IBM DATA SCIENCE CAPSTONE

ROBERT AGUEROS

23RD APRIL 2022

OUTLINE

EXECUTIVE SUMMARY

- SpaceX data is collected from SpaceX API and Wikipedia page. Class labels were created to identify successful rocket landings as (1) and unsuccessful rocket landing as (0). Data was explored using SQL statements, folium maps, and an interactive dashboard. Variables of interest were identified, and categorical variables were transformed to binary by using one hot encoding. Additionaly, data was standardized and GridSearchCV was used to identify best parameters for machine learning.
- Machine learning models:
  - Logistic Regression
  - Support Vector Machine
  - Decision Tree Classifier
  - K Nearest Neighbor

INTRODUCTION

- Background:
  - SpaceX remains competitive given the successful Falcon 9 – stage one landing. Doing so allows SpaceX to advertise a cost of $62 million vs other competitors with costs of upwards of $165 million per launch. As we, SpaceY, are bidding against SpaceX, being able to predict our rockets first stage successful landing, we can determine the cost per launch and be a viable candidate to win the bid.
- Problem:
  - SpaceY has the task of conducting an exploratory data analysis and creating machine learning models to predict a successful stage 1 recovery.

METHODOLOGY

1

Data collection methodology:

- Combine collected data from SpaceX API and SpaceX Wikipedia page

2

Perform data wrangling

- Classify successful (1) and unsuccessful (0) landings and apply one-hot encoding

3

Perform exploratory data analysis (EDA) using visualization and SQL

4

Perform interactive visual analytics using Folium and Plotly Dash

5

Perform predictive analysis using classification models

- Build model, tune model using GridSearchCV, evaluate classification models

DATA COLLECTION

- SpaceX API
  - Request API
  - Append rocket type, launchpads, payloads, landing outcome
  - Request historical launch data
  - Json_normalize for pandas dataframe
  - Filter to show only Falcon 9 rocket
  - Data wrangle and handle missing values
- Github notebook link
  - SpaceX API
- Web scraping SpaceX Wikipedia
  - Request Wikipedia html
  - BeautifulSoup html5lib parser
  - Pull launch info from third table
  - Use extract_column_from_header() to extract column_names
  - Create a dataframe
- Github notebook link
  - SpaceX Wikipedia Web Scraping

DATA WRANGLING

- Determine the number of launches on each site
  - CCAFS SLC 40: 55
  - KSC LC 39A: 22
  - VAFB SLC 4E 13
- Calculated the number of occurrences of each orbit
- Calculated the number of occurrence of mission outcome per orbit type
- Created landing outcome label from Outcome column
  - Success = 1, Otherwise = 0
- Calculated mean success rate

```
df["Class"].mean()
```

```
0.6666666666666666
```

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

Github Data Wrangling Notebook:
Data Wrangling

EDA WITH SQL

- Display names of launch sites
- Display the total payload mass carried by boosters
  - Total payload mass carried = 45,596
- AVG payload mass carried by F9 v1.1
  - The average payload mass carried by F9 v1.1 is 2,928 kg
- Successful drone ship landing with payload >4000 but <6000
  - Four: B1022, B1026, B1021.1, B1031.2
- List names of booster_versions which have carried the max payload
  - %sql select booster_version from spacextbl where payload_mass__kg_ = (select max(payload_mass__kg_) from spacextbl)
- List failed landing_outcomes on drone ship
- Github notebook link
  - EDA with SQL

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

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

Task 1

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

```
%sql select distinct(LAUNCH_SITE) from SPACEXTBL;
```

```
 * ibm_db_sa://ytm64413:***@6667d8e9-9d4d-4ccb-ba32-21da3bb5aafc.c1ogj3sd0tgtu0lqde00.databases.appdomain.cloud:30376/bludb
Done.
```

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

Task 2

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

```
%sql select * from spacextbl where(LAUNCH_SITE) like 'CCA%' limit 5;
```

```
 * ibm_db_sa://ytm64413:***@6667d8e9-9d4d-4ccb-ba32-21da3bb5aafc.c1ogj3sd0tgtu0lqde00.databases.appdomain.cloud:30376/bludb
Done.
```

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	07:44:00	F9 v1.0 B0005	CCAFS LC-40	Dragon demo flight C2	525	LEO (ISS)	NASA (COTS)	Success	No attempt
2012-10-08	00:35:00	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

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)';
```

```
 * ibm_db_sa://ytm64413:***@6667d8e9-9d4d-4ccb-ba32-21da3bb5aafc.c1ogj3sd0tgtu0lqde00.databases.appdomain.cloud:30376/bludb
Done.
```

1
45596

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';
```

```
 * ibm_db_sa://ytm64413:***@6667d8e9-9d4d-4ccb-ba32-21da3bb5aafc.c1ogj3sd0tgtu0lqde00.databases.appdomain.cloud:30376/bludb
Done.
```

1
2928

Task 5

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

Hint:Use min function

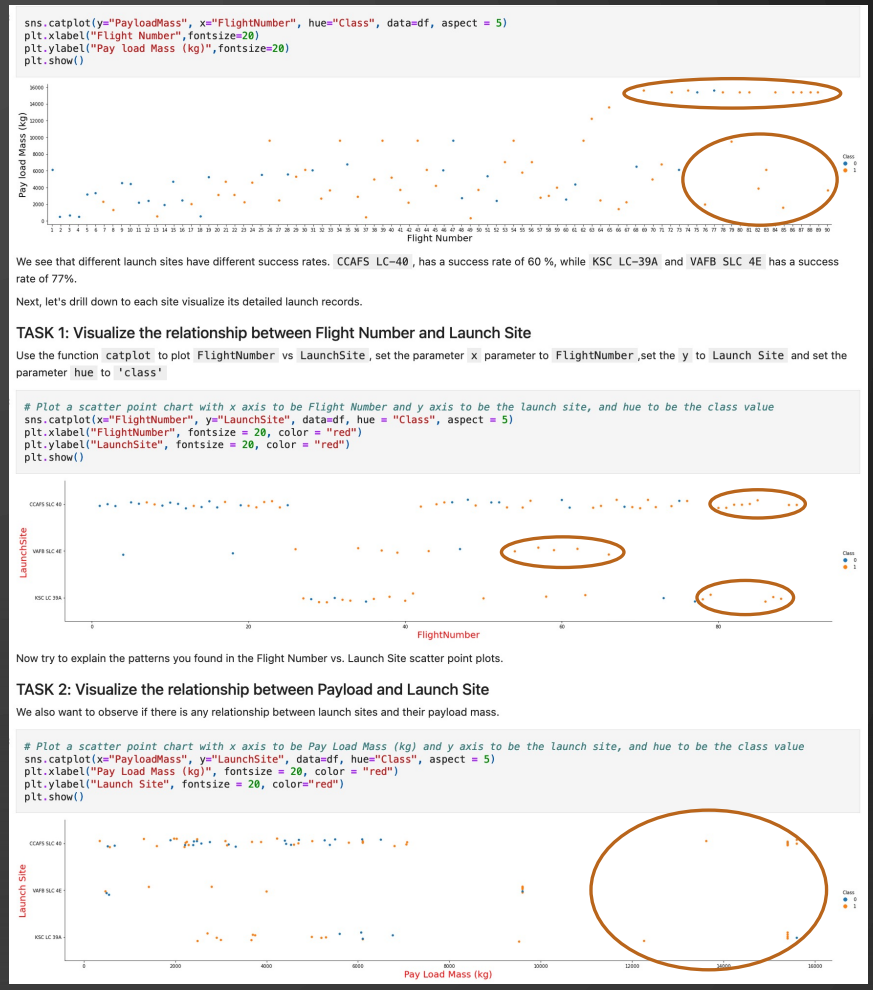
```
%sql select min(date) from spacextbl where landing__outcome= 'Success (ground pad)';
```

```
 * ibm_db_sa://ytm64413:***@6667d8e9-9d4d-4ccb-ba32-21da3bb5aafc.c1ogj3sd0tgtu0lqde00.databases.appdomain.cloud:30376/bludb
Done.
```

1
2015-12-22

EDA WITH DATA VISUALIZATION

- FlightNumber vs PayloadMass
  - On the first scatter plot, we can identify a high success rate with flights on the ladder half and that carry the max payload mass.
- FlightNumber vs LaunchSite
  - In the middle scatter plot, as more launches are conducted per site, the greater success are experienced with all sites.
- PayloadMass(kg) vs LaunchSite
  - On the third scatter plot, we see that the greater the payload, the greater the outcome of success.



```
sns.catplot(y="PayloadMass", x="FlightNumber", hue="Class", data=df, aspect = 5)
plt.xlabel("Flight Number",fontsize=20)
plt.ylabel("Pay load Mass (kg)",fontsize=20)
plt.show()
```


We see that different launch sites have different success rates. `CCAFS LC-40`, has a success rate of 60 %, while `KSC LC-39A` and `VAFB SLC 4E` has a success rate of 77%.

Next, let's drill down to each site visualize its detailed launch records.

TASK 1: Visualize the relationship between Flight Number and Launch Site

Use the function `catplot` to plot `FlightNumber` vs `LaunchSite`, set the parameter `x` parameter to `FlightNumber`,set the `y` to `Launch Site` and set the parameter `hue` to `'class'`

```
# Plot a scatter point chart with x axis to be Flight Number and y axis to be the launch site, and hue to be the class value
sns.catplot(x="FlightNumber", y="LaunchSite", data=df, hue = "Class", aspect = 5)
plt.xlabel("FlightNumber", fontsize = 20, color = "red")
plt.ylabel("LaunchSite", fontsize = 20, color = "red")
plt.show()
```


Now try to explain the patterns you found in the Flight Number vs. Launch Site scatter point plots.

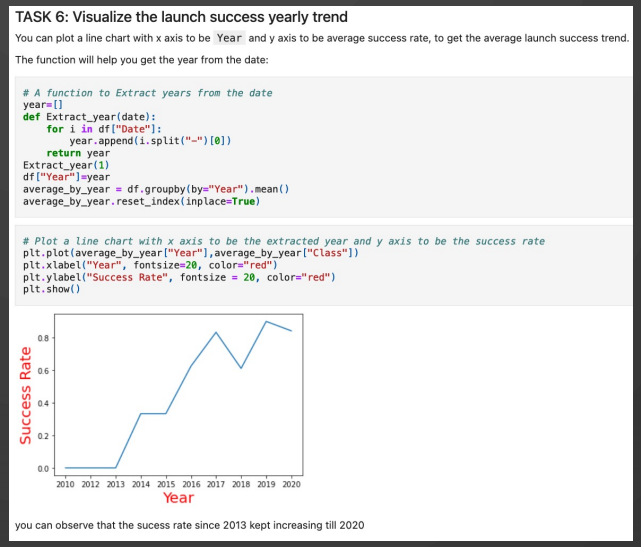
TASK 2: Visualize the relationship between Payload and Launch Site

We also want to observe if there is any relationship between launch sites and their payload mass.

```
# Plot a scatter point chart with x axis to be Pay Load Mass (kg) and y axis to be the launch site, and hue to be the class value
sns.catplot(x="PayloadMass", y="LaunchSite", data=df, hue="Class", aspect = 5)
plt.xlabel("Pay Load Mass (kg)", fontsize = 20, color = "red")
plt.ylabel("Launch Site", fontsize = 20, color="red")
plt.show()
```

EDA WITH DATA VISUALIZATION

- Visualize the relationship of success rate and orbit type
- Visualize relationship between FlightNumber and Orbit type
- Visualize relationship between payload and orbit type
- View launch success yearly trend
- Github notebook link
  - EDA Data Visualizations



TASK 6: Visualize the launch success yearly trend

You can plot a line chart with x axis to be `Year` and y axis to be average success rate, to get the average launch success trend.

The function will help you get the year from the date:

```
# A function to Extract years from the date
year=[]
def Extract_year(date):
    for i in df["Date"]:
        year.append(i.split("-")[0])
    return year
Extract_year(1)
df["Year"]=year
average_by_year = df.groupby(by="Year").mean()
average_by_year.reset_index(inplace=True)
```

```
# Plot a line chart with x axis to be the extracted year and y axis to be the success rate
plt.plot(average_by_year["Year"],average_by_year["Class"])
plt.xlabel("Year", fontsize=20, color="red")
plt.ylabel("Success Rate", fontsize = 20, color="red")
plt.show()
```


you can observe that the sucess rate since 2013 kept increasing till 2020

TASK 3: Visualize the relationship between success rate of each orbit type

Next, we want to visually check if there are any relationship between success rate and orbit type.

Let's create a `bar chart` for the sucess rate of each orbit

```
# HINT use groupby method on Orbit column and get the mean of Class column
orbit_success = df.groupby('Orbit').mean()
orbit_success.reset_index(inplace=True)
sns.barplot(x="Orbit",y="Class",data=orbit_success,hue='Class')
```

```
<AxesSubplot:xlabel='Orbit', ylabel='Class'>
```


Analyze the ploted bar chart try to find which orbits have high sucess rate.

TASK 4: Visualize the relationship between FlightNumber and Orbit type

For each orbit, we want to see if there is any relationship between FlightNumber and Orbit type.

```
# Plot a scatter point chart with x axis to be FlightNumber and y axis to be the Orbit, and hue to be the class value
sns.catplot(x="FlightNumber", y="Orbit", data=df, hue="Class")
plt.xlabel("Fight Number", fontsize = 20, color="red")
plt.ylabel("Orbit", fontsize=20, color="red")
plt.show()
```


You should see that in the 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.

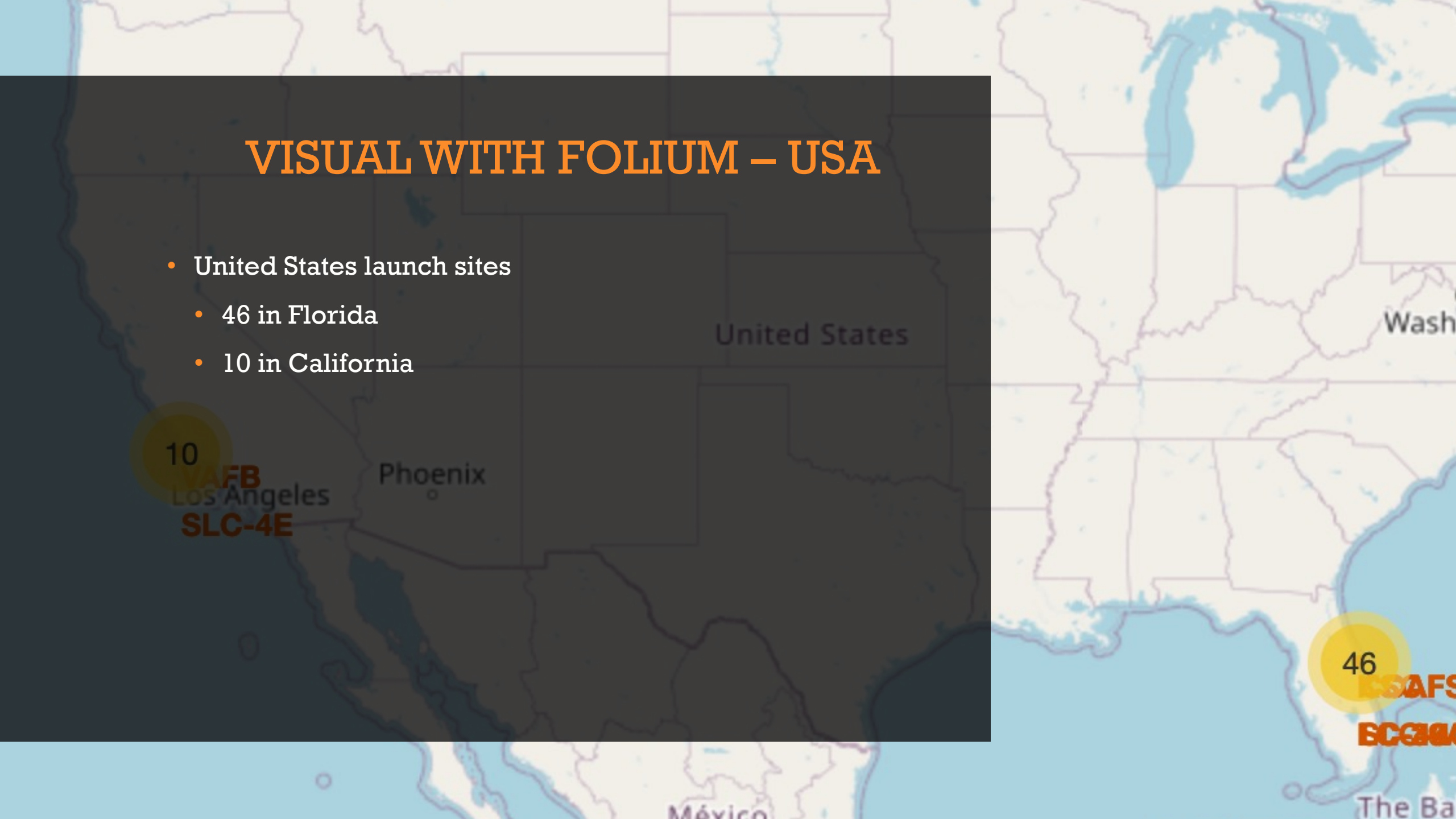
TASK 5: Visualize the relationship between Payload and Orbit type

Similarly, we can plot the Payload vs. Orbit scatter point charts to reveal the relationship between Payload and Orbit type

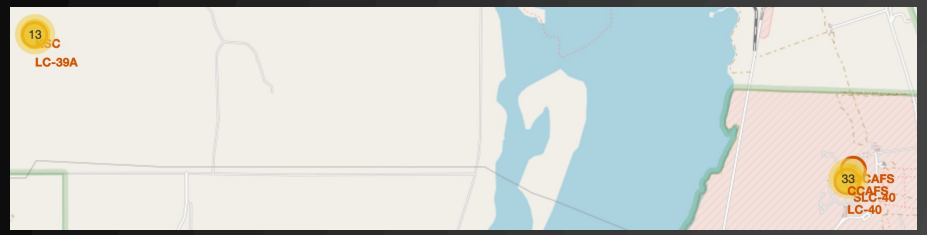
```
# Plot a scatter point chart with x axis to be Payload and y axis to be the Orbit, and hue to be the class value
sns.catplot(x="PayloadMass", y="Orbit", data=df, hue="Class")
plt.xlabel("Payload Mass", fontsize=20, color="red")
plt.ylabel("Orbit", fontsize=20, color="red")
plt.show()
```

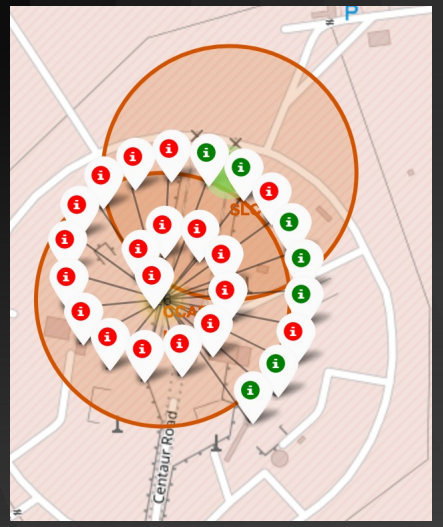
VISUAL WITH FOLIUM – USA

- United States launch sites
  - 46 in Florida
  - 10 in California

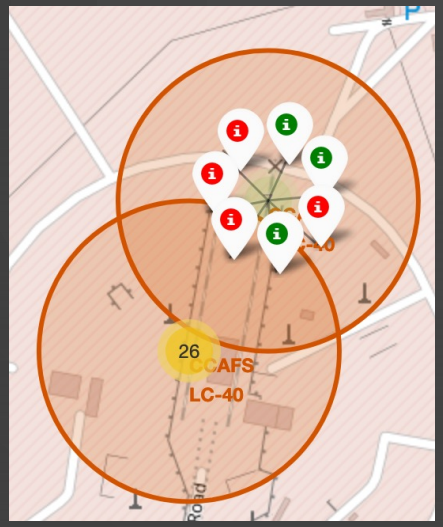


VISUAL WITH FOLIUM - FLORIDA

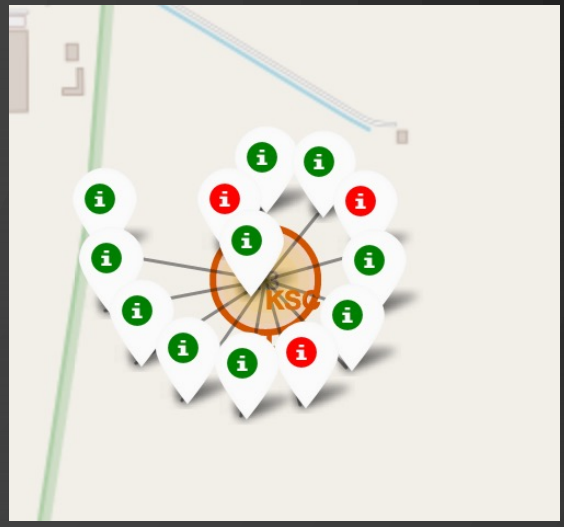




Launch site CCAFS LC-40 has majority failures



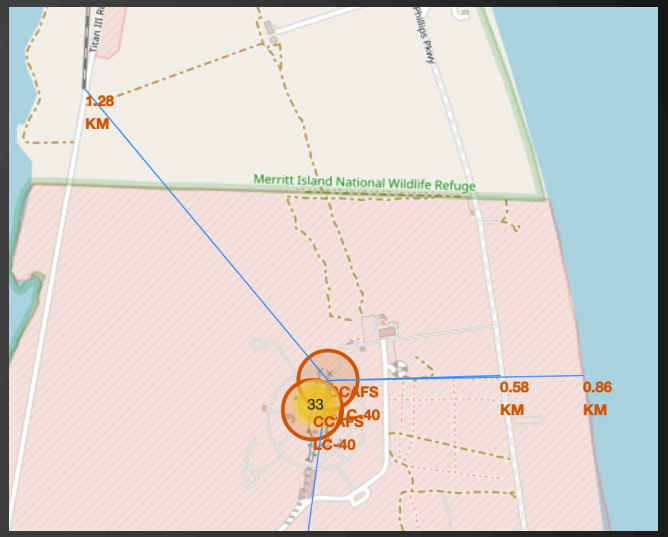
Launch site CCAFS SLC-40 has majority failures

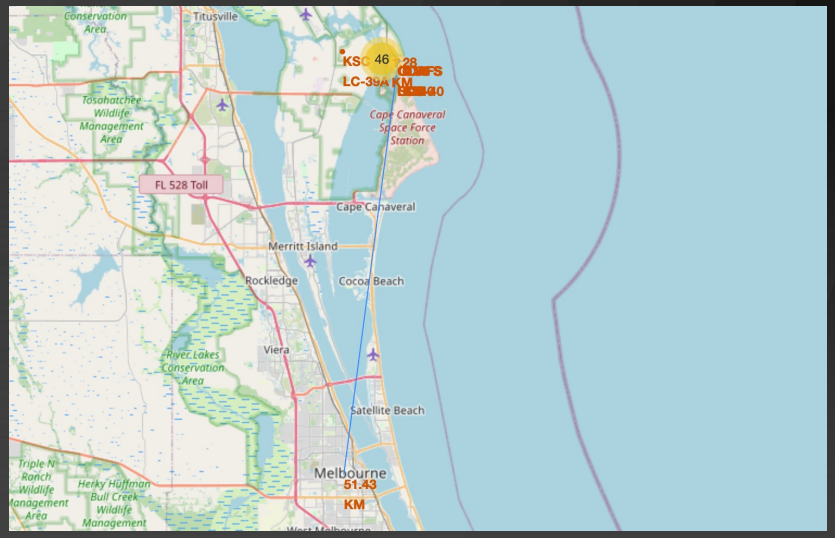


Launch site KSC LC-39A has majority successes

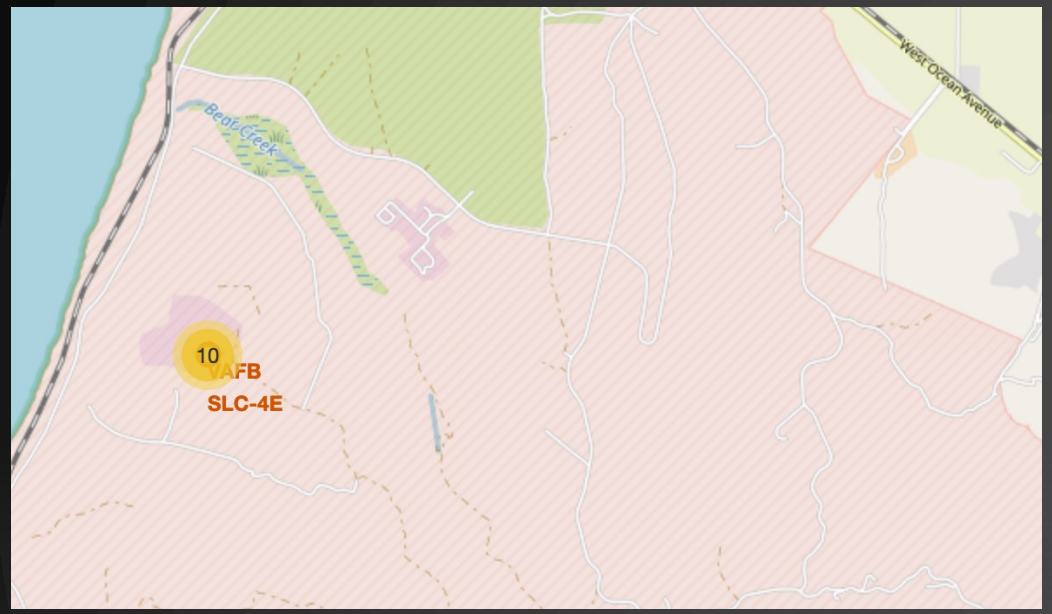
VISUAL WITH FOLIUM - FLORIDA

- Nearest launch site to coast is .86km
- Nearest highway is .58km
- Nearest rail is 1.28km
- Nearest city (Melbourne) is 51.43km

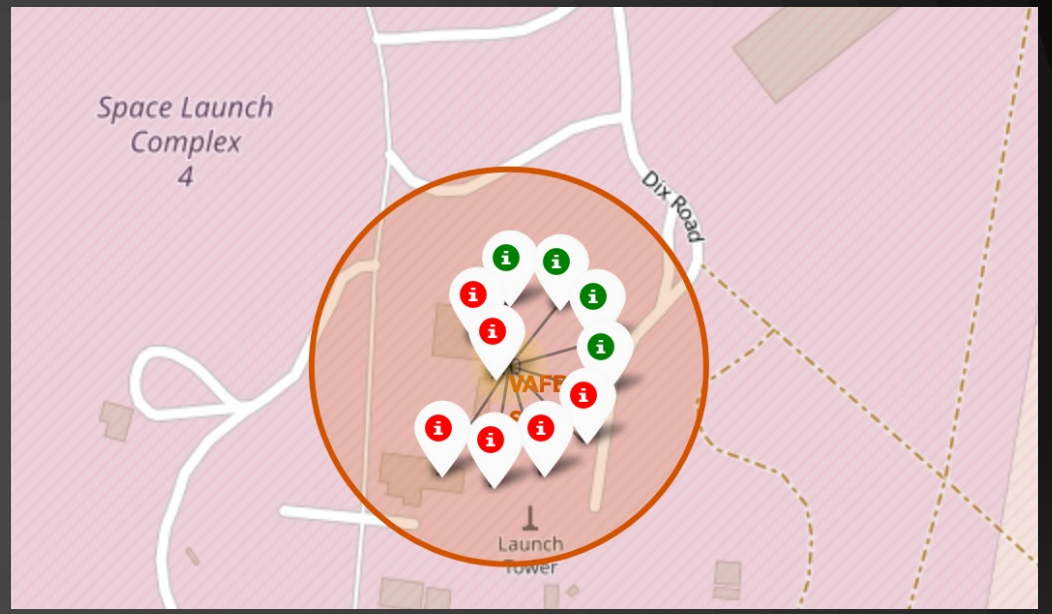




VISUAL WITH FOLIUM - CALIFORNIA

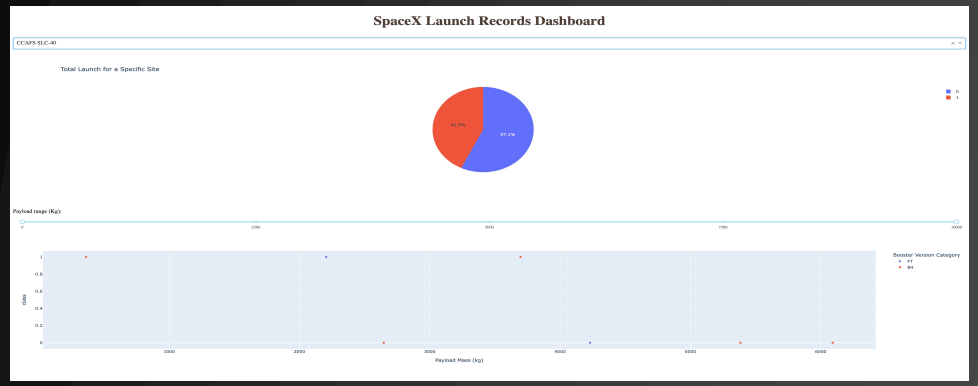


Github Folium Notebook
[Folium Visualization]

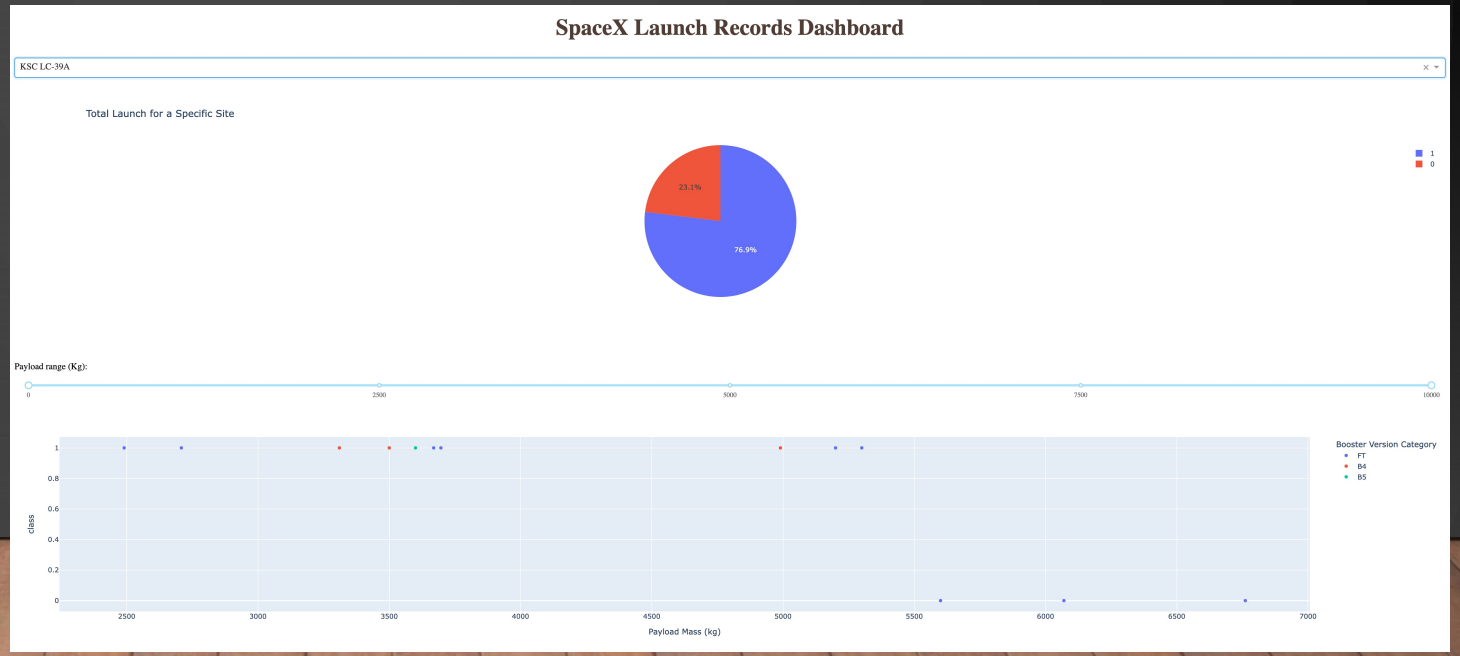


Launch site VAFB SLC-4E has 4 out of 10 successful outcomes

DASHBOARD WITH PLOTLY DASH



While CCAFS SLC-40 had a success rate of 57.1, KSC LC-39 had a higher success rate of 76.9%



PREDICTIVE ANALYTICS - CLASSIFICATION

- Create NumPy array from the column Class
  - Y = data["Class"].to_numpy()
- Standardize the data in X then reassign to variable
  - transform = preprocessing.StandardScaler()
  - x_scaled = transform.fit_transform(X)
  - X = pd.DataFrame(x_scaled)
  - X
- Use function train_test_split to partition the data
  - X_train, X_test, Y_train, Y_test = train_test_split(X,Y, test_size=0.2, random_state=2)
- Set parameter test_size to 0.2 and random_state to 2

PREDICTIVE ANALYTICS – CLASSIFICATION

Logistic Regression

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`.

```
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(lr, parameters, cv=10)
logreg_cv.fit(X_train, Y_train)
```

```
GridSearchCV(cv=10, estimator=LogisticRegression(),
             param_grid={'C': [0.01, 0.1, 1], 'penalty': ['l2'],
                         'solver': ['lbfgs']})
```

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
```

TASK 5

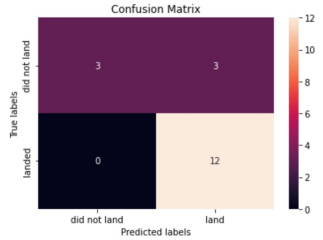
Calculate the accuracy on the test data using the method `score`:

```
accu=[]
methods=[]
accu.append(logreg_cv.score(X_test,Y_test))
methods.append('logistic regression')
logreg_cv.score(X_test,Y_test)
```

```
0.8333333333333334
```

Lets look at the confusion matrix:

```
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.

PREDICTIVE ANALYTICS – CLASSIFICATION

SVM

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`.

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

```
svm_cv= GridSearchCV(svm, parameters, cv=10)
svm_cv.fit(X_train, Y_train)
```

```
GridSearchCV(cv=10, estimator=SVC(),
             param_grid={'C': array([1.00000000e-03, 3.16227766e-02, 1.00000000e+00, 3.16227766e+01,
       1.00000000e+03]),
                         'gamma': array([1.00000000e-03, 3.16227766e-02, 1.00000000e+00, 3.16227766e+01,
       1.00000000e+03]),
                         'kernel': ('linear', 'rbf', 'poly', 'rbf', 'sigmoid')})
```

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

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

TASK 7

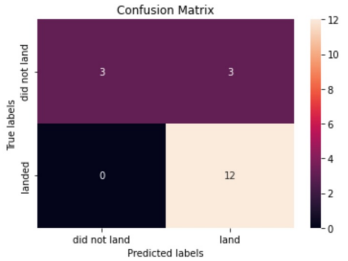
Calculate the accuracy on the test data using the method `score`:

```
accu.append(svm_cv.score(X_test,Y_test))
methods.append('support vector machine')
svm_cv.score(X_test,Y_test)
```

```
0.8333333333333334
```

We can plot the confusion matrix

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



PREDICTIVE ANALYTICS – CLASSIFICATION

Decision Tree Classifier

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`.

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

```
GridSearchCV(cv=10, estimator=DecisionTreeClassifier(),
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': [2, 4, 6, 8, 10, 12, 14, 16, 18],
                         'max_features': ['auto', 'sqrt'],
                         'min_samples_leaf': [1, 2, 4],
                         'min_samples_split': [2, 5, 10],
                         'splitter': ['best', 'random']})
```

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

```
tuned hpyerparameters :(best parameters)  {'criterion': 'entropy', 'max_depth': 18, 'max_features': 'sqrt', 'min_samples_leaf': 1, 'min_sa
mples_split': 2, 'splitter': 'random'}
accuracy : 0.8857142857142858
```

TASK 9

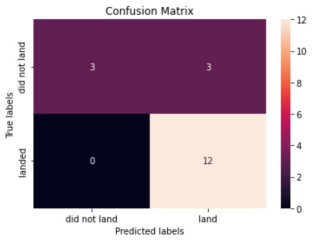
Calculate the accuracy of tree_cv on the test data using the method `score`:

```
accu.append(tree_cv.score(X_test, Y_test))
methods.append('decision tree classifier')
tree_cv.score(X_test,Y_test)
```

```
0.9444444444444444
```

We can plot the confusion matrix

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



PREDICTIVE ANALYTICS – CLASSIFICATION

K Nearest Neighbor

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`.

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

KNN = KNeighborsClassifier()
```

```
knn_cv = GridSearchCV(KNN, parameters, cv=10)
knn_cv.fit(X_train, Y_train)
```

```
GridSearchCV(cv=10, estimator=KNeighborsClassifier(),
             param_grid={'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute'],
                         'n_neighbors': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
                         'p': [1, 2]})
```

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

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

TASK 11

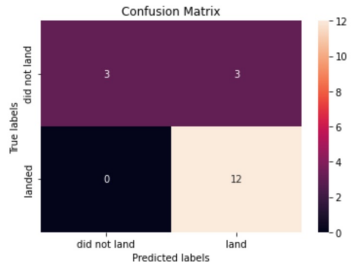
Calculate the accuracy of tree_cv on the test data using the method `score`:

```
accu.append(knn_cv.score(X_test, Y_test))
methods.append('k nearest neighbors')
knn_cv.score(X_test, Y_test)
```

```
0.8333333333333334
```

We can plot the confusion matrix

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



PREDICTIVE ANALYTICS - CLASSIFICATION

- The model with the highest accuracy is the decision tree classifier.

Find the method performs best:

```
print(methods)
print(accu)
```

```
['logistic regression', 'support vector machine', 'decision tree classifier', 'k nearest neighbors']
[0.8333333333333334, 0.8333333333333334, 0.9444444444444444, 0.8333333333333334]
```

Github notebook:
Predictive Analytics - Classification

CONCLUSION

- The data has shown that the larger the flight number and the larger the payload mass kg, the greater number of successful outcomes were achieved.
- The Decision Tree Classifier is the best machine learning model to predict successful outcomes.
- KSC LC-39A is the most successful launch site.
- Launch success rate began increasing in 2013 - 2020