Rise of Space Y

A Low-cost Reusable Rocket Company created by Allon Mask

Made by Junwei Chen
May 20, 2024

Who are we

- Space Y is a start-up company created by Mr. Allon.E.Mask that focuses on the reusable Rocket development. Different from the famous Space X, who is using the billions of dollars to test their rockets, Space Y will simply use data science to find out a best solution to reuse the rockets. We appreciated what Space X has been done for the rocket industry but Space Y will stand on their shoulders and save tons of the investors' money. From here, you will see the Rise of Space Y.
- In this very first project sponsored by Coursera and IBM, we will use multiple data science methods to understand the data from Space X and figure out the best strategy to enlarge the success rate of each launching and landing.

Outline

- Executive Summary
- Problem Introduction
- Play around with Data
- Show the Results
- Findings and Conclusions

Executive Summary

In this presentation, you will find the following data science executive methods:

- Data collection by API and web scraping
- Data wrangling and cleaning by NumPy, Pandas, and SQL
- Data visualization by static graphs, interactive maps, and also a fancy dashboard
- Data analysis and prediction using machine learning

Problem Introduction

By analyzing the data from Space X, the following problems will be solved:

- What are Space X rocket launching and landing results
- If Space X will reuse the first stage based on the rocket and launching info
- Whether the first stage will land successfully
- Correlation between launch site and the success rate

Now let’s dive into the Data!

Data collection and data wrangling

- Data collection
  - Space X API

Task 1: Request and parse the SpaceX launch data using the GET request

To make the requested JSON results more consistent, we will use the following static response object for this project:

```
[32]: static_json_url='https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNet
```

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

```
[33]: response.status_code
[33]: 200
```

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

```
[81]: # Use json_normalize meethod to convert the json result into a dataframe
data = pd.json_normalize(response.json())
```

Using the dataframe `data` print the first 5 rows

```
[82]: # Get the head of the dataframe
data.head(1)
```

- Web Scraping

TASK 1: Request the Falcon9 Launch Wiki page from its URL

First, let's perform an HTTP GET method to request the Falcon9 Launch HTML page, as an HTTP response.

```
# use requests.get() method with the provided static_url
# assign the response to a object
html_data = requests.get(static_url).text
```

Create a `BeautifulSoup` object from the HTML `response`

```
# Use BeautifulSoup() to create a BeautifulSoup object from a response text content
soup = BeautifulSoup(html_data,"html.parser")
```

Print the page title to verify if the `BeautifulSoup` object was created properly

```
# Use soup.title attribute
tag_object = soup.title
print ("tag object:", tag_object)
```

TASK 2: Extract all column/variable names from the HTML table header

Next, we want to collect all relevant column names from the HTML table header

Let's try to find all tables on the wiki page first. If you need to refresh your memory about `BeautifulSoup`, please check the external reference link towards the end of this lab

```
[38]: # Use the find_all function in the BeautifulSoup object, with element type `table`
# Assign the result to a list called `html_tables`
html_tables = soup.find_all ('table')
```

Starting from the third table is our target table contains the actual launch records.

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

```
colnames = soup.find_all('th')
extract_column_from_header(colnames[1])
```

Data collection and data wrangling

- Data Wrangling

Task 2: Filter the dataframe to only include `Falcon 9` launches

Finally we will remove the Falcon 1 launches keeping only the Falcon 9 launches. Filter the data dataframe using the `BoosterVersion` column to only keep the Falcon 9 launches. Save the filtered data to a new dataframe called `data_falcon9`.

```
# Hint data['BoosterVersion']!='Falcon 1'
data_falcon9 = df_launch[df_launch['BoosterVersion']!= 'Falcon 1']
```

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

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

•••

Task 3: Dealing with Missing Values

```
data_falcon9.isnull().sum()
```

•••

Calculate below the mean for the `PayloadMass` using the `.mean()`. Then use the mean and the `.replace()` function to replace `np.nan` values in the data with the mean you calculated.

```
# Calculate the mean value of PayloadMass column
data_falcon9['PayloadMass'].mean()
# Replace the np.nan values with its mean value
data_falcon9['PayloadMass'].fillna(value=data_falcon9['PayloadMass'].mean(),inplace = True)
```

•••

You should see the number of missing values of the `PayLoadMass` change to zero.

```
data_falcon9.isnull().sum()
```

•••

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

Data collection and data wrangling

This is what we have

- Results from the API

FlightNum	Date	BoosterVe	PayloadM	Orbit	LaunchSite	Outcome	Flights	GridFins	Reused	Legs	LandingPa	Block	ReusedCo	Serial	Longitude	Latitude
1	6/4/2010	Falcon 9	6123.548	LEO	CCSFS SLC	None Non	1	FALSE	FALSE	FALSE		1	0	B0003	-80.5774	28.56186
2	5/22/2012	Falcon 9	525	LEO	CCSFS SLC	None Non	1	FALSE	FALSE	FALSE		1	0	B0005	-80.5774	28.56186
3	3/1/2013	Falcon 9	677	ISS	CCSFS SLC	None Non	1	FALSE	FALSE	FALSE		1	0	B0007	-80.5774	28.56186
4	9/29/2013	Falcon 9	500	PO	VAFB SLC 4	False Ocea	1	FALSE	FALSE	FALSE		1	0	B1003	-120.611	34.63209
5	12/3/2013	Falcon 9	3170	GTO	CCSFS SLC	None Non	1	FALSE	FALSE	FALSE		1	0	B1004	-80.5774	28.56186

- Results from web scraping

Flight No.	Launch sit	Payload	Payload mass	Orbit	Customer	Launch outcome
1	CCAFS	Dragon Spacecraft Qualifi	0	LEO	SpaceX	
2	CCAFS	Dragon	0	LEO		Success
3	CCAFS	Dragon	525 kg	LEO	NASA	Success
4	CCAFS	SpaceX CRS-1	4,700 kg	LEO	NASA	
5	CCAFS	SpaceX CRS-2	4,877 kg	LEO	NASA	
6	VAFB	CASSIOPE	500 kg	Polar orbit	MDA	Success
7	CCAFS	SES-8	3,170 kg	GTO	SES	Success

EDA and interactive visual analytics methodology

EDA using SQL

- Display the names of the unique launch sites in the space mission
- Display 5 records where launch sites begin with the string 'CCA'
- Display the total payload mass carried by boosters launched by NASA (CRS)
- Display average payload mass carried by booster version F9 v1.1
- List the date when the first successful landing outcome in 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
- List the total number of successful and failure mission outcomes
- List the names of the booster_versions which have carried the maximum payload mass. Use a subquery
- 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.
- 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

EDA and interactive visual analytics methodology

Visual analytics

- **Data exploration**
  - **Visualize the relationship between Flight Number and Launch Site**
  - **Visualize the relationship between Payload and Launch Site**
  - **Visualize the relationship between success rate of each orbit type**
  - **Visualize the relationship between FlightNumber and Orbit type**
  - **Visualize the relationship between Payload and Orbit type**
  - **Visualize the launch success yearly trend**
- **Interactive map**
  - **Mark all launch sites on a map**
  - **Mark the success/failed launches for each site on the map**
  - **Calculate the distances between a launch site to its proximities**
- **Interactive dashboard**
  - **Add a Launch Site Drop-down Input Component**
  - **Add a callback function to render success-pie-chart based on selected site dropdown**
  - **Add a Range Slider to Select Payload**
  - **Add a callback function to render the success-payload-scatter-chart scatter plot**

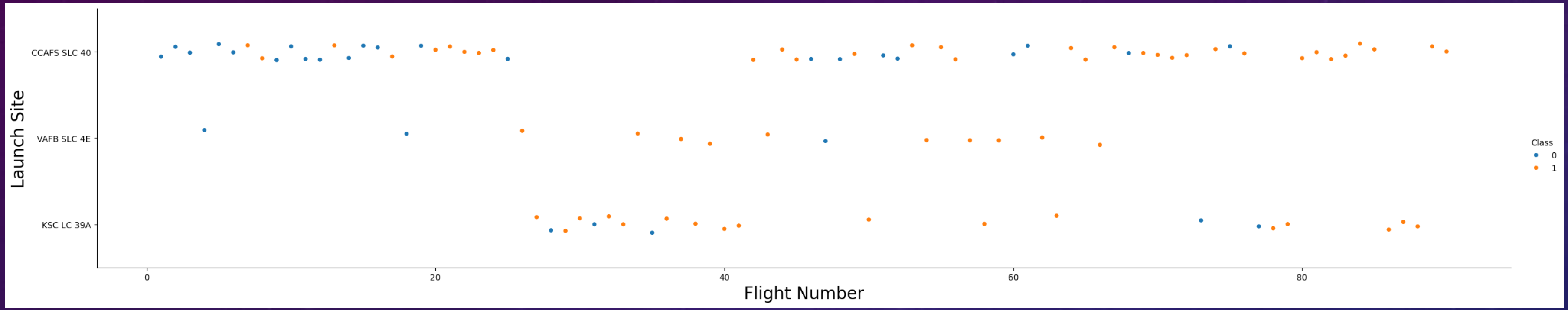
Predictive analysis methodology

- **Create a NumPy array from the column Class in data**
- **Standardize the data in X then reassign it to the variable X using provided transform**
- **Use the function train_test_split to split the data X and Y into training and test data**
- **Using regression object, SVM, decision tree, KNN, respectively to find the best parameters from the dictionary parameters**
- **Calculate the accuracy on the test data for the above models**
- **Find out which model has the best performance**

Check out the Results!

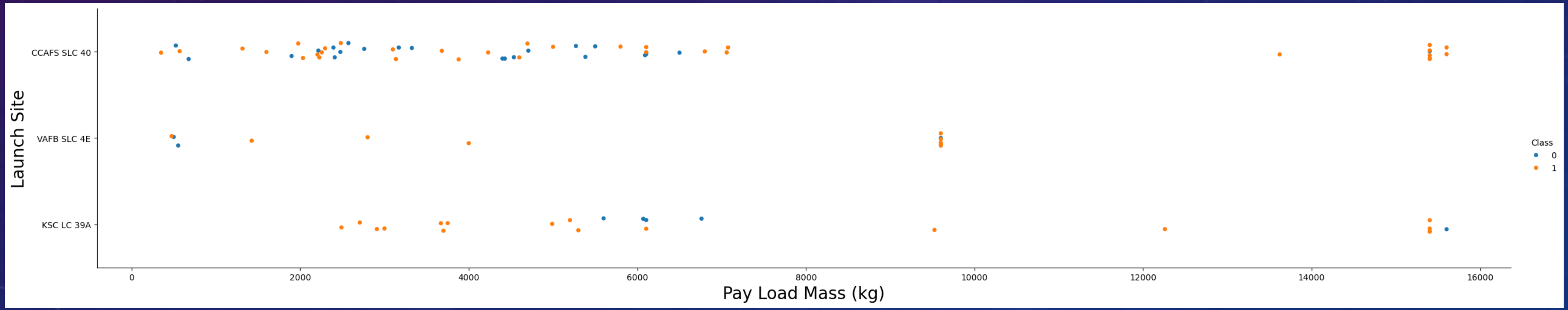
Visualization results

Relationship between Flight Number and Launch Site



- The success rates increase with more flights
- KSC LC 39 A took more flights to be successful than the other types

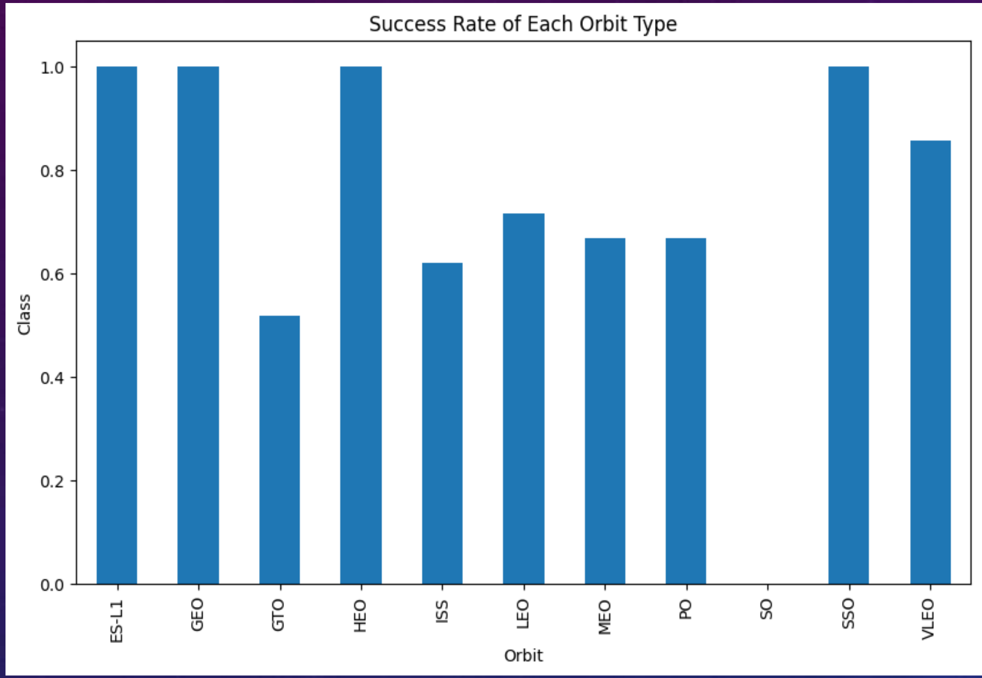
Relationship between Payload and Launch Site



- A large pay load mass can increase the success rate
- Larger pay load mass do not have as many choices as a smaller pay load mass

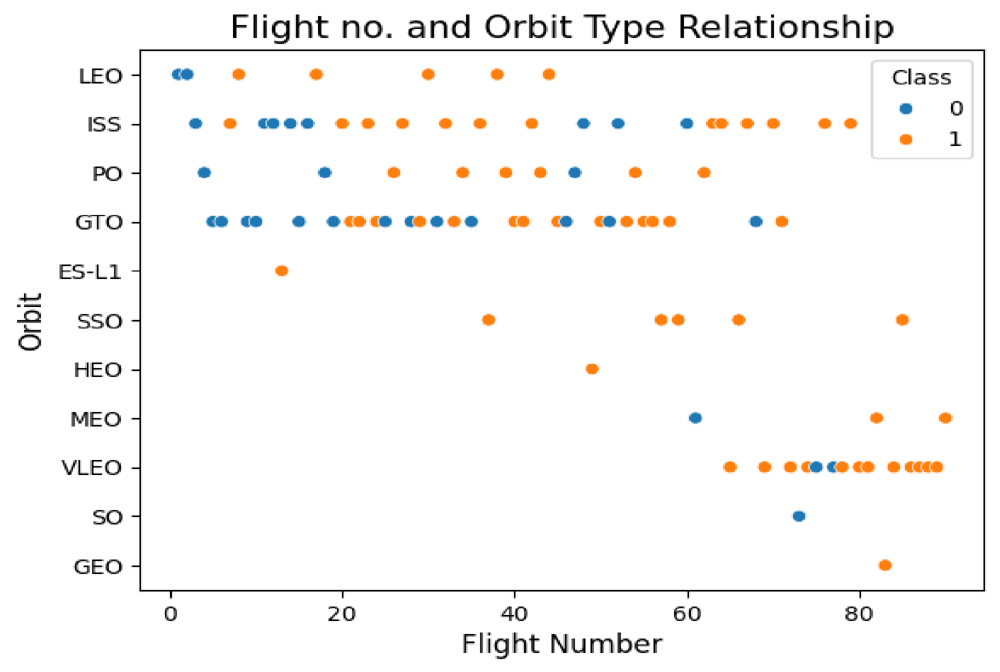
Visualization results

Relationship between success rate of each orbit type



- GTO has the lowest success rate
- ES-L1, GEO, HEO, SSO have 100% success rate
- The average success rate is about 65%. Space X still has a long way to go

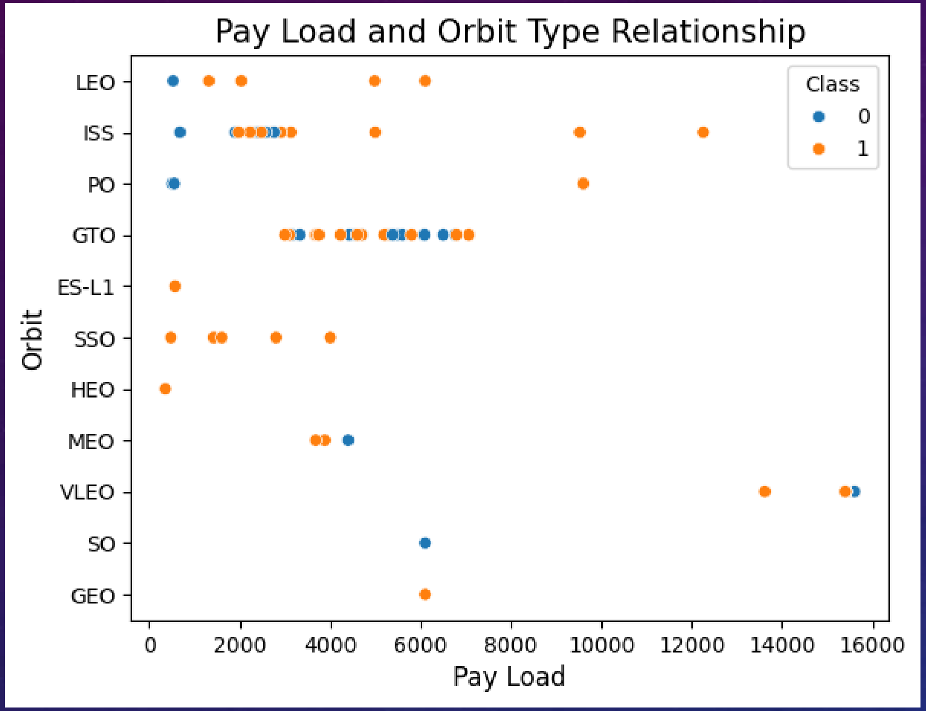
Relationship between FlightNumber and Orbit type



- A large flight number can increase the success rate
- Orbit type like HEO and ES-L1 only have 1 flight number. GTO and ISS have multiple flight numbers

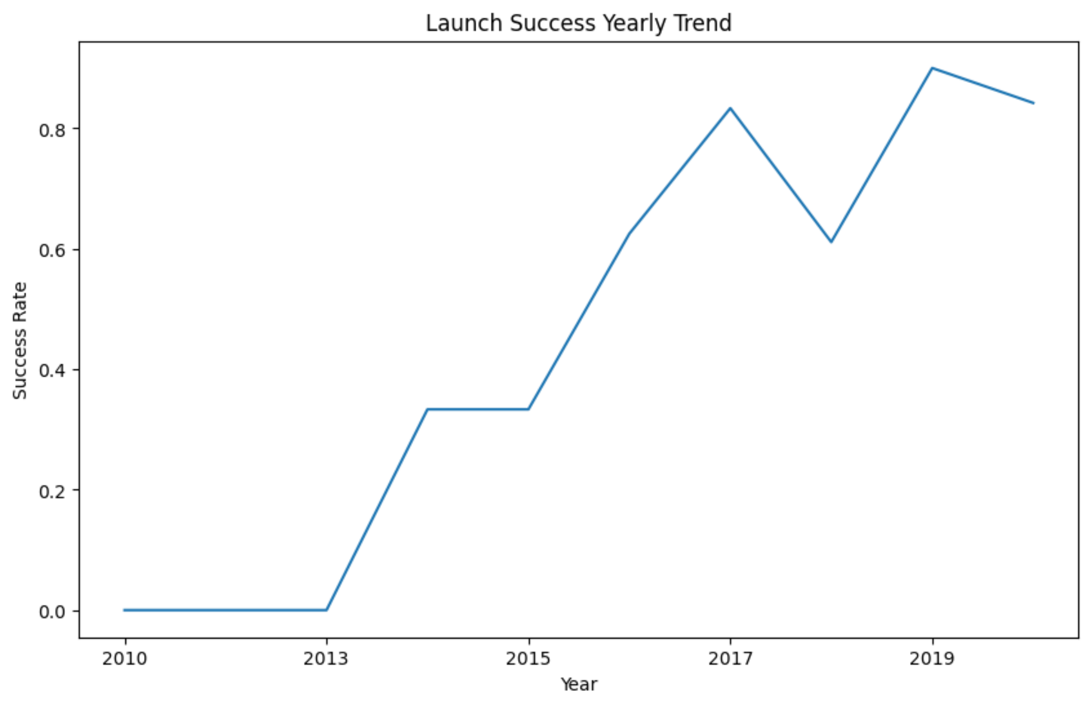
Visualization results

Relationship between Payload and Orbit type



- Success rate of orbit type like ISS, LEO will increase if there is a larger pay load mass
- Results of GTO seem mixture. Pay load mass may not have some influence on that orbit type

Launch success yearly trend



- Generally, the launch success rate increases a lot since 2010.
- There is two drops of the success rate. 2018 and 2020. Probably because of the large n value or some new designs

SQL results

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

```
%%sql
SELECT DISTINCT(Launch_Site) FROM SPACEXTABLE
```

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

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

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

```
%%sql
SELECT * FROM SPACEXTABLE
WHERE Launch_Site like('CCA%')
LIMIT 5
```

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

Date	Time (UTC)	Booster_Version	Launch_Site	Payload
2010-06-04	18:45:00	F9 v1.0 B0003	CCAFS LC-40	Dragon Spacecraft Qualification Unit
2010-12-08	15:43:00	F9 v1.0 B0004	CCAFS LC-40	Dragon demo flight C1, two CubeSats, barrel of Brouere cheese
2012-05-22	7:44:00	F9 v1.0 B0005	CCAFS LC-40	Dragon demo flight C2
2012-10-08	0:35:00	F9 v1.0 B0006	CCAFS LC-40	SpaceX CRS-1
2013-03-01	15:10:00	F9 v1.0 B0007	CCAFS LC-40	SpaceX CRS-2

SQL results

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

```
%%sql
SELECT sum(PAYLOAD_MASS__KG_) FROM SPACEXTABLE
WHERE Customer == "NASA (CRS)"
```

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

sum(PAYLOAD_MASS__KG_)
45596

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

```
%%sql
SELECT AVG(PAYLOAD_MASS__KG_) FROM SPACEXTABLE
WHERE Booster_Version like("F9 v1.1")
```

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

AVG(PAYLOAD_MASS__KG_)
2928.4

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

```
%%sql
SELECT min(Date) FROM SPACEXTABLE
WHERE Landing_Outcome == "Success (ground pad)"
```

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

min(Date)
2015-12-22

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 DISTINCT(Booster_Version) FROM SPACEXTABLE
WHERE PAYLOAD_MASS__KG_ >= 4000 AND
PAYLOAD_MASS__KG_ < 6000 AND Landing_Outcome == 'Success (drone ship)'
```

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

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

SQL results

List the total number of successful and failure mission outcomes

```
%%sql
SELECT Mission_Outcome, count(Mission_Outcome) as counts
FROM SPACEXTABLE group by Mission_Outcome;
```

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

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

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 substr(Date, 6,2) as monthnames
,Landing_Outcome
,Booster_Version
,Launch_Site
FROM SPACEXTABLE
WHERE substr(Date,0,5)='2015' AND Landing_Outcome = "Failure (drone ship)"
```

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

monthnames	Landing_Outcome	Booster_Version	Launch_Site
01	Failure (drone ship)	F9 v1.1 B1012	CCAFS LC-40
04	Failure (drone ship)	F9 v1.1 B1015	CCAFS LC-40

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

```
%%sql
SELECT DISTINCT(Booster_Version) FROM SPACEXTABLE
WHERE PAYLOAD_MASS__KG_ = (SELECT MAX(PAYLOAD_MASS__KG_) FROM SPACEXTABLE)
```

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

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

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

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

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

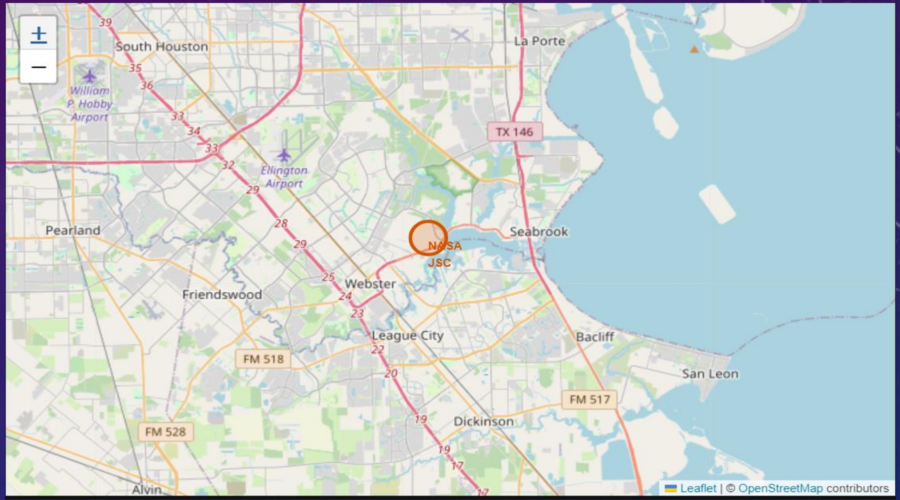
Interactive maps

Mark all launch sites on a map

```python
# Start location is NASA Johnson Space Center
nasa_coordinate = [29.559684888503615, -95.0830971930759]
site_map = folium.Map(location=nasa_coordinate, zoom_start=10)
```

We could use `folium.Circle` to add a highlighted circle area with a text label on a specific coordinate. For example,

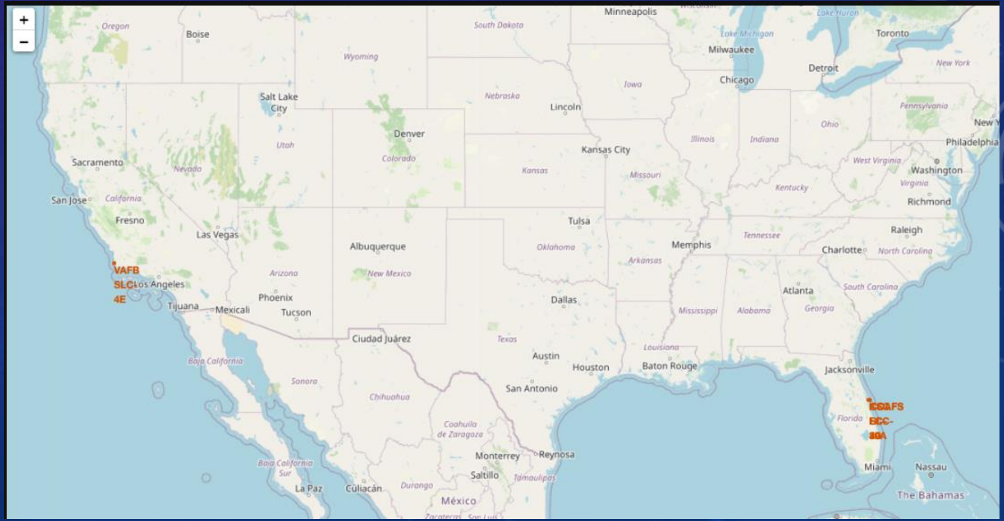
```python
# Create a blue circle at NASA Johnson Space Center's coordinate with a popup label showing its name
circle = folium.Circle(nasa_coordinate, radius=1000,
                        color='#d35400', fill=True).add_child(folium.Popup('NASA Johnson Space Center'))
# Create a blue circle at NASA Johnson Space Center's coordinate with a icon showing its name
marker = folium.map.Marker(
    nasa_coordinate,
    # Create an icon as a text label
    icon=DivIcon(
        icon_size=(20,20),
        icon_anchor=(0,0),
        html='<div style="font-size: 12; color:#d35400;"><b>%s</b></div>' % 'NASA JSC',
        )
    )
site_map.add_child(circle)
site_map.add_child(marker)
```




```python
# Initial the map
site_map = folium.Map(location=nasa_coordinate, zoom_start=5)
# For each launch site, add a Circle object based on its coordinate (Lat, Long) values.
launch_sites_dict_longlat = launch_sites_df.set_index('Launch Site').T.to_dict('list')
launch_sites_dict_longlat
```

```
{'CCAFS LC-40': [28.56230197, -80.57735648],
 'CCAFS SLC-40': [28.56319718, -80.57682003],
 'KSC LC-39A': [28.57325457, -80.64689529],
 'VAFB SLC-4E': [34.63283416, -120.6107455]}
```

```python
for x, y in launch_sites_dict_longlat.items():
    # Create a blue circle at site coordinates with a popup label showing its name
    circle = folium.Circle(y, radius=1000, color='#d35400', fill=True).add_child(folium.
    # Create a blue circle site coordinates with a icon showing its name
    marker = folium.map.Marker(
    y,
    # Create an icon as a text label
    icon=DivIcon(
        icon_size=(20,20),
        icon_anchor=(0,0),
        html='<div style="font-size: 12; color:(#ADD8E6);"><b>%s</b></div>' % x,
        )
    )
    site_map.add_child(circle)
    site_map.add_child(marker)
```



Interactive maps

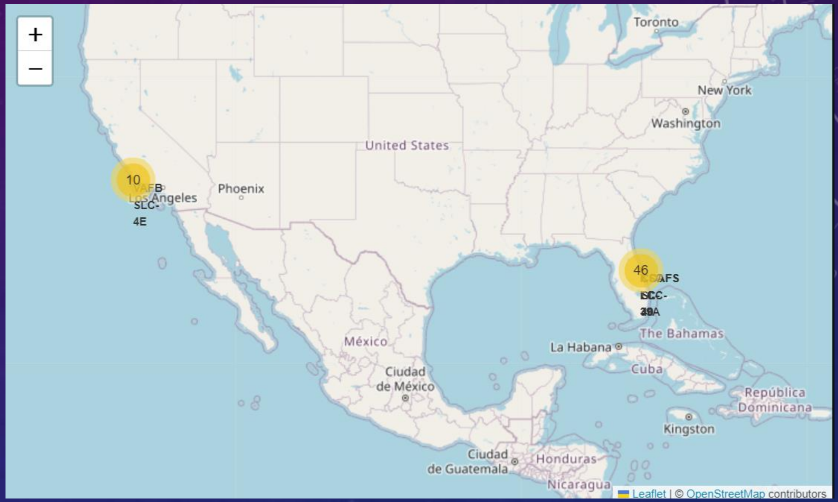
Mark the success/failed launches for each site on the map

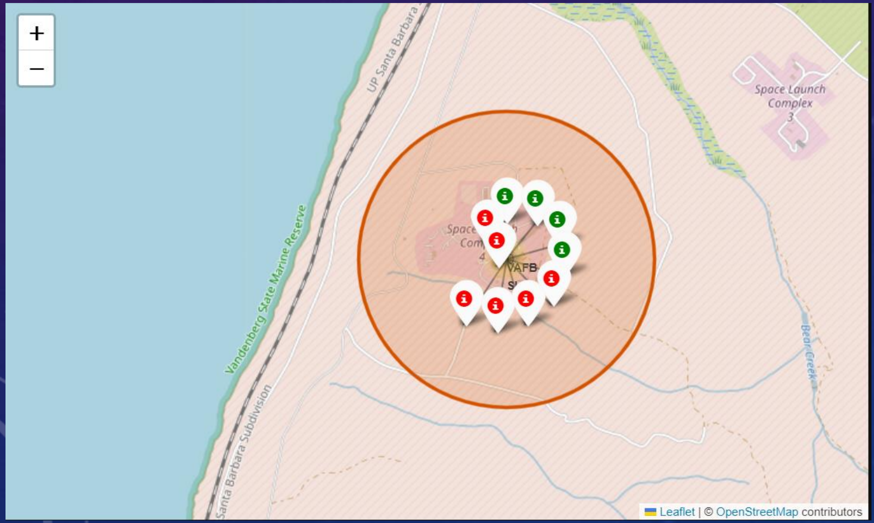
```
def assign_marker_color(launch_outcome):
    if launch_outcome == 1:
        return 'green'
    else:
        return 'red'

spacex_df['marker_color'] = spacex_df['class'].apply(assign_marker_color)
spacex_df
```

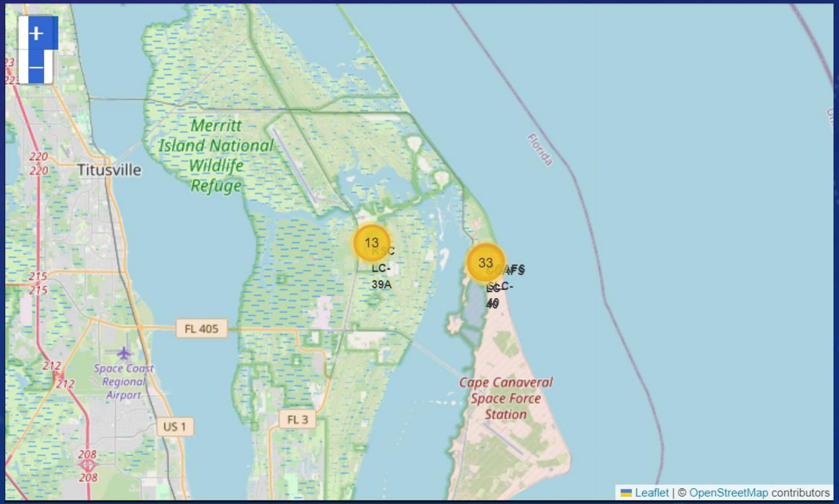
<div> •••

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

```
# Add marker_cluster to current site_map
site_map.add_child(marker_cluster)

for index, record in spacex_df.iterrows():
    # TODO: Create and add a Marker cluster to the site map
    # marker = folium.Marker(...)
    marker = folium.Marker([record[1], record[2]], icon=folium.Icon(color='white', icon_color=record[4]))
    marker_cluster.add_child(marker)

site_map
```







Interactive maps

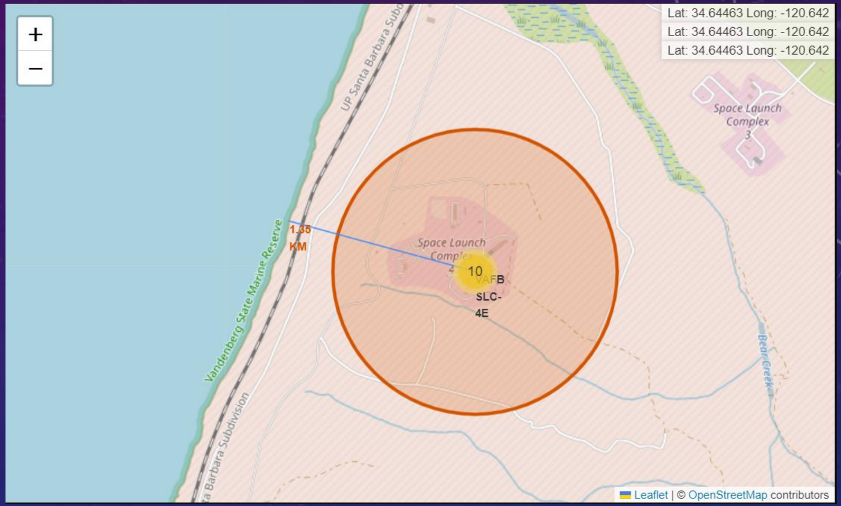
Calculate the distances between a launch site to its proximities

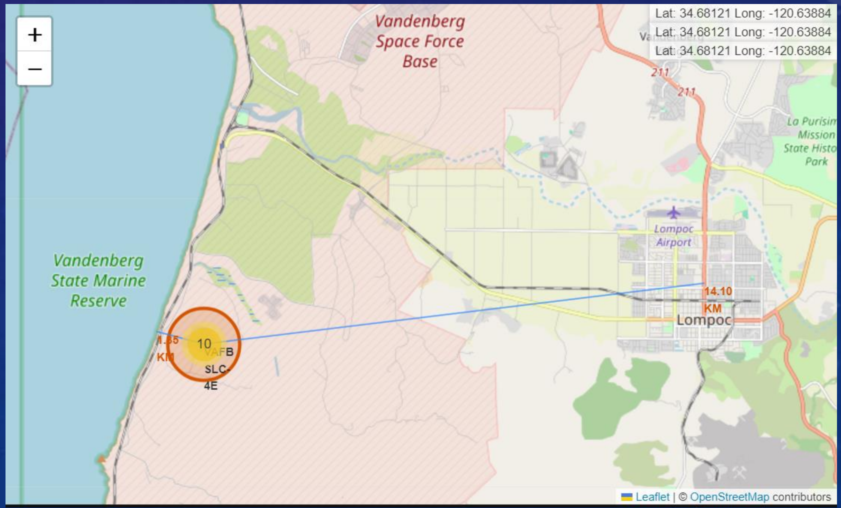
```python
from math import sin, cos, sqrt, atan2, radians

def calculate_distance(lat1, lon1, lat2, lon2):
    # approximate radius of earth in km
    R = 6373.0

    lat1 = radians(lat1)
    lon1 = radians(lon1)
    lat2 = radians(lat2)
    lon2 = radians(lon2)

    dlon = lon2 - lon1
    dlat = lat2 - lat1

    a = sin(dlat / 2)**2 + cos(lat1) * cos(lat2) * sin(dlon / 2)**2
    c = 2 * atan2(sqrt(a), sqrt(1 - a))

    distance = R * c
    return distance
```

```python
# find coordinate of the closet coastline
# e.g.,: Lat: 28.56367  Lon: -80.57163
# distance_coastline = calculate_distance(launch_site_lat, launch_site_lon, coastline_lat, coastl
distance_coastline = calculate_distance(34.63602, -120.625, 34.632834, -120.610745)
distance_marker = folium.Marker([34.63602, -120.625], icon=DivIcon(icon_size=(20,20), icon_anchor
    html='<div style="font-size: 12; color:#d35400;"><b>%s</b></div>' % "{:10.2f} KM".format(dist
```

TODO: Draw a `PolyLine` between a launch site to the selected coastline point

```python
# Create a `folium.PolyLine` object using the coastline coordinates and launch site coordinate
# lines=folium.PolyLine(locations=coordinates, weight=1)
lines=folium.PolyLine(locations=[[34.63602, -120.625], [34.632834, -120.610745]], weight=1)
site_map.add_child(distance_marker)
site_map.add_child(lines)
```

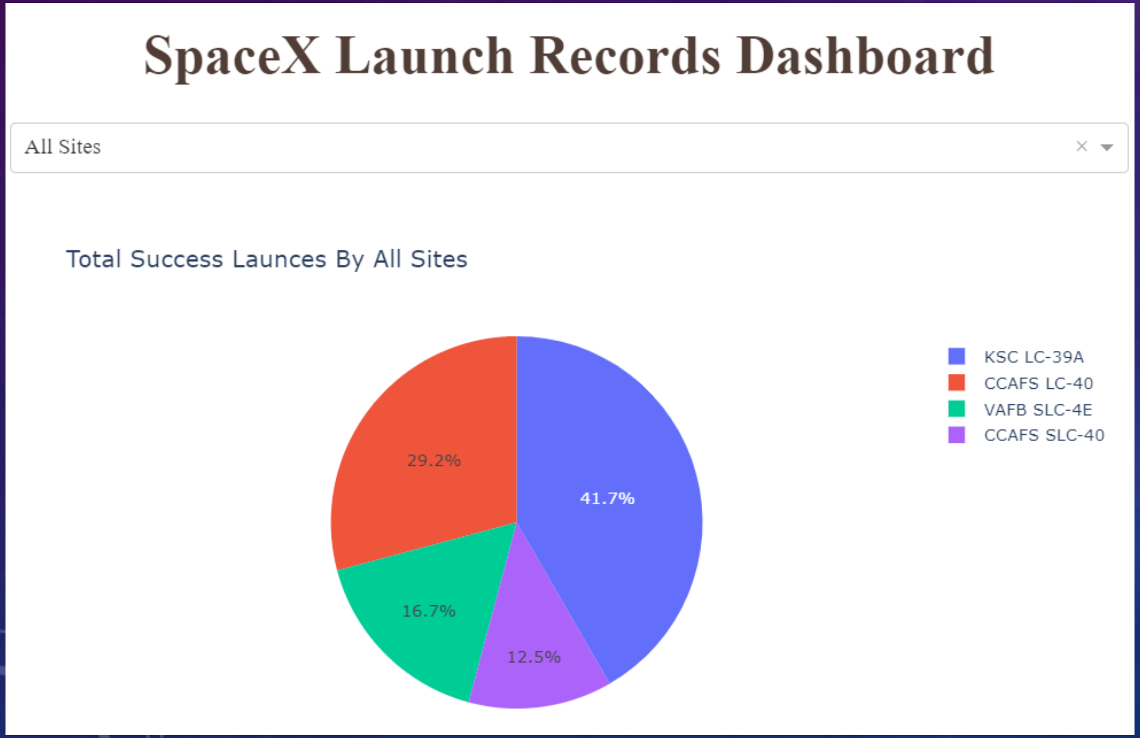




Interactive dashboard

- Add a Launch Site Drop-down Input Component
- Add a callback function to render success-pie-chart based on selected site dropdown

- Add a Range Slider to Select Payload
- Add a callback function to render the success-payload-scatter-chart scatter plot



Predictive analysis

- 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'])

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

- 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.

```
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)
```

we can see we only have 18 test samples.

```
Y_test.shape
```

```
(18,)
```

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

```
# students get this
transform = preprocessing.StandardScaler()
X= preprocessing.StandardScaler().fit(X).transform(X)
X
```

```
array([[-1.71291154e+00, -5.29526321e-17, -6.53912840e-01, ...,
        -8.35531692e-01,  1.93309133e+00, -1.93309133e+00],
       [-1.67441914e+00, -1.19523159e+00, -6.53912840e-01, ...,
        -8.35531692e-01,  1.93309133e+00, -1.93309133e+00],
       [-1.63592675e+00, -1.16267307e+00, -6.53912840e-01, ...,
        -8.35531692e-01,  1.93309133e+00, -1.93309133e+00],
       ...,
       [ 1.63592675e+00,  1.99100483e+00,  3.49060516e+00, ...,
         1.19684269e+00, -5.17306132e-01,  5.17306132e-01],
       [ 1.67441914e+00,  1.99100483e+00,  1.00389436e+00, ...,
         1.19684269e+00, -5.17306132e-01,  5.17306132e-01],
       [ 1.71291154e+00, -5.19213966e-01, -6.53912840e-01, ...,
        -8.35531692e-01, -5.17306132e-01,  5.17306132e-01]])
```

Predictive analysis

- **Find the accuracy, score, and confusion map using the Logistic Regression**

```
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)
```

```
<style>#sk-container-id-1 {color: black;}#sk-container-id-1 pre{padding: 0;}#sk-container-id-1
```

We output the `GridSearchCV` object for logistic regression. We display the best parameters using the data a
data using the data attribute `best_score_`.

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

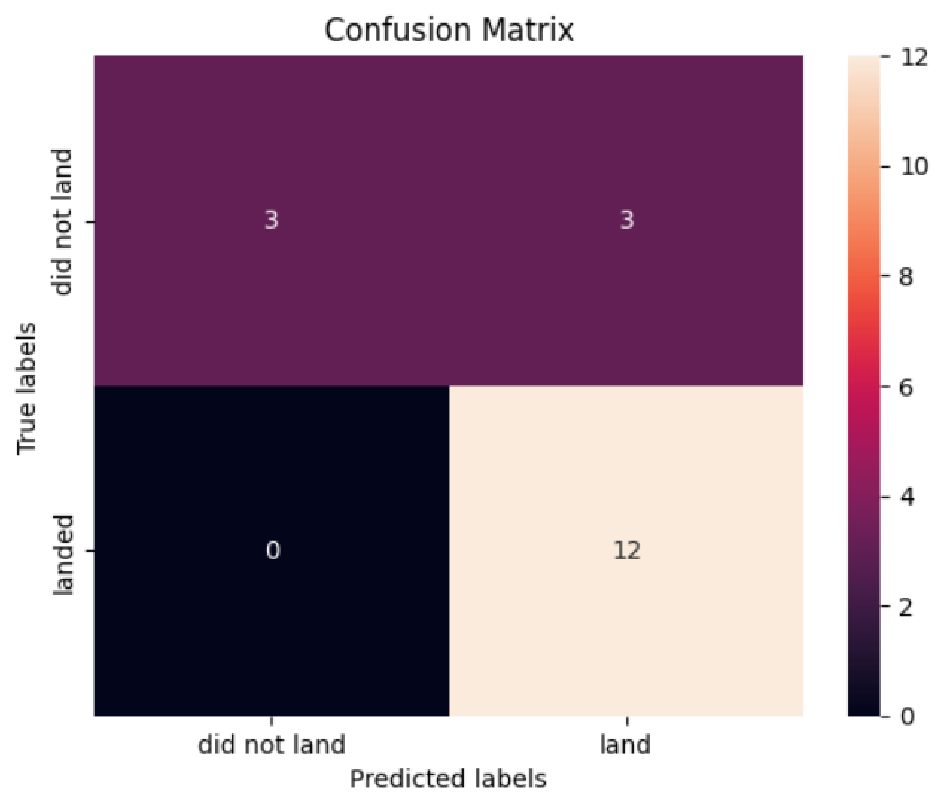
```
tuned hpyerparameters :(best parameters)  {'C': 0.01, 'penalty': 'l2', 'solver': 'lbfgs'}
accuracy : 0.8464285714285713
```

```
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)
```



Predictive analysis

- **Find the accuracy, score, and confusion map using the SVM**

```
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)
```

```
[0;31m---------------------------------------------------------------------------[0m •••
```

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

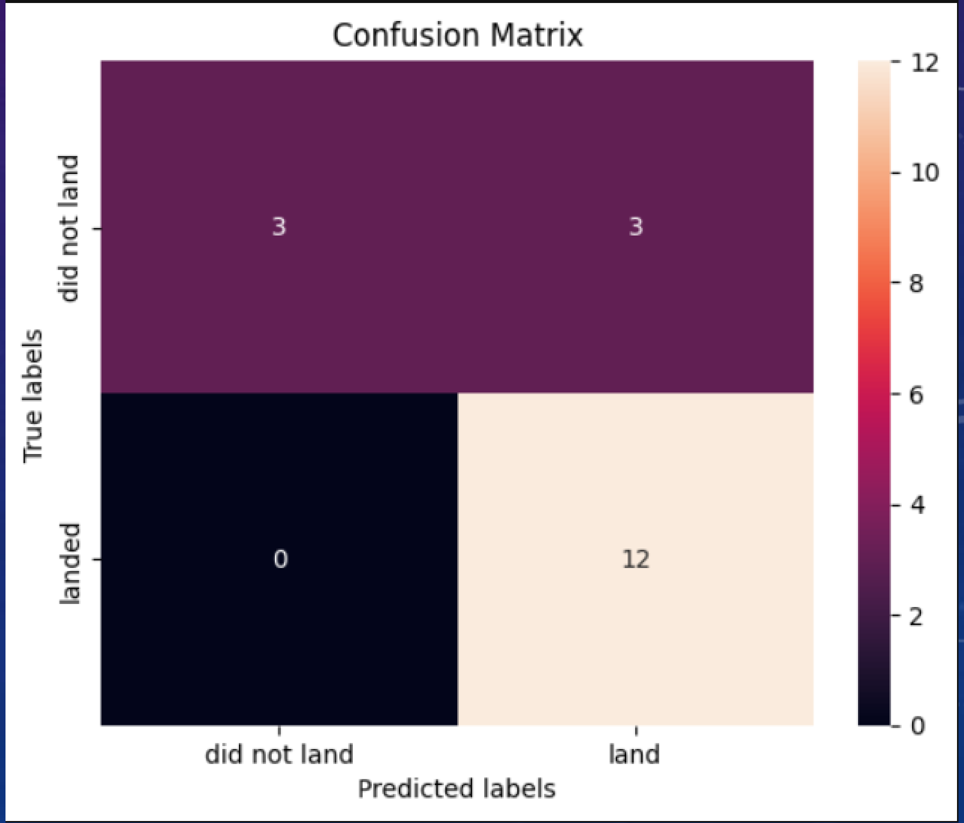
```
tuned hpyerparameters :(best parameters)  {'C': 1.0, 'gamma': 0.03162277660168379, 'kernel': 'sigmoid'}
accuracy : 0.8482142857142856
```

```
print("Test data accuracy score - SVM: ", svm_cv.score(X_test, Y_test))
```

```
Test data accuracy score - SVM:  0.8333333333333334
```

We can plot the confusion matrix

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



Predictive analysis

- **Find the accuracy, score, and confusion map using the Decision Tree**

```
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)
```

```
/lib/python3.11/site-packages/sklearn/model_selection/_validation.py:425: FitFailedWarning: •••
```

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

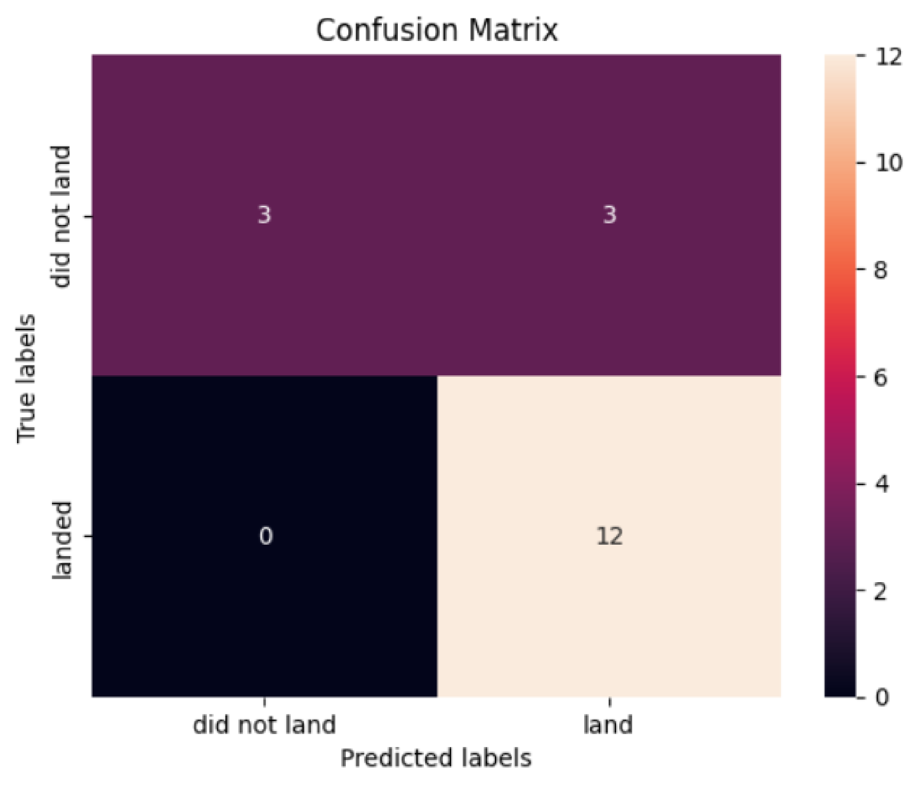
```
tuned hpyerparameters :(best parameters)  {'criterion': 'entropy', 'max_depth': 4, 'max_features'
t': 10, 'splitter': 'random'}
accuracy : 0.875
```

```
print("Test data accuracy score - Decision Tree: ", tree_cv.score(X_test, Y_test))
```

```
Test data accuracy score - Decision Tree:  0.8333333333333334
```

We can plot the confusion matrix

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



Predictive analysis

- **Find the accuracy, score, and confusion map using the KNN**

```
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)
```

```
/lib/python3.11/site-packages/threadpoolctl.py:1019: RuntimeWarning: libc not found. The ctype
```

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

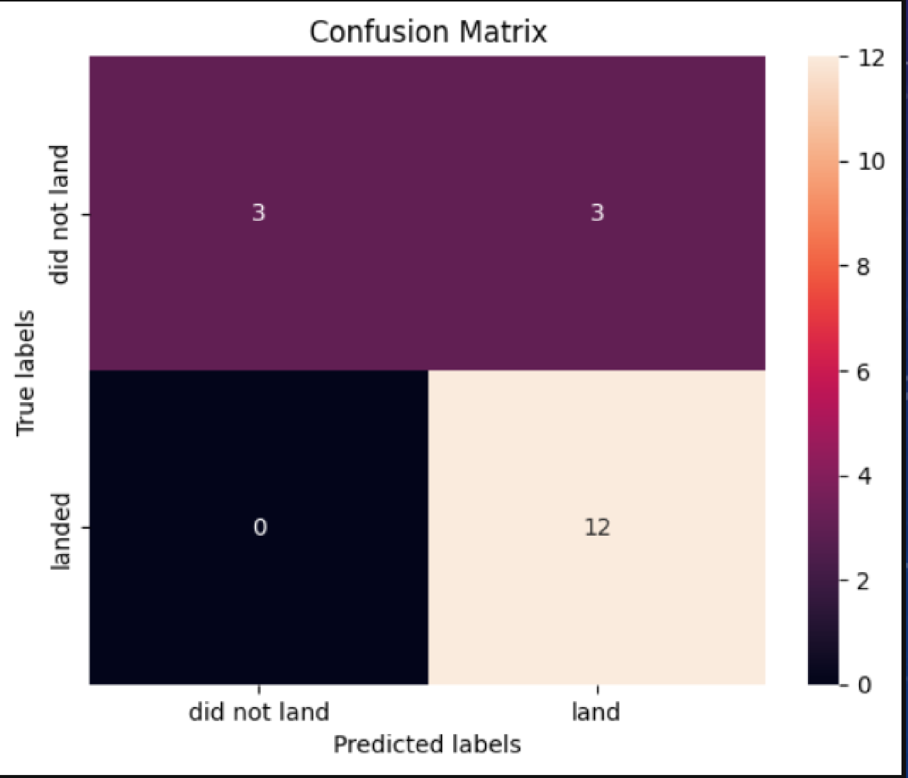
```
tuned hpyerparameters :(best parameters)  {'algorithm': 'auto', 'n_neighbors': 10, 'p': 1}
accuracy : 0.8482142857142858
```

```
print("Test data accuracy score - KNN: ", knn_cv.score(X_test, Y_test))
```

```
Test data accuracy score - KNN:  0.8333333333333334
```

We can plot the confusion matrix

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



Predictive analysis

- **Models comparison**

```
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
```

	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

All of the four models have the same test data accuracy score. But Decision Tree has the highest accuracy score, and Logistic Regression has the lowest accuracy score.

Conclusion

Conclusion and findings

- Success rates appear go up as Payload increases but there is no clear correlation between Payload mass and success rates
- As the numbers of flights increase, the first stage is more likely to land successfully
- Launch success rate increased by about 80% from 2013 to 2020
- Orbits ES-L1, GEO, HEO, and SSO have the highest launch success rates and orbit GTO the lowest
- Lunch sites are located strategically away from the cities and closer to coastline, railroads, and highways
- The best performing Machine Learning Classfication Model is the Decision Tree with an accuracy of about 87.5%. When the models were scored on the test data, the accuracy score was about 83% for all models. More data may be needed to further tune the models and find a potential better fit.

Thank you for choosing Space Y