IBM Developer
SKILLS NETWORK

Winning Space Race with Data Science

Arthur Paruzel
10/28/2022

Outline

Executive Summary

- Summary of methodologies
    - Data Collection and Aggregation using API
    - Data Collection and Aggregation using web scraping
    - Data Wrangling
    - Exploratory Data Analysis using SQL and IBM DB2
    - Exploratory Data Analysis with Python and Matplotlib/Seaborn
    - Interactive Visual Analytics with Folium
    - Interactive Visual Analytics with Python and Plotly
    - Machine Learning Modelling and Prediction
        - Supervised machine learning algorithms: K-Nearest Neighbors, Tree, Supported Vector Machine, Logistic Regression
        - Model Optimization using Grid Search and scoring: F1
    - Summary of all results
    - Exploratory Data Analysis Results
    - Interactive Visual Analysis Results
    - Predictive Analysis Results

Introduction

- Project background and context

Space X advertises Falcon 9 rocket launches on its website with a cost of 62 million dollars; other providers cost upward of 165 million dollars each, much of the savings is because Space X can reuse the first stage. Therefore, if we can determine if the first stage will land, we can determine the cost of a launch. This information can be used if an alternate company wants to bid against space X for a rocket launch. This goal of the project is to create a machine learning pipeline to predict if the first stage will land successfully.

- Problems you want to find answers

- What factors determine if the rocket will land successfully?
- The interaction amongst various features that determine the success rate of a successful landing.
- What operating conditions needs to be in place to ensure a successful landing program.

Section 1

Methodology

Methodology

Executive Summary

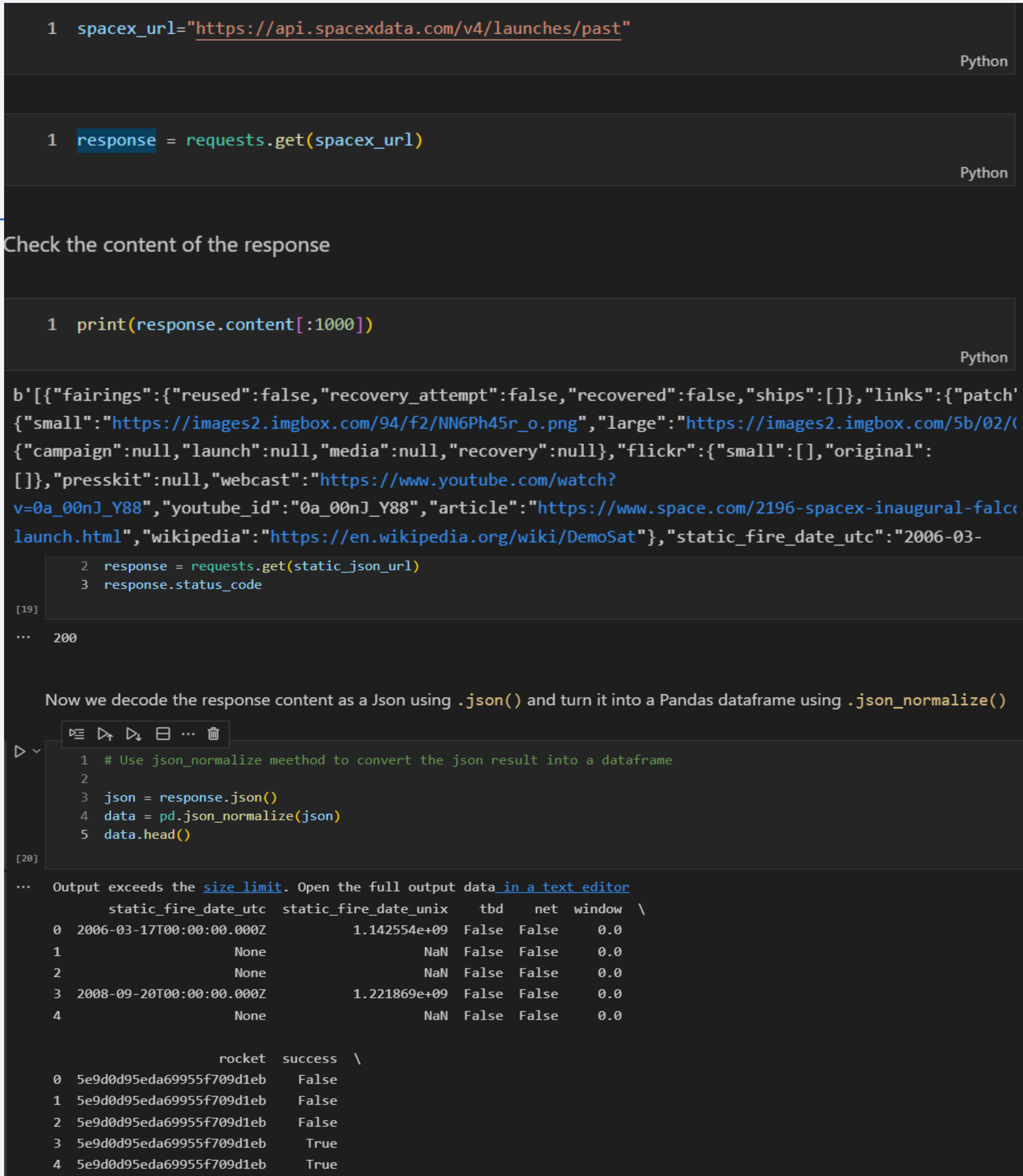
- Data collection methodology:
  - Data was collected using SpaceX API and web scraping from Wikipedia
- Data wrangling
  - One-hot encoding was applied to categorical features to standardize data for predictive modeling.
- Exploratory Data Analysis (EDA) using visualization and SQL
- Interactive visual analytics using Folium and Plotly Dash
- Predictive analysis using classification models
  - 4 classifiers were compared to determine best fitted for the Predictive Analysis.
  - All 4 classifiers were built, trained and then tested on the available datasets
  - For each model type multiple parameters were evaluated to fine tune the model to the best results.
  - Goal was to to determine the model with highest prediction accuracy.

Data Collection

- The data was collected using various methods
  - Data collection was done using get request to the SpaceX API.
  - Next, we decoded the response content as a Json using .json() function call and turn it into a pandas dataframe using .json_normalize().
  - We then cleaned the data, checked for missing values and fill in missing values where necessary.
  - In addition, we performed web scraping from Wikipedia for Falcon 9 launch records with BeautifulSoup.
  - The objective was to extract the launch records as HTML table, parse the table and convert it to a pandas dataframe for future analysis.

Data Collection – SpaceX API

- SpaceX API was used to collect data.
- Data was cleaned, normalized, and some basic wrangling was performed.
- The link to the jupyter notebook with API https://github.com/ArthurParuzel/SpaceXDataScience/blob/main/1.%20Data%20Collection%20API%20Web%20Scraping/jupyter-labs-spacex-data-collection-api.ipynb



```
spacex_url="https://api.spacexdata.com/v4/launches/past"
```

Python

```
response = requests.get(spacex_url)
```

Python

Check the content of the response

```
print(response.content[:1000])
```

Python

```
b'[{"fairings":{"reused":false,"recovery_attempt":false,"recovered":false,"ships":[]},"links":{"patch'
{"small":"https://images2.imgbox.com/94/f2/NN6Ph45r_o.png","large":"https://images2.imgbox.com/5b/02/(
{"campaign":null,"launch":null,"media":null,"recovery":null},"flickr":{"small":[],"original":
[]},"presskit":null,"webcast":"https://www.youtube.com/watch?
v=0a_00nJ_Y88","youtube_id":"0a_00nJ_Y88","article":"https://www.space.com/2196-spacex-inaugural-falc
launch.html","wikipedia":"https://en.wikipedia.org/wiki/DemoSat"},"static_fire_date_utc":"2006-03-
```

```
response = requests.get(static_json_url)
response.status_code
```

[19]

```
200
```

Now we decode the response content as a Json using `.json()` and turn it into a Pandas dataframe using `.json_normalize()`

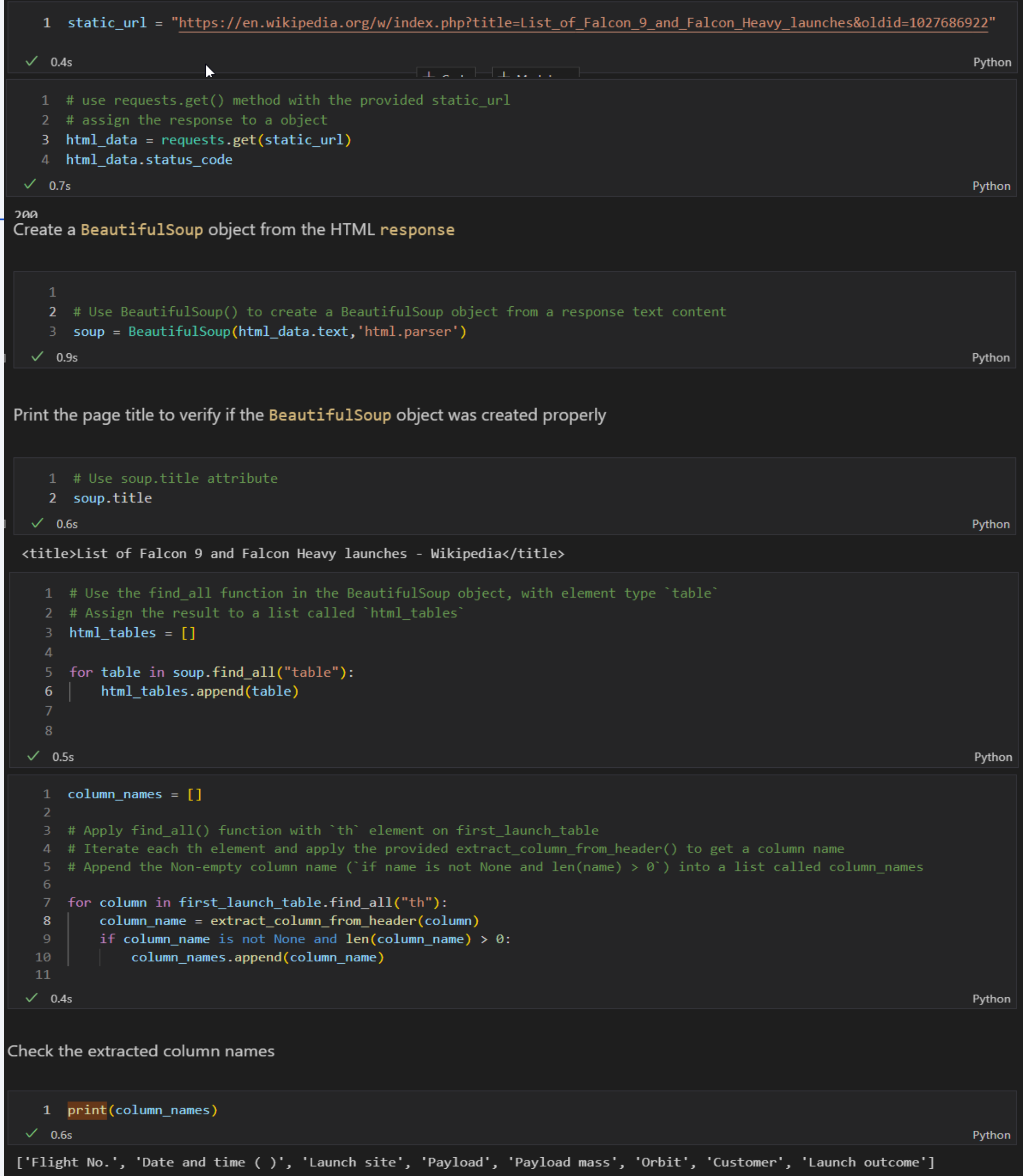
```
# Use json_normalize meethod to convert the json result into a dataframe

json = response.json()
data = pd.json_normalize(json)
data.head()
```

[20]

```
Output exceeds the size limit. Open the full output data in a text editor
       static_fire_date_utc  static_fire_date_unix    tbd    net  window  \
0  2006-03-17T00:00:00.000Z           1.142554e+09  False  False     0.0
1                      None                    NaN  False  False     0.0
2                      None                    NaN  False  False     0.0
3  2008-09-20T00:00:00.000Z           1.221869e+09  False  False     0.0
4                      None                    NaN  False  False     0.0

                     rocket  success  \
0  5e9d0d95eda69955f709d1eb    False
1  5e9d0d95eda69955f709d1eb    False
2  5e9d0d95eda69955f709d1eb    False
3  5e9d0d95eda69955f709d1eb     True
4  5e9d0d95eda69955f709d1eb     True
```

Data Collection - Scraping

- Data was web scrapped from Wikipedia website using BeautifulSoup library.
- HTML table was parsed and converted into data frame for further analysis.
- The link to the Jupiter notebook with Web Scraping https://github.com/ArthurParuzel/SpaceXDataScience/blob/main/1.%20Data%20Collection%20API%20Web%20Scraping/jupyter-labs-webscraping.ipynb



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

✓ 0.4s Python

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

✓ 0.7s Python

```
200
```

Create a `BeautifulSoup` object from the HTML `response`

```

# Use BeautifulSoup() to create a BeautifulSoup object from a response text content
soup = BeautifulSoup(html_data.text,'html.parser')
```

✓ 0.9s Python

Print the page title to verify if the `BeautifulSoup` object was created properly

```
# Use soup.title attribute
soup.title
```

✓ 0.6s Python

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

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

for table in soup.find_all("table"):
    html_tables.append(table)


```

✓ 0.5s Python

```
column_names = []

# Apply find_all() function with `th` element on first_launch_table
# Iterate each th element and apply the provided extract_column_from_header() to get a column name
# Append the Non-empty column name (`if name is not None and len(name) > 0`) into a list called column_names

for column in first_launch_table.find_all("th"):
    column_name = extract_column_from_header(column)
    if column_name is not None and len(column_name) > 0:
        column_names.append(column_name)

```

✓ 0.4s Python

Check the extracted column names

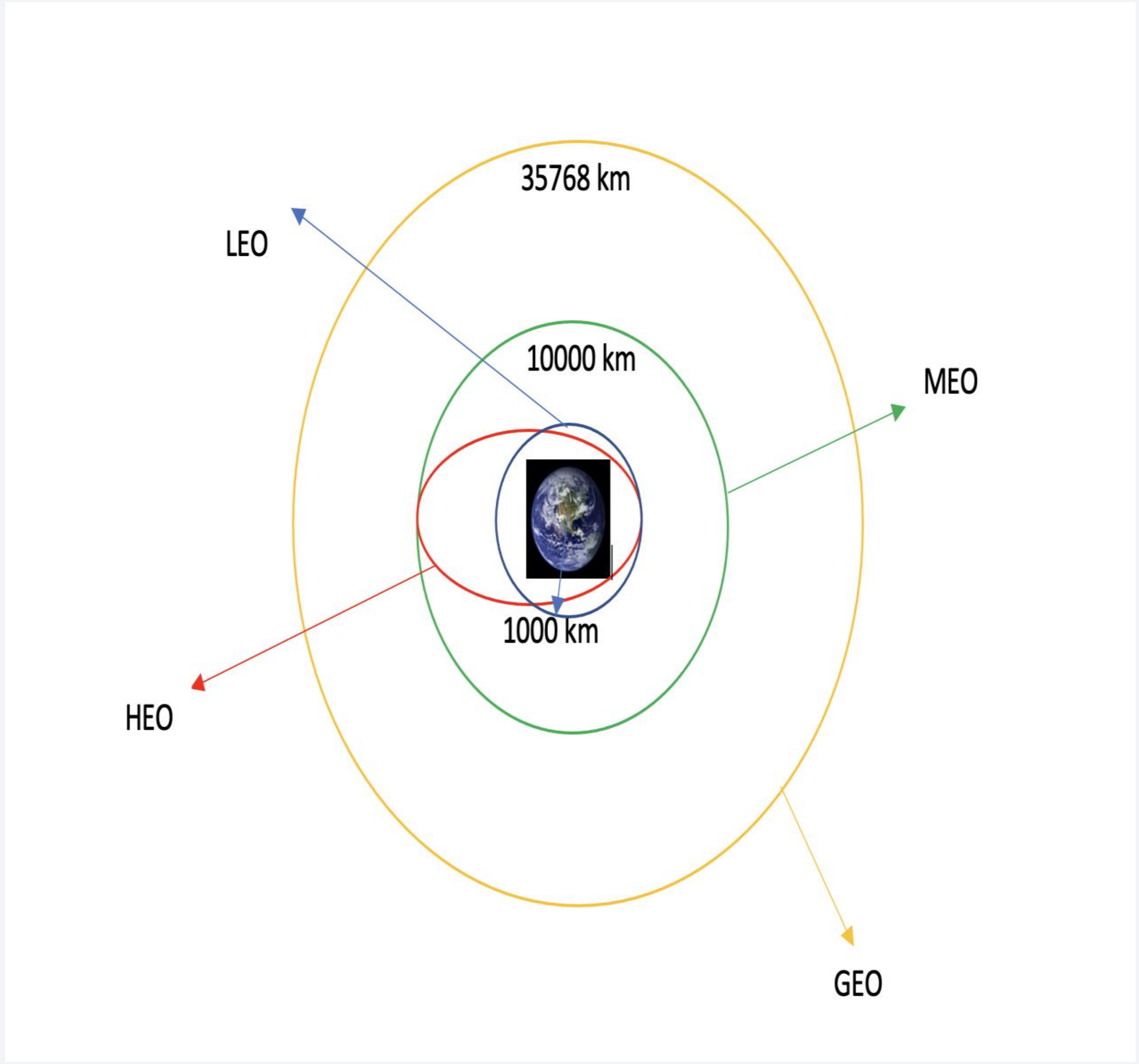
```
print(column_names)
```

✓ 0.6s Python

```
['Flight No.', 'Date and time ( )', 'Launch site', 'Payload', 'Payload mass', 'Orbit', 'Customer', 'Launch outcome']
```

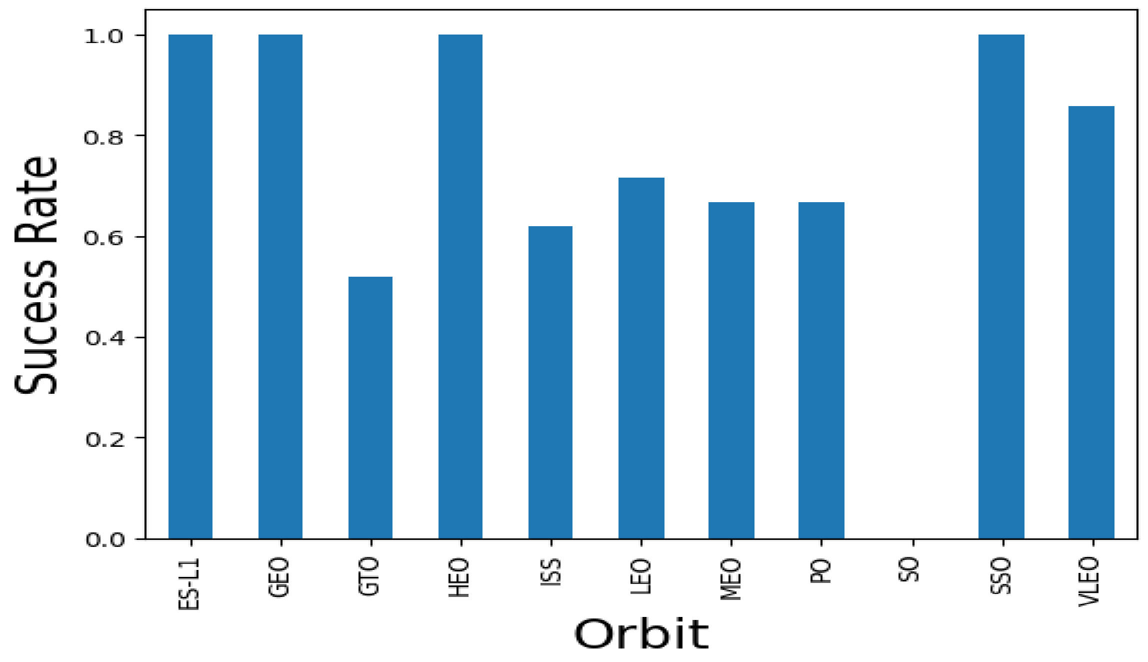
Data Wrangling

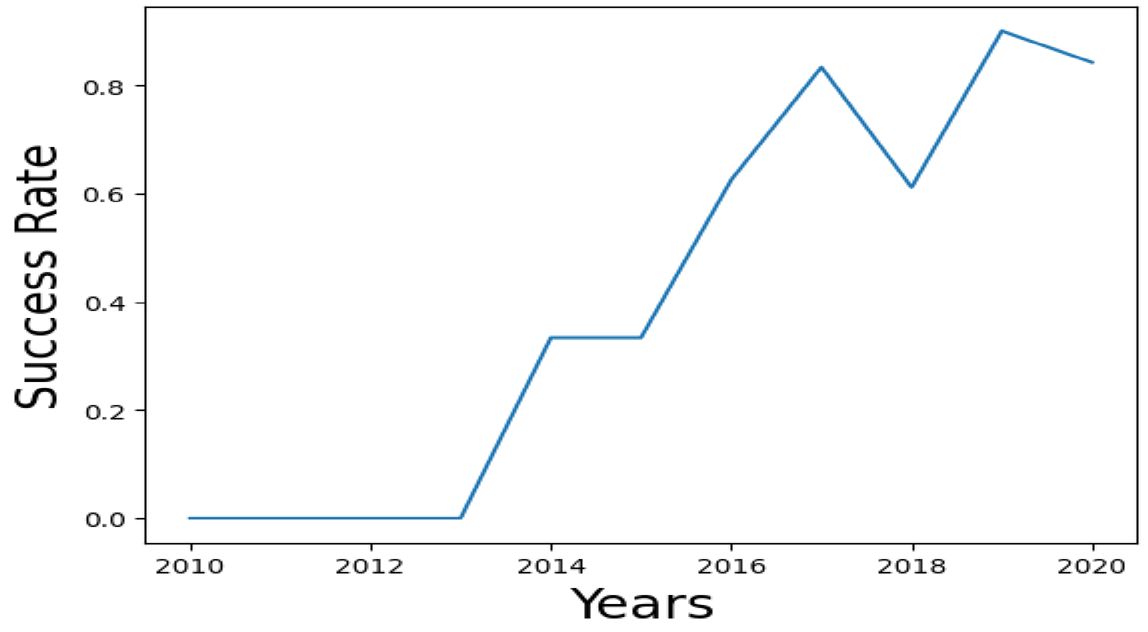
- Data was loaded into Pandas DataFrame
- We performed exploratory data analysis and determined the training labels.
- We identified the percentage of missing values for each attribute
- We identified which columns are numerical and categorical
- We calculated number of launches for each site
- We calculated the occurrences of each mission for each orbit
- We calculated distribution of mission outcome for each orbit
- We created landing outcome (“1”=success/”0”=failure) classification variable for each mission.
- Link to Jupiter notebook with data wrangling https://github.com/ArthurParuzel/SpaceXDataScience/blob/main/1.%20Data%20Collection%20API%20Web%20Scraping/labs-jupyter-spacex-Data%20wrangling.ipynb



EDA with Data Visualization

- We explored the data by visualizing the relationship between flight number and launch Site, payload and launch site, success rate of each orbit type, flight number and orbit type, the launch success yearly trend.





Link to jupyter notebook with EDA with Data Visuzalization

https://github.com/ArthurParuzel/SpaceXDataScience/blob/main/2.%20Exploratory%20Data%20Analysis%20with%20(SQL%2C%20Matplotlib)/jupyter-labs-eda-dataviz.ipynb

EDA with SQL

- SpaceX Dataset was loaded into IBM DB2 database and analyzed using jupyter notebook.
- EDA with SQL allowed to get insights from the data. SQL queries helped us answer following questions
    - The names of the unique launch sites in the space mission
    - The total payload mass carried by boosters launched by NASA (CRS)
    - Average payload mass carried by booster version F9 v1.1
    - The names of the boosters which have success in drone ship for specific payload range (4k-6k)
    - The total number of successful and failure mission outcomes
    - The names of the booster_versions which have carried the maximum payload mass
    - the failed landing_outcomes in drone ship, their booster versions, and launch site names for specific year
    - Rank and the count of all possible landing outcomes (such as Failure (drone ship) or Success (ground pad))

Link to Jupiter notebook with full SQL EDA
https://github.com/ArthurParuzel/SpaceXDataScience/blob/main/2.%20Exploratory%20Data%20Analysis%20with%20(SQL%2C%20Matplotlib)/jupyter-labs-eda-sql-coursera.ipynb

Build an Interactive Map with Folium

- All launch sites have been marked, and map objects such as markers, circles, lines to mark the success or failure of launches for each site have been added onto the folium map.
- We assigned the feature launch outcomes (failure or success) to class 0 and 1.i.e., 0 for failure, and 1 for success.
- Using the color-labeled marker clusters, we identified which launch sites have relatively high success rate.
- We calculated the distances between a launch site to its proximities. We answered some question for instance:
- 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?

Link to Jupiter notebook

https://github.com/ArthurParuzel/SpaceXDataScience/blob/main/3.%20Dashboards%20with%20Plotly/lab_jupyter_launch_site_location.ipynb

Build a Dashboard with Plotly Dash

- We built an interactive dashboard with Plotly dash. It allowed us to observe correlation between certain dataset elements dynamically.
- We plotted pie charts showing the total launches by a certain sites. It can allow simple analysis and comparison between different sites.
- We plotted scatter graph showing the relationship with Outcome and Payload Mass (Kg) for the different booster version. This allowed us to compare flight results broken down to light, medium and heavy payloads against different booster versions.

- Link to Python Plotly server code running the dashboard can be found here

https://github.com/ArthurParuzel/SpaceXDataScience/blob/main/3.%20Dashboards%20with%20Plotly/spacex_dash_app.py

Predictive Analysis (Classification)

- We loaded the data using numpy and pandas, transformed the data, split our data into training and testing.
- We built different machine learning models and tune different hyperparameters using GridSearchCV.
- We used accuracy as the metric for our model, improved the model using feature engineering and algorithm tuning.
- We found the best performing classification model.

Link to Jupiter notebook can be found here

https://github.com/ArthurParuzel/SpaceXDataScience/blob/main/5.%20Final%20Report%20and%20findings/SpaceX_Machine%20Learning%20Prediction_Part_5%20CLOUD.ipynb

Results

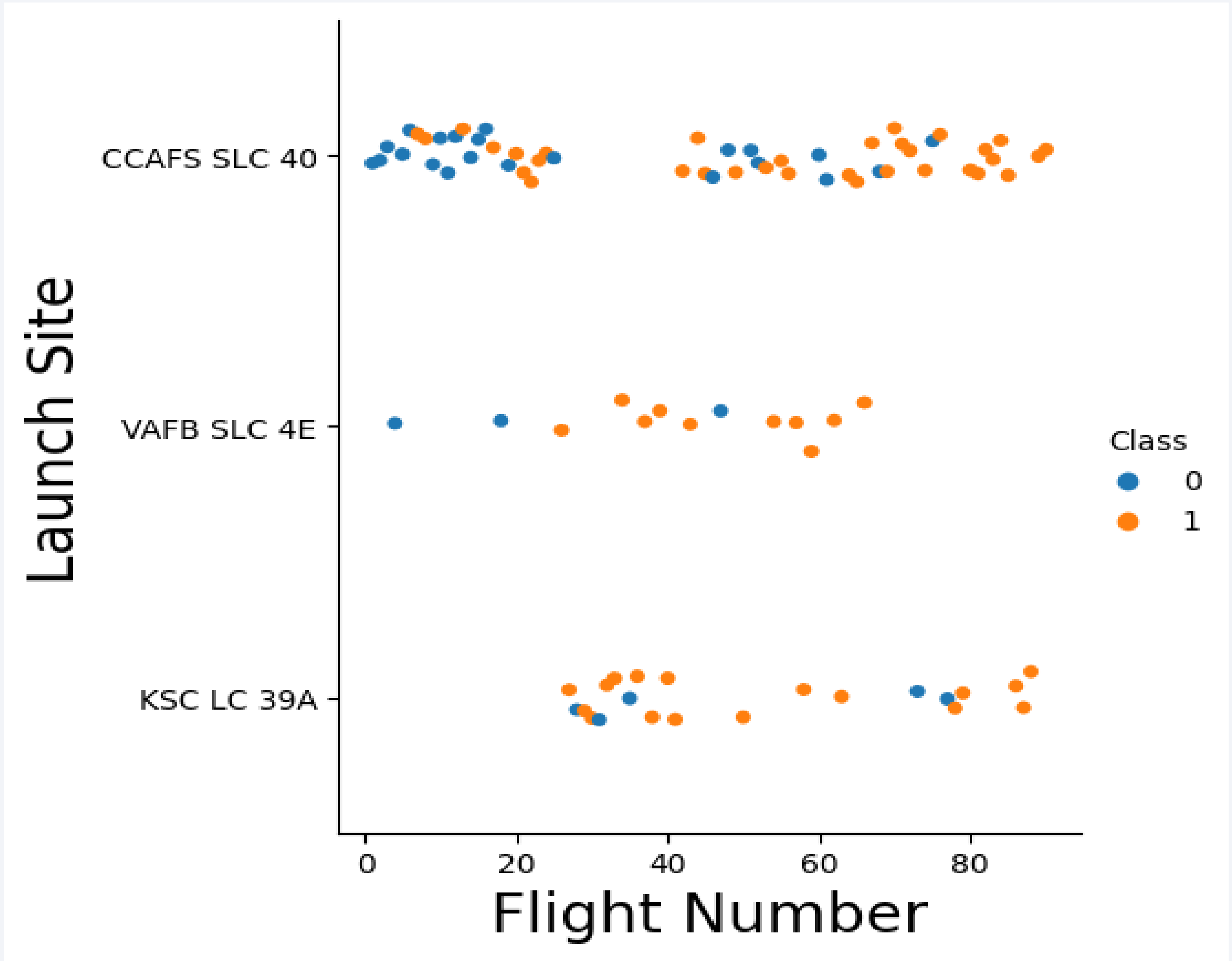
- Exploratory data analysis results
- Interactive analytics demo in screenshots
- Predictive analysis results

Section 2

Insights drawn from EDA

Flight Number vs. Launch Site

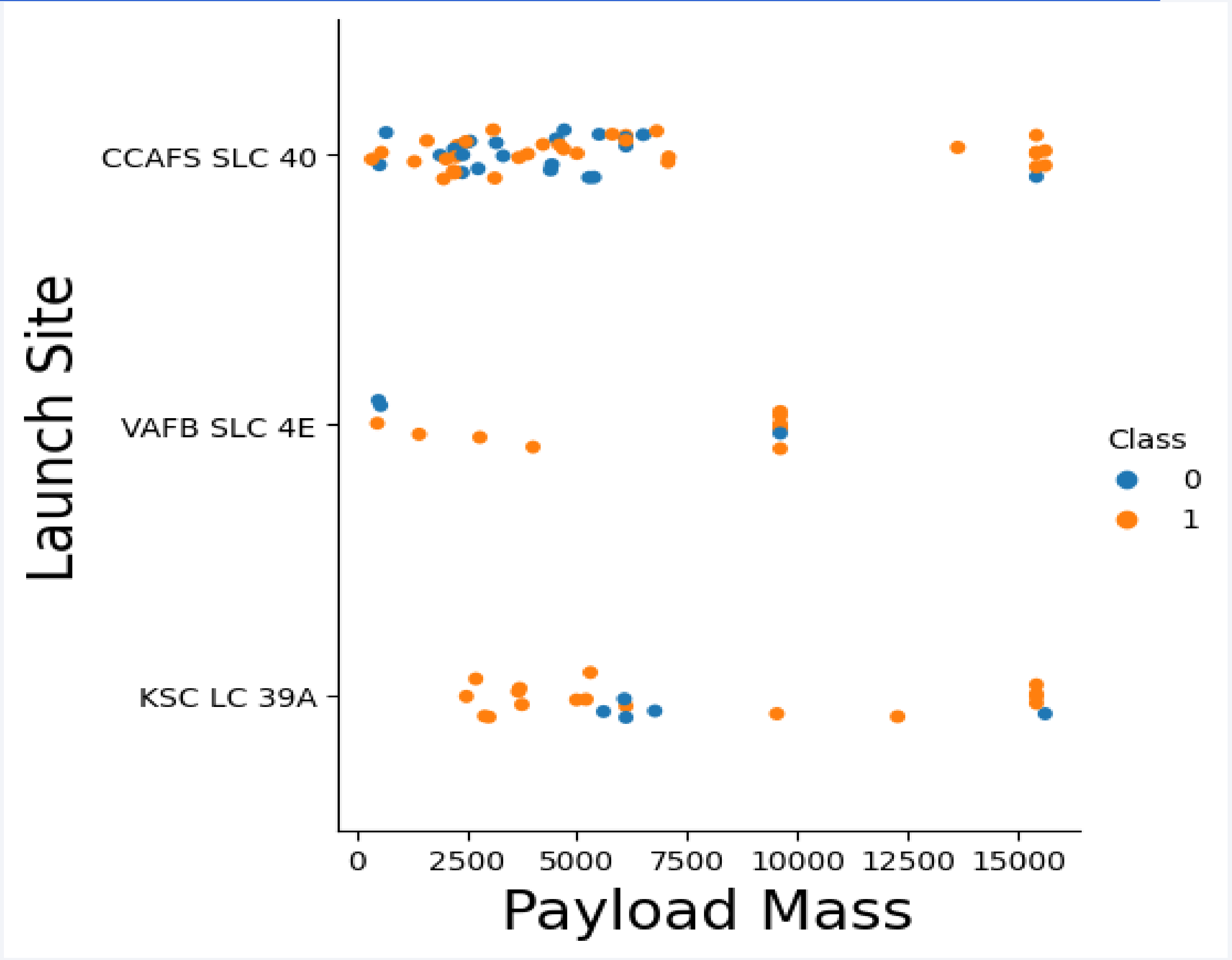
From the plot, we found that the larger the flight amount at a launch site, the greater the success rate at a launch site.



Payload vs. Launch Site

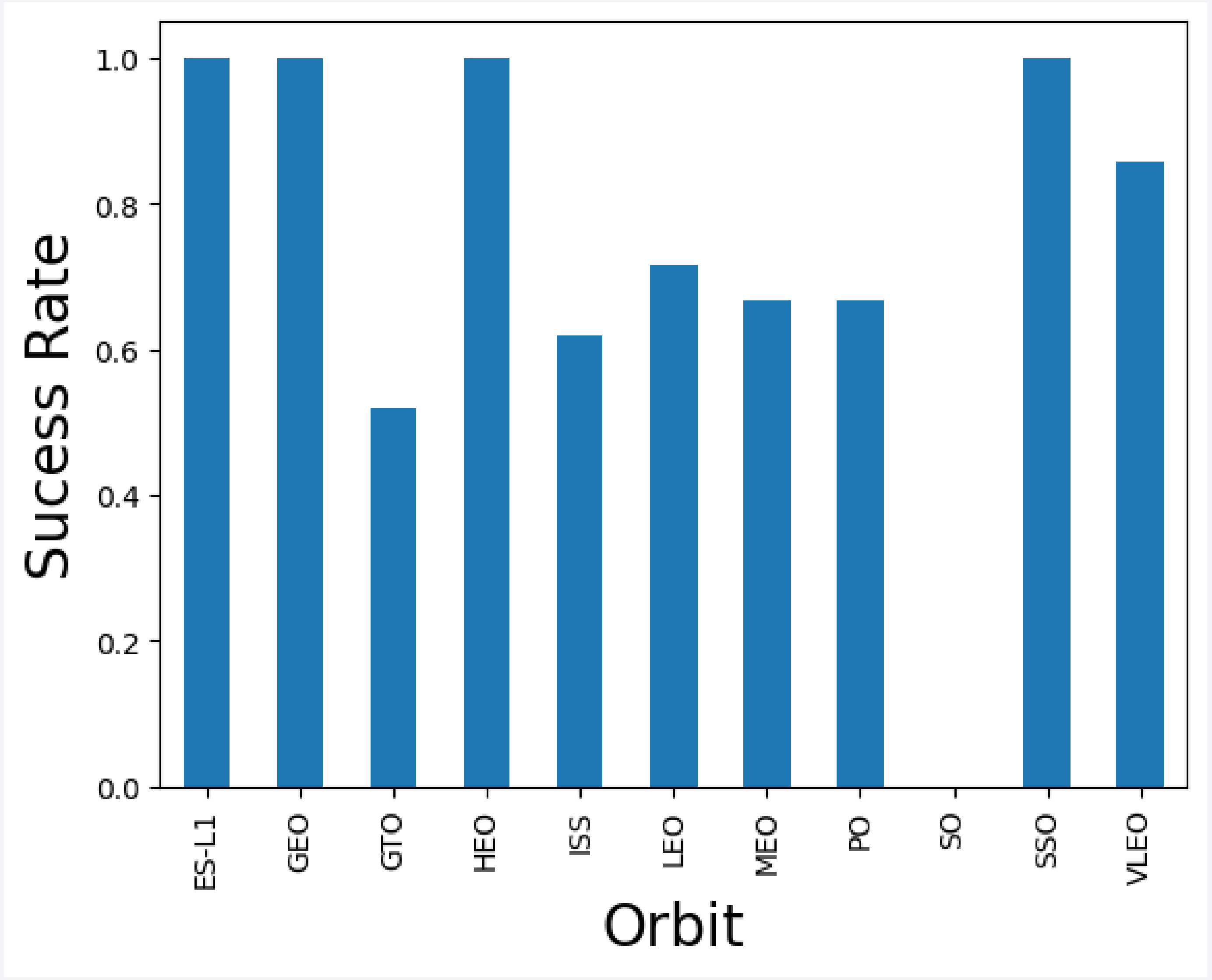
From the plot, we found that the greater the payload mass for the launch sites KSC LC 39A and CCAFS SLC 40 the higher the success rate for the rocket.

KSC LC 39A site has majority of the failures for the mid range of payload, between 5000 and 7500



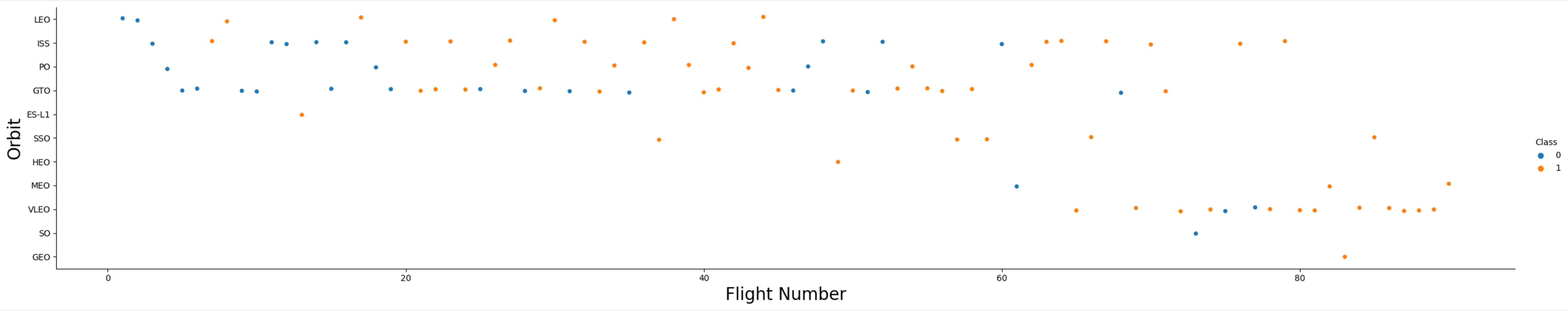
Success Rate vs. Orbit Type

From the plot, we can see that ES-L1, EO, HEO, SSO,VLEO had the most success rate.



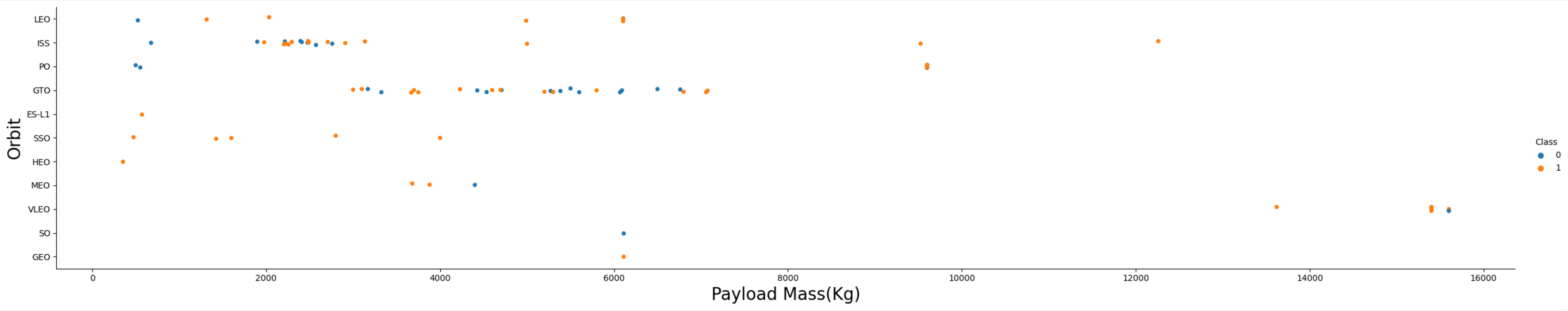
Flight Number vs. Orbit Type

- Scatter Plot of Flight number vs. Orbit type



Payload vs. Orbit Type

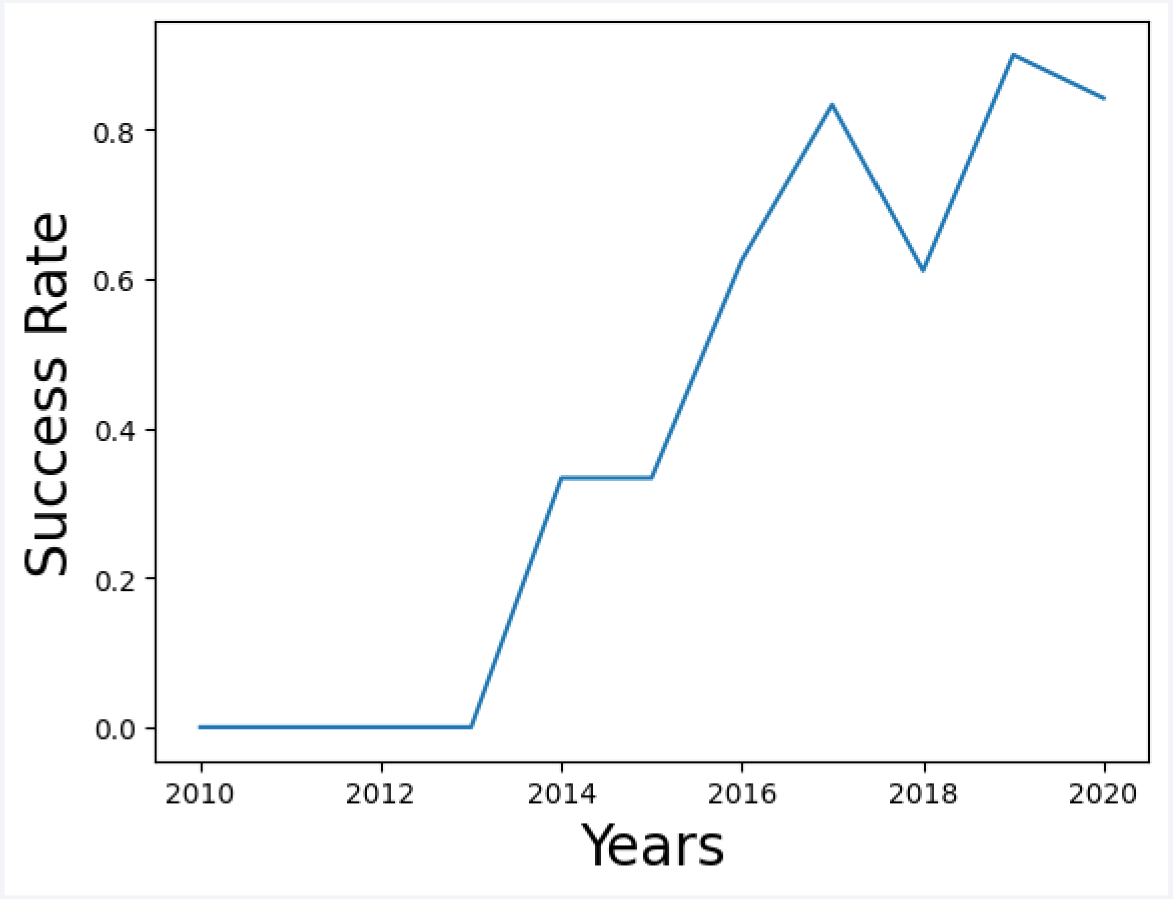
- Scatter plot of payload vs. orbit type



Launch Success Yearly Trend

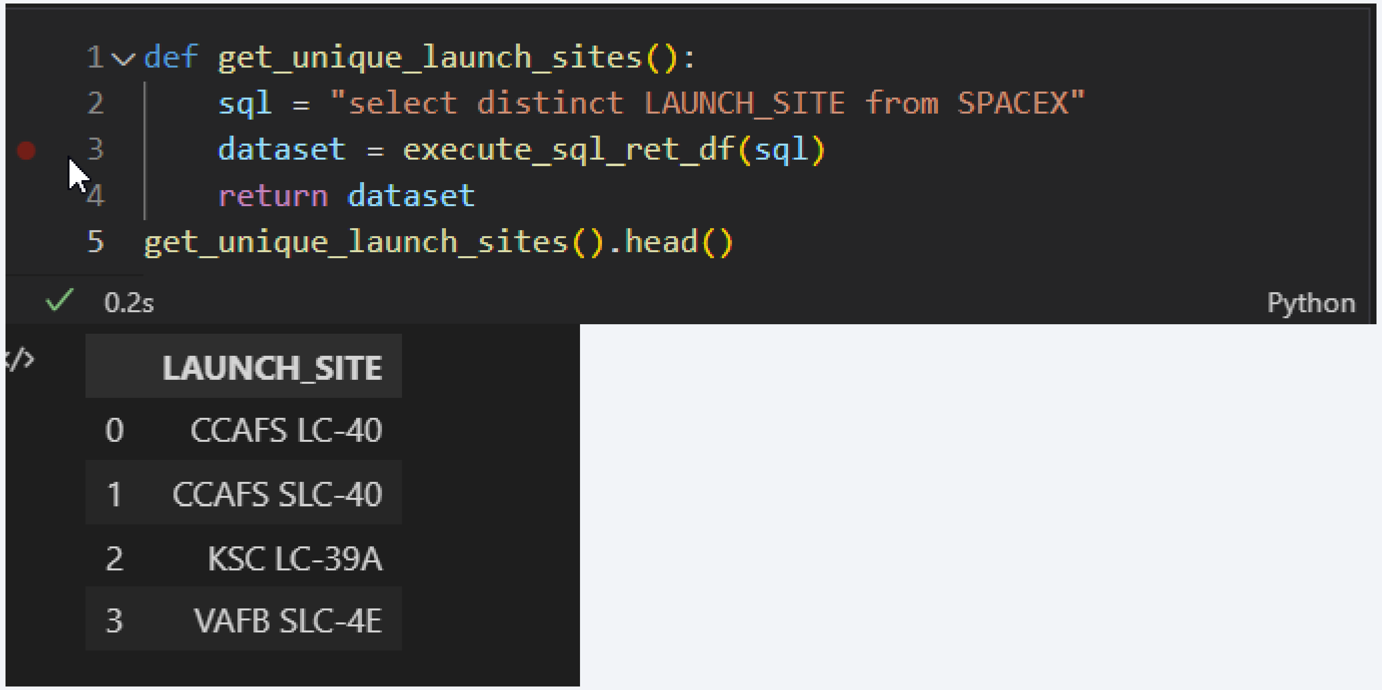
- Show a line chart of yearly average success rate

- Show the screenshot of the scatter plot with explanations



All Launch Site Names

- Unique launch sites



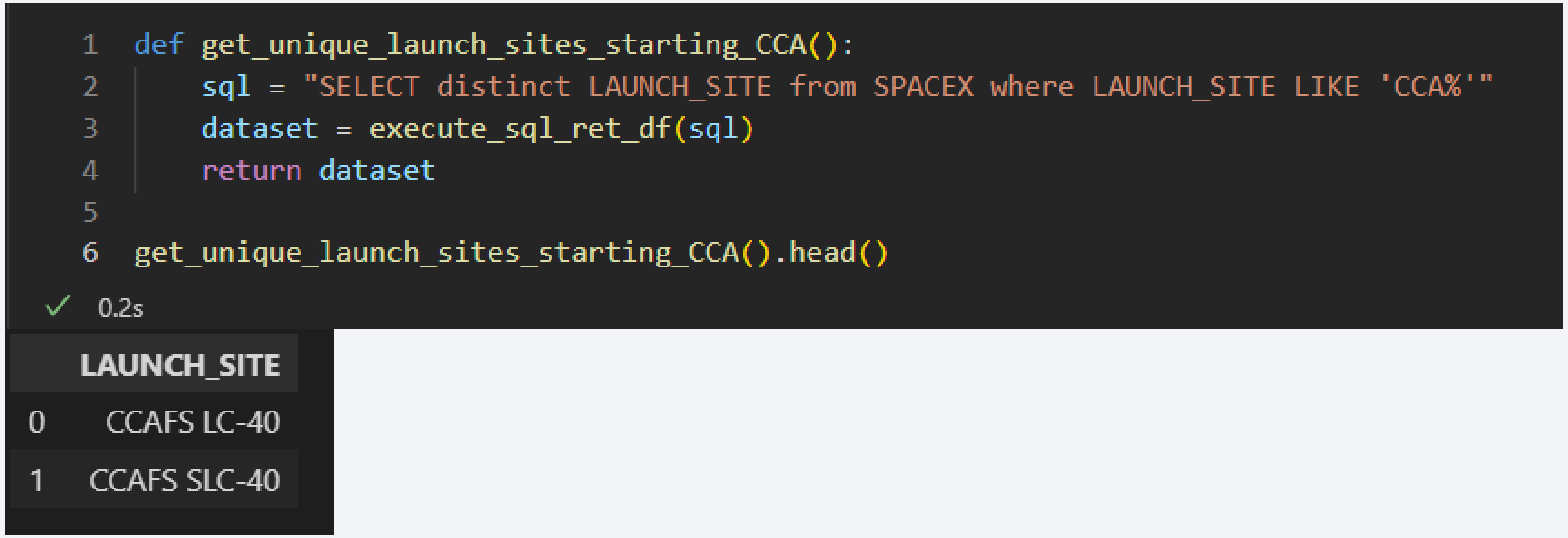
```python
def get_unique_launch_sites():
    sql = "select distinct LAUNCH_SITE from SPACEX"
    dataset = execute_sql_ret_df(sql)
    return dataset
get_unique_launch_sites().head()
```

✓ 0.2s Python

```
   LAUNCH_SITE
0  CCAFS LC-40
1  CCAFS SLC-40
2  KSC LC-39A
3  VAFB SLC-4E
```

Launch Site Names Begin with 'CCA'

- Find 5 records where launch sites begin with `CCA`



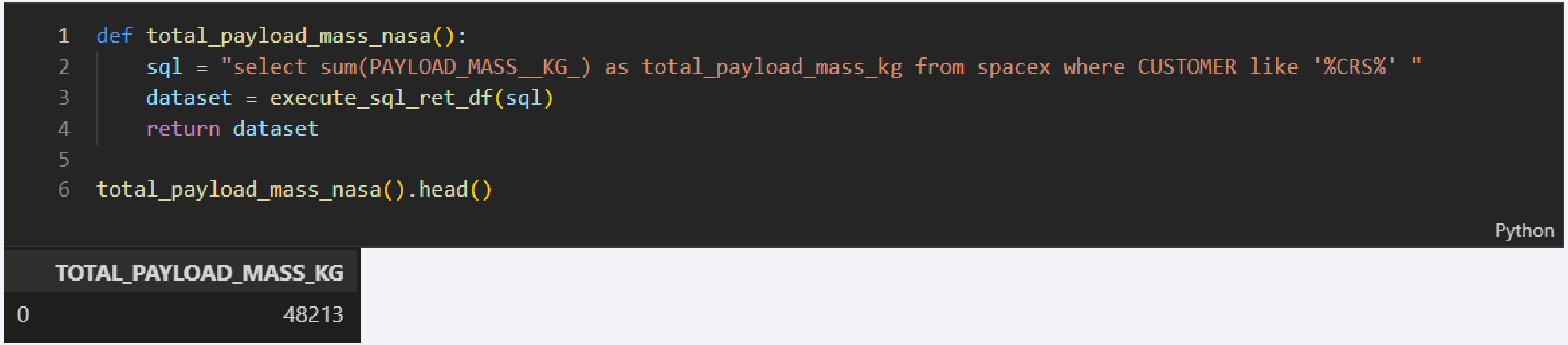
```
def get_unique_launch_sites_starting_CCA():
    sql = "SELECT distinct LAUNCH_SITE from SPACEX where LAUNCH_SITE LIKE 'CCA%'"
    dataset = execute_sql_ret_df(sql)
    return dataset

get_unique_launch_sites_starting_CCA().head()
```

✓ 0.2s

	LAUNCH_SITE
0	CCAFS LC-40
1	CCAFS SLC-40

Total Payload Mass

- Calculate the total payload carried by boosters from NASA

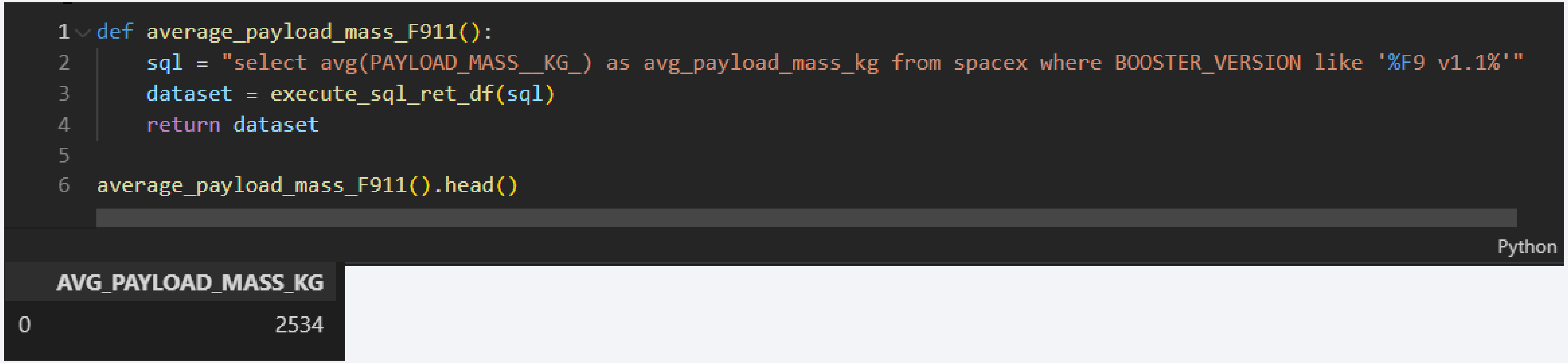


```python
def total_payload_mass_nasa():
    sql = "select sum(PAYLOAD_MASS__KG_) as total_payload_mass_kg from spacex where CUSTOMER like '%CRS%' "
    dataset = execute_sql_ret_df(sql)
    return dataset

total_payload_mass_nasa().head()
```

```
   TOTAL_PAYLOAD_MASS_KG
0                  48213
```

Average Payload Mass by F9 v1.1

- Calculate the average payload mass carried by booster version F9 v1.1



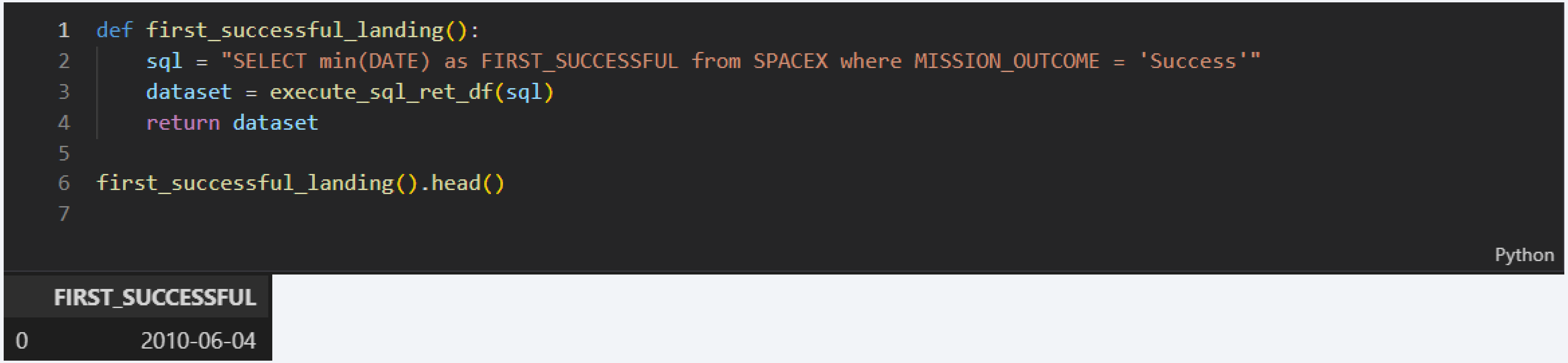
```
def average_payload_mass_F911():
    sql = "select avg(PAYLOAD_MASS__KG_) as avg_payload_mass_kg from spacex where BOOSTER_VERSION like '%F9 v1.1%'"
    dataset = execute_sql_ret_df(sql)
    return dataset

average_payload_mass_F911().head()
```

Python

	AVG_PAYLOAD_MASS_KG
0	2534

First Successful Ground Landing Date

- Find the dates of the first successful landing outcome on ground pad

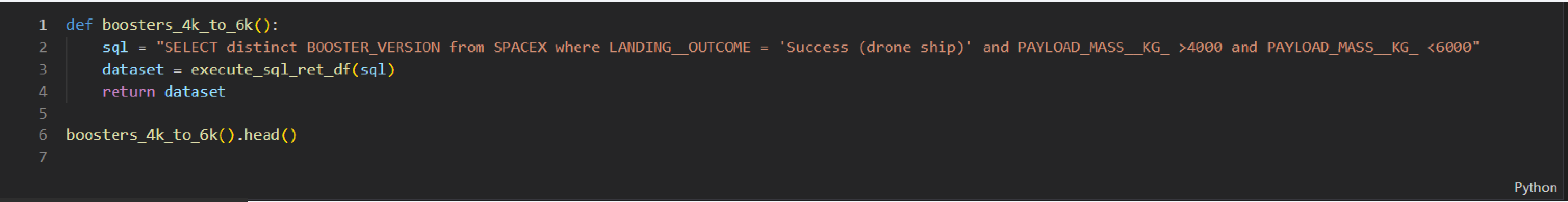


```python
def first_successful_landing():
    sql = "SELECT min(DATE) as FIRST_SUCCESSFUL from SPACEX where MISSION_OUTCOME = 'Success'"
    dataset = execute_sql_ret_df(sql)
    return dataset

first_successful_landing().head()

```

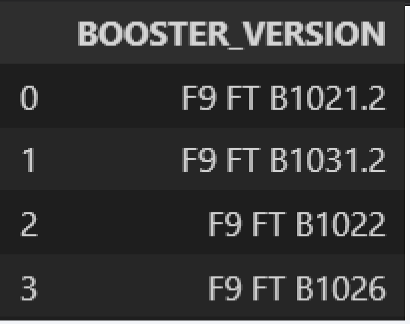
```
  FIRST_SUCCESSFUL
0       2010-06-04
```

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



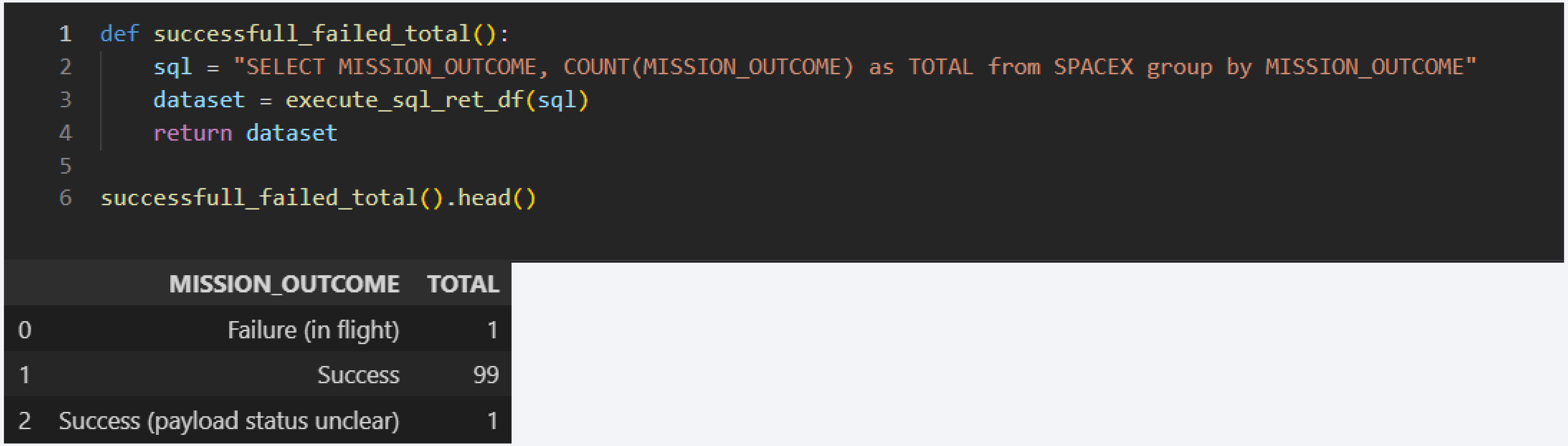
```python
def boosters_4k_to_6k():
    sql = "SELECT distinct BOOSTER_VERSION from SPACEX where LANDING__OUTCOME = 'Success (drone ship)' and PAYLOAD_MASS__KG_ >4000 and PAYLOAD_MASS__KG_ <6000"
    dataset = execute_sql_ret_df(sql)
    return dataset

boosters_4k_to_6k().head()

```



	BOOSTER_VERSION
0	F9 FT B1021.2
1	F9 FT B1031.2
2	F9 FT B1022
3	F9 FT B1026

Total Number of Successful and Failure Mission Outcomes

- Calculate the total number of successful and failure mission outcomes

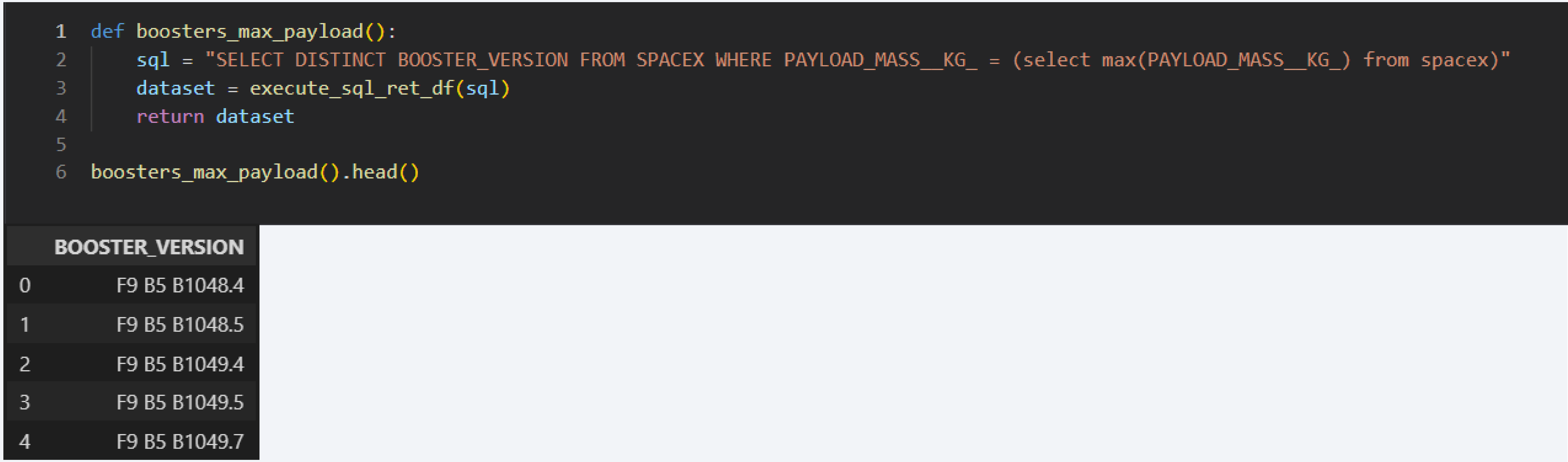


```python
def successfull_failed_total():
    sql = "SELECT MISSION_OUTCOME, COUNT(MISSION_OUTCOME) as TOTAL from SPACEX group by MISSION_OUTCOME"
    dataset = execute_sql_ret_df(sql)
    return dataset

successfull_failed_total().head()
```

	MISSION_OUTCOME	TOTAL
0	Failure (in flight)	1
1	Success	99
2	Success (payload status unclear)	1

Boosters Carried Maximum Payload

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

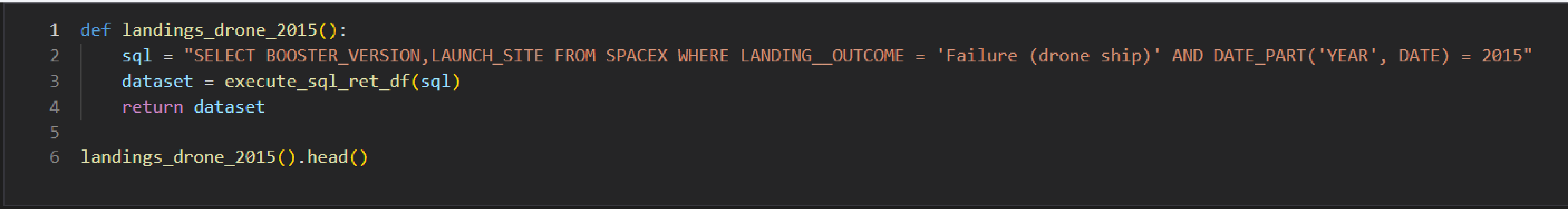


```python
def boosters_max_payload():
    sql = "SELECT DISTINCT BOOSTER_VERSION FROM SPACEX WHERE PAYLOAD_MASS__KG_ = (select max(PAYLOAD_MASS__KG_) from spacex)"
    dataset = execute_sql_ret_df(sql)
    return dataset

boosters_max_payload().head()
```

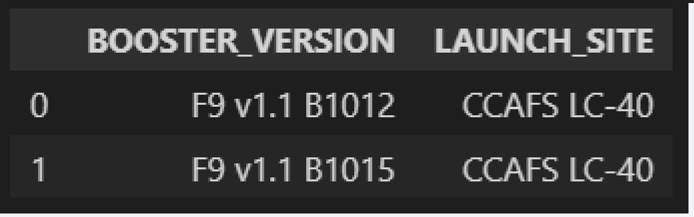
	BOOSTER_VERSION
0	F9 B5 B1048.4
1	F9 B5 B1048.5
2	F9 B5 B1049.4
3	F9 B5 B1049.5
4	F9 B5 B1049.7

2015 Launch Records

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



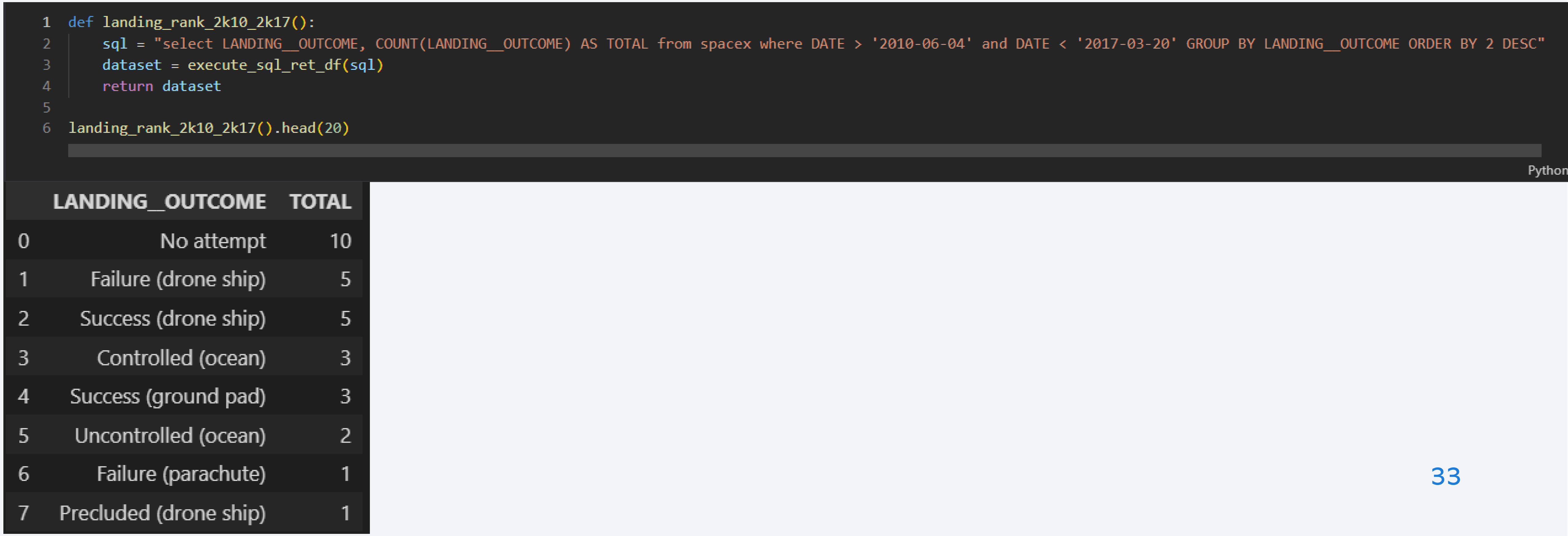
```
def landings_drone_2015():
    sql = "SELECT BOOSTER_VERSION,LAUNCH_SITE FROM SPACEX WHERE LANDING__OUTCOME = 'Failure (drone ship)' AND DATE_PART('YEAR', DATE) = 2015"
    dataset = execute_sql_ret_df(sql)
    return dataset

landings_drone_2015().head()
```



	BOOSTER_VERSION	LAUNCH_SITE
0	F9 v1.1 B1012	CCAFS LC-40
1	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



```python
def landing_rank_2k10_2k17():
    sql = "select LANDING__OUTCOME, COUNT(LANDING__OUTCOME) AS TOTAL from spacex where DATE > '2010-06-04' and DATE < '2017-03-20' GROUP BY LANDING__OUTCOME ORDER BY 2 DESC"
    dataset = execute_sql_ret_df(sql)
    return dataset

landing_rank_2k10_2k17().head(20)
```

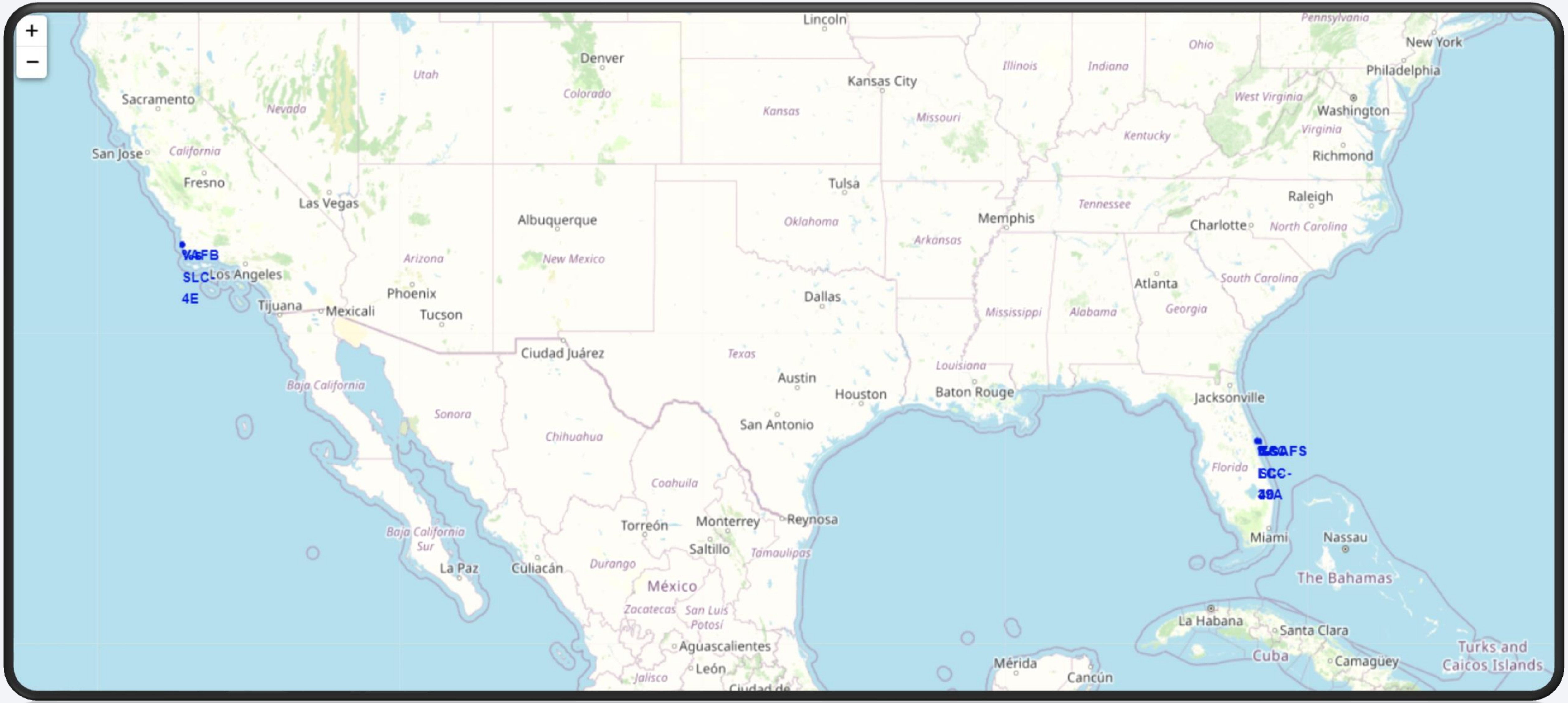
Python

	LANDING__OUTCOME	TOTAL
0	No attempt	10
1	Failure (drone ship)	5
2	Success (drone ship)	5
3	Controlled (ocean)	3
4	Success (ground pad)	3
5	Uncontrolled (ocean)	2
6	Failure (parachute)	1
7	Precluded (drone ship)	1

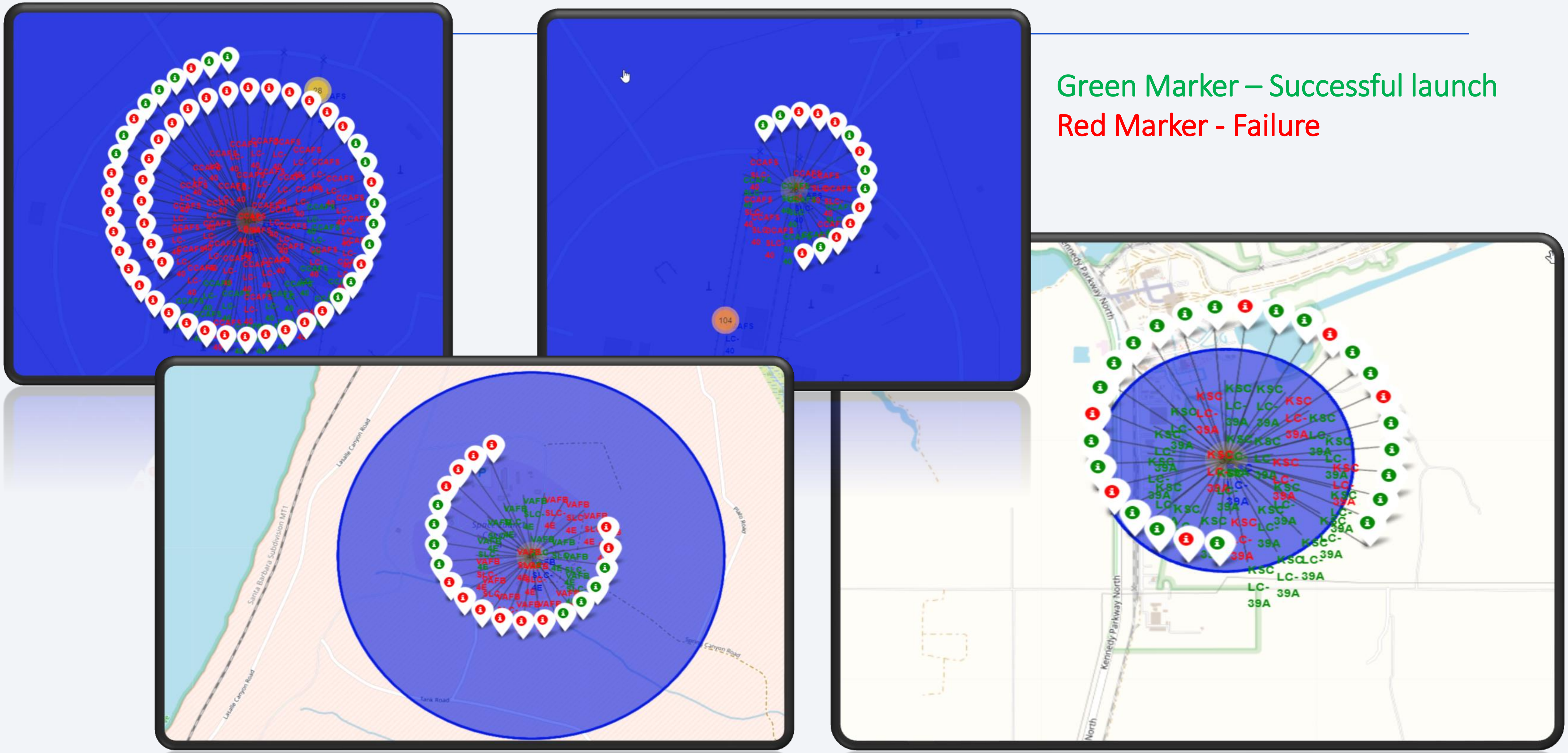
Section 3

Launch Sites Proximities Analysis

<Folium Map Screenshot 1>


+
−
VAFB
SLC-4E
CCAFS
SLC-40
KSC
39A

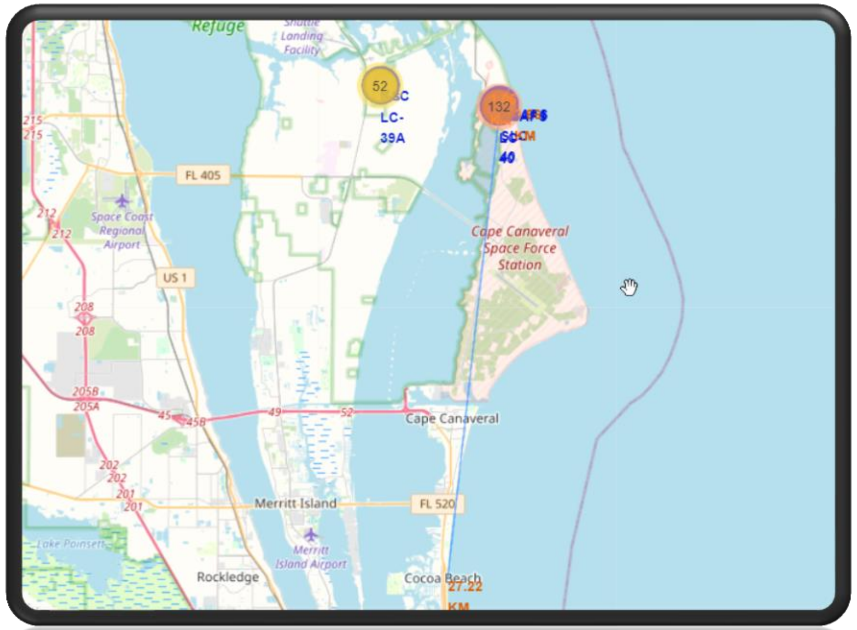
Markers showing launch sites with color labels

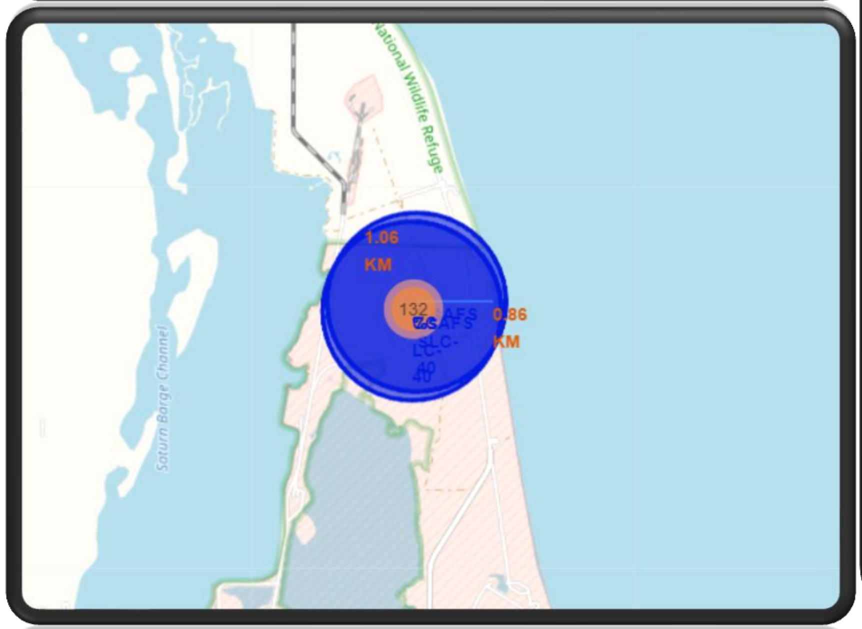


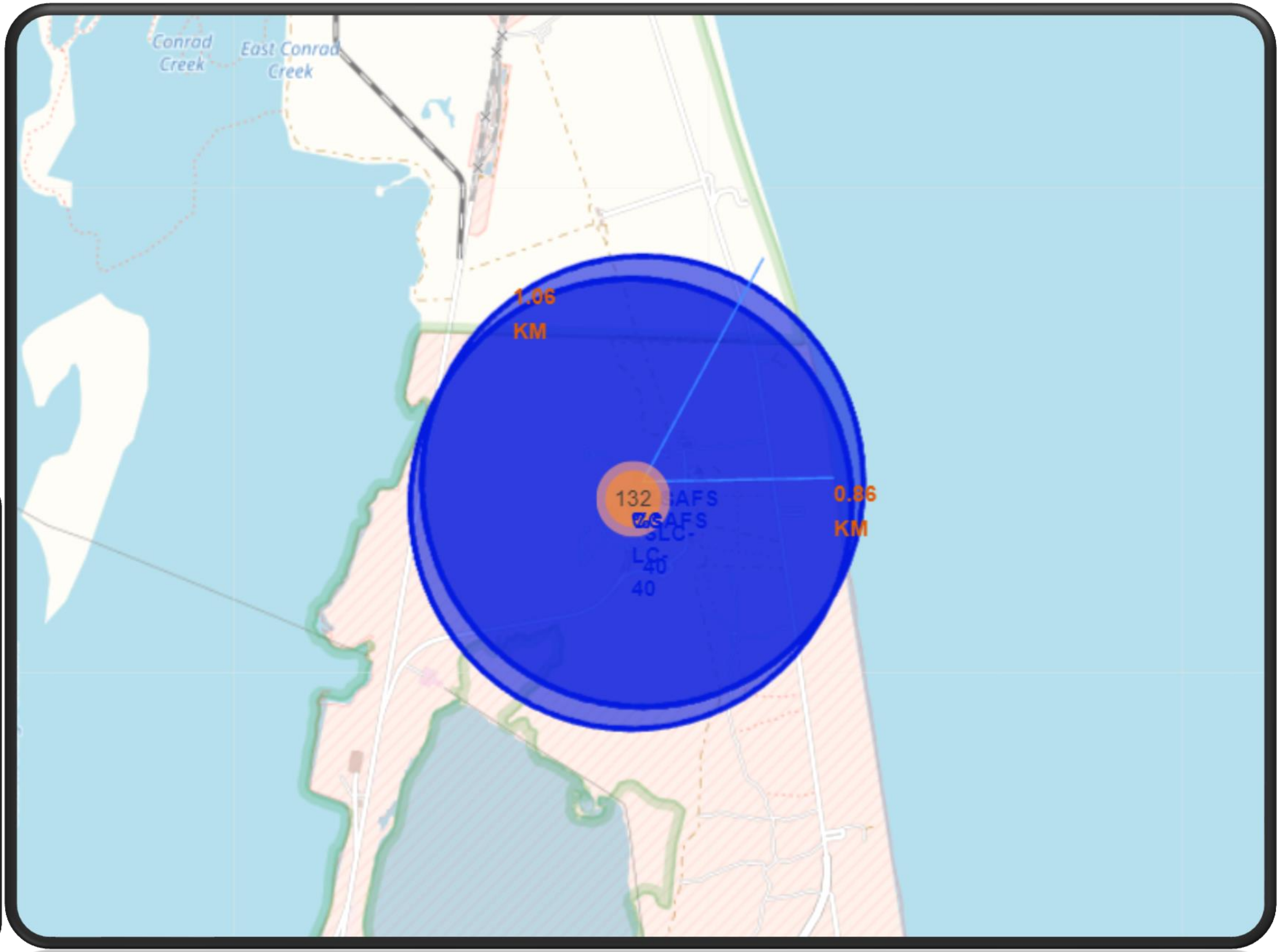
Green Marker – Successful launch

Red Marker - Failure

Launch site distance to critical infrastructure



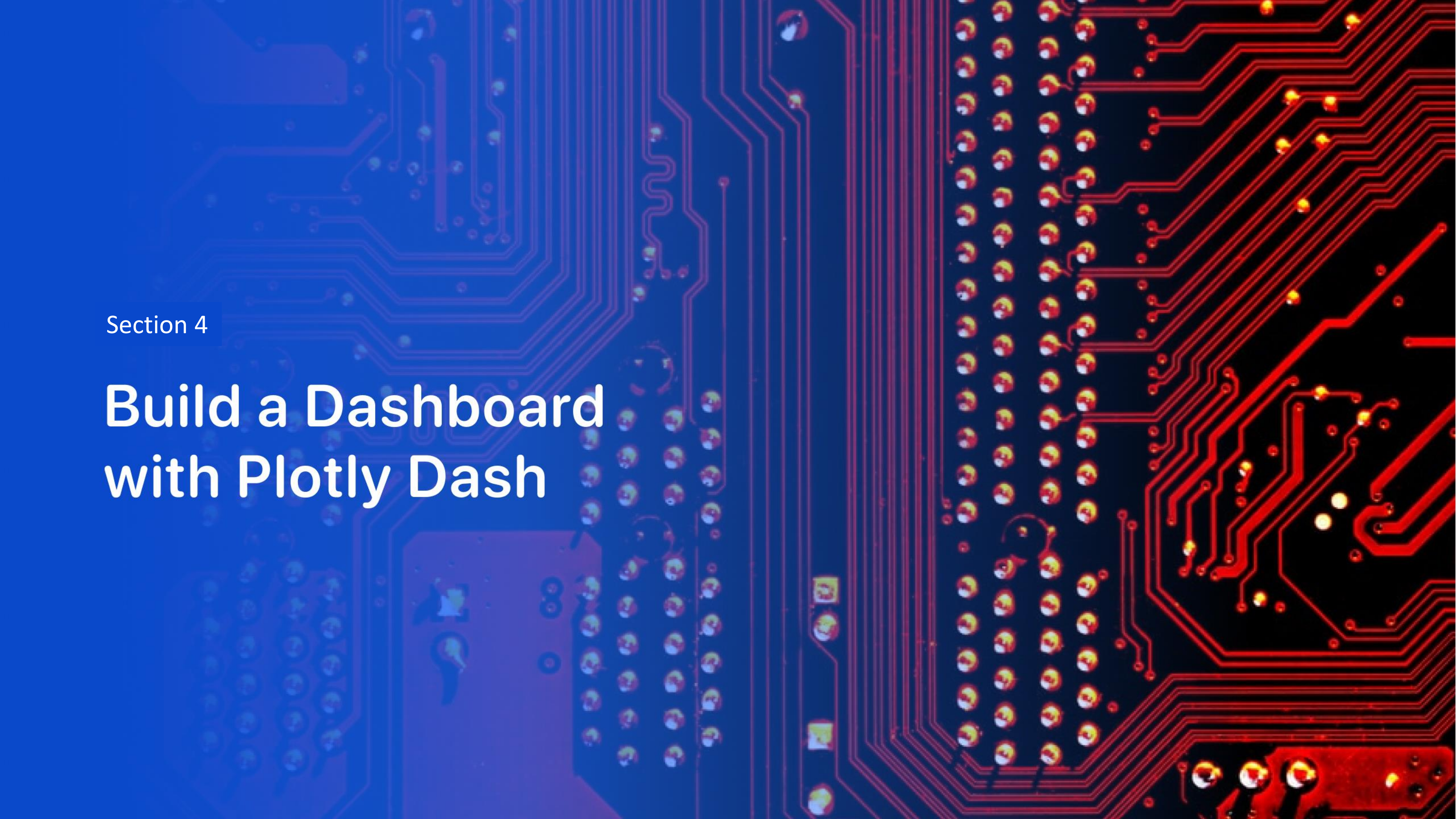




Map exploration analysis

During the visual exploratory analysis, we asked ourselves few questions about the launching infrastructure in relation to the key geographical and industrial features.

- Q1: Are launch sites in close proximity to railways?
- **A1: railways play key component, next to highways, to accommodate cargo needs of a launch site. Launch sites are fairly close to railways allowing shipments go directly to the site.**
- Q2: Are launch sites in close proximity to highways?
- **A2: As crucial as railways, yet available to much higher traffic, highways are somehow close to the launch site, but are never too close for**
- Q3: Are launch sites in close proximity to coastline?
- **A3: Landing on floating platform and having proximity to closed marine zone allows for safety buffer and to mitigate all unsuccessful starts and landings. All launch sites were built very close to the coastlines.**
- Q4: Do launch sites keep certain distance away from cities?
- **A4: As a security risk, all launch sites were properly spaced between closest cities and agglomerations.**

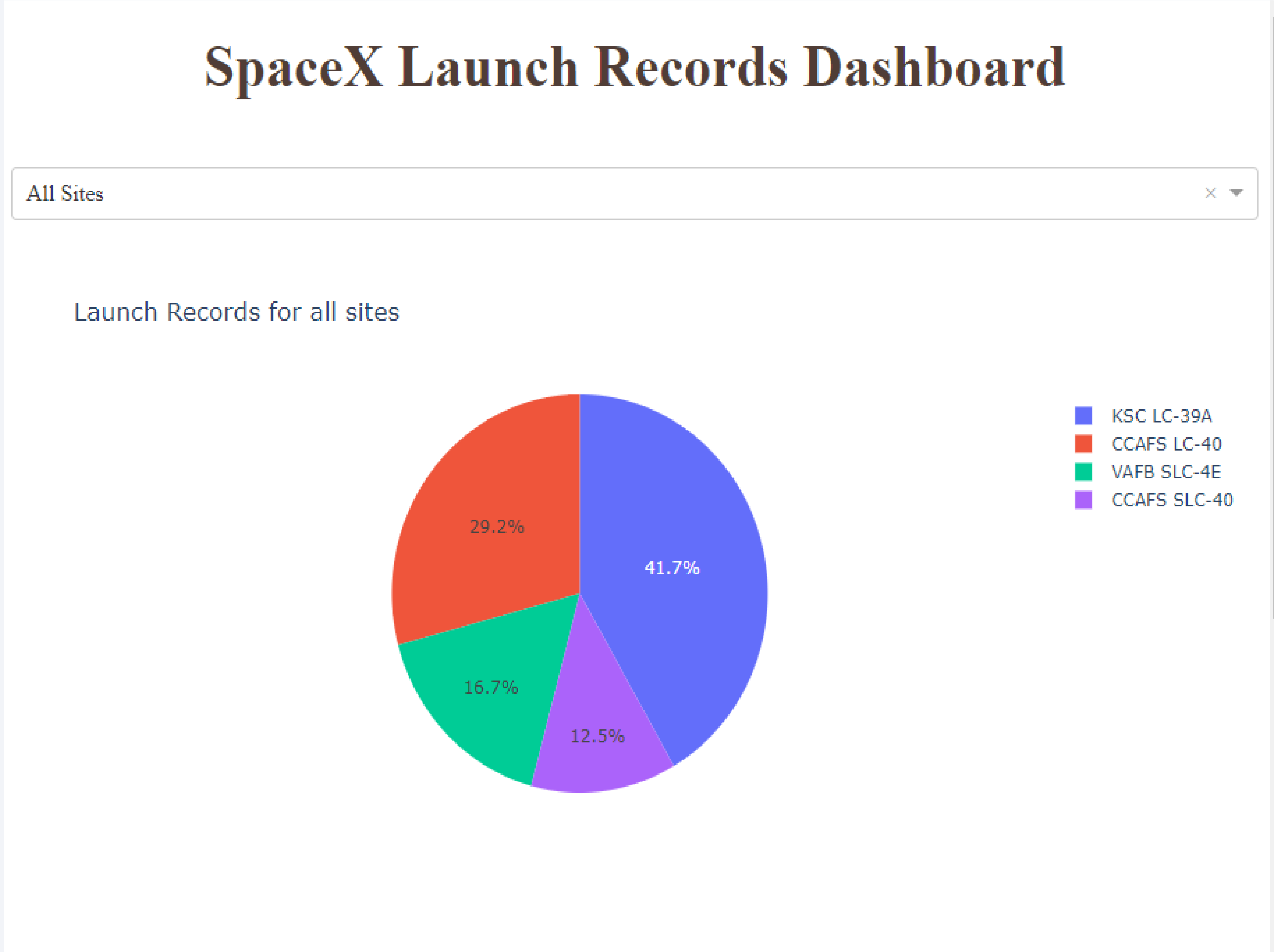


Section 4

Build a Dashboard with Plotly Dash

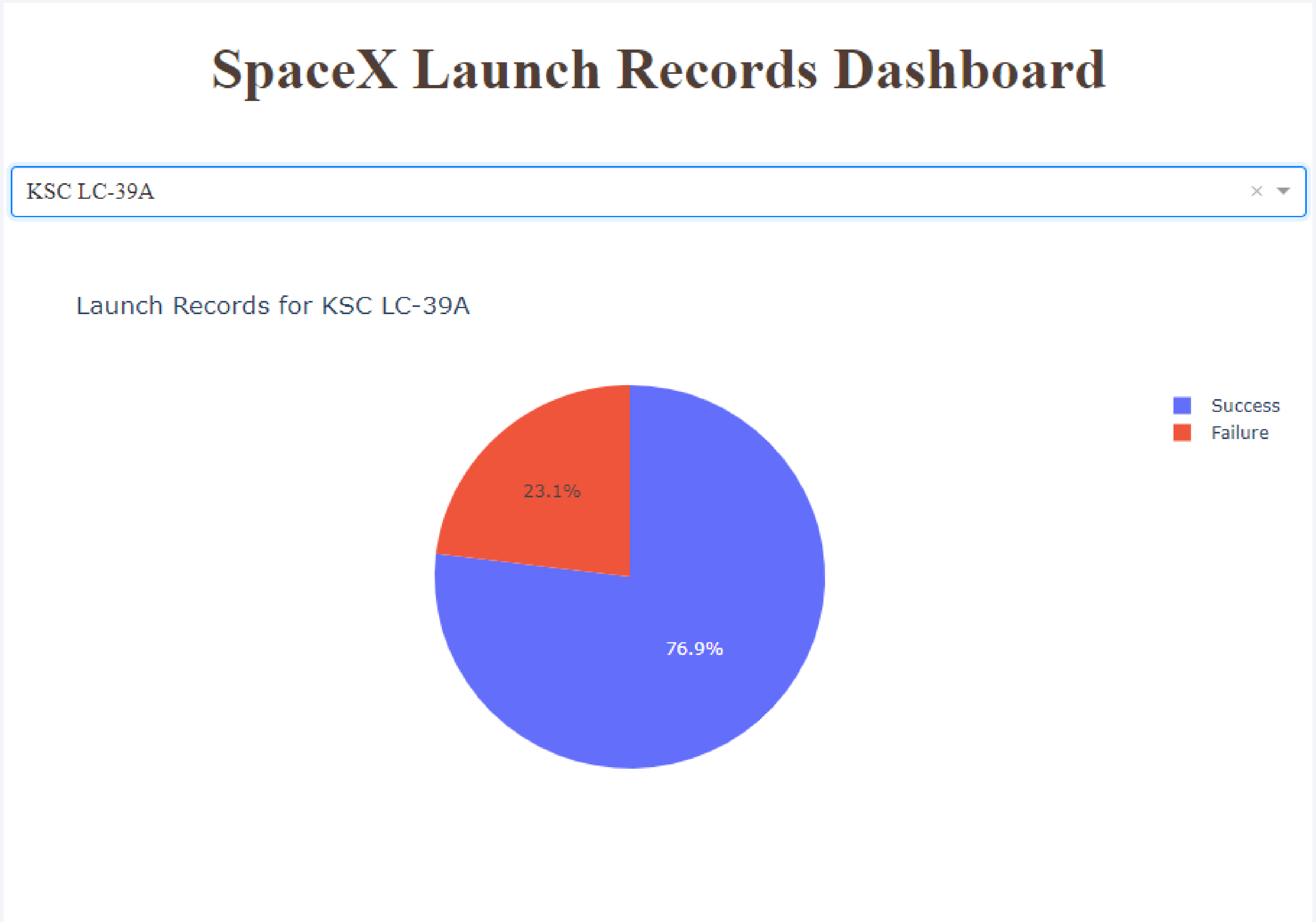
SpaceX Launch Records Dashboard

- KSC LC-39A has the greatest amount of launch/landing attempts
- CCAFS SLC-40 site has the lowest amount of launch/landing attempts



SpaceX Launch Records Dashboard

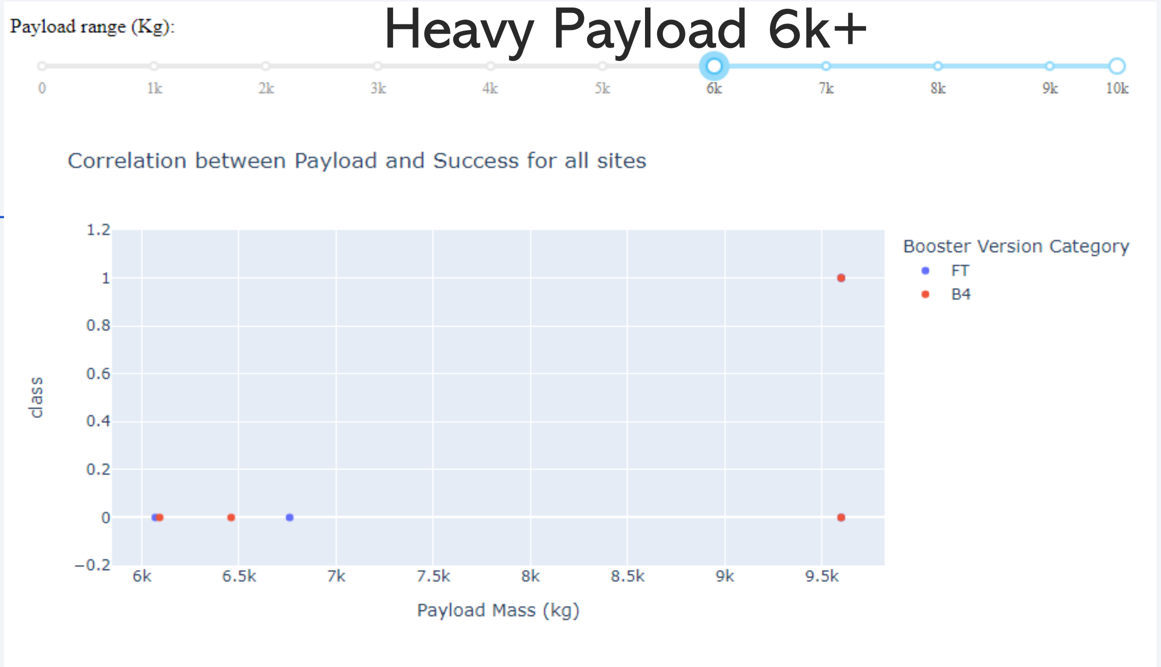
- KSC LC-39A location has the best Success to Failure ratio from all the sites.


SpaceX Launch Records Dashboard
KSC LC-39A
Launch Records for KSC LC-39A
23.1%
76.9%
Success
Failure

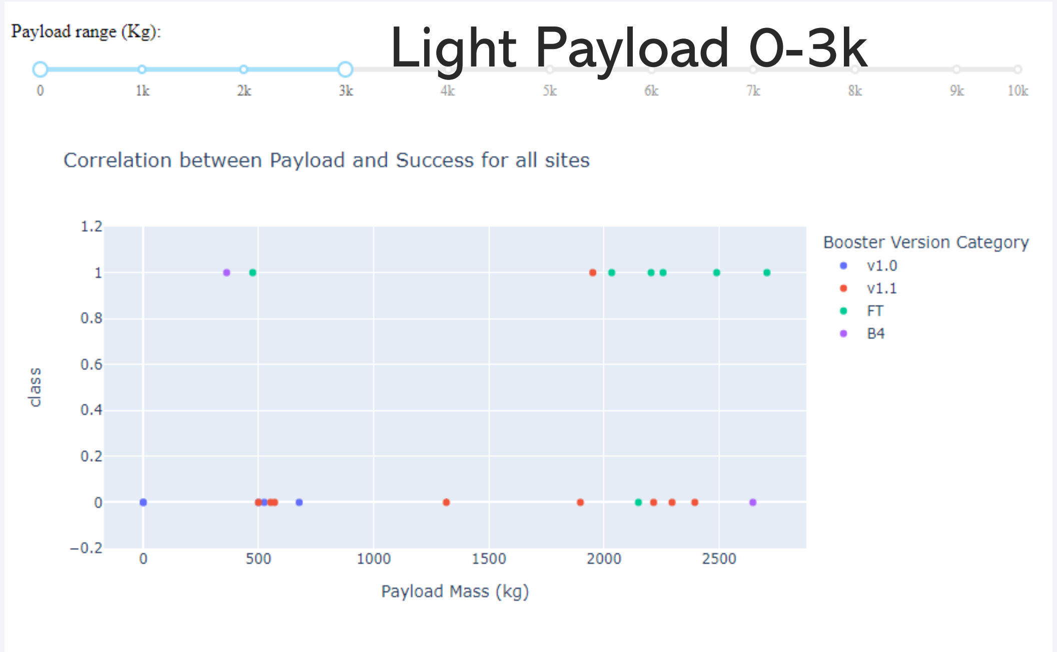
Dashboard Payload vs Success

- Heavier Payloads using B4 and FT boosters are the most successful
- FT booster has the greatest fail rate for light payloads
- Medium payloads are best served by B4 booster, v1.1 booster results in the greatest fail rate.

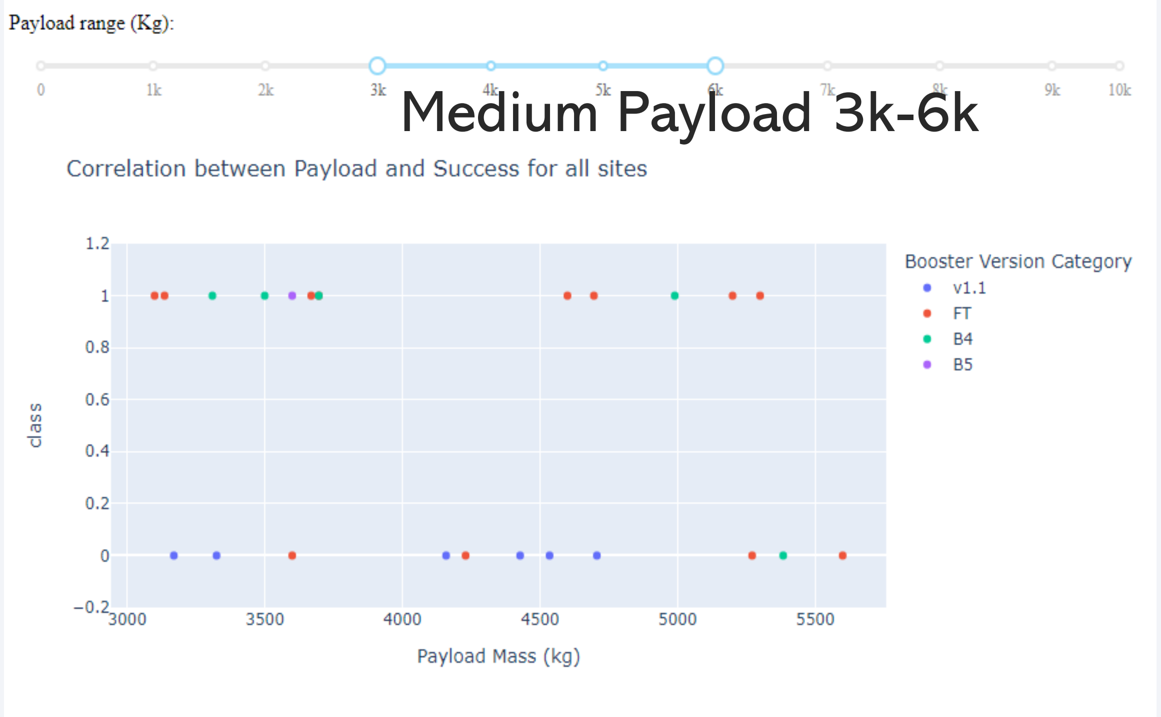
Heavy Payload 6k+



Light Payload 0-3k



Medium Payload 3k-6k



Section 5

Predictive Analysis (Classification)

Classification Accuracy

We used GridSearchCV sklearn library to provide exhaustive search over specified parameter values for all compared estimators. GridSearchCV implements a “fit” and a “score” method. We used fit with train data, and score on test data. All models were compared using GridSearchCV score, defining model accuracy, and F1 score implemented in sklearn library.

LogisticRegression
C=0.01,

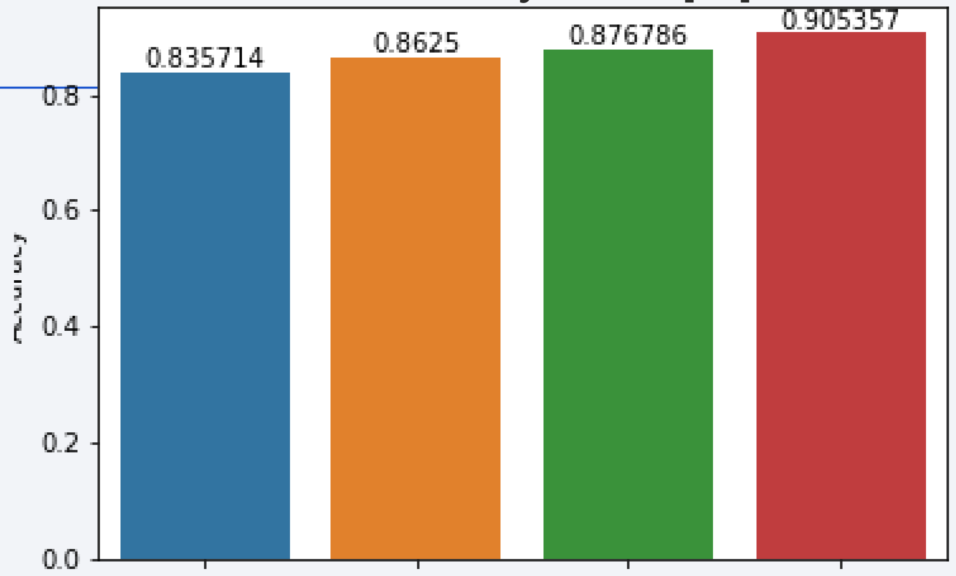
Supported Vector Machine
gamma=0.0316227766016837 9, kernel=sigmoid

K-Nearest Neighbours
k=4, p=1

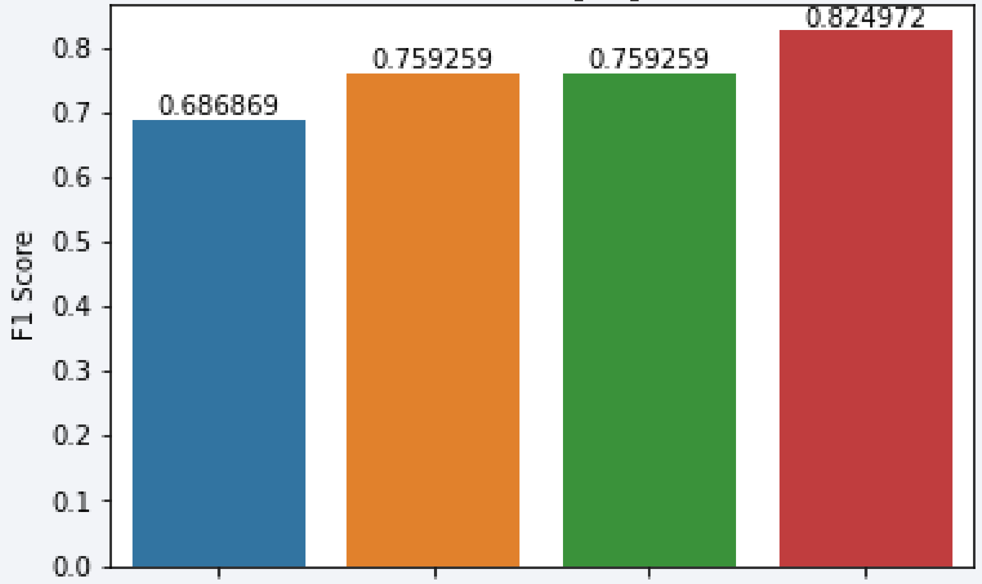
Decision Tree (WINNER)
depth=4, max_features=sqrt, min_samples_split=5

Link to IBM Watson Jupiter Notebook with Machine Learning Prediction
Classification Algorighthm Selection - IBM Cloud A.I.

Grid Search Accuracy Score [%]

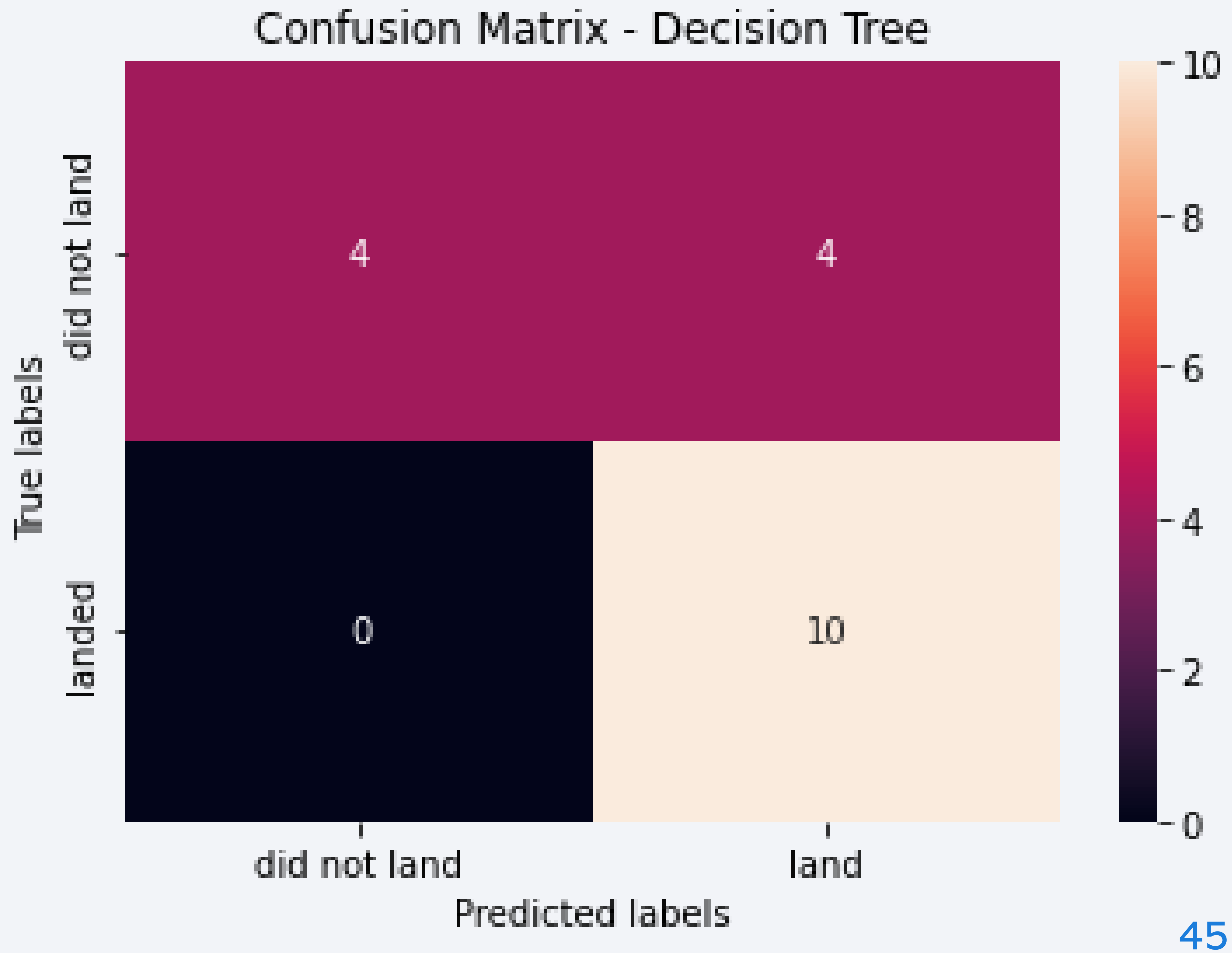


Best models F1 Score [%]



Confusion Matrix

- The confusion matrix for the decision tree classifier shows that the classifier can distinguish between the different classes. The major problem is the false positives .i.e., unsuccessful landing marked as successful landing by the classifier.



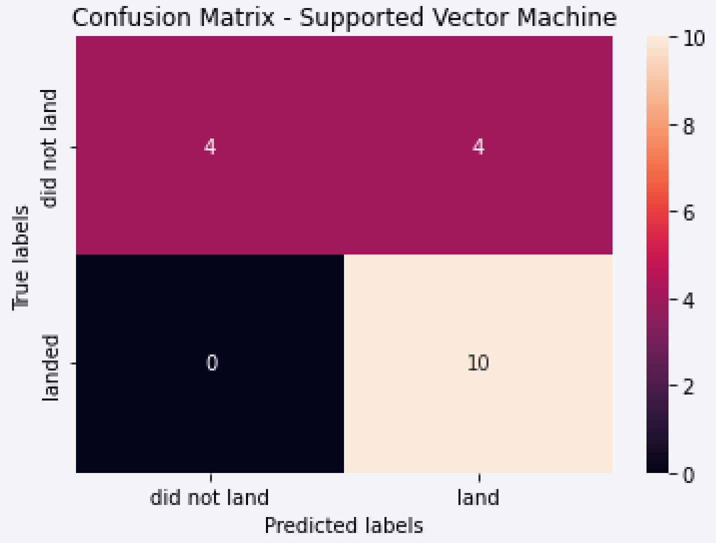
Conclusions

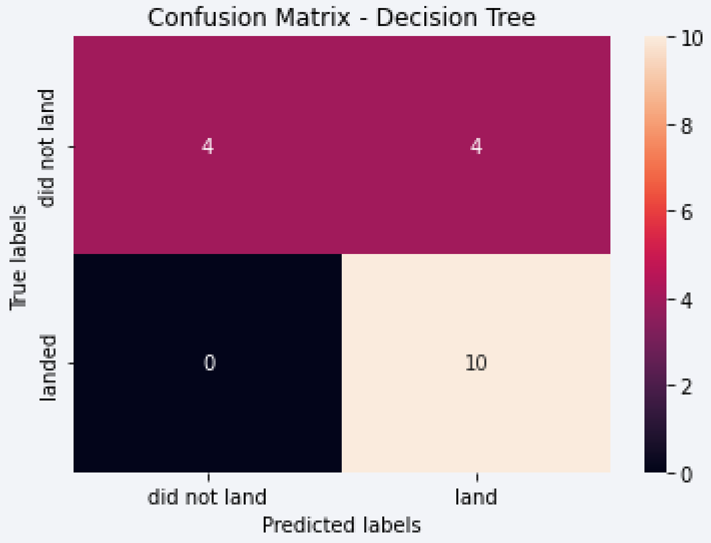
We can conclude that:

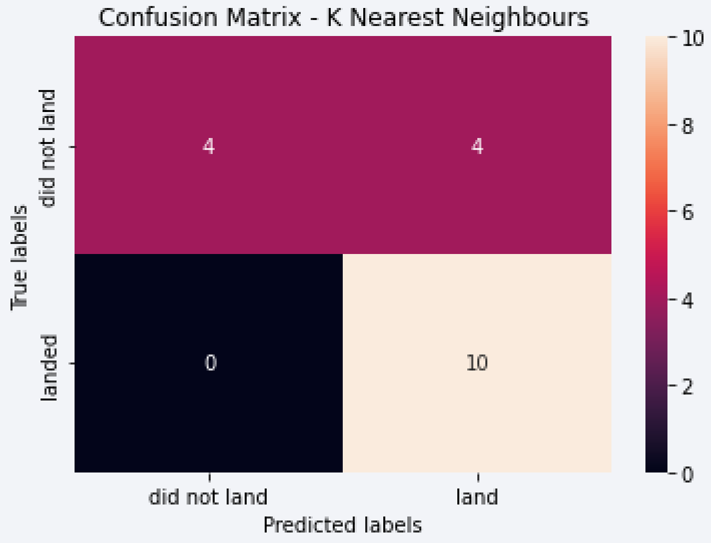
- Booster type and payload have huge impact on whether the rocket will land successfully.
- The larger the flight amount at a launch site, the greater the success rate at a launch site.
- Launch success rate started to increase in 2013 until 2020.
- Orbits ES-L1, GEO, HEO, SSO, VLEO had the most success rate.
- KSC LC-39A had the most successful launches of any sites.
- The Decision tree classifier is the best machine learning algorithm for the task of predicting launch and landing outcome

Confusion Matrix for remaining prediction models

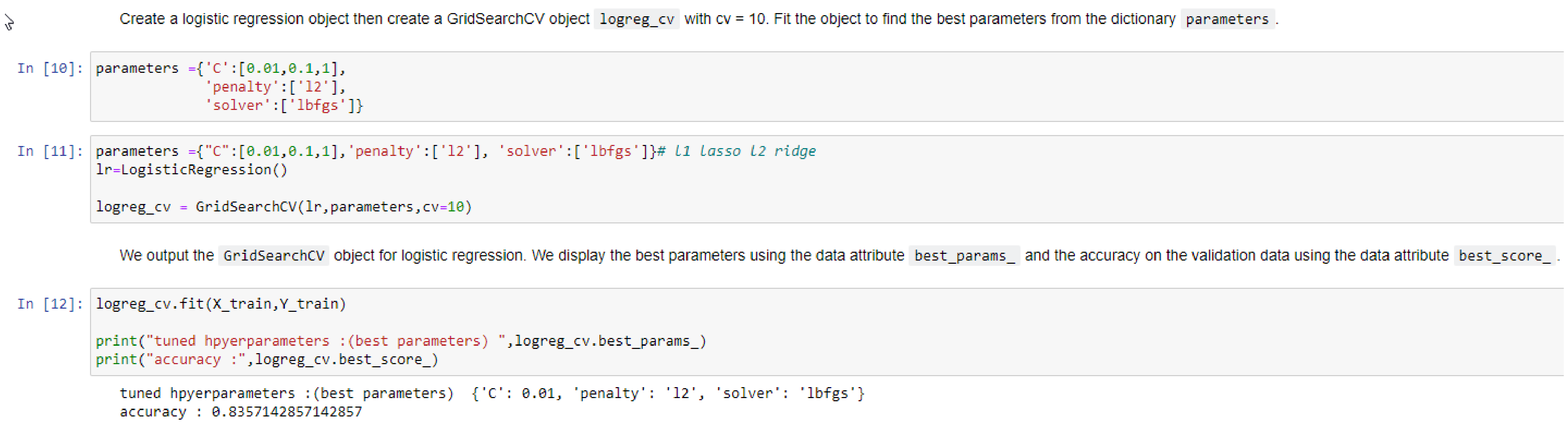
- Include any relevant assets like Python code snippets, SQL queries, charts, Notebook outputs, or data sets that you may have created during this project







Predictive Model Fine Tunning – Logistic Regression



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`.

```
In [10]: parameters ={'C':[0.01,0.1,1],
                      'penalty':['l2'],
                      'solver':['lbfgs']}
```

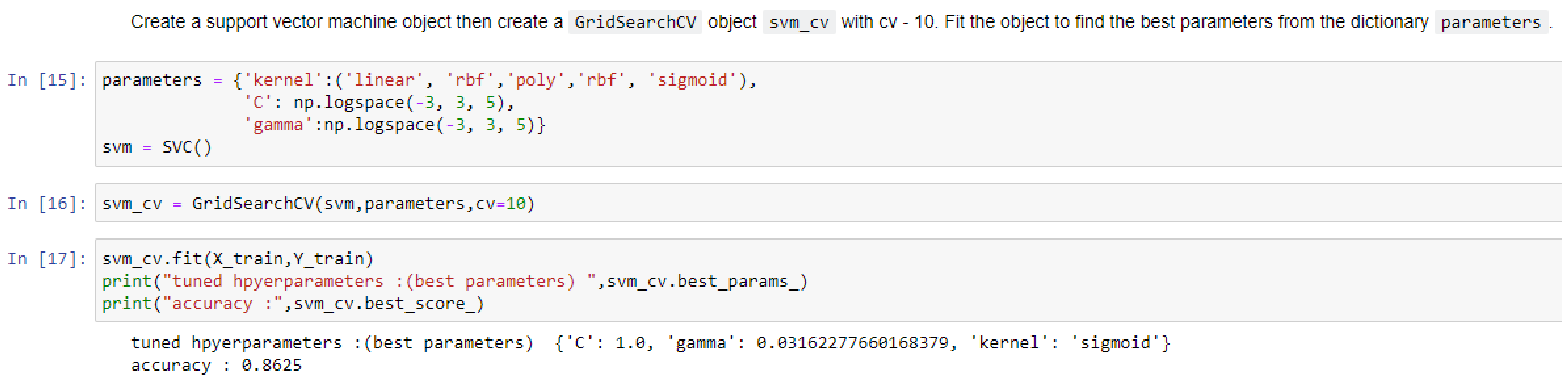
```
In [11]: parameters ={"C":[0.01,0.1,1],'penalty':['l2'], 'solver':['lbfgs']}# l1 lasso l2 ridge
         lr=LogisticRegression()

         logreg_cv = GridSearchCV(lr,parameters,cv=10)
```

We output the `GridSearchCV` object for logistic regression. We display the best parameters using the data attribute `best_params_` and the accuracy on the validation data using the data attribute `best_score_`.

```
In [12]: logreg_cv.fit(X_train,Y_train)

         print("tuned hpyerparameters :(best parameters) ",logreg_cv.best_params_)
         print("accuracy :",logreg_cv.best_score_)
```

```
tuned hpyerparameters :(best parameters)  {'C': 0.01, 'penalty': 'l2', 'solver': 'lbfgs'}
accuracy : 0.8357142857142857
```

Predictive Model Fine Tunning – SVM



Create a support vector machine object then create a `GridSearchCV` object `svm_cv` with cv - 10. Fit the object to find the best parameters from the dictionary `parameters`.

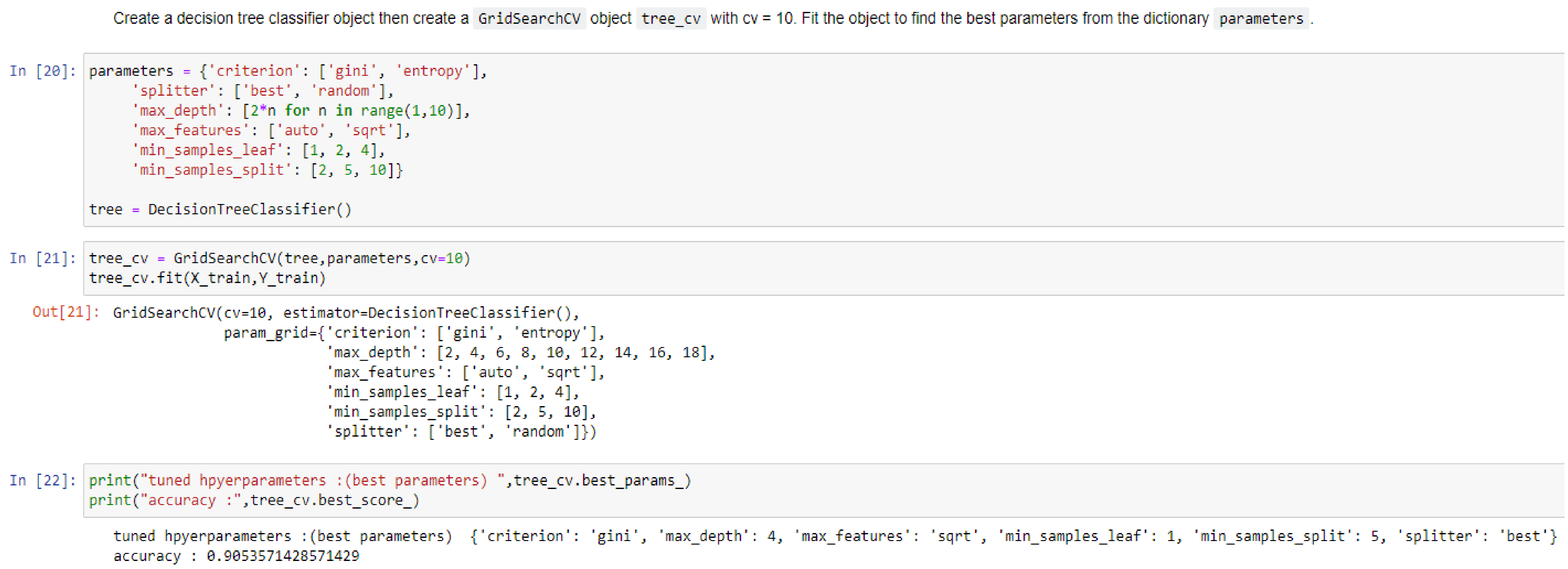
```
In [15]: parameters = {'kernel':('linear', 'rbf','poly','rbf', 'sigmoid'),
                       'C': np.logspace(-3, 3, 5),
                       'gamma':np.logspace(-3, 3, 5)}
         svm = SVC()
```

```
In [16]: svm_cv = GridSearchCV(svm,parameters,cv=10)
```

```
In [17]: svm_cv.fit(X_train,Y_train)
         print("tuned hpyerparameters :(best parameters) ",svm_cv.best_params_)
         print("accuracy :",svm_cv.best_score_)
```

```
tuned hpyerparameters :(best parameters)  {'C': 1.0, 'gamma': 0.03162277660168379, 'kernel': 'sigmoid'}
accuracy : 0.8625
```

Predictive Model Fine Tunning – Decision Tree



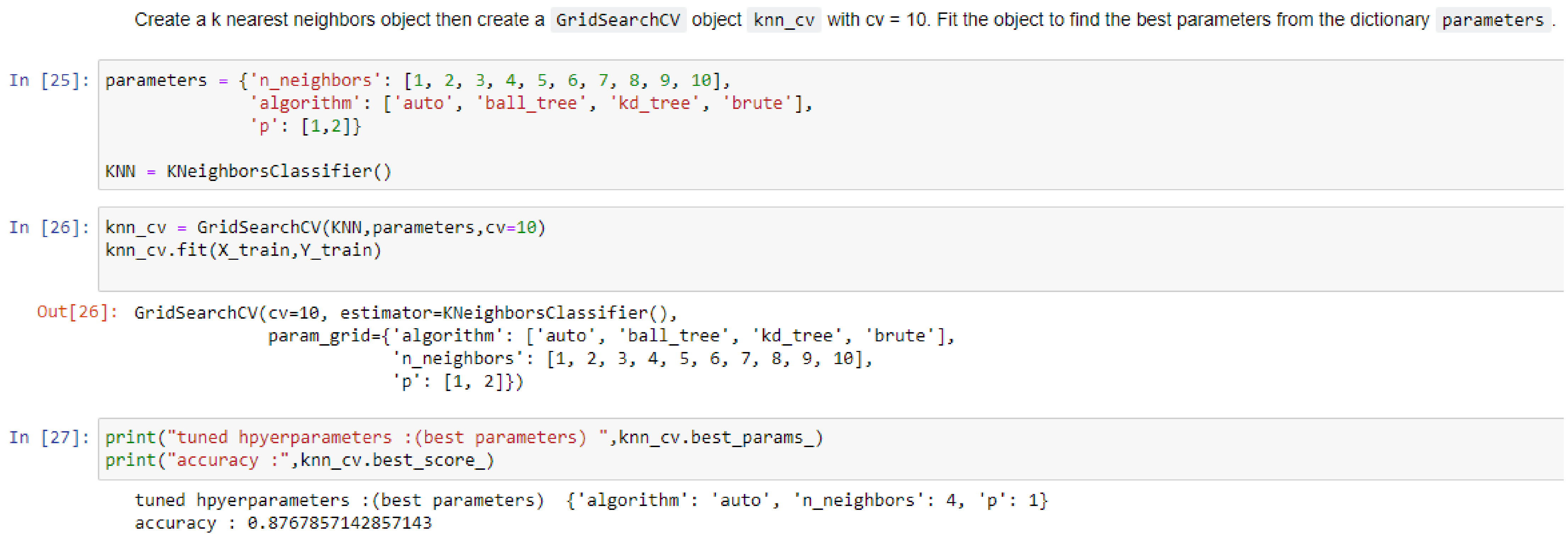
Create a decision tree classifier object then create a `GridSearchCV` object `tree_cv` with cv = 10. Fit the object to find the best parameters from the dictionary `parameters`.

```
In [20]: parameters = {'criterion': ['gini', 'entropy'],
              'splitter': ['best', 'random'],
              'max_depth': [2*n for n in range(1,10)],
              'max_features': ['auto', 'sqrt'],
              'min_samples_leaf': [1, 2, 4],
              'min_samples_split': [2, 5, 10]}

         tree = DecisionTreeClassifier()
```

```
In [21]: tree_cv = GridSearchCV(tree,parameters,cv=10)
         tree_cv.fit(X_train,Y_train)

Out[21]: GridSearchCV(cv=10, estimator=DecisionTreeClassifier(),
                      param_grid={'criterion': ['gini', 'entropy'],
                                  'max_depth': [2, 4, 6, 8, 10, 12, 14, 16, 18],
                                  'max_features': ['auto', 'sqrt'],
                                  'min_samples_leaf': [1, 2, 4],
                                  'min_samples_split': [2, 5, 10],
                                  'splitter': ['best', 'random']})
```

```
In [22]: print("tuned hpyerparameters :(best parameters) ",tree_cv.best_params_)
         print("accuracy :",tree_cv.best_score_)

    tuned hpyerparameters :(best parameters)  {'criterion': 'gini', 'max_depth': 4, 'max_features': 'sqrt', 'min_samples_leaf': 1, 'min_samples_split': 5, 'splitter': 'best'}
    accuracy : 0.9053571428571429
```

Predictive Model Fine Tunning – K Nearest Neighbours



Create a k nearest neighbors object then create a `GridSearchCV` object `knn_cv` with cv = 10. Fit the object to find the best parameters from the dictionary `parameters` .

In [25]:
```
parameters = {'n_neighbors': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
              'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute'],
              'p': [1,2]}

KNN = KNeighborsClassifier()
```

In [26]:
```
knn_cv = GridSearchCV(KNN,parameters,cv=10)
knn_cv.fit(X_train,Y_train)
```

Out[26]:
```
GridSearchCV(cv=10, estimator=KNeighborsClassifier(),
             param_grid={'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute'],
                         'n_neighbors': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
                         'p': [1, 2]})
```

In [27]:
```
print("tuned hpyerparameters :(best parameters) ",knn_cv.best_params_)
print("accuracy :",knn_cv.best_score_)
```

```
tuned hpyerparameters :(best parameters)  {'algorithm': 'auto', 'n_neighbors': 4, 'p': 1}
accuracy : 0.8767857142857143
```

Thank you!