IBM Developer
SKILLS NETWORK

Winning Space Race with Data Science

Mikias Woldetensae
December 5, 2024

Outline

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

Executive Summary

- This project analyzes SpaceX's historical launch data to predict the success of future rocket landings.
- By leveraging classification models such as Decision Trees, K-Nearest Neighbors, Random Forests, and XGBoost, the study explores how machine learning can assist in improving operational efficiency.
- Results demonstrate consistent accuracy across models, highlighting the role of feature selection and data quality in predictive performance. As well as proving that model selection is an iterative process that requires careful exploration and consideration. In essence a more complex model is not guaranteed to give a better outcome.

Introduction

- SpaceX has revolutionized space travel by developing reusable rocket technology, significantly reducing launch costs. Predicting the success of rocket landings is critical for ensuring safety, optimizing operations, and advancing the goal of interplanetary travel.

- This project uses machine learning to analyze SpaceX's historical data, focusing on factors like payload, orbit type, and launch site to classify landing outcomes.

Objectives

1. Understand how different factors affect landing success.

2. Develop predictive models to estimate landing outcomes.

3. Evaluate and compare model performance across various algorithms.

Section 1

Methodology

Methodology

Logistic Regression:

- **Best Parameters**: `{'C': 0.01, 'penalty': 'l2', 'solver': 'lbfgs'}`
- **Validation Accuracy**: 84.64%
- **Test Set Accuracy**: 83.33%

Support Vector Machine (SVM):

- **Best Parameters**: `{'C': 1.0, 'gamma': 0.0316, 'kernel': 'sigmoid'}`
- **Validation Accuracy**: 84.82%
- **Test Set Accuracy**: 83.33%

Decision Tree Classifier:

- **Best Parameters**: `{'criterion': 'gini', 'max_depth': 4, 'max_features': 'sqrt', 'min_samples_leaf': 1, 'min_samples_split': 5, 'splitter': 'best'}`
- **Validation Accuracy**: 88.75%
- **Test Set Accuracy**: 83.33%

K-Nearest Neighbors (KNN):

- **Best Parameters**: `{'algorithm': 'auto', 'n_neighbors': 10, 'p': 1}`
- **Validation Accuracy**: 84.82%
- **Test Set Accuracy**: 83.33%

Methodology Part 2

Random Forest Classifier:

- **Best Parameters**: `{'bootstrap': True, 'criterion': 'gini', 'max_depth': 10, 'max_features': 'sqrt', 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 10}`
- **Validation Accuracy**: 87.5%
- **Test Set Accuracy**: 83.33%
- **Observation**: Despite significant computational effort, Random Forests yielded results comparable to simpler models.

XGBoost:

- **Best Parameters**: `{'colsample_bytree': 0.8, 'gamma': 0, 'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 100, 'reg_alpha': 0, 'reg_lambda': 1, 'subsample': 0.8}`
- **Validation Accuracy**: 86.25%
- **Test Set Accuracy**: 83.33%

General Observation:

- Across all models, the validation and test set accuracies consistently hovered around 83-84%. This suggests that adding model complexity does not necessarily improve performance, likely due to inherent data constraints.

Data Collection

- **API Integration**:
  - Used to retrieve SpaceX launch data programmatically.
- **Web Scraping**:
  - Extracted additional details from web pages related to SpaceX launches.
- **Manual Data Compilation**:
  - Supplemented missing or incomplete data points with manually curated information.
- **SQLite Database**:
  - Stored processed data in a structured format for easy querying and retrieval.

Data Collection – SpaceX API

- GitHub URL: https://github.com/MikiasHWT/ibm_cert/blob/main/SpaceX_Landing_Prediction.ipynb

Fetch Data

In [9]:

```python
# Static JSON URL in case of API failure
static_json_url='https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/datasets/API_

try:
    # Try to fetch data from the SpaceX API
    spacex_url = "https://api.spacexdata.com/v4/launches/past"  # Define SpaceX API endpoint
    response = requests.get(spacex_url)
    response.raise_for_status()  # Ensure the response is successful
    data = response.json()  # Decode the response content as JSON
    print("Successfully fetched data from the SpaceX API.")
except requests.RequestException:
    # Use the static JSON URL if the API call fails
    print("API call failed. Using static JSON data instead.")
    data = pd.read_json(static_json_url)

# Normalize the JSON data into a Pandas DataFrame
df = pd.json_normalize(data)
```

```
Successfully fetched data from the SpaceX API.
```

In [10]:

```python
# Data Dimensions
print(f"The dataset contains {df.shape[0]} observations and {df.shape[1]} features.\n")

# Broad information on features
print(df.info(), "\n")

# First 5 observations
df.head()
```

```
The dataset contains 187 observations and 43 features.

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 187 entries, 0 to 186
Data columns (total 43 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   static_fire_date_utc   121 non-null    object
 1   static_fire_date_unix  121 non-null    float64
 2   net                    187 non-null    bool
 3   window                 117 non-null    float64
 4   rocket                 187 non-null    object
 5   success                186 non-null    object
 6   failures               187 non-null    object
 7   details                134 non-null    object
 8   crew                   187 non-null    object
 9   ships                  187 non-null    object
```

Data Collection - Scraping

- GitHub URL: https://github.com/MikiasHWT/ibm_cert/blob/main/SpaceX_Landing_Prediction.ipynb

In [29]:
```python
# Define static Wiki page for consistency.
static_wiki_url = "https://en.wikipedia.org/w/index.php?title=List_of_Falcon_9_and_Falcon_Heavy_launches&oldid=1027686922"

# Check URL and inspect page content
```

In [30]:
```python
# Send a GET request to the page
response = requests.get(static_wiki_url)

# Check if the request was successful
if response.status_code == 200:
    print("Successfully fetched the webpage!")
else:
    print(f"Failed to fetch the webpage. Status code: {response.status_code}")
```

```
Successfully fetched the webpage!
```

In [31]:
```python
# Parse the page content
soup = BeautifulSoup(response.content, 'html.parser')

# Print page title
print(soup.title)
```

```
<title>List of Falcon 9 and Falcon Heavy launches - Wikipedia</title>
```

In [32]:
```python
# Find all tables in wiki page
html_tables = soup.find_all("table")
```

In [33]:
```python
# Note our table of interest is the 3rd element
launch_table = html_tables[2]

print(launch_table)
```

```
<table class="wikitable plainrowheaders collapsible" style="width: 100%;">
<tbody><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,<b
r/>Booster</a> <sup class="reference" id="cite_ref-booster_11-0"><a href="#cite_note-booster-11"><span class="cite-bracket">
[</span>b<span class="cite-bracket">]</span></a></sup>
</th>
```

Data Wrangling

Missing Value Handling:

- Imputed missing values with appropriate statistical methods or domain-specific assumptions.

Data Normalization:

- Scaled numeric variables to a consistent range to improve model performance.

Categorical Encoding:

- Converted categorical variables (e.g., launch site, orbit type) into numerical format using one-hot encoding.

Feature Filtering:

- Removed irrelevant columns and cleaned duplicate or redundant entries.

SQL Queries:

- Utilized SQL to filter, sort, and aggregate data for preprocessing.

- GitHub URL: https://github.com/MikiasHWT/ibm_cert/blob/main/SpaceX_Landing_Prediction.ipynb

Data Wrangling Part 2

- **Feature Extraction**:
  - Created new features from existing ones, such as binary success flags for landing outcomes.
- **Domain-Specific Feature Engineering**:
  - Derived meaningful variables like payload mass categories and specific orbit types.
- **Correlation Analysis**:
  - Used visualizations and statistical metrics to assess relationships between features and the target variable.
- **Dimensionality Reduction**:
  - Selected a subset of features based on their predictive power and relevance to the target variable.

- GitHub URL: https://github.com/MikiasHWT/ibm_cert/blob/main/SpaceX_Landing_Prediction.ipynb

EDA with Data Visualization

Objective: Understand the dataset's structure and key patterns through visual analysis.

Key Methods:

- Used `matplotlib` and `seaborn` for exploratory data analysis (EDA).
- Visualized distributions of payload masses, launch outcomes, and booster versions.
- Correlation heatmaps revealed relationships between features, aiding feature selection.
- Bar plots and pie charts highlighted successful vs. failed launches across different launch sites.

- GitHub URL: https://github.com/MikiasHWT/ibm_cert/blob/main/SpaceX_Landing_Prediction.ipynb

EDA with SQL

Objective: Leverage SQL queries to analyze SpaceX data efficiently.

Key Methods:

- Queried a SQLite database containing SpaceX launch data.
- Aggregated launch outcomes by site and booster type using `GROUP BY`.
- Used SQL `JOIN` operations to combine tables and uncover launch trends.
- Extracted payload ranges and their corresponding success rates with conditional queries.

- GitHub URL: https://github.com/MikiasHWT/ibm_cert/blob/main/SpaceX_Landing_Prediction.ipynb

Build an Interactive Map with Folium

Objective: Provide a geographical perspective on SpaceX launch sites and outcomes.

Key Methods:

- Used the `folium` library to create an interactive map.
- Plotted launch sites with markers, color-coded by success rate.
- Added pop-ups displaying site-specific details like total launches and success percentages.
- Enhanced usability by enabling zoom and hover interactions to explore spatial patterns.

- GitHub URL: https://github.com/MikiasHWT/ibm_cert/blob/main/SpaceX_Landing_Prediction.ipynb

Build a Dashboard with Plotly Dash

Objective: Create an interactive dashboard for real-time exploration of launch records.

Key Features:

- **Dropdown Selector**:
  - Allows users to filter data by specific launch sites or view all sites together.
- **Pie Chart**:
  - Displays the proportion of successful and failed launches for selected sites.
- **Payload Range Slider**:
  - Filters data by payload mass range, dynamically updating visualizations.
- **Scatter Plot**:
  - Shows correlations between payload mass and success rate, segmented by booster version.
- **Interactivity**:
  - Users can adjust parameters to visualize specific trends in launch outcomes.
- Built using `Dash` with `Plotly` for seamless interactivity and modern visuals.

- GitHub URL: https://github.com/MikiasHWT/ibm_cert/blob/main/spacex_dash_app_Final.py

Predictive Analysis (Classification)

Logistic Regression:

- **Best Parameters**: `{'C': 0.01, 'penalty': 'l2', 'solver': 'lbfgs'}`
- **Validation Accuracy**: 84.64%
- **Test Set Accuracy**: 83.33%

Support Vector Machine (SVM):

- **Best Parameters**: `{'C': 1.0, 'gamma': 0.0316, 'kernel': 'sigmoid'}`
- **Validation Accuracy**: 84.82%
- **Test Set Accuracy**: 83.33%

Decision Tree Classifier:

- **Best Parameters**: `{'criterion': 'gini', 'max_depth': 4, 'max_features': 'sqrt', 'min_samples_leaf': 1, 'min_samples_split': 5, 'splitter': 'best'}`
- **Validation Accuracy**: 88.75%
- **Test Set Accuracy**: 83.33%

K-Nearest Neighbors (KNN):

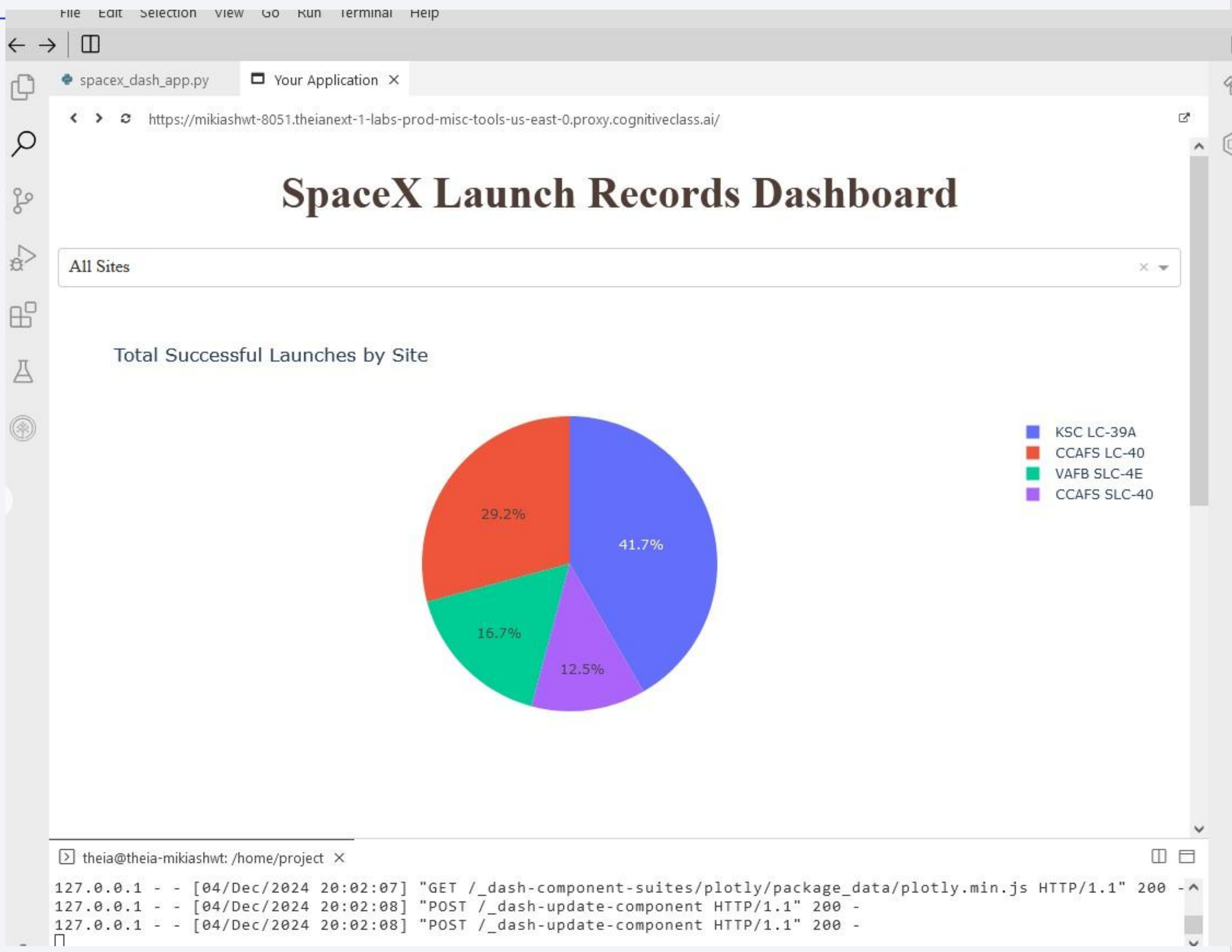
- **Best Parameters**: `{'algorithm': 'auto', 'n_neighbors': 10, 'p': 1}`
- **Validation Accuracy**: 84.82%
- **Test Set Accuracy**: 83.33%

GitHub URL: https://github.com/MikiasHWT/ibm_cert/blob/main/SpaceX_Landing_Prediction.ipynb

Results

- **General Observation**:
- Across all models, the validation and test set accuracies consistently hovered around 83-84%. This suggests that adding model complexity does not necessarily improve performance, likely due to inherent data constraints.

- GitHub URL: https://github.com/MikiasHWT/ibm_cert/blob/main/spacex_dash_app_Final.py
- GitHub URL: https://github.com/MikiasHWT/ibm_cert/blob/main/SpaceX_Landing_Prediction.ipynb



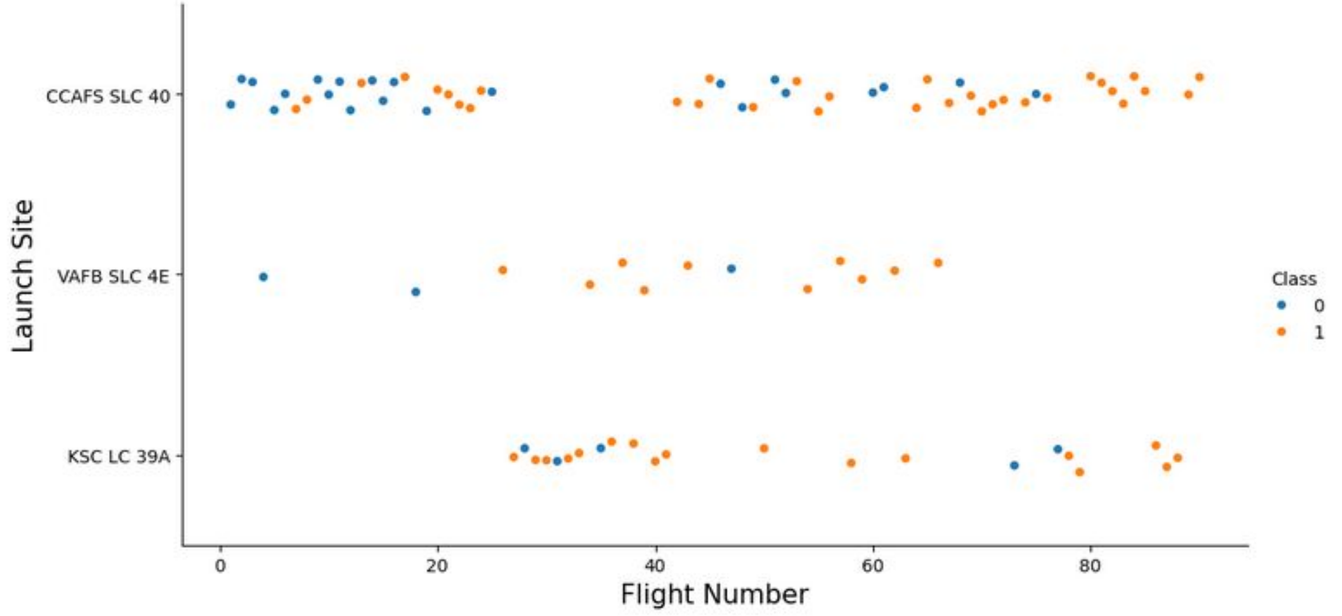
Section 2

Insights drawn from EDA

Flight Number vs. Launch Site

- With the combination of the plot and the exploratory code, we can tell that the number of launches vary by site. Additionally it seems that CCAFS site has a lower success rate than the two other launch sites. This may be attributed to the fact that there have been many more launches from that site than the two others

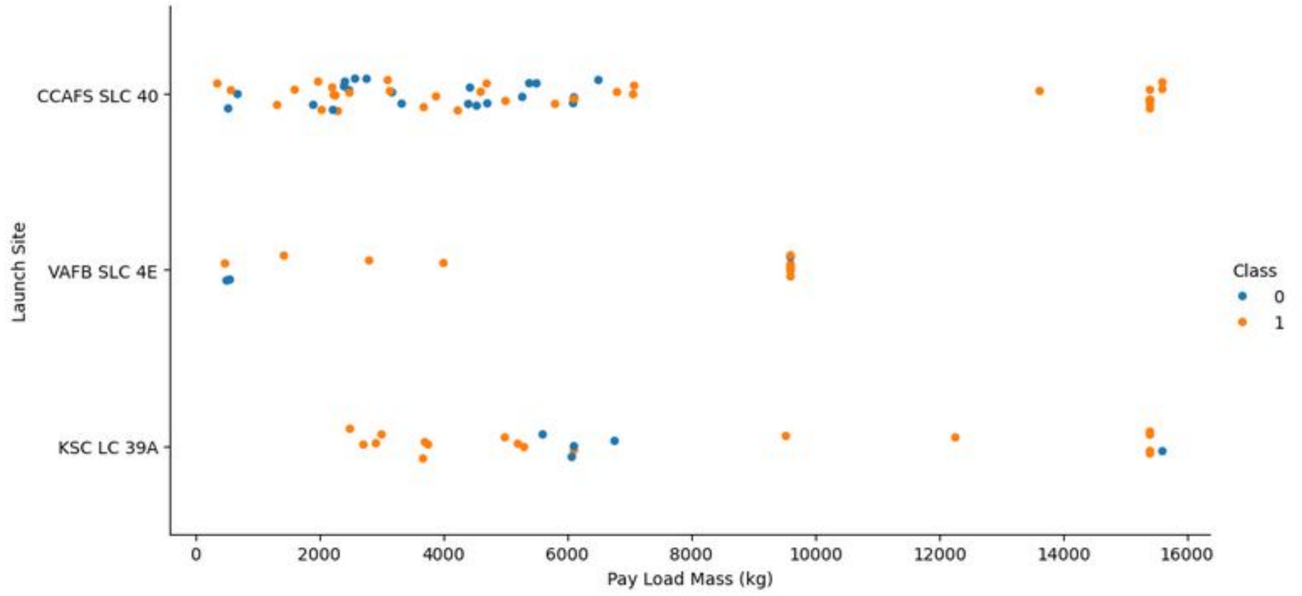
```
# Categorical plot of FlightNumber, Launchsite and Outcome
sns.catplot(df5, x="FlightNumber", y="LaunchSite", hue="Class", aspect=2)
plt.xlabel("Flight Number", fontsize=15)
plt.ylabel("Launch Site", fontsize=15)
plt.show();
```



Payload vs. Launch Site

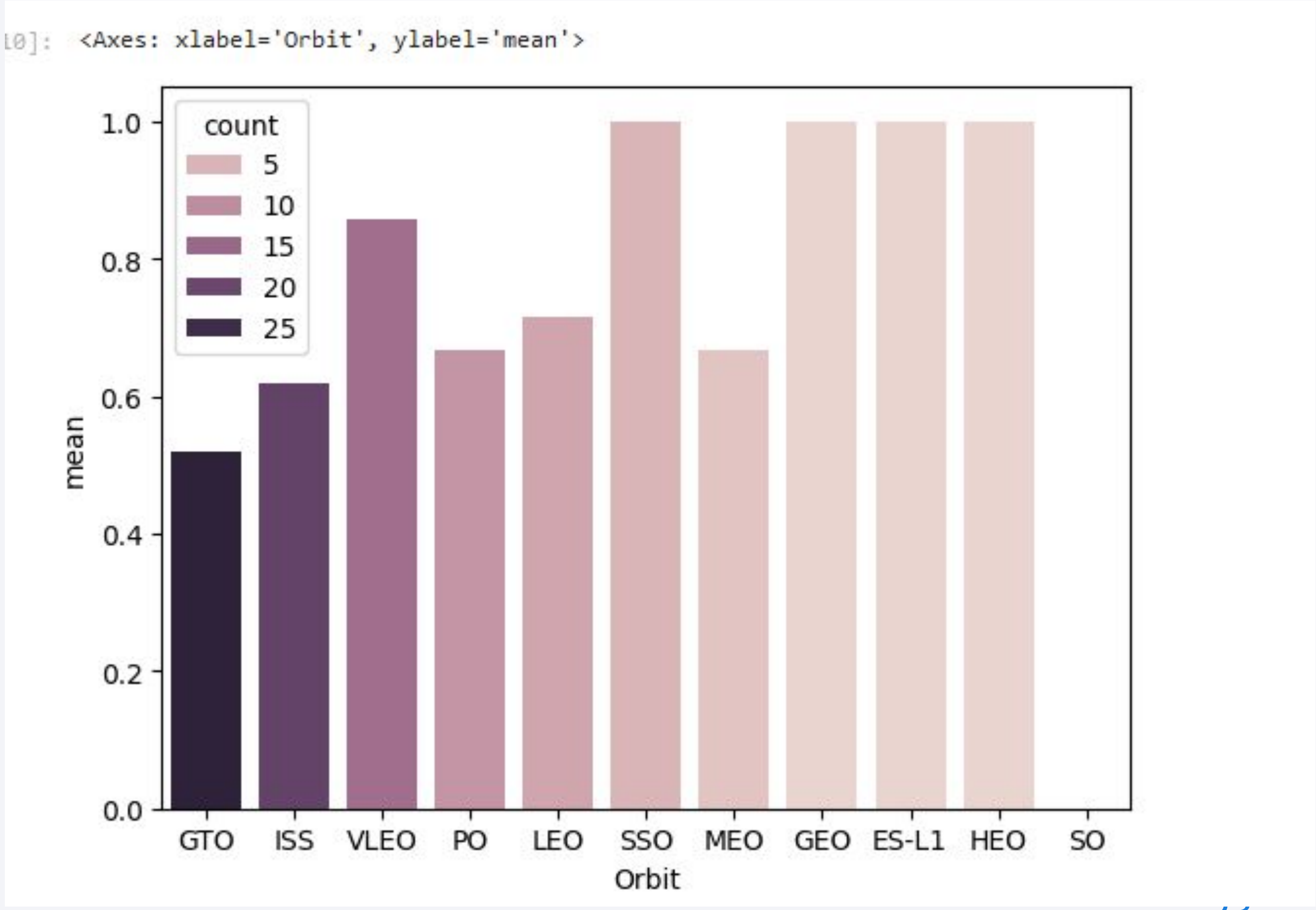
- There appears to be a 10,000kg limit on launches from VAFB SLC 4E Launch site.

```
# Visualize Payload Mass by Launch Site
sns.catplot(df5, x="PayloadMass", y="LaunchSite", hue="Class", aspect=2)
plt.xlabel("Pay Load Mass (kg)")
plt.ylabel("Launch Site")
plt.show();
```



Success Rate vs. Orbit Type

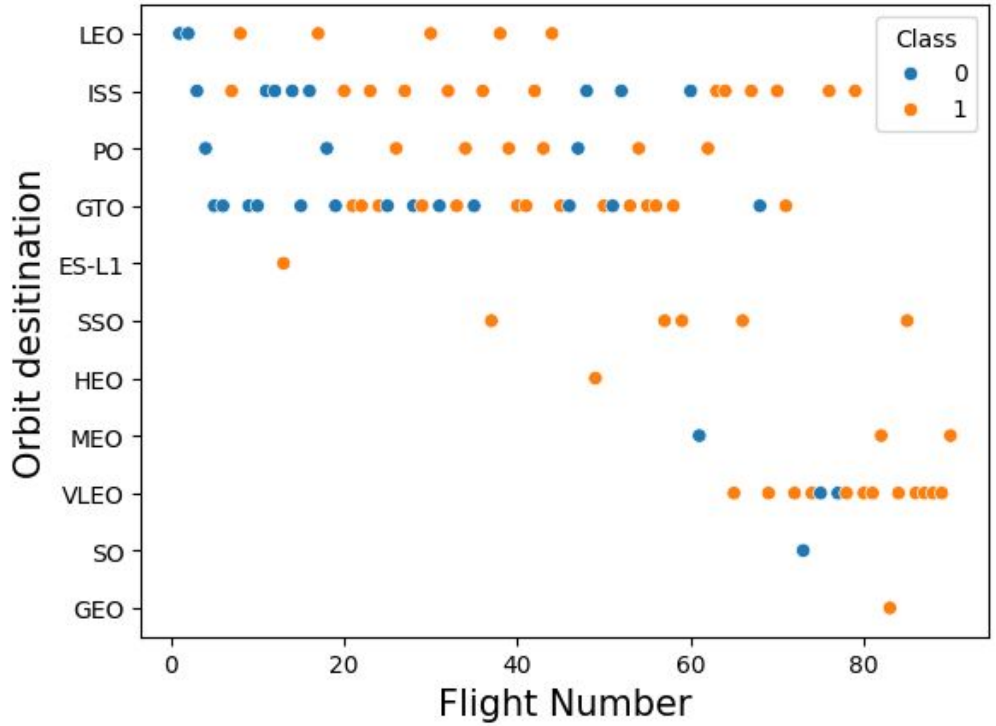
- We can see that as the attempts for a certain orbital destination increase, its success rate generally drops. That being said, VLEO appears to have retained a fairly high success rate at 85% with 14 attempts made.



Flight Number vs. Orbit Type

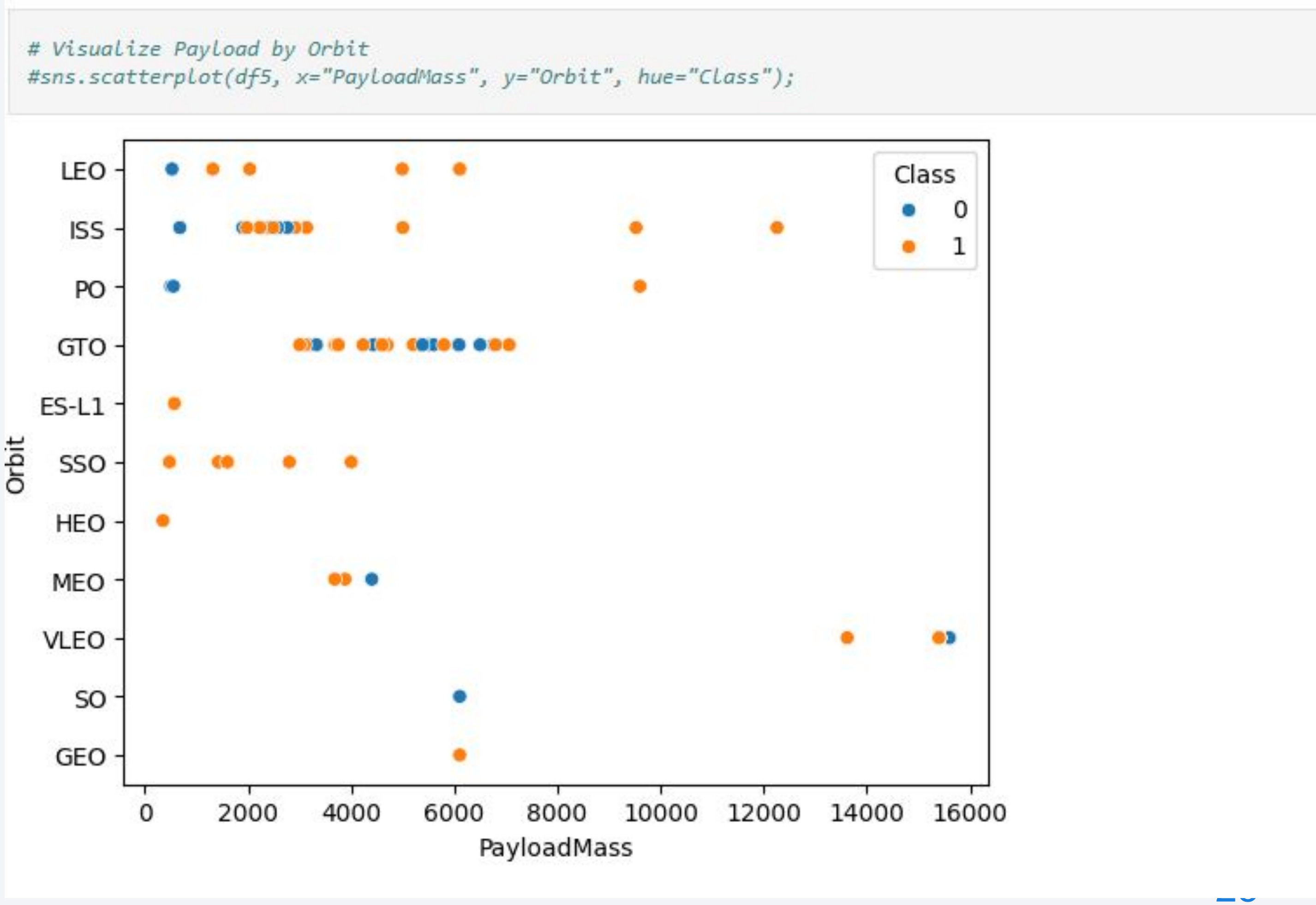
- There appears to be a increase in success rate for some of the orbit desintations such as LEO, while other destinations appear to experiance the periodic failure despite number of attempts. There is also a pattern of diversifying orbit destinations over time.

```python
# Visualize FlightNumber and Orbit
sns.scatterplot(df5, x="FlightNumber", y="Orbit", hue="Class")
plt.xlabel("Flight Number", fontsize=15)
plt.ylabel("Orbit desitination", fontsize=15)
plt.show();
```



Payload vs. Orbit Type

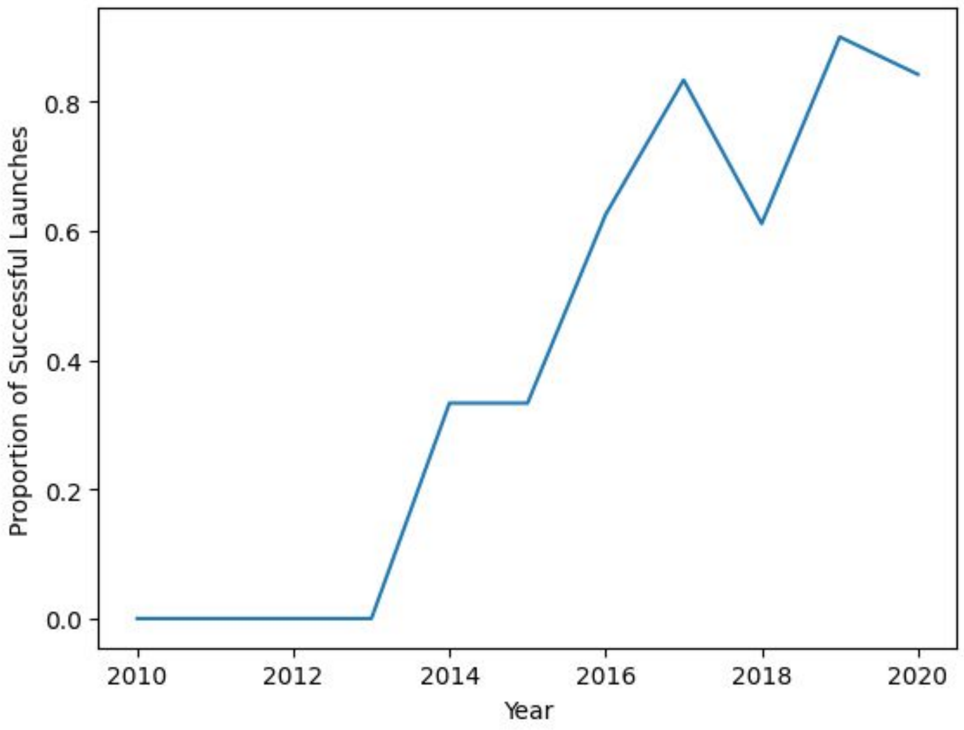
- GTO had many launch attempts with medium payload. Some orbits are only attempted with light payloads while VLEO has been attempted with the highest payloads.



Launch Success Yearly Trend

- We can see a increase in success rate starting in 2013. Some stable period and reduction in successful attempts are seen later on.

```
# Lineplot
sns.lineplot(yearly_success, x="Year", y="Class")
plt.ylabel("Proportion of Successful Launches")
plt.show();
```



All Launch Site Names

- Select all distinct launch sites from table

```
]: # Display unique launch sites
   %sql SELECT DISTINCT Launch_Site FROM SPACEXTABLE
```

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

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

Launch Site Names Begin with 'CCA'

- Select all launch sites beginning with CCA, but limit the output to 5

```
# Display 5 records where launch site begins with 'CCA'
%sql SELECT * FROM SPACEXTABLE WHERE Launch_Site LIKE 'CCA%' LIMIT 5
```

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

Date	Time (UTC)	Booster_Version	Launch_Site	Payload	PAYLOAD_MASS__KG_	Orbit	Customer	Mission_Ou
2010-06-04	18:45:00	F9 v1.0 B0003	CCAFS LC-40	Dragon Spacecraft Qualification Unit	0	LEO	SpaceX	S
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	S
2012-05-22	7:44:00	F9 v1.0 B0005	CCAFS LC-40	Dragon demo flight C2	525	LEO (ISS)	NASA (COTS)	S
2012-10-08	0:35:00	F9 v1.0 B0006	CCAFS LC-40	SpaceX CRS-1	500	LEO (ISS)	NASA (CRS)	S
2013-03-01	15:10:00	F9 v1.0 B0007	CCAFS LC-40	SpaceX CRS-2	677	LEO (ISS)	NASA (CRS)	S

Total Payload Mass

- Sum the payloads from NASA (CRS) and rename the newly aggregated variable.

```
# Display the total payload mass carried by boosters launched by NASA (CRS)
%sql SELECT SUM(PAYLOAD_MASS__KG_) AS Total_Payload_Mass FROM SPACEXTABLE WHERE Customer = 'NASA (CRS)'
```

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

Total_Payload_Mass
45596

Average Payload Mass by F9 v1.1

- Average the payloads from Booster versions like “F9…” and rename the newly aggregated variable.

In [57]:
```
# Display average payload mass carried by booster version F9 v1.1
%sql SELECT AVG(PAYLOAD_MASS__KG_) AS Average_Payload_Mass FROM SPACEXTABLE WHERE Booster_Version LIKE 'F9
```

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

Out[57]:

Average_Payload_Mass
2534.6666666666665

First Successful Ground Landing Date

- Select successful landing outcomes that equal a predefined string, limit output to 1.

```
# List the date when the first successfull landing outcome in ground pad was achieved
%sql SELECT Date FROM SPACEXTABLE WHERE Landing_Outcome = 'Success (ground pad)' LIMIT 1
```

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

Date
2015-12-22

Successful Drone Ship Landing with Payload between 4000 and 6000

- List the names of boosters which have successfully landed on drone ship and had payload mass greater than 4000 but less than 6000

```
# List the names of the booster which have sucess in drone ship and had a payload mass greater than 4000 bu
%sql SELECT Booster_Version FROM SPACEXTABLE WHERE Landing_Outcome = 'Success (drone ship)' AND PAYLOAD_MAS
```

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

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

Total Number of Successful and Failure Mission Outcomes

- Calculate the total number of successful and failure mission outcomes

```
# List the total number of success and failure mission outcomes
%sql SELECT Mission_Outcome, COUNT(*) AS Total_Missions FROM SPACEXTABLE GROUP BY Mission_Outcome
```

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

Mission_Outcome	Total_Missions
Failure (in flight)	1
Success	98
Success	1
Success (payload status unclear)	1

Boosters Carried Maximum Payload

- List the names of the booster which have carried the maximum payload mass using a subquery

```
In [61]:  # List the names of the booster_versions which have carried the maximum payload mass (Using a subquery)
          %sql SELECT Booster_Version FROM SPACEXTABLE WHERE PAYLOAD_MASS__KG_ = (SELECT MAX(PAYLOAD_MASS__KG_) FROM

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

Out[61]:

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

2015 Launch Records

- List the failed landing_outcomes in drone ship, their booster versions, and launch site names for in year 2015

```
In [62]:  # List the records which will display the month names, failure landing_outcomes in drone ships, booster ver

          %sql SELECT strftime('%m', DATE) AS Month_Number, strftime('%Y', DATE) AS Year, Landing_Outcome, Booster_Ve

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

Out[62]:

Month_Number	Year	Landing_Outcome	Booster_Version	Launch_Site
01	2015	Failure (drone ship)	F9 v1.1 B1012	CCAFS LC-40
04	2015	Failure (drone ship)	F9 v1.1 B1015	CCAFS LC-40

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

- 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

In [63]:

```
# Rank the count of landing outcomes (such as Failure (drone ship) or Success (ground pad)) between the dat
%sql SELECT Landing_Outcome, COUNT(*) AS Outcome_Count FROM SPACEXTABLE WHERE DATE BETWEEN '2010-06-04' AND
```

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

Out[63]:

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

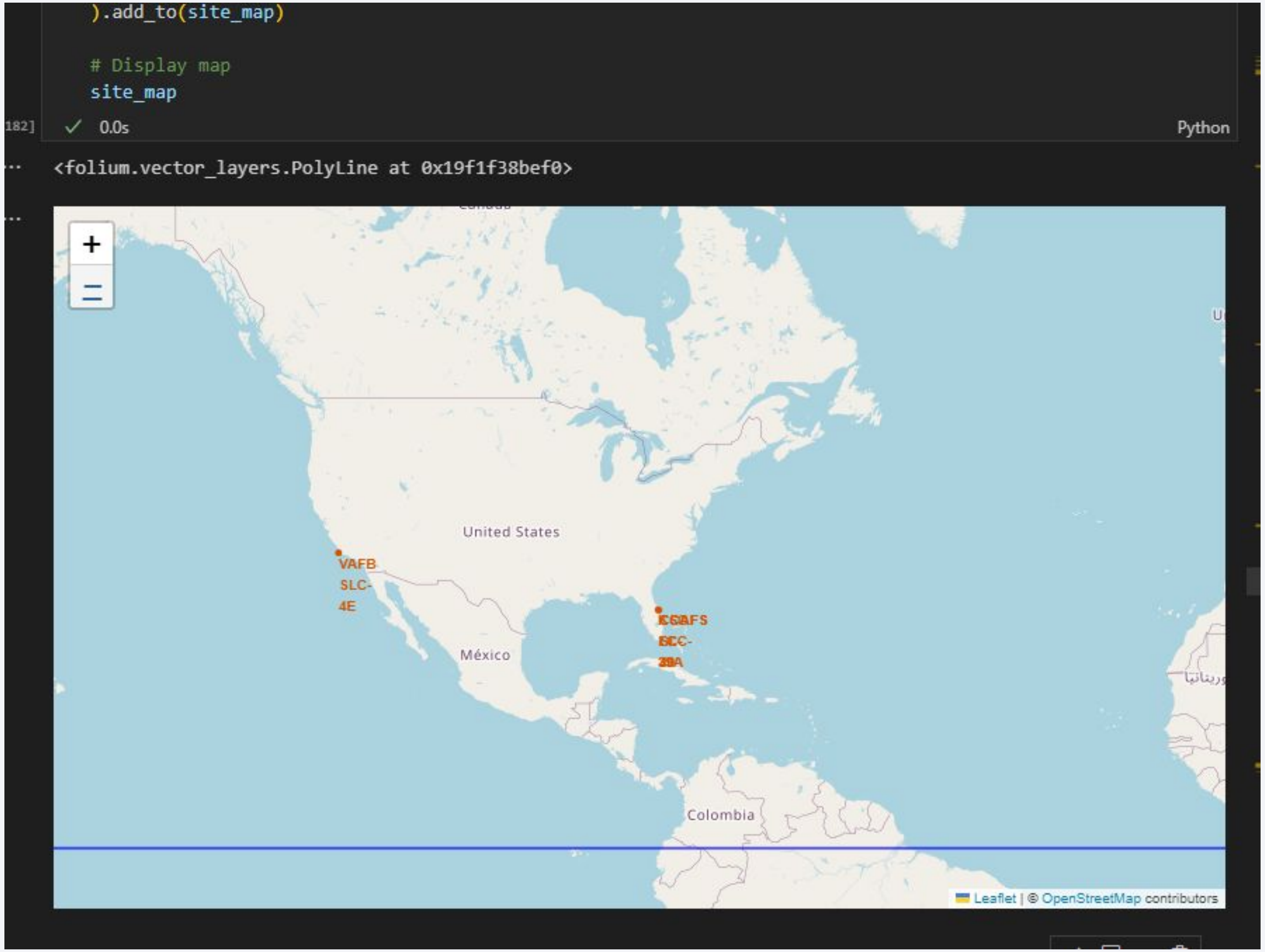
In [ ]:

Section 3

Launch Sites Proximities Analysis

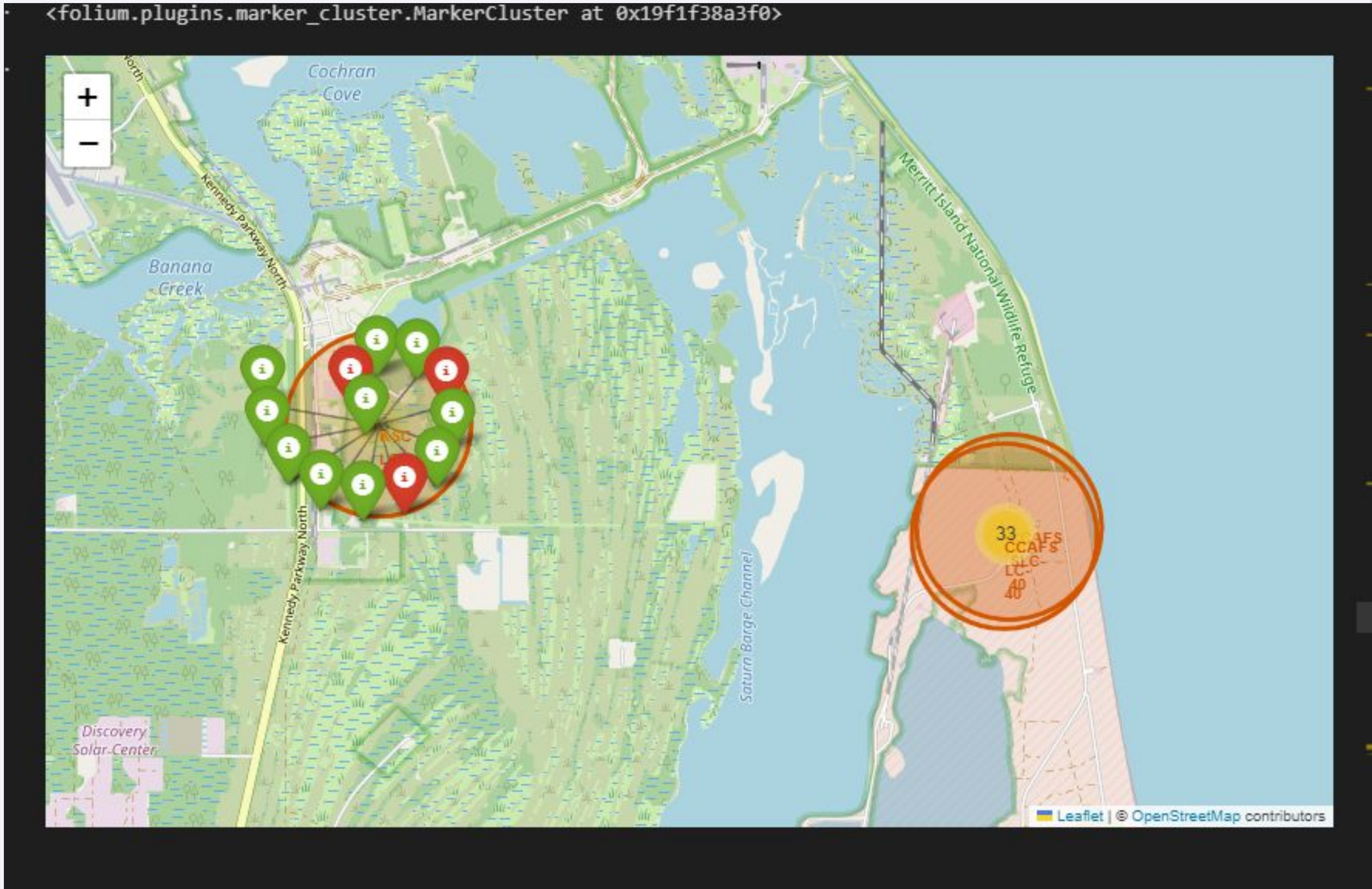
Launch Sites

From the map we can tell that all the launch sites are very close to the coast, and seem to favor the american coasts that are closest to the equator.



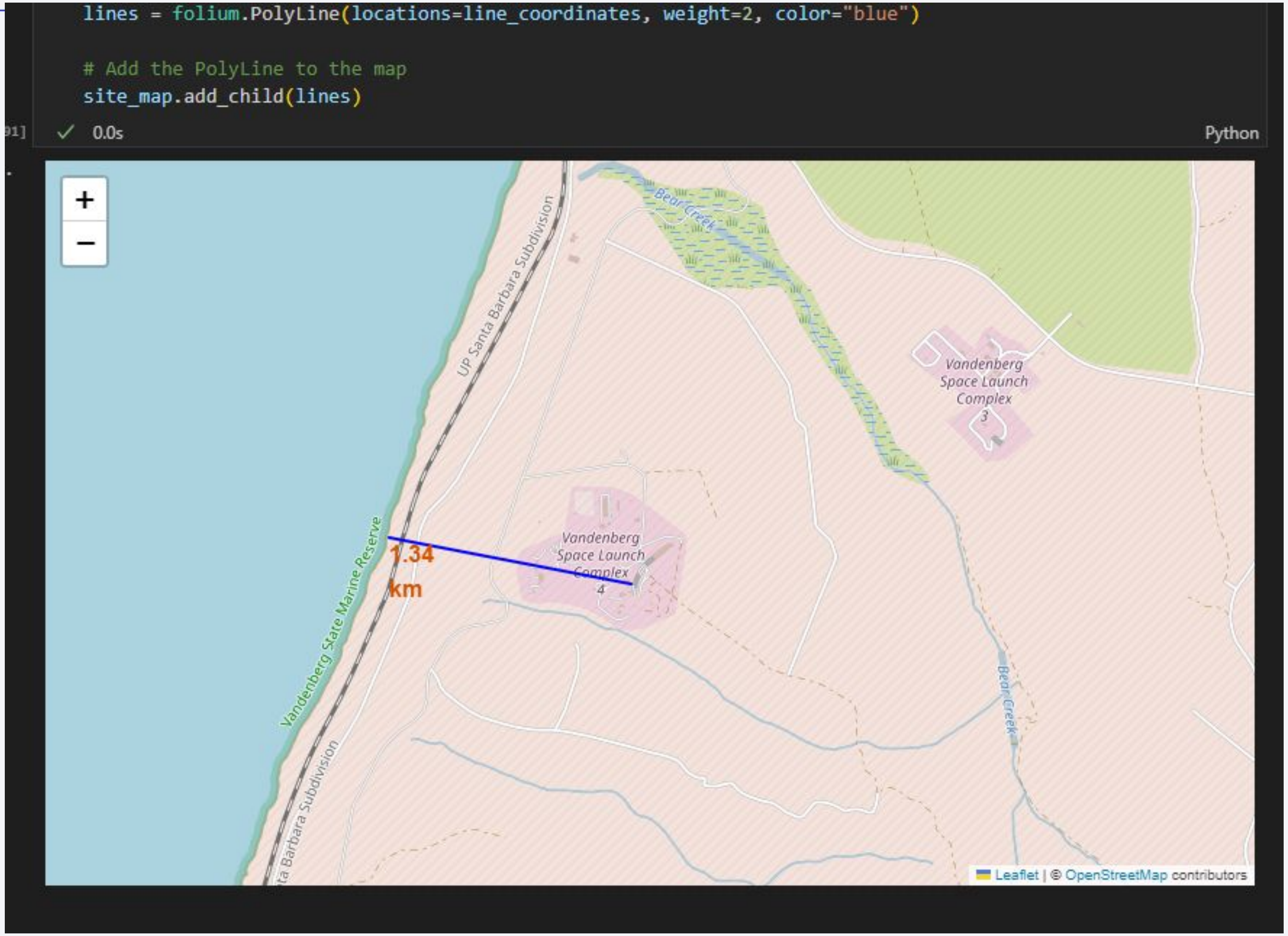
Launch Outcomes map

- We can see a variety of successful and failed launch attempts.



<Folium Map Screenshot 3>

- All launch sites are close to at least one railroad and/or freeway but a good distance from populated area’s such as cities. They are also all very close to the coast.

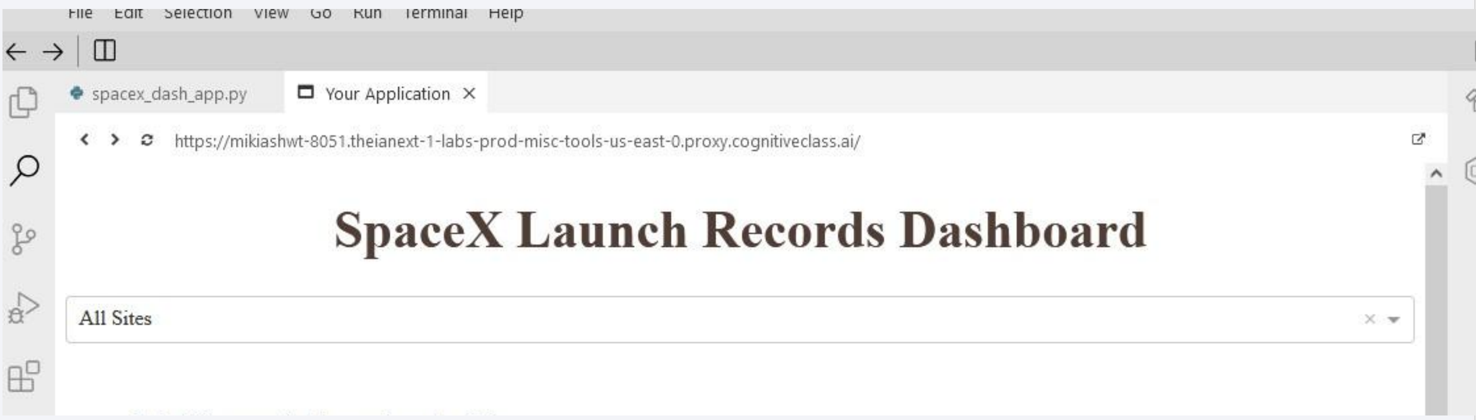


Section 4

Build a Dashboard with Plotly Dash

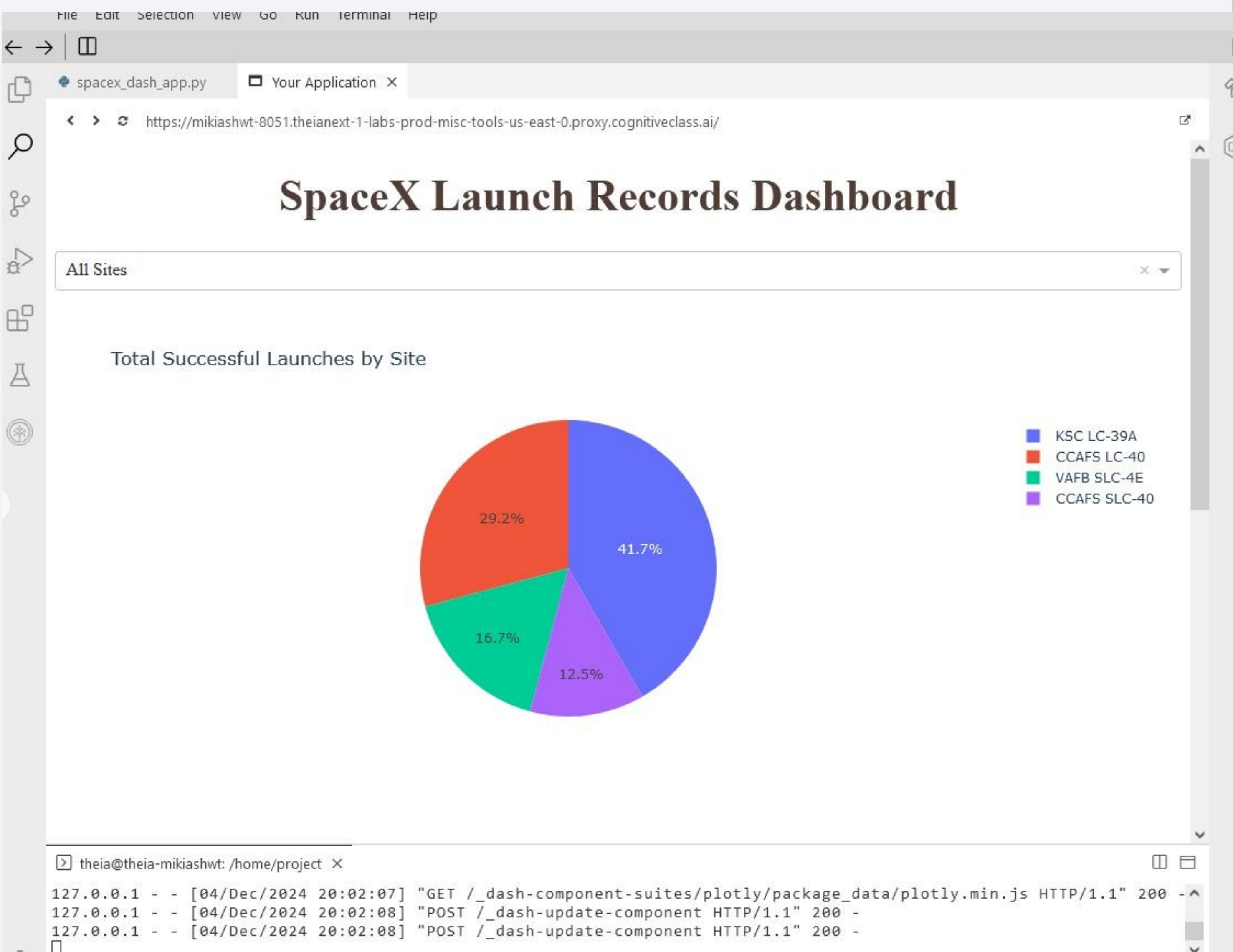
DropDown Dashboard

- Title Screen showing a useable dropdown menu option



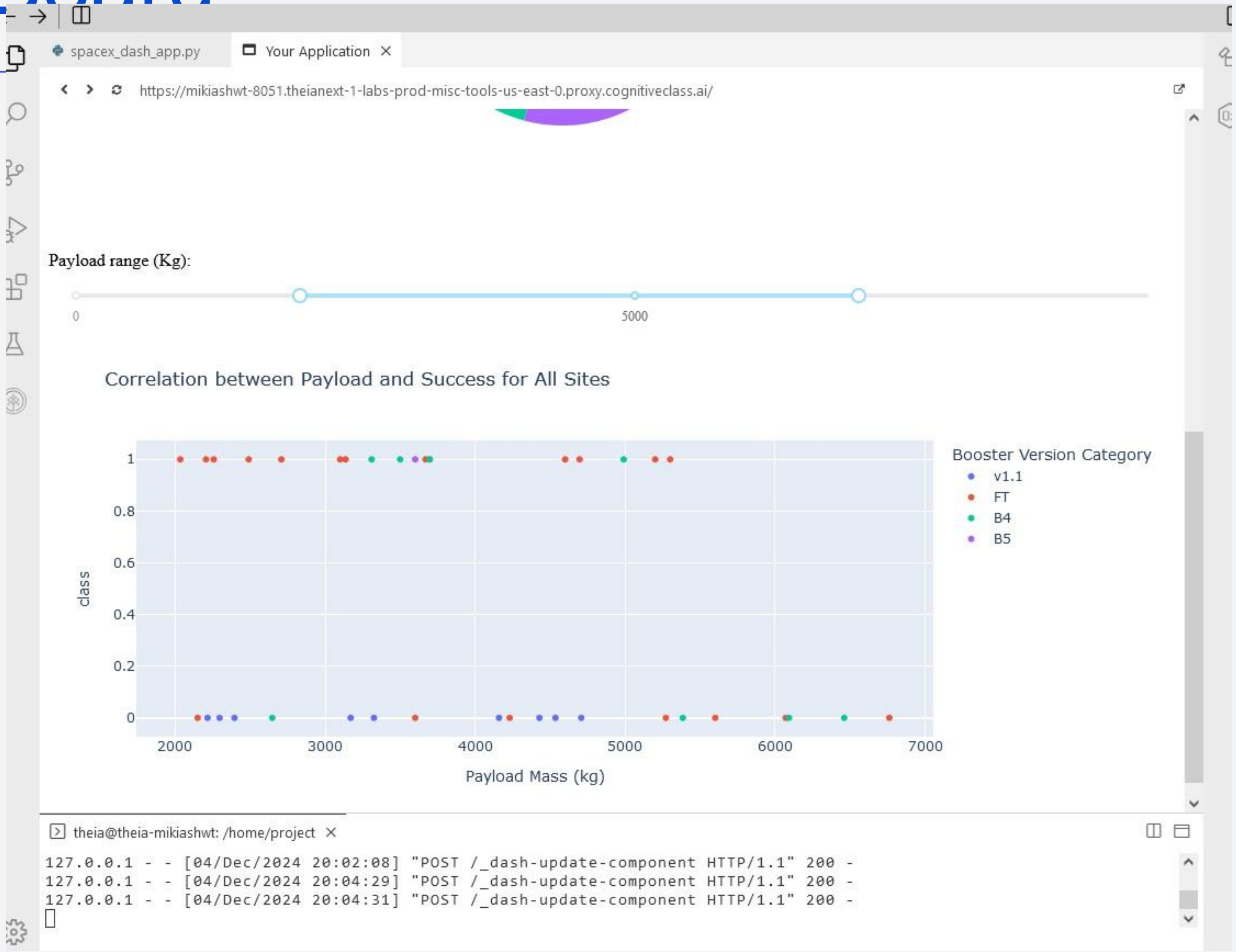
Pie Chart DashBoard

- Responsive element outputs a piechart when prompted



Scatterplot DashBoard

- Responsive slides and output of scatterplot when prompted by user



Section 5

Predictive Analysis (Classification)

Confusion Matrix

- Show the confusion matrix of the best performing model with an explanation

Conclusions

- **Discoveries**:
  - There was no significant improvement in accuracy despite using more complex models.
- **Future Directions**:
  - Collect newer data from recent years.
  - Return to feature engineering.
  - Apply metrics to determine most influential variables.
  - Verify that features are not confounded.

Appendix

1. **GitHub Repository**: Contains all code and data files for the project, including preprocessing, EDA, and model training. Link
2. **Jupyter Notebook**:
   - Provides a detailed walkthrough of all steps taken during the analysis.
   - Includes visualizations, code snippets, and model performance metrics.
3. **Key Libraries Used**:
   - `scikit-learn` for model building and evaluation.
   - `pandas` and `numpy` for data manipulation.
   - `matplotlib` and `seaborn` for visualizations.
   - `xgboost` for advanced gradient boosting.
4. **Session Information**:
   - Python version and library versions are documented in the notebook for reproducibility.

Thank you!