

IBM

Nick Belgau
9/7/21

SPACEX

IBM Data Science Capstone



Outline

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

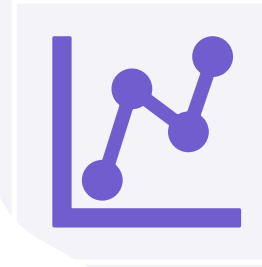


Executive Summary



Summary of methodologies

Data collection
Data wrangling
EDA with data visualization
EDA with SQL
Building an interactive map with Folium
Building an interactive dashboard with Plotly Dash
Predictive analysis (classification)



Summary of all results

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

https://github.com/belgau/spacex_capstone




Introduction

Project Background

- This project predicts if the Falcon 9 first stage will land successfully. SpaceX advertises the Falcon 9 rocket launches with a cost of \$62MM whereas other providers cost upwards to \$165MM. If it can be determined that the first stage will land successfully, then the cost of the launch can be determined. This information can be used to keep SpaceX competitive for rocket launches.

Common Problems

- What influences if the rocket will land successfully?
 - What is the relationship with certain rocket variables in determining the success rate of a landing?
 - What conditions does SpaceX need to have the highest success landing rate?
- 

METHODOLOGY



Methodology

Data collection methodology

- SpaceX Rest API
- Web scrapping launch sites

Perform data wrangling

- Data was transformed into a Feature Matrix for Machine Learning
- LabelEncoder to transform categorical variables for ML and dropping irrelevant columns

Perform EDA using visualization and SQL

- Plotting using Seaborn & Matplotlib
- Scatter graphs & bar graphs to show patterns in the data

Interactive visual analytics using Folium and Plotly Dash

- Web hosted dashboard

Perform predictive analysis using classification models

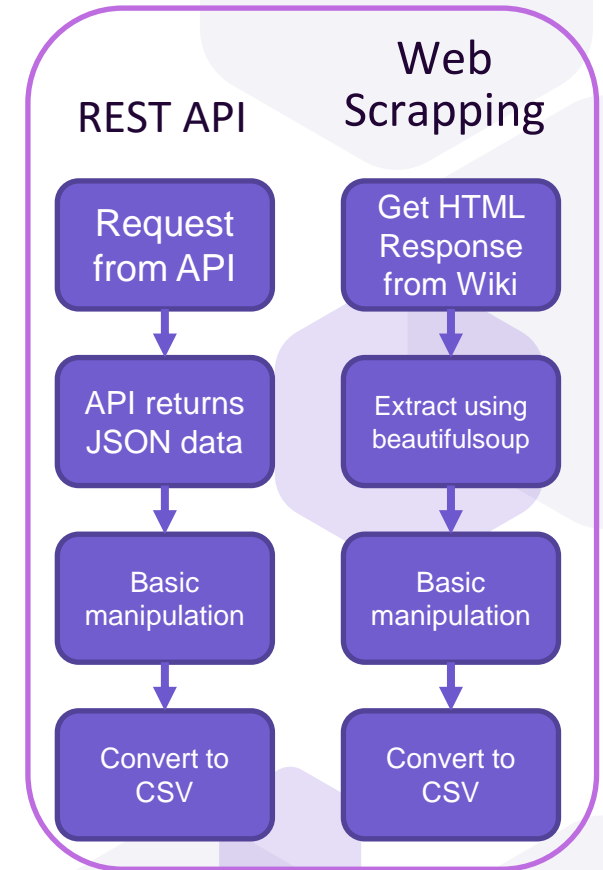
- How to build, tune, evaluate classification models

Data Collection Process Overview

To predict whether a SpaceX rocket will attempt to land a rocket or not, data was compiled from multiple sources to build the dataset.

- Launch data was available from the SpaceX REST API
 - This provided data about launches, rocket use, payload, launch specs, landing specs, and landing outcome
 - <https://api.spacexdata.com/v4/launches/past>
- Additional data was scrapped from Wikipedia using Beautiful Soup to supplement the launch data

Dataset



Data Collection – SpaceX REST API

1. Get response from HTML

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

response = requests.get(spacex_url)
```

2. Convert response to JSON

```
static_json_url='https://cf-courses-data.s3.us-west-2.amazonaws.com/02-spacex-api-dataset/static_json_data.json'

response = requests.get(static_json_url)

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

3. Apply custom function to manipulate data

```
getLaunchSite(data)

getPayloadData(data)

getCoreData(data)
```

4. Assign list to dictionary

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

6. Convert to dictionary to dataframe

```
data_falcon9 = pd.DataFrame(launch_dict)
```

7. Filter dataframe to only show 'Falcon 9'

```
data_falcon9 = data_falcon9[data_falcon9.BoosterVersion != 'Falcon 1']
```

8. Deal with Missing Values

```
# Calculate the mean value of PayloadMass column
PayloadMass_mean = data_falcon9['PayloadMass'].mean()
print("PayloadMass mean: ", PayloadMass_mean)

# Replace the np.nan values with its mean value
data_falcon9['PayloadMass'].replace(np.nan, PayloadMass_mean)
```

9. Export dataframe as .CSV

```
data_falcon9.to_csv('dataset_part\1.csv', index=False)
```


Data Collection – Web Scrapping

1. Get response from HTML

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

2. Create BeautifulSoup object

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

3. Finding the HTML web tables

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

4. Get column names

```
temp = soup.find_all('th')
for x in range(len(temp)):
    try:
        name = extract_column_from_header(temp[x])
        if (name is not None and len(name) > 0):
            column_names.append(name)
    except:
        pass
```

5. Create dictionary

```
launch_dict= dict.fromkeys(column_names)

# Remove an irrelevant column
del launch_dict['Date and time ( )']

# Let's initial the launch_dict with each value to be an empty list
launch_dict['Flight No.'] = []
launch_dict['Launch site'] = []
launch_dict['Payload'] = []
launch_dict['Payload mass'] = []
launch_dict['Orbit'] = []
launch_dict['Customer'] = []
launch_dict['Launch outcome'] = []
# Added some new columns
launch_dict['Version Booster']=[]
launch_dict['Booster landing']=[]
launch_dict['Date']=[]
launch_dict['Time']=[]
```

6. Appending data to keys

```
extracted_row = 0
#Extract each table
for table_number,table in enumerate(soup.find_all('table',"wikitable plain")):
    # get table row
    for rows in table.find_all("tr"):
        #check to see if first table heading is as number corresponding
        if rows.th:
            if rows.th.string:
                flight_number=rows.th.string.strip()
                flag=flight_number.isdigit()
            else:
                flag=False

        row=rows.find_all('td') #get table element

        #if it is number save cells in a dictionary
        if flag:
            extracted_row += 1

            # Flight Number value
            launch_dict["Flight No."].append(flight_number)
            datatimelist=date_time(row[0])
```

7. Convert dictionary to dataframe

```
df = pd.DataFrame.from_dict(launch_dict)
```

8. Export dataframe as .CSV

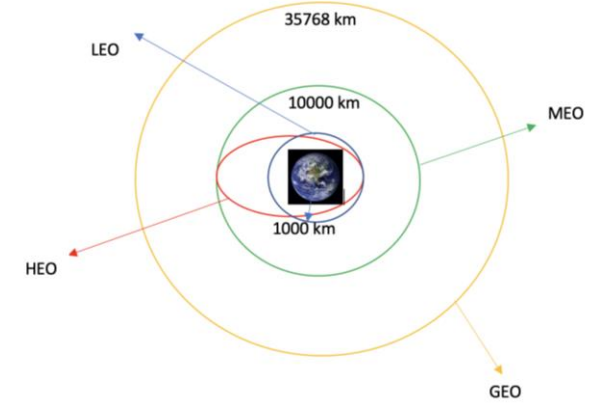
```
df.to_csv('spacex_web_scraped.csv', index=False)
```

Data Wrangling

Once the CSV dataset was imported and converted to a pandas dataframe, a brief inspection of the following characteristics of the data was conducted

- For each column, the data type and percentage of missing values was inspected
- Using `value_counts`, the following was determined
 - Number of launches on each site
 - Number of occurrence by orbit
 - Number of mission
- The various outcomes were classified as either success or failure and assigned as a Boolean to a new dataframe column.
 - Success: 1
 - Failure: 0
- The success rate of the overall dataset could be determined, but will be further dissected in deeper data analyses techniques later

https://github.com/belgau/spacex_capstone/tree/main/Data_Wrangling



```
df.isnull().sum()/df.count()*100
```

FlightNumber	0.000
Date	0.000
BoosterVersion	0.000
PayloadMass	0.000
Orbit	0.000
LaunchSite	0.000
Outcome	0.000
Flights	0.000
GridFins	0.000
Reused	0.000
Legs	0.000
LandingPad	40.625
Block	0.000
ReusedCount	0.000
Serial	0.000
Longitude	0.000
Latitude	0.000
dtype:	float64

```
df.dtypes
```

FlightNumber	int64
Date	object
BoosterVersion	object
PayloadMass	float64
Orbit	object
LaunchSite	object
Outcome	object
Flights	int64
GridFins	bool
Reused	bool
Legs	bool
LandingPad	object
Block	float64
ReusedCount	int64
Serial	object
Longitude	float64
Latitude	float64
dtype:	object

```
df['LaunchSite'].value_counts()
```

CCAFS SLC 40	55
KSC LC 39A	22
VAFB SLC 4E	13

Name: LaunchSite, dtype: int64



EDA with Data Visualization

- **Seaborn**, which is a high-level plotting library on top of matplotlib, was used for statistical figures
 - ◊ Various forms of **Scatterplots** were used to visually represent relationships of different variables and show how much one variable affects the other
 - ◊ **Bar Charts** were also used to display numerical stats, delineated by a categorical variable such as orbit type
- After EDA, a **feature matrix** of the best variables was developed
- Categorical variables were considered candidates for **OneHotEncoding**, but were determined to be non-binary.



EDA with SQL

- Data was uploaded from CSV to SQL database
- Various queries were executed to pull specific data and the corresponding landing outcome



- More information regarding these queries can be found at my GitHub

https://github.com/belgau/spacex_capstone/tree/main/EDA_SQL

Interactive Map & Launch Site Proximity Analysis



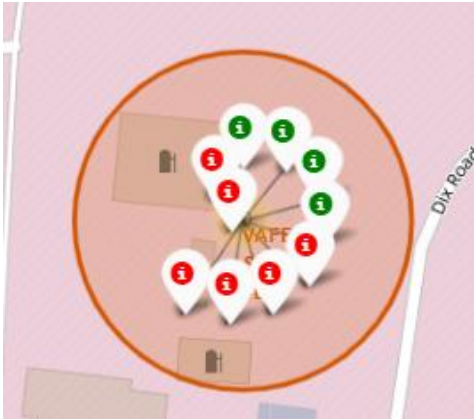
The web scrapped latitudinal and longitudinal coordinates from Wikipedia were used to develop an interactive map using folium.

Interactive Map Generation

- ◆ The initial map object was defined
- ◆ All launch sites were pinned on the map
- ◆ Each launch site was labeled with success or failure markers



	Launch Site	Lat	Long
0	CCAFS LC-40	28.562302	-80.577356
1	CCAFS SLC-40	28.563197	-80.576820
2	KSC LC-39A	28.573255	-80.646895
3	VAFB SLC-4E	34.632834	-120.610746



Launch Site Proximity Analysis

- ◆ The data was explored to investigate if there was a relationship between distance to nearby objects and success rate
- ◆ For example, distance to highway and distance to Orlando was measured

Interactive Web Dashboard



Dash was used to develop an interactive dashboard hosted on the web

Features of the dashboard include

- ◆ Pie Chart
 - ◇ Displays total successful launches by site
 - ◇ Dropdown to filter pie chart by site to show successful launch rate
- ◆ Scatterplot
 - ◇ Channels all version types by success, plotted against payload
 - ◇ Slider bar to filter payload range

PREDICTIVE ANALYSIS (CLASSIFICATION)

Model Building Process

- Load dataset from csv to pandas dataframe
- Define feature matrix (X) and response variable (Y)
- Preprocessing of the data: standardize the feature matrix
- Split data into training and test data sets
- Identify ML algorithms to use: **Logistic Regression, SVM, Decision Tree, KNN**
- Train each model type while using GridSearchCV to optimize hyperparameters and solver methods

Feature matrix

```
data.columns
```

```
|: Index(['FlightNumber', 'Date', 'BoosterVersion', 'PayloadMass', 'Orbit',  
        'LaunchSite', 'Outcome', 'Flights', 'GridFins', 'Reused', 'Legs',  
        'LandingPad', 'Block', 'ReusedCount', 'Serial', 'Longitude', 'Latitude',  
        'Class'],  
        dtype='object')
```

Response Variable

```
print(data['Class'].value_counts())
```

```
1    60  
0    30  
Name: Class, dtype: int64
```

1: Success
0: Failure

PREDICTIVE ANALYSIS (CLASSIFICATION)

Evaluating Each model

- Output the tuned hyperparameters from GridSearchCV using model.**best_params_** method
- Output the accuracy of each model using model.**best_score_** method
- Plot the confusion matrix

Finding the Best Model

- Best performing model = highest accuracy of training set
- A function was developed to automatically rank them
- Having this function is helpful for iterative processes where feature matrix improvement and tuning continue

```
algorithms = {'KNN':knn_cv.best_score_,'Decision Tree':tree_cv.best_score_,'LogisticRegression':logreg_cv.best_score_}
bestalgorithm = max(algorithms, key=algorithms.get)
print('Best Algorithm:',bestalgorithm,'with a score of',algorithms[bestalgorithm])
```

Best Algorithm: Decision Tree with a score of 0.8892857142857142

```
if bestalgorithm == 'Decision Tree':
    print('Best Parameters:',tree_cv.best_params_)
if bestalgorithm == 'KNN':
    print('Best Parameters:',knn_cv.best_params_)
if bestalgorithm == 'LogisticRegression':
    print('Best Parameters:',logreg_cv.best_params_)
```

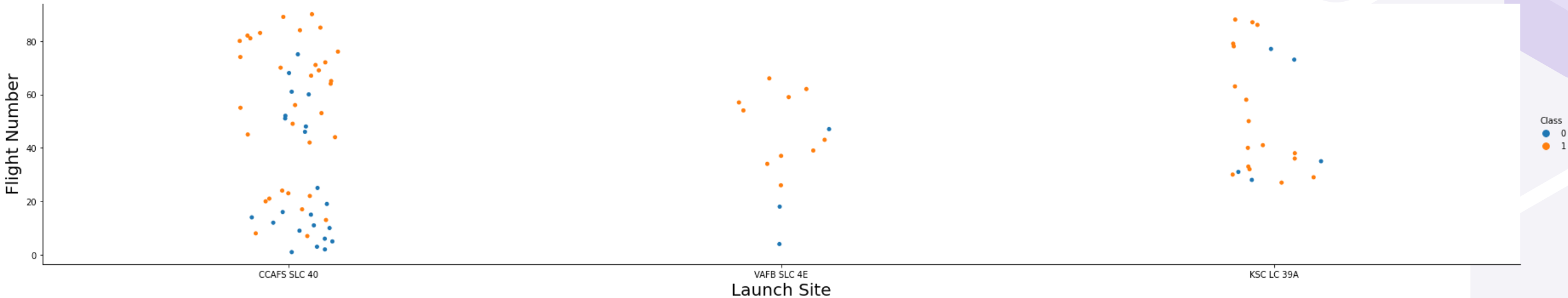
Best Parameters: {'criterion': 'gini', 'max_depth': 10, 'max_features': 'auto', 'min_samples_leaf': 2, 'min_samples_s

RESULTS



Flight Number Vs. Launch Site

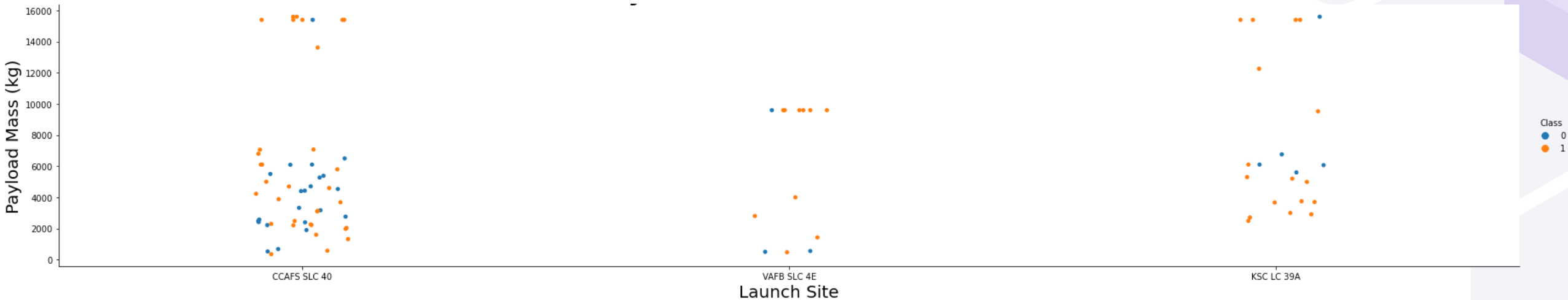
EDA with Data Visualization



- Flights became more successful with more launches at each site.
- CCAFS SLC-40 had mostly unsuccessful launches at the beginning.

Payload vs. Launch Site

EDA with Data Visualization



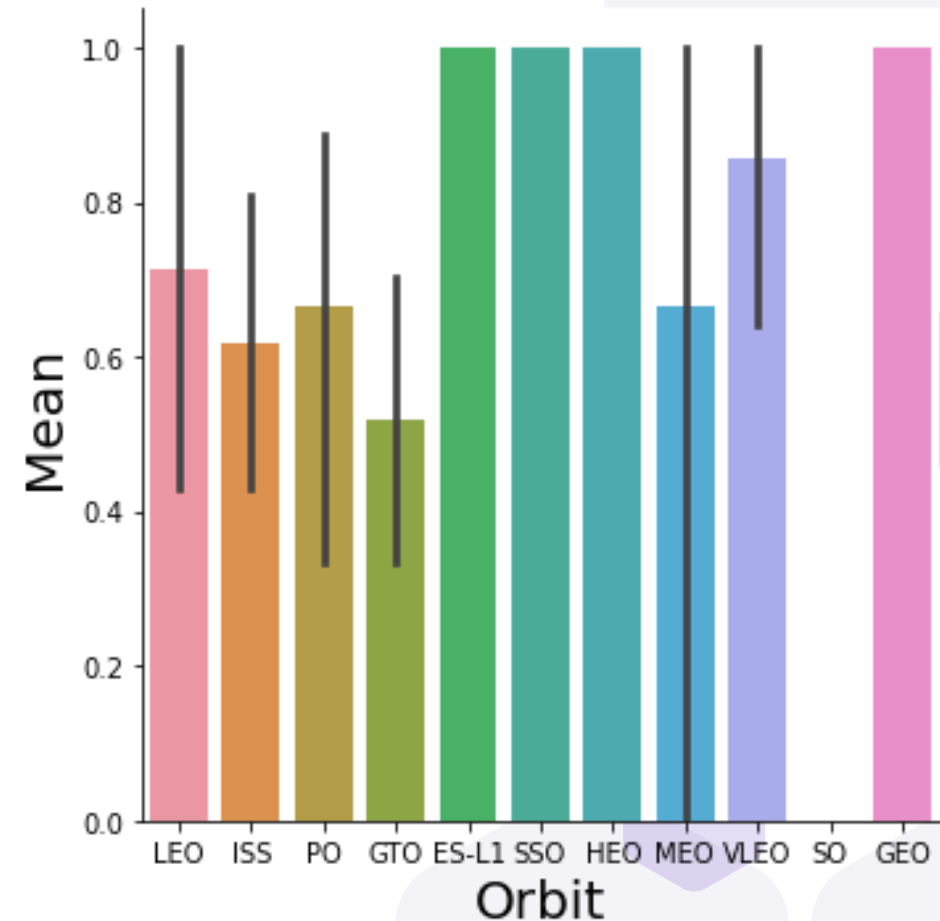
- VAFB SLC-4E typically launched payloads around 10,000 kg and had a few smaller loads as well.
- The payload distribution at CCAFS SLC-40 was bimodal split at <8,000 kg and 14,000-15,000 kg
- Heavier payloads appears to have fairly high success rate, but this may be a function of another variable like time.
- CCAFS SLC-40 appears to have the lowest success rate when considering all payload ranges.

https://github.com/belgau/spacex_capstone/tree/main/EDA_Data_Visualization

Orbit Success Rate

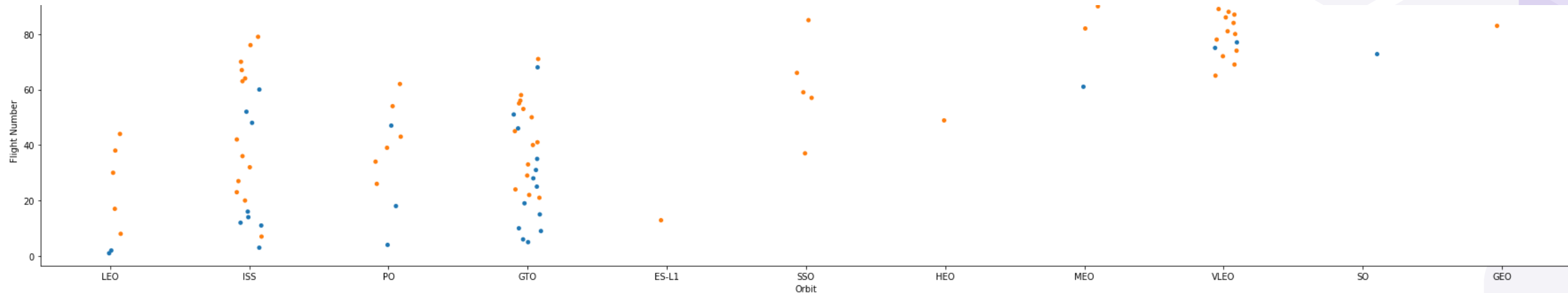
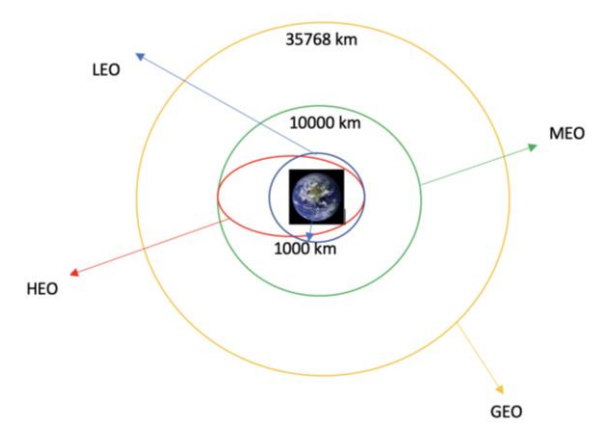
EDA with Data Visualization

- Several orbits had 100% success!
 - ES-L1, SSO, HEO, GEO
- However, they lacked clear confidence intervals, so more data may be required to make a concrete conclusion



Flight Number vs. Orbit

EDA with Data Visualization



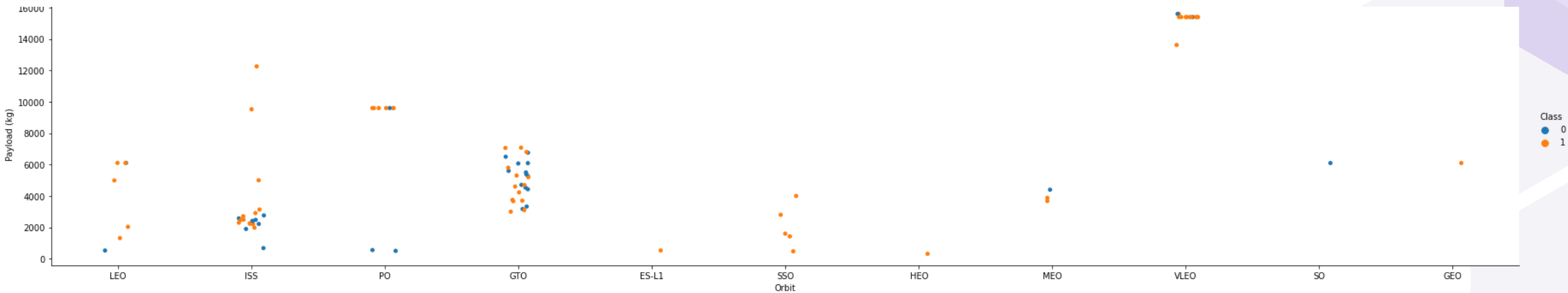
- LEO appears to be related to flight number since it is evenly spaced on the swarm plot.
- The highest orbit, GEO, was a more recent flight.
- ISS flights occurred throughout the entire flight number range.
- Although, many ISS flights were unsuccessful – it appears to have a lower success rate.
- SSO was the only orbit that was conducted more than once (5 total) while maintaining 100% success.

https://github.com/belgau/spacex_capstone/tree/main/EDA_Data_Visualization



Payload vs. Orbit

EDA with Data Visualization

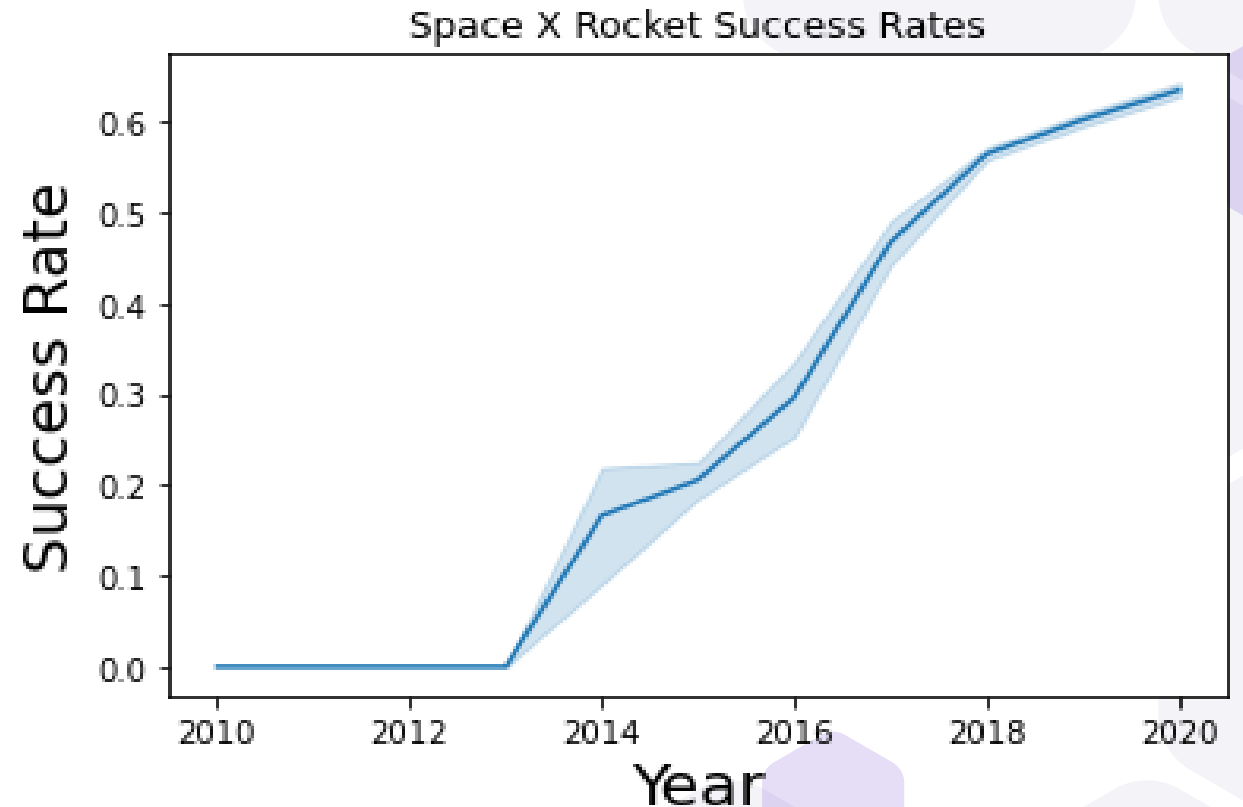


- The heaviest payloads were on the VLEO orbit
- However, payload has a negative influence on GTO, SSO, MEO, and HEO orbits.
 - This statement is most conclusive on GTO and SSO orbits since it has significant samples
- PO orbits tend to be bimodal, with the most occurring at 10,000 kg (and successful!)

Success Rate vs. Year

EDA with Data Visualization

- Success rate clearly increases over time!
- The variability in the success rate appears to be tightening over time.
- This indicates that not only is SpaceX becoming more successful, but they are becoming more consistent!
- The success Rate does not appear to be linear which may just be a nature of the compiled metric.



Launch Sites

EDA with SQL

Different Launch Sites

- Querying the **local** SQL database for unique launch sites
- These launch sites can be paired with the geographical coordinates

```
MySQL localhost:33060+ ssl generaldb SQL > SELECT DISTINCT launch_site from spacex ;
```

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

Launch Site Names Begin with 'KSC'

- Jupyter Notebook** with MySQL plugin was used for further queries to speed up the process
- This script finds sites that match the partial name provided

	Date	Time_UTC	Booster_Version	Launch_Site	Payload	PAYLOAD_MASS_KG_	Orbit
0	19-02-2017	2021-07-02 14:39:00.0000000	F9 FT B1031.1	KSC LC-39A	SpaceX CRS-10	2490	LEO (ISS)
1	16-03-2017	2021-07-02 06:00:00.0000000	F9 FT B1030	KSC LC-39A	EchoStar 23	5600	GTO
2	30-03-2017	2021-07-02 22:27:00.0000000	F9 FT B1021.2	KSC LC-39A	SES-10	5300	GTO
3	01-05-2017	2021-07-02 11:15:00.0000000	F9 FT B1032.1	KSC LC-39A	NROL-76	5300	LEO
4	15-05-2017	2021-07-02 23:21:00.0000000	F9 FT B1034	KSC LC-39A	Inmarsat-5 F4	6070	GTO

```
select TOP 5 * from tblSpaceX WHERE Launch_Site LIKE 'KSC%'
```

https://github.com/belgau/spacex_capstone/tree/main/EDA_SQL

Payload Mass

EDA with SQL

Total Payload Mass

- This script finds the total payload mass of boosters launched by NASA

```
select SUM(PAYLOAD_MASS_KG_) TotalPayloadMass from tblSpaceX where Customer = 'NASA (CRS)', 'TotalPayloadMass')
```

Total Payload Mass

45596

Average Payload Mass by F9 v1.1

- This script finds the average payload mass of the F9 version 1.1

```
select AVG(PAYLOAD_MASS_KG_) AveragePayloadMass from tblSpaceX where Booster_Version = 'F9 v1.1', 'AveragePayloadMass')
```

Average Payload Mass

2928

Boosters Carried Maximum Payload

- This script finds the max payload by using a **subquery**

```
"SELECT DISTINCT Booster_Version, MAX(PAYLOAD_MASS_KG_) AS [Maximum Payload Mass] FROM tblSpaceX GROUP BY Booster_Version ORDER BY [Maximum Payload Mass] DESC")
```

	Booster_Version	Maximum Payload Mass
0	F9 B5 B1048.4	15600
1	F9 B5 B1048.5	15600
2	F9 B5 B1049.4	15600
3	F9 B5 B1049.5	15600
4	F9 B5 B1049.7	15600

https://github.com/belgau/spacex_capstone/tree/main/EDA_SQL



Successful Landings

EDA with SQL

First Successful Drone Ship Landing Date

```
select MIN(Date) SLO from tblSpaceX where Landing_Outcome = 'Success (drone ship)', 'SLO')
```

Successful Ground Pad Landings of Boosters with Payload Mass Between 4,000 -6,000 kg

```
select Booster_Version from tblSpaceX where Landing_Outcome = 'Success (ground pad)' AND Payload_MASS_KG_ > 4000 AND Pa
AND Payload_MASS_KG_ < 6000", 'Booster_Version')
```

Total Number of Successful and Failure Mission Outcomes

```
SELECT(SELECT Count(Mission_Outcome) from tblSpaceX where Mission_Outcome LIKE '%Success%') as Successful
_Mission_Outcomes,(SELECT Count(Mission_Outcome) from tblSpaceX where Mission_Outcome LIKE '%Failure%') as Fa
e_Mission_Outcomes"
```

Successful_Mission_Outcomes	Failure_Mission_Outcomes
0	1

Launch Records

EDA with SQL

First Successful Drone Ship Landing Date

```
cursor.execute("SELECT DateName( month , DateAdd( month , MONTH(CONVERT(date,Date, 105)) , 0 ) - 1 ) as Month, Booster_Version, Launch_Site, Landing_Outcome FROM tblSpaceX WHERE (Landing_Outcome LIKE N'%Success%') AND YEAR(CONVERT(date,Date, 105)) = '2015'")
```

	Month	Booster_Version	Launch_Site	Landing_Outcome
0	January	F9 FT B1029.1	VAFB SLC-4E	Success (drone ship)
1	February	F9 FT B1031.1	KSC LC-39A	Success (ground pad)
2	March	F9 FT B1021.2	KSC LC-39A	Success (drone ship)
3	May	F9 FT B1032.1	KSC LC-39A	Success (ground pad)
4	June	F9 FT B1035.1	KSC LC-39A	Success (ground pad)
5	June	F9 FT B1029.2	KSC LC-39A	Success (drone ship)

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

```
db.GetRecordsOfColumn("SELECT COUNT(Landing_Outcome) AS s1 FROM dbo.tblSpaceX WHERE (Landing_Outcome LIKE '%Success%') AND (Date > '04-06-2010') AND (Date < '20-03-2017')", 's1')
```

0

34

Launch Site Proximity Analysis with Folium

```
#Distance to Highway
coordinates = [
    [28.56342, -80.57674],
    [28.411780, -80.820630]]

lines=folium.PolyLine(locations=coordinates, weight=1)
site_map.add_child(lines)
distance = calculate_distance(coordinates[0][0], coordinates[0][1], co
distance_circle = folium.Marker(
    [28.411780, -80.820630],
    icon=DivIcon(
        icon_size=(20,20),
        icon_anchor=(0,0),
        html='<div style="font-size: 12; color:#252526;"><b>%s</b></div>'
    )
)
site_map.add_child(distance_circle)
site_map
```

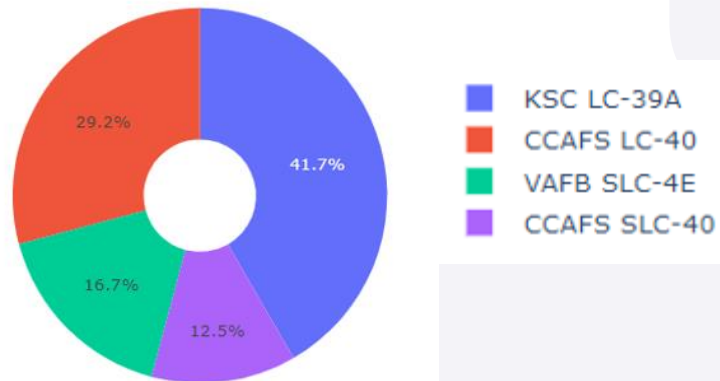
Significant Findings

- Are launch sites in close proximity to railways → No
- Are launch sites in close proximity to highways → No
- Are launch sites in close proximity to coastline → Yes
- Do launch sites keep certain distance away from cities → Yes

Interactive Dashboard (Plotly & Dash)

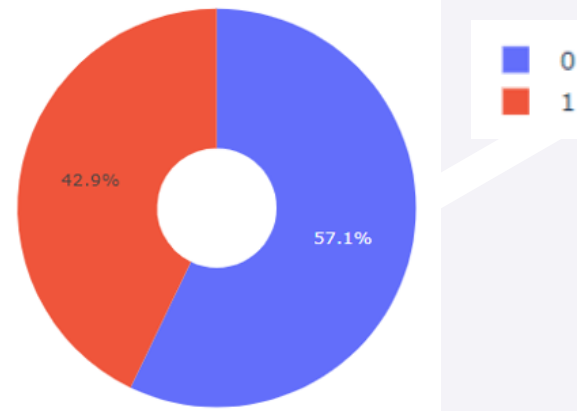
Launch Success by Site

Total Success Launches By all sites



- KSV LC-39A had the most successful launches, totaling 41.7% of the 'successful launches' of the total successful launches.
- This is not to be confused with the success rate of the site

Total Success Launches for site CCAFS SLC-40



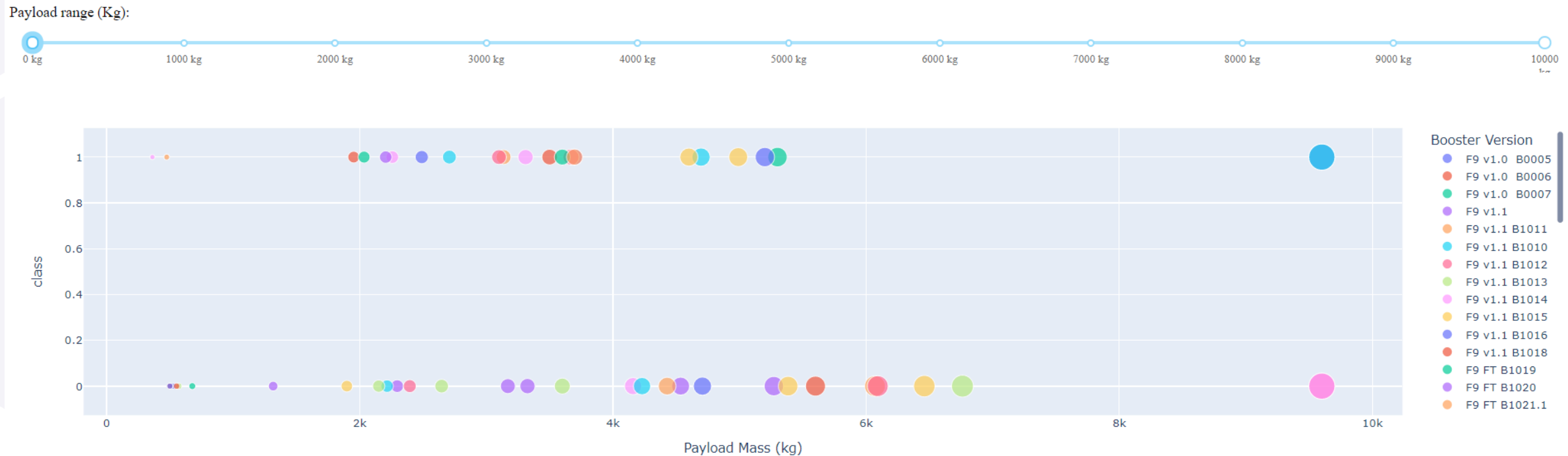
- CCAFS SLC-40 had the highest successful launch rate at 42.9%
- This is insightful data that can only be discovered by *zooming in* deeper into the data!

https://github.com/belgau/spacex_capstone/tree/main/Interactive_Web_Dashboard

Interactive Dashboard

Launch Rate vs. Payload Overview

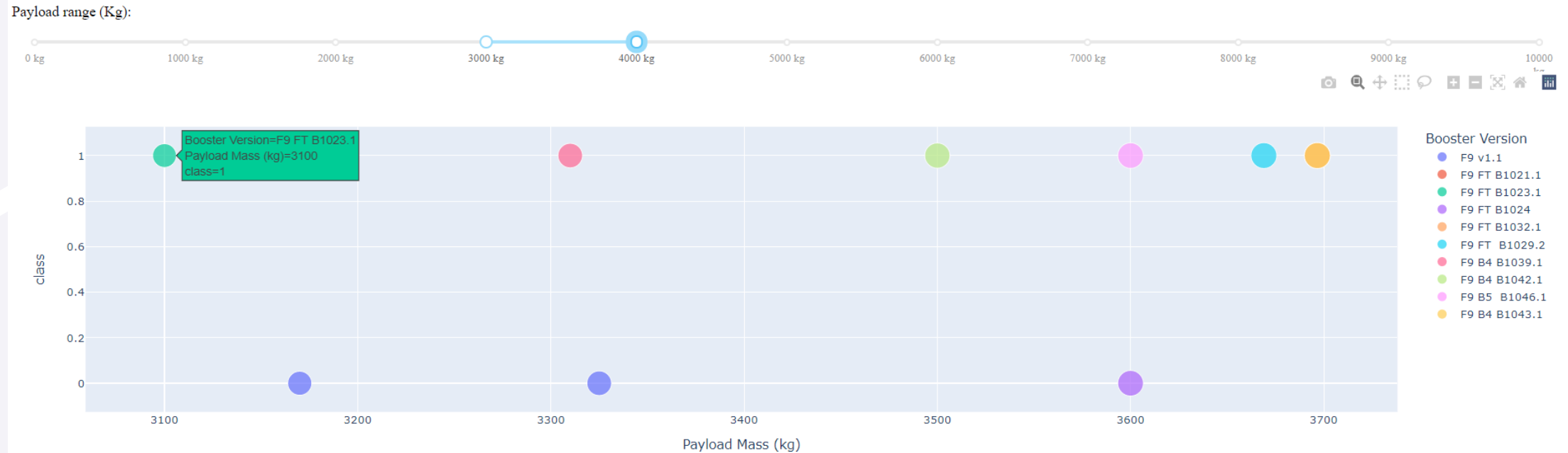
- ◆ This displays an overview of scatterplot
... now when we filter....



https://github.com/belgau/spacex_capstone/tree/main/Interactive_Web_Dashboard

Interactive Dashboard

Launch Rate vs. Payload



- The highest successful launch rate was for payloads between 3,000 – 4,000 kg

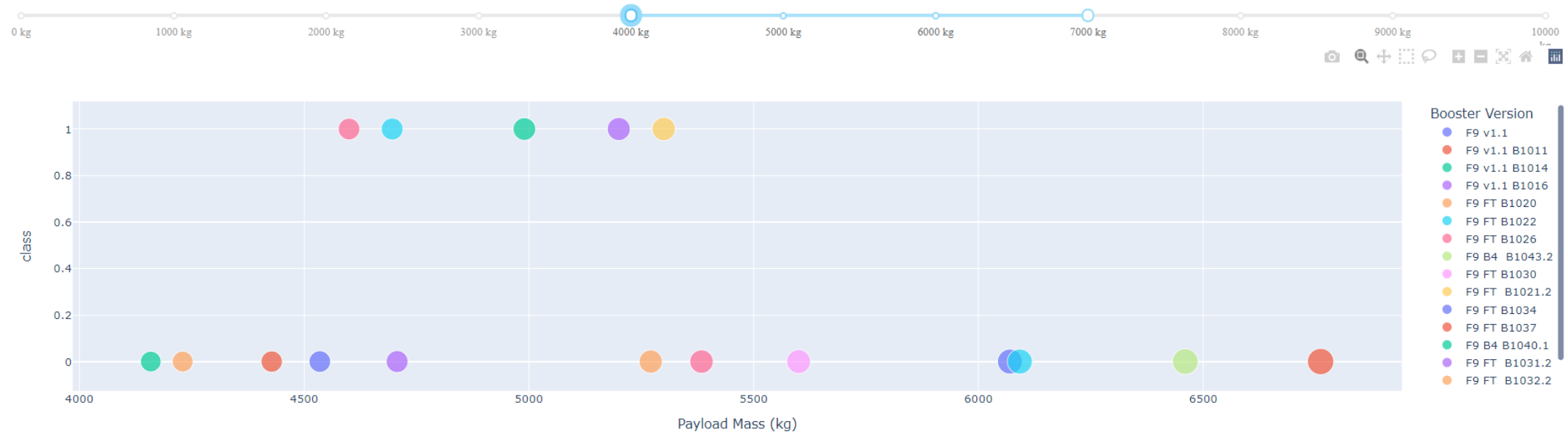
https://github.com/belgau/spacex_capstone/tree/main/Interactive_Web_Dashboard

Interactive Dashboard

Launch Rate vs. Payload

- The lowest successful launch rate was for payloads between 3,000 – 4,000 kg at 41.67%
- However... 0 - 2,000 kg is a close second at 42.86%

Payload range (Kg):

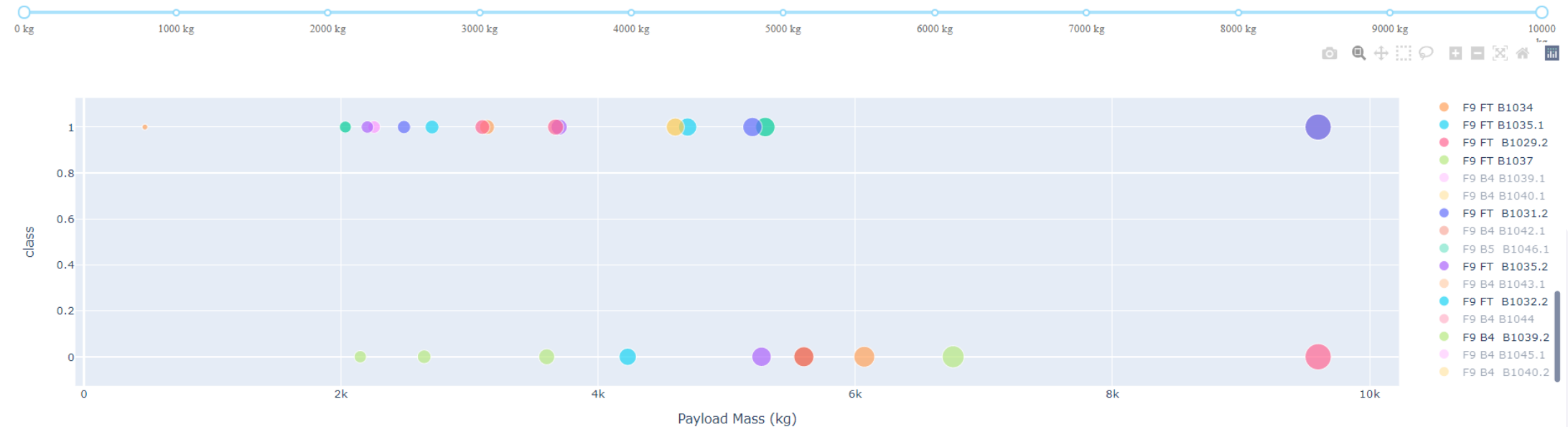


https://github.com/belgau/spacex_capstone/tree/main/Interactive_Web_Dashboard

Interactive Dashboard

Launch Rate vs. Booster Type

Payload range (Kg):



- The F9 Booster Version 'FT' had the highest successful launch rate at 62.5%.

https://github.com/belgau/spacex_capstone/tree/main/Interactive_Web_Dashboard

PREDICTIVE ANALYSIS (CLASSIFICATION)

- From EDA, the following feature matrix was developed for model training and success rate prediction

FlightNumber	PayloadMass	Orbit	LaunchSite	Flights	GridFins	Reused	Legs	LandingPad	Block	ReusedCount	Serial
--------------	-------------	-------	------------	---------	----------	--------	------	------------	-------	-------------	--------

- Non-binary categorical variables were converted to numerical values using **LabelEncoder**
- The encodings were saved in dictionaries for later reference

```
from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()

label_encoder.fit(features['Orbit'])
Orbit_mapping = dict(zip(label_encoder.classes_, label_encoder.transform(label_encoder.classes_)))

features['Orbit'] = label_encoder.fit_transform(features['Orbit'])

Orbit_mapping
```

```
{'ES-L1': 0,
 'GEO': 1,
 'GTO': 2,
 'HEO': 3,
 'ISS': 4,
 'LEO': 5,
 'MEO': 6,
 'PO': 7,
 'SO': 8,
 'SSO': 9,
 'VLEO': 10}
```

PREDICTIVE ANALYSIS (CLASSIFICATION)

THE WINNER: Decision Tree Classifier

Fit the object to find the best parameters from the dictionary `parameters`.

```
: 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()
```

```
: gscv = GridSearchCV(tree,parameters,scoring='accuracy',cv=10)  
tree_cv = gscv.fit(X_train,Y_train)
```

```
: print("tuned hyperparameters :(best parameters) ",tree_cv.best_params_)  
print("accuracy :",tree_cv.best_score_)
```

```
tuned hyperparameters :(best parameters) {'criterion': 'gini', 'max_de  
accuracy : 0.8892857142857142
```

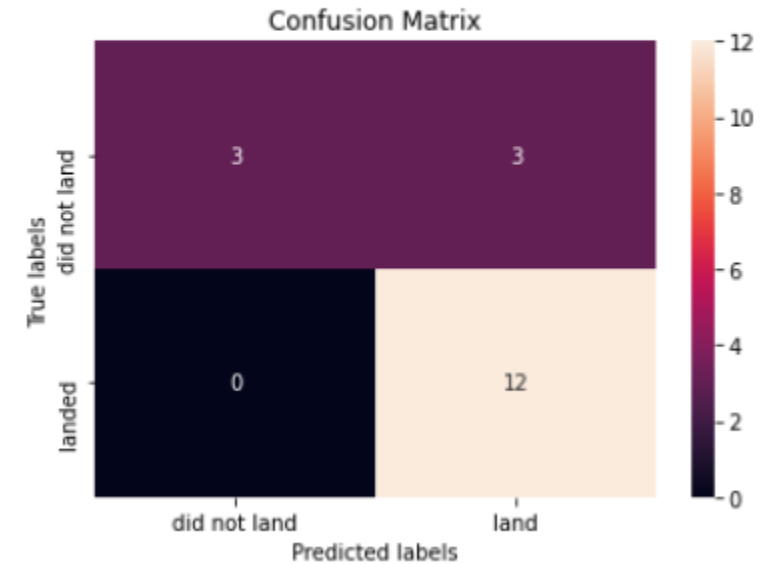
Calculating the accuracy of `tree_cv` on the test data using the method `score`:

```
: print("accuracy: ", tree_cv.score(X_test,Y_test))
```

```
accuracy: 0.7777777777777778
```

We can plot the confusion matrix

```
yhat = svm_cv.predict(X_test)  
plot_confusion_matrix(Y_test,yhat)
```



We see that a major problem is False Positives

		Predicted Values	
		Negative	Positive
Actual Values	Negative	TN	FP
	Positive	FN	TP

CONCLUSIONS

Conclusions

- The success rates for SpaceX launches is directly proportional to time since first launch
- KSC LC-39A had the most successful launches from all the sites
- Orbit GEO, HEO, SSO, ES-L1 had the highest Success Rate
- The low weighted payloads perform better than the heavier payloads
- The Tree Classifier Algorithm is the best for Machine Learning this dataset

APPENDICES



MySQL Server

- The SpaceX.csv data was stored on a locally hosted SQL database since the IBM SQL connection was not working.
- The setup process was as follows:
 - Download MySQL executable and launch the shell terminal to connect to root
 - A database and table with proper labeling was created using a python IDE
 - The csv was loaded using LOAD DATA INFILE command

```
MySQL localhost:33060+ ssl generaldb SQL > LOAD DATA INFILE 'C:\\ProgramData\\MySQL\\MySQL Server 8.0\\Uploads\\Spac
ex.csv' INTO TABLE spacex FIELDS TERMINATED BY ',' ENCLOSED BY '"' LINES TERMINATED BY '\\n' IGNORE 1 ROWS ;
Query OK, 101 rows affected (0.0120 sec)

Records: 101 Deleted: 0 Skipped: 0 Warnings: 0
```

Haversine Formula

- This was used to calculate the distance between two points for the calculations done on the folium map interface
- It uses trigonometry to calculate the distance given the lat and long coordinates
- It is critical for navigation as it considers the spherical nature of the earth and the spherical triangles created when drawing straight lines to two points

```
from math import sin, cos, sqrt, atan2, radians

def calculate_distance(lat1, lon1, lat2, lon2):
    # approximate radius of earth in km
    R = 6373.0

    lat1 = radians(lat1)
    lon1 = radians(lon1)
    lat2 = radians(lat2)
    lon2 = radians(lon2)

    dlon = lon2 - lon1
    dlat = lat2 - lat1

    a = sin(dlat / 2)**2 + cos(lat1) * cos(lat2) * sin(dlon / 2)**2
    c = 2 * atan2(sqrt(a), sqrt(1 - a))

    distance = R * c
    return distance
```


A photograph of a space shuttle launch at night. The shuttle is ascending vertically, leaving a bright, glowing trail of fire and smoke. Large plumes of white smoke and fire are visible at the base of the launch pad. Several tall, slender service towers are positioned around the launch pad. In the foreground, a body of water reflects the bright light from the launch. The sky is dark, providing a stark contrast to the intense light of the rocket.

IBM

Thank you!