Competing with SpaceX

Applied Data Science Capstone

Abhishek Deodhar

JULY 24, 2022

Table of Contents

Executive summary

Main targets to be achieved :

- Collecting data from API and web scraping.
- Data wrangling.
- Exploratory Data Analysis with SQL.
- Exploratory Data Analysis with Data visualization.
- Interactive maps using Folium.
- Creating a Plotly Dash dashboard.
- Predictions using Machine learning.

Introduction

Background and Goal: SpaceX claims that it will cost them 62 million dollars as compared to other providers having a cost of more than 165M dollars. The main reason for this cost saving is reuse of the first stage.

Our goal in this project is to determine the cost of project by determining whether the first stage landing will be successful or not. We will achieve this by using machine learning techniques.

Data collection methodology

Objectives:

- Request to the SpaceX API
- Clean the requested data

Helper functions:

```python
In [2]:
# Takes the dataset and uses the rocket column to call the API and append the data to the list
def getBoosterVersion(data):
    for x in data['rocket']:
        response = requests.get("https://api.spacexdata.com/v4/rockets/"+str(x)).json()
        BoosterVersion.append(response['name'])
```

From the `launchpad` we would like to know the name of the launch site being used, the logitude, and the latitude.

```python
In [3]:
# Takes the dataset and uses the launchpad column to call the API and append the data to the list
def getLaunchSite(data):
    for x in data['launchpad']:
        response = requests.get("https://api.spacexdata.com/v4/launchpads/"+str(x)).json()
        Longitude.append(response['longitude'])
        Latitude.append(response['latitude'])
        LaunchSite.append(response['name'])
```

Steps:

- Request APIs and normalize using json_normalize.
- Create helper functions for extracting info from API like boosterversion, Launchsite,Payload data etc.
- Apply the functions and create the data frame.
- Filter DF to include only falcon 9.
- Deal with the missing values.
- Refer Appendix slide no. 17,18.

Data Wrangling

1. As the first step we calculate the percentage of missing values & identified whether the columns are categorical or numerical.

```
In [6]: df.isnull().sum()/df.count()*100

FlightNumber       0.000
Date               0.000
BoosterVersion     0.000
PayloadMass        0.000
Orbit              0.000
LaunchSite         0.000
Outcome            0.000
Flights            0.000
GridFins           0.000
Reused             0.000
Legs               0.000
LandingPad        40.625
Block              0.000
ReusedCount        0.000
Serial             0.000
Longitude          0.000
Latitude           0.000
dtype: float64
```

```
In [7]: df.dtypes

FlightNumber        int64
Date               object
BoosterVersion     object
PayloadMass       float64
Orbit              object
LaunchSite         object
Outcome            object
Flights             int64
GridFins             bool
Reused               bool
Legs                 bool
LandingPad         object
Block             float64
ReusedCount         int64
Serial             object
Longitude         float64
Latitude          float64
dtype: object
```

2.Then we calculate number of launches in each site.

```
In [8]: # Apply value_counts() on column LaunchSite
        df['LaunchSite'].value_counts()

CCAFS SLC 40    55
KSC LC 39A      22
VAFB SLC 4E     13
Name: LaunchSite, dtype: int64
```

Data Wrangling

3. Calculate the number and occurrence of each orbit.

4. Count the different type of landing outcomes

5. Create a label where the value was 1 for successful landings and 0 for failures.

6. Store the landing values in a column called “class”.

7. Finally determine the success rate by calculating the mean of the column class.

In [14]:

```python
# landing_class = 0 if bad_outcome
# landing_class = 1 otherwise
#df.head()

def funA(input1):
    if input1 in bad_outcomes:
        return 0
    else:
        return 1
landing_class= df["Outcome"].apply(funA)
landing_class
```

```
0     0
1     0
2     0
3     0
4     0
     ..
85    1
86    1
87    1
88    1
89    1
Name: Outcome, Length: 90, dtype: int64
```

Data Wrangling

In [16]:

```
df.head(5)
```

LaunchSite	Outcome	Flights	GridFins	Reused	Legs	LandingPad	Block	ReusedCount	Serial	Longitude	Latitude	Class
CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B0003	-80.577366	28.561857	0
CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B0005	-80.577366	28.561857	0
CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B0007	-80.577366	28.561857	0
VAFB SLC 4E	False Ocean	1	False	False	False	NaN	1.0	0	B1003	-120.610829	34.632093	0
CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B1004	-80.577366	28.561857	0

We can use the following line of code to determine the success rate:

In [17]:

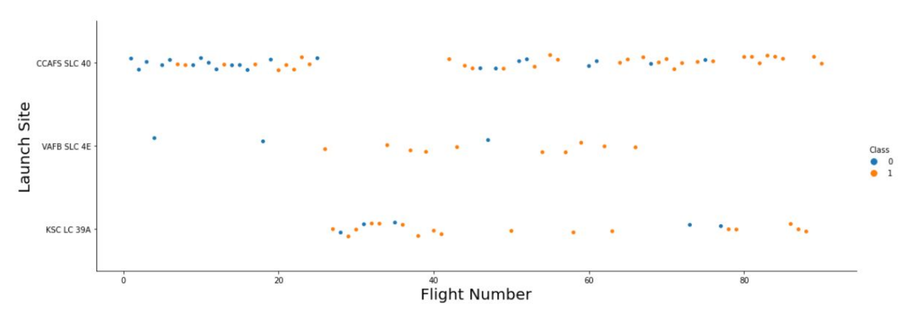
```
df["Class"].mean()
```

```
0.6666666666666666
```

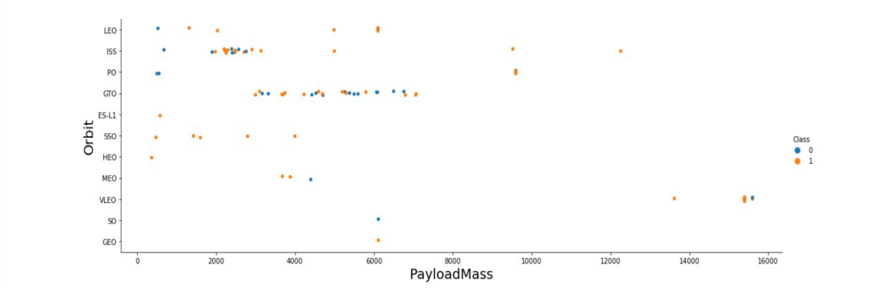
EDA and interactive visual analytics methodology

Here we first load the data set and plot relevant parameters along the X and Y axis for exploring the data. The parameters explored were - FlightNumber vs. PayloadMass, FlightNumber vs LaunchSite, PayloadMass vs LaunchSite, Orbit vs Class.mean(),FlightNumber vs orbit, PayloadMass vs orbit, Year vs average success rate etc.

```
sns.catplot(y="LaunchSite", x="FlightNumber", hue="Class", data=df, aspect = 3)
plt.xlabel("Flight Number",fontsize=20)
plt.ylabel("Launch Site",fontsize=20)
plt.show()
```




```
In [8]:
# Plot a scatter point chart with x axis to be Payload and y axis to be the Orbit, and hue to be the class
sns.catplot(y="Orbit", x="PayloadMass", hue="Class", data=df, aspect = 3)
plt.xlabel("PayloadMass",fontsize=20)
plt.ylabel("Orbit",fontsize=20)
plt.show()
```



EDA and interactive visual analytics methodology

Feature engineering: Identifying the importance of each variable in finding the success rate. We apply one hot encoding to selected features - Orbits, LaunchSite, LandingPad, and Serial. This will make our data more relevant as we convert the categorical columns to a binary 0/1.

In [23]:

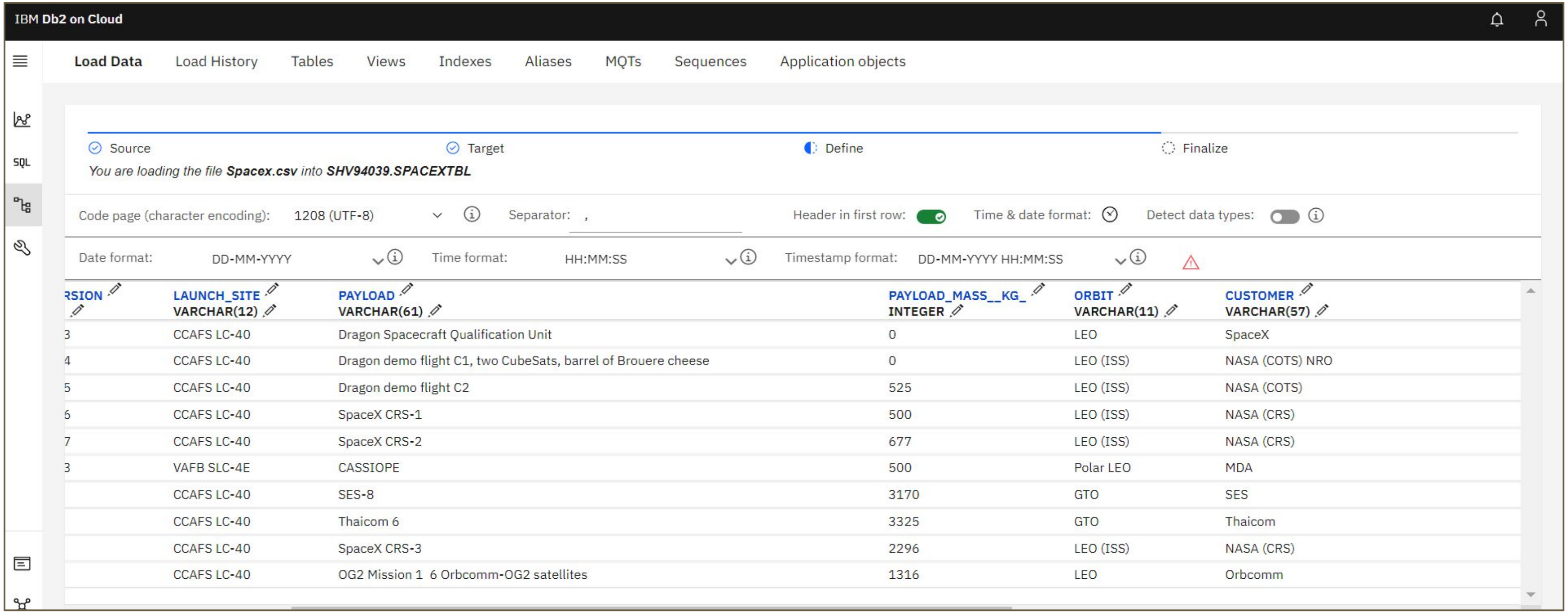
```
# HINT: Use get_dummies() function on the categorical columns
features_one_hot = pd.get_dummies(features, columns=["Orbit","LaunchSite","LandingPad","Serial"])
features_one_hot
```

	FlightNumber	PayloadMass	Flights	GridFins	Reused	Legs	Block	ReusedCount	Orbit_ES-L1	Orbit_GEO	...	Serial_B104
0	1	6104.959412	1	False	False	False	1.0	0	0	0	...	0
1	2	525.000000	1	False	False	False	1.0	0	0	0	...	0
2	3	677.000000	1	False	False	False	1.0	0	0	0	...	0
3	4	500.000000	1	False	False	False	1.0	0	0	0	...	0
4	5	3170.000000	1	False	False	False	1.0	0	0	0	...	0
...	...	...	...	...	...	...	...	...	...	...	...	...
85	86	15400.000000	2	True	True	True	5.0	2	0	0	...	0
86	87	15400.000000	3	True	True	True	5.0	2	0	0	...	0
87	88	15400.000000	6	True	True	True	5.0	5	0	0	...	0
88	89	15400.000000	3	True	True	True	5.0	2	0	0	...	0
89	90	3681.000000	1	True	False	True	5.0	0	0	0	...	0

90 rows × 80 columns

EDA with SQL

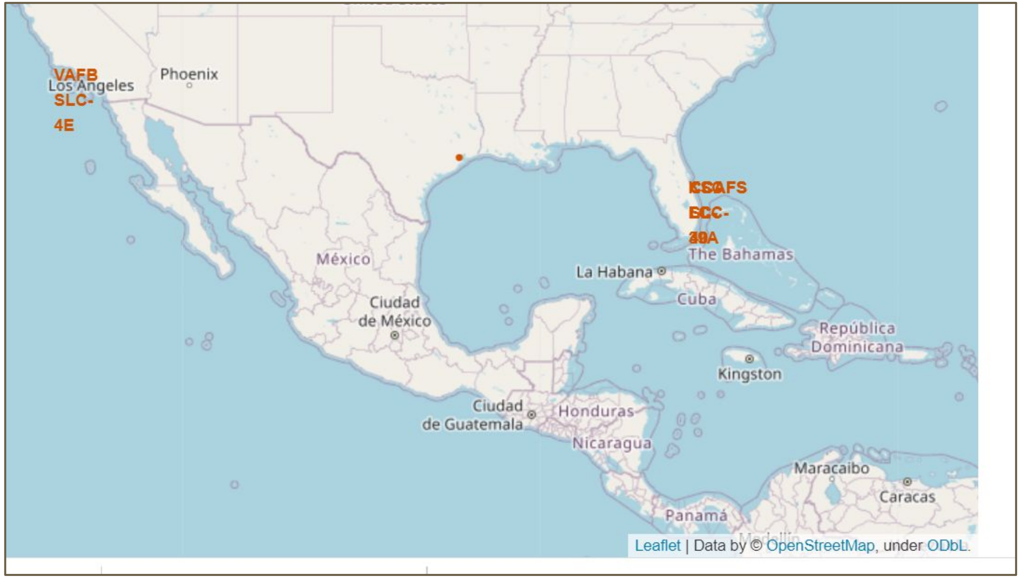
Under this activity we load the data into the database, query the data using python, understand the data to get more information on parameters like launch sites, mission outcomes etc. Some of the tasks have been shown in the below slides. Refer appendix slide no. 19,20.



Interactive map with Folium

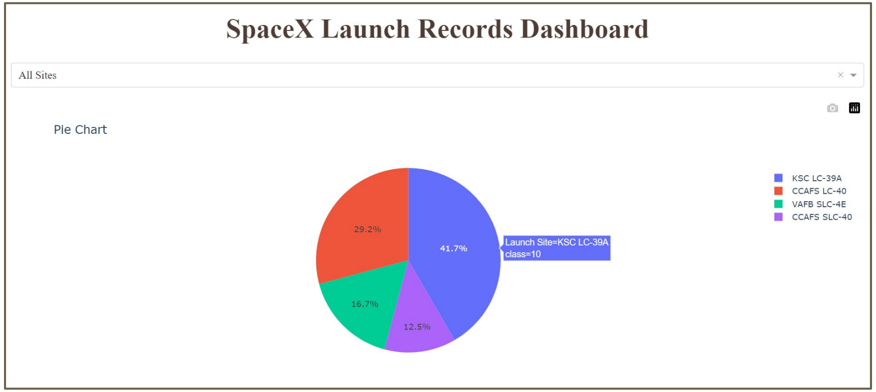
Here we are trying to plot launch sites and their proximity to key locations like the highway,coast etc. This helps us in understand the logic behind selection of these sites at specific locations.

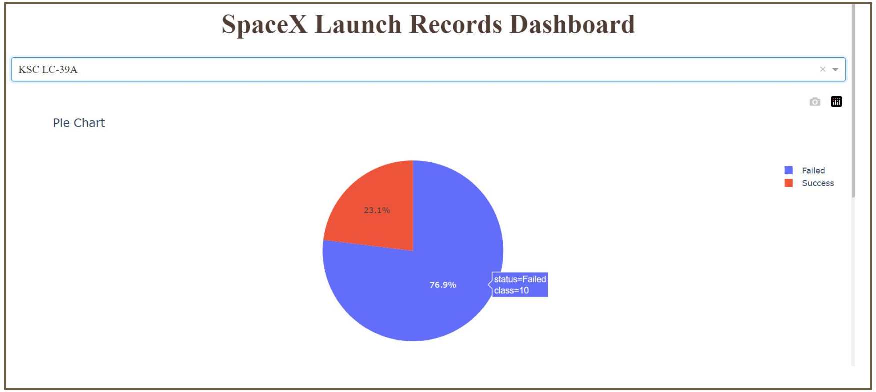
For e.g. most of the launch locations are near the coast and close to the equator. I think the main reason for that could be to get a radial path along the earth while exiting into outer space.

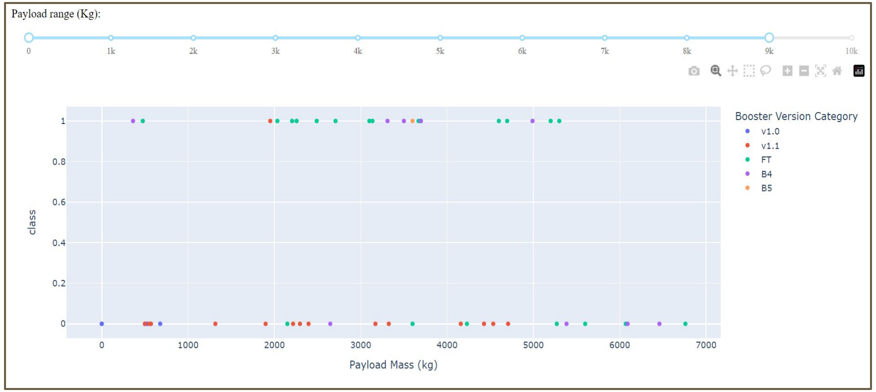


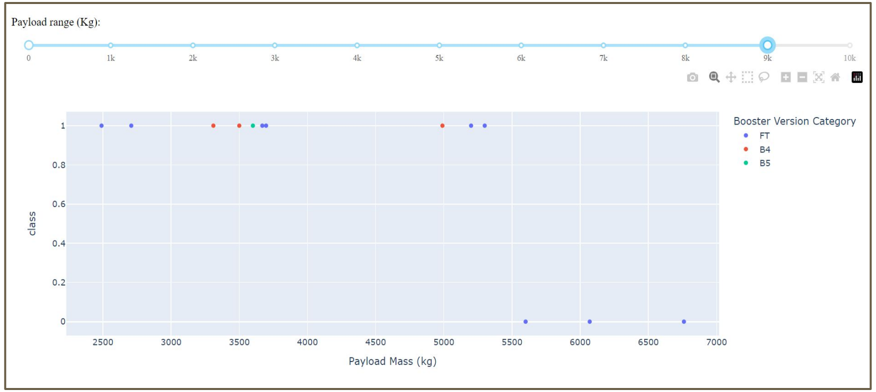
We also observed that the launch sites are close to railway routes and highways. It could be to facilitate easy transportation of large equipments.

Plotly Dash dashboard results









Predictive analysis (classification) results

The main purpose of this exercise is to predict whether first stage of falcon9 will land successfully or not. First we pre-process and standardize the dataset. Then we split the data into test-train samples. Post that we test the performance of Logistic Regression, Support Vector machines, Decision Tree Classifier, and K-nearest neighbors algorithms by training the models, performing grid-search and identifying the one with the best accuracy.

	Model_Name	Score
0	LogisticRegression	0.846429
1	SVM	0.848214
2	DecisionTree	0.887500
3	KNN	0.848214

In [120]:

```
best=model_df[model_df['Score']==max(model_df['Score'])]
best
#print("The Best performing method is ",best['Model_Name'].apply(str)," with a score of ",best['Score'])
```

	Model_Name	Score
2	DecisionTree	0.8875

Conclusions

- The main objective of this project was to identify whether a landing cheaper than what SpaceX is claiming is possible or not. Also, the focus was on the entire data science life cycle right from the retrieval of data till giving a result out. We started initially by understanding how to request data from an API.
- Then the next step was to wrangle the data, clean it and make it standard and easy to use. Post that we did Exploratory data analysis to understand the significance of each variable in the data set.
- Since this data set had some geographical relevance we used Folium to map certain coordinates and understand their relevance with the nearby geography. A plotly dashboard was also created to easily visualize and monitor the results.
- At the end we implemented a few classification algorithms to identify the success rate of the first landing. Among the various methods of classification the best one was identified.

Appendix

Appendix - Data collection methodology

- Request and parse the SpaceX launch data using the GET request

In [9]:
```
static_json_url='https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetw
response1 = requests.get(static_json_url)
```

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

In [10]:
```
response1.status_code
```

```
200
```

Now we decode the response content as a Json using .json() and turn it into a Pandas dataframe.json_normalize ()

In [11]:
```
# Use json_normalize meethod to convert the json result into a dataframe
df1 = response1.json()
df2 = pd.json_normalize(df1)
```

Using the dataframe `data` print the first 5 rows

In [12]:
```
# Get the head of the dataframe
df2.head()
```

	static_fire_date_utc	static_fire_date_unix	tbd	net	window	rocket	success	details	crew	ships	c
0	2006-03-17T00:00:00.000Z	1.142554e+09	False	False	0.0	5e9d0d95eda69955f709d1eb	False	Engine failure at 33 seconds and loss of vehicle	[]	[]	[]

Appendix - Data collection methodology

Filtering the data frame to only include Falcon 9 launches:

In [28]:
```
# Hint data['BoosterVersion']!='Falcon 1'
data_falcon9=dataframe1[dataframe1['BoosterVersion']!='Falcon 1']
```

Now that we have removed some values we should reset the FlgihtNumber column

In [30]:
```
data_falcon9.loc[:,'FlightNumber'] = list(range(1, data_falcon9.shape[0]+1))
data_falcon9
```

```
C:\Users\User\anaconda3\lib\site-packages\pandas\core\indexing.py:1676: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  self._setitem_single_column(ilocs[0], value, pi)
```

	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins	Reused	Legs	
4	1	2010-06-04	Falcon 9	NaN	LEO	CCSFS SLC 40	None None	1	False	False	False	None
5	2	2012-05-22	Falcon 9	525.0	LEO	CCSFS SLC 40	None None	1	False	False	False	None
6	3	2013-03-01	Falcon 9	677.0	ISS	CCSFS SLC 40	None None	1	False	False	False	None
7	4	2013-09-29	Falcon 9	500.0	PO	VAFB SLC 4E	False Ocean	1	False	False	False	None

Appendix - EDA with SQL

Task 1

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

In [8]:
```
%sql select DISTINCT LAUNCH_SITE from SPACEXTBL
```

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

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

Task 2

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

In [9]:
```
%sql select * from SPACEXTBL where launch_site like 'CCA%' limit 5
```

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

Date	Time (UTC)	Booster_Version	Launch_Site	Payload	PAYLOAD_MASS__KG_	Orbit	Customer	Mission_Outcome	Landing _Outcome
04-06-2010	18:45:00	F9 v1.0 B0003	CCAFS LC-40	Dragon Spacecraft Qualification Unit	0	LEO	SpaceX	Success	Failure (parachute)
08-12-2010	15:43:00	F9 v1.0 B0004	CCAFS LC-40	Dragon demo flight C1, two CubeSats, barrel of Brouere	0	LEO (ISS)	NASA (COTS) NRO	Success	Failure (parachute)

Task 3

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

In [11]:
```
%sql select sum(payload_mass__kg_) as sum from SPACEXTBL where customer like 'NASA (CRS)'
```

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

sum
45596

Task 4

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

In [12]:
```
%sql select avg(payload_mass__kg_) as Average from SPACEXTBL where booster_version like 'F9 v1.1%'
```

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

Average
2534.6666666666665

Appendix - EDA with SQL

Task 5

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

Hint:Use min function

```
In [72]: %sql select min(date) as Date from SPACEXTBL where mission_outcome like 'Success'
```

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

Date
01-03-2013

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

```
In [99]: #%sql PRAGMA table_info(SPACEXTBL)
         %sql select booster_version from SPACEXTBL where Landing_Outcome like 'Success (drone ship)' \
          AND PAYLOAD_MASS_KG>4000 AND PAYLOAD_MASS_KG<6000
```

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

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

Task 7

List the total number of successful and failure mission outcomes

```
In [117]: %sql SELECT mission_outcome, count(*) as Count FROM SPACEXTBL GROUP by\
          mission_outcome ORDER BY mission_outcome
```

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

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

Task 8

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

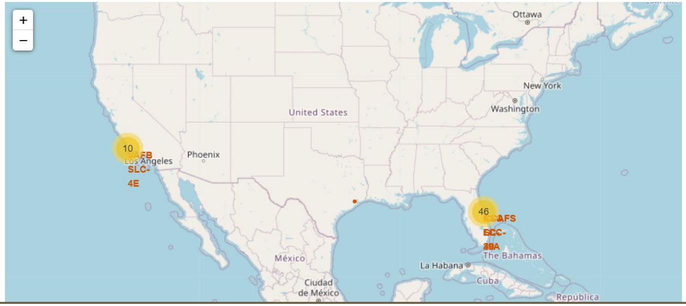
```
In [105]: %sql select booster_version from SPACEXTBL where \
          PAYLOAD_MASS_KG=(select max(PAYLOAD_MASS_KG) from SPACEXTBL)
```

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

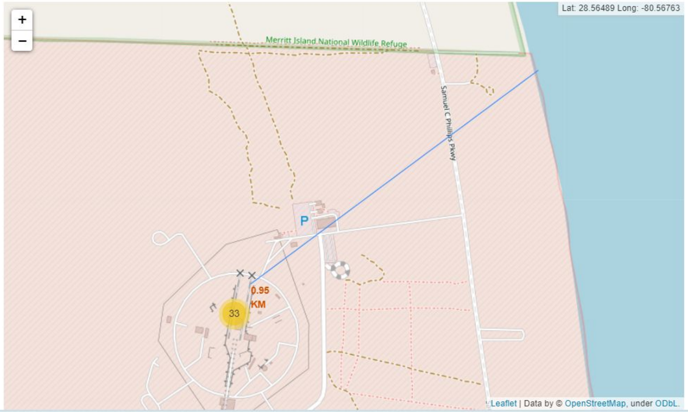
Booster_Version
F9 B5 B1048.4
F9 B5 B1049.4
F9 B5 B1051.3
F9 B5 B1056.4

Appendix - Interactive map with Folium

```
site_map_new
```




```
In [82]: # Create a `folium.PolyLine` object using the coastline coordinates and launch site coordinate
lines=folium.PolyLine(locations=[[coastline_lat,coastline_lon],[launch_site_lat,launch_site_lon]], weight=1
site_map_new.add_child(lines)
```



From the color-labeled markers in marker clusters, we are able to easily identify which launch sites have relatively high success rates.

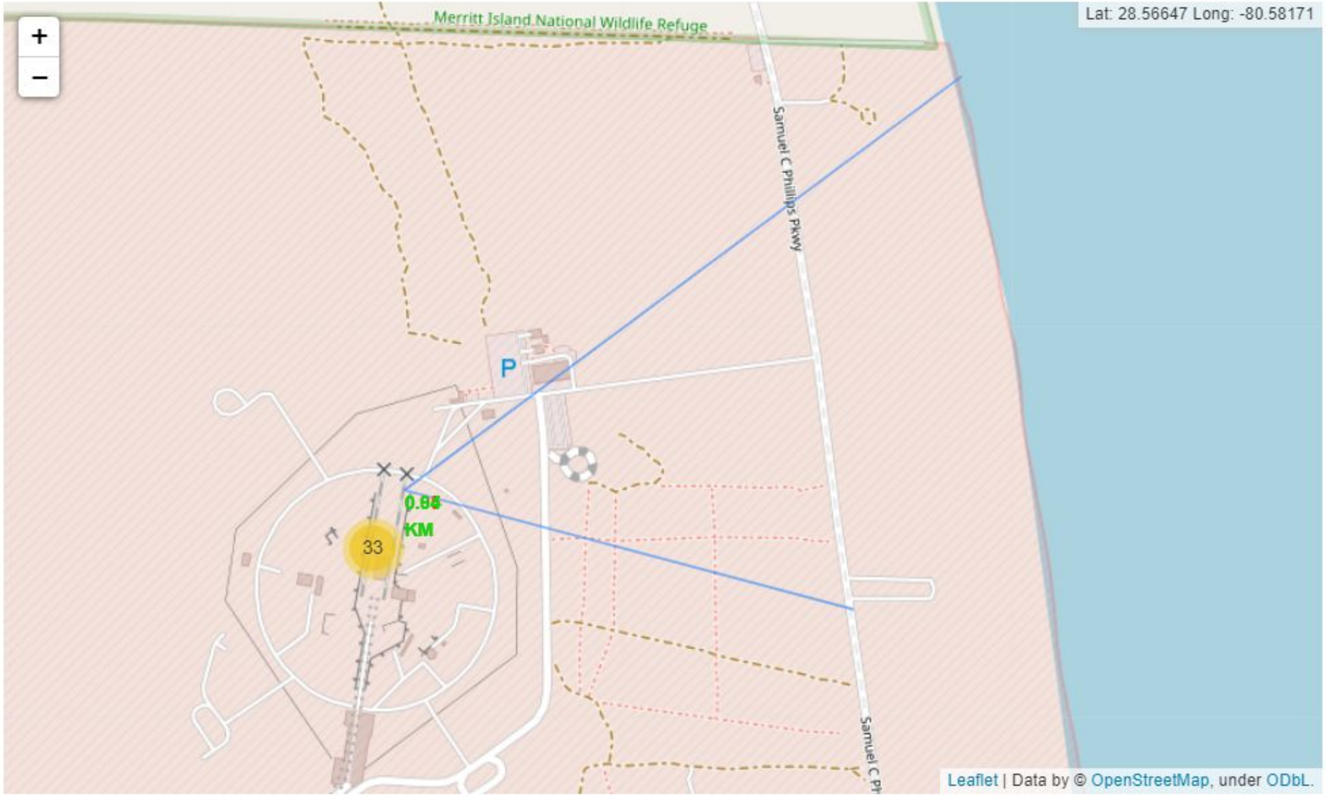
TODO: For each launch result in `spacex_df` data frame, add a `folium.Marker` to `marker_cluster`

```
In [69]: # Add marker_cluster to current site_map
#CONTINUE here
marker_cluster = MarkerCluster()
site_map_new = folium.Map()
site_map_new.add_child(marker_cluster)

# for each row in spacex_df data frame
# create a Marker object with its coordinate
# and customize the Marker's icon property to indicate if this launch was successed or failed,
# e.g., icon=folium.Icon(color='white', icon_color=row['marker_color']
for index,record in spacex_df.iterrows():
        marker=folium.Marker([record['Lat'],record['Long']]
                                 , icon=folium.Icon(color='white',icon_color=record['marker_color']))
        marker_cluster.add_child(marker)
site_map_new
```

Appendix - Interactive map with Folium

```
lines2=folium.PolyLine(locations=[[launch_site_lat,launch_site_lon],[highway_lat,highway_lon]], weight=1)
site_map_new.add_child(lines2)
```



Appendix - 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`, make sure the output is a Pandas series (only one bracket df['name of column']).

```
In [5]: Y=pd.DataFrame.to_numpy(data['Class'])
        Y
```

```
array([0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1,
       1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1,
       1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1], dtype=int64)
```

TASK 2

Standardize the data in `X` then reassign it to the variable `X` using the transform provided below.

```
In [10]: X = preprocessing.StandardScaler().fit(X).transform(X)
         X[0:5]
```

```
array([[-1.71291154e+00, -3.32153339e-17, -6.53912840e-01,
        -1.57589457e+00, -9.73440458e-01, -1.05999788e-01,
        -1.05999788e-01, -6.54653671e-01, -1.05999788e-01,
        -5.51677284e-01,  3.44342023e+00, -1.85695338e-01,
        -3.33333333e-01, -1.05999788e-01, -2.42535625e-01,
        -4.29197538e-01,  7.97724035e-01, -5.68796459e-01,
        -4.10890702e-01, -4.10890702e-01, -1.50755672e-01,
```

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`

```
In [11]: 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)
```

```
Train set: (72, 83) (72,)
Test set: (18, 83) (18,)
```

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`.

```
In [13]: parameters ={'C':[0.01,0.1,1],
                      'penalty':['l2'],
                      'solver':['lbfgs']}
```

```
In [14]: parameters ={"C":[0.01,0.1,1],'penalty':['l2'], 'solver':['lbfgs']}# l1 lasso l2 ridge
         #parameters = [{'max_depth': list(range(10, 15)), 'max_features': list(range(0,14))}]

         lr=LogisticRegression()

         logreg_cv = GridSearchCV(lr, parameters, cv=10,scoring='accuracy')

         logreg_cv.fit(X_train, y_train)

         logreg_cv
```

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

Appendix - Predictive analysis

TASK 5

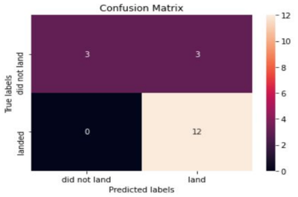
Calculate the accuracy on the test data using the method `score` :

```
In [16]:
#logreg_cv.fit(X_test, y_test)
#logreg_cv.fit(X_train, y_train)
print("accuracy :",logreg_cv.score(X_test, y_test))
```

```
accuracy : 0.8333333333333334
```

Lets look at the confusion matrix:

```
In [17]:
yhat=logreg_cv.predict(X_test)
plot_confusion_matrix(y_test,yhat)
```



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` .

```
In [18]:
parameters = {'kernel':('linear', 'rbf','poly','rbf', 'sigmoid'),
              'C': np.logspace(-3, 3, 5),
              'gamma':np.logspace(-3, 3, 5)}
svm = SVC()
```

```
In [19]:
svm_cv = GridSearchCV(svm, parameters, cv=10, scoring='accuracy')
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')},
             scoring='accuracy')
```

```
In [20]:
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

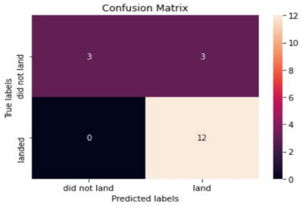
Calculate the accuracy on the test data using the method `score` :

```
In [21]:
#svm_cv.fit(X_test, y_test)
print("accuracy :",svm_cv.score(X_test, y_test))
```

```
accuracy : 0.8333333333333334
```

We can plot the confusion matrix

```
In [22]:
yhat=svm_cv.predict(X_test)
plot_confusion_matrix(y_test,yhat)
```



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` .

```
In [23]:
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()
```

```
In [24]:
tree_cv = GridSearchCV(tree, parameters, cv=10, scoring='accuracy')
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']},
             scoring='accuracy')
```