IBM Developer
SKILLS NETWORK

Winning Space Race with Data Science

SHANKAR DURAI
08-11-2023

Outline

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

Executive Summary

- Summary of methodologies:

Following concepts and methods were used to collect and analyze data, build and evaluate machine learning models, and make predictions:

- Collect data through API and Web scraping
- Transform data through data wrangling
- Conduct exploratory data analysis with SQL and data visuals
- Build an interactive map with folium to analyze launch site proximity
- Build a dashboard to analyze launch records interactively with Plotly Dash
- Finally, build a predictive model to predict if the first stage of Falcon 9 will land successfully

- Summary of all results

- Data analysis results
- Data visuals, interactive dashboards
- Predictive model analysis results

INTRODUCTION

PROJECT BACKGROUND AND CONTEXT

SpaceX is most successful commercial space company making affordable space travel possible. One reason SpaceX can do this is the rocket launches are relatively inexpensive. SpaceX advertises Falcon 9 rocket launches on its website with a cost of 62 million dollars; other providers cost upwards 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. Spaces X's Falcon 9 launch like regular rockets.

EXPLORE

Determine the price of eachlaunch. Gathering information about Space X and creating dashboards.

Determine if SpaceX will reuse the first stage using machine learning models.

Section 1

Methodology

Methodology

Executive Summary

- Data collection methodology:
  - SpaceX REST API and web scrapping techniques
- Perform data wrangling
  - Filtering data , handling missing, invalid data and applying one hot encoding
- Perform exploratory data analysis (EDA) using visualization and SQL
- Perform interactive visual analytics using Folium and Plotly Dash
- Perform predictive analysis using classification models
  - Tune and evaluate to find best parameter and models

Data Collection

Steps

- Request data from SpaceX API (rocket launch data)
- Decode response using .json() and convert to a dataframe using .json_normalize()
- Request information about the launches from SpaceX API using custom functions
- Create dictionary from the data
- Create dataframe from the dictionary
- Filter dataframe to contain only Falcon 9 launches
- Replace missing values of Payload Mass with calculated .mean()
- Export data to csv file

Data Collection – SpaceX API

1. API Request and read response into DF → 2. Declare global variables → 3. Call helper functions with API calls to populate global vars

4.Construct data using dictionary → 5.Con dic to Dataframe, filter for Falcon9 launches, covert to CSV

GIT link

1

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

We should see that the request was successfull with the 200 status response code

```
response.status_code
```

```
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
response=requests.get(static_json_url)
response.json()
data=pd.json_normalize(response.json())
```

Using the dataframe `data` print the first 5 rows

2.

```
#Global variables 
BoosterVersion = []
PayloadMass = []
Orbit = []
LaunchSite = []
Outcome = []
Flights = []
GridFins = []
Reused = []
Legs = []
LandingPad = []
Block = []
ReusedCount = []
Serial = []
Longitude = []
Latitude = []
```

3.

```
[15]: # Call getBoosterVersion
getBoosterVersion(data)
```

the list has now been update

```
[16]: BoosterVersion[0:5]
```

```
[16]: ['Falcon 1', 'Falcon 1', 'Falcon 1', 'Falcon 1', 'Falcon 9']
```

we can apply the rest of the functions here:

```
[17]: # Call getLaunchSite
getLaunchSite(data)
```

```
[18]: # Call getPayloadData
getPayloadData(data)
```

```
[19]: # Call getCoreData
getCoreData(data)
```

4

Finally lets construct our dataset using the data we have obtained. We we combine the columns into a 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}
```

5

Then, we need to create a Pandas data frame from the dictionary launch_dict.

Then, we need to create a Pandas data frame from the dictionary la

```
21]: # Create a data from launch_dict
df=pd.DataFrame.from_dict(launch_dict,orient='index')
df1=df.transpose()
```

5

data to a new dataframe called `data_falcon9`.

```
1]: # Hint data['BoosterVersion']!='Falcon 1'
data_falcon9 = df1[df1['BoosterVersion'] != 'Falcon 1']
data_falcon9
```

Data Collection - Scraping

1. Request the Falcon9 Launch Wiki page from its URL

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

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

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

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

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

2. Extract all column/variable names from the HTML table header

```
column_names = []
for i in first_launch_table.find_all('th'):
    column_names.append(i.text.replace('\n','').strip())

# 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_nam
```

Check the extracted column names

```
print(column_names)
```

```
['Flight No.', 'Date andtime (UTC)', 'Version,Booster [b]', 'Launch site', 'Payload[c]', 'Payload mass', 'Orbit', 'Customer', 'Launchoutcome', 'Boosterlanding']
```

3.Create a data frame by parsing the launch HTML tables

```
launch_dict= dict.fromkeys(column_names)

# Remove an irrelvant column
del launch_dict['Date andtime (UTC)']

# 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']=[]
```

After you have fill in the parsed launch record values into `launch_dict`, you can create a dataframe from it.

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

GIT link

Data Wrangling

• Conducted Exploratory Data Analysis (EDA) to find patters in data and define labels for training supervised models

git wrangling

The data set contained various mission outcomes that were converted into Training Labels with 1 meaning the booster successfully landed and 0 meaning booster was unsuccessful in landing.

Following landing scenarios were considered to create labels:

- True Ocean means the mission outcome was successfully landed to a specific region of the ocean
- False Ocean means the mission outcome was unsuccessfully landed to a specific region of the ocean
- RTLS means the mission outcome was successfully landed to a ground pad
- False RTLS means the mission outcome was unsuccessfully landed to a ground pad
- True ASDS means the mission outcome was successfully landed on a drone ship
- False ASDS means the mission outcome was unsuccessfully landed on a drone ship

- We performed exploratory data analysis and determined the training labels.
- We calculated the number of launches at each site, and the number and occurrence of each orbits
- We created landing outcome label from outcome column and exported the results to csv.

We create a set of outcomes where the second stage did not land successfully:

```python
bad_outcomes=set(landing_outcomes.keys()[[1,3,5,6,7]])
bad_outcomes
```

```
{'False ASDS', 'False Ocean', 'False RTLS', 'None ASDS', 'None None'}
```

TASK 4: Create a landing outcome label from Outcome column ¶

Using the `Outcome`, create a list where the element is zero if the corresponding row in `Outcome` is in the set `bad_outcome`; otherwise, it's one. Then assign it to the variable `landing_class`:

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

This variable will represent the classification variable that represents the outcome of each launch. If the

```python
# landing_outcomes = values on Outcome column
landing_outcomes = df['Outcome'].value_counts()
```

`True Ocean` means the mission outcome was successfully lanc while `False Ocean` means the mission outcome was unsucces ocean. `True RTLS` means the mission outcome was successfull means the mission outcome was unsuccessfully landed to a grou outcome was successfully landed to a drone ship `False ASDS` unsuccessfully landed to a drone ship. `None ASDS` and `None N`

```python
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 False Ocean
6 None ASDS
7 False RTLS
```

We create a set of outcomes where the second stage did not lar

```python
df['Class']=landing_class
df[['Class']].head(8)
```

	Class
0	0
1	0
2	0
3	0
4	0
5	0
6	1
7	1

EDA with Data Visualization

- As part of the Exploratory Data Analysis (EDA), following charts were plotted to gain further insights into the dataset:

1. Scatter plot:
   - Shows relationship or correlation between two variables making patterns easy to observe
   - Plotted following charts to visualize:
     Relationship between Flight Number and Launch Site
     Relationship between Payload and Launch Site
     Relationship between Flight Number and Orbit Type
     Relationship between Payload and Orbit Type

2. Bar Chart:
   - Commonly used to compare the values of a variable at a given point in time.
     Bar charts makes it easy to see which groups are highest/common and how other groups compare against each other.
     Length of each bar is proportional to the value of the items that it represents
   - Plotted following Bar chart to visualize:
   - Relationship between success rate of each orbit type

3. Line Chart:
   - Commonly used to track changes over a period of time. It helps depict trends over time.

EDA with SQL LINK GIT_SQL

- To better understand SpaceX data set, following SQL queries/operations were performed on an IBM DB2 cloud instance

1.Display the names of the unique launch sites in the space mission

2. Display 5 records where launch sites begin with the string 'CCA’

3. Display the total payload mass carried by boosters launched by NASA (CRS)

4. Display average payload mass carried by booster version F9 v1.1

5. List the date when the first successful landing outcome in ground pad was achieved.

6. List the names of the boosters which have success in drone ship and have payload mass greater than 4000 but less than 6000

7. List the total number of successful and failure mission outcomes

8. List the names of the booster_versions which have carried the maximum payload mass. Use a subquery

9. List the failed landing_outcomes in drone ship, their booster versions, and launch site names for in year 2015 10.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

Build an Interactive Map with Folium

- Folium interactive map helps analyze geospatial data to perform more interactive visual analytics and better understand factors such location and proximity of launch sites that impact launch success rate
- Following map object were created and added to the map:
- Mark all launch sites on the map. This allowed to visually see the launch sites on the map. Added ‘folium.circle’ and ‘folium.marker’ to highlight circle area with a text label over each launch site.
- Added a ‘MarkerCluster()’ to show launch success (green) and failure (red) markers for each launch site
- Calculated distances between a launch site to its proximities (e.g., coastline, railroad, highway, city)

FOLIUM_GIT

Build a Dashboard with Plotly Dash

• Built a Plotly Dash web application to perform interactive visual analytics on SpaceX launch data in real-time. Added Launch Site Drop-down, Pie Chart, Payload range slide, and a Scatter chart to the Dashboard.

1.Added a Launch Site Drop-down Input component to the dashboard to provide an ability to filter Dashboard visual by all launch sites or a particular launch site

2. Added a Pie Chart to the Dashboard to show total success launches when ‘All Sites’ is selected and show success and failed counts when a particular site is selected

3. Added a Payload range slider to the Dashboard to easily select different payload ranges to identify visual patterns

4. Added a Scatter chart to observe how payload may be correlated with mission outcomes for selected site(s). The color-label Booster version on each scatter point provided missions outcomes with different boosters

- Dashboard helped answer following questions:

1. Which site has the largest successful launches? KSC LC-39A with 10

2. Which site has the highest launch success rate? KSC LC-39A with 76.9% success
3. Which payload range(s) has the highest launch success rate? 2000 – 5000 kg
4. Which payload range(s) has the lowest launch success rate? 0-2000 and 5500 - 7000
5. Which F9 Booster version (v1.0, v1.1, FT, B4, B5, etc.) has the highest launch success rate? FT

GIT_PLOTLY

Predictive Analysis (Classification)

1. Read dataset into Dataframe and create a ‘Class’ array

2. Standardize the data

3. Train/Test/Split data in to training and test data sets

4. Create and Refine Models

5. Find the best performing Model

```
Y = data['Class'].to_numpy()
```

```
# students get this
transform = preprocessing.StandardScaler()
X = transform.fit(X).transform(X)
```

`X_train, X_test, Y_train, Y_test`

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

we can see we only have 18 test samples.

```
Y_test.shape
```

```
(18,)
```

```
tree_cv = GridSearchCV(estimator=tree, cv=10, param_grid=parameters).fit(X_train, Y_train)
```

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

```
tuned hpyerparameters :(best parameters)  {'criterion': 'gini', 'max_depth': 2, 'max_features': 'sq
les_leaf': 1, 'min_samples_split': 2, 'splitter': 'random'}
accuracy : 0.9017857142857142
```

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

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

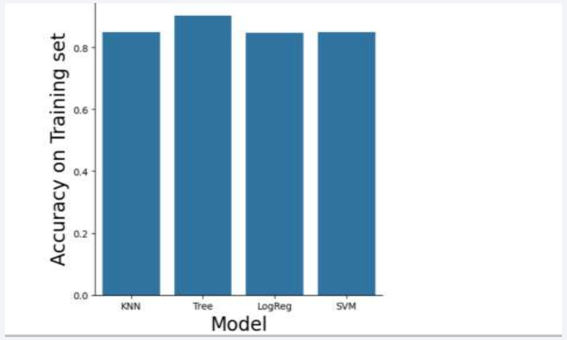
We output the `GridSearchCV` object for logistic regression. We display the best parameters using the data attribute `best_params_` and the accuracy on the validation data using the data attribute `best_score_`.

```
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 : 0.8464285714285713
```

```
logreg_score = logreg_cv.score(X_test, Y_test)
print("score :", logreg_score)
```

```
score : 0.8333333333333334
```

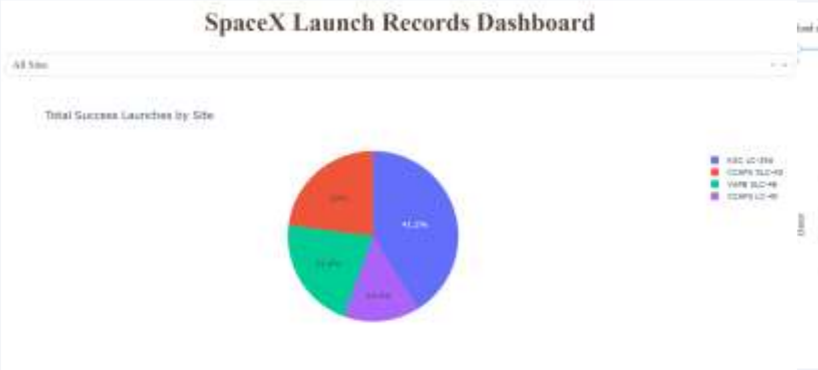


The best performing model is decision tree with score of 0.90 approx.
GITHUB link: [GIT_PREDICT]

Results

- Exploratory data analysis results

- Interactive analytics demo in screenshots



- Predictive analysis results

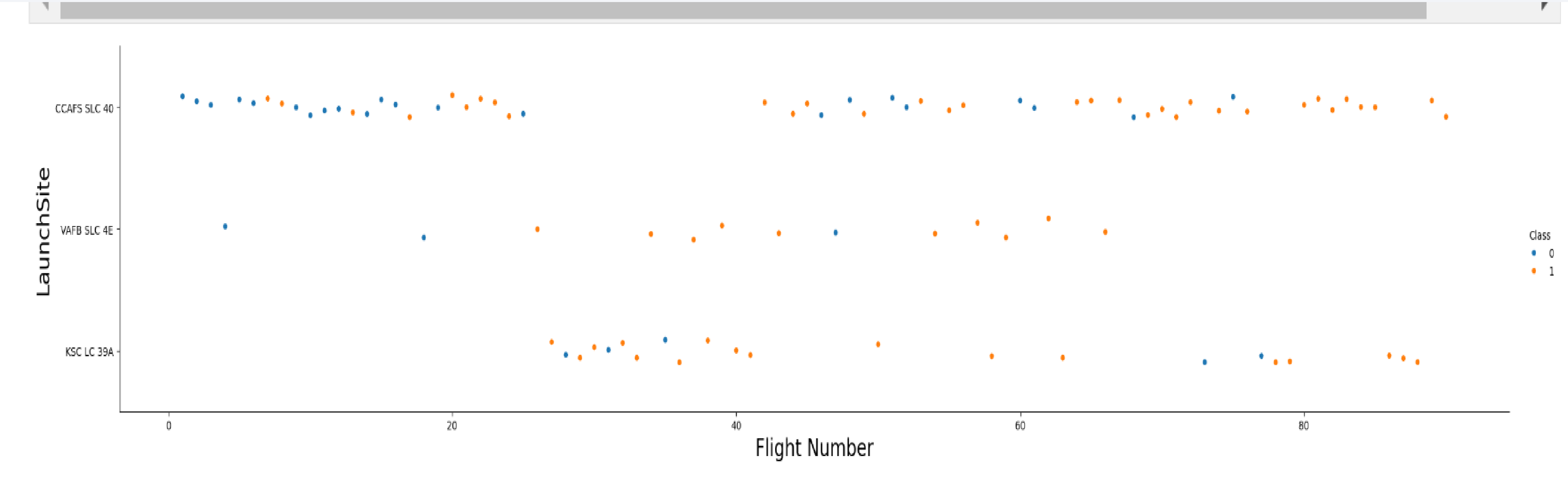
Section 2

Insights drawn from EDA

Flight Number vs. Launch Site

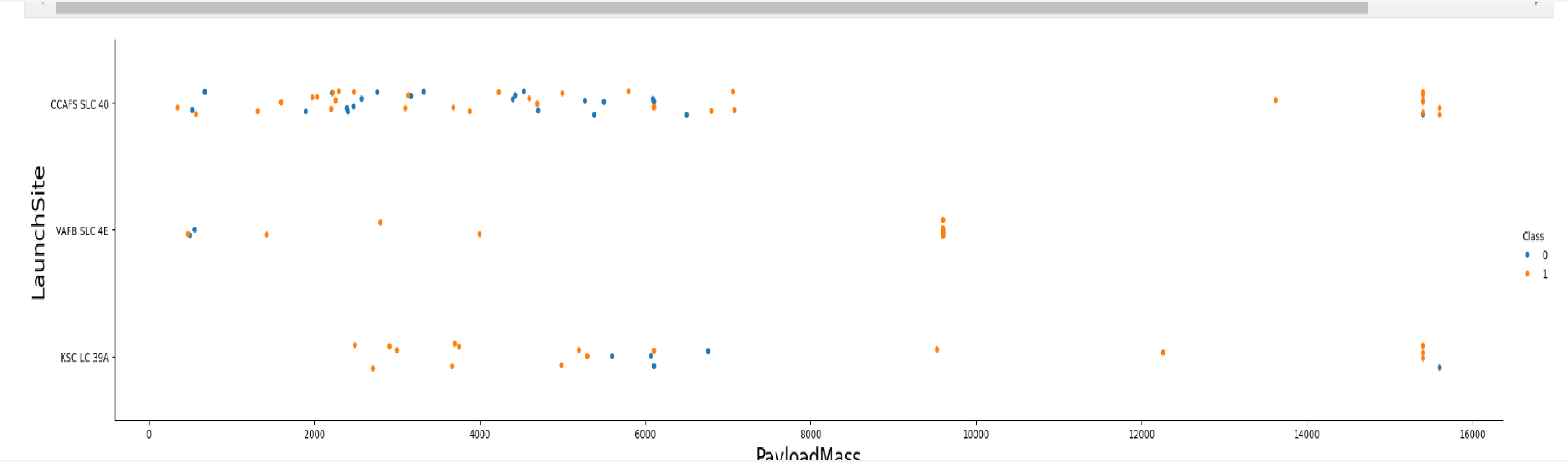
- Success rates (Class=1) increases as the number of flights increase

- For launch site ‘KSC LC 39A’, it takes at least around 25 launches before a first successful launch



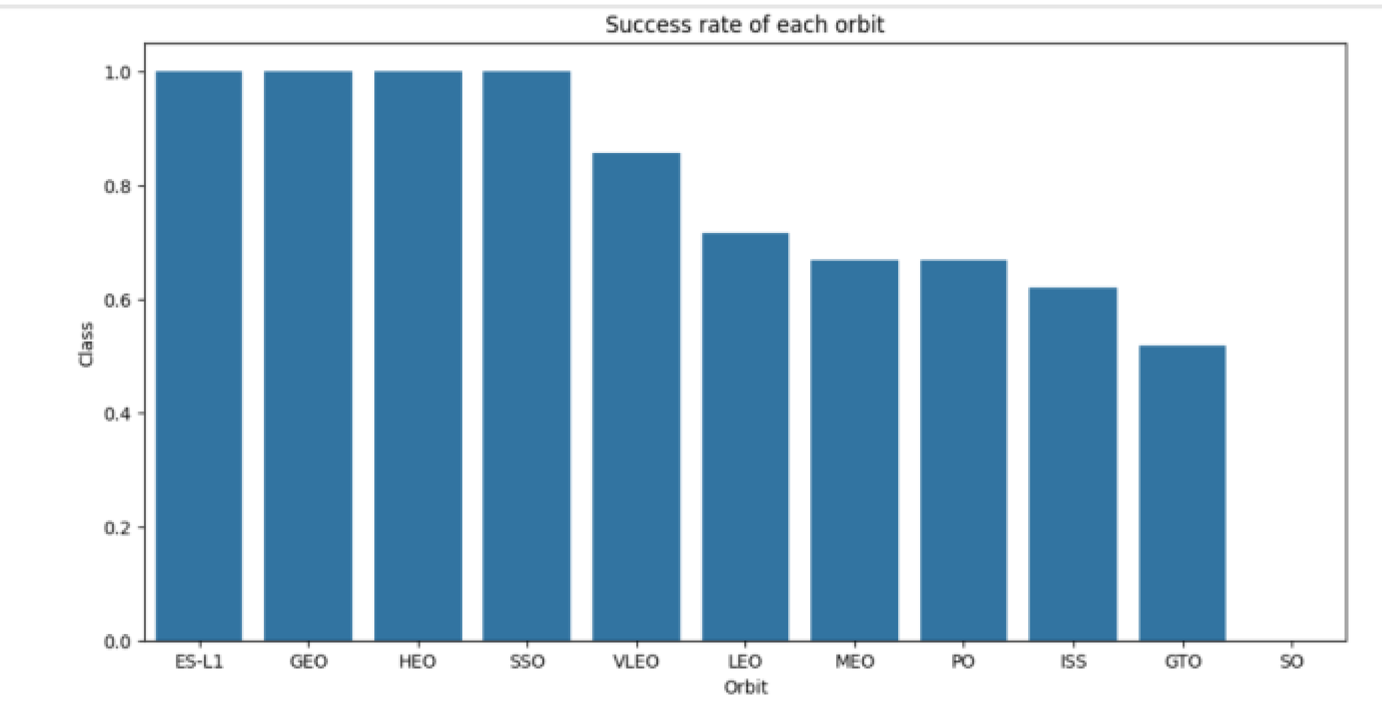
Payload vs. Launch Site

- For launch site ‘VAFB SLC 4E’, there are no rockets launched for payload greater than 10,000 kg
- Percentage of successful launch (Class=1) increases for launch site ‘VAFB SLC 4E’ as the payload mass increases
- There is no clear correlation or pattern between launch site and payload mass



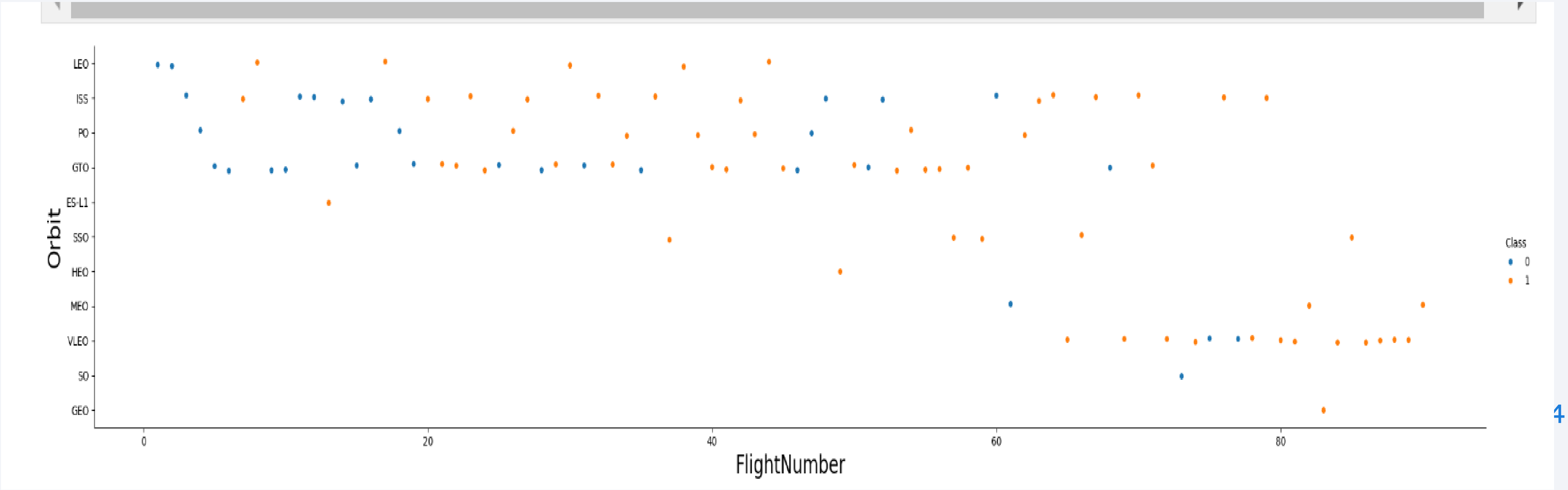
Success Rate vs. Orbit Type

- Orbits ES-LI, GEO, HEO, and SSO have the highest success rates
- GTO orbit has the lowest success rate



Flight Number vs. Orbit Type

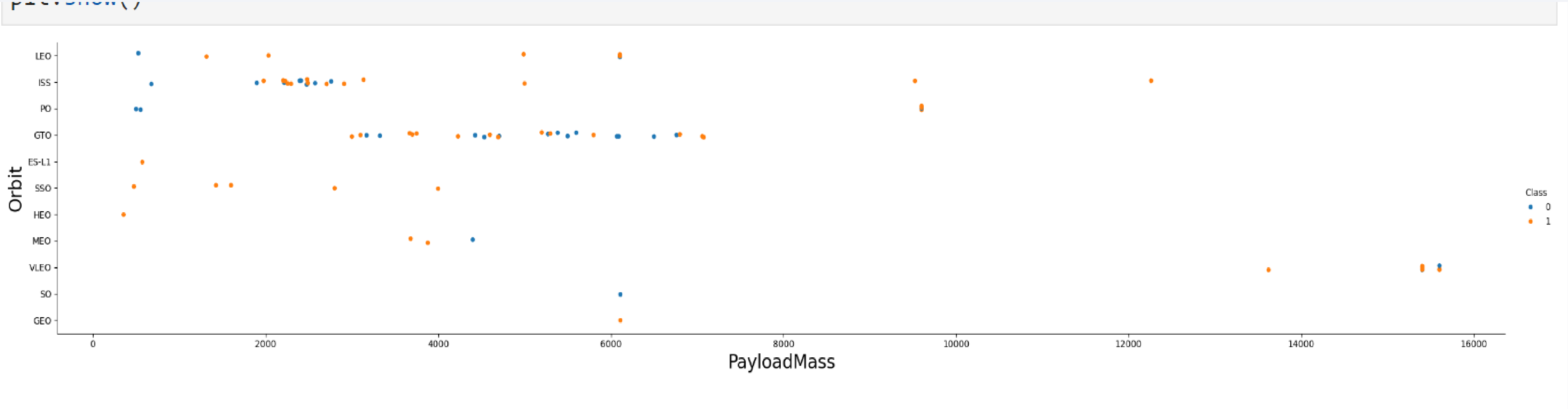
- For orbit VLEO, first successful landing (class=1) doesn't occur until 60+ number of flights
- For most orbits (LEO, ISS, PO, SSO, MEO, VLEO) successful landing rates appear to increase with flight numbers
- There is no relationship between flight number and orbit for GTO



Payload vs. Orbit Type

- Successful landing rates (Class=1) appear to increase with pay load for orbits LEO, ISS, PO, and SSO

- For GEO orbit, there is not clear pattern between payload and orbit for successful or unsuccessful landing

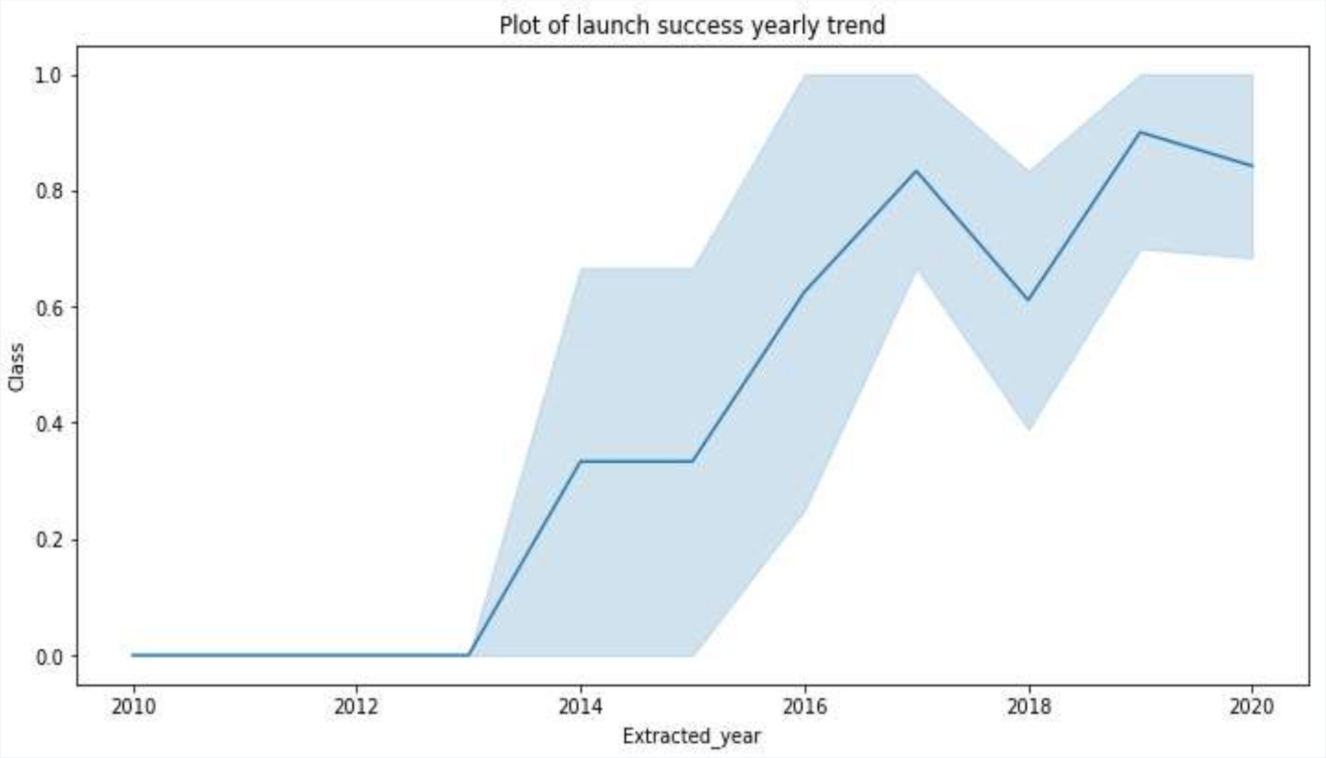


Launch Success Yearly Trend

- Success rate (Class=1) increased by about 80% between 2013 and 2020

- Success rates remained the same between 2010 and 2013 and between 2014 and 2015

- Success rates decreased between 2017 and 2018 and between 2019 and 2020



All Launch Site Names

- ‘distinct’ returns only unique values from the queries column (Launch_Site)

- There are 4 unique launch sites

```
%sql SELECT DISTINCT Launch_Site FROM SPACEXTABLE
```

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

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

Launch Site Names Begin with 'CCA'

- Using keyword ‘Like’ and format ‘CCA%’, returns records where ‘Launch_Site’ column starts with “CCA”.
- Limit 5, limits the number of returned records to 5

Display 5 records where launch sites begin with the string 'CCA'

```
%sql SELECT * FROM SPACEXTABLE WHERE Launch_Site LIKE '%CCA%' LIMIT 5
 * sqlite:///my_data1.db
Done.
```

Date	Time (UTC)	Booster_Version	Launch_Site	Payload	PAYLOAD_MASS__KG_	Orbit	Customer	Mission_Outcome
2010-04-06	18:45:00	F9 v1.0 B0003	CCAFS LC-40	Dragon Spacecraft Qualification Unit	0	LEO	SpaceX	Success
2010-08-12	15:43:00	F9 v1.0 B0004	CCAFS LC-40	Dragon demo flight C1, two CubeSats, barrel of Brouere cheese	0	LEO (ISS)	NASA (COTS) NRO	Success
2012-05-22	07:44:00	F9 v1.0 B0005	CCAFS LC-40	Dragon demo flight C2	525	LEO (ISS)	NASA (COTS)	Success
2012-08-10	00:35:00	F9 v1.0 B0006	CCAFS LC-40	SpaceX CRS-1	500	LEO (ISS)	NASA (CRS)	Success
2013-01-03	15:10:00	F9 v1.0 B0007	CCAFS LC-40	SpaceX CRS-2	677	LEO (ISS)	NASA (CRS)	Success

Total Payload Mass

- ‘sum’ adds column ‘PAYLOAD_MASS_KG’ and returns total payload mass for customers named ‘NASA (CRS)’

Display the total payload mass carried by boosters launched by NASA (CRS)

```
%sql SELECT SUM(PAYLOAD_MASS__KG_) FROM SPACEXTABLE WHERE Customer='NASA (CRS)'
```

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

SUM(PAYLOAD_MASS__KG_)
45596

Average Payload Mass by F9 v1.1

- ‘avg’ keyword returns the average of payload mass in ‘PAYLOAD_MASS_KG’ column where booster version is ‘F9 v1.1’

Display average payload mass carried by booster version F9 v1.1

```
%sql SELECT AVG(PAYLOAD_MASS__KG_) FROM SPACEXTABLE WHERE Booster_Version = 'F9 v1.1'
```

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

AVG(PAYLOAD_MASS__KG_)
2928.4

First Successful Ground Landing Date

- ‘min(Date)’ selects the first or the oldest date from the ‘Date’ column where first successful landing on group pad was achieved
- Where clause defines the criteria to return date for scenarios where ‘Landing_Outcome’ value is equal to ‘Success (ground pad)’

```
%sql SELECT min(Date) FROM SPACEXTABLE WHERE Landing_Outcome LIKE 'Success (ground pad)'
 * sqlite:///my_data1.db
Done.
```

min(Date)
2015-12-22

Successful Drone Ship Landing with Payload between 4000 and 6000

- The query finds the booster version where payload mass is greater than 4000 but less than 6000 and the landing outcome is success in drone ship
- The ‘and’ operator in the where clause returns booster versions where both conditions in the where clause are true

```
%sql SELECT Booster_Version FROM SPACEXTABLE WHERE Landing_Outcome like 'Success (drone ship)'\
AND PAYLOAD_MASS__KG_ > 4000 AND PAYLOAD_MASS__KG_ < 6000
```

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

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

Total Number of Successful and Failure Mission Outcomes

- like operator used to find success and failure mission outcome

```
success= %sql SELECT count(Mission_Outcome) from SPACEXTABLE where Mission_Outcome LIKE 'Success%'

failure =%sql SELECT count(Mission_Outcome) from SPACEXTABLE where Mission_Outcome LIKE 'Failure%'

print(success)
print(failure)
```

```
 * sqlite:///my_data1.db
Done.
 * sqlite:///my_data1.db
Done.
+-----------------------+
| count(Mission_Outcome) |
+-----------------------+
|          100          |
+-----------------------+
+-----------------------+
| count(Mission_Outcome) |
+-----------------------+
|           1           |
+-----------------------+
```

Boosters Carried Maximum Payload

- The sub query returns the maximum payload mass by using keywork ‘max’ on the pay load mass column
- The main query returns booster versions and respective payload mass where payload mass is maximum with value of 15600

```
%sql select booster_version,PAYLOAD_MASS__KG_ from SPACEXTABLE where PAYLOAD_MASS__KG_ in (select \
max(PAYLOAD_MASS__KG_) from SPACEXTABLE)

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

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

2015 Launch Records

• The query lists landing outcome, booster version, and the launch site where landing outcome is failed in drone ship and the year is 2015

• The 'and' operator in the where clause returns booster versions where both conditions in the where clause are true

• The 'year' keywork extracts the year from column 'Date

• The results identify launch site as 'CCAFS LC-40' and booster version as F9 v1.1 B1012 and B1015 that had failed landing outcomes in drop ship in the year 2015

```
%sql select substr(Date, 6,2) as month,booster_version,launch_site,landing_outcome \
from SPACEXTABLE where substr(Date,0,5)='2015' and Landing_Outcome like 'Failure (drone ship)'
```

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

month	Booster_Version	Launch_Site	Landing_Outcome
10	F9 v1.1 B1012	CCAFS LC-40	Failure (drone ship)
04	F9 v1.1 B1015	CCAFS LC-40	Failure (drone ship)

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

- The ‘group by’ key word arranges data in column ‘Landing__Outcome’ into groups
- The ‘between’ and ‘and’ keywords return data that is between 2010-06-04 and 2017-03-20
- The result of the query is a ranked list of landing outcome counts per the specified date range

```
%sql select Landing_Outcome,count(Landing_Outcome) as count from SPACEXTABLE where \
Date BETWEEN '2010-06-04'and '2017-03-20' group by Landing_Outcome order by \
count(Landing_Outcome) desc
```

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

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

Section 3

Launch Sites Proximities Analysis

SpaceX Falcon9 - Launch Sites Map

1.ALL LAUNCH SITES

LAUNCH SITES:KSC LC-39A (FL) ,
CCAFS LC-40 (FL) ,CCAFS SLC-40 (FL)

3. CCAFS LC-40 (FL) ,CCAFS SLC-40 (FL)

4.VAFB SLC-4E (CA)

SpaceX Falon9 – Success/Failed Launch Map for all Launch Sites

Fig 1 – US map with all Launch Sites

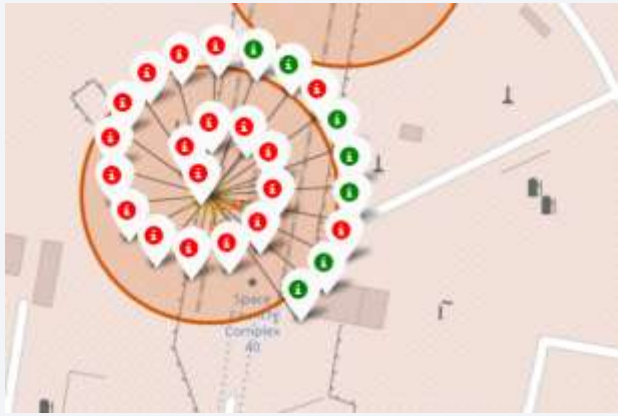


FIG 2-CCAFS LC-40

Fig 1 is the US map with all the Launch Sites.
Fig 2,3,4,5 zoom in to each sites and success markers
Fig 4 FIG4-KSC LC-39A (FL) high success rate in green

FIG3-VAFB SLC-4E (CA)

FIG4-KSC LC-39A (FL)

FIG 5-CCAFS SLC-40

Launch Site distance to landmarks

1 Are launch sites in close proximity to railways?yes.

Nearest railway distance is approx 1.27kms from launch site CCAFS SLC-40

2 Are launch sites in close proximity to highways? yes.

Nearest highway is approx 0.58kms from launch site

CCAFS SLC-40

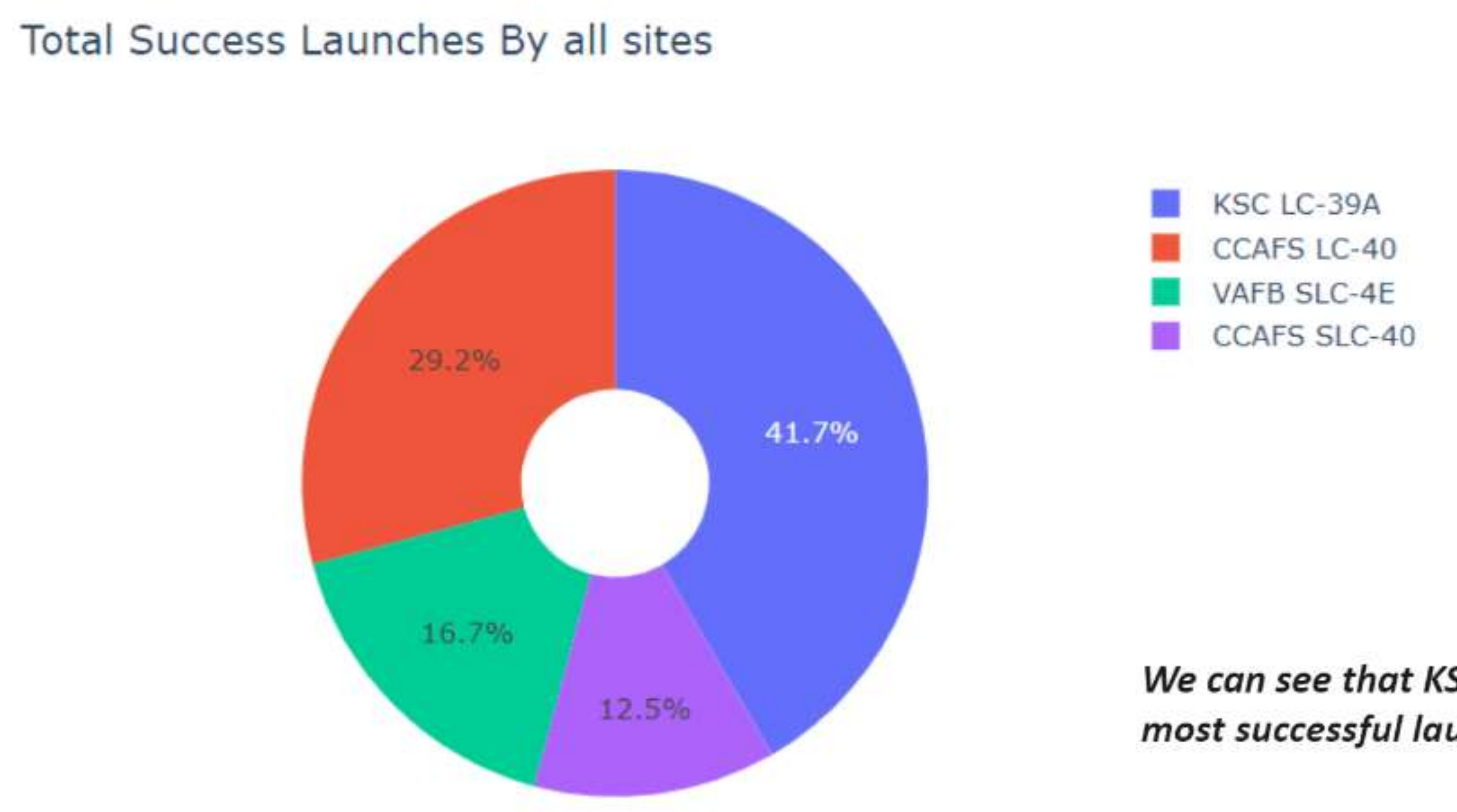
3.Are launch sites in close proximity to coastline?yes.Nearest coastline is approx 0.90kms from launch site CCAFS SLC-40

Do launch sites keep certain distance away from cities? Yes city cape canaveral is approx 19.29kms from launch site CCAFS SLC-40

Section 4

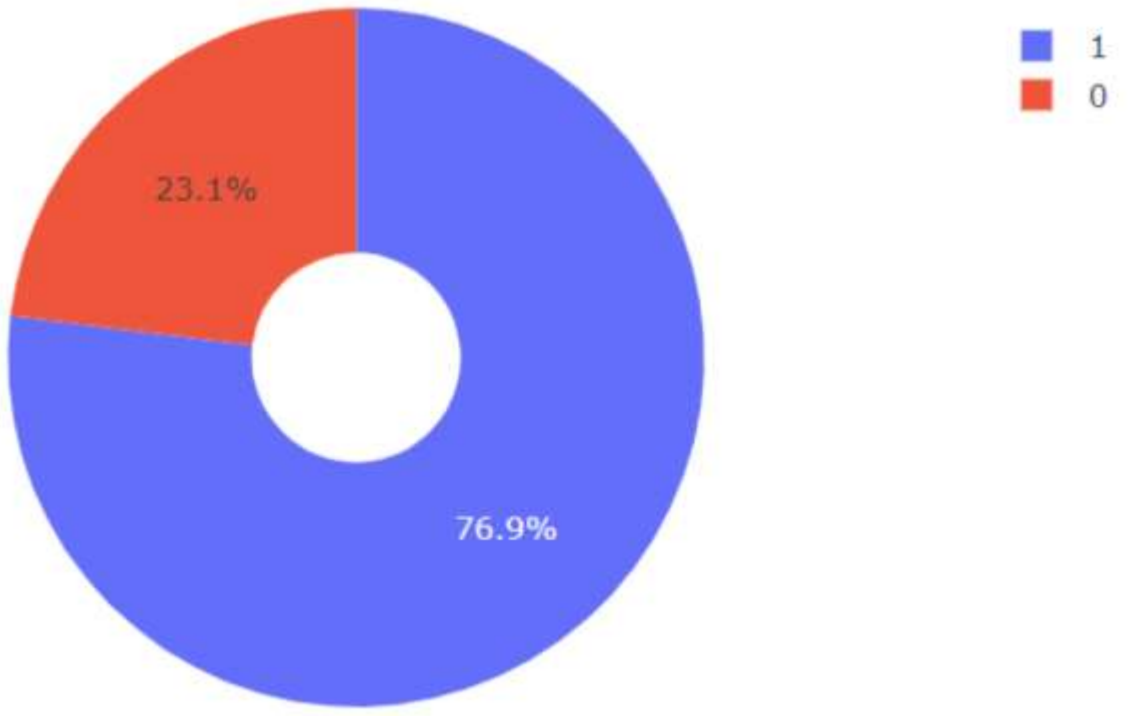
Build a Dashboard with Plotly Dash

Pie chart showing the success percentage achieved by each launch site



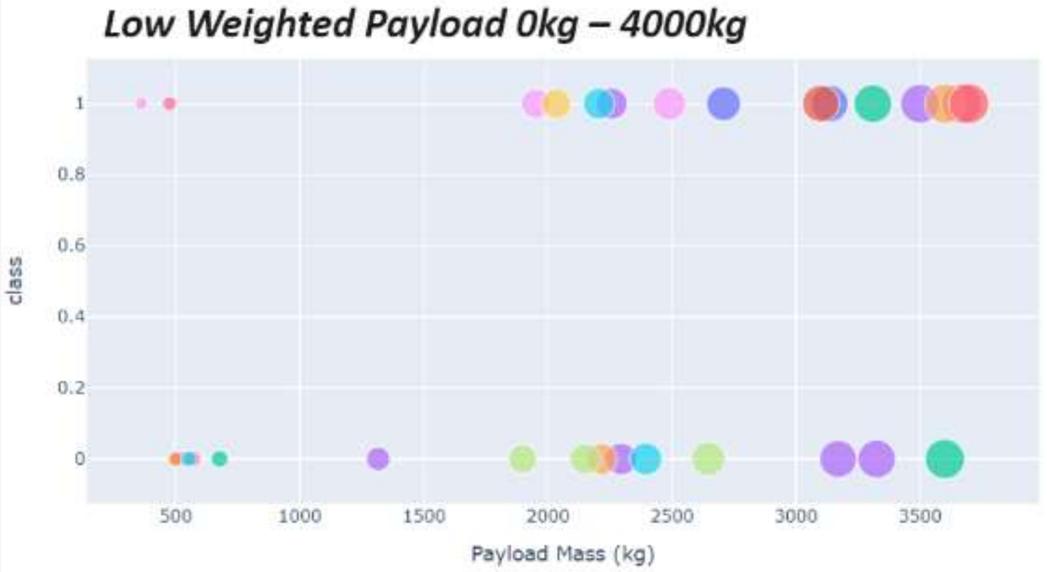
We can see that KSC LC-39A had the most successful launches from all the sites

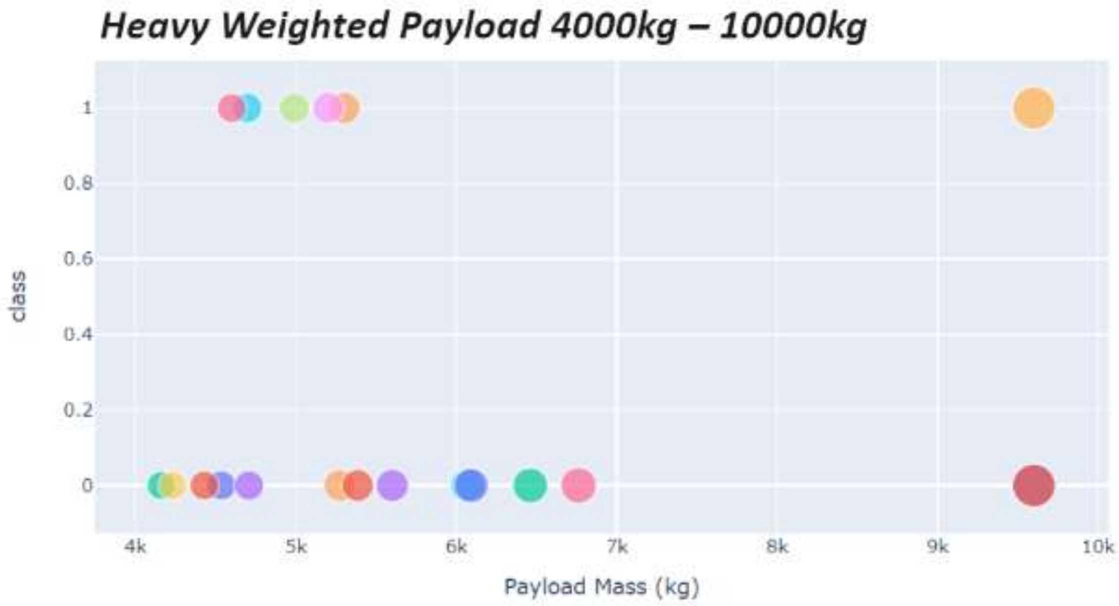
Pie chart showing the Launch site with the highest launch success ratio



KSC LC-39A achieved a 76.9% success rate while getting a 23.1% failure rate

Scatter plot of Payload vs Launch Outcome for all sites, with different payload selected in the range slider





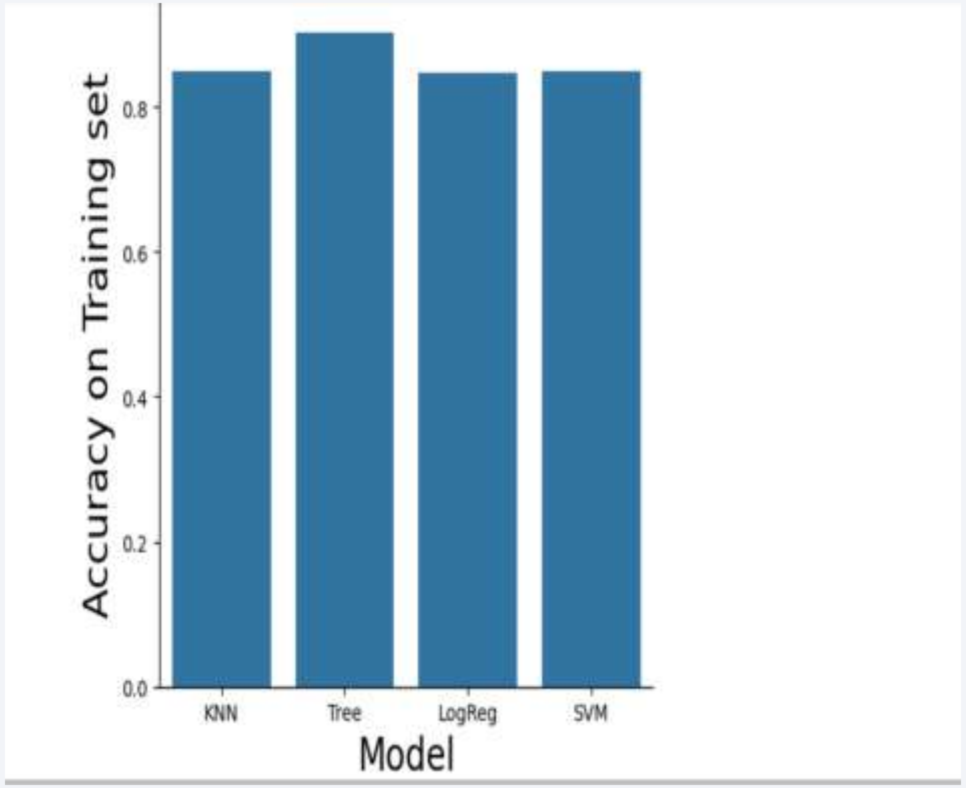
We can see the success rates for low weighted payloads is higher than the heavy weighted payloads

Section 5

Predictive Analysis (Classification)

Classification Accuracy

The decision tree classifier is the model with the highest classification accuracy



Confusion Matrix

• The confusion matrix is same for all the models (LR, SVM, Decision Tree, KNN)

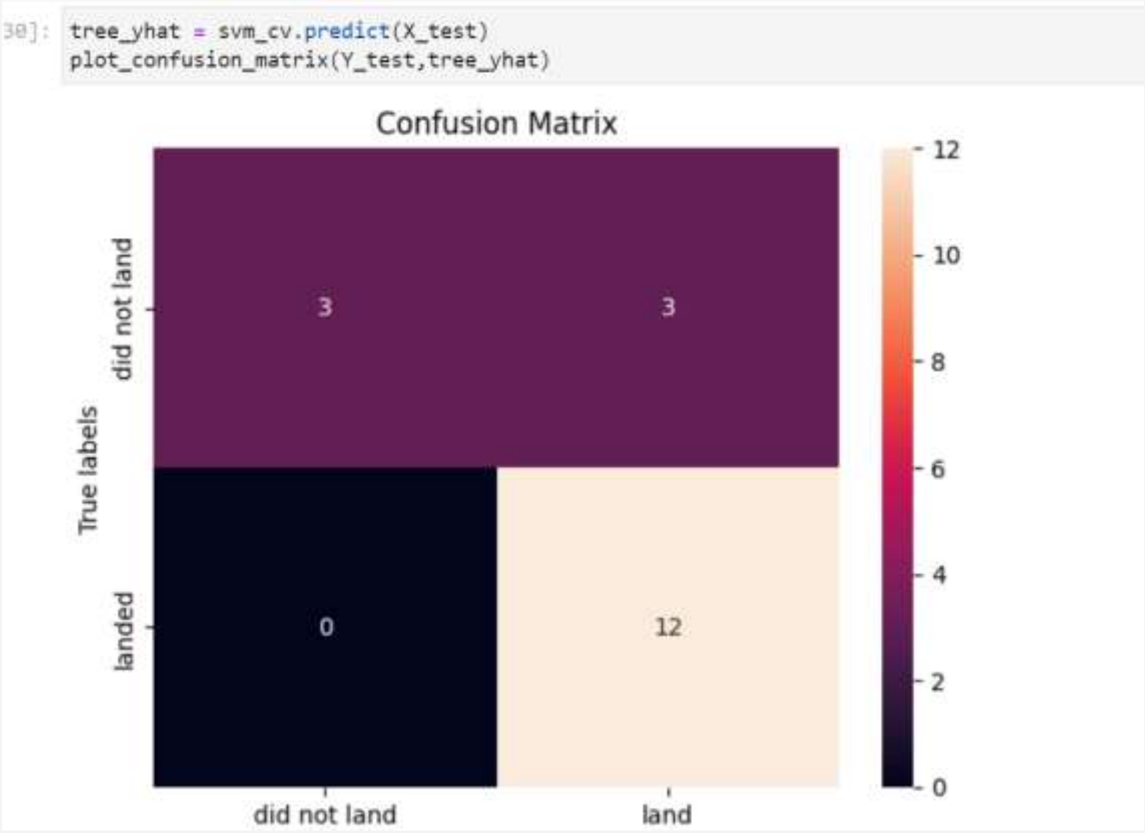
• Per the confusion matrix, the classifier made 18 predictions

• 12 scenarios were predicted Yes for landing, and they did land successfully (True positive)

• 3 scenarios (top left) were predicted No for landing, and they did not land (True negative)

• 3 scenarios (top right) were predicted Yes for landing, but they did not land successfully (False positive)

• Overall, the classifier is correct about 83% of the time ((TP + TN) / Total) with a misclassification or error rate ((FP + FN) / Total) of about 16.5%



Conclusions

- As the numbers of flights increase, the first stage is more likely to land successfully
- Success rates appear go up as Payload increases but there is no clear correlation between Payload mass and success rates
- Launch success rate increased by about 80% from 2013 to 2020
- Launch Site ‘KSC LC-39A’ has the highest launch success rate and Launch Site ‘CCAFS SLC40’ has the lowest launch success rate
- Orbits ES-L1, GEO, HEO, and SSO have the highest launch success rates and orbit GTO the lowest
- Launch sites are located strategically away from the cities and closer to coastline, railroads, and highways
- The best performing Machine Learning Classfication Model is the Decision Tree with an accuracy of about 87.5%. When the models were scored on the test data, the accuracy score was about 83% for all models. More data may be needed to further tune the models and find a potential better fit

Appendix

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

Thank you!