One Step Ahead: Using Data Science to Optimize the Falcon 9

William Jones

Jan 7th 2022

IBM Data Science Capstone Project

Outline

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

Executive Summary

- Methodologies
  - Data Collection
  - Data Wrangling
  - Exploratory Data Analysis with SQL
  - Exploratory Data Analysis with Visualization
  - Building Interactive Map with Folium
  - Buidling a Dashboard with Plotly Dash
  - Predictive Analysis
- Results
  - Exploratory Data Analysis Results
  - Interactive Analysis Results
  - Predictive Analysis Results

Introduction

Problems we face

- What gives the best chance for SpaceX to land a rocket successfully?
- What factors affect the SpaceX rocket from landing?

Today is the age of commercially used rockets has come and with that comes competition to find the most affordable means of space travel. One of the most affordable rockets today is the SpaceX Falcon 9.

On SpaceX's website it is stated that the Falcon 9 rocket cost $62 million dollars while other competitors rockets cost up to $165 million dollars. SpaceX reduces the cost of the rocket not by cutting corners, but by being able to reuse the first stage of the rocket with it landing itself after takeoff.

In the Presentation we will predict if the Falcon 9 rocket will land successfully. Therefore, if we can determine if the first stage will land, we can find the cost of launch. This information can be used by another company to bid against SpaceX.

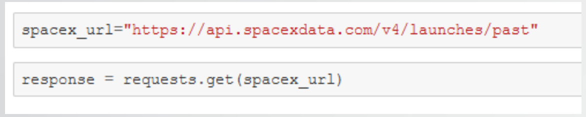
Methodology

Data Collection

- SpaceX Rest API
  - Provided launch data which included rocket type, payload, launch details, and landing details.
- Web Scrapping
  - Scrapped Wikipedia about site that contained launch data on Falcon 9

Data Collection of SpaceX REST API

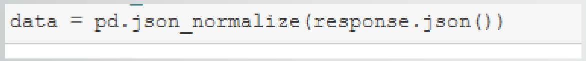
1. Obtaining launch data from SpaceX REST API



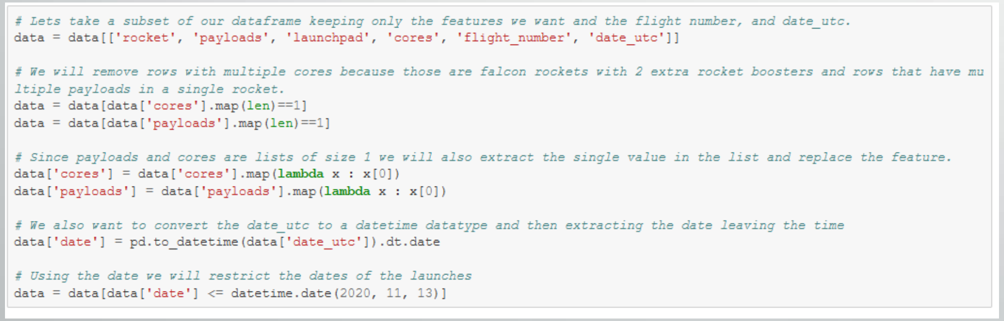
```
spacex_url="https://api.spacexdata.com/v4/launches/past"
```

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

2. Converted the data into json file then into a Pandas data frame

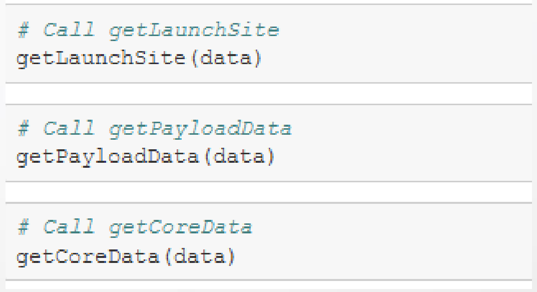


```
data = pd.json_normalize(response.json())
```



```
# 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 have mu
ltiple payloads in a single rocket.
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 feature.
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)]
```

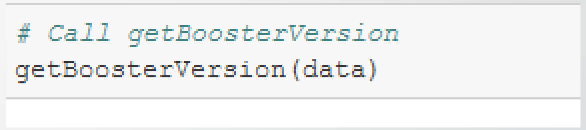
3. Cleaned the data using functions.



```
# Call getLaunchSite
getLaunchSite(data)
```

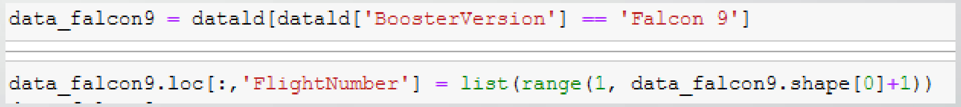
```
# Call getPayloadData
getPayloadData(data)
```

```
# Call getCoreData
getCoreData(data)
```



```
# Call getBoosterVersion
getBoosterVersion(data)
```

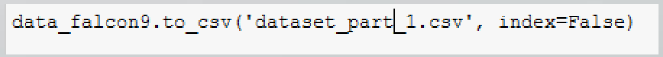
4. Filtered the data to contain just Falcon 9 rocket



```
data_falcon9 = datald[datald['BoosterVersion'] == 'Falcon 9']
```

```
data_falcon9.loc[:,'FlightNumber'] = list(range(1, data_falcon9.shape[0]+1))
```

5. Exported the data to a csv file



```
data_falcon9.to_csv('dataset_part_1.csv', index=False)
```

Full Code: Github

Data Collection With Web Scrapping

1. Requested html from URL

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

2. Used BeautifulSoup to create object from response

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

3. Found tables from object

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

4. Extracted the column names from the table header

```
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 row in first_launch_table.find_all('th'):
    col = extract_column_from_header(row)
    if(col!= None and len(col)>0):
        column_names.append(col)
```

5. Created dictionary and turning data into keys

```
launch_dict= dict.fromkeys(column_names)
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
            # Flight Number value
            launch_dict['Flight No.'].append(flight_number)
            print(flight_number)
            datatimelist=date_time(row[0])
            # Date value
            date = datatimelist[0].strip(',')
            print(date)
            launch_dict['Date'].append(date)
            # Time value
```

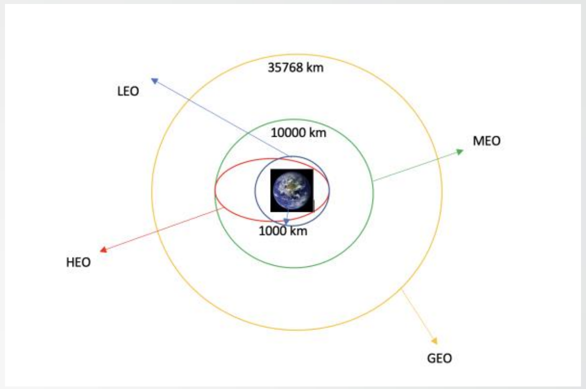
6. Turned dictionary into data frame then into csv file

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

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

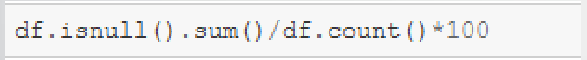
Full Code: Github

Data Wrangling



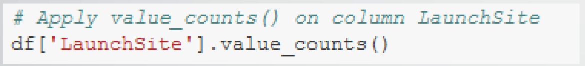
Here we clean the data and make it easy for data analysis. We change the Outcomes column data into 1 (successful landing) and 0 (unsuccessful landing) for the training labels

1. Loaded data and checked for missing values



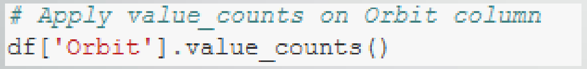
```
df.isnull().sum()/df.count()*100
```

2.Calculated the number of launches each site



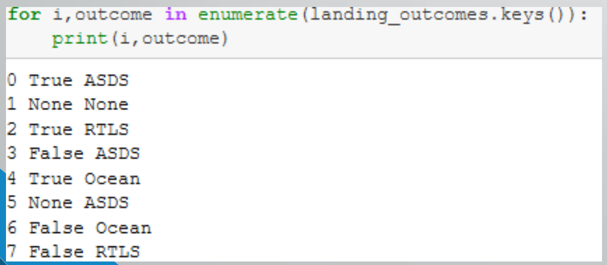
```
# Apply value_counts() on column LaunchSite
df['LaunchSite'].value_counts()
```

3.Calculated the number and occurrence of each orbit



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

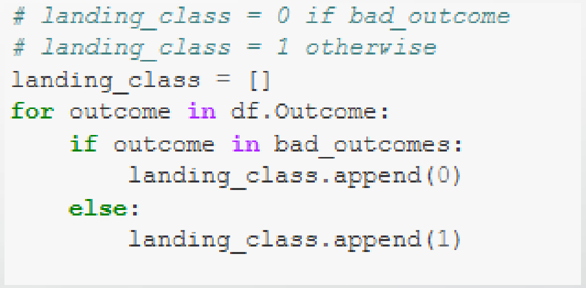
4.Calculated the number and occurrence of mission outcome per orbit type



```
for i,outcome in enumerate(landing_outcomes.keys()):
    print(i,outcome)
```

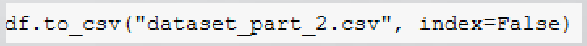
```
0 True ASDS
1 None None
2 True RTLS
3 False ASDS
4 True Ocean
5 None ASDS
6 False Ocean
7 False RTLS
```

5. Created a landing outcome label from Outcome column



```
# landing_class = 0 if bad_outcome
# landing_class = 1 otherwise
landing_class = []
for outcome in df.Outcome:
    if outcome in bad_outcomes:
        landing_class.append(0)
    else:
        landing_class.append(1)
```

6.Exported the data to a csv file



```
df.to_csv("dataset_part_2.csv", index=False)
```

Full Code: Github

Exploratory Data Analysis with Data Visualization

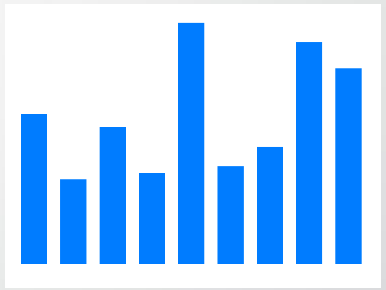
Using graphs, we can show how different variables correlate to each other (scatter plot), relationship of different values (bar chart), and trends of the data (line charts)

Scatter plots Graphed:

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

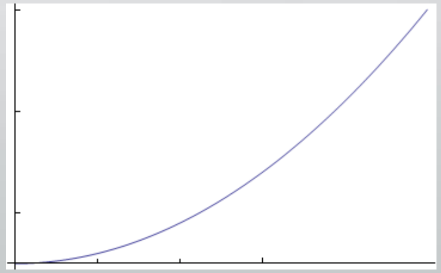
Bar chart Graphed:

- Success Rate vs. Orbit Type



Line Plot Graphed:

- Year vs. Success Rate



We will view the results in the Results section

Full Code: [Github]()

Exploratory Data Analysis with SQL

We used SQL queries to get analyze and information about the dataset. We used the IBM DB2 cloud base to store our dataset.

SQL queries used to get information:

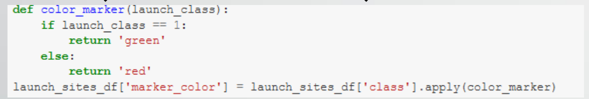
- Display the names of the unique launch sites in the space mission
- Display 5 records where launch sites begin with the string 'CCA'
- Display the total payload mass carried by boosters launched by NASA (CRS
- Display average payload mass carried by booster version F9 v1.1
- List the date when the first successful landing outcome in ground pad was achieved.
- List the names of the boosters which have success in drone ship and have payload mass greater than 4000 but less than 6000
- List the total number of successful and failure mission outcomes
- List the names of the booster_versions which have carried the maximum payload mass. Use a subquery
- List the failed landing_outcomes in drone ship, their booster versions, and launch site names for in year 2015
- 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

Full Code: Github

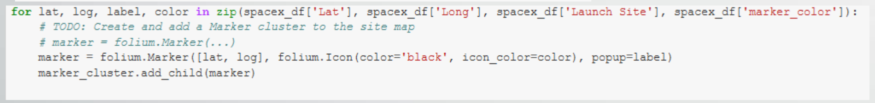
Building An Interactive Map with Folium

With Folium we can visualize data to an interactive geospatial map . Using the longitude and latitude of the launch site coordinates we can map locations of each launch. Here are some other ways we used Folium.

- Marked each launch as green (success) and red(unsuccessful)

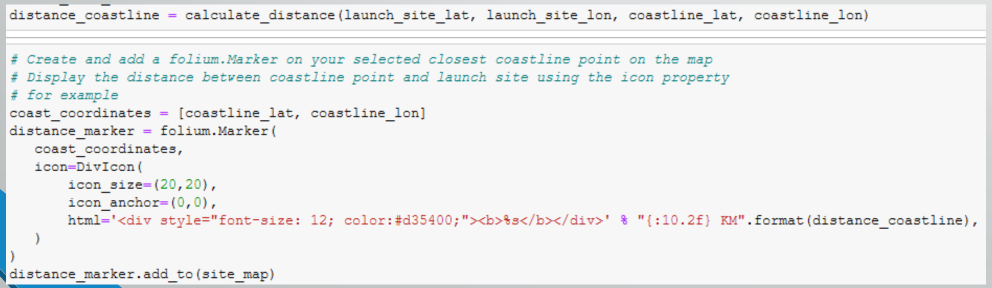


```python
def color_marker(launch_class):
    if launch_class == 1:
        return 'green'
    else:
        return 'red'
launch_sites_df['marker_color'] = launch_sites_df['class'].apply(color_marker)
```



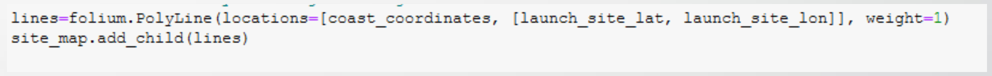
```python
for lat, log, label, color in zip(spacex_df['Lat'], spacex_df['Long'], spacex_df['Launch Site'], spacex_df['marker_color']):
    # TODO: Create and add a Marker cluster to the site map
    # marker = folium.Marker(...)
    marker = folium.Marker([lat, log], folium.Icon(color='black', icon_color=color), popup=label)
    marker_cluster.add_child(marker)
```

- Found how far launch was from coastline



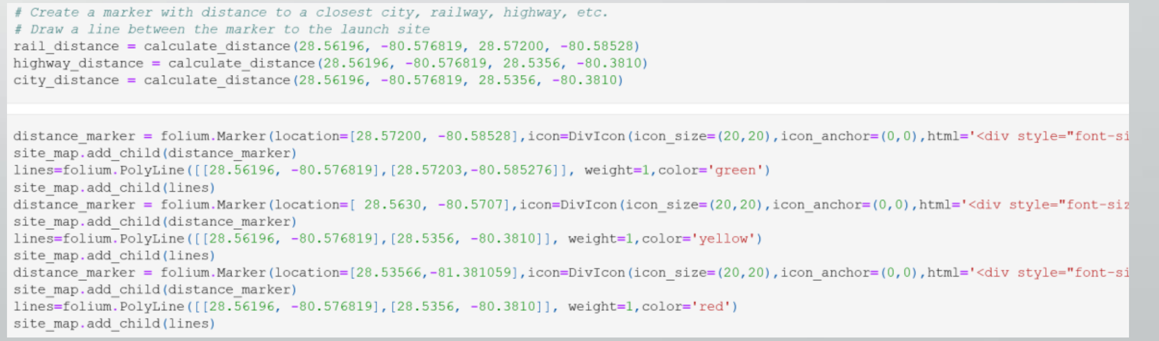
```python
distance_coastline = calculate_distance(launch_site_lat, launch_site_lon, coastline_lat, coastline_lon)

# Create and add a folium.Marker on your selected closest coastline point on the map
# Display the distance between coastline point and launch site using the icon property
# for example
coast_coordinates = [coastline_lat, coastline_lon]
distance_marker = folium.Marker(
   coast_coordinates,
   icon=DivIcon(
       icon_size=(20,20),
       icon_anchor=(0,0),
       html='<div style="font-size: 12; color:#d35400;"><b>%s</b></div>' % "{:10.2f} KM".format(distance_coastline),
   )
)
distance_marker.add_to(site_map)
```

- Drew a line from launch to coastline



```python
lines=folium.PolyLine(locations=[coast_coordinates, [launch_site_lat, launch_site_lon]], weight=1)
site_map.add_child(lines)
```

- Calculated its distance to its proximities



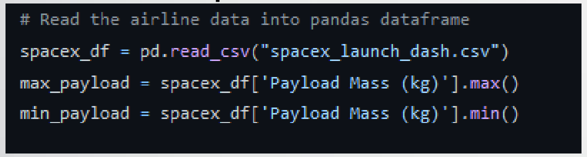
```python
# Create a marker with distance to a closest city, railway, highway, etc.
# Draw a line between the marker to the launch site
rail_distance = calculate_distance(28.56196, -80.576819, 28.57200, -80.58528)
highway_distance = calculate_distance(28.56196, -80.576819, 28.5356, -80.3810)
city_distance = calculate_distance(28.56196, -80.576819, 28.5356, -80.3810)

distance_marker = folium.Marker(location=[28.57200, -80.58528],icon=DivIcon(icon_size=(20,20),icon_anchor=(0,0),html='<div style="font-si
site_map.add_child(distance_marker)
lines=folium.PolyLine([[28.56196, -80.576819],[28.57203,-80.585276]], weight=1,color='green')
site_map.add_child(lines)
distance_marker = folium.Marker(location=[ 28.5630, -80.5707],icon=DivIcon(icon_size=(20,20),icon_anchor=(0,0),html='<div style="font-siz
site_map.add_child(distance_marker)
lines=folium.PolyLine([[28.56196, -80.576819],[28.5356, -80.3810]], weight=1,color='yellow')
site_map.add_child(lines)
distance_marker = folium.Marker(location=[28.53566,-81.381059],icon=DivIcon(icon_size=(20,20),icon_anchor=(0,0),html='<div style="font-si
site_map.add_child(distance_marker)
lines=folium.PolyLine([[28.56196, -80.576819],[28.5356, -80.3810]], weight=1,color='red')
site_map.add_child(lines)
```

Full Code: Github

Building a Dashboard with Plotly Dash

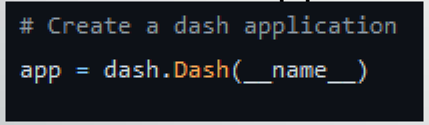
Plotly Dash builds interactive apps without the need of front-end web development. Here we will use it to create a dashboard application that features a dropdown list and range slider to interact with a pie chart and scatter point chart of the SpaceX data.

- 1. Load in SpaceX data into a data frame



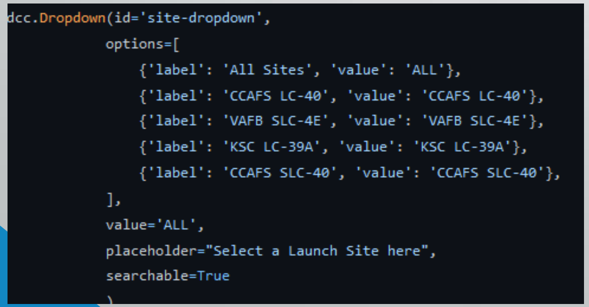
```
# Read the airline data into pandas dataframe
spacex_df = pd.read_csv("spacex_launch_dash.csv")
max_payload = spacex_df['Payload Mass (kg)'].max()
min_payload = spacex_df['Payload Mass (kg)'].min()
```

- 2. Create the application



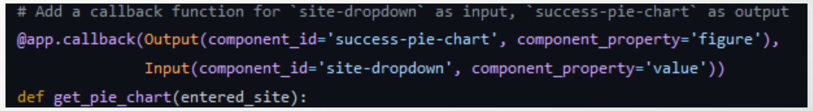
```
# Create a dash application
app = dash.Dash(__name__)
```

- 3. Add a dropdown list to choose between different launch sites



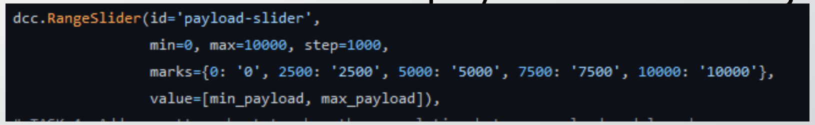
```
dcc.Dropdown(id='site-dropdown',
             options=[
                 {'label': 'All Sites', 'value': 'ALL'},
                 {'label': 'CCAFS LC-40', 'value': 'CCAFS LC-40'},
                 {'label': 'VAFB SLC-4E', 'value': 'VAFB SLC-4E'},
                 {'label': 'KSC LC-39A', 'value': 'KSC LC-39A'},
                 {'label': 'CCAFS SLC-40', 'value': 'CCAFS SLC-40'},
             ],
             value='ALL',
             placeholder="Select a Launch Site here",
             searchable=True
             )
```

- Add a pie chart that creates a success/failures chart of selected launch site



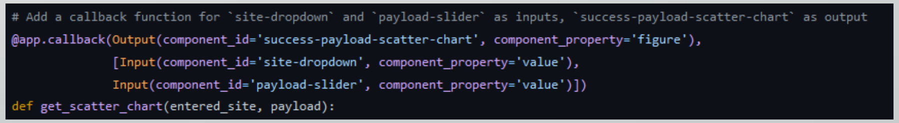
```
# Add a callback function for `site-dropdown` as input, `success-pie-chart` as output
@app.callback(Output(component_id='success-pie-chart', component_property='figure'),
              Input(component_id='site-dropdown', component_property='value'))
def get_pie_chart(entered_site):
```

- Add a slider to select payload mass range



```
dcc.RangeSlider(id='payload-slider',
                min=0, max=10000, step=1000,
                marks={0: '0', 2500: '2500', 5000: '5000', 7500: '7500', 10000: '10000'},
                value=[min_payload, max_payload]),
```

- Create scatter plot for payload mass vs. launch success



```
# Add a callback function for `site-dropdown` and `payload-slider` as inputs, `success-payload-scatter-chart` as output
@app.callback(Output(component_id='success-payload-scatter-chart', component_property='figure'),
              [Input(component_id='site-dropdown', component_property='value'),
               Input(component_id='payload-slider', component_property='value')])
def get_scatter_chart(entered_site, payload):
```

Full code: Github

Predictive Analysis (Classification)

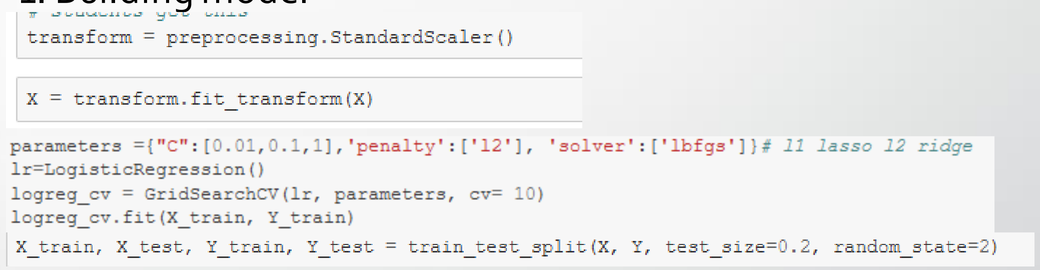
We use the data collected and cleaned from the previous steps to build our model to predict the success of a Falcon 9 rocket landing. Classification models are the best fit here because the outcome we want is discrete values success or unsuccessful. We will determine which classification algorithm will work the best. Algorithms we will be comparing are support vector machine, logistic regression, decision tree, and k nearest neighbors

Machine learning process:

- Build the model
  - Standardize and transform data
  - Split data into training and testing sets
  - Fit into GridSearchCV objects
- Evaluate the model
  - Check accuracy
  - Get hyperparameter
  - Plot confusion matrix
- Find best performing model
  - Comparing accuracy scores

Looking at logistic regression algorithm:

- 1. Building model



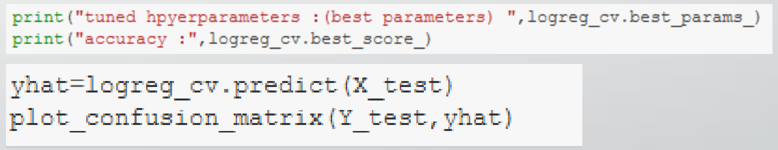
```
transform = preprocessing.StandardScaler()
```

```
X = transform.fit_transform(X)
```

```
parameters ={"C":[0.01,0.1,1],'penalty':['l2'], 'solver':['lbfgs']}# l1 lasso l2 ridge
lr=LogisticRegression()
logreg_cv = GridSearchCV(lr, parameters, cv= 10)
logreg_cv.fit(X_train, Y_train)
```

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

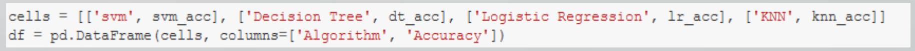
- Evaluating model



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

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

- Finding best performing model



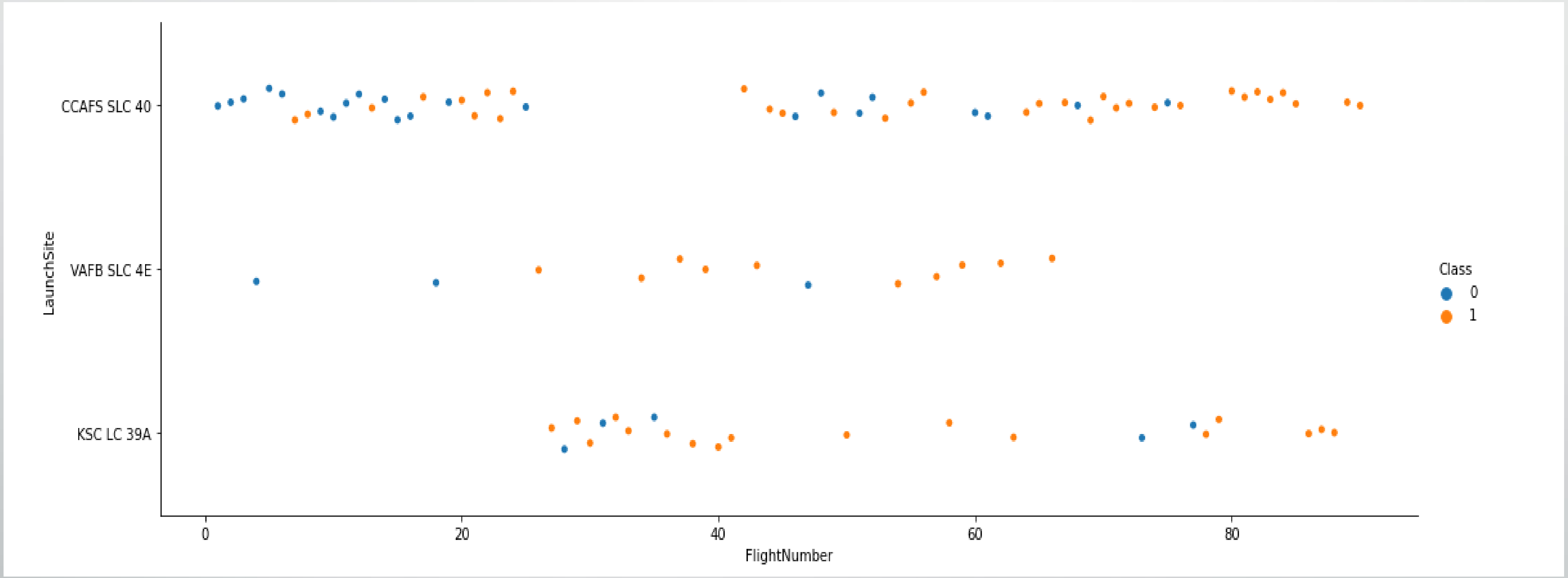
```
cells = [['svm', svm_acc], ['Decision Tree', dt_acc], ['Logistic Regression', lr_acc], ['KNN', knn_acc]]
df = pd.DataFrame(cells, columns=['Algorithm', 'Accuracy'])
```

Full Code : Github

Results

Exploratory Data Analysis Results

Flight Number vs. Launch Site

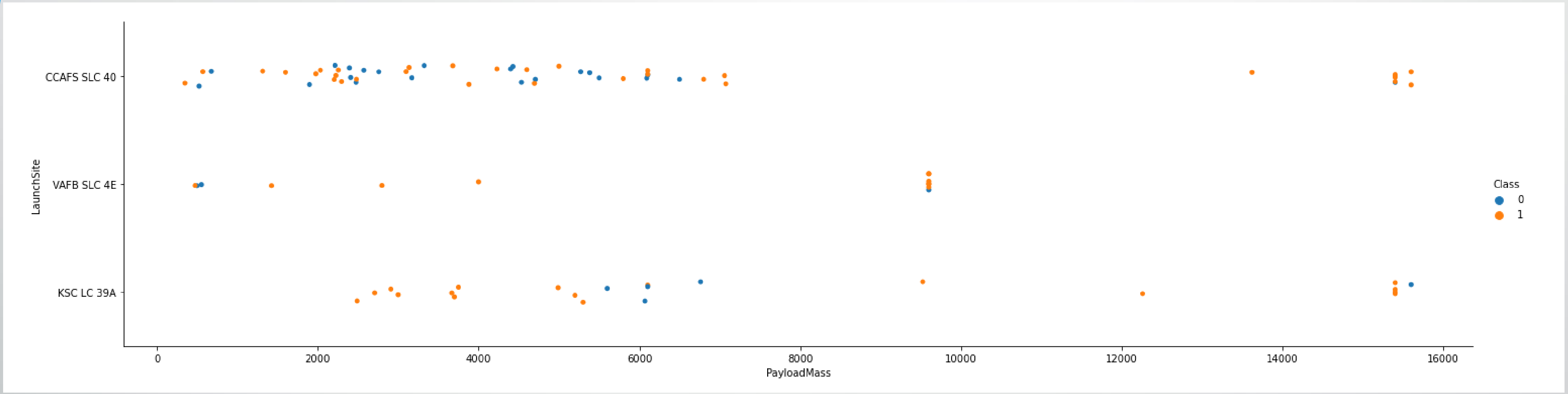


Here we can see that as the flight number becomes higher the greater success rate is.

Successful = 1
Unsuccessful = 0

Exploratory Data Analysis Results

Payload Mass vs. Launch Site

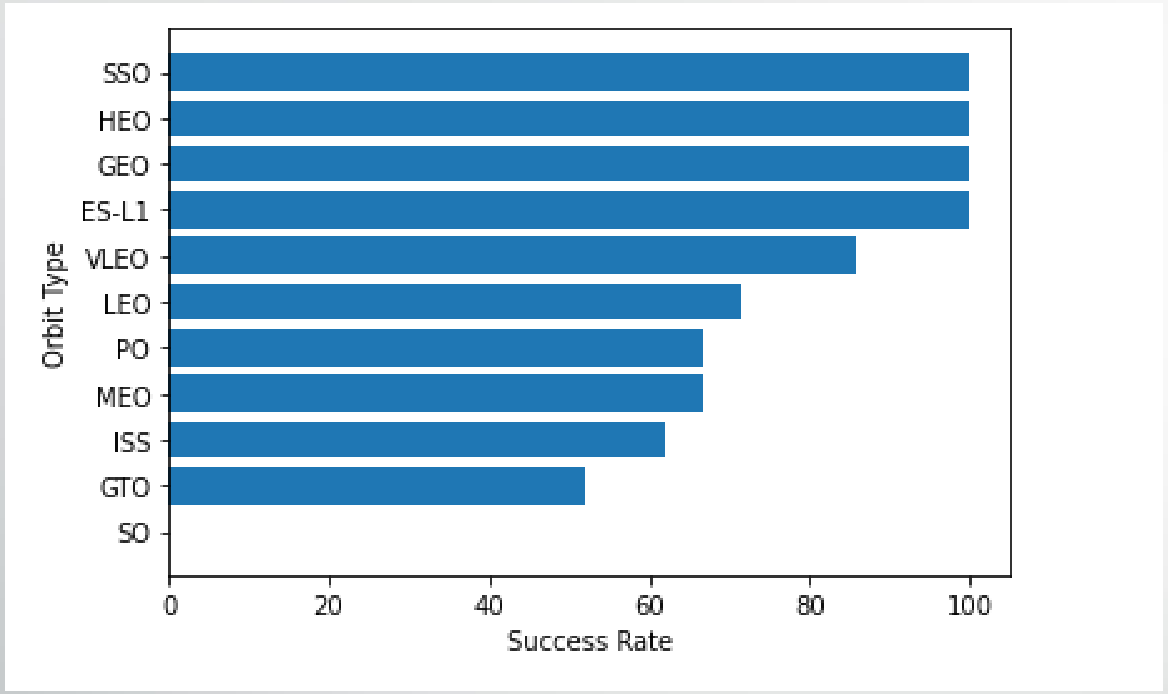


Large amount of payload masses are under 7000 lbs but as the mass increases past 7000 lbs the success rate is far greater

Successful = 1
Unsuccessful = 0

Exploratory Data Analysis Results

Success Rate vs. Orbit Type



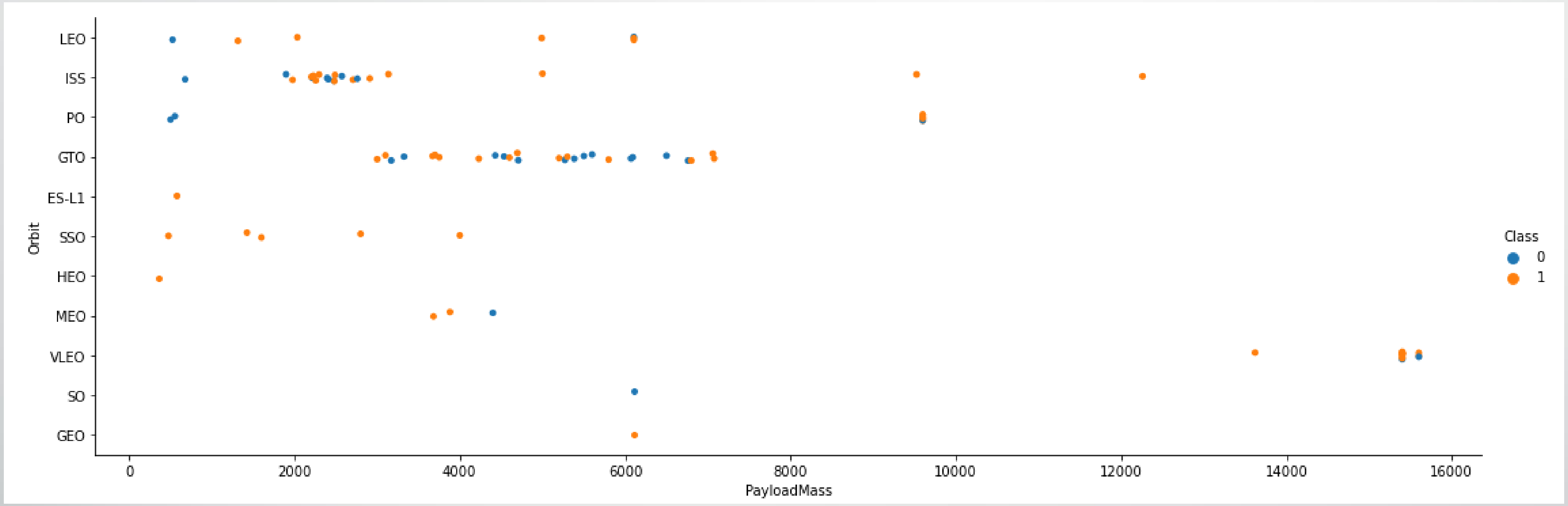
Orbit Types:

- SO / SSO – Sun synchronous orbit
- GTO – Geosynchronous orbit
- ISS – Low earth orbit to space station
- MEO – Intermediate circular orbit
- PO – Satellite that orbits above both poles
- LEO – Low Earth orbit
- VLEO – Very low Earth orbit
- ES-L1 – Sun-Earth Lagrange point orbit
- GEO – Circular geosynchronous orbit
- HEO – Geocentric orbit

SSO, HEO, GEO, and ES-L1 have the highest success rate while GTO has the lowest success rate.

Exploratory Data Analysis Results

Payload Mass vs. Orbit Type

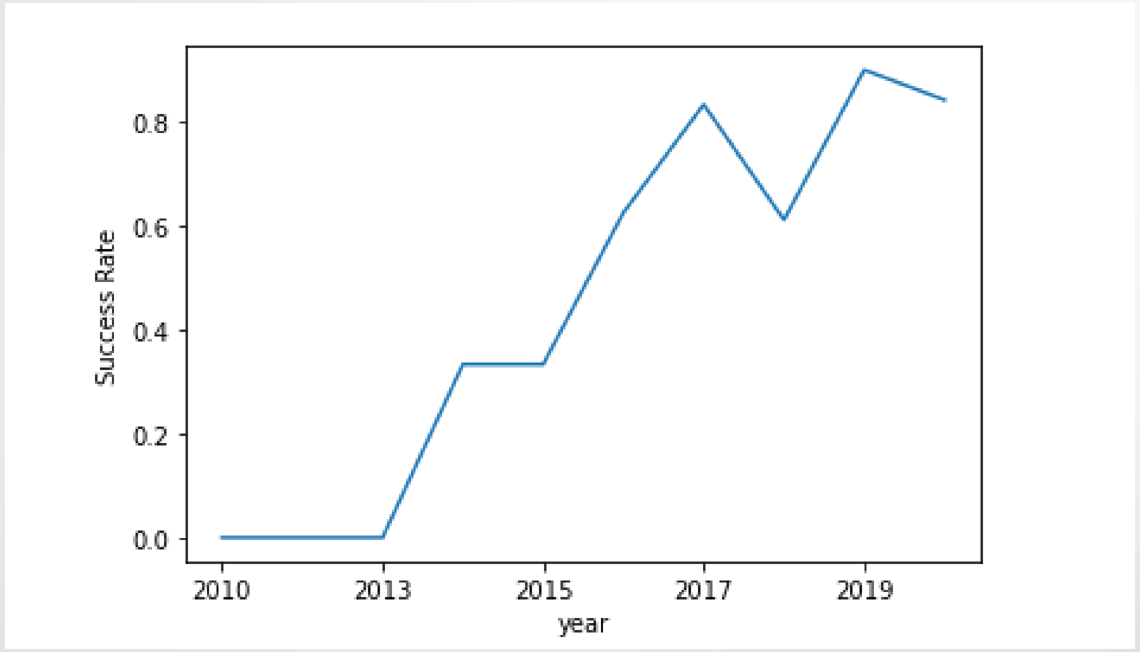


We can see that the most successful orbit types had light payloads. Other orbit types had more success at heavier payloads. There is no correlation for the GTO orbit.

Success = 1
Unsuccessful = 0

Exploratory Data Analysis Results

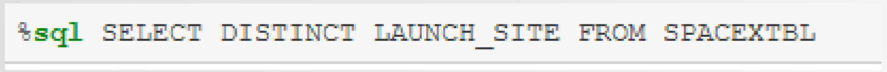
Year vs. Success Rate



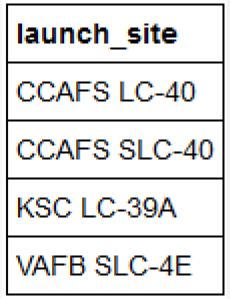
As we can see the first two initial years SpaceX had no successes. However, as the years increase the success rate increases.

Exploratory Data Analysis Results

Here we look for different launch sites using DISTINCT

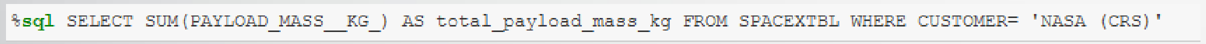


```
%sql SELECT DISTINCT LAUNCH_SITE FROM SPACEXTBL
```

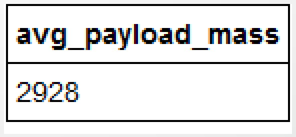


launch_site
CCAFS LC-40
CCAFS SLC-40
KSC LC-39A
VAFB SLC-4E

Here we find the total payload mass using WHERE filters to find customer that is NASA (CRS)

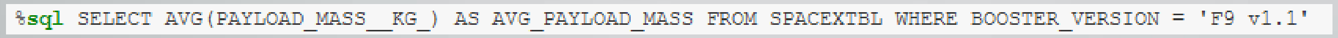


```
%sql SELECT SUM(PAYLOAD_MASS__KG_) AS total_payload_mass_kg FROM SPACEXTBL WHERE CUSTOMER= 'NASA (CRS)'
```

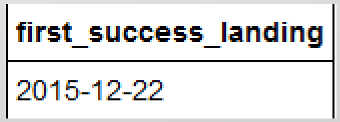


avg_payload_mass
2928

Here we find the average payload mass using AVG() on payload mass, and using WHERE to find only Falcon 9 rockets



```
%sql SELECT AVG(PAYLOAD_MASS__KG_) AS AVG_PAYLOAD_MASS FROM SPACEXTBL WHERE BOOSTER_VERSION = 'F9 v1.1'
```



first_success_landing
2015-12-22

Exploratory Data Analysis Results

Here we find the first successful landing using MIN() on to get oldest data WHERE the outcome is ground pad

```
%sql SELECT MIN(DATE) AS FIRST_SUCCESS_LANDING FROM SPACEXTBL WHERE LANDING__OUTCOME = 'Success (ground pad)'
```

first_success_landing
2015-12-22

Getting the Booster Versions of successful drone ship landing using WHERE and the payload mass being 4000> x > 6000 using Boolean operators

```
%sql SELECT BOOSTER_VERSION FROM SPACEXTBL WHERE LANDING__OUTCOME = 'Success (drone ship)' AND (PAYLOAD_MASS__KG_ BETWEEN 400
0 AND 6000)
```

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

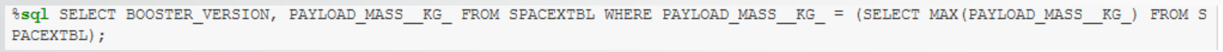
Here we get the number of successful mission outcomes using COUNT(*) and GROUP BY to group them by outcomes

```
%sql SELECT MISSION_OUTCOME, COUNT(*) AS TOTAL FROM SPACEXTBL GROUP BY MISSION_OUTCOME
```

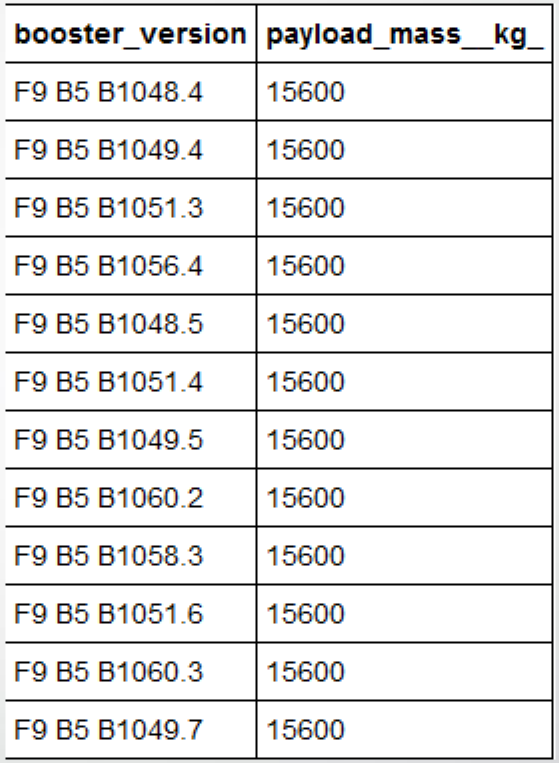
mission_outcome	total
Failure (in flight)	1
Success	99
Success (payload status unclear)	1

Exploratory Data Analysis Results

Here we list the boosters who carried the max payload mass. Using WHERE to subquery payload mass and MAX() to get the max mass

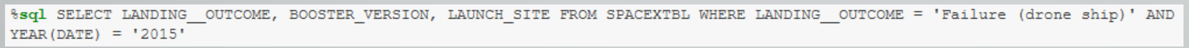


```
%sql SELECT BOOSTER_VERSION, PAYLOAD_MASS__KG_ FROM SPACEXTBL WHERE PAYLOAD_MASS__KG_ = (SELECT MAX(PAYLOAD_MASS__KG_) FROM S
PACEXTBL);
```

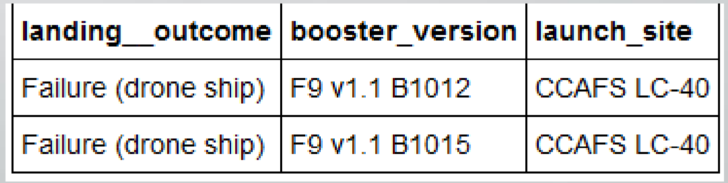


booster_version	payload_mass__kg_
F9 B5 B1048.4	15600
F9 B5 B1049.4	15600
F9 B5 B1051.3	15600
F9 B5 B1056.4	15600
F9 B5 B1048.5	15600
F9 B5 B1051.4	15600
F9 B5 B1049.5	15600
F9 B5 B1060.2	15600
F9 B5 B1058.3	15600
F9 B5 B1051.6	15600
F9 B5 B1060.3	15600
F9 B5 B1049.7	15600

Here we list the failed landing outcomes along with their booster number and launch site name in 2015 using WHERE to find where landing outcome equals failure(drone ship) and the date equals 2015



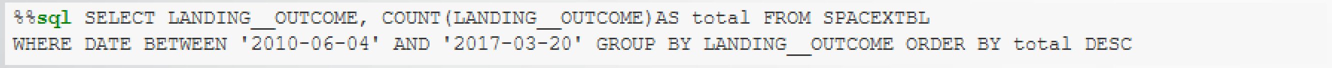
```
%sql SELECT LANDING__OUTCOME, BOOSTER_VERSION, LAUNCH_SITE FROM SPACEXTBL WHERE LANDING__OUTCOME = 'Failure (drone ship)' AND
YEAR(DATE) = '2015'
```



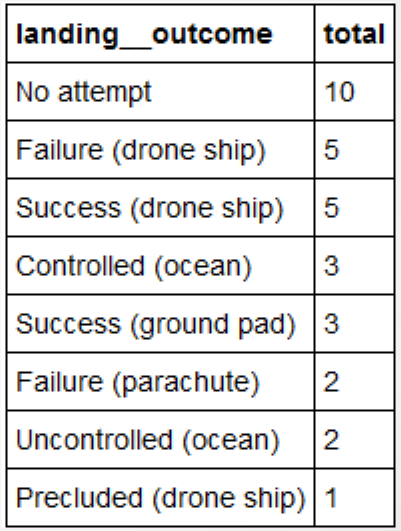
landing__outcome	booster_version	launch_site
Failure (drone ship)	F9 v1.1 B1012	CCAFS LC-40
Failure (drone ship)	F9 v1.1 B1015	CCAFS LC-40

Exploratory Data Analysis Results

Here we rank the landing outcomes by total number between June 4th 2010 and March 20th 2017. Using COUNT() to total the outcomes and AS to set it to a variable name. WHERE command to get the date then GROUP BY to group the outcomes BY the total using DESC to make it descending order

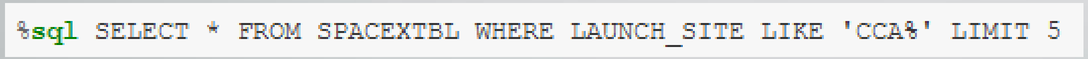


```
%%sql SELECT LANDING__OUTCOME, COUNT(LANDING__OUTCOME)AS total FROM SPACEXTBL
WHERE DATE BETWEEN '2010-06-04' AND '2017-03-20' GROUP BY LANDING__OUTCOME ORDER BY total DESC
```

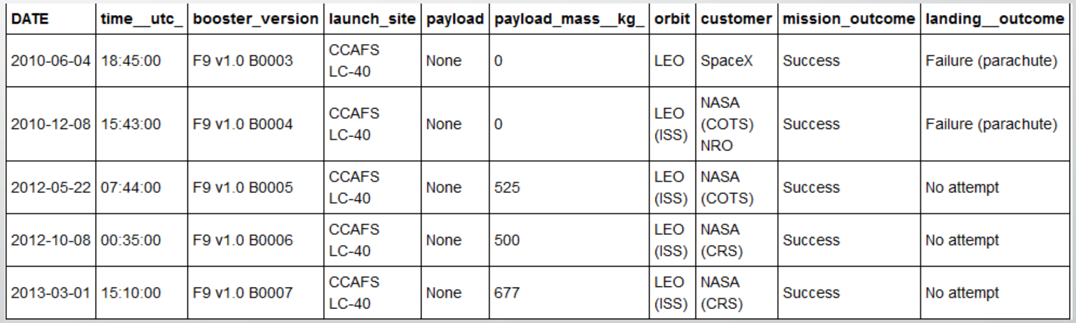


landing__outcome	total
No attempt	10
Failure (drone ship)	5
Success (drone ship)	5
Controlled (ocean)	3
Success (ground pad)	3
Failure (parachute)	2
Uncontrolled (ocean)	2
Precluded (drone ship)	1

Here we display 5 records where the string starts with CCA. We can use * to get the whole record of a row then WHERE to find the launch site. Using a wild card 'CCA%' we locate the sites that start with CCA then LIMIT to limit to only 5 records

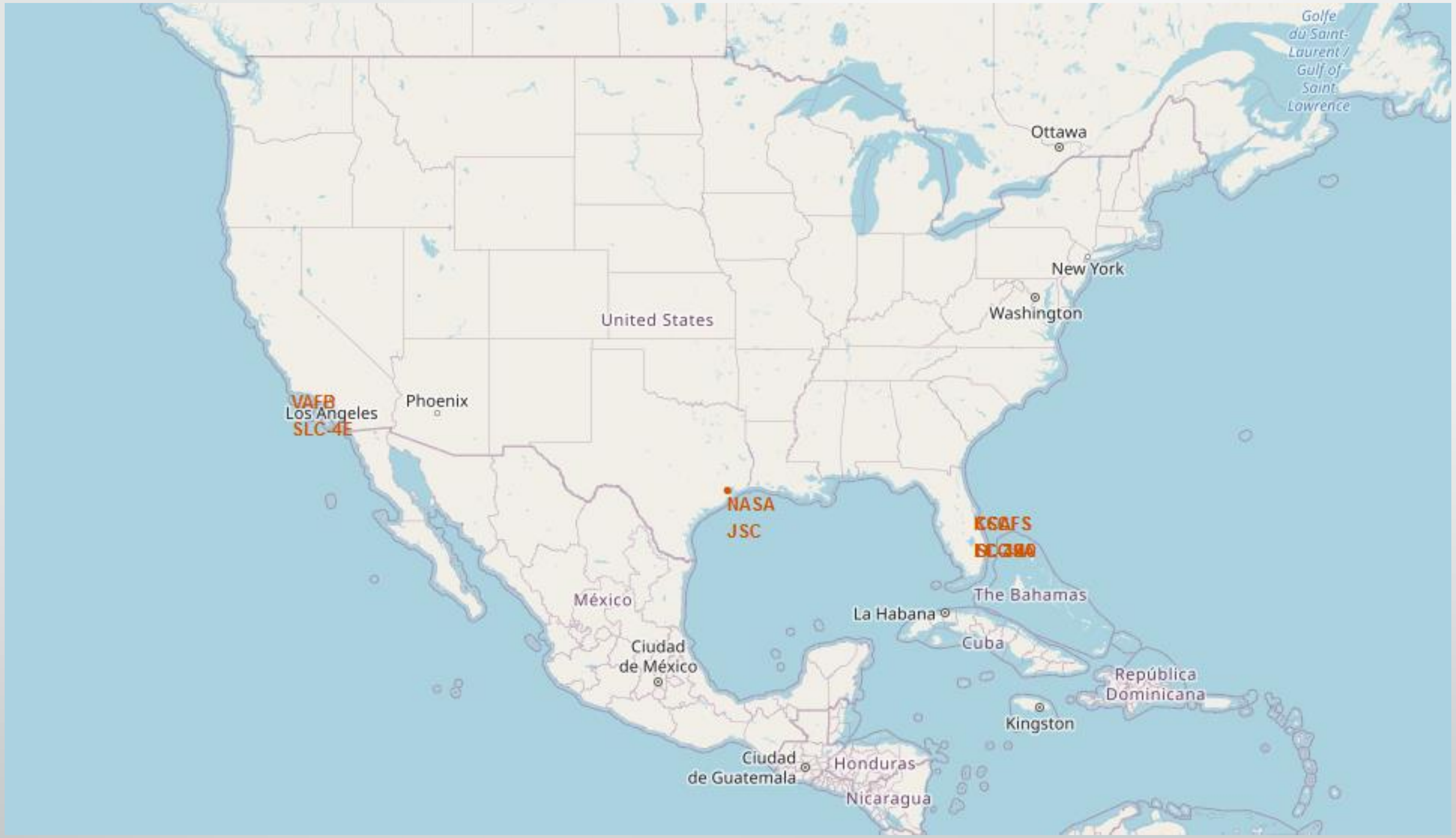


```
%sql SELECT * FROM SPACEXTBL WHERE LAUNCH_SITE LIKE 'CCA%' LIMIT 5
```



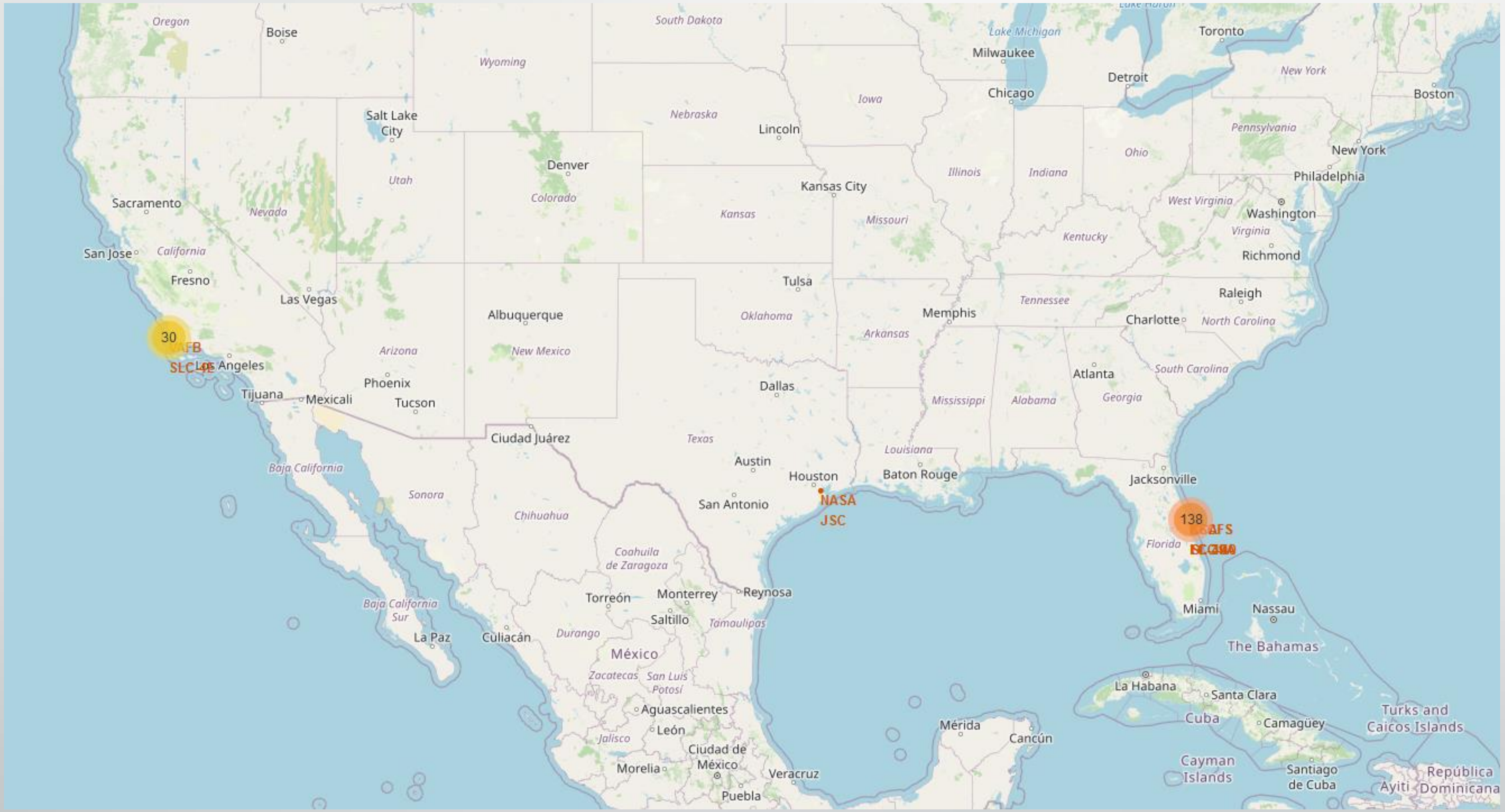
DATE	time__utc_	booster_version	launch_site	payload	payload_mass__kg_	orbit	customer	mission_outcome	landing__outcome
2010-06-04	18:45:00	F9 v1.0 B0003	CCAFS LC-40	None	0	LEO	SpaceX	Success	Failure (parachute)
2010-12-08	15:43:00	F9 v1.0 B0004	CCAFS LC-40	None	0	LEO (ISS)	NASA (COTS) NRO	Success	Failure (parachute)
2012-05-22	07:44:00	F9 v1.0 B0005	CCAFS LC-40	None	525	LEO (ISS)	NASA (COTS)	Success	No attempt
2012-10-08	00:35:00	F9 v1.0 B0006	CCAFS LC-40	None	500	LEO (ISS)	NASA (CRS)	Success	No attempt
2013-03-01	15:10:00	F9 v1.0 B0007	CCAFS LC-40	None	677	LEO (ISS)	NASA (CRS)	Success	No attempt

Interactive Analysis Results



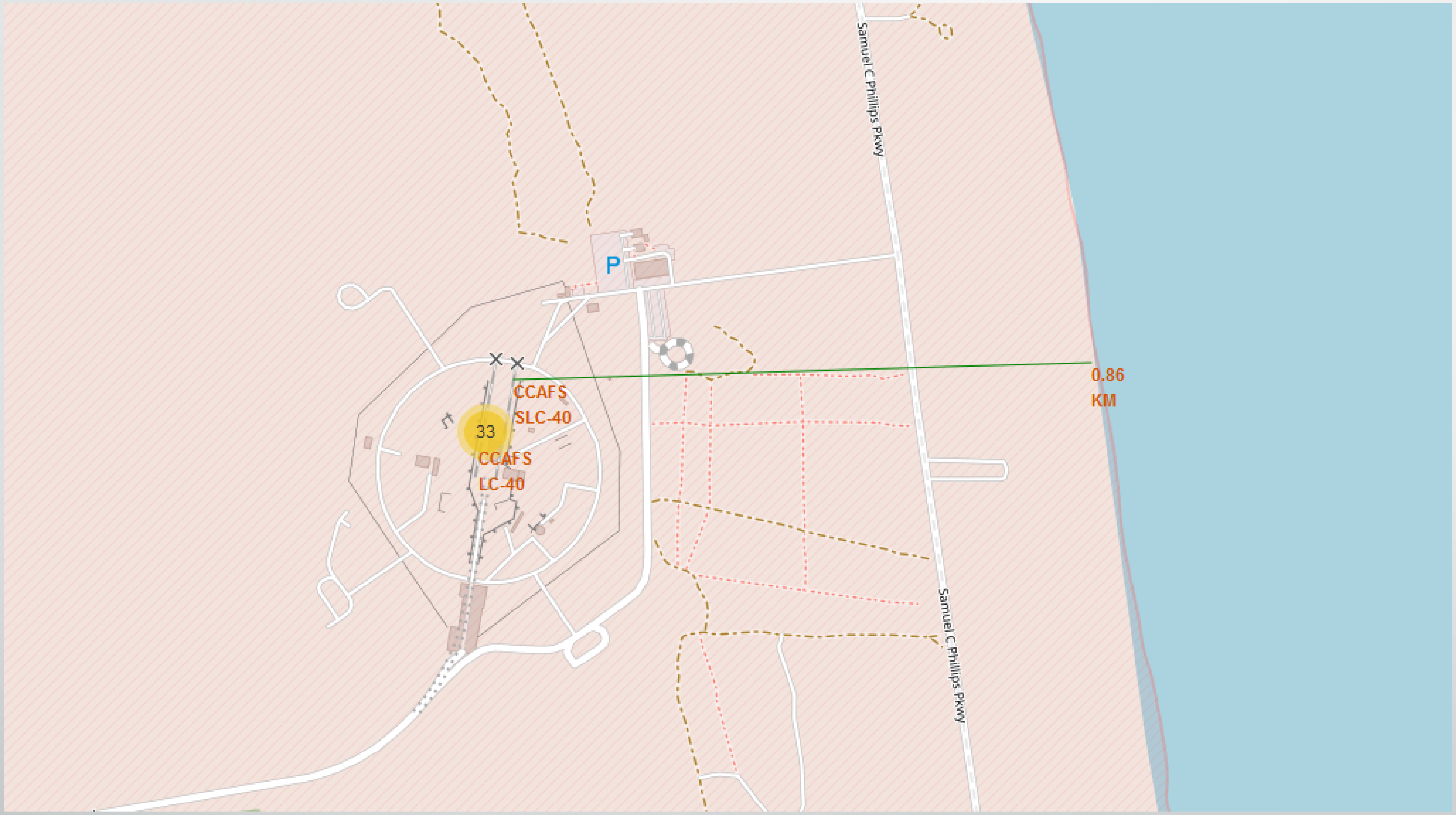
All launch sites are marked on the map of the United states

Interactive Analysis Results



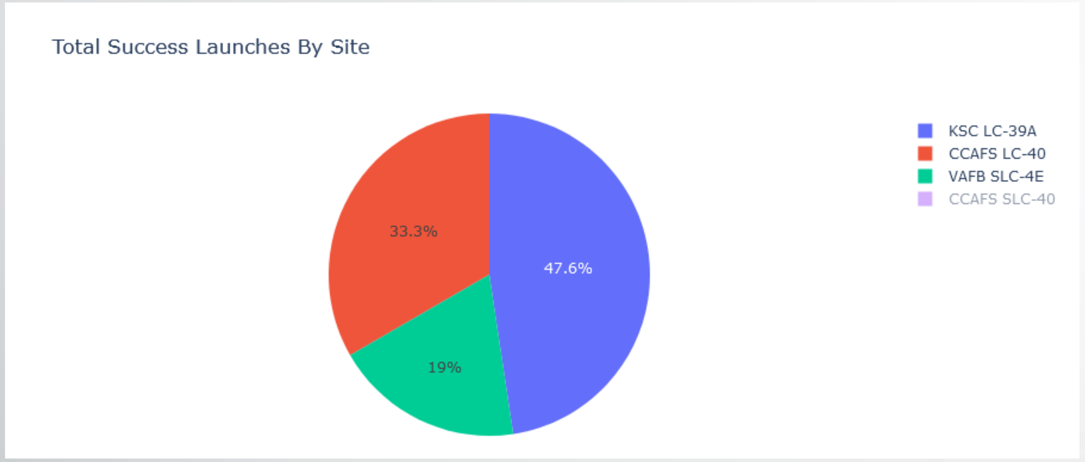
The launches are labeled on each site with the cluster total posted from a zoomed-out view

Interactive Analysis Results



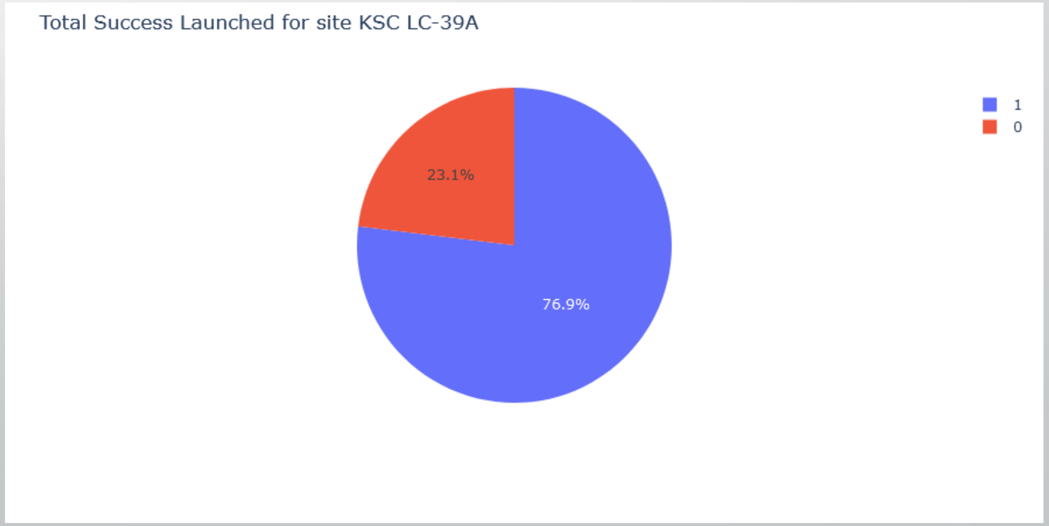
Distance to the coastline from CCAFS SLC-40

Interactive Analysis Results



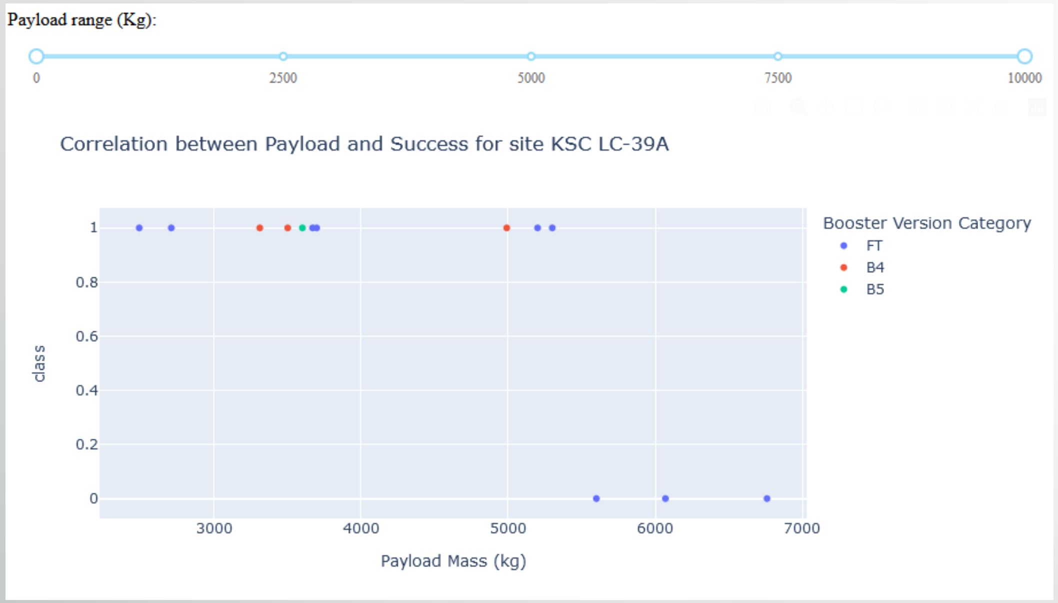
We can see that most of the successful launches happen at Kennedy Space Center

Looking further into KSC we see the Falcon 9 has a 76.6% success rate (denoted by a 1)



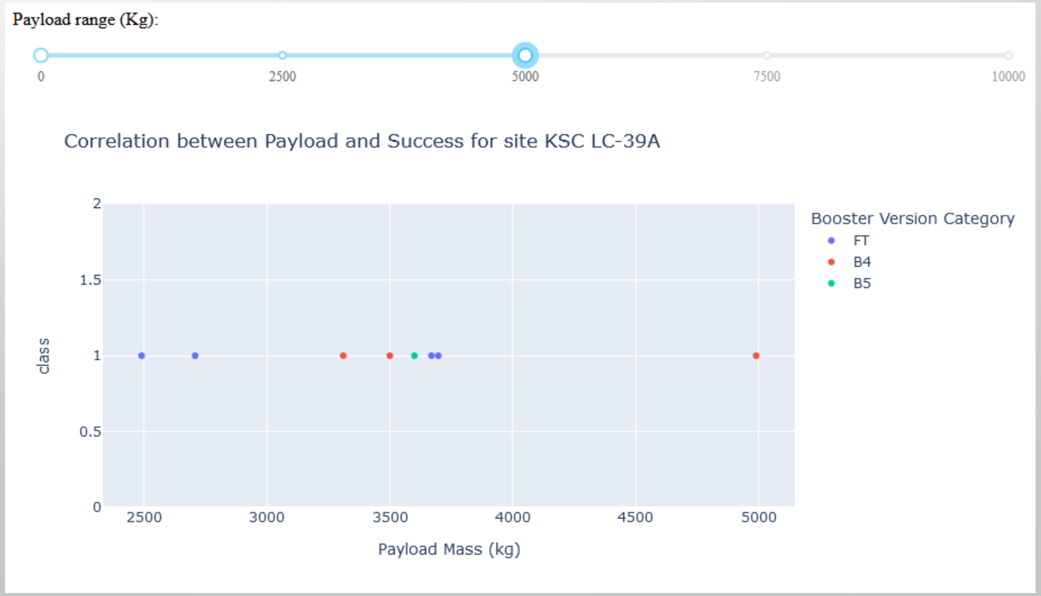
Interactive Analysis Results

Here we look at the Kenndey Space Center launches and see their correlation of payload mass of different booster versions and success rate

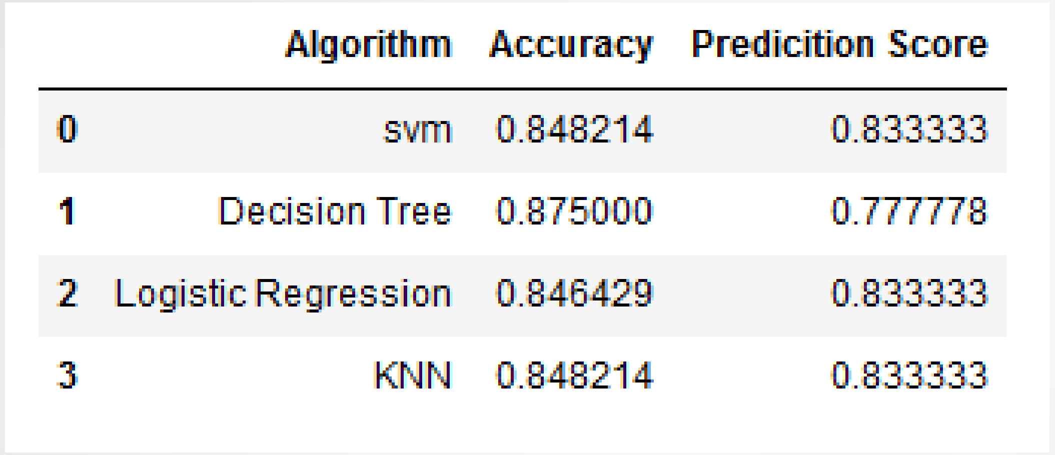


The booster version FT has the most diverse payload mass but at about 5500 kg it has lower success as it gains more payload mass

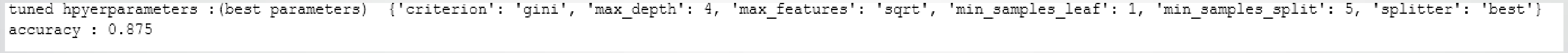
Cutting down to 0 to 5000 kg we can see that there has only been successes



Predictive Analysis Results



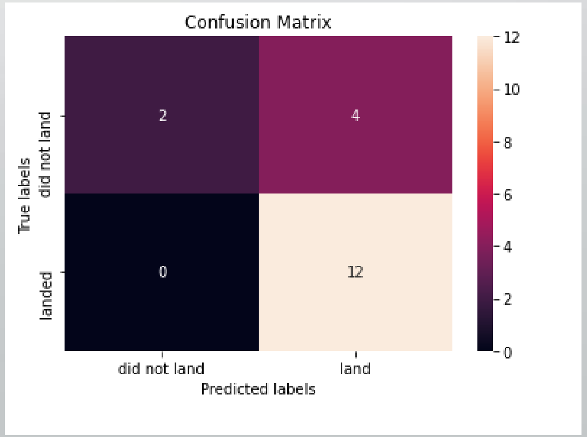
	Algorithm	Accuracy	Predicition Score
0	svm	0.848214	0.833333
1	Decision Tree	0.875000	0.777778
2	Logistic Regression	0.846429	0.833333
3	KNN	0.848214	0.833333



```
tuned hpyerparameters :(best parameters)  {'criterion': 'gini', 'max_depth': 4, 'max_features': 'sqrt', 'min_samples_leaf': 1, 'min_samples_split': 5, 'splitter': 'best'}
accuracy : 0.875
```

In testing the algorithms, the best one is the decision tree with these hyperparameters. There can be a case made for the SVM and KNN as they have the next best accuracy and have a higher test data prediction score

In looking at its confusion matrix we can see that about 25% of the data it predicted were false positives



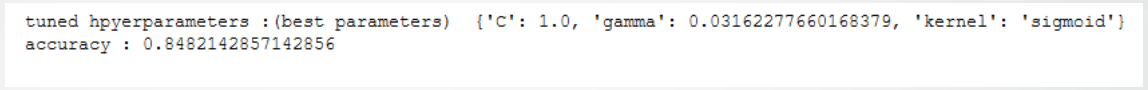
Conclusion

After the analysis we can conclude several things:

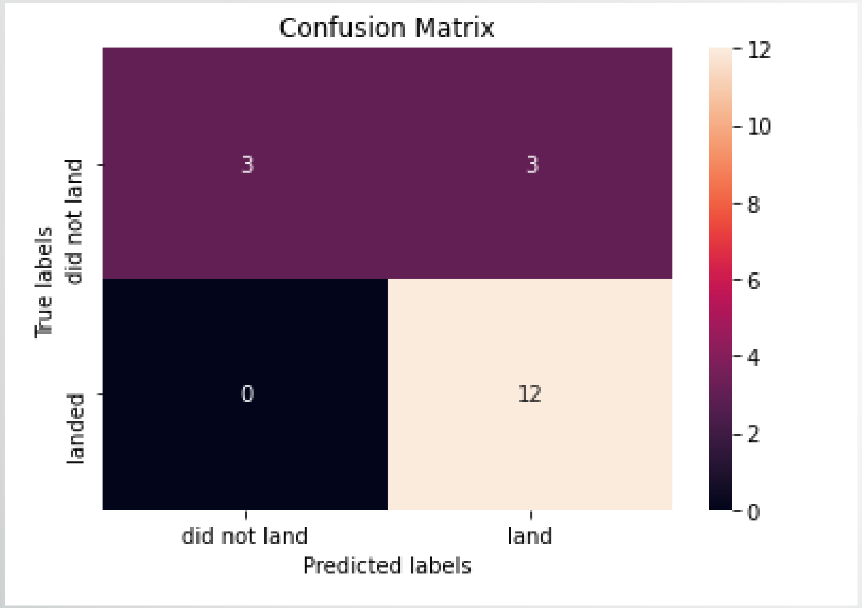
- SpaceX successes is proportional to time(years) eventually their success rate will be near perfect
- Among its orbit types its most successful are GEO, SSO, HEO, and ES-L1
- The most successful launch site is KSC LC-39A
- Lighter payload masses are the best performing among the payload weights
- The decision tree algorithm is the best to model the data

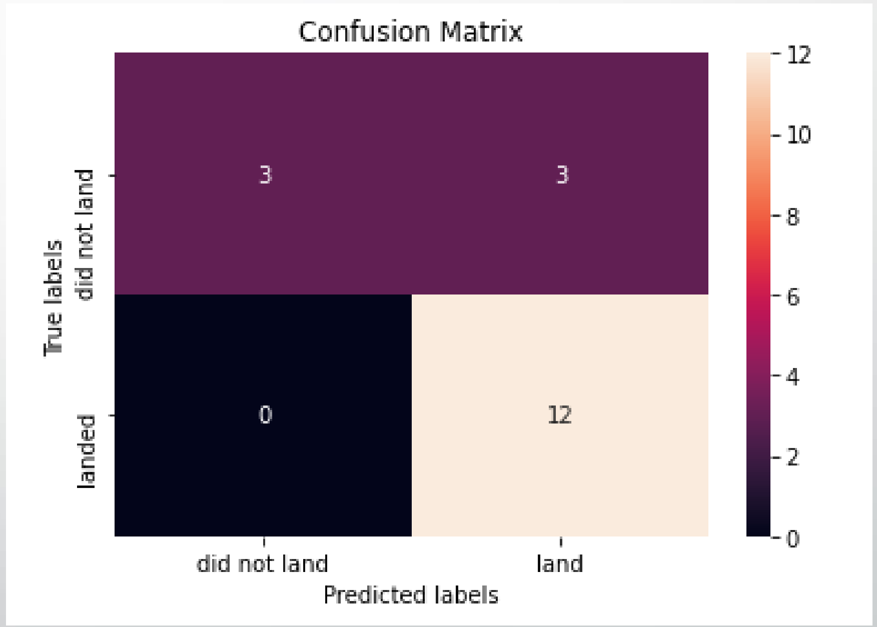
Appendix

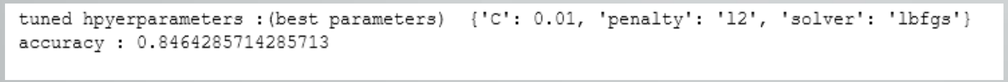
SVM confusion matrix and hyperparameters



```
tuned hpyerparameters :(best parameters)  {'C': 1.0, 'gamma': 0.03162277660168379, 'kernel': 'sigmoid'}
accuracy : 0.8482142857142856
```



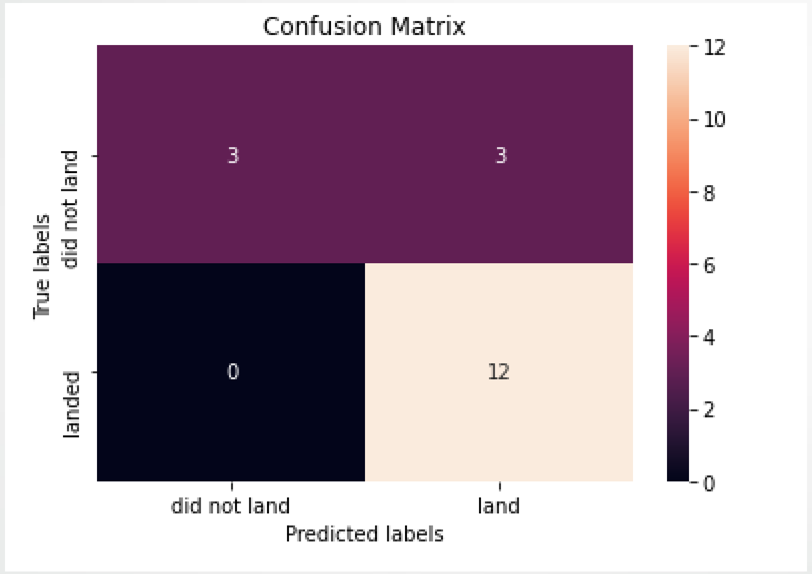




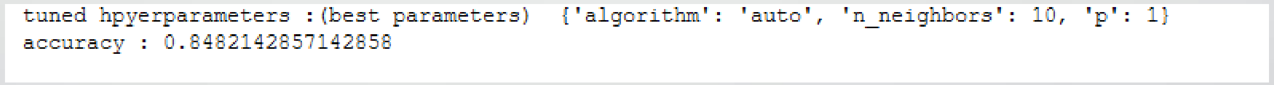
```
tuned hpyerparameters :(best parameters)  {'C': 0.01, 'penalty': 'l2', 'solver': 'lbfgs'}
accuracy : 0.8464285714285713
```

Logistic regression confusion matrix and hyperparameters

Appendix



Knn confusion matrix and hyperparameters



```
tuned hpyerparameters :(best parameters)  {'algorithm': 'auto', 'n_neighbors': 10, 'p': 1}
accuracy : 0.8482142857142858
```