Applied Data Science Capstone:

SpaceX Falcon9 First Stage Landing Prediction

Ksenia Kai

March 8, 2024

OUTLINE

- Executive Summary
- Introduction
- Methodology
- Results
  - Charts
  - Dashboard
- Discussion
  - Findings & Implications
- Conclusion
- Appendix

EXECUTIVE SUMMARY

- Cost unpredictability – major risk for commercial space exploration companies
- Reusable rockets reduce the cost of launching payloads to orbit 10x to 100x
    - SpaceX's Falcon 9 reuses the most expensive part of the rocket - Stage One
- Ability to predict Stage One's success helps determine the cost of the launch
- Focus on features contributing to successful launches will drive down the cost for SpaceY

INTRODUCTION

- Commercial space age: Private companies revolutionizing space exploration and travel
- Top business strategy: Driving down launch costs
- Answer: Reusable rockets
- SpaceX's Falcon 9: Reusing most expensive element - Stage 1
- Predicting Falcon 9's Stage 1 success = Determining the cost of launch
- ML Model to predict launch outcome + highlight features contributing to success => Cost forecast for SpaceY

DATASET

- Rocket launch data from SpaceX REST-API
- Web-scraping data from Wikipedia
  - HTML page parsed using BeautifulSoup
- Missing values: Replaced with the 'mean' value
- Landing outcomes converted to Classes (0=Failure, 1=Success) with Y = classification variable representing the outcome of each launch.

METHODOLOGY: OVERVIEW

- Analytic Approach: Predictive Modeling using multiple Classification Algorithms:
  - Logistic Regression
  - SVM
  - Decision Tree Classifier
  - KNN
- Train-Test-Split: 80/20
- Cross-Validation using GridSearchCV for all models
- Accuracy score checked for training and test sets for all models
- Supported by Confusion Matrix for all models

METHODOLOGY: DATA COLLECTION AND DATA WRANGLING

DATA COLLECTION FROM API:

- SpaceX REST-API as data source
- Auxiliary functions to extract data
- Parse received JSON file and normalize it into a pandas dataframe
- Store data as lists and combine into a dictionary
- Create a pandas dataframe from the dictionary
- Filter the dataframe to only include Falcon9 launches
- Deal with missing values ('mean' for 'PayloadMass')
- Export the final dataframe to CSV

METHODOLOGY: DATA COLLECTION AND DATA WRANGLING

CONTINUED (DATA COLLECTION FROM WEB-SCRAPING):

- Web-scrape launch records from Wikipedia
- Extract Falcon9 launch records HTML table
- Parse HTML table with BeautifulSoup
- Extract column/variable names from the HTML table header and create an empty dictionary with keys from the extracted column names
- Remove HTML noise from the parsed data and fill in the dictionary
- Create a dataframe from the clean dictionary

METHODOLOGY: DATA COLLECTION AND DATA WRANGLING

CONTINUED (DATA WRANGLING):

- Launch Sites: CCAFS LC-40, CCAFS SLC-40, KSC LC-39A, VAFB SLC-4E; placed in the column 'LaunchSite'
- Calculate the number of launches for each site
- Calculate the occurrence of mission outcomes
- Create a landing outcome label from 'Outcome' column
- Create a classification variable 'Class' to represent the outcome of each launch (0 = Failure, 1 = Success).
- Determine the success rate (df["Class"].mean())

METHODOLOGY: EDA AND INTERACTIVE VISUAL ANALYTICS

- Load the SQL extension and connect to the database
- Load SpaceX dataset into SQLIte table
- Remove blank rows from table
- Perform EDA using SQL queries with the SQL magic commands
  - Identify the attributes that can be used to determine if the first stage can be reused
  - Combine multiple features for additional information (i.g. LaunchSite+PayloadMass)
  - Determine if there is an annual trend for launch success rate

METHODOLOGY: EDA AND INTERACTIVE VISUAL ANALYTICS

CONTINUED (FEATURE ENGINEERING):

- Via plotting, determine which attributes are correlated with successful landings
- Based on plotting results, select the features to be used in success prediction: 'FlightNumber', 'PayloadMass', 'Orbit', 'LaunchSite', 'Flights', 'GridFins', 'Reused', 'Legs', 'LandingPad', 'Block', 'ReusedCount', 'Serial'
- One Hot Encoding: convert categorical variables to numeric, preparing the data for an ML model
- Cast the entire dataframe to variable type 'float64'

METHODOLOGY: EDA AND INTERACTIVE VISUAL ANALYTICS

CONTINUED (INTERACTIVE VISUAL ANALYTICS):

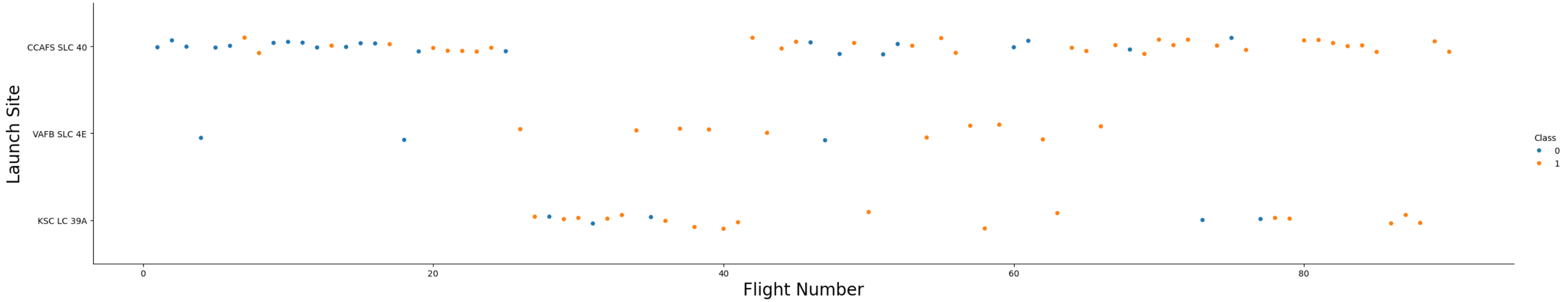
- Build interactive visual analytics for stakeholders: Folium map and Plotly Dash dashboard
- Analyze launch site geo and proximities with Folium:
  - Mark launch site locations and their proximities on a Folium map
  - Identify the optimal launch site
- Build a dashboard application with Dash
  - The application to contain input components (a dropdown list and a range slider) to interact with a pie chart and a scatter point chart

METHODOLOGY: PREDICTIVE ANALYSIS

- Analytic Approach: Predictive Modeling using multiple Classification Algorithms:
  - Logistic Regression
  - SVM
  - Decision Tree Classifier
  - KNN
- Train-Test-Split: 80/20
- Cross-Validation using GridSearchCV for all models
- Accuracy score checked for training and test sets for all models
- Supported by Confusion Matrix for all models

• RESULTS: EDA WITH VISUALIZATION

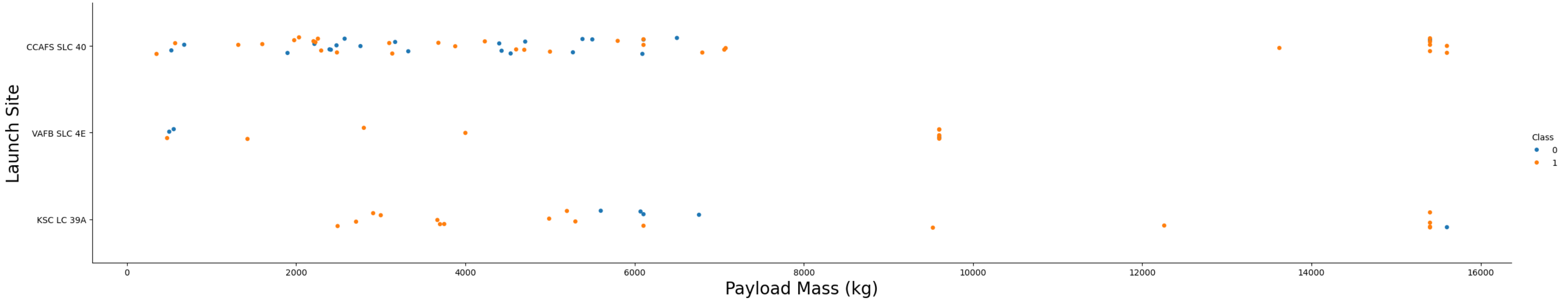
TASK 1: Visualize relationship between Flight Number and Launch Site



INSIGHT: There are more successful outcomes (Class 1, Red) for launch sites VAFB SLC 4E and KSC LC 39A, than for launch site CCAFS SLC 40

• RESULTS: EDA WITH VISUALIZATION

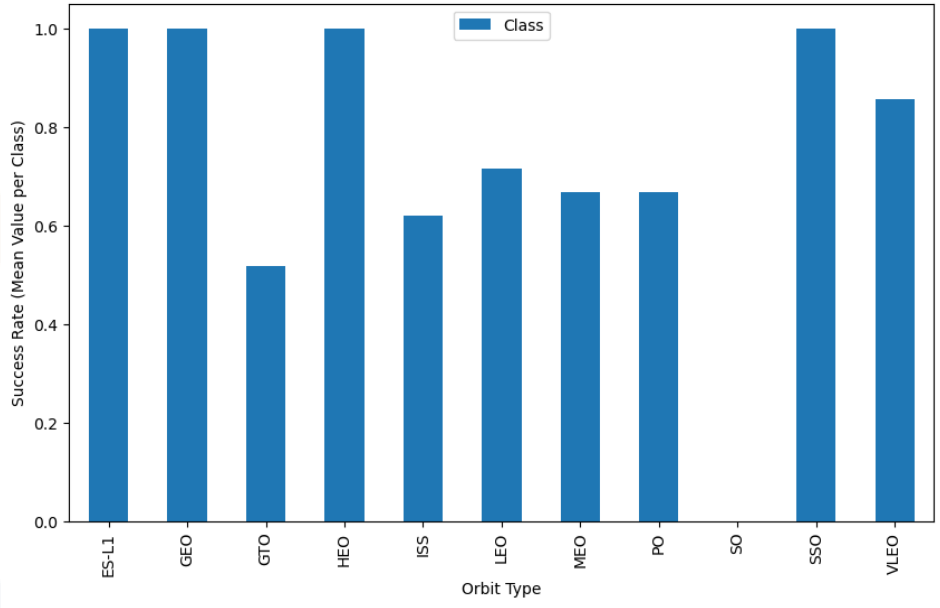
TASK 2: Visualize the relationship between Payload and Launch Site



INSIGHT: For the VAFB-SLC launch site, there are no rockets launched with heavy payload mass (greater than 10000)

• RESULTS: EDA WITH VISUALIZATION

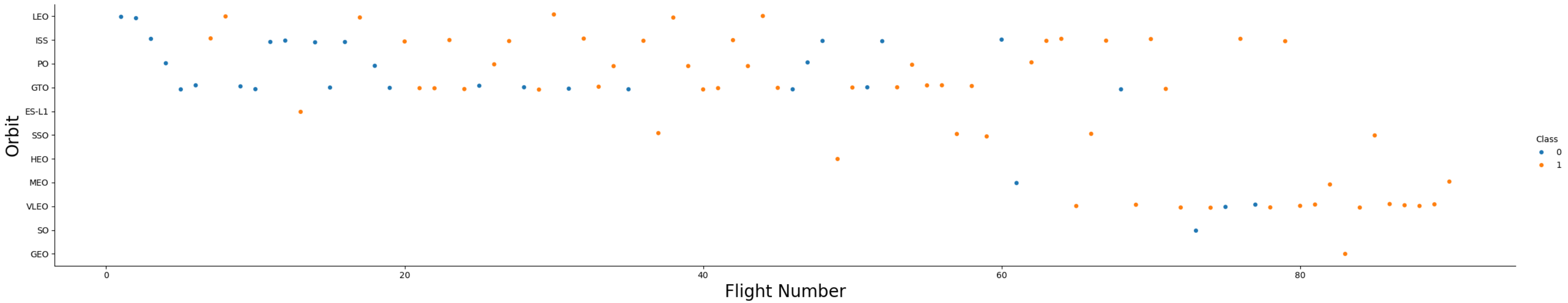
TASK 3: Visualize relationship between Success Rate and Orbit Type:



INSIGHT: Orbit types with highest success rate (=1) are: ES-L1, GEO, HEO, SSO.

• RESULTS: EDA WITH VISUALIZATION

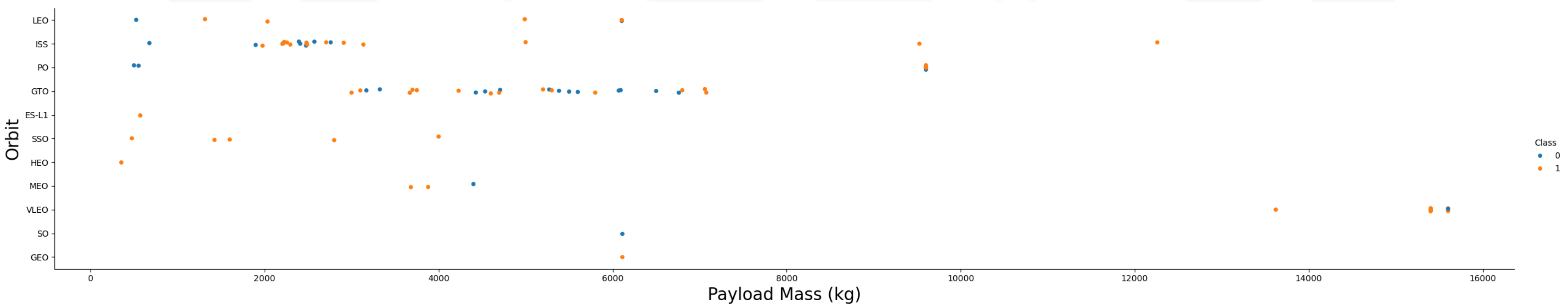
TASK 4: Visualize the relationship between FlightNumber and Orbit Type



INSIGHT: In the LEO orbit, success appears related to the number of flights; there seems to be no relationship between success rate and flight number when in GTO orbit.

• RESULTS: EDA WITH VISUALIZATION

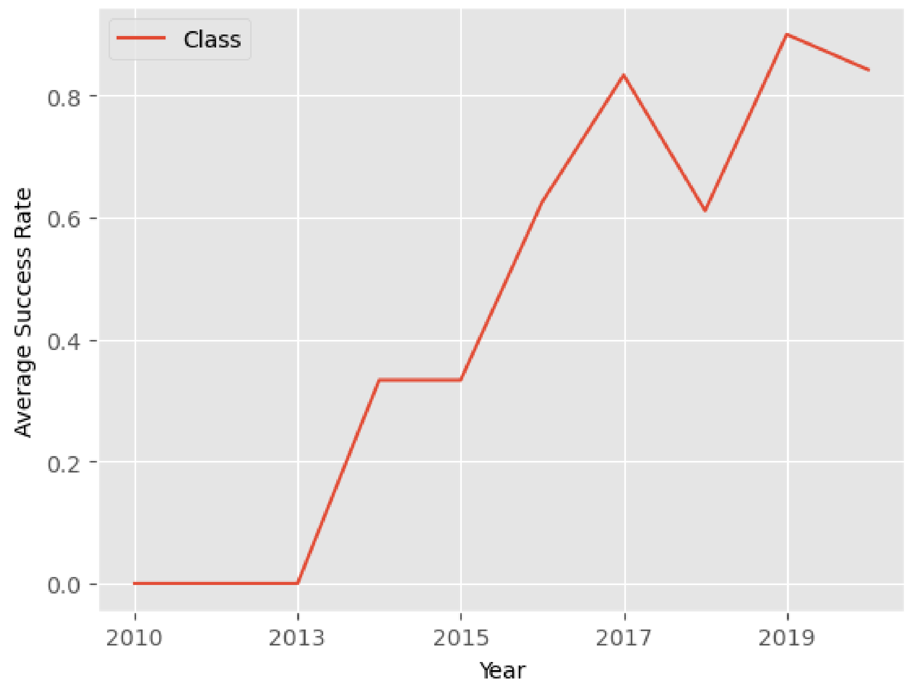
TASK 5: Visualize the relationship between Payload and Orbit Type



INSIGHT: With heavy payloads, the success rate is higher for Polar, LEO, and ISS. However, for GTO we cannot distinguish this as clearly since both positive and negative landings are common.

• RESULTS: EDA WITH VISUALIZATION

TASK 6: Visualize Launch Success Yearly Trend



INSIGHT: Success rate was increasing between 2013 and 2020.

• RESULTS: EDA with SQL

Connecting to the database:

```
%reload_ext sql
%config SqlMagic.displaylimit = 15
```

```
!pip install --upgrade sqlalchemy
```

...

```
# Connecting to the database
%sql mysql+pymysql://root:iruka123@localhost/SpaceX

# Verifying the connection is successful
result = %sql SELECT 1
if result:
    print("Connected to the database.")
else:
    print("Failed to connect to the database.")
```

```
Connecting to 'mysql+pymysql://root:***@localhost/SpaceX'

Running query in 'mysql+pymysql://root:***@localhost/SpaceX'

1 rows affected.

Connected to the database.
```

• RESULTS: EDA with SQL

Task 1: Display the names of the unique launch sites in the space mission

```
%%sql

select distinct Launch_Site from spacextbl
```

Running query in 'mysql+pymysql://root:***@localhost/SpaceX'

4 rows affected.

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

• RESULTS: EDA with SQL

Task 2: Display 5 records where launch sites begin with the string 'CCA'

```
%sql select * from spacextbl where Launch_Site like 'CCA%' limit 5
```

Running query in 'mysql+pymysql://root:***@localhost/SpaceX'

5 rows affected.

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

• RESULTS: EDA with SQL

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

```
%sql SELECT sum(PAYLOAD_MASS__KG_) from spacextbl where Customer = 'NASA (CRS)'
```

Running query in 'mysql+pymysql://root:***@localhost/SpaceX'

1 rows affected.

sum(PAYLOAD_MASS__KG_)
45596.0

• RESULTS: EDA with SQL

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

```
%sql SELECT AVG(PAYLOAD_MASS__KG_) from spacextbl where Booster_Version = 'F9 v1.1'
```

Running query in 'mysql+pymysql://root:***@localhost/SpaceX'

1 rows affected.

AVG(PAYLOAD_MASS__KG_)
2928.4

• RESULTS: EDA with SQL

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

```
%sql SELECT MIN(Date) from spacextbl where Landing_Outcome = 'Success (ground pad)'
```

Running query in 'mysql+pymysql://root:***@localhost/SpaceX'

1 rows affected.

MIN(Date)
2015-12-22

• RESULTS: EDA with SQL

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

```
%%sql

SELECT Booster_Version from spacextbl where Landing_Outcome = 'Success (drone ship)'
and PAYLOAD_MASS__KG_ between 4001 and 5999
```

Running query in 'mysql+pymysql://root:***@localhost/SpaceX'

4 rows affected.

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

• RESULTS: EDA with SQL

Task 7: List the total number of successful and failure mission outcomes

```
%%sql

SELECT COUNT(*) AS Total_Outcome_Count, SUM(Mission_Outcome like '%Success%')
AS Successful, SUM(Mission_Outcome like '%Failure%') AS Failed
FROM spacextbl
```

Running query in 'mysql+pymysql://root:***@localhost/SpaceX'

1 rows affected.

Total_Outcome_Count	Successful	Failed
101	100	1

• RESULTS: EDA with SQL

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

```
%%sql

SELECT Booster_Version, PAYLOAD_MASS__KG_ from spacextbl
where PAYLOAD_MASS__KG_ = (SELECT MAX(PAYLOAD_MASS__KG_) from spacextbl)
```

Running query in 'mysql+pymysql://root:***@localhost/SpaceX'

7 rows affected.

Booster_Version	PAYLOAD_MASS__KG_
F9 FT B1029.1	9600
F9 FT B1036.1	9600
F9 B4 B1041.1	9600
F9 FT B1036.2	9600
F9 B4 B1041.2	9600
F9 B5B1048.1	9600
F9 B5 B1049.2	9600

• RESULTS: EDA with SQL

Task 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.

```
%%sql

SELECT Year(Date) as "Year", Monthname(Date) as "Month", Booster_Version, Launch_Site,
Landing_Outcome from spacextbl
where Landing_Outcome = 'Failure (drone ship)' and Year(Date) = 2015
```

Running query in 'mysql+pymysql://root:***@localhost/SpaceX'

2 rows affected.

Year	Month	Booster_Version	Launch_Site	Landing_Outcome
2015	January	F9 v1.1 B1012	CCAFS LC-40	Failure (drone ship)
2015	April	F9 v1.1 B1015	CCAFS LC-40	Failure (drone ship)

• RESULTS: EDA with SQL

Task 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.

```
%%sql

SELECT Landing_Outcome, COUNT(Landing_Outcome) as Count
FROM spacextbl WHERE Date between '2010-06-04' and '2017-03-20'
GROUP BY Landing_Outcome
ORDER BY Count DESC;
```

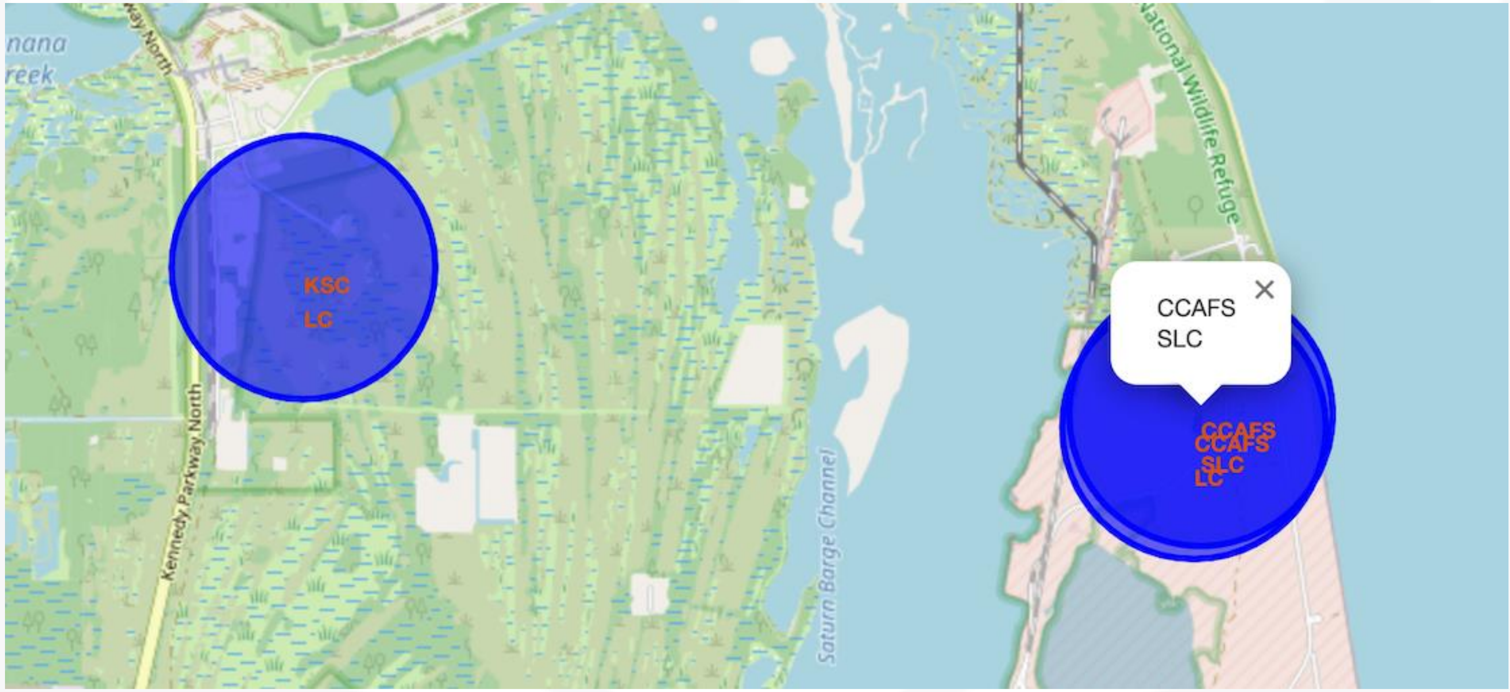
Running query in 'mysql+pymysql://root:***@localhost/SpaceX'

8 rows affected.

Landing_Outcome	Count
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

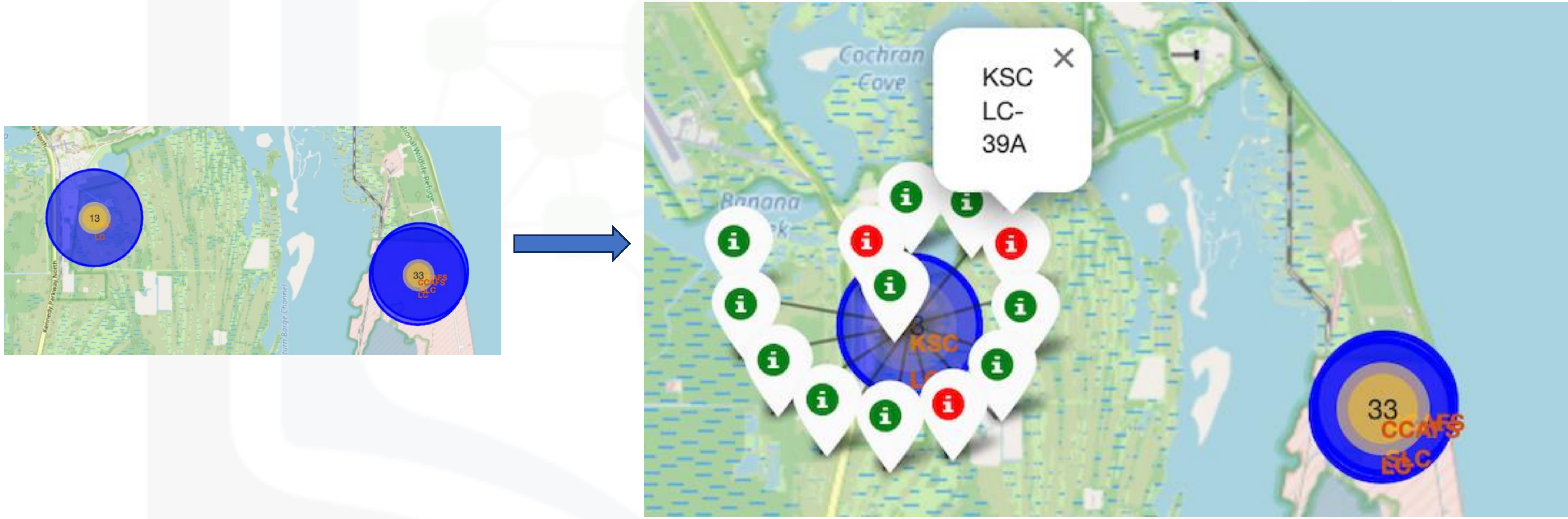
• RESULTS: Interactive FOLIUM Map

Task 1: Create and add *folium.Circle* and *folium.Marker* for each launch site on the site map.



• RESULTS: Interactive FOLIUM Map

TASK 2: Mark the successful/failed launches for each site on the map



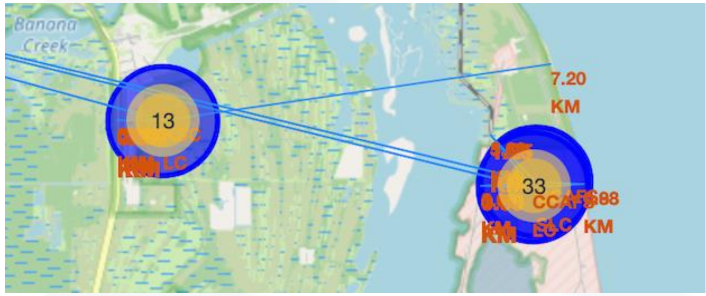
IBM Developer

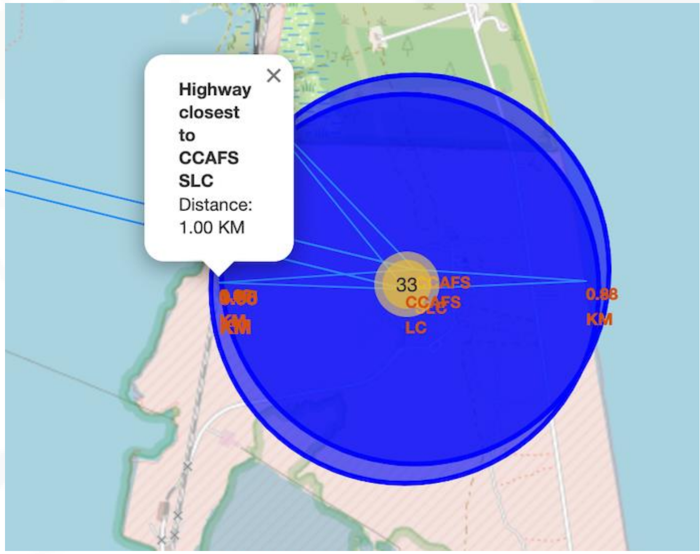
SKILLS NETWORK

• RESULTS: Interactive FOLIUM Map

TASK 3: Calculate the distance between a launch site and its proximities

- Add a *MousePosition* for coordinates
- Create and add a *folium.Marker* on the selected proximity
- Display the distance between the proximity and launch site
- Draw a *PolyLine* between the proximity and launch site

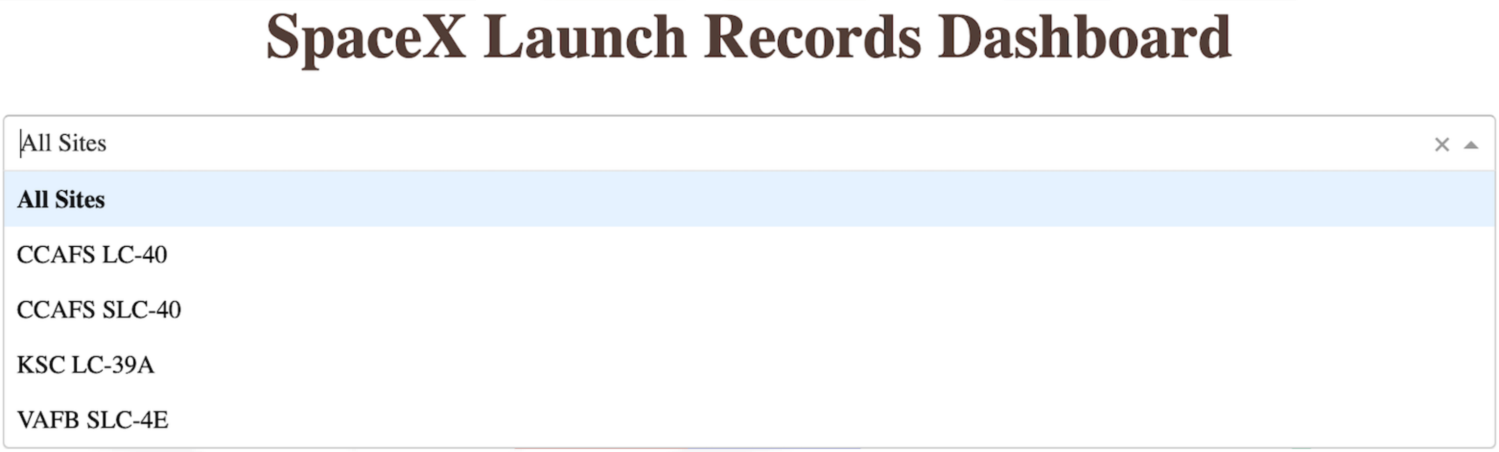




• RESULTS: Plotly DASH Dashboard

TASK 1: Add a Launch Site Drop-Down Input Component

```
# TASK 1: Add a dropdown list to enable Launch Site selection
# The default select value is for ALL sites
dcc.Dropdown(id='site-dropdown',
             options=[
                 {'label': 'All Sites', 'value': 'ALL'},
                 {'label': 'CCAFS LC-40', 'value': 'CCAFS LC-40'},
                 {'label': 'CCAFS SLC-40', 'value': 'CCAFS SLC-40'},
                 {'label': 'KSC LC-39A', 'value': 'KSC LC-39A'},
                 {'label': 'VAFB SLC-4E', 'value': 'VAFB SLC-4E'},
                   ],
             value='ALL',
             placeholder="Select a Launch Site HERE",
             searchable=True
             ),
```



• RESULTS: Plotly DASH Dashboard

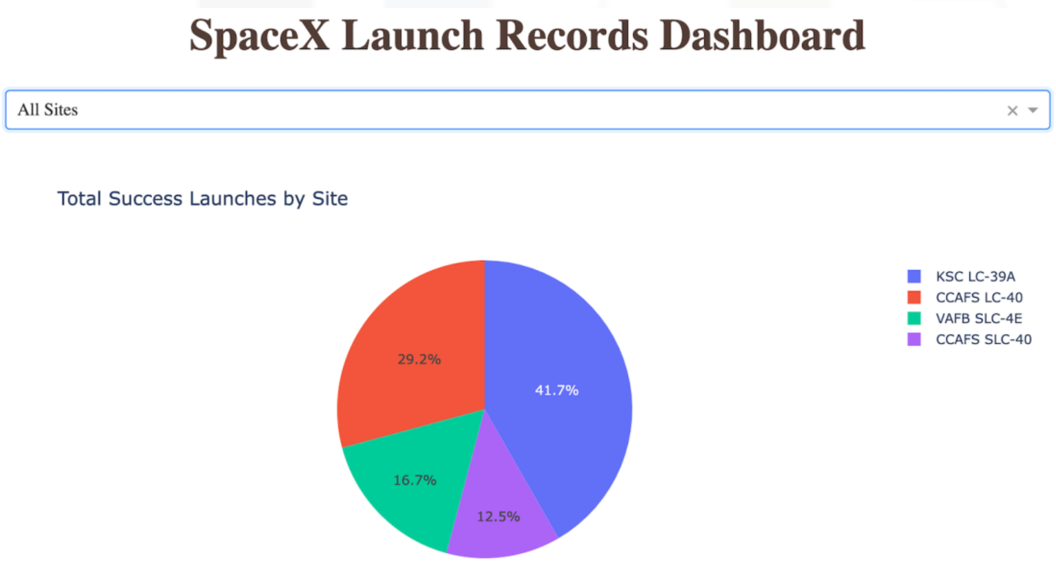
TASK 2: Add a callback function to render success-pie-chart based on selected site dropdown

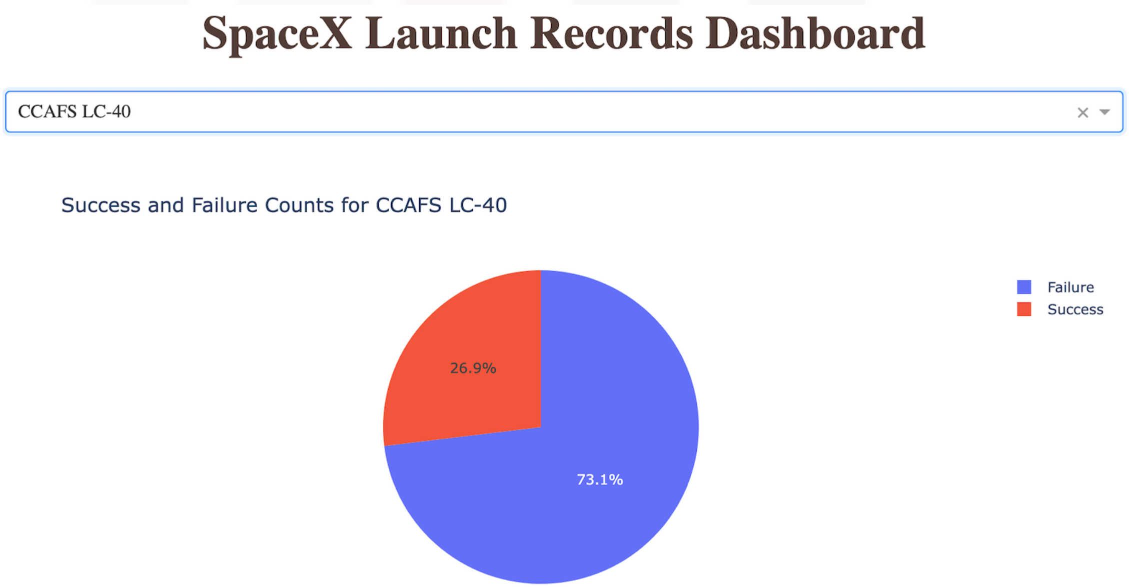
```python
# Callback for launch sites dropdown and pie chart
@app.callback(
    Output(component_id='success-pie-chart', component_property='figure'),
    Input(component_id='site-dropdown', component_property='value'))

def get_pie_chart(entered_site):
    filtered_df = spacex_df
    if entered_site == 'ALL':
        fig = px.pie(filtered_df, values='class',
                     names='Launch Site',
                     title='Total Success Launches by Site',
                     color_discrete_map={'0': 'blue', '1': 'red'})
    else:
        filtered_df = spacex_df[spacex_df['Launch Site'] == entered_site]
        success_count = filtered_df[filtered_df['class'] == 1].shape[0]
        failure_count = filtered_df[filtered_df['class'] == 0].shape[0]

        fig = px.pie(
            names=['Success', 'Failure'],
            values=[success_count, failure_count],
            title=f'Success and Failure Counts for {entered_site}',
            color_discrete_map={'Failure': 'blue', 'Success': 'red'}
        )
```

• RESULTS: Plotly DASH Dashboard

TASK 2: *Continued*

```
# TASK 2: Add a pie chart to show the total successful launches count for all sites
# If a specific launch site was selected, show the Success vs. Failed counts for the site
html.Div(dcc.Graph(id='success-pie-chart')),
html.Br(),
```





• RESULTS: Plotly DASH Dashboard

TASK 3: Add a Range Slider to select payload

```python
# TASK 3: Add a slider to select payload range
dcc.RangeSlider(
                id='payload-slider',
                min=0,
                max=10000,
                step=1000,
                marks={0: '0',
                       2500: '2500',
                       5000: '5000',
                       7500: '7500',
                       10000: '10000'},
                value=[0, 10000]

html.Div(id='slider-output-container'),
html.Br(),
```

Payload range (Kg):

Selected payload range: 0 to 10000 Kg

• RESULTS: Plotly DASH Dashboard

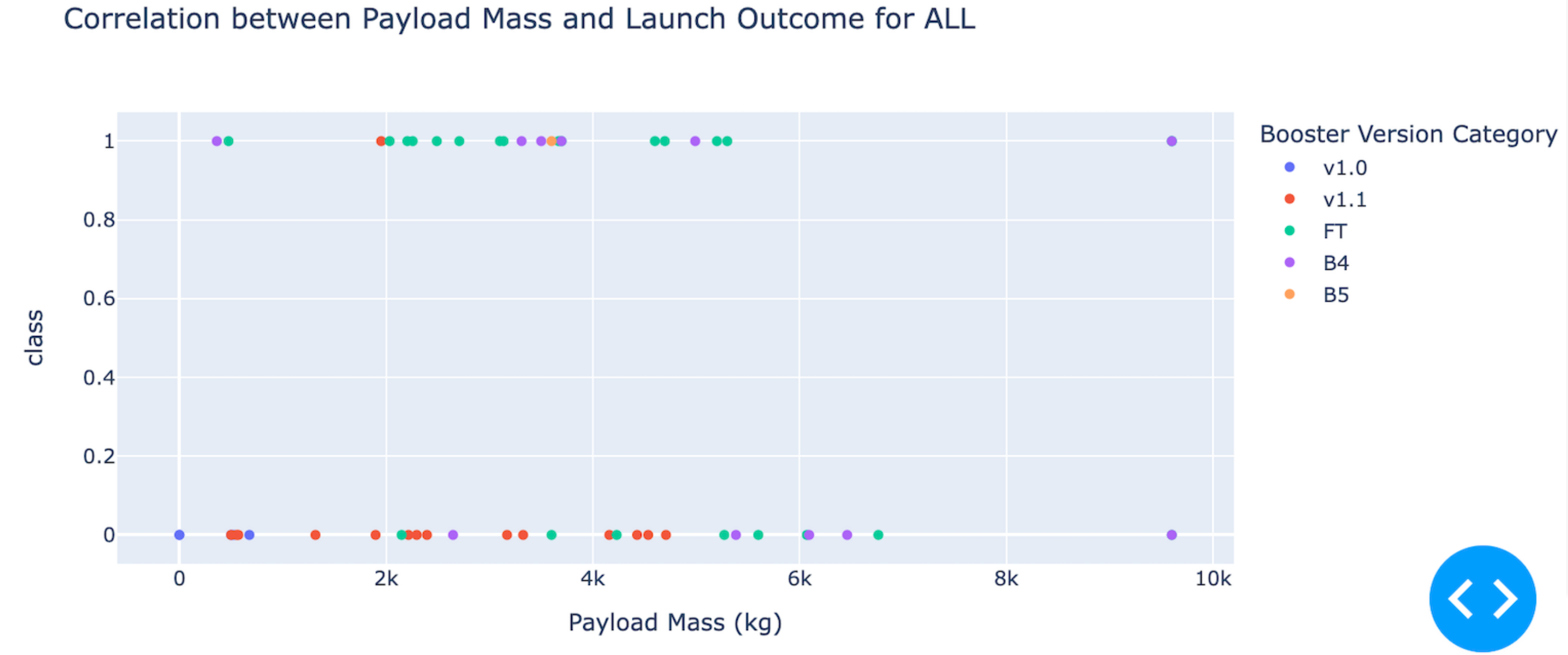
TASK 4: Add a callback function to render the success-payload-scatter-chart scatter plot

```
# Callback for scatter plot
@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 update_scatter_chart(entered_site, payload_range):
    scatter_df = spacex_df.copy()

    # Apply filtering based on site
    if entered_site != 'ALL':
        scatter_df = scatter_df[scatter_df['Launch Site'] == entered_site]

    # Apply filtering based on payload range
    scatter_df = scatter_df[(scatter_df['Payload Mass (kg)'] >= payload_range[0]) &
                            (scatter_df['Payload Mass (kg)'] <= payload_range[1])]

    # Create the scatter plot
    fig = px.scatter(scatter_df,
                     x='Payload Mass (kg)',
                     y='class',
                     color='Booster Version Category',
                     hover_name='Booster Version Category',
                     title=f'Correlation between Payload Mass and Launch Outcome for {entered_site}',
                     size_max=60)

    return fig
```

• RESULTS: Plotly DASH Dashboard

TASK 4: *Continued*

```
# TASK 4: Add a scatter chart to show the correlation between payload and launch success
html.Div(dcc.Graph(id='success-payload-scatter-chart')),
])
```



• RESULTS: Plotly DASH Dashboard

DASH App Link:

DashAppKseniaKai

• RESULTS: Predictive Analysis

TASK 1: Create a NumPy array from the column Class in data, by applying the method to_numpy() then assign it to the variable Y, making sure the output is a Pandas series (only one bracket df['name of column']).

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

print(Y[0:10])
```

```
[0 0 0 0 0 0 1 1 0 0]
```

• RESULTS: Predictive Analysis

TASK 2: Standardize the data in X, then reassign it to the variable X using the provided transform: *transform = preprocessing.StandardScaler()*

```
from sklearn import preprocessing

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

print('Normalized X Arrays:\n', X[:1, :5])
```

```
Normalized X Arrays:
 [[-1.71291154e+00 -1.94814463e-16 -6.53912840e-01 -1.57589457e+00
  -9.73440458e-01]]
```

• RESULTS: Predictive Analysis

TASK 3: Use the function train_test_split to split the data X and Y into training and test data. Set the parameter test_size to 0.2 and random_state to 2. The training data and test data should be assigned to the following labels: *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,)
```

• RESULTS: Predictive Analysis

TASK 4: Create a *logistic regression* object, then create a *GridSearchCV* object logreg_cv with cv = 10. Fit the object to find the best parameters from the dictionary parameters.

```
LR = LogisticRegression()
LR
```

```
▾ LogisticRegression
LogisticRegression()
```

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

logreg_cv = GridSearchCV(LR, parameters, cv=10)
```

```
logreg_cv.fit(X_train, Y_train)
```

```
▸ GridSearchCV
▸ estimator: LogisticRegression
    ▸ LogisticRegression
```

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 hyperparameters:', logreg_cv.best_params_)
print('Accuracy:', logreg_cv.best_score_)
```

```
Tuned hyperparameters: {'C': 0.01, 'penalty': 'l2', 'solver': 'lbfgs'}
Accuracy: 0.8464285714285713
```

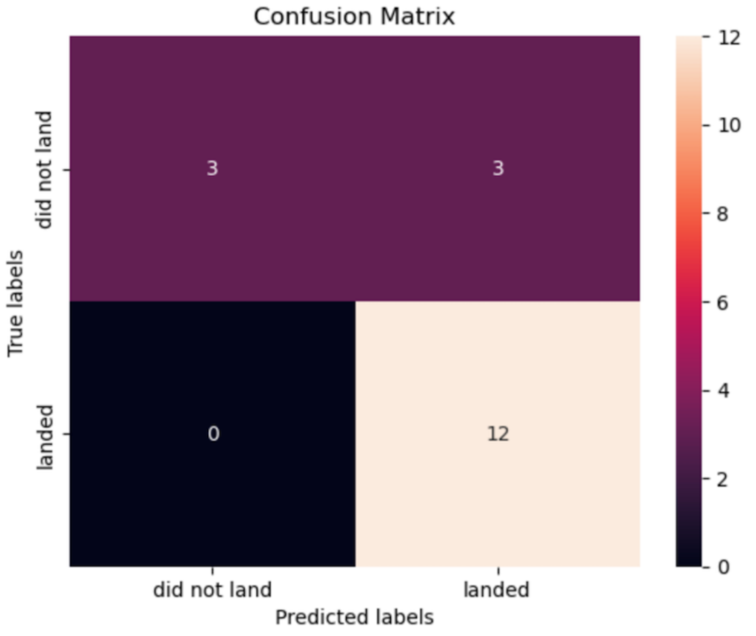
• RESULTS: Predictive Analysis

TASK 5: Calculate the accuracy on the test data using the method score:

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

0.8333333333333334

```
yhat=logreg_cv.predict(X_test)

plot_confusion_matrix(Y_test,yhat)
```



Examining the confusion matrix, we see that logistic regression can distinguish between the different classes.

We see that the major problem is ***false positives.***

• RESULTS: Predictive Analysis

TASK 6: Create a support vector machine object, then create a GridSearchCV object svm_cv with cv - 10. Fit the object to find the best parameters from the dictionary parameters.

```
svm = SVC()
svm
```

```
▾ SVC
SVC()
```

```
parameters = {'kernel':('linear','rbf','poly','rbf','sigmoid'),
              'C': np.logspace(-3, 3, 5),
              'gamma':np.logspace(-3, 3, 5)}

svm_cv = GridSearchCV(svm, parameters, cv=10)
svm_cv
```

```
▸ GridSearchCV
▸ estimator: SVC
    ▸ SVC
```

```
svm_cv.fit(X_train, Y_train)

print('Tuned hyperparameters:',svm_cv.best_params_)
print('Accuracy:',svm_cv.best_score_)
```

```
Tuned hyperparameters: {'C': 1.0, 'gamma': 0.03162277660168379, 'kernel': 'sigmoid'}
Accuracy: 0.8482142857142856
```

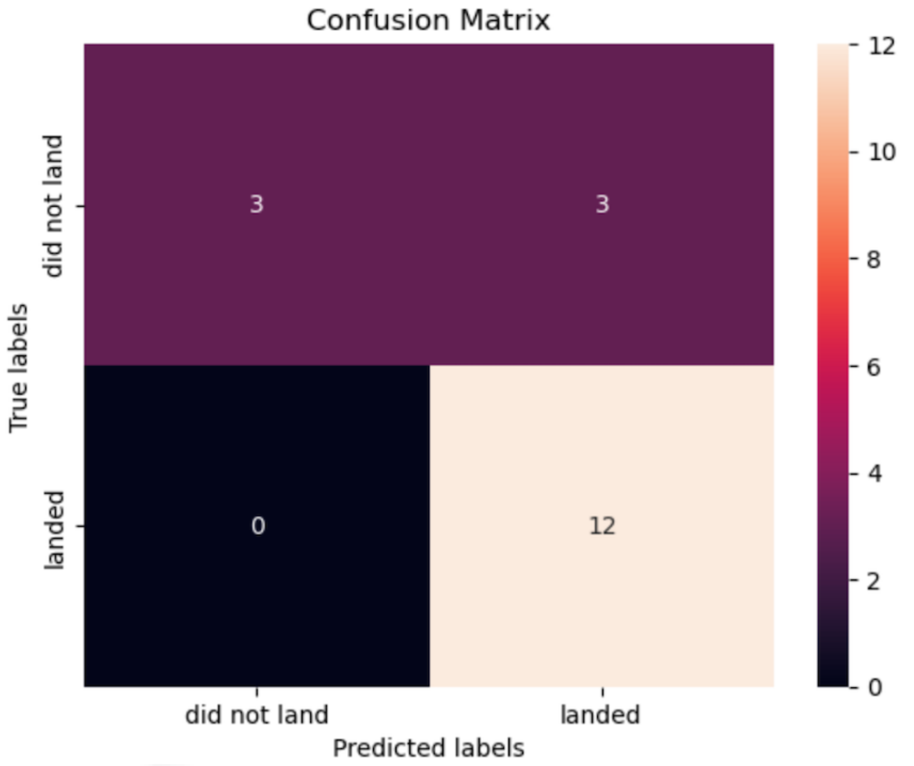
• RESULTS: Predictive Analysis

TASK 7: Calculate the accuracy on the test data using the method *score*:

```
print(svm_cv.score(X_test, Y_test))
```

0.8333333333333334

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



• RESULTS: Predictive Analysis

TASK 8: Create a decision tree classifier object, then create a GridSearchCV object tree_cv with cv = 10. Fit the object to find the best parameters from the dictionary parameters.

```
tree = DecisionTreeClassifier()
tree
```

```
▾ DecisionTreeClassifier
DecisionTreeClassifier()
```

```
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_cv = GridSearchCV(tree, parameters, cv=10)
tree_cv
```

```
▸ GridSearchCV
▸ estimator: DecisionTreeClassifier
  ▸ DecisionTreeClassifier
```

```
tree_cv.fit(X_train,Y_train)
```

...

```
print('Tuned hyperparameters:',tree_cv.best_params_)
print('Accuracy:',tree_cv.best_score_)
```

```
Tuned hyperparameters: {'criterion': 'gini', 'max_depth': 4, 'max_features': 'sqrt', 'min_samples_leaf': 1, 'min_sa
mples_split': 2, 'splitter': 'random'}
Accuracy: 0.8875
```

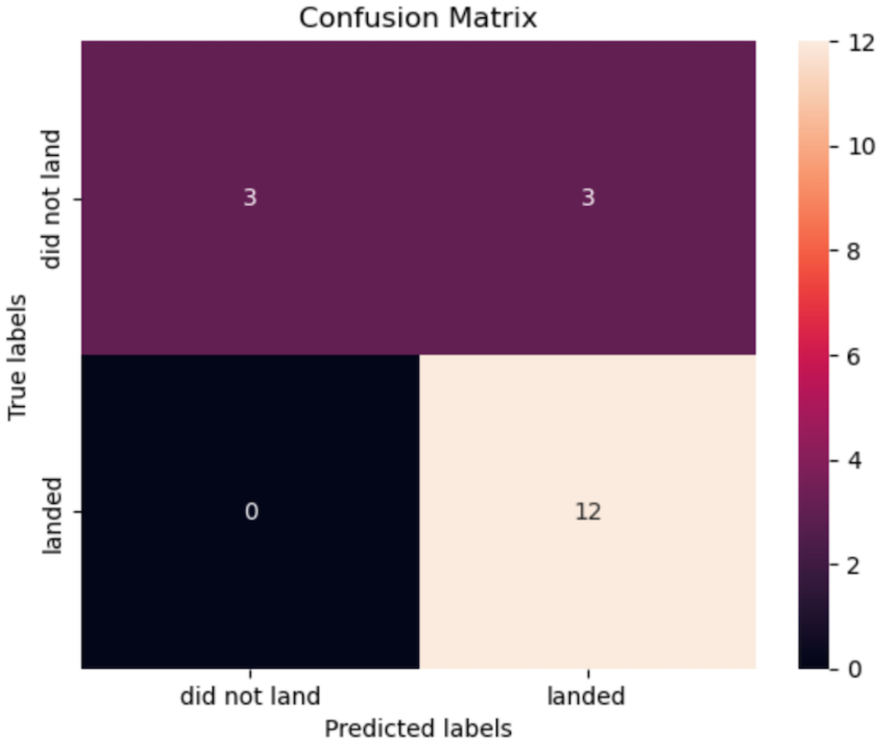
• RESULTS: Predictive Analysis

TASK 9: Calculate the accuracy of tree_cv on the test data using the method *score*:

```
print(tree_cv.score(X_test, Y_test))
```

```
0.8333333333333334
```

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



• RESULTS: Predictive Analysis

TASK 10: Create a k-nearest neighbors object, then create a GridSearchCV object knn_cv with cv = 10. Fit the object to find the best parameters from the dictionary parameters.

```
KNN = KNeighborsClassifier()
KNN
```

```
▾ KNeighborsClassifier
KNeighborsClassifier()
```

```
parameters = {'n_neighbors': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
              'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute'],
              'p': [1,2]}

KNN_cv = GridSearchCV(KNN, parameters, cv=10)
KNN_cv
```

```
▸ GridSearchCV
▸ estimator: KNeighborsClassifier
  ▸ KNeighborsClassifier
```

```
KNN_cv.fit(X_train,Y_train)

print('Tuned hyperparameters:',KNN_cv.best_params_)
print('Accuracy:',KNN_cv.best_score_)
```

```
Tuned hyperparameters: {'algorithm': 'auto', 'n_neighbors': 10, 'p': 1}
Accuracy: 0.8482142857142858
```

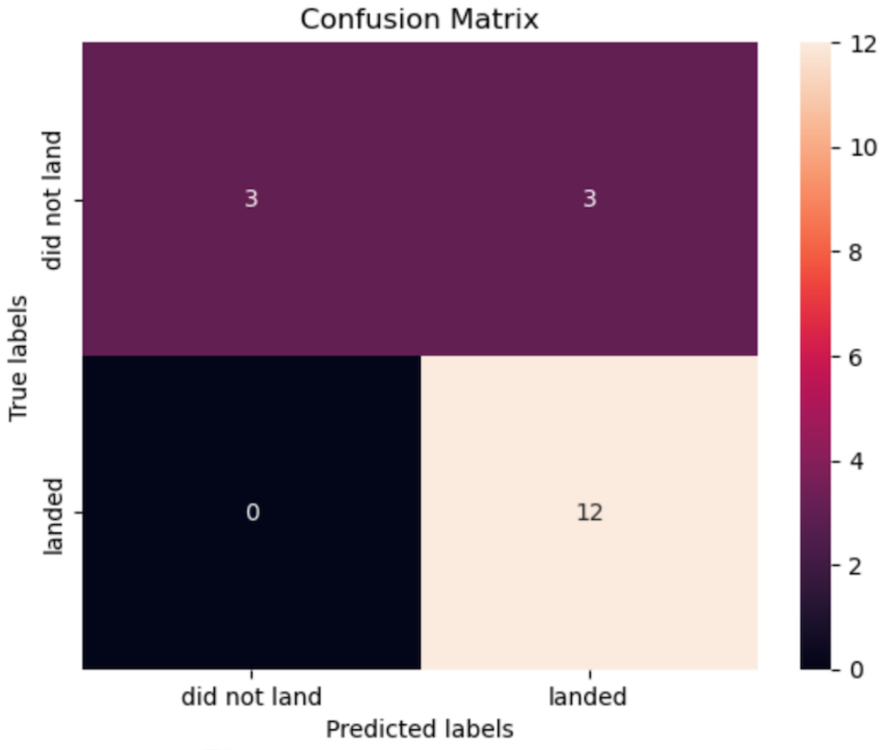
• RESULTS: Predictive Analysis

TASK 11: Calculate the accuracy of KNN_cv on the test data using the method *score*:

```
print(KNN_cv.score(X_test, Y_test))
```

0.8333333333333334

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



• RESULTS: Predictive Analysis

TASK 12: Find the method that performs best

Tested Models:

- **Logistic Regression**
- **Support Vector Machine**
- **Decision Tree Classifier**
- **K-Nearest Neighbors**

All tested models perform very similarly with the accuracy score of 83.33% on a test set. The difference is observed with the accuracy score on a training set, where Decision Tree Classifier stands out with training accuracy of 0.875.

DISCUSSION

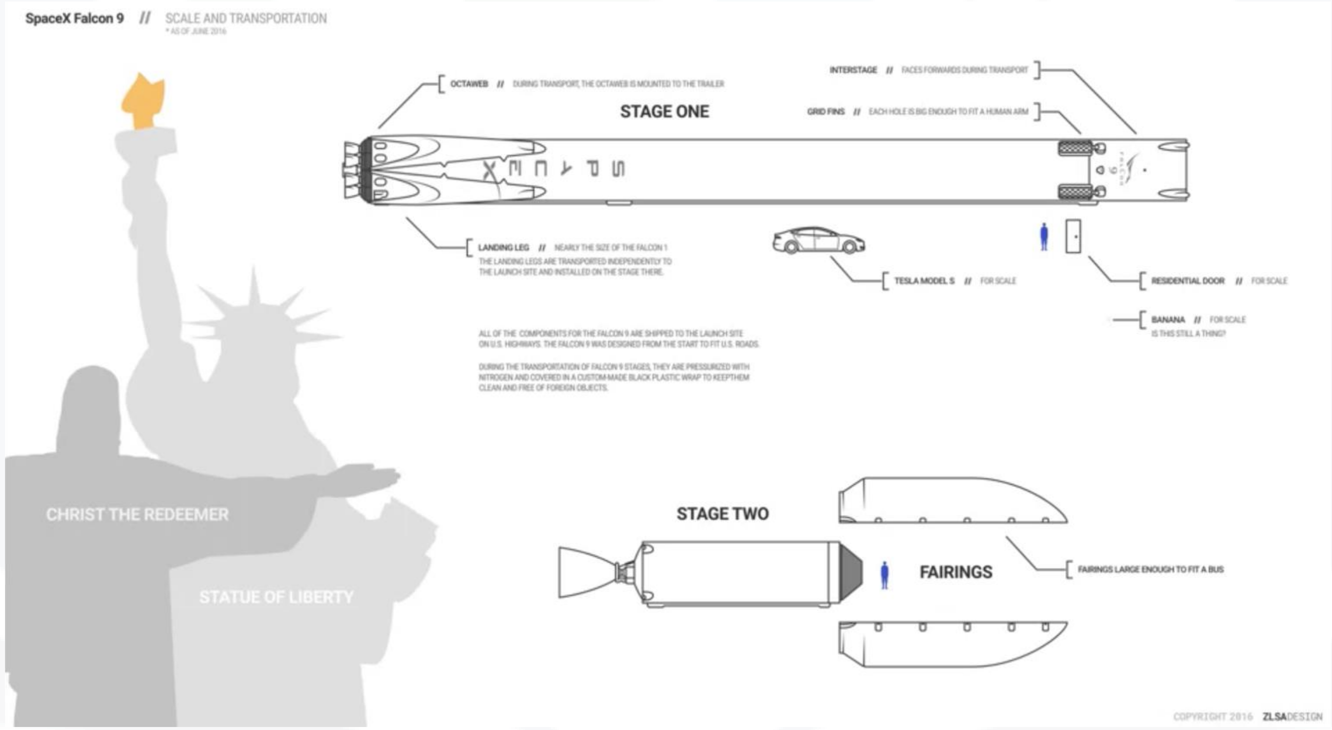
- Is it possible to predict a successful launch/landing?
- Yes! By analyzing multiple features pertaining to a launch, and creating an ML model, it is possible to predict a success rate of a launch.
- *SpaceY* can focus on the features that contribute most to a successful launch/landing =>
- More efficient budget allocation =>
- Staying competitive in a commercial rocket market

CONCLUSION

- Multiple factors and features contribute to launch/landing success
- Importance of geo factors (proximities: railroads, highways, coastline) + launch site location => More successful outcomes for VAFB SLC 4E and KSC LC 39A (77% Success Rate)
- BUT: for the VAFB-SLC, no rockets launched with heavy payloads (>10K) => Payload mass IMPORTANCE
- Multiple ML models tested: similar performance on training sets, BUT on test set – Decision Tree Classifier stands out

APPENDIX

- **Falcon 9 Visual Size Guide** ***(by Forest Katsch, at zlsadesign.com)***



APPENDIX

- **Falcon 9 Specs Overview (*https://www.spacex.com/vehicles/falcon-9/*)**

FALCON 9

OVERVIEW

HEIGHT	70 m / 229.6 ft
DIAMETER	3.7 m / 12 ft
MASS	549,054 kg / 1,207,920 lb
PAYLOAD TO LEO	22,800 kg / 50,265 lb
PAYLOAD TO GTO	8,300 kg / 18,300 lb
PAYLOAD TO MARS	4,020 kg / 8,860 lb