SpaceX Data Collection and Presentation

By Joshua Oh

9/14/2023

Table of Contents

Executive Summary

SpaceX is the leading company in the innovation of space traveling vehicles. The result of the procedures in this presentation is to find the method of best success for the SpaceX Falcon 9 first landing stage.

Using dataset provided by SpaceX, we were able to find the success rates of different launch sites, link the success rates with different parameters of each rocket, and use different machine learning algorithms to find the best algorithm in predicting the landing outcome of the rockets.

The results shows us a few things:

- Success is correlated with payload mass, launch site location, orbits, and repeated launches at a site.
- More data for specifics on orbits such as weather would increase the precision of our predictions
- The Decision Tree Algorithm is the best model for predicting landing success.

Introduction Slide

SpaceX 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 SpaceX 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 SpaceX for a rocket launch.

Through this Applied Data Science Capstone, we have learned several methods of obtaining data from the SpaceX API, formatting the data, and visualizing it in multiple different ways.

Methodologies

Methodologies of data collection, analysis, and visualization

Data Collection and Data Wrangling

Collect data from Database from cloud into a Pandas Data frame

```
static_json_url='https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/datasets/API_
```

```
# Use json_normalize meethod to convert the json result into a dataframe
data = pd.json_normalize(response.json())
```

Refine data frame results to only contain needed columns

```
# Lets take a subset of our dataframe keeping only the features we want and the flight number, and date_utc.
data = data[['rocket', 'payloads', 'launchpad', 'cores', 'flight_number', 'date_utc']]

# We will remove rows with multiple cores because those are falcon rockets with 2 extra rocket boosters and rows that h
data = data[data['cores'].map(len)==1]
data = data[data['payloads'].map(len)==1]

# Since payloads and cores are lists of size 1 we will also extract the single value in the list and replace the featur
data['cores'] = data['cores'].map(lambda x : x[0])
data['payloads'] = data['payloads'].map(lambda x : x[0])

# We also want to convert the date_utc to a datetime datatype and then extracting the date leaving the time
data['date'] = pd.to_datetime(data['date_utc']).dt.date

# Using the date we will restrict the dates of the launches
data = data[data['date'] <= datetime.date(2020, 11, 13)]
```

Data Collection and Data Wrangling

Collect data from a website table like Wikipedia: Webscraping

Set up functions to parse each table column

```
def get_mass(table_cells):
    mass=unicodedata.normalize("NFKD", table_cells.text).strip()
    if mass:
        mass.find("kg")
        new_mass=mass[0:mass.find("kg")+2]
    else:
        new_mass=0
    return new_mass
```

Request data tables and convert into Beautiful Soup object

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

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

```
# Use BeautifulSoup() to create a BeautifulSoup obj
soup = BeautifulSoup(response)
```

Set up functions to parse each column header

```
def extract_column_from_header(row):
    """
    This function returns the landing status from the HTML table cell
    Input: the  element of a table data cell extracts extra row
    """
    if (row.br):
        row.br.extract()
    if row.a:
        row.a.extract()
    if row.sup:
        row.sup.extract()

    colunm_name = ' '.join(row.contents)

    # Filter the digit and empty names
    if not(colunm_name.strip().isdigit()):
        colunm_name = colunm_name.strip()
        return colunm_name
```

Data Collection and Data Wrangling

Convert Webscraped Tables into Pandas Data frame

Create Dictionary to use for Pandas Data frame structure

```
launch_dict= dict.fromkeys(column_names)

# Remove an irrelvant 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']=[]
```

Append each part of the row towards its respective column in the dict.

```
launch_site = row[2].a.string
#print(launch_site)
launch_dict['Launch site'].append(launch_site)
```

Iterate through table and filter out unusable or N/A data, and pull each row

```
extracted_row = 0
#Extract each table
for table_number,table in enumerate(soup.find_all('table',"wikitable plainrowheaders collapsible"))
   # get table row
    for rows in table.find_all("tr"):
        #check to see if first table heading is as number corresponding to launch a number
        if rows.th:
            if rows.th.string:
                flight_number=rows.th.string.strip()
                flag=flight_number.isdigit()
        else:
            flag=False
        #get table element
        row=rows.find_all('td')
        #if it is number save cells in a dictonary
        if flag:
            extracted_row += 1
```

Create Pandas Data frame using each dict array as a column

```
df= pd.DataFrame({ key:pd.Series(value) for key, value in launch_dict.items() })
```

Data Collection and Data Wrangling

Data Wrangling Methods

Check for missing data

```
df.isnull().sum()/df.shape[0]*100
```

```
FlightNumber      0.000000
Date              0.000000
BoosterVersion    0.000000
PayloadMass       0.000000
Orbit             0.000000
LaunchSite        0.000000
Outcome           0.000000
Flights           0.000000
GridFins          0.000000
Reused            0.000000
Legs              0.000000
LandingPad       28.888889
```

Check data type

```
df.dtypes
```

```
FlightNumber      int64
Date             object
BoosterVersion   object
PayloadMass     float64
Orbit            object
LaunchSite       object
Outcome          object
```

Count distinct occurrences

```
# Apply value_counts on Orbit column
df['Orbit'].value_counts()
```

```
GTO     27
ISS     21
VLEO    14
PO       9
LEO      7
SSO      5
MEO      3
```

Replace data or fill in missing data

```
# landing_class = 0 if bad_outcome
# landing_class = 1 otherwise
landing_class = df['Outcome'].replace({'False Ocean': 0, 'False ASDS': 0, 'None None': 0, 'None ASDS': 0, 'False RTLS': 0
df.info()
```

	Class
0	0
1	0
2	0
3	0

EDA and Interactive Visual Analytics

There are many different ways to visualize information, and the ways we visualized it for this project was through graphs, charts, maps, and tables.

Types of Graphs and Charts:

Scatter Graphs

Scatter plots show relationship between two variables. This relationship is called the correlation.

- Flight Number vs. Payload Mass
- Flight Number vs. Launch Site
- Payload vs. Launch Site
- Orbit vs. Flight Number
- Payload vs. Orbit Type
- Orbit vs. Payload

Bar Graph or Bar Chart

Bar graphs compare values of numeric and categorical values through differing heights of the graphs.

- Success rate vs. Orbit

Line Graph

Line Graphs are like scatter graphs except they show growth rates or trends of the data.

- Success rate vs. Year

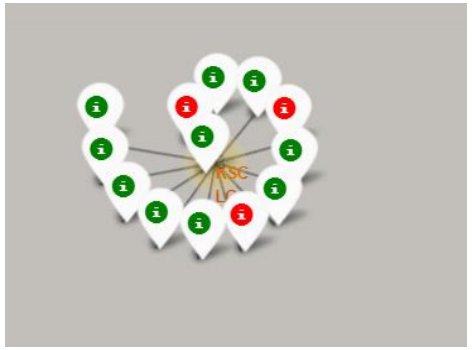
Pie Chart

Pie Charts show numerical proportion

EDA and Interactive Visual Analytics with Visualizations

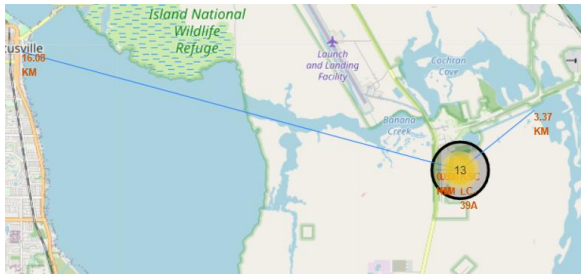
Maps for EDA

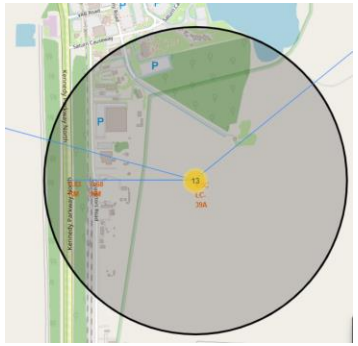
Using folium, we created maps containing markers and other features which allows us to analyze the physical location of the launch sites and nearby areas to understand what makes a successful launch site.



We can see markers representing fails and successes with Red and Green

Through the map and other functions of folium, we were able to see that launch sites must be near areas of transport and large bodies of water, while staying a far distance away from cities or areas of residence.

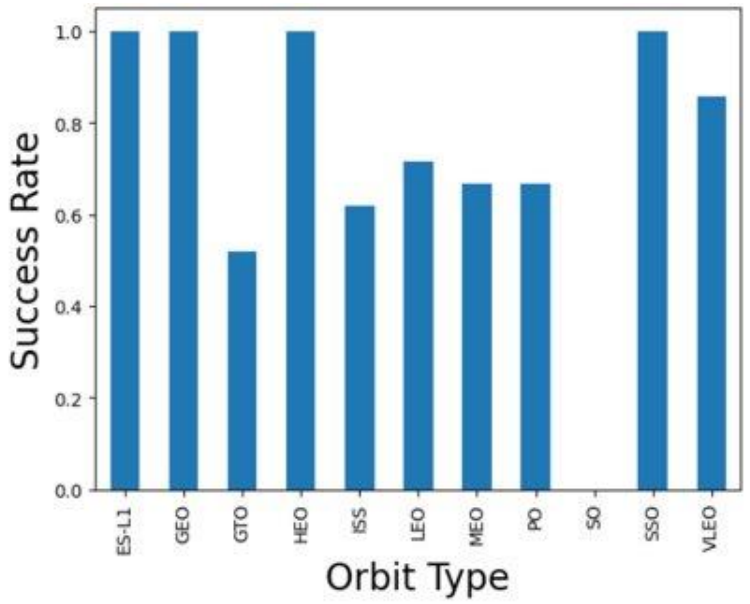




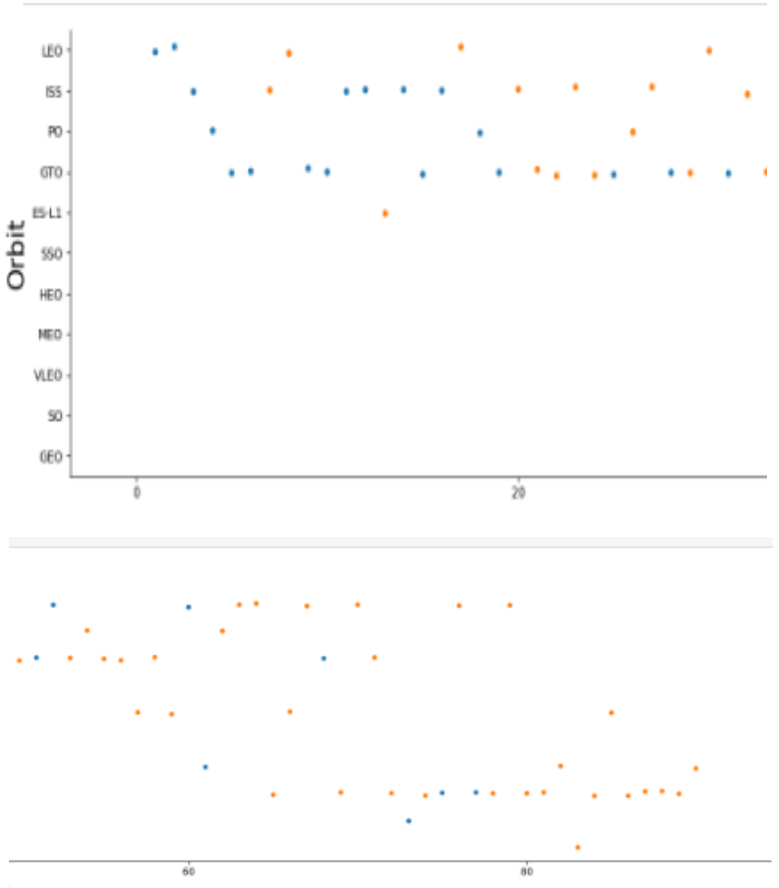
EDA and Interactive Visual Analytics with Visualizations

Using several different graphing methods, we can find different stat comparisons such as Success Rate vs Orbit Type

```
df.groupby("Orbit").mean()['Class'].plot(kind='bar')
plt.xlabel("Orbit Type", fontsize=20)
plt.ylabel("Success Rate", fontsize=20)
plt.show()
```

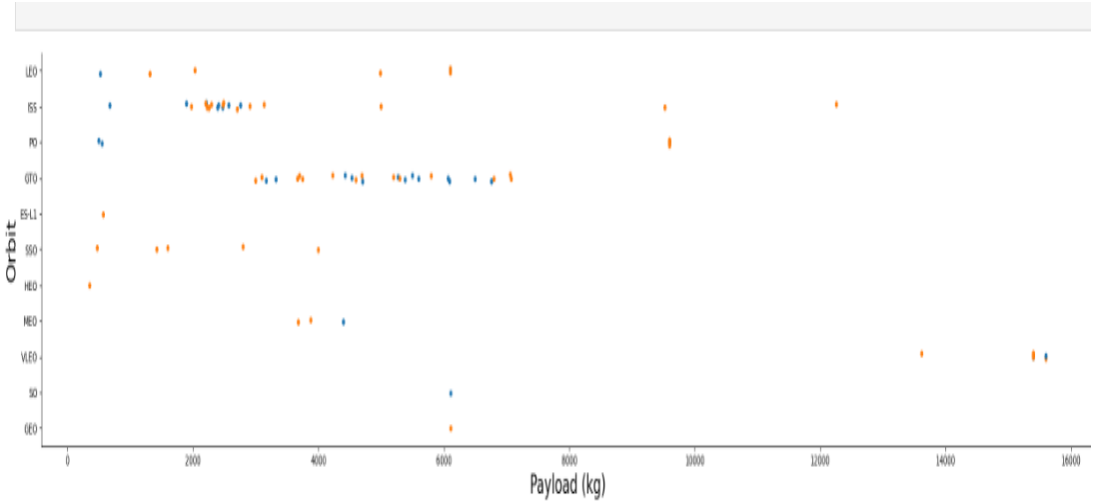


We can also see that different parameters have different correlations for every launch site



EDA and Interactive Visual Analytics with Visualizations

On the graph of Payload Mass vs Orbit with successes and failures, we can see that a payload above 10000 kg is not used and it shows that large payloads are more difficult to make successful. Lower payloads are likelier to be used.



EDA and Interactive Visual Analytics with Visualizations

SQL can also be used to access and manipulate Pandas Data frames

Load SQLite for notebooks

```
%load_ext sql
```

```
import csv, sqlite3

con = sqlite3.connect("my_data1.db")
cur = con.cursor()
```

```
!pip install -q pandas==1.1.5
```

```
%sql sqlite:///my_data1.db
```

Bring the loaded data frame into SQL

```
import pandas as pd
df = pd.read_csv("https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/labs/module
df.to_sql("SPACEXTBL", con, if_exists='replace', index=False,method="multi")
```

```
%sql create table SPACEXTABLE as select * from SPACEXTBL where Date is not null
```

SQL for creating custom table example

```
%sql select LAUNCH_SITE from SPACEXTBL where LAUNCH_SITE like "CCA%" limit 5
```

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

Launch_Site
CCAFS LC-40
CCAFS LC-40
CCAFS LC-40
CCAFS LC-40
CCAFS LC-40

Calculations for stats using SQL example

```
%sql select avg(PAYLOAD_MASS__KG_) as AvgPayloadMass
```

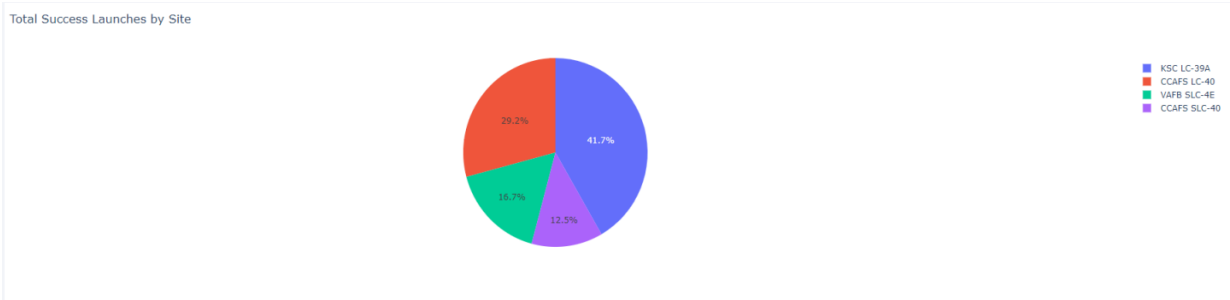
```
 * sqlite:///my_data1.db
Done.
```

AvgPayloadMass
2928.4

EDA and Interactive Visual Analytics with Visualizations

Plotly Dash Dashboard

Using Plotly's Dash, we created a dashboard that could utilize plotly's graphs and display them on an HTML webpage. This allowed us to utilize python for information organization and dash for data visualization



Predictive Analysis

Predicting future outcomes, such as the success of the F-9 Rocket landing, based on gathered data is important to show likelihood of success for investors to place their money in.

To start predictive analysis, we create test variables to train our models with.

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=2)
```

Using this test data, we create different predictive models and sample the accuracy of the model. Here is one for the logistic regression model.

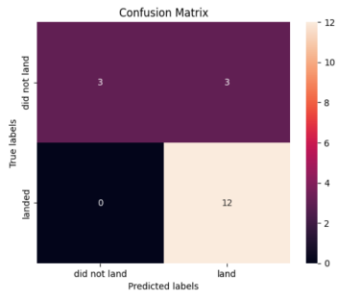
```
parameters ={"C":[0.01,0.1,1],'penalty':['l2'], 'solver':['lbfgs']}#
lr=LogisticRegression()

gsvc= GridSearchCV(lr,parameters, scoring = 'accuracy',cv=10)
logreg_cv = gsvc.fit(X_train, Y_train)
```

Then we check the accuracy score or the model and plot a confusion matrix to

```
logreg_cv.score(X_test, Y_test)
```

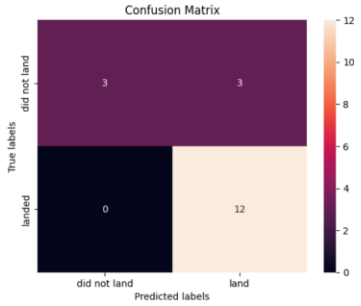
```
0.8333333333333334
```

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

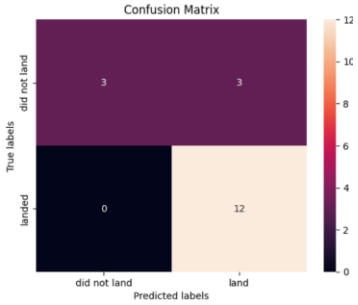


Predictive Analysis

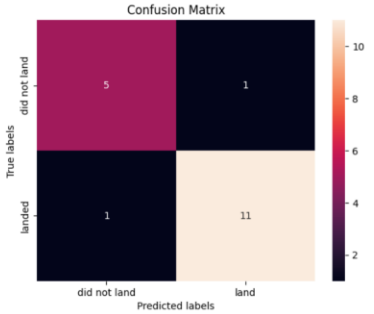
Logistic Regression



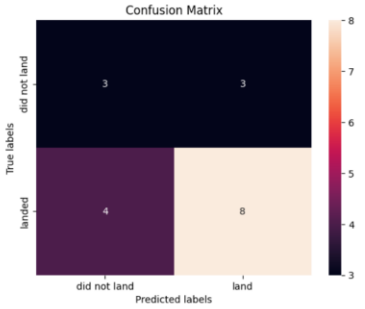
SVM



Decision Tree



KNN



Here we can see the results of the different kinds of predictive algorithms. Out of all of them, the Decision Tree gave the most accurate model, with an accuracy of about 89%.

This accuracy can be seen in the Confusion Matrices as well.

The problem with the SVM and Logistic Regression models was that although there were many correct positives, there were also too many false positives. However, the KNN model had too many false negatives.

The Decision Tree model had the most correctly placed samples and was the most accurate of them all.

Reading the Confusion Matrix

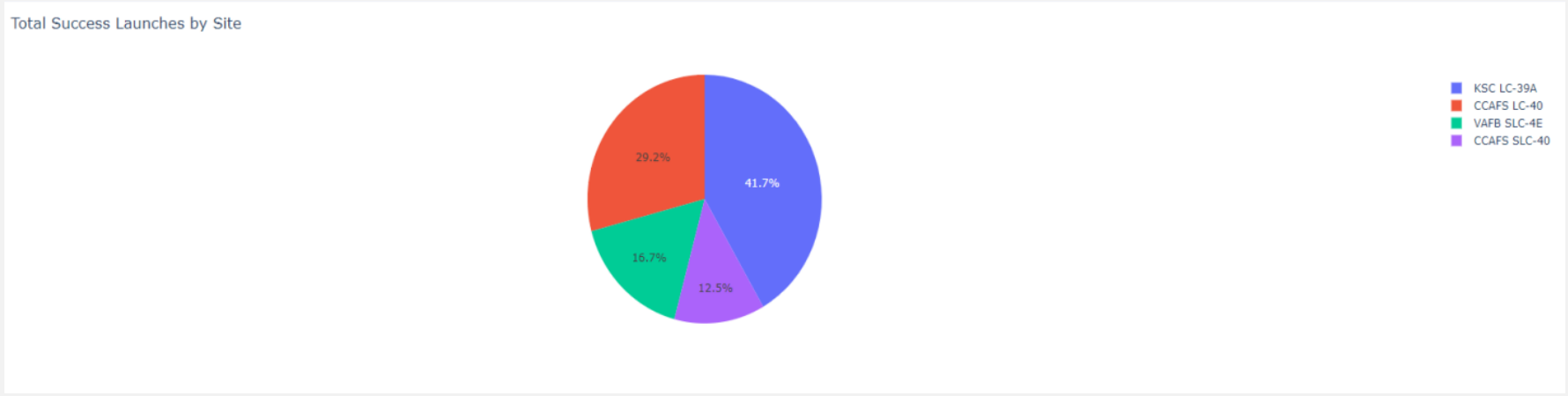
Actual Result	Negative	Positive
Negative	True Negative	False Positive
Positive	False Negative	True Positive

Prediction

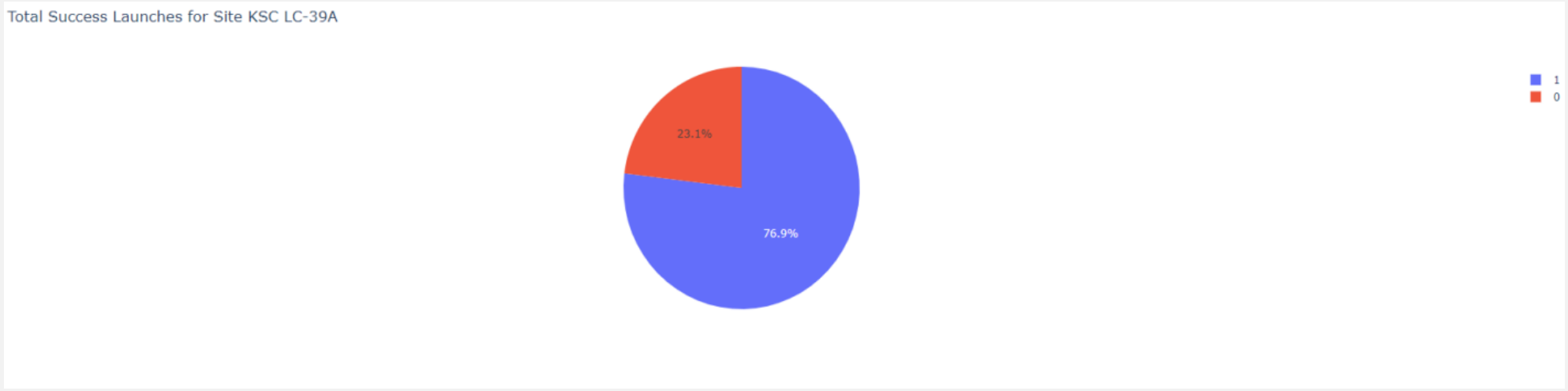
Results

Results of Methodologies

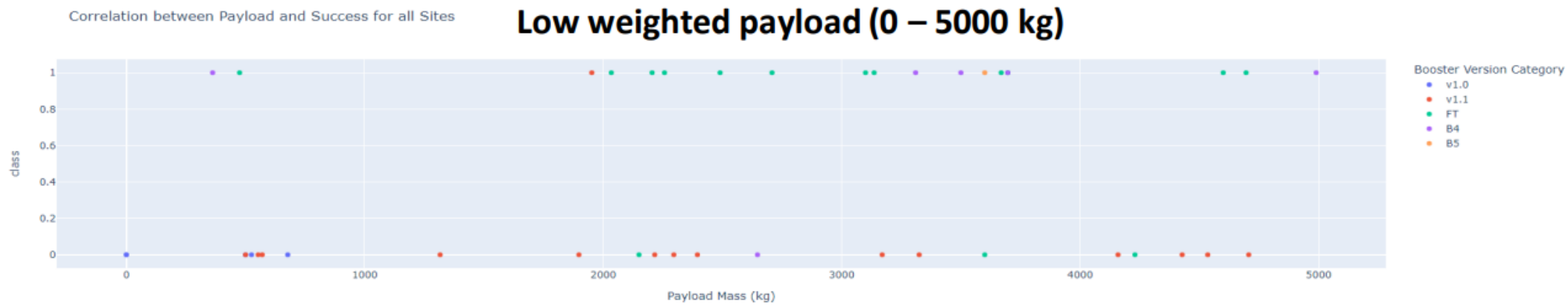
Total Success By Site: KSC LC-39A has the highest amount of successes

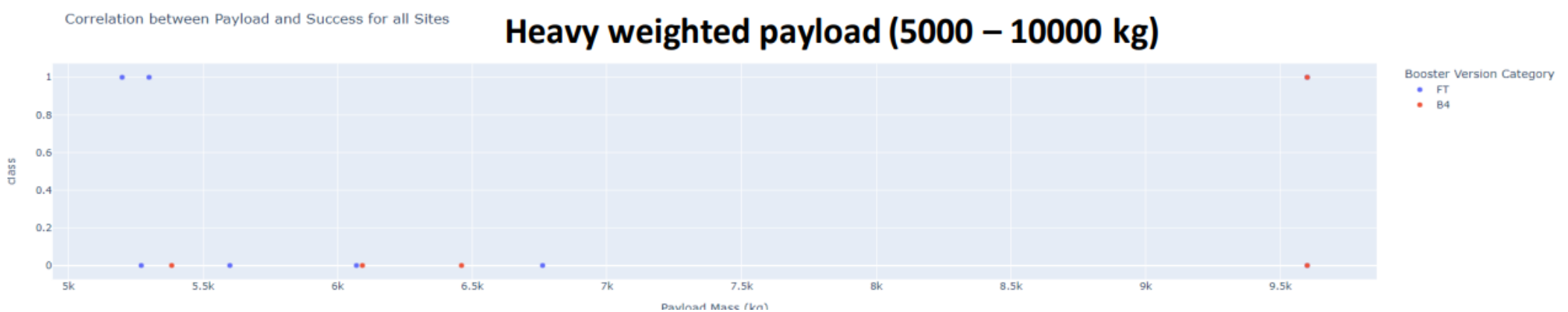


KSC LC-39A Has a 76.9% Success Rate



Payload vs Success Rate for Low and High Payload Mass





Low weighted Payloads have a higher success rate.

Conclusion & Insights

- The success of a mission can be explained by several factors such as the launch site, the orbit and especially the number of previous launches. We can safely assume that repetition increases knowledge and perfection of the launch's success.
- The orbits with the best success rates are GEO, HEO, SSO, ES-L1. Lower payload masses had higher success rates.
- The Launch sites with the best success rates is KSC LC-39A .
- There is still more data to gain as we cannot accurately pinpoint the reason why KSC LC-39A is the most successful site. Possible data such as weather conditions and rocket mechanical failures could give error to the accuracy our data or analysis.
- The Decision Tree prediction model was the most accurate of the 4 models tested. 16/18 samples were predicted correctly, giving us an accuracy of 89%. Since only 1 false positive was found, investors could rely on this to decide see whether or not the mission will succeed with a 94.4% accuracy.


Thank you