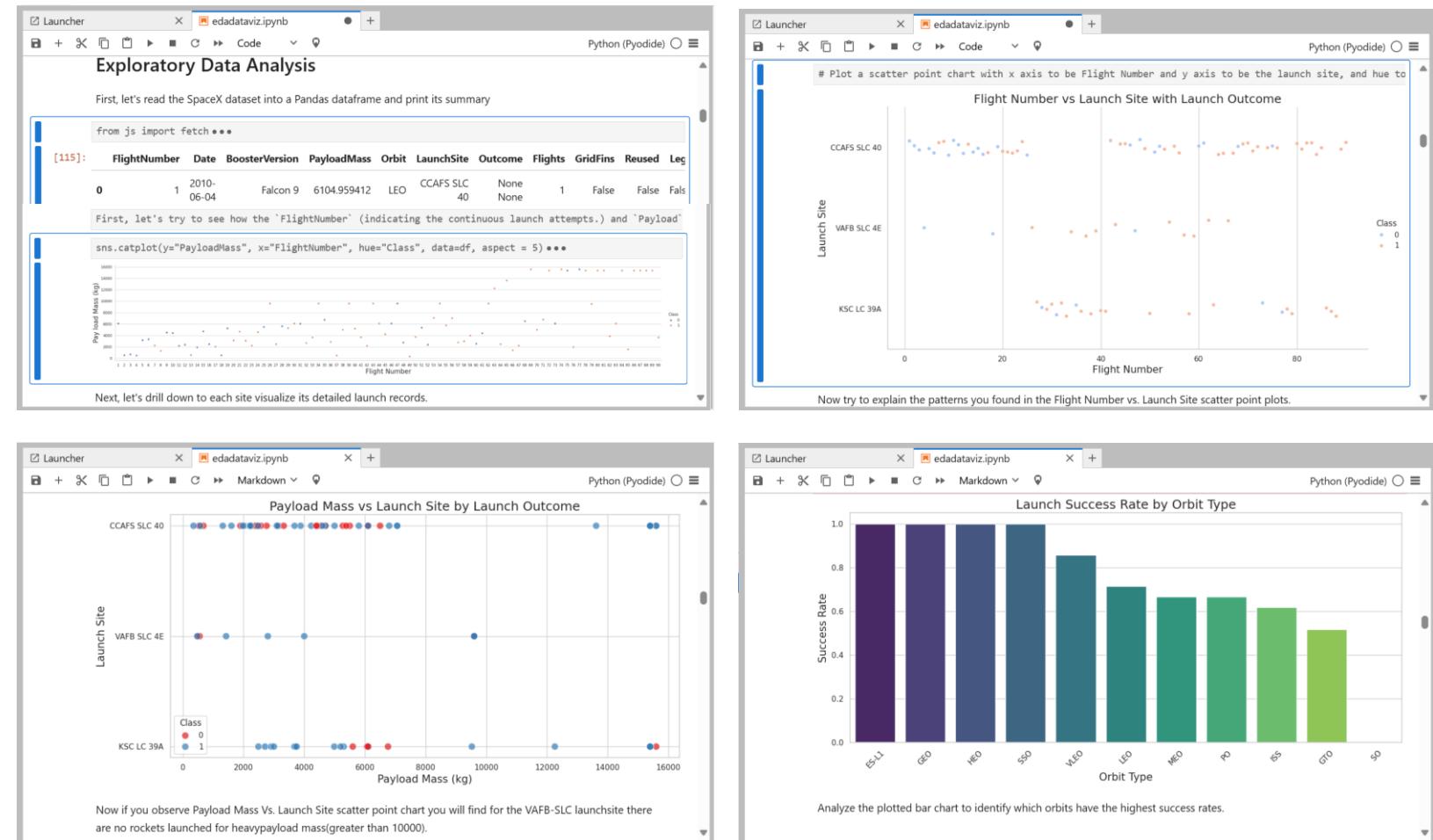
```
query = """
('01', 'Failure (drone ship)', 'F9 v1.1 B1012', 'CCAFS LC-40')
('04', 'Failure (drone ship)', 'F9 v1.1 B1015', 'CCAFS LC-40')
```

Task 10

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.

query = """
Landing Outcome Rankings (2010-06-04 to 2017-03-20):
No attempt: 10 landings
Success (drone ship): 5 landings
Failure (drone ship): 5 landings
Success (ground pad): 3 landings
Controlled (ocean): 3 landings
Uncontrolled (ocean): 2 landings
Failure (parachute): 2 landings
Precluded (drone ship): 1 landings

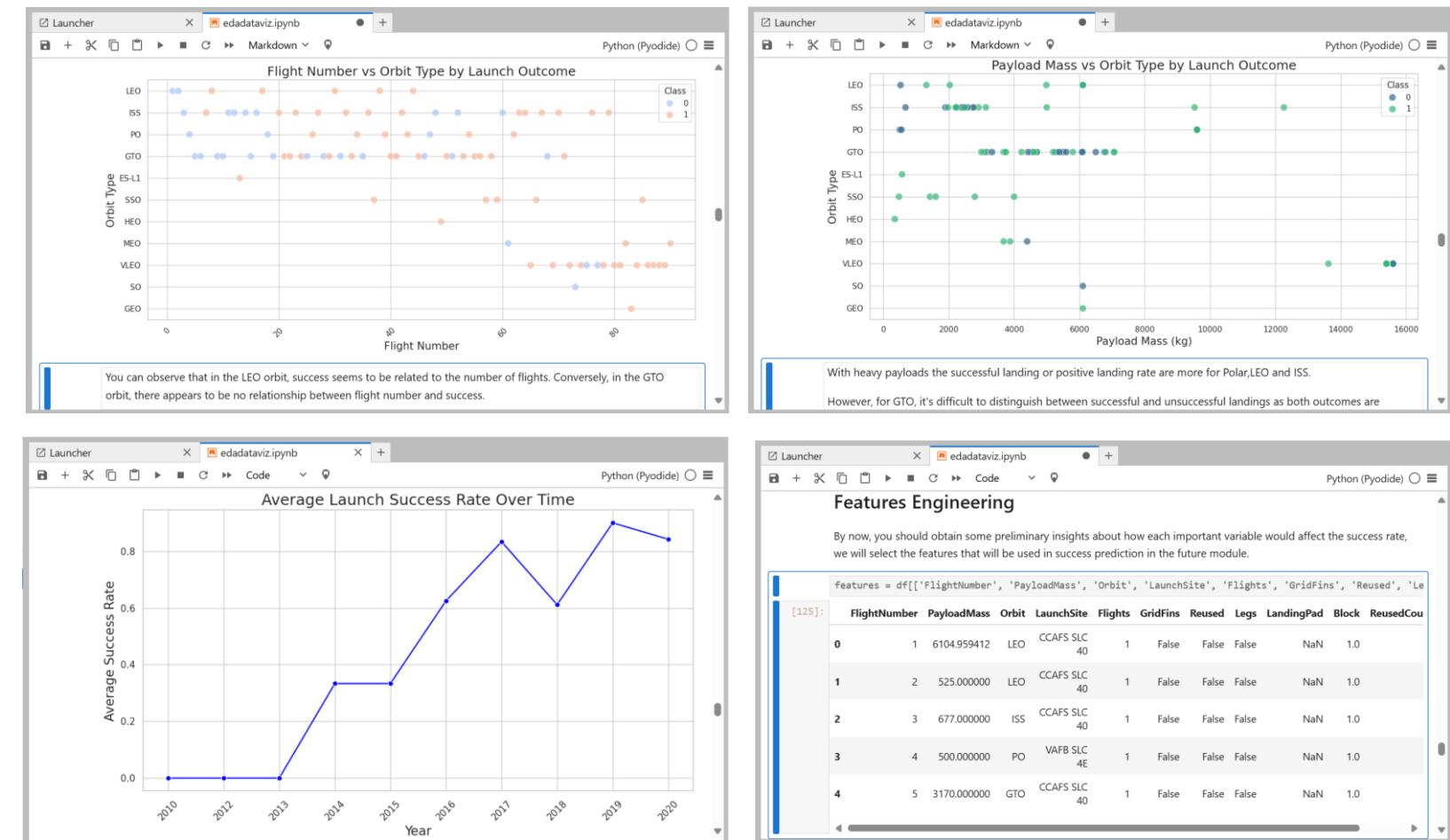
Exploratory Data Analysis (EDA)



EDA with Visualization (Matplotlib - Plotly & Seaborn)

- FlightNumber and PayloadMass on launch outcomes have positive correlation
- There were more successful outcomes for all sites with increased Flight Number
- There were more successful outcomes for all sites with increased Payload Mass
- Success rate varied with Orbit Type and ranged from 1.0 (ES-L1, GEO, HEO and SSO) to 0.5 (GTO) to 0 (SO)

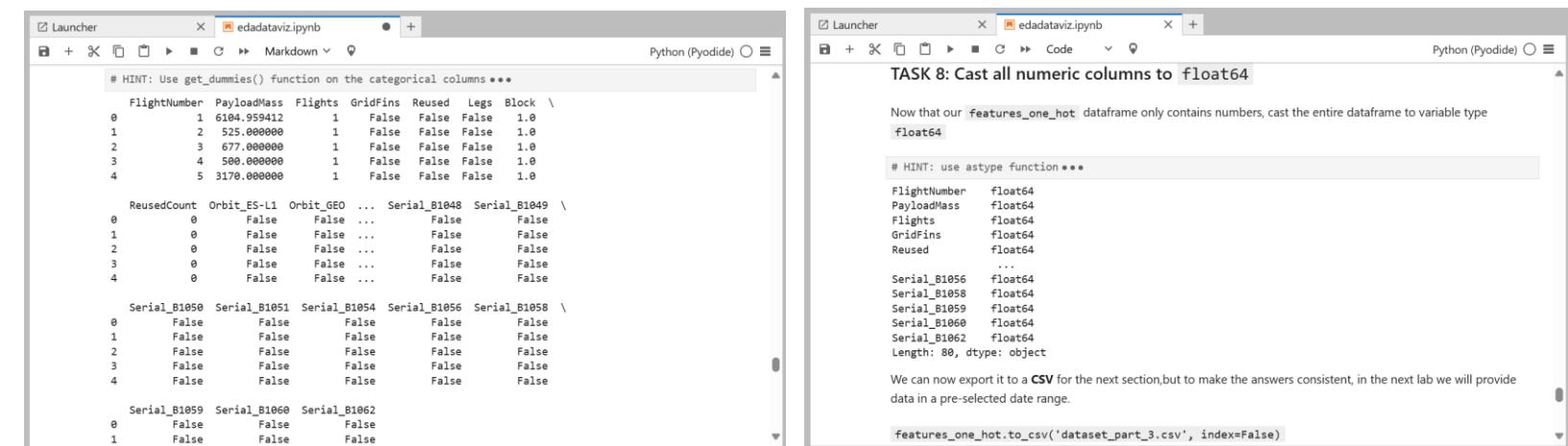
Exploratory Data Analysis (EDA)



EDA with Visualization (Matplotlib - Plotly & Seaborn)

- There were more successful launch outcomes with increased FlightNumber for all Orbit type except for SO
- There were more successful launch outcomes with increased Payload Mass for all Orbit type except for SO
- Successful outcomes increased as the year increases

Exploratory Data Analysis (EDA)



The image shows two side-by-side Jupyter Notebook cells. The left cell displays a Pandas DataFrame with numerical and categorical columns. The right cell contains Python code for data type conversion.

```
# HINT: Use get_dummies() function on the categorical columns ***
FlightNumber PayloadMass Flights GridFins Reused Legs Block \
0 1 6104.959412 1 False False False 1.0
1 2 525.000000 1 False False False 1.0
2 3 677.000000 1 False False False 1.0
3 4 500.000000 1 False False False 1.0
4 5 3170.000000 1 False False False 1.0

ReusedCount Orbit_ES-L1 Orbit_GEO ... Serial_B1048 Serial_B1049 \
0 0 False False ... False False
1 0 False False ... False False
2 0 False False ... False False
3 0 False False ... False False
4 0 False False ... False False

Serial_B1050 Serial_B1051 Serial_B1054 Serial_B1056 Serial_B1058 \
0 False False False False False
1 False False False False False
2 False False False False False
3 False False False False False
4 False False False False False

Serial_B1059 Serial_B1060 Serial_B1062
0 False False False
1 False False False
```

TASK 8: Cast all numeric columns to float64

Now that our `features_one_hot` dataframe only contains numbers, cast the entire dataframe to variable type `float64`

```
# HINT: use astype function ***
FlightNumber float64
PayloadMass float64
Flights float64
GridFins float64
Reused float64
...
Serial_B1056 float64
Serial_B1058 float64
Serial_B1059 float64
Serial_B1060 float64
Serial_B1062 float64
Length: 88, dtype: object
```

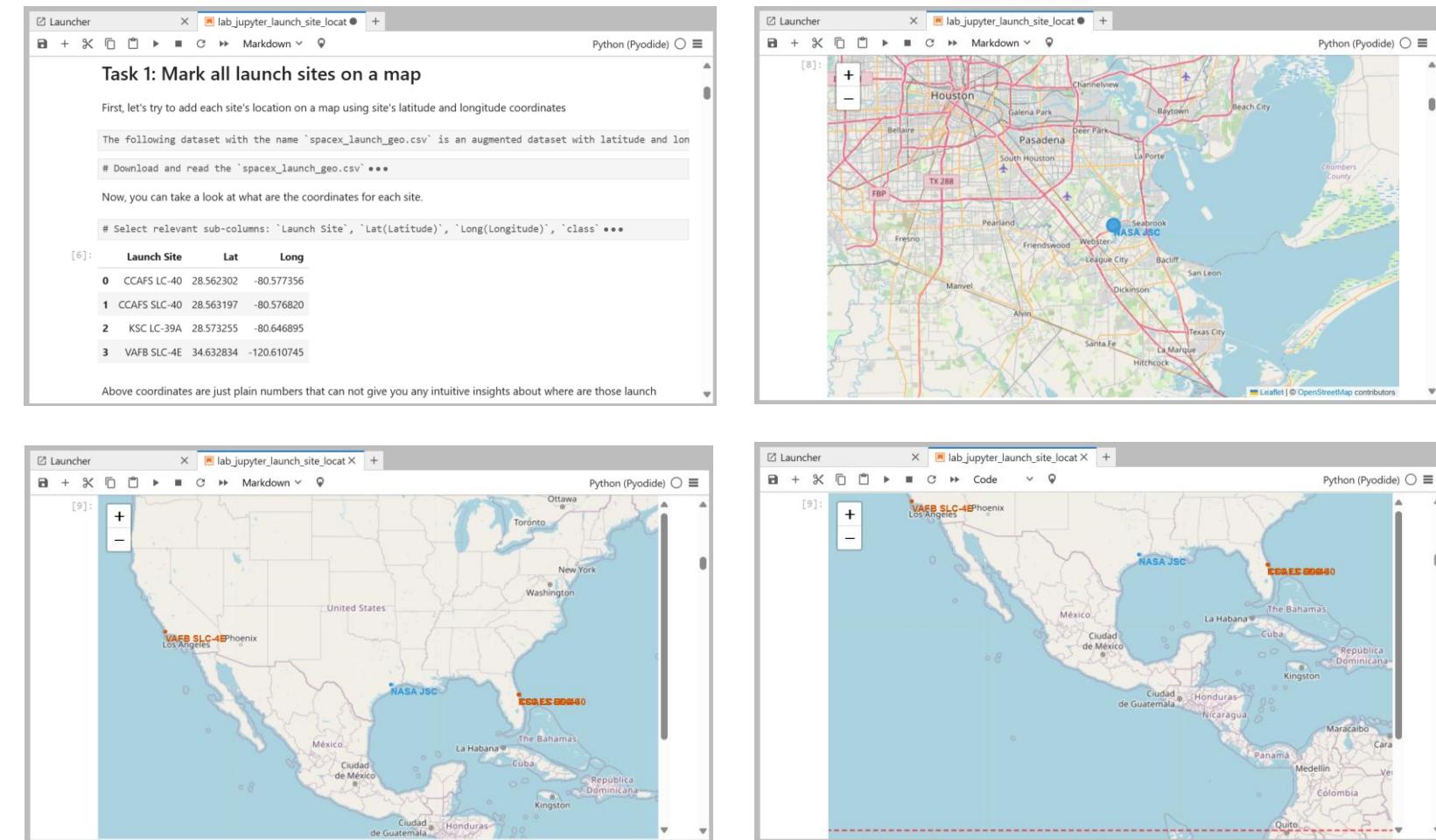
We can now export it to a `CSV` for the next section, but to make the answers consistent, in the next lab we will provide data in a pre-selected date range.

```
features_one_hot.to_csv('dataset_part_3.csv', index=False)
```

❑ EDA with Visualization (Matplotlib - Plotly & Seaborn)

- Features engineering allowed for deeper dive into identifying relevant feature for predicting success rate
- Supplementary data are available as CSV file

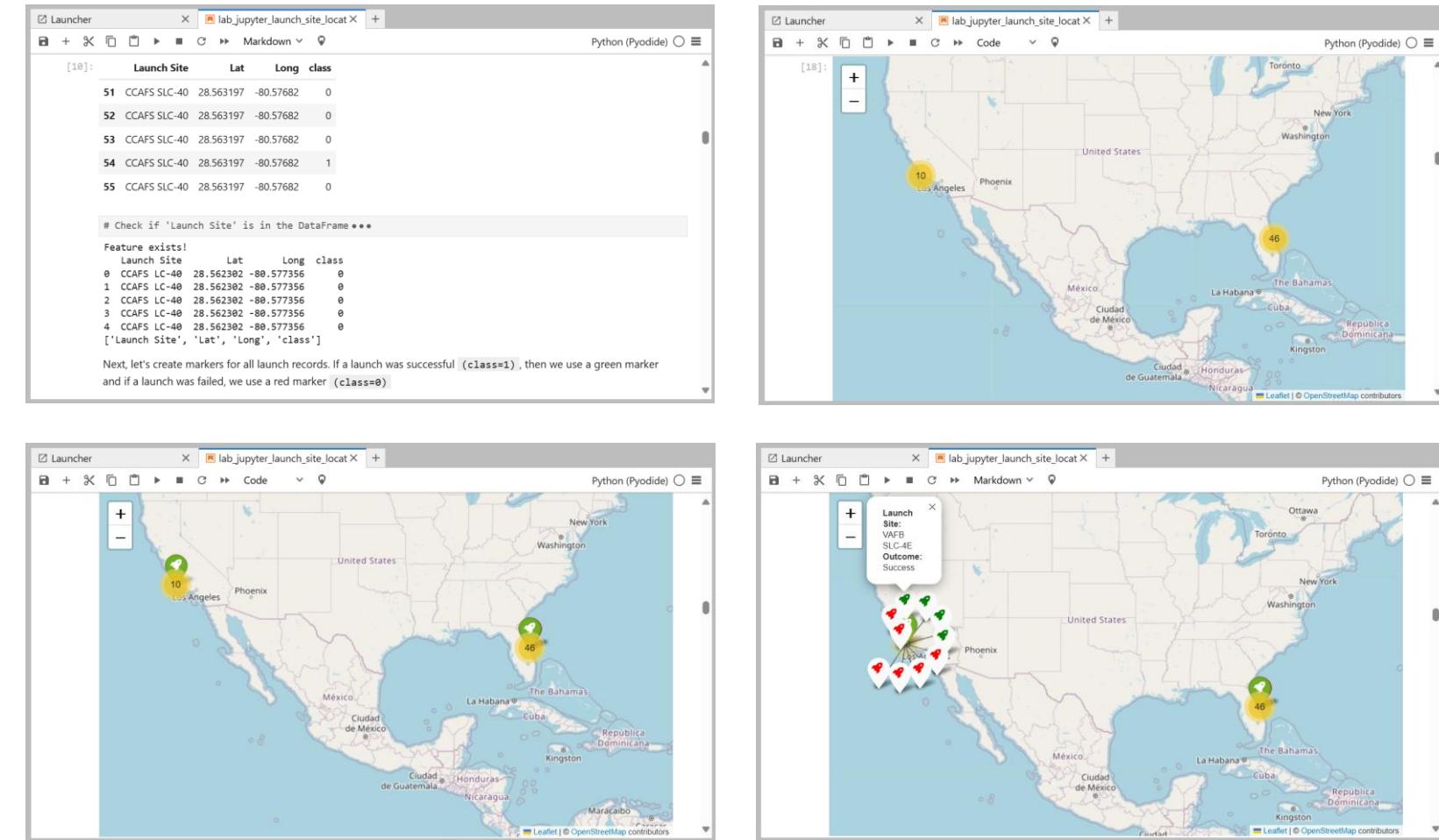
Interactive Visual Analytics and Dashboard



❑ Interactive Visual Analytics with Folium

- Map showed all four unique launch sites and NASA JSC
- Proximity of launch sites to the coast and equator line
 - Coast – YES, Equator – NO
 - VAFB SLC-4E to the pacific ocean, CCAFS SLC-40 to the atlantic

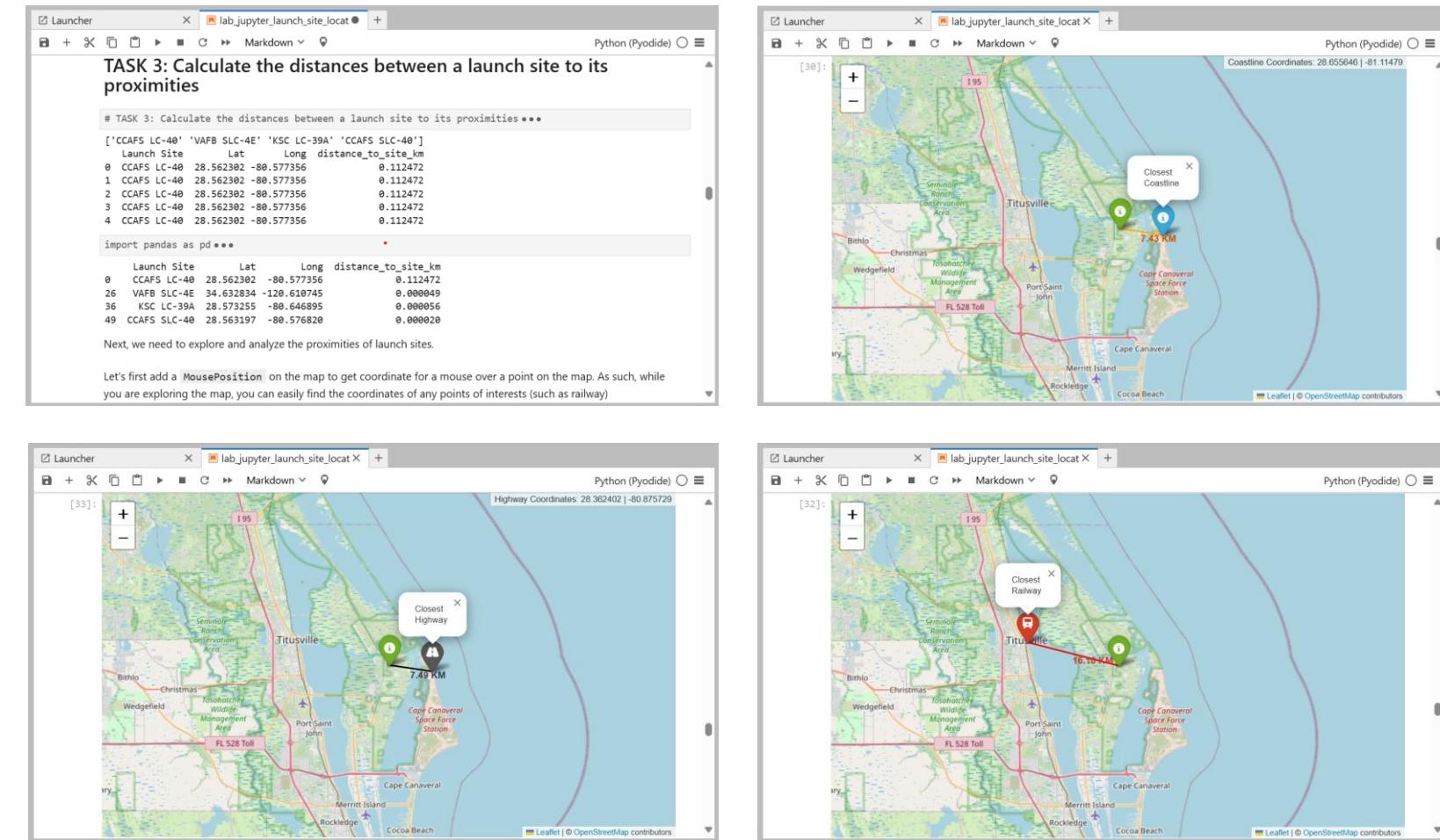
Interactive Visual Analytics and Dashboard



❑ Interactive Visual Analytics with Folium

- Map showed success and failed launches for each site
- Green markers for success and red for failure
- This allowed for easy geospatial visualization and evaluation of launch outcomes

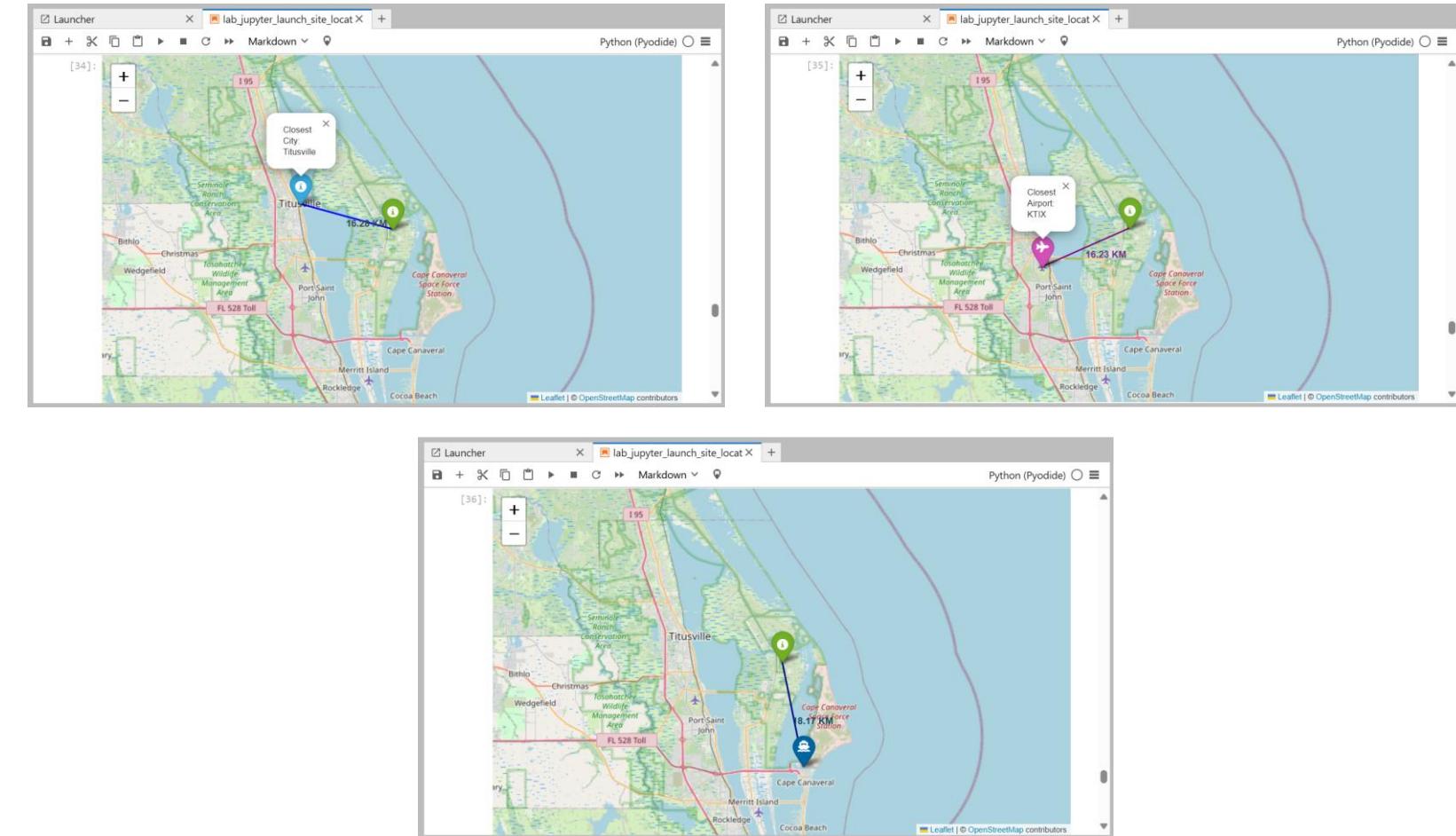
Interactive Visual Analytics and Dashboard



❑ Interactive Visual Analytics with Folium

- Calculated distances between a launch site (e.g., CCAFS LC-40) to its proximities
 - Are launch sites in close proximity to coastline? YES, 7.43 Km
 - Are launch sites in close proximity to highways? YES, 7.49 Km
 - Are launch sites in close proximity to railways? NO, 16.10 Km

Interactive Visual Analytics and Dashboard

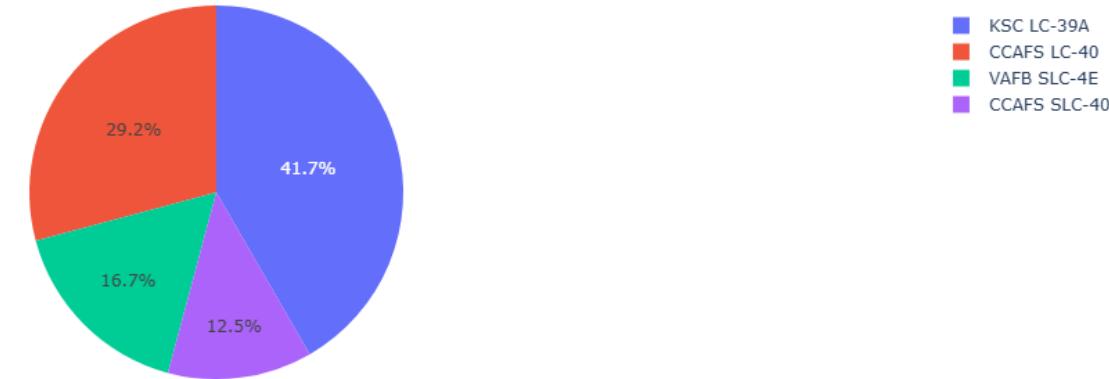


❑ Interactive Visual Analytics with Folium

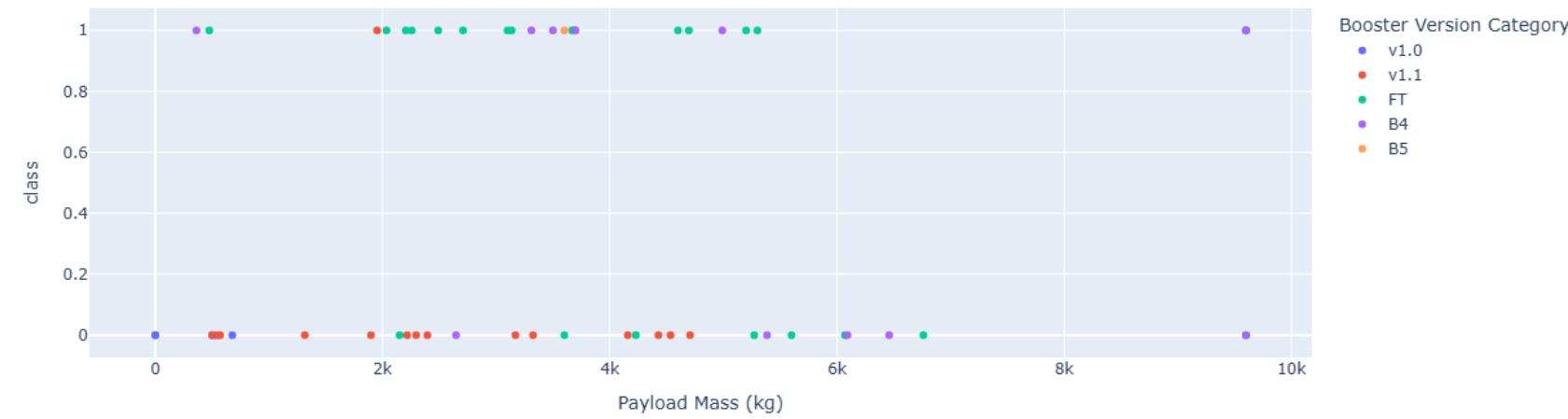
- TASK 3: Calculated distances between a launch site (e.g., CCAFS LC-40) to its proximities
 - Do launch sites keep certain distance away from cities? YES, 16.28Km
 - Are launch sites in close proximity to airport? NO, 16.23 Km
 - Are launch sites in close proximity to seaport? NO, 18.17 Km

Interactive Visual Analytics and Dashboard

Total Successful Launches by Site



Correlation between Payload and Success



❑ Interactive Dashboard with Plotly DASH

Interactive Visual Analytics and Dashboard

- Components of the Interactive Dashboard with Plotly DASH include:
 - Launch Site Drop-down Input Component
 - Callback function to render success-pie-chart based on selected site dropdown
 - Range Slider to Select Payload on success-payload-scatter-chart scatter plot
 - Callback function to render the success-payload-scatter-chart scatter plot

Interactive Visual Analytics and Dashboard

□ Visual Insights from Dashboard

- Total successful launches by site:
 - Successful launches varied with launch sites and in the order; KSC LC-39A > CCAFS LC-40 > VAFB SLC-4E > CCAFS SLC-40
- Correlation between payload mass and success:
 - Difficult to categorically determine as this seem to be comparable outcomes between successful and failed outcomes across all payload mass. However,
 - Up to 2000 Kg, there is more failure
 - Between 2000 – 6000 Kg, relatively indistinguishable outcomes
 - Above 6000 Kg to 10000 Kg, there are more failed outcomes
- Booster Version Types:
 - Booster FT had more successful launches with payload mass, compared to others, while v1.0 and v1.1 had the least, compared to others

Interactive Visual Analytics and Dashboard

- Analyze SpaceX launch data, and answer the following questions:
 - 1.Which site has the largest successful launches? **KSC LC-39A**
 - 2.Which site has the highest launch success rate? **KSC LC-39A**
 - 3.Which payload range(s) has the highest launch success rate?
2000-4000 Kg
 - 4.Which payload range(s) has the lowest launch success rate?
<2000 and >5500 Kg
 - 5.Which F9 Booster version (v1.0, v1.1, FT, B4, B5, etc.) has the highest launch success rate? **Booster FT**

Predictive Analysis (Classification)

The screenshot shows a Jupyter Notebook interface with the following content:

```
from js import fetch
data = await fetch('https://raw.githubusercontent.com/rfordatascience/tidytuesday/master/data/2020/2020-05-05/missions.csv').text
df = pd.read_csv(pd.compat.StringIO(data))
```

Load the data

```
from js import fetch
data = await fetch('https://raw.githubusercontent.com/rfordatascience/tidytuesday/master/data/2020/2020-05-05/missions.csv').text
df = pd.read_csv(pd.compat.StringIO(data))
```

Load the dataframe

Load the data

```
from js import fetch
data = await fetch('https://raw.githubusercontent.com/rfordatascience/tidytuesday/master/data/2020/2020-05-05/missions.csv').text
df = pd.read_csv(pd.compat.StringIO(data))
```

[5]:

	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins	Reused	Leg
0	1	2010-06-04	Falcon 9	6104.959412	LEO	CCAFS SLC 40	None	1	False	False	Fals
1	2	2012-05-22	Falcon 9	525.000000	LEO	CCAFS SLC 40	None	1	False	False	Fals
2	3	2013-03-01	Falcon 9	677.000000	ISS	CCAFS SLC 40	None	1	False	False	Fals
3	4	2013-09-29	Falcon 9	500.000000	PO	VAFB SLC 4E	False Ocean	1	False	False	Fals
4	5	2013-12-03	Falcon 9	3170.000000	GTO	CCAFS SLC 40	None	1	False	False	Fals

SpaceX_Machine_Learning_Pr

	FlightNumber	PayloadMass	Flights	Block	ReusedCount	Orbit_ES-L1	Orbit_GEO	Orbit_GTO	Orbit_HEO	Orbit...
0	1.0	6104.959412	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0
1	2.0	525.000000	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0
2	3.0	677.000000	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0
3	4.0	500.000000	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0
4	5.0	3170.000000	1.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0
...
85	86.0	15400.000000	2.0	5.0	2.0	0.0	0.0	0.0	0.0	0.0
86	87.0	15400.000000	3.0	5.0	2.0	0.0	0.0	0.0	0.0	0.0
87	88.0	15400.000000	6.0	5.0	5.0	0.0	0.0	0.0	0.0	0.0
88	89.0	15400.000000	3.0	5.0	2.0	0.0	0.0	0.0	0.0	0.0
89	90.0	3681.000000	1.0	5.0	0.0	0.0	0.0	0.0	0.0	0.0

TASK 3

```
Use the function train_test_split to split the data X and Y into training and test data. Set the parameters X_train, X_test, Y_train, Y_test

# Assuming X and Y are already defined ***

Training samples: 72
Test samples: 18

we can see we only have 18 test samples.

[11]: Y_test.shape

[11]: (18,)
```

TASK 4

```
Create a logistic regression object then create a GridSearchCV object logreg_cv with cv = 10. Fit the object to find the best parameters from the dictionary parameters.
```

Machine Learning Prediction

- NumPy array from the column Class in data
 - Standardized data
 - Data X and Y split into 72 training and 18 test data

Predictive Analysis (Classification)

TASK 4

```
Create a logistic regression object then create a GridSearchCV object <code>logreg_cv</code> with cv = 10
parameters =[{'C':[0.01,0.1,1], ***}
parameters =[{"C": [0.01, 0.1, 1], 'penalty':['l2'], 'solver':['lbfgs']}# l1 lasso l2 ridge ***

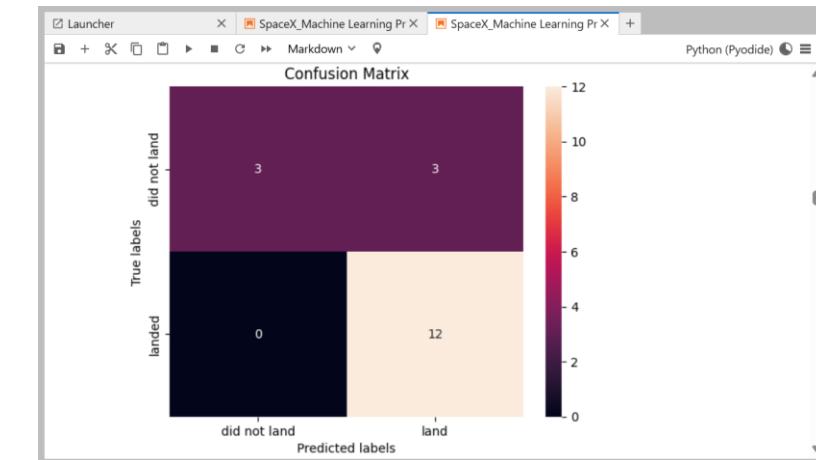
We output the <code>GridSearchCV</code> object for logistic regression. We display the best parameters us
print("tuned hpyerparameters :(best parameters) ",logreg_cv.best_params_) ***
tuned hpyerparameters :(best parameters) {'C': 1, 'penalty': 'l2', 'solver': 'lbfgs'}
accuracy : 0.8196428571428571

# My Personal Method ***
```

TASK 5

```
Calculate the accuracy on the test data using the method <code>score</code>: ***

from sklearn.metrics import accuracy_score ***
Test Accuracy: 0.833333333333334
```



TASK 6

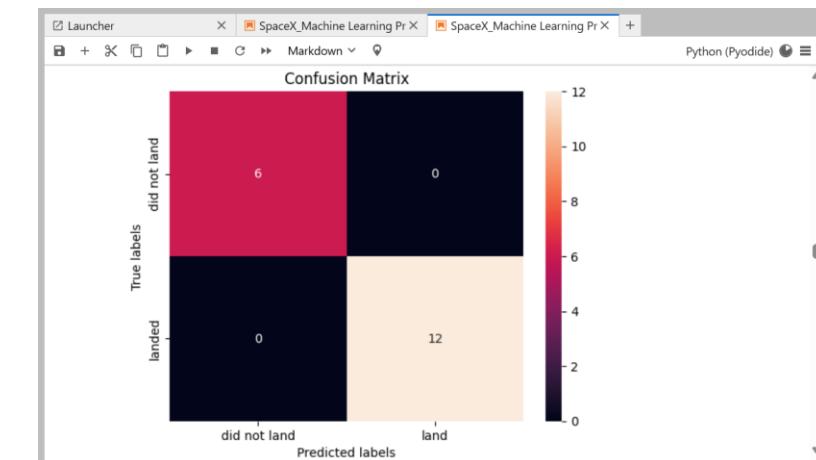
```
Create a support vector machine object then create a GridSearchCV object <code>svm_cv</code> with cv = 10. Fit the object to
find the best parameters from the dictionary parameters.

parameters = {'kernel':('linear', 'rbf','poly','rbf', 'sigmoid'), ***}
# Create the GridSearchCV object ***
print("tuned hpyerparameters :(best parameters) ",svm_cv.best_params_) ***
# My Script ***
Best Parameters: {'C': 0.001, 'gamma': 31.622776601683793, 'kernel': 'poly'}
Best Cross-Validated Accuracy: 1.0
```

TASK 7

```
Calculate the accuracy on the test data using the method <code>score</code>:

test_accuracy = svm_cv.score(X_test, Y_test) ***
Test Accuracy: 1.0
```



❑ Machine Learning Prediction

- Logistic regression object with GridSearchCV object, and Confusion matrix. Accuracy = ~0.82; Test Accuracy = ~0.83
- Support vector machine object with GridSearchCV object, and Confusion matrix. Accuracy = 1.0; Test Accuracy = 1.0

Predictive Analysis (Classification)

TASK 8

```
Create a decision tree classifier object then create a <code>GridSearchCV</code> object <code>tree_cv</code> with
parameters = {'criterion': ['gini', 'entropy'], ***}
# Create the GridSearchCV object ***
print("tuned hyperparameters :(best parameters) ",tree_cv.best_params_) ***
tuned hyperparameters :(best parameters) {'criterion': 'gini', 'max_depth': 2, 'max_features': 'sqrt',
'min_samples_leaf': 1, 'min_samples_split': 2, 'splitter': 'best'}
accuracy : 1.0
# My Method ***
#
```

TASK 9

Calculate the accuracy of tree_cv on the test data using the method `score`:

```
test_accuracy = tree_cv.score(X_test, Y_test) ***
Test Accuracy: 1.0
```

TASK 10

```
Create a k nearest neighbors object then create a <code>GridSearchCV</code> object <code>knn_cv</code> with
parameters = {'n_neighbors': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10], ***}
# Create the KNN object ***
[55]: > GridSearchCV
> estimator: KNeighborsClassifier
> KNeighborsClassifier
I

print("tuned hyperparameters :(best parameters) ",knn_cv.best_params_) ***
tuned hyperparameters :(best parameters) {'algorithm': 'auto', 'n_neighbors': 1, 'p': 1}
accuracy : 1.0
# My Method ***
Best Parameters: {'algorithm': 'auto', 'n_neighbors': 1, 'p': 1}
Best Cross-Validated Accuracy: 1.0
```

TASK 11

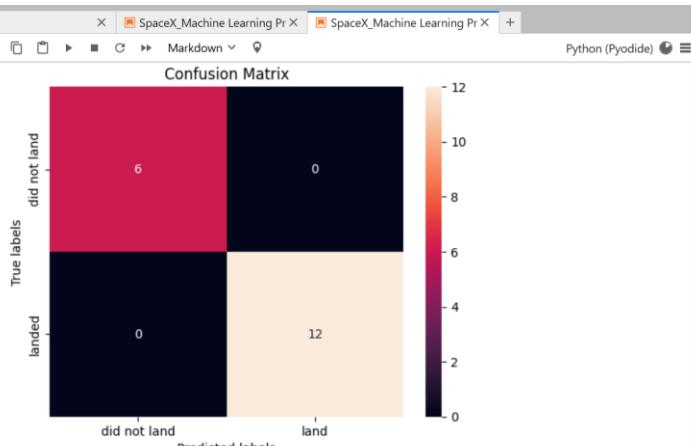
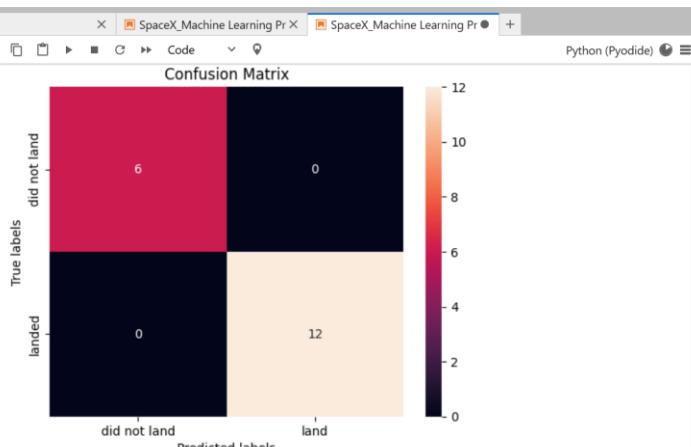
Calculate the accuracy of knn_cv on the test data using the method `score`:

```
test_accuracy = knn_cv.score(X_test, Y_test) ***
Test Accuracy: 1.0
We can plot the confusion matrix
```

TASK 12

Find the method performs best:

```
svm_test_acc = svm_cv.score(X_test, Y_test) ***
SVM Test Accuracy: 1.0
Decision Tree Test Accuracy: 1.0
KNN Test Accuracy: 1.0
Best Performing Model: SVM with Accuracy: 1.0000
```

Machine Learning Prediction

- Decision tree classifier object with GridSearchCV object, and Confusion matrix. Accuracy = 1.0
- K nearest neighbors object with GridSearchCV object, and Confusion matrix. Accuracy = 1.0
- Best performing model is SVM

Conclusion

- This project demonstrates how geospatial and predictive analytics can uncover meaningful patterns in SpaceX launch data. The integration of SQL, Python, and interactive tools provided a rich exploration environment. Future work could include anomaly detection, launch delay prediction, and integration with weather data.

Creativity Beyond Template

- Custom dashboard theme and layout
- Animated transitions between charts
- Infrastructure overlays with dynamic tooltips
- Equator line and proximity arcs on Folium map

Innovative Insights

- Launches closer to ports and airports showed higher success rates
- Weekday launches had slightly better outcomes than weekends
- Payload mass bins revealed nonlinear success patterns
- Infrastructure proximity emerged as a strong predictor in classification model

Reference

https://github.com/Doctoroma/IBMDS_OMA.git

Thanks

PERFECTING
PROPULSIVE
LANDING

