IBM Developer
SKILLS NETWORK

Winning Space Race with Data Science

<Name>
<Date>

Outline

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

Executive Summary

- Summary of methodologies
  - Data Collection using API calls and Web-scrapping.
  - Data Wrangling.
  - Use SQL and visuals to conduct Exploratory Data Analysis.
  - Folium used to make highly interactive map of launch sites.
  - Plotly Dash to build Dashboard for deeper analysis of data.
  - Predictive Model to predict landing success of failure of Falcon 9 First Stage.
- Summary of all results
  - Data Visuals and Interactive Dashboards
  - Predictive Model Results

Introduction

- Project background and context

  Due to successes of space companies such as SpaceX, the Space Industry is experiencing a boom in popularity and investment. The main bottleneck holding startups back from entering this industry is the Cost of Launch to put a payload in orbit.

  SpaceX has had significant success reusing their First Stage. This has allowed spaceX to perform launches for the price of $62,000,000 compared to competitors which are offering launches at around $165,000,000.

- Problems you want to find answers

  Can we tell if the First Stage of a Falcon 9 launch will land successfully?

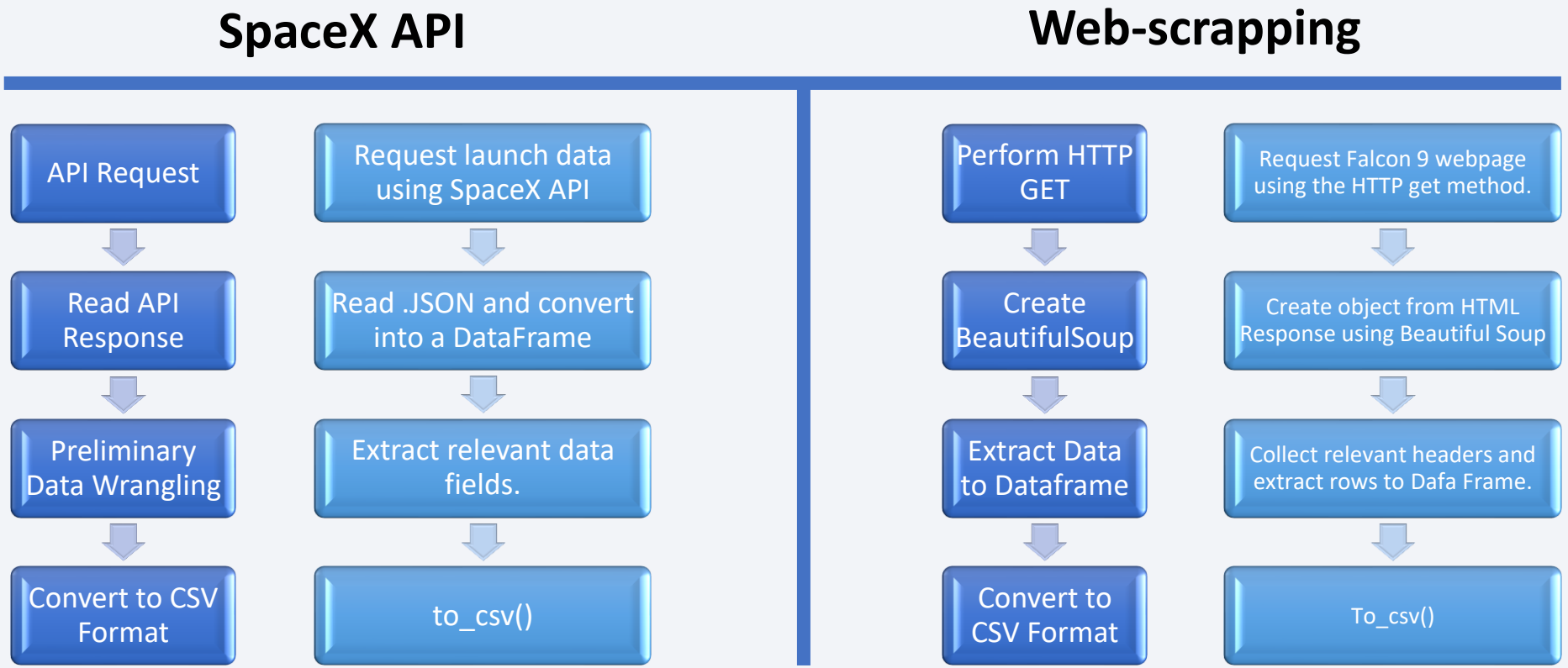
  What are the major variables and their effect on landing outcomes?

Section 1

Methodology

Data Collection

- Data Collection was performed using both official SpaceX provided APIs as well as web scrapping publicly available information on Wikipedia.



Data Collection – SpaceX API

1. API Request and Read Response into Data Frame

2. Declare Global Variables. These will store data returned by helper functions.

```
spacex_url="https://api.spacexdata.com/v4/launches/past"
response = requests.get(spacex_url)
```

```
json = response.json()
data = json_normalize(json)
data = data[['rocket', 'payloads', 'launchpad', 'cores', 'flight_number', 'date_utc']]

data = data[data['cores'].map(len)==1]
data = data[data['payloads'].map(len)==1]

data['cores'] = data['cores'].map(lambda x : x[0])
data['payloads'] = data['payloads'].map(lambda x : x[0])

data['date'] = pd.to_datetime(data['date_utc']).dt.date
data = data[data['date'] <= datetime.date(2020, 11, 13)]

data.head()
```

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

Data Collection – SpaceX API

3. Call helper functions to populate global variables.
4. Construct data using dictionary.
5. Convert dictionary into data frame, filter for Falcon 9 launches, and convert to CSV.

```
getBoosterVersion(data)
getLaunchSite(data)
getPayloadData(data)
getCoreData(data)
```

```
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}
```

```
data = pd.DataFrame(launch_dict)
data_falcon9 = data[data['BoosterVersion'] != 'Falcon 1']
data_falcon9.loc[:,'FlightNumber'] = list(range(1, data_falcon9.shape[0]+1))
data_falcon9
```

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

Data Collection - Scraping

1. Create HTTP GET method and request Falcon 9 launch webpage.
2. Create Beautiful soup object.
3. Find all Tables on the wiki page and extract column names from table headers.
4. Create dictionary with keys from extracted column names.

```
url = "https://en.wikipedia.org/w/index.php?title=List_of_Falcon_9_and_Falcon_Heavy_launches&oldid=1027686922"
response = requests.get(url)

soup = BeautifulSoup(response.text, 'html.parser')

print(soup.prettify())
```

```
tables = soup.find_all('table')
first_launch_table = tables[2]

print(first_launch_table)
```

```
column_names = []

colnames = soup.find_all('th')
for x in range (len(colnames)):
    name2 = extract_column_from_header(colnames[x])
    if (name2 is not None and len(name2) > 3):
        column_names.append(name2)

print(column_names)
```

```
launch_dict= dict.fromkeys(column_names)

del launch_dict['Date and time ( )']

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

Data Collection - Scraping

5. Call helper functions to fill up dictionary with records of launches
6. Convert dictionary to dataframe and then use to_csv() to extract the data as a CSV.

```
def date_time(table_cells):
    return [data_time.strip() for data_time in list(table_cells.strings)][0:2]

def booster_version(table_cells):
    out=''.join([booster_version for i,booster_version in enumerate( table_cells.strings) if i%2==0][0:-1])
    return out
--------------------
def landing_status(table_cells):
    out = [i for i in table_cells.strings][0]
    return out
--------------------
def get_mass(table_cells):
    mass = unicodedata.normalize("NFKD", table_cells.text).strip()
    if mass:
        mass.find("kg")
        new_mass = mass[0:mass.find("kg")+2]
    else:
        new_mass = 0
        return new_mass
--------------------
def extract_column_from_header(row):
    """
    This function returns the landing status from the HTML table cell_
    Input: the  element of a table data cell extracts extra row
    """
    if (row.br):
        row.br.extract()
    if row.a:
        row.a.extract()
    if row.sup:
        row.sup.extract()
```

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

Data Wrangling

1. Load Dataset in as a Data Frame.
2. Find patterns that are relevant to the problem being solved.
3. Create a landing outcome label.

```
count_col2 = df['Orbit'].value_counts()
print(count_col2)
```

```
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['Outcome'].value_counts()
print(landing_outcomes)
```

```
True ASDS      41
None None      19
True RTLS      14
False ASDS      6
True Ocean      5
False Ocean     2
None ASDS       2
False RTLS      1
Name: Outcome, dtype: int64
```

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

EDA with Data Visualization

- Scatter Plot:
  - Payload Mass vs. Flight Number
  - Launch Site vs. Flight Number
  - Payload Mass vs. Launch Site
  - Flight Number vs. Orbit Type
  - Payload vs. Orbit Type
- Bar Graph
  - Orbit Type vs. Success Rate
- Line Chart
  - Year vs. Success Rate

EDA with SQL

- Displayed names of the unique launch sites.
- Displayed 5 records where launch sites began with the string ‘CCA’.
- Displayed total payload mass carries by boosters launched by NASA.
- Displayed average payload mass carried by booster version F9 v1.1
- List the date when the first successful landing outome on 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.
- Listed total number of failure and success outcomes.
- Listed names of booster versions which have caried maximum payload mass.
- Ranked the count of landing outcomes between the date 6/4/2010 and 3/20/2017.

Build an Interactive Map with Folium

- Markers for all launch sites on the map.
  - Folium.circle and folium.marker to highlight launch sites.
- MarkerCluster() to show success (green) and failure (red) for each launch site.
- Calculated distances between launch site and its proximities (Coast, Railroad, Highway, City)
  - MousePosition() to get coordinate for mouse position over map.
  - Folium.Marker() to show distance in KM on the map.
  - Folium.Polyline() to draw line between launch site and each proximity.

Build a Dashboard with Plotly Dash

- Built Plotly Dash application to perform interactive visual analysis on SpaceX launch data.
  - Added Launch site drop-down input component to dashboard to filter data by launch site.
  - Added Pie chart to dashboard to show successes and failures of each launch site.
  - Added Payload range slider to easily select different payload ranges to identify patterns.
  - Added scatter plot to observe payloads correlation with launch outcomes for selected sites.

Predictive Analysis (Classification)

1. Load SpaceX dataset to Data Frame and crease NumPy array from column class in Data.
2. Normalized Data in X and reassigned to variable using transform.
3. Train/Test/Split X and Y into training and test sets.
4. Created and refined models using following classification algorithms.
   - Logistic Regression
   - Grid Search
   - Displaying hyperparameters and accuracy score.

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

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

```
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split( X, Y, test_size=0.2, random_state=2)
print ('Train set:', X_train.shape,  Y_train.shape)
print ('Test set:', X_test.shape,  Y_test.shape)
```

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

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

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

Predictive Analysis (Classification)

5. Find the best performing model.

```
Model_Performance_df = pd.DataFrame({'Algo Type': ['Logistic Regression', 'SVM','Decision Tree','KNN'],
'Accuracy Score': [logreg_cv.best_score_, svm_cv.best_score_, tree_cv.best_score_, knn_cv.best_score_],
'Test Data Accuracy Score': [logreg_cv.score(X_test, Y_test), svm_cv.score(X_test, Y_test),
tree_cv.score(X_test, Y_test), knn_cv.score(X_test, Y_test)]})
```

```
Model_Performance_df.sort_values(['Accuracy Score'], ascending = False, inplace=True)
```

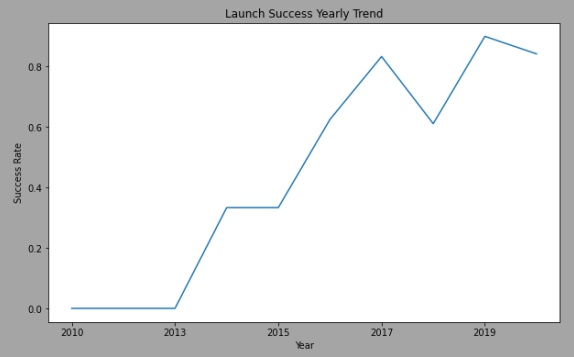
```
Model_Performance_df
```

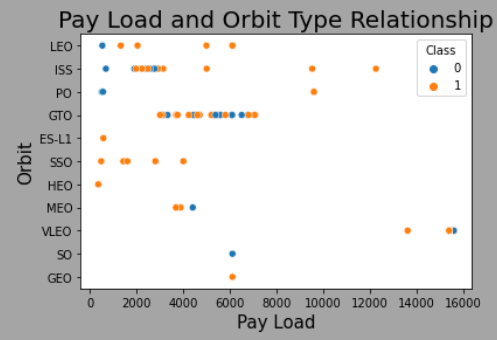
```
i = Model_Performance_df['Accuracy Score'].idxmax()
print('The best performing alogrithm is '+ Model_Performance_df['Algo Type'][i]
+ ' with score ' + str(Model_Performance_df['Accuracy Score'][i]))
```

```
The best performing alogrithm is Decision Tree with score 0.875
```

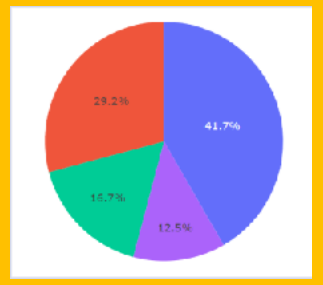
Results

- Exploratory data analysis results



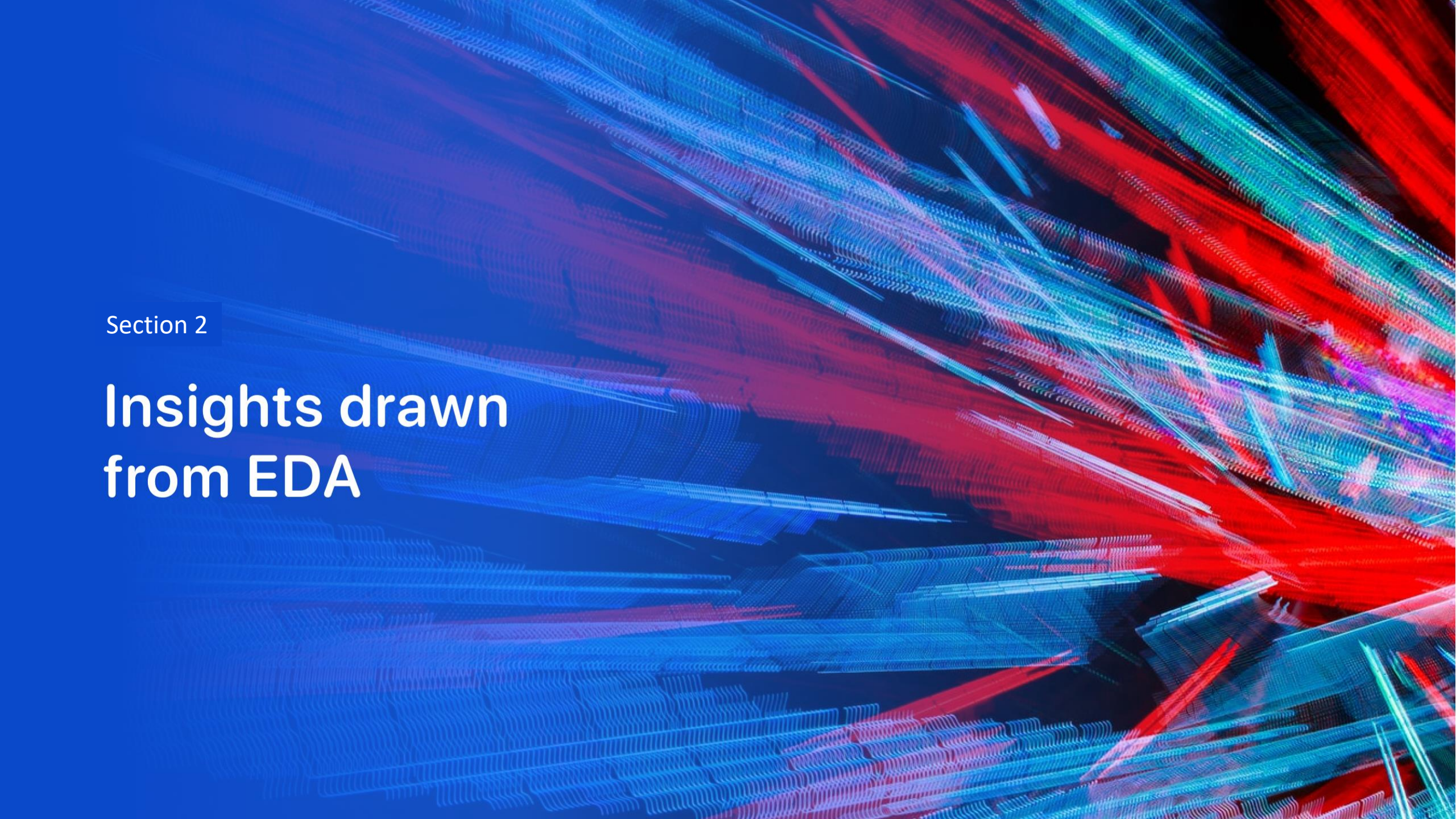


- Interactive analytics demo in screenshots

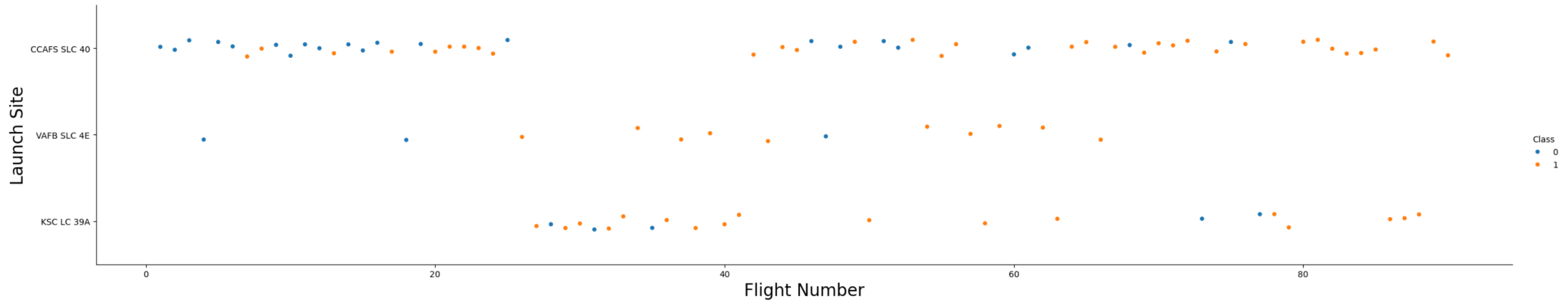


- Predictive analysis results

	Algo Type	Accuracy Score	Test Data Accuracy Score
2	Decision Tree	0.875000	0.833333
3	KNN	0.848214	0.833333
1	SVM	0.848214	0.833333
0	Logistic Regression	0.846429	0.833333


Section 2
Insights drawn from EDA

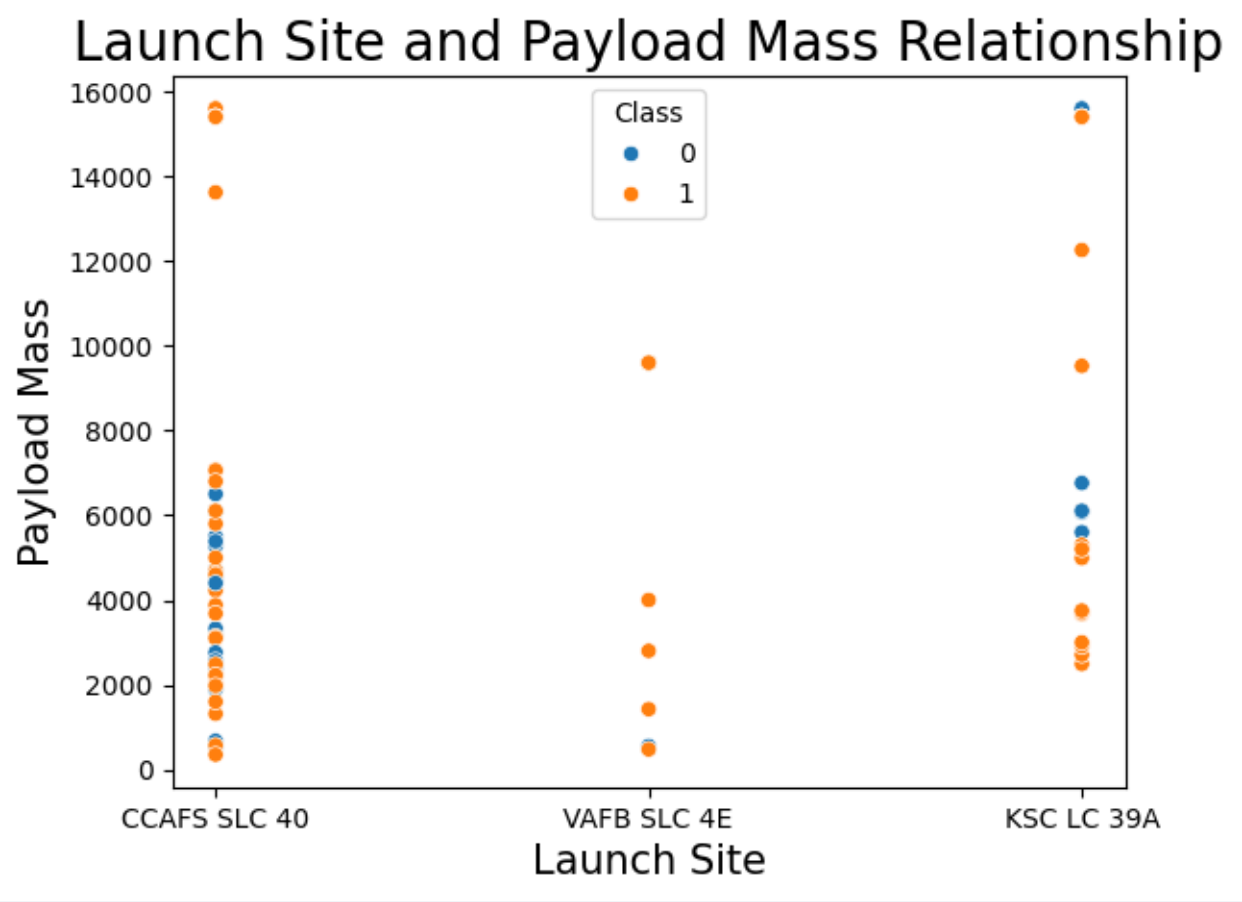
Flight Number vs. Launch Site



- As Flight Number increases so do successes (Class 1 – Orange)
- Launch site KSC LC 39A was not used in any initial launches and it took 26 attempts before a launch was done there.

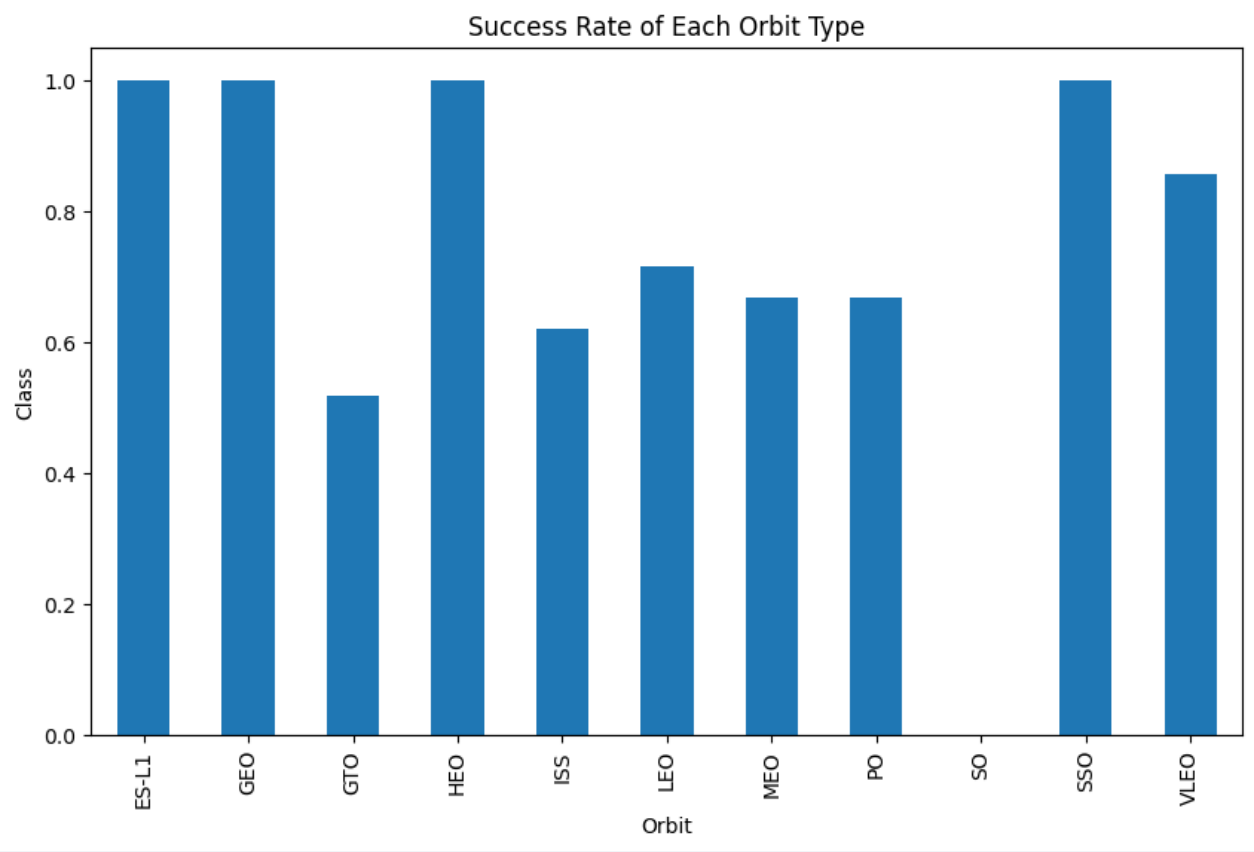
Payload vs. Launch Site

- There are no rockets launched with a payload greater than 10,000 kg at launch site VAFB SLC 4E.
- Payloads above 8500 kg and below 10000 kg tend to be attempted at VAFB SLC 4E launch site.
- There are no clear correlations between the other 2 launch sites and payload mass.



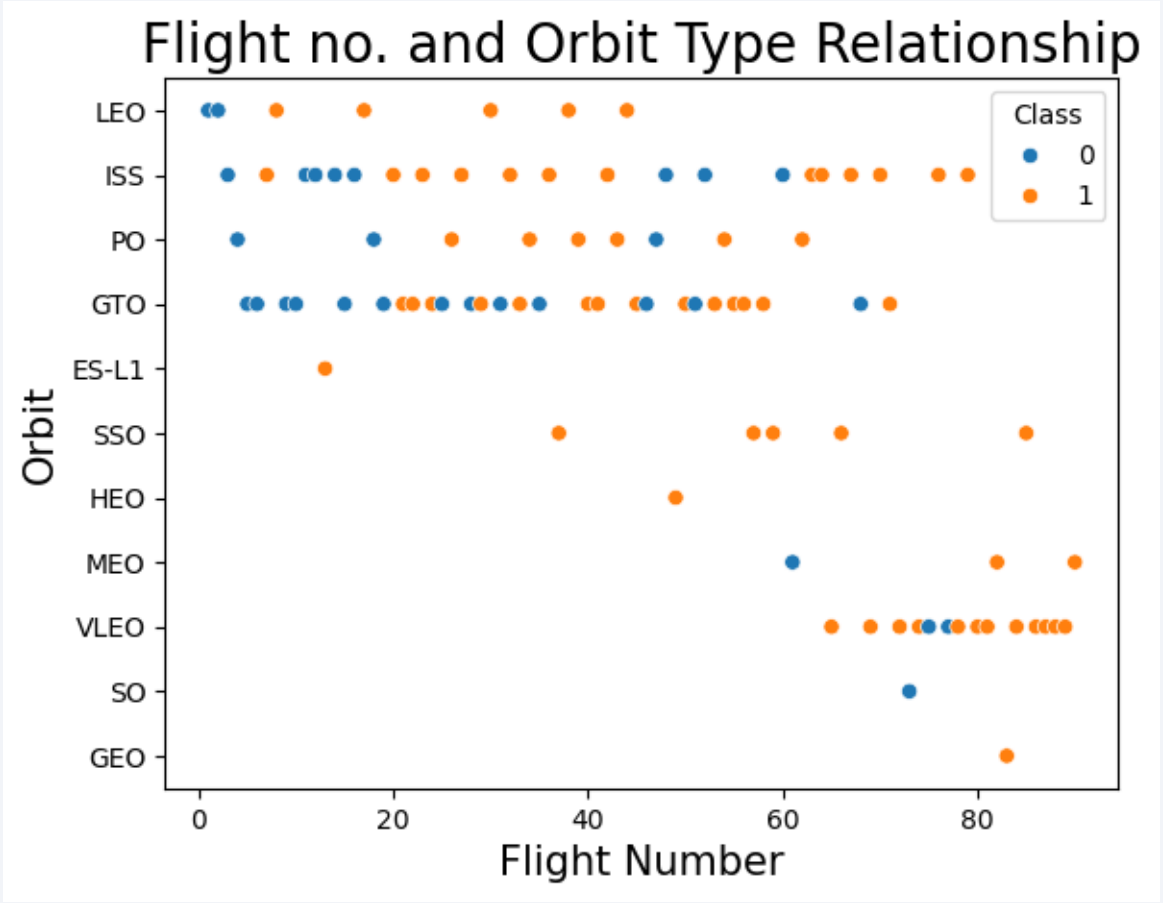
Success Rate vs. Orbit Type

- ES-LI, GEO, HEO, and SSO have highest success rates.
- GTO has lowest.



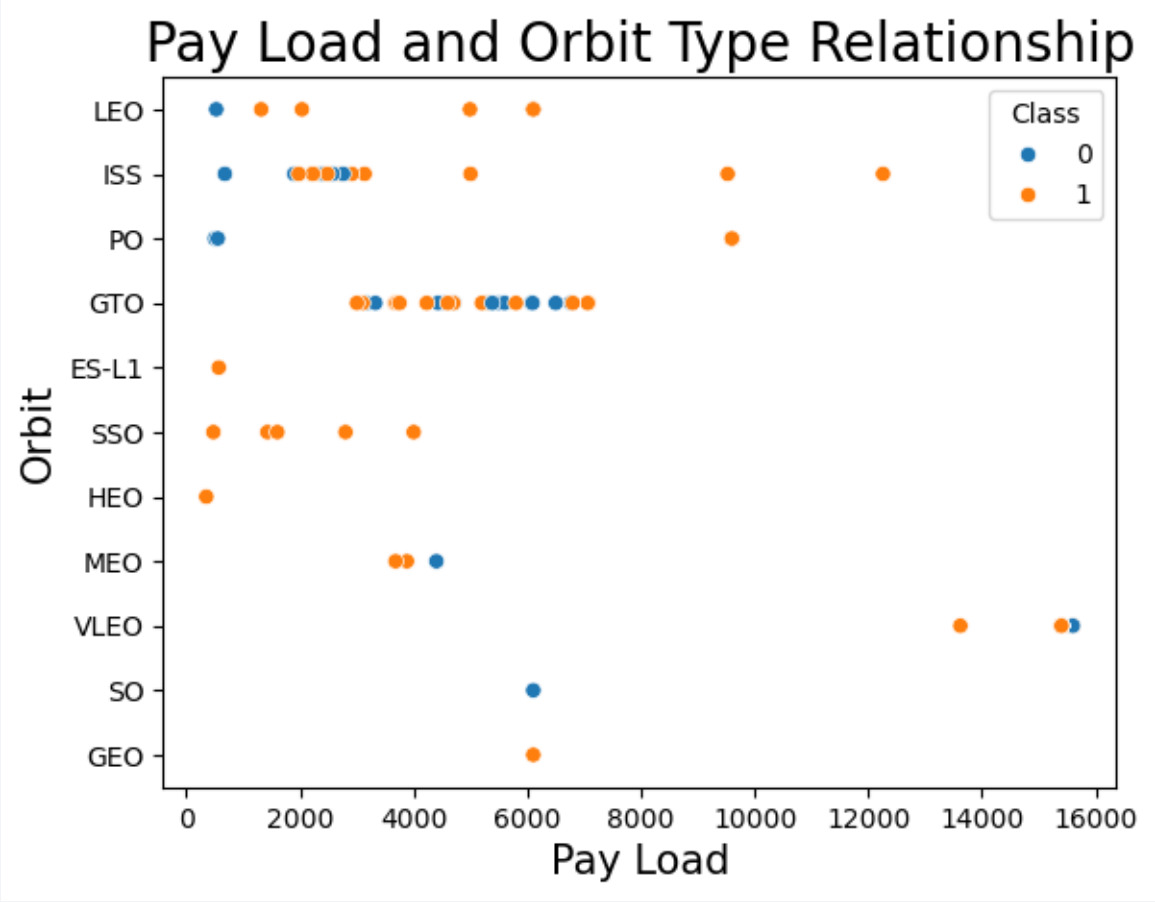
Flight Number vs. Orbit Type

- First successful landing of VLEO doesn't occur until around 63rd flight.
- For most orbits (excluding VLEO) successful landing attempts increase with flight number.



Payload vs. Orbit Type

- Show a scatter point of payload vs. orbit type
- Show the screenshot of the scatter plot with explanations



Launch Success Yearly Trend

- Show a line chart of yearly average success rate

- Show the screenshot of the scatter plot with explanations

All Launch Site Names

- Query

```
select distinct Launch_Site from spacextbl
```

- Distinct returns only unique values from the Launch_Site column.

- Result

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

Launch Site Names Begin with 'CCA'

- Query:

```
select * from spacextbl where Launch_Site LIKE 'CCA%' limit 5;
```

- Like ‘CCA%’ returns any results with CCA in the string. Limit 5 ensures that only 5 results are returned.
- Result:

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	Dragon Spacecraft Qualification Unit	0	LEO	SpaceX	Success	Failure (parachute)
2010-12-08	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	Failure (parachute)
2012-05-22	7:44:00	F9 v1.0 B0005	CCAFS LC-40	Dragon demo flight C2	525	LEO (ISS)	NASA (COTS)	Success	No attempt
2012-10-08	0:35:00	F9 v1.0 B0006	CCAFS LC-40	SpaceX CRS-1	500	LEO (ISS)	NASA (CRS)	Success	No attempt
2013-03-01	15:10:00	F9 v1.0 B0007	CCAFS LC-40	SpaceX CRS-2	677	LEO (ISS)	NASA (CRS)	Success	No attempt

Total Payload Mass

- Query:

```
select sum(PAYLOAD_MASS__KG_) from spacextbl where Customer = 'NASA (CRS)'
```

- *Where Customer ='NASA (CRS)'* added to the end of the query selects only rows that have NASA as a customer.
- *Sum(PAYLOAD_MASS_KG_)* sums up all values in this column that meet the criteria specified by the where operation.
- Results:

sum(PAYLOAD_MASS__KG_)
45596

Average Payload Mass by F9 v1.1

- **QUERY:**

```
select avg(PAYLOAD_MASS__KG_) from spacextbl where Booster_Version LIKE 'F9 v1.1';
```

- *WHERE Booster_Version LIKE 'F9 v1.1'* added at the end returns only values in the Booster_Version column that have the string F9 v1.1 in them.
- *Avg(PAYLOAD_MASS_KG_)* averages every value that meets the criteria specified using the where operation.
- **RESULTS:**

avg(PAYLOAD_MASS__KG_)
2928.4

First Successful Ground Landing Date

- **QUERY:**

```
select min(Date) as min_date from spacextbl where Landing_Outcome = 'Success (ground pad)';
```

- ***Min(Date) as min_date*** searches for the lowest value in the date column and returns it under the header min_date.

- **RESULT:**

min_date
2015-12-22

Successful Drone Ship Landing with Payload between 4000 and 6000

- QUERY:

```
select Booster_Version from spacextbl where (PAYLOAD_MASS__KG_> 4000 and PAYLOAD_MASS__KG_ < 6000)
and (Landing_Outcome = 'Success (drone ship)');
```

- ***And*** operation allows the querying of data based on several conditions. In this case that ***(PAYLOAD_MASS_KG_ > 4000 and PAYLOAD_MASS_KG_ < 6000)*** as well as the ***Landing_Outcome*** column having a value of ***Success (drone ship)***

- **RESULTS:**

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

Total Number of Successful and Failure Mission Outcomes

- **QUERY:**

```
select Mission_Outcome, count(Mission_Outcome) as counts from spacextbl group by Mission_Outcome;
```

- ***Group by Mission_Outcome*** consolidates all of the data by distinct values in the Mission_Outcome column
- ***Count(Mission_Outcome)*** as counts returns a new column of the quantity of each mission outcome type.
- **RESULTS:**

Mission_Outcome	counts
Failure (in flight)	1
Success	98
Success	1
Success (payload status unclear)	1

Boosters Carried Maximum Payload

- **QUERY:**

```
select Booster_Version, PAYLOAD_MASS__KG_ from spacextbl where PAYLOAD_MASS__KG_ = (select max(PAYLOAD_MASS__KG_) from spacextbl);
```

- **RESULTS:**

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

- *Where PAYLOAD_MASS_KG = Select max(PAYLOAD_MASS_KG_)* returns only rows where the payload mass is equal to the maximum *PAYLOAD_MASS_KG_* value present within the dataset.
- In this case the maximum payload mass carried by any Booster Version is 15600 KG

2015 Launch Records

- **QUERY:**

```
select Landing_Outcome, Booster_Version, Launch_Site from spacextbl where Landing_Outcome = 'Failure (drone ship)' and year(Date) = '2015'
```

- **RESULT:**

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

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

- **QUERY:**

```
select Landing_Outcome, count(*) as LandingCounts from spacextbl where Date between '2010-06-04' and '2017-03-20'
group by Landing_Outcome
order by count(*) desc;
```

- ***Order by count(*) desc;*** arranges all of the query results based on tallying up the Landing Outcomes and displaying them in descending order of quantity.
- **RESULTS:**

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

Section 3

Launch Sites Proximities Analysis

SpaceX Falcon 9 – Launch Sites Map

- SpaceX has 4 total launch sites.
- 1 is located in California, US
  - VAFB SLC-4E
- 3 are located in Florida, US
  - KSC LC-39A
  - CCAFS LC40
  - CCAFS SLC-40

<Folium Map Screenshot 2>

<Folium Map Screenshot 3>

- Replace <Folium map screenshot 3> title with an appropriate title
- Explore the generated folium map and show the screenshot of a selected launch site to its proximities such as railway, highway, coastline, with distance calculated and displayed
- Explain the important elements and findings on the screenshot

Section 5

Predictive Analysis (Classification)

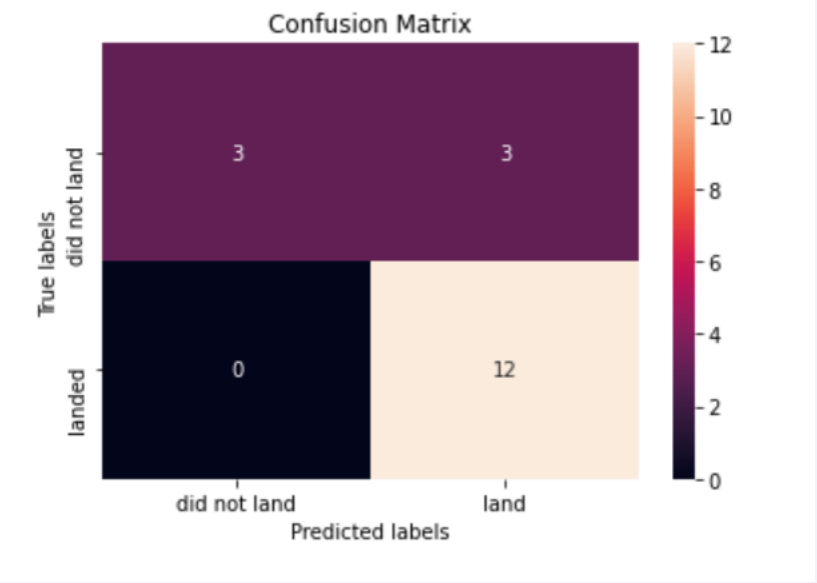
Classification Accuracy

- By looking at the accuracy scores it is evident that Decision tree was the most accurate predictive model.
- There was no difference between testing accuracy on any model. (all = 0.833)

	Algo Type	Accuracy Score	Test Data Accuracy Score
2	Decision Tree	0.875000	0.833333
3	KNN	0.848214	0.833333
1	SVM	0.848214	0.833333
0	Logistic Regression	0.846429	0.833333

Confusion Matrix

- 18 Predictions made.
- All models had same confusion matrix.
- 12 predicted yes for landing.
- 3 predicted no for landing but did successfully land.
- 3 predicted yes for landing that did not land successfully.
- Error rate = 16.5%



Conclusions

- As time goes on and more attempts are made (and presumably learned from) the rate of success on relanding the first stage improves.
- There was a marked 80% improvement in success rate from 2013 to 2020.
- GTO is the orbit with the lowest success rate.
- Launch Sites typically are located on coastal areas closer to the equator.
- The Decision Tree was the Machine Learning model with the highest predictive accuracy though with a small sample size it is hard to say for certain that it would be the best approach long term.

Thank you!