Prediction Of Falcon 9 Landing Successfully

By: Keshav Subramaniyam

Executive Summary

The goal of the capstone is to use exploratory data analysis and machine learning prediction analysis after data pre-processing and data wrangling to predict whether the Falcon 9 Spaceship will land successfully. Our goal is to utilize your data analysis tools to load a dataset, clean it, and find out interesting insights from it.

The following steps are being taken in order for us to analyze the given data:

1. Data Collection
2. Data Collection With Webscraping
3. Data Wrangling
4. Exploratory Data Analysis With SQL
5. Exploratory Data Analysis With Visualization
6. Interactive Visual Analytics With Folium
7. Interactive Visual Analytics With Plotly Dash
8. Machine Learning Prediction Lab

Introduction

SpaceX, an aerospace company that manufactures Spacecraft, is advertising Falcon 9 rocket launches on its website for 62 million dollars, while it cost 165 million for other providers. Considering that SpaceX can reuse its first stage, we can risk to analyze the first stage of the Falcon 9 launch. If we can determine whether the first stage will land or not, we can also determine the cost of the launch as well. We can use this to see if an alternative company wants to bid against SpaceX for a rocket launch. There are a series of tasks we will follow delineated in the bullets below that explains how data science plays a role in Falcon 9's efficacy:

1. Data Collection with REST API's and Web Scraping will be done. We will also convert the data into a dataframe and perform some data wrangling.
2. Building a dashboard in order to analyze launch records interactively with Plotly Dash. We will then build an interactive map to analyze launch site proximity with Folium.
3. Use machine learning training and testing data in order to find the best hyperparameter between SVM, Classification Trees, and Logistic Regression.

Data Collection And Data Wrangling

Data Collection

```
data.head(5)
```

	static_fire_date_utc	static_fire_date_unix	net	window	rocket	success	failures	details	crew	ships	capsules	payloads	launc
0	2006-03-17T00:00:00.000Z	1.142554e+09	False	0.0	5e9d0d95eda69955f709d1eb	False	[{'time': 33, 'altitude': None, 'reason': 'merlin engine failure'}]	Engine failure at 33 seconds and loss of vehicle	[]	[]	[]	[5eb0e4b5b6c3bb0006eeb1e1]	5e9e4502f5090995de5
1	None	NaN	False	0.0	5e9d0d95eda69955f709d1eb	False	[{'time': 301, 'altitude': 289, 'reason': 'harmonic oscillation leading to premature engine shutdown'}]	Successful first stage burn and transition to second stage, maximum altitude 289 km, Premature engine shutdown at T+7 min 30 s, Failed to reach orbit, Failed to recover first stage	[]	[]	[]	[5eb0e4b6b6c3bb0006eeb1e2]	5e9e4502f5090995de5
2	None	NaN	False	0.0	5e9d0d95eda69955f709d1eb	False	[{'time': 140, 'altitude': 35, 'reason': 'residual stage-1	Residual stage 1 thrust led to collision between	[]	[]	[]	[5eb0e4b6b6c3bb0006eeb1e3, 5eb0e4b6b6c3bb0006eeb1e4]	5e9e4502f5090995de5

Layout of the Dataframe

	Array
Category	
FlightNumber	[1, 2, 4, 5, 6, 8, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 22, 23, 24, 25, 26, 27, 28, 29, 30, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 57, 58, 59, 60, 61, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 76, 78, 79, 80, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 100, 101, 102, 103, 104, 105, 106]
Date	[2006-03-24, 2007-03-21, 2008-09-28, 2009-07-13, 2010-06-04, 2012-05-22, 2013-03-01, 2013-09-29, 2013-12-03, 2014-01-06, 2014-04-18, 2014-07-14, 2014-08-05, 2014-09-07, 2014-09-21, 2015-01-10, 2015-02-11, 2015-04-14, 2015-04-27, 2015-06-28, 2015-12-22, 2016-01-17, 2016-03-04, 2016-04-08, 2016-05-06, 2016-05-27, 2016-07-18, 2016-08-14, 2016-09-01, 2017-01-14, 2017-02-19, 2017-03-16, 2017-03-30, 2017-05-01, 2017-05-15, 2017-06-03, 2017-06-23, 2017-06-25, 2017-07-05, 2017-08-14, 2017-08-24, 2017-09-07, 2017-10-09, 2017-10-11, 2017-10-30, 2017-12-15, 2017-12-23, 2018-01-08, 2018-01-31, 2018-03-06, 2018-03-30, 2018-04-02, 2018-04-18, 2018-05-11, 2018-06-04, 2018-06-29, 2018-07-22, 2018-07-25, 2018-08-07, 2018-09-10, 2018-10-08, 2018-11-15, 2018-12-03, 2018-12-05, 2018-12-23, 2019-01-11, 2019-03-02, 2019-05-04, 2019-05-24, 2019-06-12, 2019-07-25, 2019-08-06, 2019-11-11, 2019-12-05, 2019-12-17, 2020-01-07, 2020-01-19, 2020-01-29, 2020-02-17, 2020-03-07, 2020-03-18, 2020-04-22, 2020-05-30, 2020-06-04, 2020-06-13, 2020-06-30, 2020-07-20, 2020-08-18, 2020-08-30, 2020-09-03, 2020-10-06, 2020-10-18, 2020-10-24, 2020-11-05]
BoosterVersion	[Falcon 1, Falcon 1, Falcon 1, Falcon 1, Falcon 9]
PayloadMass	[20, None, 165, 200, None, 525, 677, 500, 3170, 3325, 2296, 1316, 4535, 4428, 2216, 2395, 570, 1898, 4707, 2477, 2034, 553, 5271, 3136, 4696, 3100, 2257, 4600, 5500, 9600, 2490, 5600, 5300, None, 6070, 2708, 3669, 9600, 6761, 2910, 475, 4990, 9600, 5200, 3700, 2205, 9600, None, 4230, 6092, 9600, 2760, 350, 3750, 5383.85, 2410, 7076, 9600, 5800, 7060, 2800, 3000, 4000, 2573, 4400, 9600, 12259, 2482, 13200, 1425, 2227.7, 6500, 15600, 5000, 6800, 15600, None, 15600, 15600, 1977, 15600, 15600, 9525, 15600, 15600, 3880, None, 15600, 1600, 15600, 15600, 15600, 15600, 3681]
Orbit	[LEO, LEO, LEO, LEO, LEO, LEO, ISS, PO, GTO, GTO, ISS, LEO, GTO, GTO, ISS, ISS, ES-L1, ISS, GTO, ISS, LEO, PO, GTO, ISS, GTO, GTO, ISS, GTO, GTO, PO, ISS, GTO, GTO, LEO, GTO, ISS, GTO, PO, GTO, ISS, SSO, LEO, PO, GTO, GTO, ISS, PO, LEO, GTO, GTO, PO, ISS, HEO, GTO, GTO, ISS, GTO, PO, GTO, GTO, SSO, GTO, SSO, ISS, MEO, PO, ISS, ISS, VLEO, SSO, ISS, GTO, VLEO, ISS, GTO, VLEO, SO, VLEO, VLEO, ISS, VLEO, VLEO, ISS, VLEO, VLEO, MEO, GEO, VLEO, SSO, VLEO, VLEO, VLEO, VLEO, MEO]
LaunchSite	[Kwajalein Atoll, Kwajalein Atoll, Kwajalein Atoll, Kwajalein Atoll, CCSFS SLC 40, CCSFS SLC 40, CCSFS SLC 40, VAFB SLC 4E, CCSFS SLC 40, CCSFS SLC 40, CCSFS SLC 40, CCSFS SLC 40, CCSFS SLC 40, CCSFS SLC 40, CCSFS SLC 40, CCSFS SLC 40, CCSFS SLC 40, CCSFS SLC 40, CCSFS SLC 40, CCSFS SLC 40, CCSFS SLC 40, VAFB SLC 4E, CCSFS SLC 40, CCSFS SLC 40, CCSFS SLC 40, CCSFS SLC 40, CCSFS SLC 40, CCSFS SLC 40, CCSFS SLC 40, VAFB SLC 4E, KSC LC 39A, KSC LC 39A, KSC LC 39A, KSC LC 39A, KSC LC 39A, KSC LC 39A, KSC LC 39A, VAFB SLC 4E, KSC LC 39A, KSC LC 39A, VAFB SLC 4E, KSC LC 39A, VAFB SLC 4E, KSC LC 39A, KSC LC 39A, CCSFS SLC 40, VAFB SLC 4E, CCSFS SLC 40, CCSFS SLC 40, CCSFS SLC 40, VAFB SLC 4E, CCSFS SLC 40, CCSFS SLC 40, KSC LC 39A, CCSFS SLC 40, CCSFS SLC 40, CCSFS SLC 40, VAFB SLC 4E, CCSFS SLC 40, CCSFS SLC 40, VAFB SLC 4E, KSC LC 39A, VAFB SLC 4E, CCSFS SLC 40, CCSFS SLC 40, VAFB SLC 4E, KSC LC 39A, CCSFS SLC 40, CCSFS SLC 40, VAFB SLC 4E, CCSFS SLC 40, CCSFS SLC 40, CCSFS SLC 40, CCSFS SLC 40, CCSFS SLC 40, CCSFS SLC 40, KSC LC 39A, CCSFS SLC 40, CCSFS SLC 40, CCSFS SLC 40, KSC LC 39A, KSC LC 39A, KSC LC 39A, CCSFS SLC 40, CCSFS SLC 40, CCSFS SLC 40, CCSFS SLC 40, CCSFS SLC 40, CCSFS SLC 40, KSC LC 39A, KSC LC 39A, KSC LC 39A, CCSFS SLC 40, CCSFS SLC 40, Kwajalein Atoll, Kwajalein Atoll, Kwajalein Atoll, Kwajalein Atoll, CCSFS SLC 40, CCSFS SLC 40, ...]

Combining All Of The Columns Into The Dictionary

```
[62]: my_array = np.array(df.iloc[3]["Array"])
      my_array = np.where(my_array == None, 0, my_array)
      mean = sum(my_array) / len(my_array)
      my_array = np.where(my_array ==0, mean, my_array)
      my_array

[62]: array([5783.350555555556, 525, 677, 500, 3170, 3325, 2296, 1316, 4535,
             4428, 2216, 2395, 570, 1898, 4707, 2477, 2034, 553, 5271, 3136,
             4696, 3100, 2257, 4600, 5500, 9600, 2490, 5600, 5300,
             5783.350555555556, 6070, 2708, 3669, 9600, 6761, 2910, 475, 4990,
             9600, 5200, 3700, 2205, 9600, 5783.350555555556, 4230, 6092, 9600,
             2760, 350, 3750, 5383.85, 2410, 7076, 9600, 5800, 7060, 2800, 3000,
             4000, 2573, 4400, 9600, 12259, 2482, 13200, 1425, 2227.7, 6500,
             15600, 5000, 6800, 15600, 5783.350555555556, 15600, 15600, 1977,
             15600, 15600, 9525, 15600, 15600, 3880, 5783.350555555556, 15600,
             1600, 15600, 15600, 15600, 15600, 3681], dtype=object)
```

Replacing NaNs in Payload with Payload Mean Of Known Values

Web Scraping

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

You should able to see the columns names embedded in the table header elements `<th>` as follows:

```
<tr>
<th scope="col">Flight No.
</th>
<th scope="col">Date and<br/>time (<a href="/wiki/Coordinated_Universal_Time" title="Coordinated Universal Time">UTC</a>)
</th>
<th scope="col"><a href="/wiki/List_of_Falcon_9_first-stage_boosters" title="List of Falcon 9 first-stage boosters">Version,<br/>Booster</a> <sup
class="reference" id="cite_ref-booster_11-0"><a href="#cite_note-booster-11">[b]</a></sup>
</th>
<th scope="col">Launch site
</th>
<th scope="col">Payload<sup class="reference" id="cite_ref-Dragon_12-0"><a href="#cite_note-Dragon-12">[c]</a></sup>
</th>
<th scope="col">Payload mass
</th>
<th scope="col">Orbit
</th>
<th scope="col">Customer
</th>
<th scope="col">Launch<br/>outcome
</th>
<th scope="col"><a href="/wiki/Falcon_9_first-stage_landing_tests" title="Falcon 9 first-stage landing tests">Booster<br/>landing</a>
</th></tr>
```

List Of Columns In DataFrame

```
[40]: landing_outcomes = df["Outcome"].value_counts()
landing_outcomes

[40]: True ASDS      41
None None      19
True RTLS      14
False ASDS      6
True Ocean      5
False Ocean     2
None ASDS       2
False RTLS      1
Name: Outcome, dtype: int64
```

Count Of Each Landing Outcome

```
[31]: df["Orbit"].value_counts()

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

Count Of Each Orbit

Exploratory Data Analysis

EDA With SQL

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

[46]:

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

Booster Versions

[15]:
```
%sql SELECT DISTINCT("Launch_Site") FROM SPACEXTABLE;
```
```
* sqlite:///my_data1.db
Done.
```

[15]:

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

Distinct Site From SpaceTable

[18]:
```
%sql SELECT * FROM SPACEXTABLE WHERE Launch_Site LIKE "CCA%" LIMIT 5;
```
```
* sqlite:///my_data1.db
Done.
```

[18]:

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	7:44:00	F9 v1.0 B0005	CCAFS LC-40	Dragon demo flight C2	525	LEO (ISS)	NASA (COTS)	Success	No attempt
2012-10-08	0:35:00	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

Launch Sites With Beginning Title CCA

```
%sql SELECT SUM(PAYLOAD_MASS__KG_) FROM SPACEXTABLE WHERE Customer = "NASA (CRS)";
```
```
* sqlite:///my_data1.db
Done.
```

SUM(PAYLOAD_MASS__KG_)
45596

Sum Of Payload Mass With NASA CRS

EDA With SQL

```
%sql SELECT MIN(Date) FROM SPACEXTABLE WHERE Landing_Outcome = "Success (ground pad)";
```

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

MIN(Date)
2015-12-22

Earliest Date with Successful Landing At Ground Pad

```
[46]: %sql SELECT Booster_Version FROM SPACEXTABLE WHERE Landing_Outcome = 'Success (drone ship)' AND PAYLOAD_MASS__KG_ BETWEEN 4000 AND 6000;
```

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

[46]:

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

Earliest Date with Successful Landing At Ground Pad And Payload Mass Between 4000 and 6000.

```
[51]: %sql SELECT COUNT(*) FROM SPACEXTABLE GROUP BY("Mission_Outcome");
```

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

[51]:

COUNT(*)
1
98
1
1

Count Of Different Mission Outcomes

EDA With SQL

```
[54]: %sql SELECT Booster_Version FROM SPACEXTABLE WHERE PAYLOAD_MASS__KG_ = (SELECT MAX(PAYLOAD_MASS__KG_) FROM SPACEXTABLE);
 * sqlite:///my_data1.db
Done.
```

[54]:

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

Find Booster Version With Maximum Payload Mass

```
[69]: %sql SELECT COUNT(*) FROM SPACEXTABLE WHERE Date BETWEEN "2010-06-04" AND "2017-03-20" GROUP BY Landing_Outcome ORDER BY Date DESC;
 * sqlite:///my_data1.db
Done.
```

[69]:

COUNT(*)
5
3
1
5
3
2
10
2

Would you like to receive official Jupyter news?
Please read the privacy policy.

Landing Outcomes Between 03/20/2017 to 06/04/2020

```
[61]: %sql SELECT SUBSTR(Date, 6, 2), Booster_Version, Launch_Site FROM SPACEXTABLE WHERE Landing_Outcome = "Failure (drone ship)" AND SUBSTR(Date, 0, 5) = "2015";
 * sqlite:///my_data1.db
Done.
```

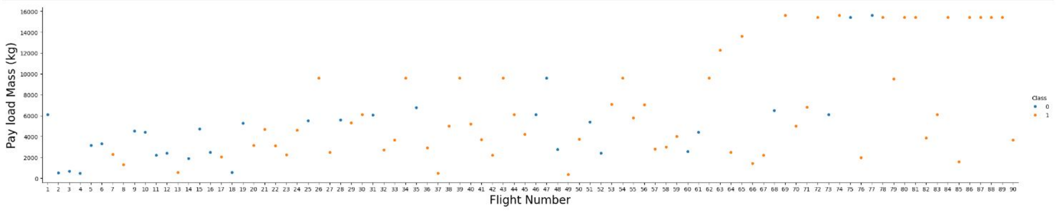
[61]:

SUBSTR(Date, 6, 2)	Booster_Version	Launch_Site
01	F9 v1.1 B1012	CCAFS LC-40
04	F9 v1.1 B1015	CCAFS LC-40

Failure Drone Ship and Year 2015

EDA With Visualization

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

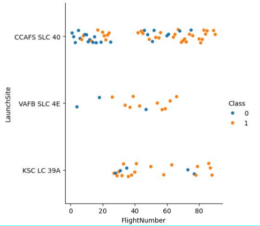


Catplot Between Payload And Flight Number

Use the function `catplot` to plot `FlightNumber` vs `LaunchSite`, set the parameter `x` parameter to `FlightNumber`,set the `y` to `Launch Site` and set the parameter `hue` to `'class'`

```
# Plot a scatter point chart with x axis to be Flight Number and y axis to be the launch site, and hue to be the class value
sns.catplot(x="FlightNumber", y="LaunchSite", hue = "Class", data=df)
```

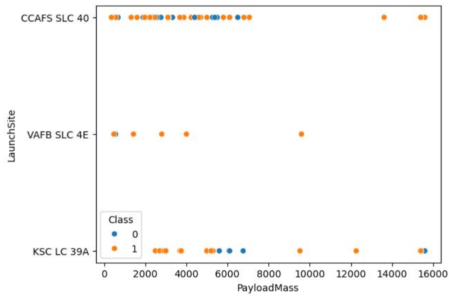
<seaborn.axisgrid.FacetGrid at 0x73cd458>



Catplot Between FlightNumber and LaunchSite

```
# Plot a scatter point chart with x axis to be Pay Load Mass (kg) and y axis to be the launch site, and hue to be the class value
sns.scatterplot(x="PayloadMass", y="LaunchSite", hue="Class", data=df)
```

<AxesSubplot:xlabel='PayloadMass', ylabel='LaunchSite'>

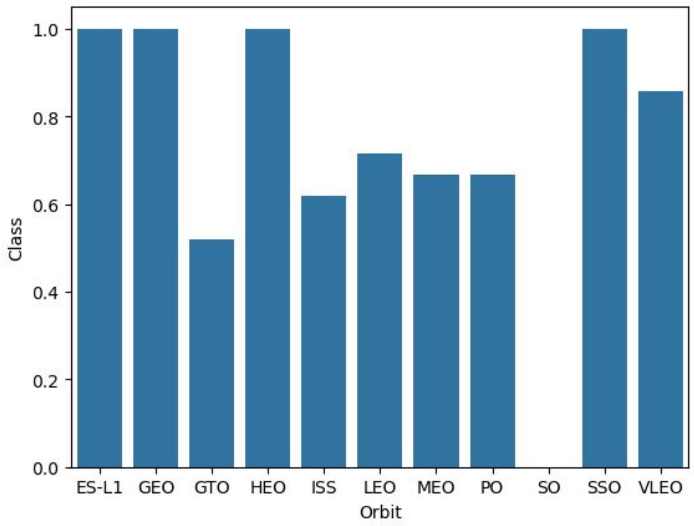


Scatterplot Between Payload Mass And Launch Site

EDA With Visualization

```python
# HINT use groupby method on Orbit column and get the mean of Class column
new_df = pd.DataFrame(df.groupby("Orbit")["Class"].mean())
sns.barplot(new_df, x="Orbit", y="Class")
```

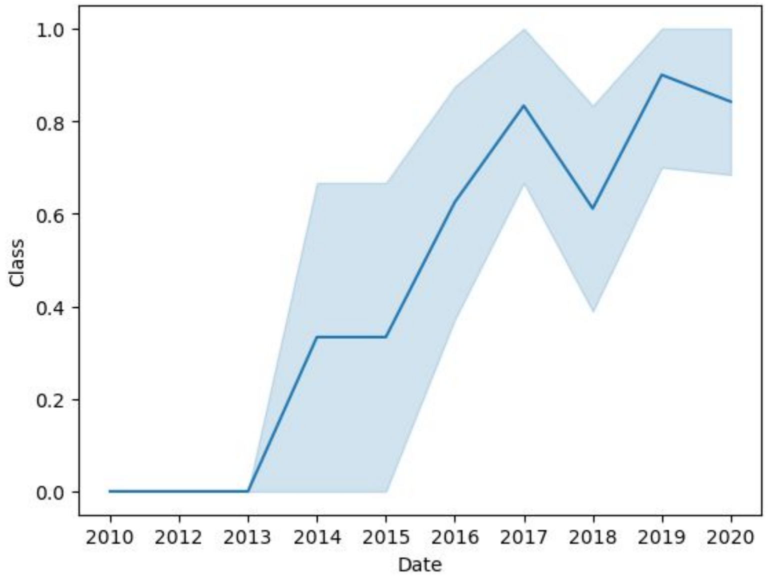
<AxesSubplot:xlabel='Orbit', ylabel='Class'>



Barplot between new_df and Orbit

```python
# Plot a line chart with x axis to be the extracted year and y axis to be the success rate
sns.lineplot(x="Date", y="Class", data=df)
```

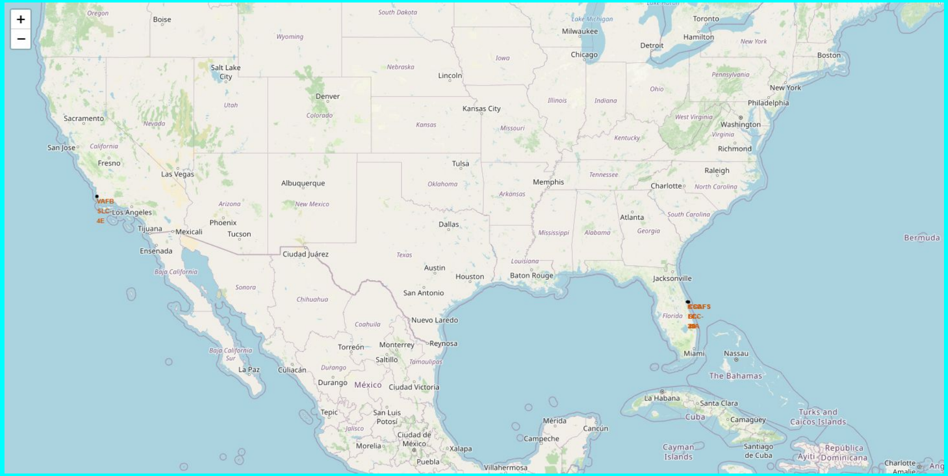
<AxesSubplot:xlabel='Date', ylabel='Class'>



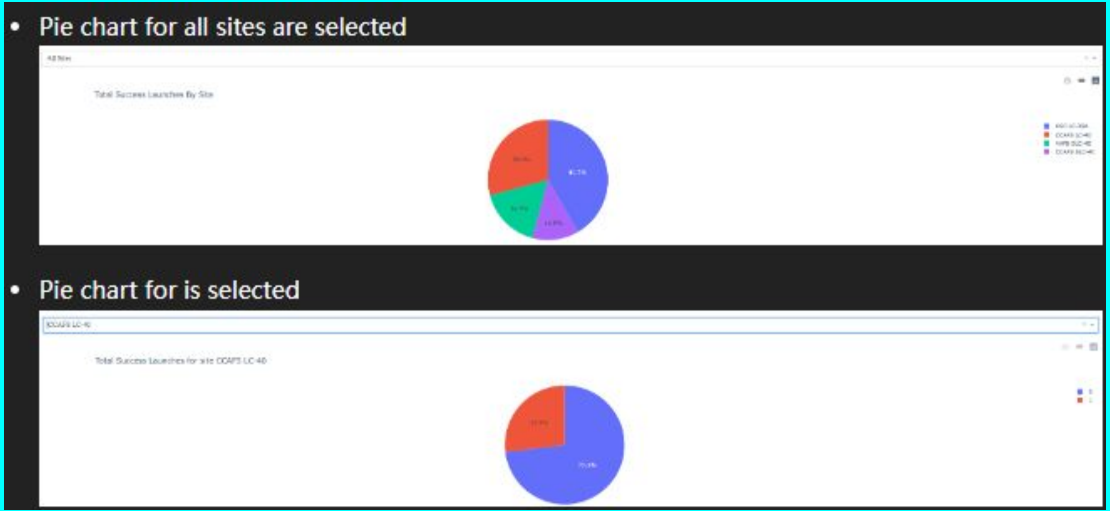
LinePlot between Date And Class

Folium And Plotly Dash

Interactive Map With Folium



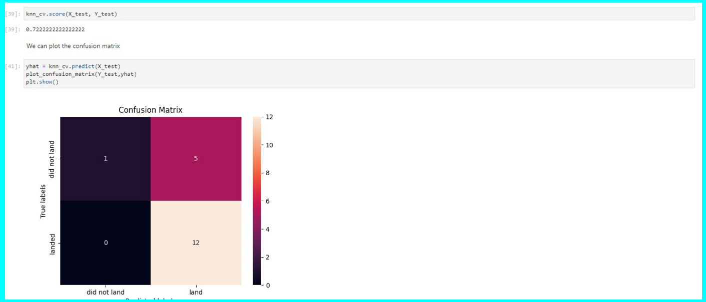
Plotly Dash Dashboard



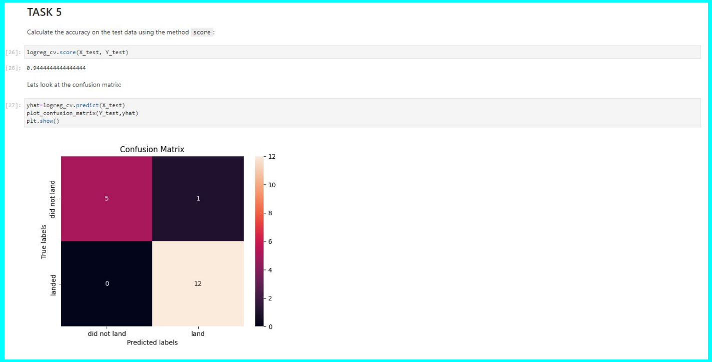
Predictive Analysis Classification

Predictive Analysis Classification

Decision Tree: Score And Confusion Matrix



KNN: Score And Confusion Matrix



Logistic Regression: Score And Confusion

Conclusion

We came up with a lot of interesting data and information from this experiment:

- The logistic regression has the best score, with the support vector machine maxing out.
- The locations of SPACEX are very close to the equator.
- The average success rate of the SSO object is the highest.
- There are three unique booster versions.