IBM Developer
SKILLS NETWORK

Winning Space Race with Data Science

Nicholas Sauer
8-3-2023

Outline

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

Executive Summary

- Summary of methodologies
  - Collect data through web scraping public information
  - Transform and preprocess data
  - Perform exploratory and data analysis with sql and visuals python's matplotlib library
  - Build machine learning classification model to predict if first stage of Falcon 9 will land successfully
- Summary of all results
  - Data Analysis, Visuals and Interactive dashboard used to display results
  - 4 classification models evaluated and decision tree model found to be optimal method

Introduction

- Project background and context
  - Space X often reuses the first stage of its rocket launches, which provides a key advantage against competitors.
- Problems you want to find answers
  - Develop a reliable method to predict whether or not Space X's Falcon 9 first stage will land successfully – and therefore be reusable – using machine learning and analysis of public information.

Section 1

Methodology

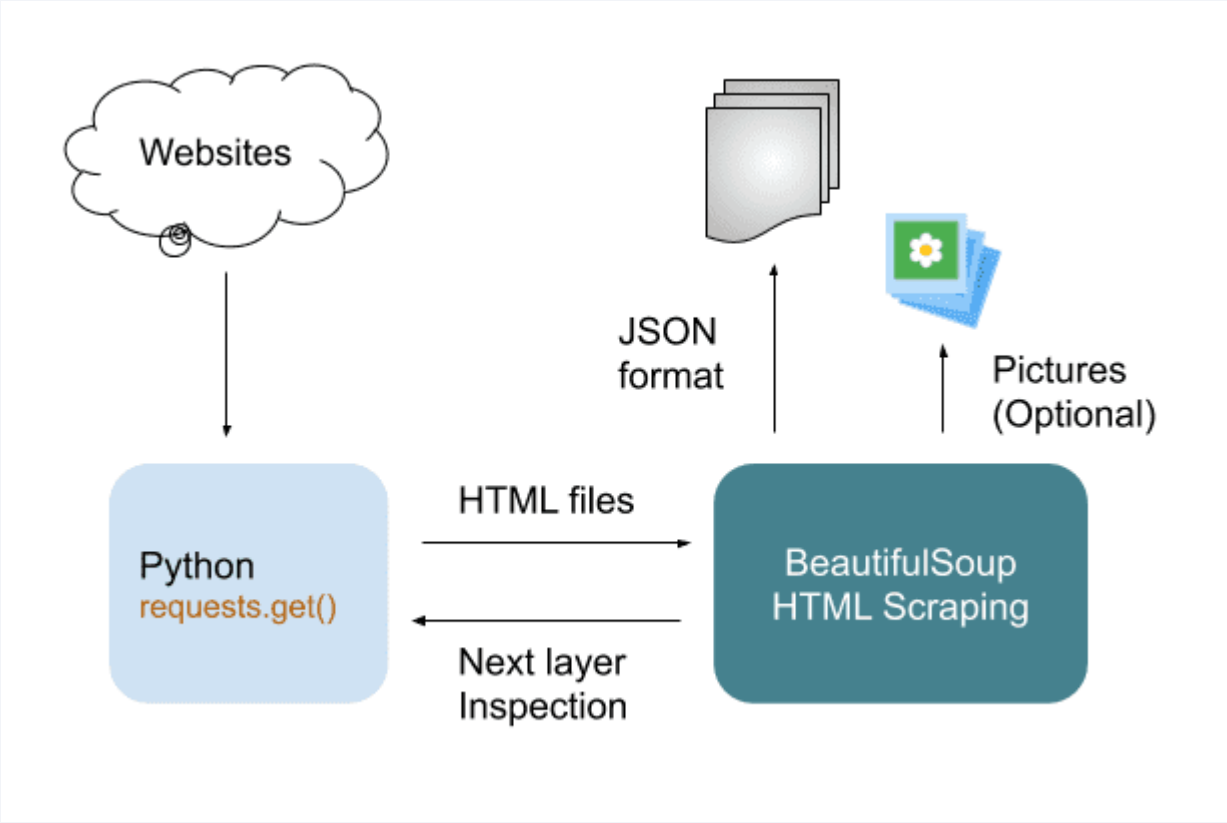
Methodology

Executive Summary

- Data collection methodology:
  - Data collected from SpaceX API & Falcon 9 Launches Records scraped from wikipedia
- Perform data wrangling
  - Outcomes converted into Training Labels (1 =booster landing success; 0 =unsuccessful)
- Perform exploratory data analysis (EDA) using visualization and SQL
- Perform interactive visual analytics using Folium and Plotly Dash
- Perform predictive analysis using classification models
  - Data is Preprocessed, transformed, and split into train/test data sets. Classification models fit using grid search for hyperparameter tuning and evaluated to determine the ideal model.

Data Collection

- Data was collected by webscraping relevant public launch data from wiki pages and via connecting to SpaceX API.



Data Collection – SpaceX API

1.) Request rocket launch data from SpaceX API a URL

2.) decode the response content as a json

3.) Create a dictionary from variables

4.) Load dictionary to Dataframe

5.) Save down as CSV

1.)

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

response = requests.get(spacex_url)
```

2.)

```
# Use json_normalize meethod to convert the json result in
response = requests.get(static_json_url)
data = pd.json_normalize(response.json())
```

3.)

```
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}
```

4.)

```
# Create a data from launch_dict
df = pd.DataFrame(launch_dict)
```

5.)

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

See Github URL

Data Collection - Scraping

1.) Use requests.get() method with the provided static url. Then, create a BeautifulSoup obj from response text content

2.) Identify table with the target data to scrape. Then Identify columns, creating a list of columns

1.)

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

```
# Use BeautifulSoup() to create a BeautifulSoup object fro
soup = BeautifulSoup(data.text, 'html.parser')
```

2.)

```
html_tables = []
for table in soup.find_all('table'):
    html_tables.append(table)
# print(html_tables)
```

Starting from the third table is our target table contains the actual

```
# Let's print the third table and check its content
first_launch_table = html_tables[2]
print(first_launch_table)
```

```
column_names = []
for row in first_launch_table.tbody.findAll('th'):
    col_ = extract_column_from_header(row)
    if col_ != None and len(col_) >0:
        column_names.append(col_)
```

See GitHub URL

Data Collection – Scraping (continued.)

3.) create dictionary from column names and add additional columns

4.) Extract rows from the table to populate dictionary using helper functions.

5.) load the dictionary to a dataframe

3.)

```
launch_dict= dict.fromkeys(column_names)
print(launch_dict)
# Remove an irrelvant column
del launch_dict['Date and time ( )']

for i, key in enumerate(launch_dict):
    # print(i)
    launch_dict[key]=[]

# Added some new columns
launch_dict['Version Booster']=[]
launch_dict['Booster landing']=[]
launch_dict['Date']=[]
launch_dict['Time']=[]

print(launch_dict)
```

4.)

```
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
            # TODO: Append the flight_number into launch_dict with key `Flight No.`
            launch_dict['Flight No.'].append(flight_number)
            #print(flight_number)
            datatimelist=date_time(row[0])


            # Date value
            # TODO: Append the date into launch_dict with key `Date`
            date = datatimelist[0].strip(',')
            launch_dict['Date'].append(date)
            #print(date)

            # Time value
            # TODO: Append the time into launch_dict with key `Time`
            time = datatimelist[1]
            launch_dict['Time'].append(time)
            #print(time)

            # Booster version
```

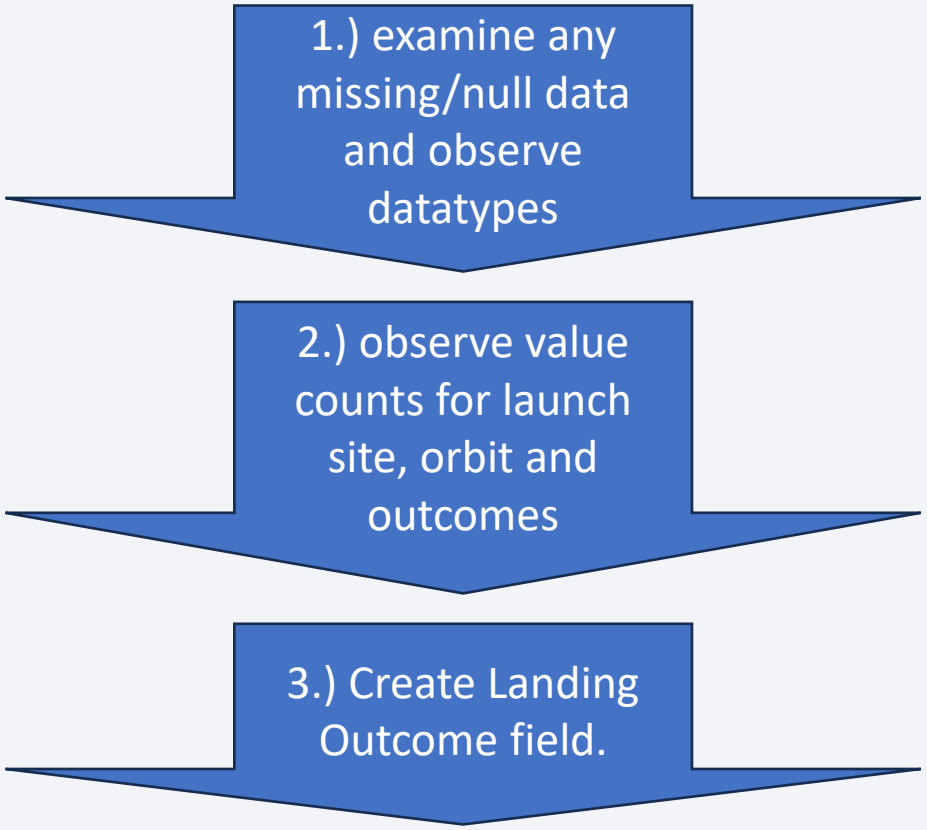
5.)

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

See GitHub URL

Data Wrangling

- Data is Analyzed for patterns and a new column is defined for outcomes converted to training labels 1 and 0. Steps listed below:



1.)

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

```
df.dtypes
```

2.)

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

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

```
GTO      27
ISS      21
VLEO     14
PO        9
LEO       7
SSO       5
MEO       3
ES-L1     1
HEO       1
SO        1
GEO       1
Name: Orbit, dtype: int64
```

```
landing_outcomes = df['Orbit'].value_counts()
landing_outcomes
```

3.)

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

```
"[{'ES-L1', 'MEO', 'ISS', 'SSO', 'PO'}]"
```

```
# landing_class
def landing_class():
    new_list = []
    for  i, outcome in enumerate(df['Orbit']):
        if  str(outcome) in str([bad_outcomes]):
            new = 0
        else:
            new = 1
        print(new)
        new_list.append(new)
        landing_class = new_list
    return landing_class
# landing_class()
```

- See GitHub URL

EDA with Data Visualization

- The below charts were plotted to analyze patterns in the data:

1. **Scatter Charts** – *visualize relationship between two variables*
    1. Payload vs flight number
    2. Flight number vs launch site
    3. Flight number vs orbit
    4. Payload vs orbit
    5. Payload vs launch site
2. **Histogram** – *Depicts frequency of data by group*
    1. Frequency of successful landing by orbit type
3. **Line Charts** – *Depicts trends and changes over time*
    1. Success Rate over time, trending yearly

- See GitHub URL

EDA with SQL

- Dataset was loaded into the corresponding table in a Db2 database. Queries were run to answer the following questions:

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 succesful landing outcome in ground pad was acheived.*
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 records which will display the month names, failure landing_outcomes in drone ship ,booster versions, launch_site for the months 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.*

- See GitHub URL

Build an Interactive Map with Folium

- Utilized the Following Map objects
  - Folium marker and circle object - to Circle and label launch sites on the site map
  - Marker Clusters - to illustrate number of launches/outcomes for launch sites (successes in green and failures in red)
  - Folium Mouse Position - to get the coordinate (Lat, Long) for a mouse over on the map
  - Folium Marker - to display distance between launch site and Melbourne as well as coastline
  - Folium Polyline - to display a line connecting the launch site with the coastline and Melbourne

- See GitHub URL

Build a Dashboard with Plotly Dash

- Interactive dashboard available for to view the following:
  - Success/failures by launch site in a pie chart
  - Correlation between payload & mission outcomes by launch site in a scatter chart.
- Launch site drop down and range slider available for user convenience.
- User can answer the following questions:
  - Which site has the largest successful launches?
  - Which site has the highest launch success rate?
  - Which payload range(s) has the highest launch success rate?
  - Which payload range(s) has the lowest launch success rate?
  - Which F9 Booster version (v1.0, v1.1, FT, B4, B5, etc.) has the highest launch success rate?

- See GitHub URL

Predictive Analysis (Classification)

1.) Standardize data

1.)
```
transform = preprocessing.StandardScaler()
X = transform.fit_transform(X)
```

2.) Split the data into Train and Test sets

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

3.) Fit classification models using grid search for optimizing hyperparameters

3.)
```
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()
```

```
tree_cv=GridSearchCV(DecisionTreeClassifier(),parameters,cv=10)
tree_cv.fit(X_train,Y_train)
```

GridSearchCV
estimator: DecisionTreeClassifier
DecisionTreeClassifier

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

yhat_dec = tree_cv.predict(X_test)
plot_confusion_matrix(Y_test,yhat_dec)
print("DecisionTrees's Accuracy: ", f1_score(Y_test, yhat_dec))
```

4.) Compare models to find determine the best performing method

4.)
```
In [49]: from sklearn.metrics import f1_score

eval_type = ['ml_method_type','Accuracy_Score', 'Best_Score', 'F1_Score']
ml_algo_results= dict.fromkeys(eval_type)
for i, key in enumerate(ml_algo_results):
    # print(i)
    ml_algo_results[key]=[]

ml_algo_results['ml_method_type'] = ['logreg', 'svm', 'decision_tree', 'knn']
ml_algo_results['Accuracy_Score'] = [logreg_cv.score(X_test,Y_test), svm_cv.score(X_test, Y_test),tree_cv.score(X_test, Y_test),k
ml_algo_results['Best_Score']=[logreg_cv.best_score_ , svm_cv.best_score_, tree_cv.best_score_, knn_cv.best_score_]
ml_algo_results['F1_Score']=[f1_score(Y_test, yhat_lrcv), f1_score(Y_test, yhat_svm) ,f1_score(Y_test, yhat_dec) , f1_score(Y_tes

df_results = pd.DataFrame(ml_algo_results)
df_results.set_index(['ml_method_type'], inplace = True)
df_results.head()
```

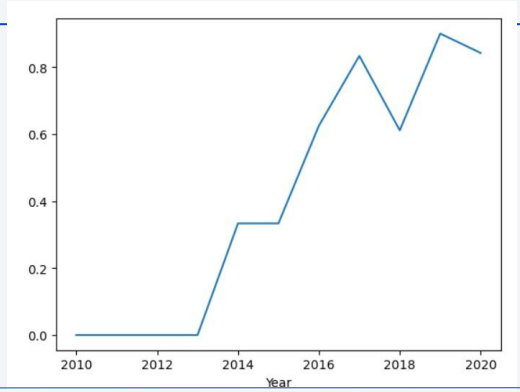
Out[49]:

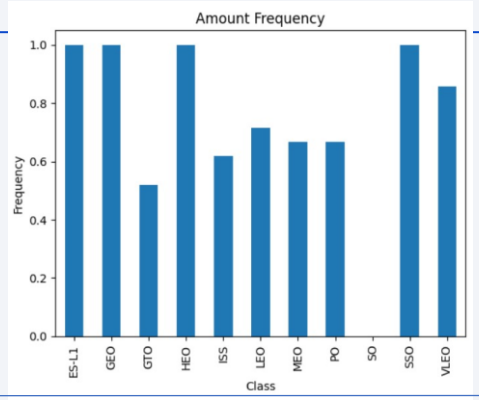
ml_method_type	Accuracy_Score	Best_Score	F1_Score
logreg	0.833333	0.846429	0.888889
svm	0.833333	0.848214	0.888889
decision_tree	0.888889	0.887500	0.916667
knn	0.833333	0.848214	0.888889

- See GitHub URL

Results

Exploratory data analysis results

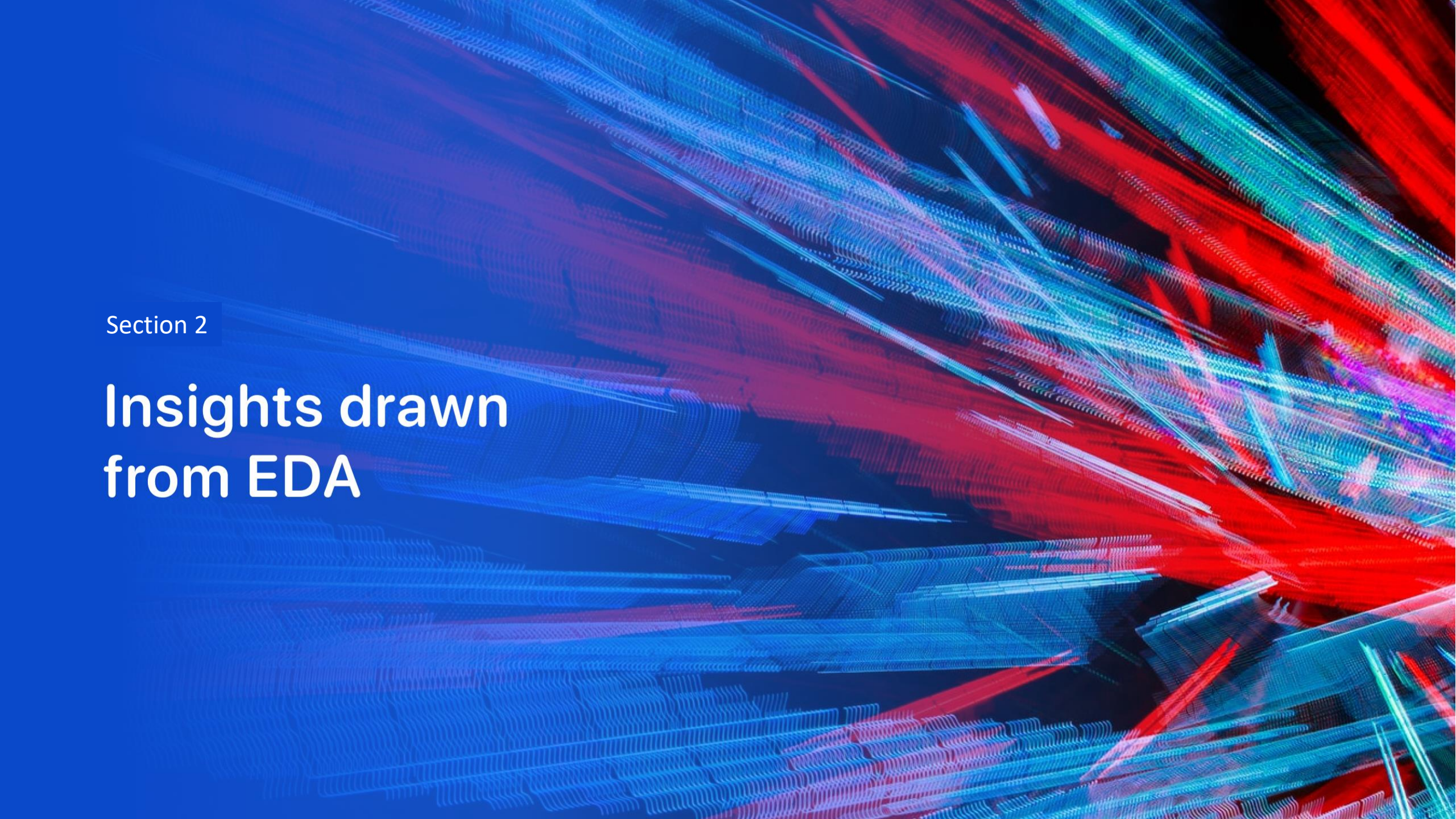




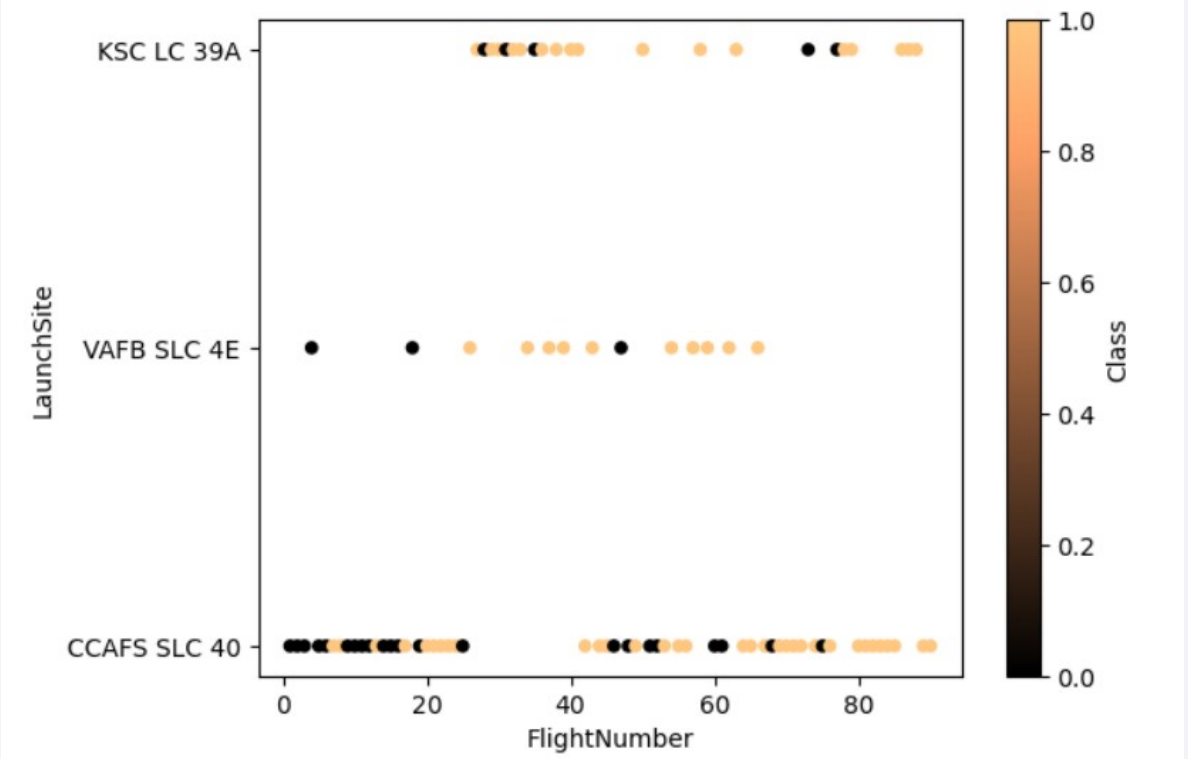
Interactive analytics screenshots

Predictive analysis results

ml_method_type	Accuracy_Score	Best_Score	F1_Score
logreg	0.833333	0.846429	0.888889
svm	0.833333	0.848214	0.888889
decision_tree	0.888889	0.887500	0.916667
knn	0.833333	0.848214	0.888889

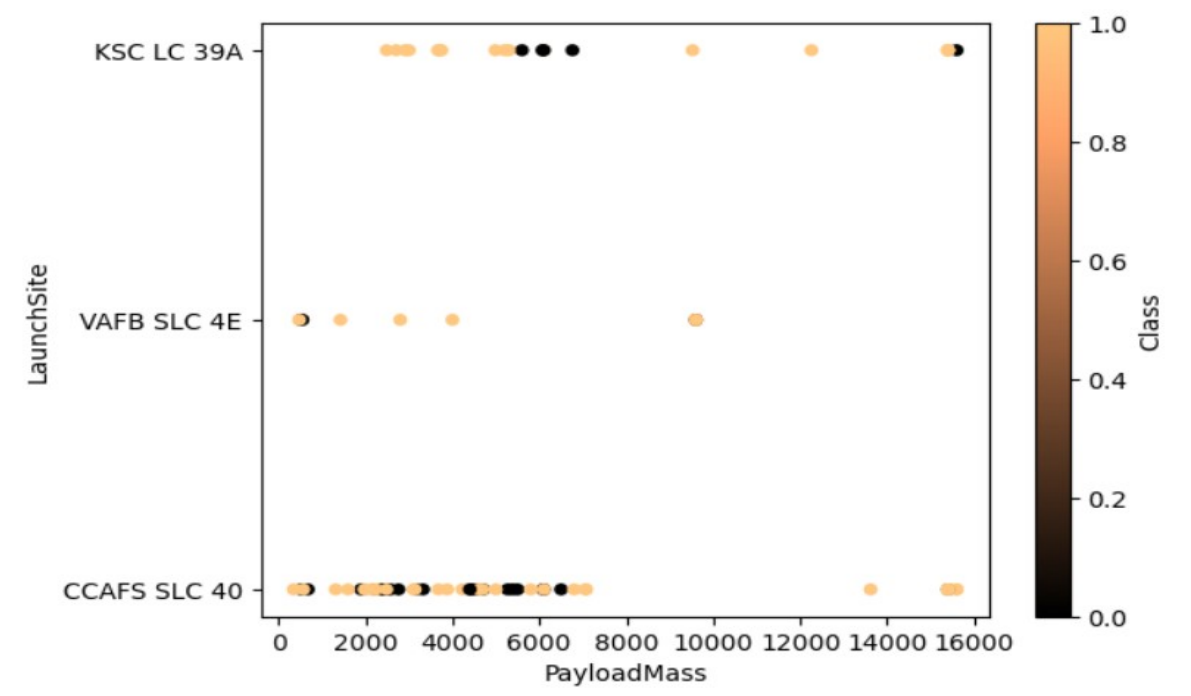

Section 2
Insights drawn from EDA

Flight Number vs. Launch Site



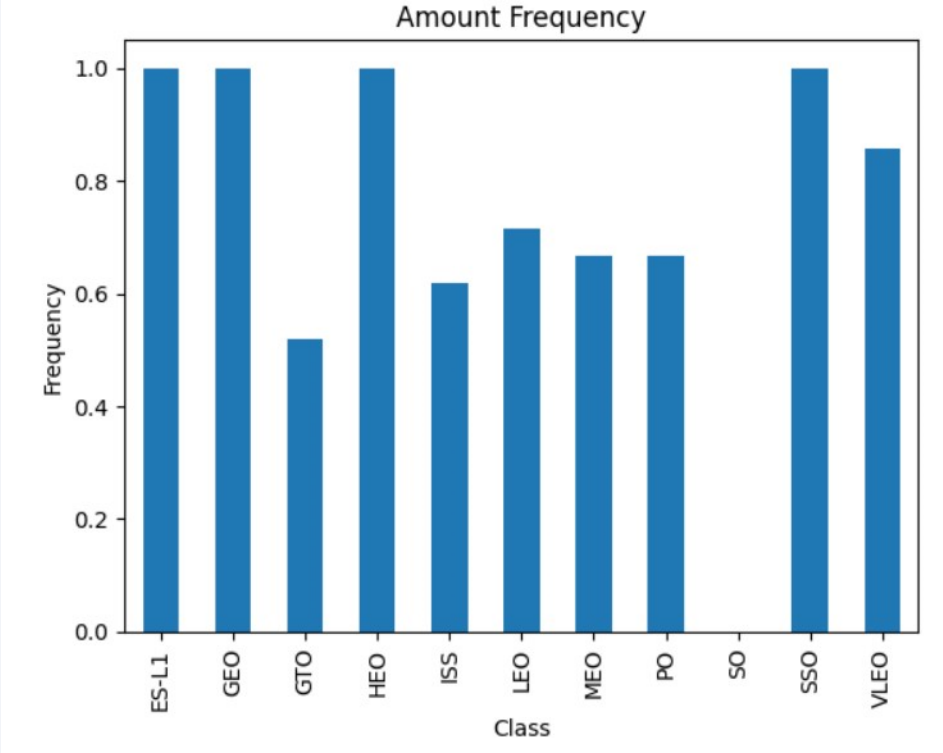
- Chart shows that successful Landing increases with number of flights

Payload vs. Launch Site



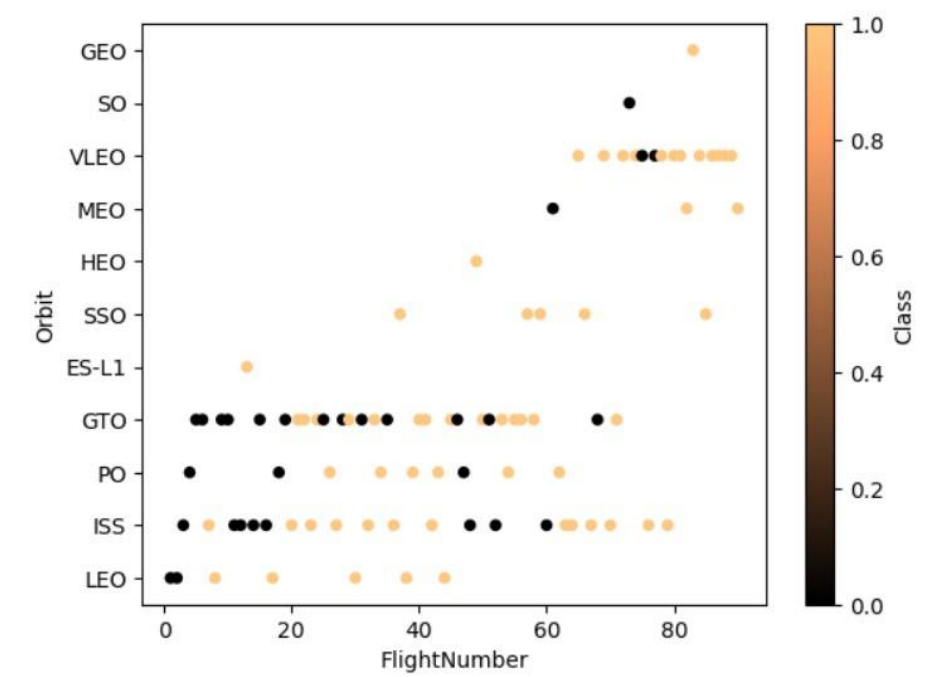
- for the VAFB-SLC launchsite there are no rockets launched for heavypayload mass(greater than 10000).
- For CCAFS SLC launch site, higher payload have a higher success rate

Success Rate vs. Orbit Type



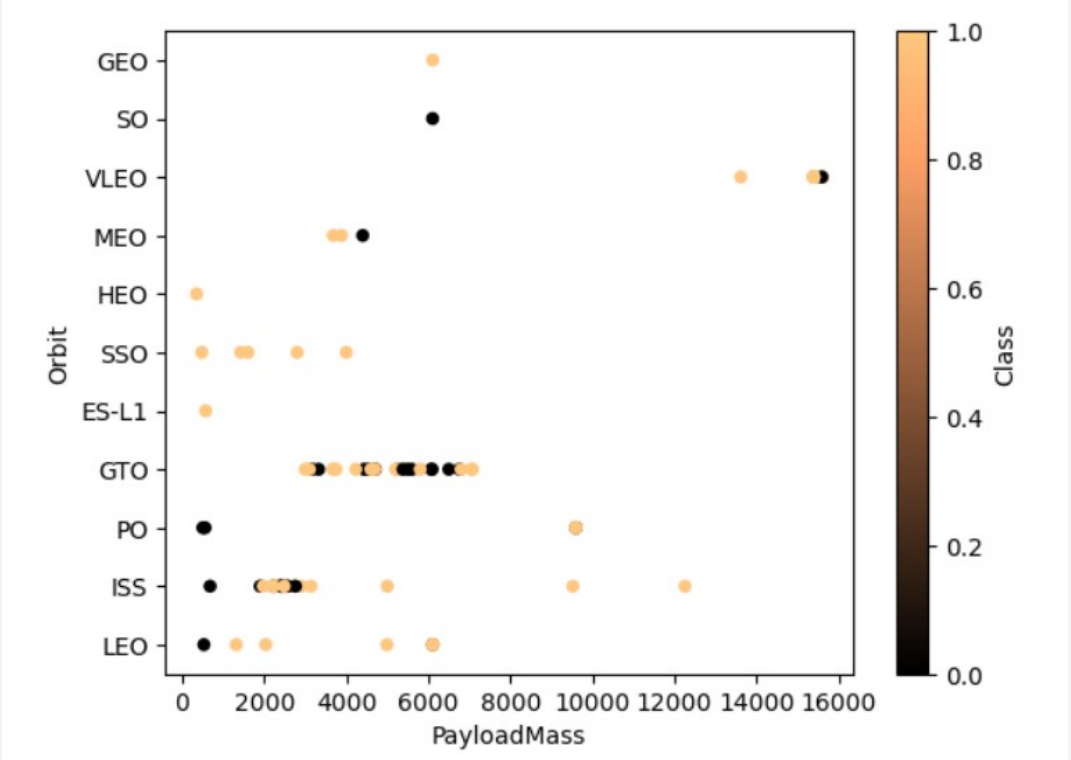
- GTO and SO have lowest success rate while, ES-L1, GEO, HEO and SSO have highest success rate.

Flight Number vs. Orbit Type



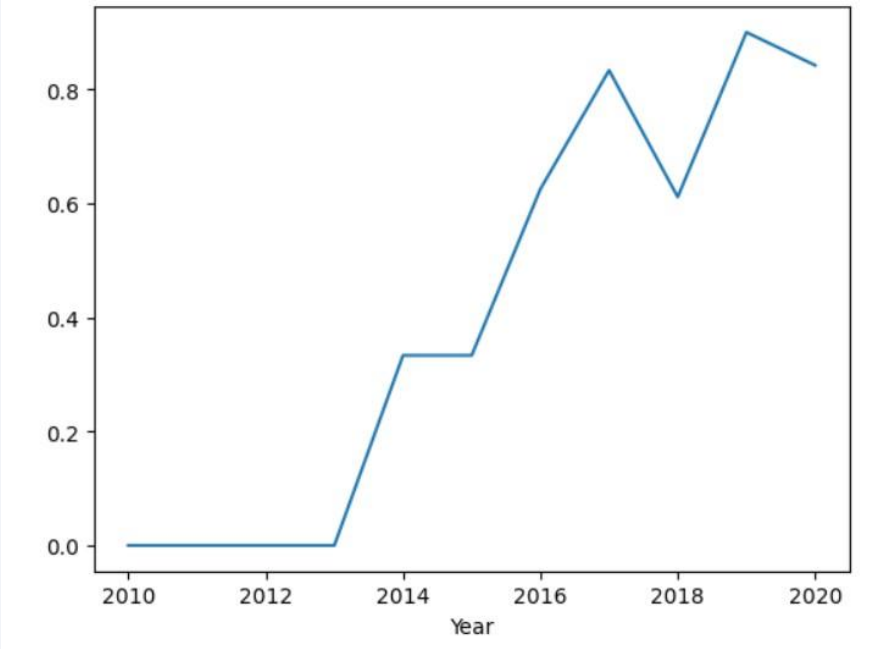
- orbit type is more frequent as number of flights increases
- For LEO orbit the Success appears related to the number of flights; on the other hand, there seems to be no relationship between flight number when in GTO orbit.

Payload vs. Orbit Type



- With heavy payloads the successful landing or positive landing rate are more for Polar,LEO and ISS.
- However for GTO we cannot distinguish this well as both positive landing rate and negative landing(unsuccessful mission) are both there here

Launch Success Yearly Trend



- The success rate since 2013 continued to increase until 2020

All Launch Site Names

- Find the names of the unique launch sites

```
73]: query = """
         SELECT DISTINCT Launch_Site
         from SPACEXTBL"""
     df2 = pd.read_sql_query(query, con)
     df2.head(10)
```

73]:

	Launch_Site
0	CCAFS LC-40
1	VAFB SLC-4E
2	KSC LC-39A
3	CCAFS SLC-40
4	None

Launch Site Names Begin with 'CCA'

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

```
query = """
    SELECT *
    from SPACEXTBL
    WHERE Launch_Site LIKE 'CCA%'
    """
df2 = pd.read_sql_query(query, con)
df2.head()
```

	Date	Time (UTC)	Booster_Version	Launch_Site	Payload	PAYLOAD_MASS__KG_	Orbit	Customer	Mission_Outcome	Landing_Outcome
0	06/04/2010	18:45:00	F9 v1.0 B0003	CCAFS LC-40	Dragon Spacecraft Qualification Unit	0.0	LEO	SpaceX	Success	Failure (parachute)
1	12/08/2010	15:43:00	F9 v1.0 B0004	CCAFS LC-40	Dragon demo flight C1, two CubeSats, barrel of...	0.0	LEO (ISS)	NASA (COTS) NRO	Success	Failure (parachute)
2	22/05/2012	7:44:00	F9 v1.0 B0005	CCAFS LC-40	Dragon demo flight C2	525.0	LEO (ISS)	NASA (COTS)	Success	No attempt
3	10/08/2012	0:35:00	F9 v1.0 B0006	CCAFS LC-40	SpaceX CRS-1	500.0	LEO (ISS)	NASA (CRS)	Success	No attempt
4	03/01/2013	15:10:00	F9 v1.0 B0007	CCAFS LC-40	SpaceX CRS-2	677.0	LEO (ISS)	NASA (CRS)	Success	No attempt

Total Payload Mass

- Calculate the total payload carried by boosters from NASA

```
]: query = """
       SELECT
       Booster_Version,
       sum(PAYLOAD_MASS__KG_) total_payload_mass
       from SPACEXTBL
       WHERE UPPER(Customer) LIKE '%NASA%'
       GROUP BY Booster_Version
       """

   df2 = pd.read_sql_query(query, con)
   df2.head()
```

]:

	Booster_Version	total_payload_mass
0	F9 B4 B1039.2	2647.0
1	F9 B4 B1043.2	6460.0
2	F9 B4 B1039.1	3310.0
3	F9 B4 B1045.1	362.0
4	F9 B4 B1045.2	2697.0

Average Payload Mass by F9 v1.1

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

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

```
query = """
    SELECT
    Booster_Version,
    AVG(PAYLOAD_MASS__KG_) avg_payload_mass
    from SPACEXTBL
    WHERE Booster_Version LIKE '%F9 v1.1%' """
df2 = pd.read_sql_query(query, con)
df2.head()
```

	Booster_Version	avg_payload_mass
0	F9 v1.1 B1003	2534.666667

First Successful Ground Landing Date

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

```
query = """
    SELECT min(Date) first_successful_launch_date
    from SPACEXTBL
    WHERE Mission_Outcome LIKE '%Success%'
    AND Landing_Outcome = 'Success (ground pad)'
    """
df2 = pd.read_sql_query(query, con)
df2.head()
```

	first_successful_launch_date
0	01/08/2018

Successful Drone Ship Landing with Payload between 4000 and 6000

- List the names of boosters which have successfully landed on drone ship and had payload mass greater than 4000 but less than 6000

```
query = """
    SELECT DISTINCT Booster_Version
    from SPACEXTBL
    WHERE
    Landing_Outcome = 'Success (drone ship)'
    AND Mission_Outcome LIKE '%Success%'
    AND PAYLOAD_MASS__KG_ > 4000 AND PAYLOAD_MASS__KG_ < 6000
    """
df2 = pd.read_sql_query(query, con)
df2.head()
```

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

Total Number of Successful and Failure Mission Outcomes

- Calculate the total number of successful and failure mission outcomes

```
query = """
    SELECT
    Mission_Outcome,
    COUNT(*) CT_OUTCOME
    from SPACEXTBL
    GROUP BY Mission_Outcome
    """
df2 = pd.read_sql_query(query, con)
df2.head()
```

	Mission_Outcome	CT_OUTCOME
0	None	898
1	Failure (in flight)	1
2	Success	98
3	Success	1
4	Success (payload status unclear)	1

Boosters Carried Maximum Payload

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

```
query = """
    SELECT
    DISTINCT Booster_Version
    from SPACEXTBL
    WHERE PAYLOAD_MASS__KG_ = (SELECT MAX(PAYLOAD_MASS__KG_) FROM  SPACEXTBL )
    """
df2 = pd.read_sql_query(query, con)
df2.head()
```

	Booster_Version
0	F9 B5 B1048.4
1	F9 B5 B1049.4
2	F9 B5 B1051.3
3	F9 B5 B1056.4
4	F9 B5 B1048.5

2015 Launch Records

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

```
query = """
    SELECT
    ---TRUNC(DATE, 'MM') AS MONTH,
    substr(Date,7,4) YEAR,
    substr(Date, 4, 2) MONTH,
    Mission_Outcome,
    Booster_Version,
    Launch_Site,
    COUNT(*) CT_OUTCOME
    from SPACEXTBL
    WHERE substr(Date,7,4) = '2015'
    GROUP BY
    substr(Date,7,4),
    substr(Date, 4, 2),
    Mission_Outcome,
    Booster_Version,
    Launch_Site
    """
df2 = pd.read_sql_query(query, con)
df2.head()
```

	YEAR	MONTH	Mission_Outcome	Booster_Version	Launch_Site	CT_OUTCOME
0	2015	02	Success	F9 v1.1 B1014	CCAFS LC-40	1
1	2015	04	Success	F9 v1.1 B1015	CCAFS LC-40	1
2	2015	04	Success	F9 v1.1 B1016	CCAFS LC-40	1
3	2015	06	Failure (in flight)	F9 v1.1 B1018	CCAFS LC-40	1
4	2015	10	Success	F9 v1.1 B1012	CCAFS LC-40	1

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

- Rank the count of landing outcomes (such as Failure (drone ship) or Success (ground pad)) between the date 2010-06-04 and 2017-03-20, in descending order

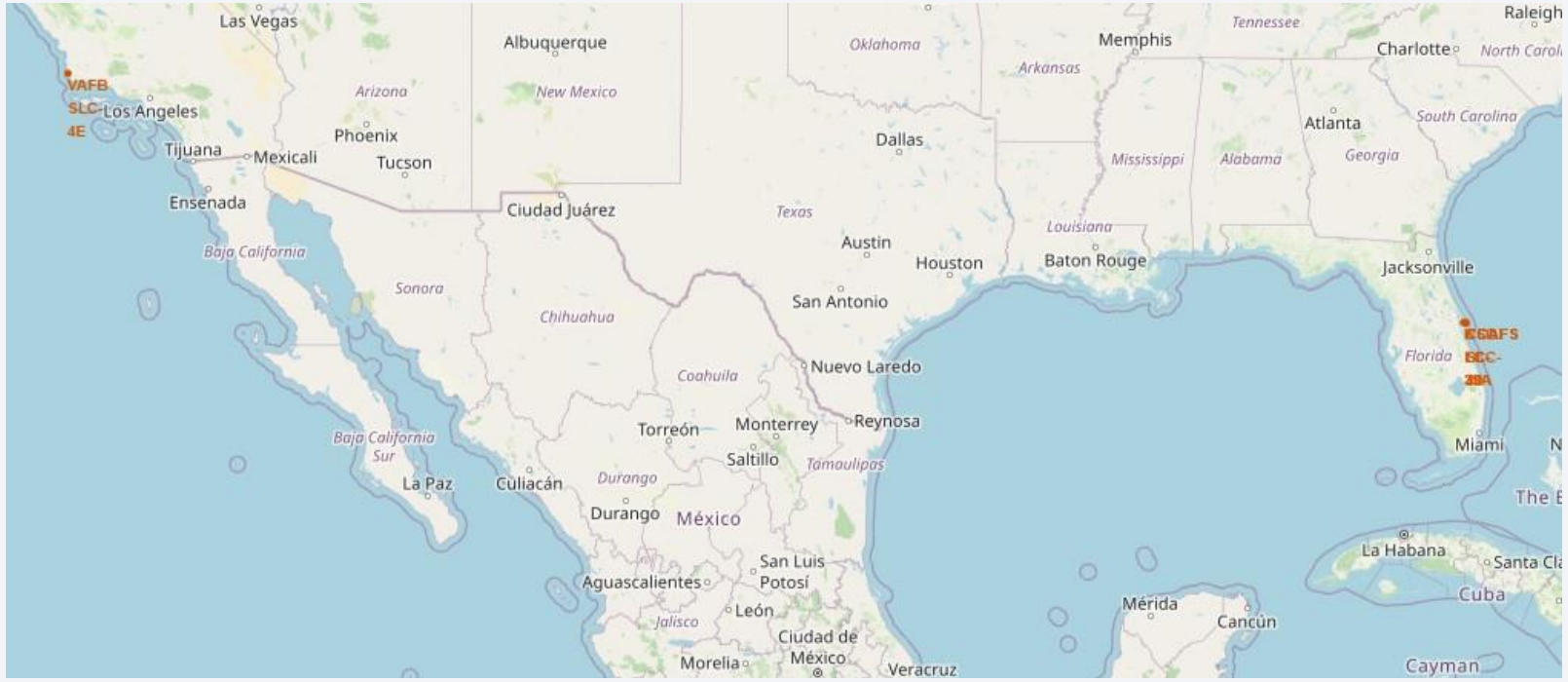
```
query = """
    SELECT
    --substr(Date,7,4) || substr(Date, 4, 2) || substr(Date, 1, 2) ,
    Landing_Outcome,
    COUNT(*) CT_OUTCOME
    from SPACEXTBL
    WHERE substr(Date,7,4) || substr(Date, 4, 2) || substr(Date, 1, 2)
        BETWEEN '20100406' and '20170320'
    GROUP BY  Landing_Outcome
    ORDER BY COUNT(*) DESC
    """
df2 = pd.read_sql_query(query, con)
df2.head()
```

	Landing_Outcome	CT_OUTCOME
0	No attempt	10
1	Success (ground pad)	5
2	Success (drone ship)	5
3	Failure (drone ship)	5
4	Controlled (ocean)	3

Section 3

Launch Sites Proximities Analysis

Falcon 9 Launch Sites

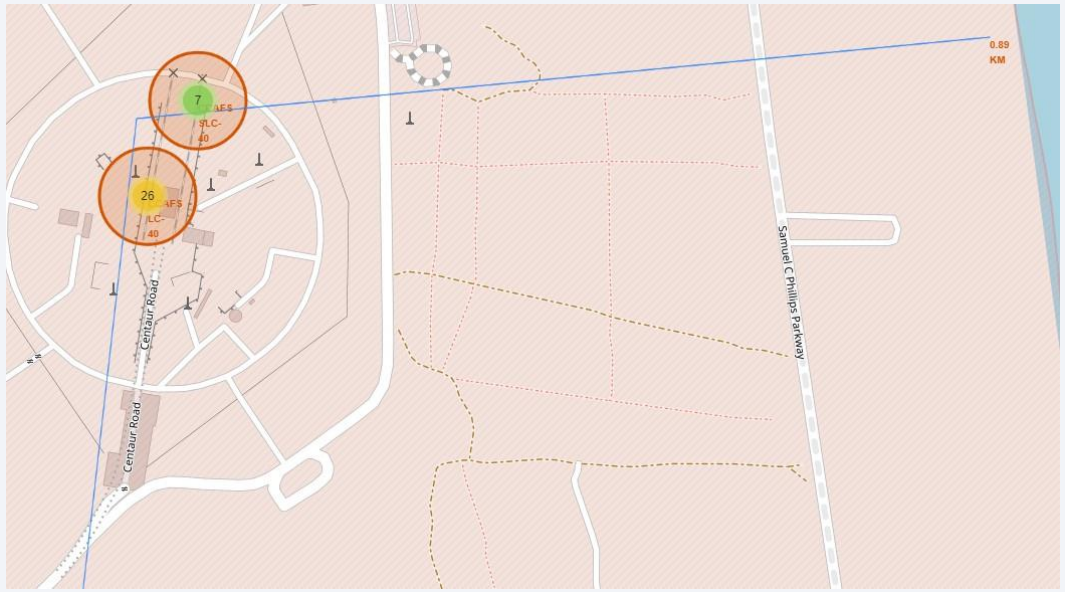


- The map above shows the Falcon 9 launch sites circled and labeled
- We can see there are three launch sites (1 in California and 3 in Florida)

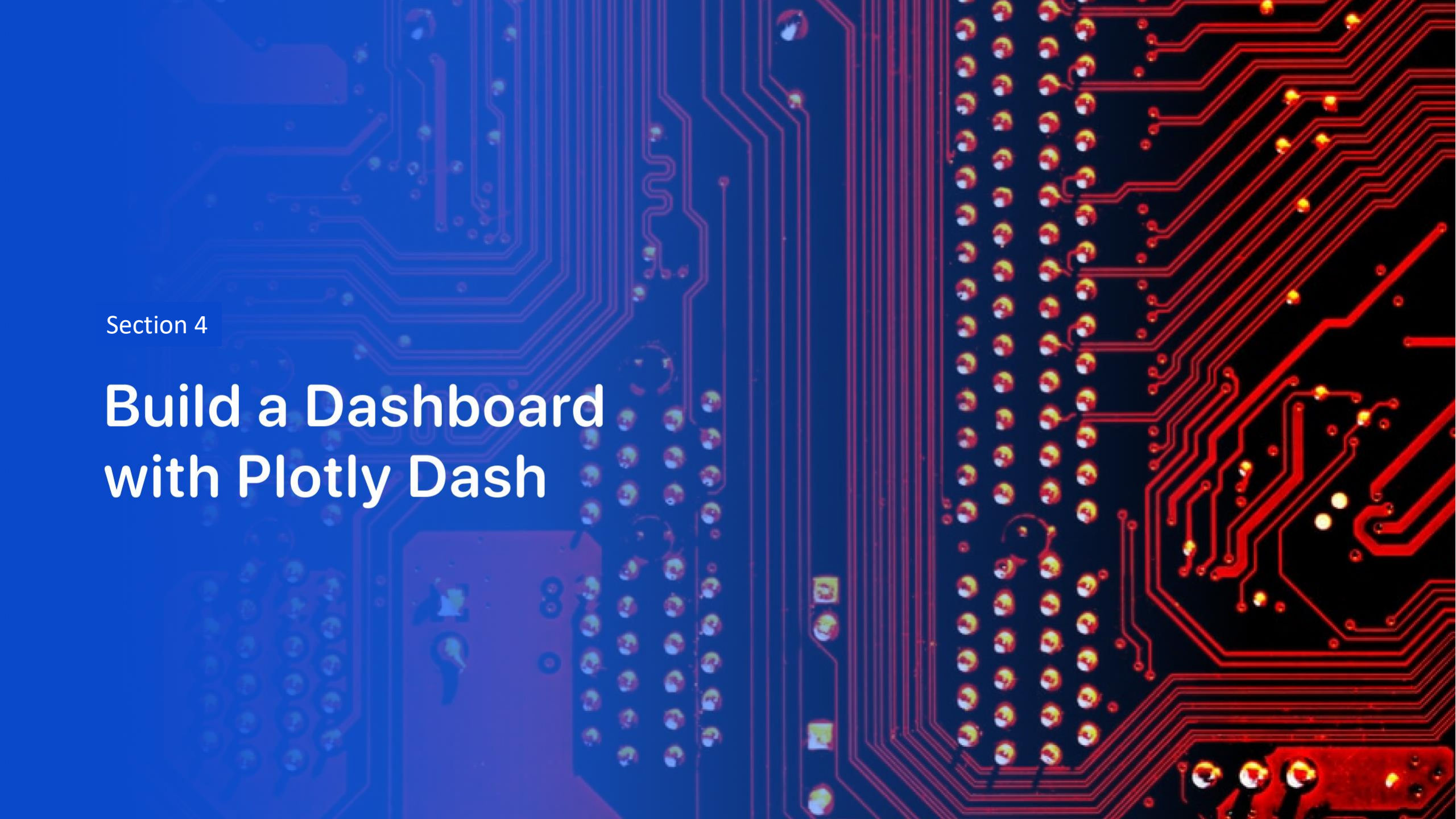
Falcon 9 Launch Success and Failures

- The map above shows the launch sites.
- The number on the launch site shows the total number of Launches.
- Use can zoom to each site to view successes and failures in green and red.

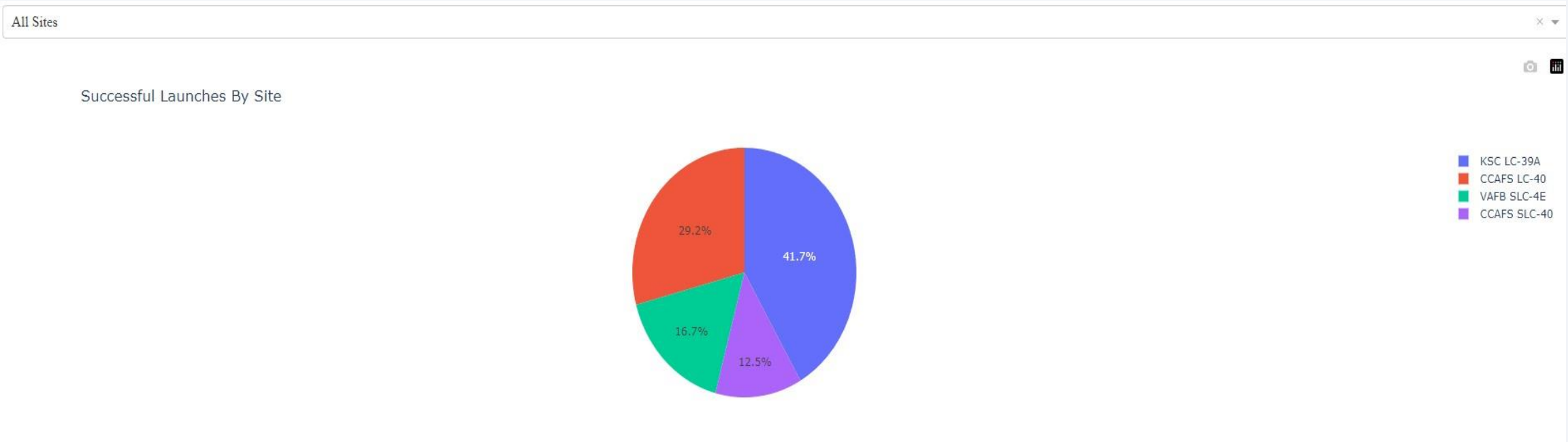
Falcon 9 Launch Site and Proximity Distance



- The illustrations above show the distance between the launch site in Florida and the coastline (.89 km) as well as the city of Melbourne (51.05 km)

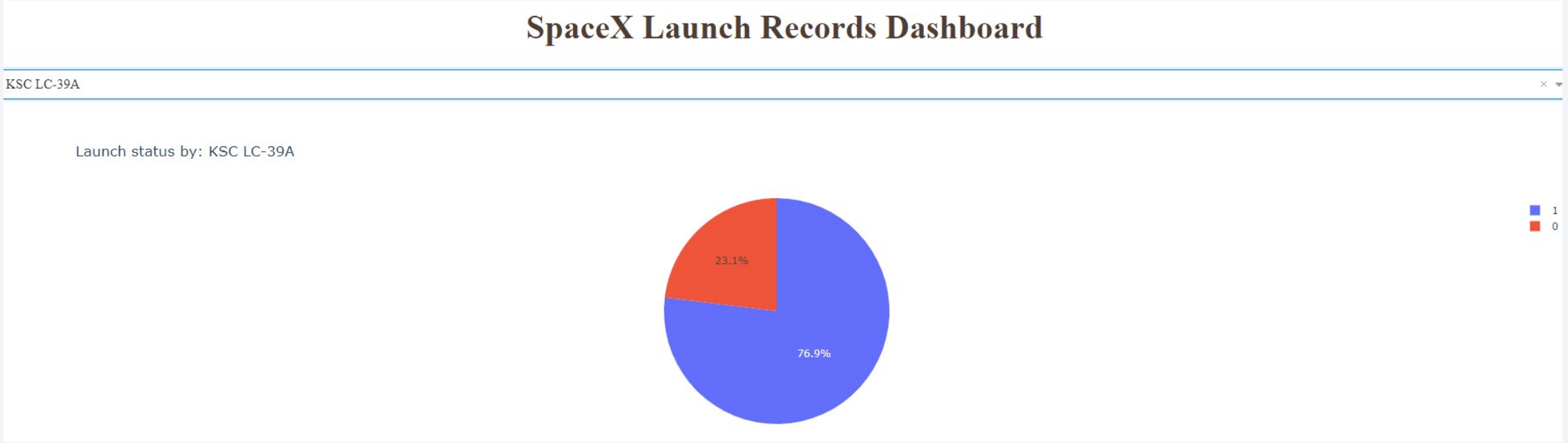

Section 4
Build a Dashboard with Plotly Dash

Successful Launches by Site



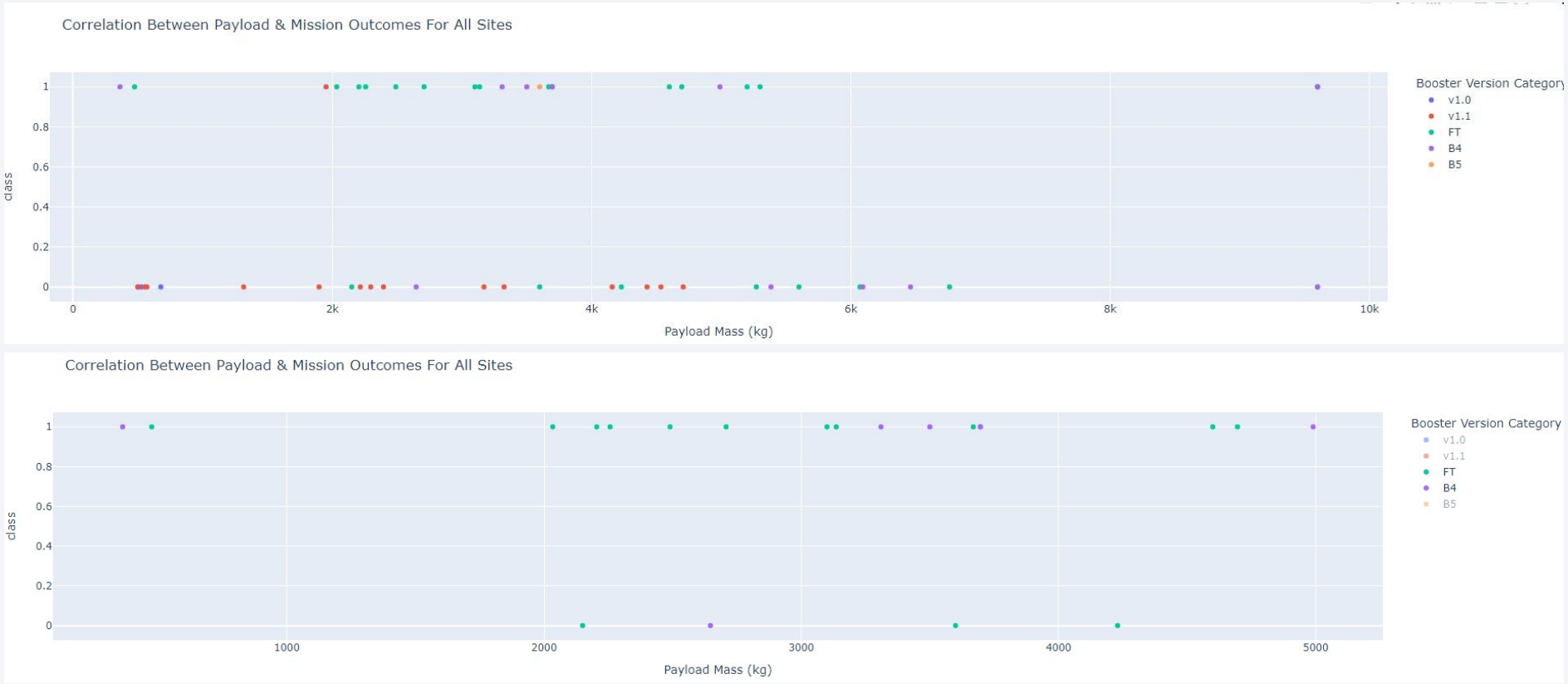
- KSC LC-39A has the highest success rate while CCAFS SLC-40 has the lowest

Successful Launches – KSC LC-39A



- KSC LC-39A has the highest success rate
- Success rate = 76.9%
- Failure rate = 23.1%

Correlation Between Payload & Mission Outcomes

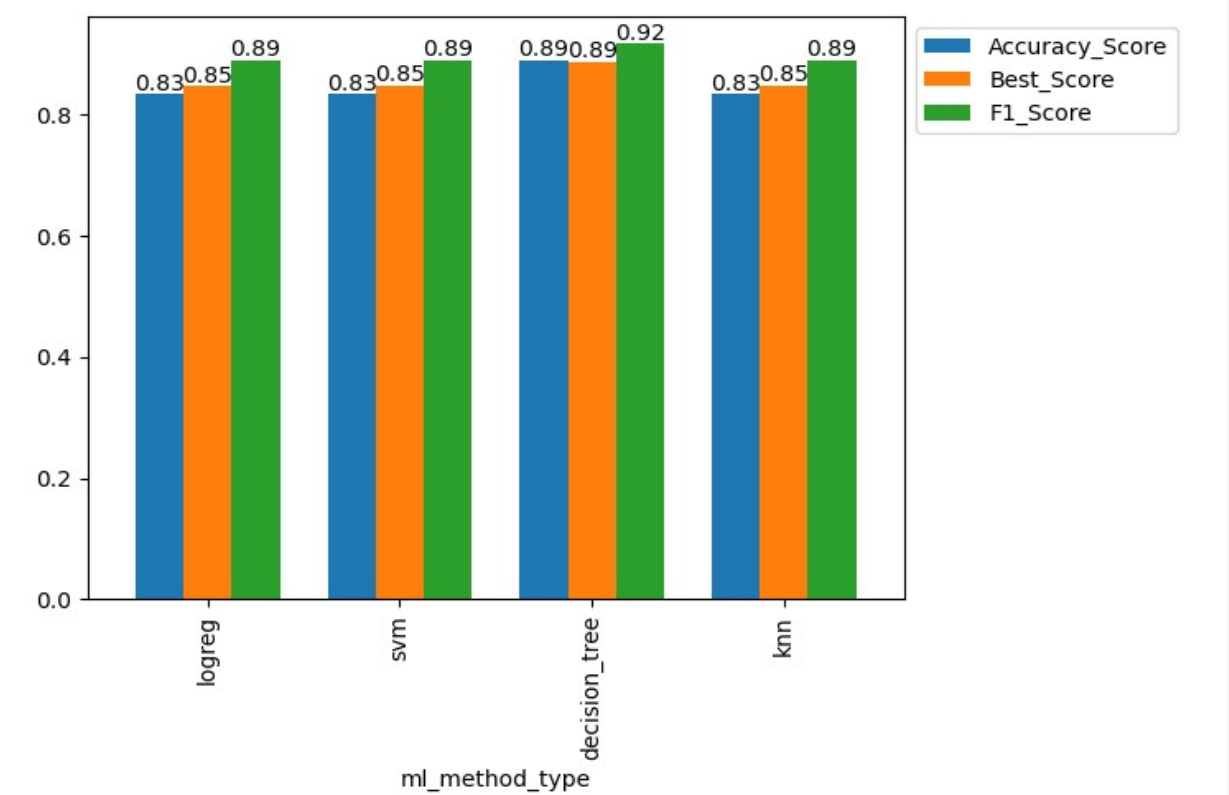


- B4 and FT have a higher success rate with lower payload (<5000 KG)
- V1.0, V1.1, and B5 have the lowest success rate

Section 5

Predictive Analysis (Classification)

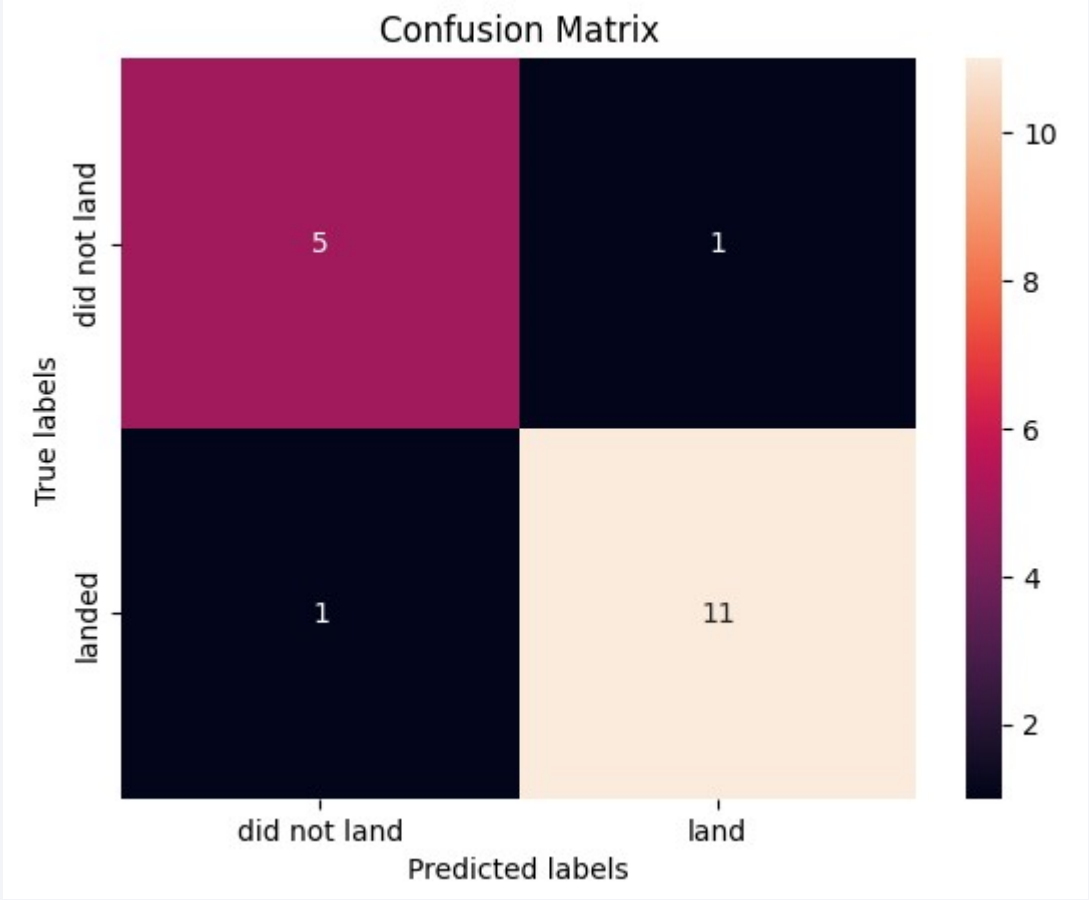
Classification Accuracy



- The above visual shows that Decision Tree has the highest classification accuracy

Confusion Matrix

- The confusion matrix is a visual representation of the accuracy of the Decision Tree Model, showing True Positives, True Negatives, False Positives and False Negatives.
- Accuracy Score is (TP + TN) / (TP + FP + TN + FN) = .888



Conclusions

- You can observe that the success rate since 2013 kept increasing till 2020
- GTO and SO Orbit Types have lowest success rate while, ES-L1, GEO, HEO and SSO have highest success rate.
- KSC LC-39A launch site has the highest success rate (76.9%)
- The best performing machine learning model is the Decision Tree Model as it scores highest on accuracy (0.887), and F-1 score (0.916).

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!