SpaceX Project

Akshara Gade

Contents

Executive Summary

Data Collection and Wrangling

The data was collected from the SpaceX Falcon 9 rocket launches API and prepared to begin with the data analysis.

Exploratory Data Analysis

EDA and Feature Engineering using SQL and Python was performed to visualise the relationships between the variables.

Interactive Data Analysis

Interactive maps for visual analysis of the Falcon 9 launches were created using Folium..

Machine Learning Prediction

A predictive analysis model was created using classification.

Introduction

SpaceX is a commercial space company that specialises in advancing rocket propulsion, resuable launch vehicles, human spaceflight, and satellite constellation technology.
With a vision to reduce launch costs, SpaceX designed the Falcon 9 that launches with a cost of 62 milllion dollars (whereas other providers cost more than 192 million dollars) by creating the first stage to be resuable.
Hence, we can predict the cost of each launch depending on whether the first stage can land.
This project develops a machine learning model to predict whether the first stage of the Falcon 9 can land successfully.

Link to Jupyter Notebook

https://github.com/bluedreamyyxy/IBM-Data-Science/blob/main/IBM%20Data%20Science%20Project.ipynb

Data Collecting

Using the SpaceX API

1. Send a request to the API for the required data.
2. Create a database using json.normalise().
3. Create a dictionary and then a dataframe.
4. Filter the values to include only the Falcon 9 launches.
5. Replace any missing data with the mean of the remaining datapoints.

Using web-scraping

1. Request the data from an online website.
2. Create the BeautifulSoup object.
3. Extract all the column names.
4. Parse the data into the columns.
5. Create a dictionary, and then create a dataframe.

Data Wrangling

Since our main aim is to find out whether a Falcon 9 launch lands successfully or not, we use this section to look into the variables which describes this the best.
By looking into the Landing_Outcomes, we see there are 8 different outcomes on both land and ocean which are either a planned landing site(True) or a unexpected landing site(False).
For this, we create a new column called ‘Class’ with the values 1(True) for successful landing and 0(False) for unsuccessful landing outcome.

[40]:
```
# To calculate the number and occurence of mission outcome of the orbits.
landing_outcomes = data['Outcome'].value_counts()
landing_outcomes
```

[40]:
```
Outcome
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: count, dtype: int64
```

[42]:
```
for i,outcome in enumerate(landing_outcomes.keys()):
    print(i,outcome)
```

```
0 True ASDS
1 None None
2 True RTLS
3 False ASDS
4 True Ocean
5 False Ocean
6 None ASDS
7 False RTLS
```

To create a landing outcome label from Outcome column

landing_class = [] for outcome in data['Outcome']: if outcome in bad_outcomes: landing_class.append(0) else: landing_class.append(1)

[54]:
```
data['Class']=landing_class
data[['Class']].head(8)
```

[54]:

	Class
0	0
1	0
2	0
3	0
4	0
5	0
6	1
7	1

[58]:
```
data["Class"].mean()
```

[58]:
```
0.6666666666666666
```

Exploratory Data Analysis with SQL

SQL helps us to understand the information in the dataset a little more clearly, as visual tables. We can connect to the SQL server using sqlalchemy and prettytable.

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

```
[34]: %%sql
      SELECT DISTINCT Launch_Site
      FROM SPACEXTABLE;

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

[34]:

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

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

```
[45]: %%sql
      SELECT SUM(PAYLOAD_MASS__KG_)
      FROM SPACEXTABLE
      WHERE Launch_Site = 'NASA (CRS)';

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

[45]:

SUM(PAYLOAD_MASS__KG_)
None

Exploratory Data Analysis with SQL

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

```
[48]: %%sql
      SELECT AVG(PAYLOAD_MASS__KG_)
      FROM SPACEXTABLE
      WHERE Booster_Version LIKE 'F9 v1.0%';

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

[48]: AVG(PAYLOAD_MASS__KG_)
340.4

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

```
[51]: %%sql
      SELECT MIN(Date)
      FROM SPACEXTABLE
      WHERE Landing_Outcome = 'Success (ground pad)';

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

[51]: MIN(Date)
2015-12-22

Exploratory Data Analysis with SQL

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

```
[54]: %%sql
SELECT Booster_Version
FROM SPACEXTABLE
WHERE Landing_Outcome = 'Success (drone ship)'
AND 4000 < Payload_Mass__Kg_ < 6000;
```

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

[54]:

Booster_Version
F9 FT B1021.1
F9 FT B1022
F9 FT B1023.1
F9 FT B1026
F9 FT B1029.1
F9 FT B1021.2
F9 FT B1029.2
F9 FT B1036.1
F9 FT B1038.1
F9 B4 B1041.1
F9 FT B1031.2
F9 B4 B1042.1
F9 B4 B1045.1
F9 B5 B1046.1

6. List the total number of successful and failure mission outcomes.

```
[57]: %%sql
SELECT Mission_Outcome, Count(Mission_Outcome) AS Total_Number
FROM SPACEXTABLE
GROUP BY Mission_Outcome;
```

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

[57]:

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

Exploratory Data Analysis with SQL

7. List all the booster_versions that have carried the maximum payload mass.

```
[60]: %%sql
SELECT DISTINCT Booster_Version
FROM SPACEXTABLE
WHERE PAYLOAD_MASS__KG_ = (
    SELECT MAX(PAYLOAD_MASS__KG_)
    FROM SPACEXTABLE);
```

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

[60]:

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

8. 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

```
[69]: %%sql
SELECT LANDING_OUTCOME, COUNT(LANDING_OUTCOME) AS TOTAL_NUMBER
FROM SPACEXTABLE
WHERE DATE BETWEEN '2010-06-04' AND '2017-03-20'
GROUP BY LANDING_OUTCOME
ORDER BY TOTAL_NUMBER DESC
```

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

[69]:

Landing_Outcome	TOTAL_NUMBER
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

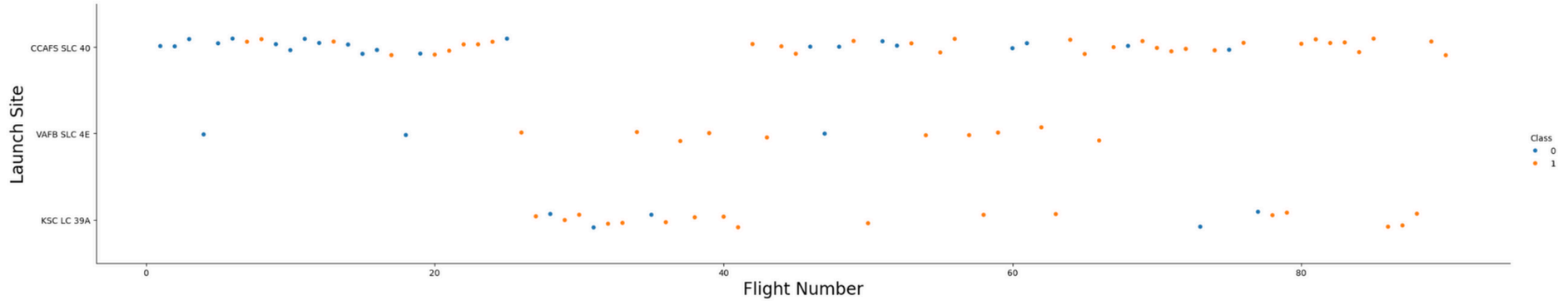
Exploratory Data Analysis with Data Visualisation

We can now use the Pandas and Seaborn libraries for data visualisation in EDA to get a clearer view of the data.
By using feature engineering, we can gain some preliminary insights into how the different features of the launch effect the landing.

EDA with Data Visualisation

1. Visualize the relationship between Flight Number and Launch Site.

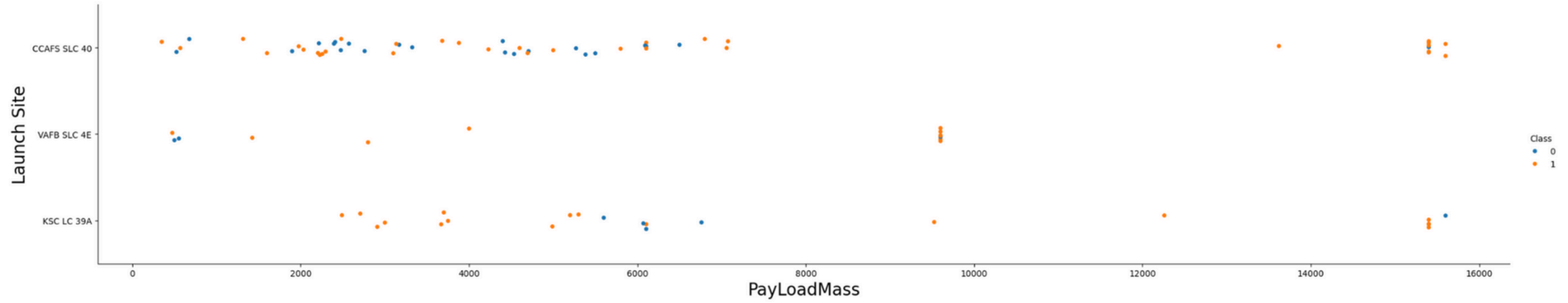
```
sns.catplot(x="FlightNumber", y="LaunchSite", hue = "Class", data=data, aspect=5)
plt.xlabel("Flight Number", fontsize=20)
plt.ylabel("Launch Site", fontsize=20)
plt.show()
```



EDA with Data Visualisation

2. Visualize the relationship between Payload Mass and Launch Site.

```
sns.catplot(x="PayloadMass", y="LaunchSite", hue="Class", data=data, aspect =5)
plt.xlabel("PayLoadMass", fontsize=20)
plt.ylabel("Launch Site", fontsize=20)
plt.show()
```



EDA with Data Visualisation

3. Visualize the relationship between success rate of each orbit type

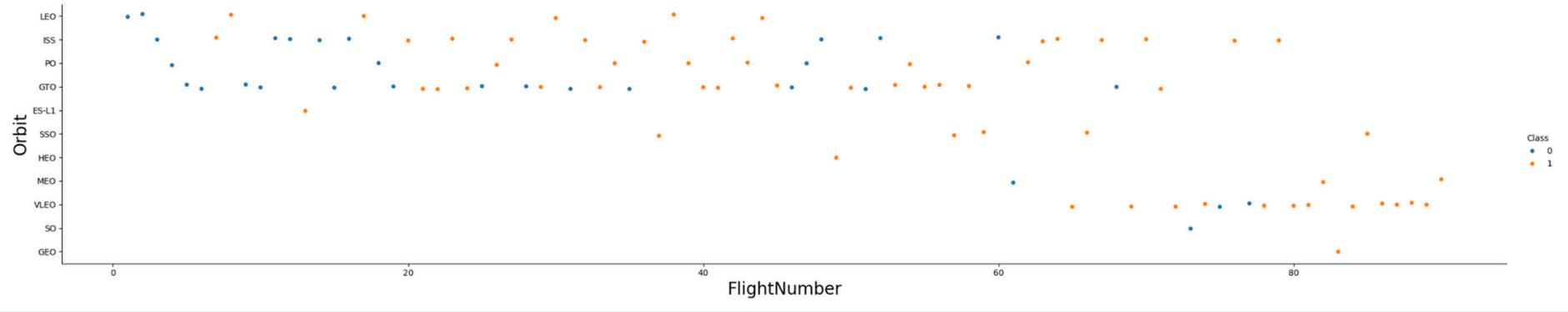
```
y = data['Orbit']
x= data['Outcome']

data_rank = data.groupby("Orbit").count()

data_rank.head()
```

Orbit	FlightNumber	Date	BoosterVersion	PayloadMass	LaunchSite	Outcome	Flights	GridFins	Reused	Legs	LandingPad	Block	ReusedCount	Serial
ES-L1	1	1	1	1	1	1	1	1	1	1	0	1	1	1
GEO	1	1	1	1	1	1	1	1	1	1	1	1	1	1
GTO	27	27	27	27	27	27	27	27	27	27	15	27	27	27
HEO	1	1	1	1	1	1	1	1	1	1	1	1	1	1
ISS	21	21	21	21	21	21	21	21	21	21	16	21	21	21

EDA with Data Visualisation

4. Visualize the relationship between FlightNumber and Orbit type.

```
sns.catplot(x="FlightNumber", y="Orbit", hue="Class", data=data, aspect = 5)
plt.xlabel("FlightNumber", fontsize=20)
plt.ylabel("Orbit", fontsize=20)
plt.show()
```



EDA with Data Visualisation

5. Visualize the relationship between Payload Mass and Orbit type.

```
sns.catplot(x="PayloadMass", y="Orbit", hue="Class", data=data, aspect = 5)
plt.xlabel("PayLoadMass", fontsize=20)
plt.ylabel("Orbit", fontsize=20)
plt.show()
```

EDA with Data Visualisation

Now, we will move on to Feature Engineering, where we will select the Features that would be most useful in predicting the machine learning model.

```
features = data[['FlightNumber', 'PayloadMass', 'Orbit', 'LaunchSite', 'Flights', 'GridFins', 'Reused', 'Legs', 'LandingPad', 'Block', 'Reus
features.head()
```

	FlightNumber	PayloadMass	Orbit	LaunchSite	Flights	GridFins	Reused	Legs	LandingPad	Block	ReusedCount	Serial
0	1	6104.959412	LEO	CCAFS SLC 40	1	False	False	False	NaN	1.0	0	B0003
1	2	525.000000	LEO	CCAFS SLC 40	1	False	False	False	NaN	1.0	0	B0005
2	3	677.000000	ISS	CCAFS SLC 40	1	False	False	False	NaN	1.0	0	B0007
3	4	500.000000	PO	VAFB SLC 4E	1	False	False	False	NaN	1.0	0	B1003
4	5	3170.000000	GTO	CCAFS SLC 40	1	False	False	False	NaN	1.0	0	B1004

```
features_one_hot.astype('float64')
```

	FlightNumber	PayloadMass	Flights	GridFins	Reused	Legs	Block	ReusedCount	Orbit_ES-L1	Orbit_GEO	...	Serial_B1048	Serial_B1049	Serial_B1050
0	1.0	6104.959412	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	...	0.0	0.0	0.0
1	2.0	525.000000	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	...	0.0	0.0	0.0
2	3.0	677.000000	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	...	0.0	0.0	0.0
3	4.0	500.000000	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	...	0.0	0.0	0.0
4	5.0	3170.000000	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	...	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
85	86.0	15400.000000	2.0	1.0	1.0	1.0	5.0	2.0	0.0	0.0	...	0.0	0.0	0.0
86	87.0	15400.000000	3.0	1.0	1.0	1.0	5.0	2.0	0.0	0.0	...	0.0	0.0	0.0
87	88.0	15400.000000	6.0	1.0	1.0	1.0	5.0	5.0	0.0	0.0	...	0.0	0.0	0.0
88	89.0	15400.000000	3.0	1.0	1.0	1.0	5.0	2.0	0.0	0.0	...	0.0	0.0	0.0
89	90.0	3681.000000	1.0	1.0	0.0	1.0	5.0	0.0	0.0	0.0	...	0.0	0.0	0.0

90 rows × 80 columns

Interactive Visual Analysis

The launch depends on many factors, including location and proximity to a launch site.
We now will find the optimal launchsite using interactive visual analysis using Folium.

Interactive Visual Analysis

Task 1: Mark all launch sites on a map.

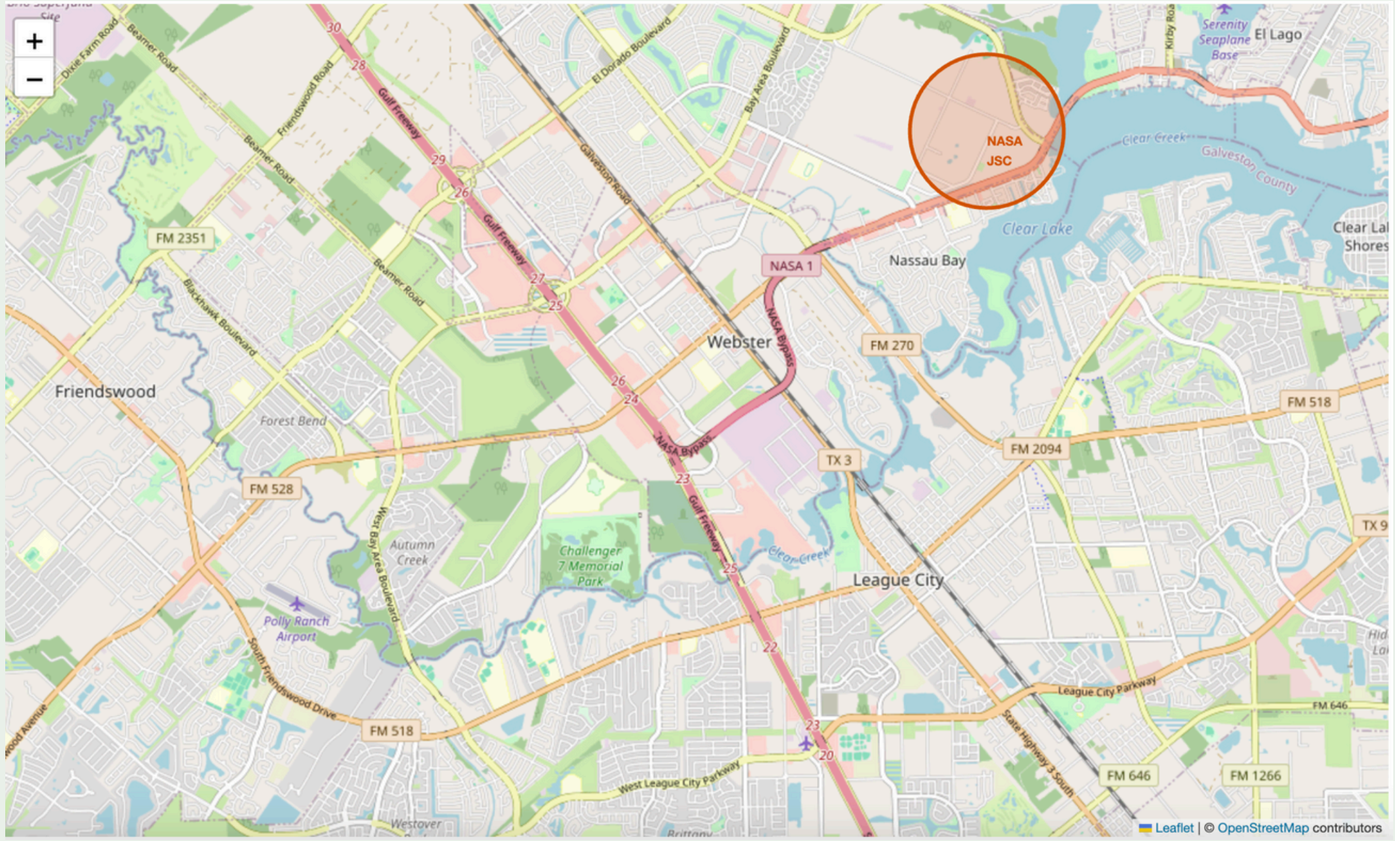
[22]:
```
va_data = data[['Launch Site', 'Lat', 'Long', 'class']]
launch_sites_df = va_data.groupby(['Launch Site'], as_index=False).first()
launch_sites_df = launch_sites_df[['Launch Site', 'Lat', 'Long']]
launch_sites_df
```

[22]:

	Launch Site	Lat	Long
0	CCAFS LC-40	28.562302	-80.577356
1	CCAFS SLC-40	28.563197	-80.576820
2	KSC LC-39A	28.573255	-80.646895
3	VAFB SLC-4E	34.632834	-120.610745

[24]:
```
# Start location is NASA Johnson Space Center
nasa_coordinate = [29.559684888503615, -95.0830971930759]
site_map = folium.Map(location=nasa_coordinate, zoom_start=10)
```

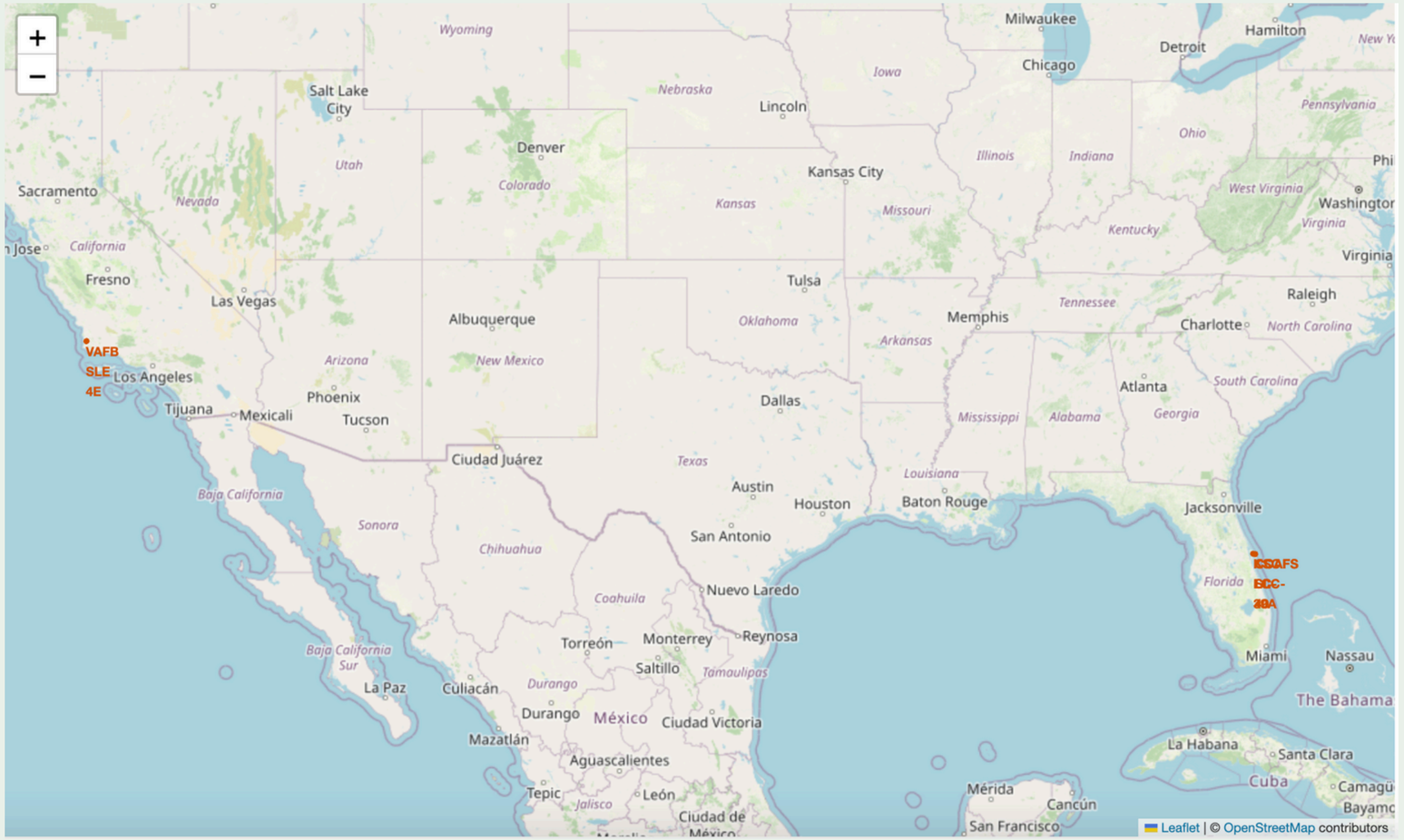
[32]:
```
# Create a blue circle at NASA Johnson Space Center's coordinate with a popup label showing its name
circle = folium.Circle(nasa_coordinate, radius=1000, color='#d35400', fill=True).add_child(folium.Popup('NASA Johnson Space Center'))
# Create a red circle at NASA Johnson Space Center's coordinate 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)
```


NASA
JSC
NASA 1
Webster
Nassau Bay
El Lago
Clear Lake
Clear Creek
Galveston County
Friendswood
Forest Bend
League City
Autumn Creek
Challenger 7 Memorial Park
Polly Ranch Airport
Clear Creek
FM 2351
FM 528
FM 518
FM 270
FM 2094
TX 3
FM 646
FM 1266
Gulf Freeway
NASA Bypass
Beamer Road
Friendswood Road
El Dorado Boulevard
Bay Area Boulevard
Galveston Road
Blackhawk Boulevard
West Bay Area Boulevard
South Friendswood Drive
West League City Parkway
League City Parkway
State Highway 3 South
Leaflet | © OpenStreetMap contributors

Interactive Visual Analysis

Task 1: Mark all launch sites on a map.

```
# 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. In addition, add Launch site name as a popup label
vafb_coordinates = [34.750144,-120.521294]
ksc_coordinates = [28.573255,-80.646895]
ccafs_coordinates = [28.562302,-80.577356]
ccafs_slc_coordinates = [28.563197,-80.576820]

vafb_circle = folium.Circle(vafb_coordinates, radius=1000, color='#d35400', fill=True).add_child(folium.Popup('Vandenberg Space Force Base')
vafb_marker = folium.map.Marker(vafb_coordinates, icon=DivIcon(icon_size=(20,20),icon_anchor=(0,0),html='<div style="font-size: 12; color:#d
site_map.add_child(vafb_circle)
site_map.add_child(vafb_marker)

ksc_circle = folium.Circle(ksc_coordinates, radius=1000, color='#d35400', fill=True).add_child(folium.Popup('Kennedy Space Center'))
ksc_marker = folium.map.Marker(ksc_coordinates, icon=DivIcon(icon_size=(20,20),icon_anchor=(0,0),html='<div style="font-size: 12; color:#d35
site_map.add_child(ksc_circle)
site_map.add_child(ksc_marker)

ccafs_circle = folium.Circle(ccafs_coordinates, radius=1000, color='#d35400', fill=True).add_child(folium.Popup('Cape Canaveral LC'))
ccafs_marker = folium.map.Marker(ccafs_coordinates, icon=DivIcon(icon_size=(20,20),icon_anchor=(0,0),html='<div style="font-size: 12; color:
site_map.add_child(ccafs_circle)
site_map.add_child(ccafs_marker)

ccafs_slc_circle = folium.Circle(ccafs_slc_coordinates, radius=1000, color='#d35400', fill=True).add_child(folium.Popup('Cape Canaveral SLC'
ccafs_slc_marker = folium.map.Marker(ccafs_slc_coordinates, icon=DivIcon(icon_size=(20,20),icon_anchor=(0,0),html='<div style="font-size: 12
site_map.add_child(ccafs_slc_circle)
site_map.add_child(ccafs_slc_marker)
```


VAFB
SLE
4E
CCAFS
SLC-
40
Leaflet | © OpenStreetMap contributors

Interactive Visual Analysis

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

```
va_data.tail(10)
```

	Launch Site	Lat	Long	class
46	KSC LC-39A	28.573255	-80.646895	1
47	KSC LC-39A	28.573255	-80.646895	1
48	KSC LC-39A	28.573255	-80.646895	1
49	CCAFS SLC-40	28.563197	-80.576820	1
50	CCAFS SLC-40	28.563197	-80.576820	1
51	CCAFS SLC-40	28.563197	-80.576820	0
52	CCAFS SLC-40	28.563197	-80.576820	0
53	CCAFS SLC-40	28.563197	-80.576820	0
54	CCAFS SLC-40	28.563197	-80.576820	1
55	CCAFS SLC-40	28.563197	-80.576820	0

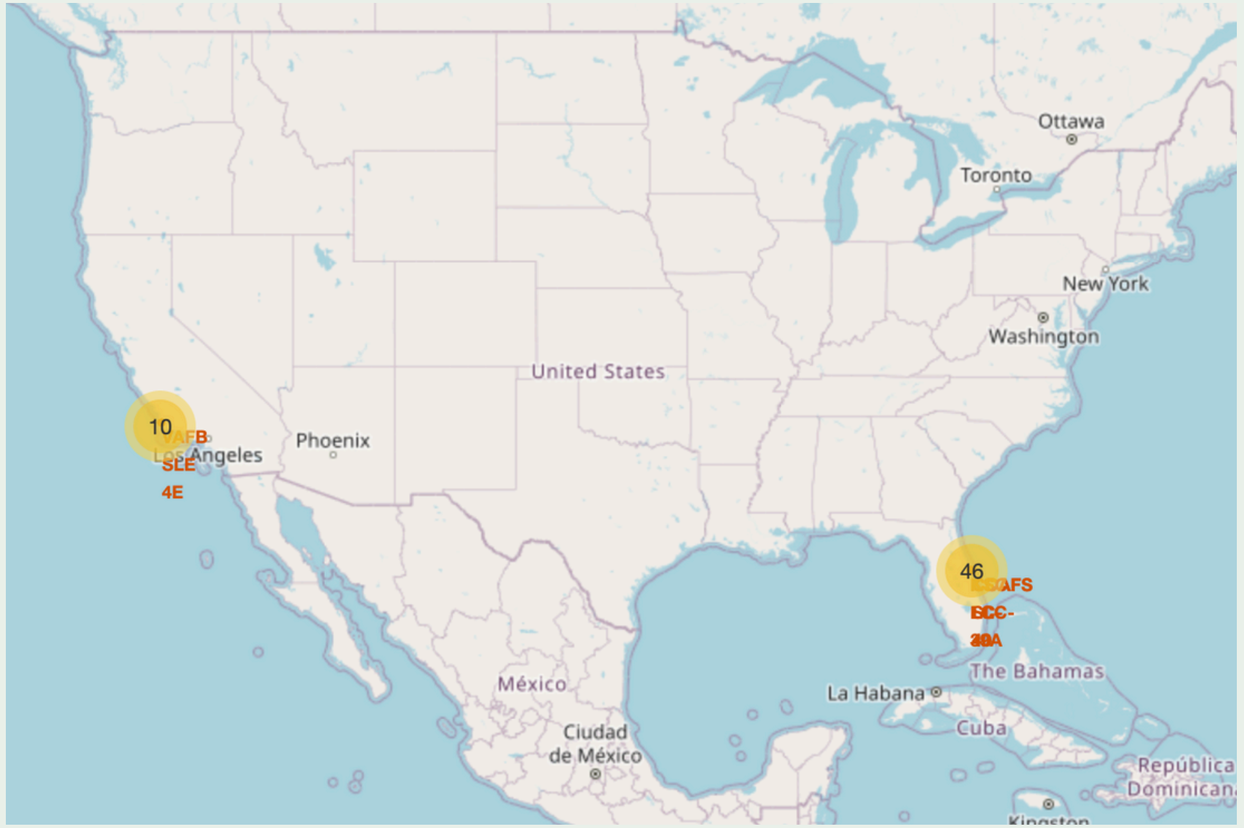
```
# To create Markers of the successful/failed launches of each site
# Apply a function to check the value of `class` column
# If class=1, marker_color value will be green
# If class=0, marker_color value will be red
marker_cluster = MarkerCluster()
def assign_marker_color(launch_outcome):
    if launch_outcome == 1:
        return 'green'
    else:
        return 'red'

va_data['marker_color'] = va_data['class'].apply(assign_marker_color)
va_data.tail(10)
```

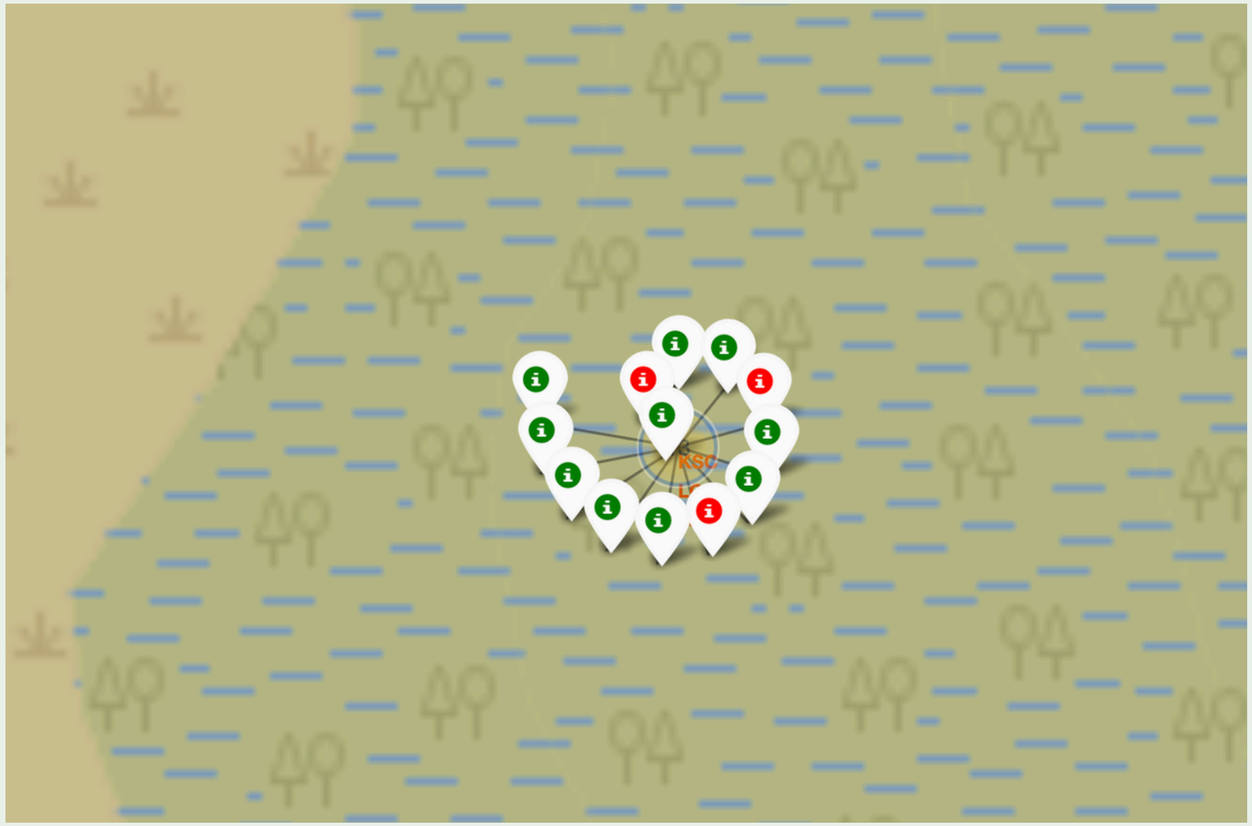
[18]:

	Launch Site	Lat	Long	class	marker_color
46	KSC LC-39A	28.573255	-80.646895	1	green
47	KSC LC-39A	28.573255	-80.646895	1	green
48	KSC LC-39A	28.573255	-80.646895	1	green
49	CCAFS SLC-40	28.563197	-80.576820	1	green
50	CCAFS SLC-40	28.563197	-80.576820	1	green
51	CCAFS SLC-40	28.563197	-80.576820	0	red
52	CCAFS SLC-40	28.563197	-80.576820	0	red
53	CCAFS SLC-40	28.563197	-80.576820	0	red
54	CCAFS SLC-40	28.563197	-80.576820	1	green
55	CCAFS SLC-40	28.563197	-80.576820	0	red

[22]:

```
# Add marker_cluster to current site_map
site_map.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,
for site_lat, site_long, marker_color in zip(va_data['Lat'], va_data['Long'], va_data['marker_color']):
    site_coordinate = [site_lat, site_long]
    marker = folium.map.Marker(
        site_coordinate,
        # Create an icon as a text label
        icon=folium.Icon(color='white',
                         icon_color=marker_color)
    )
    marker.add_to(marker_cluster)

site_map
```


Ottawa
Toronto
New York
Washington
United States
10
Phoenix
Los Angeles
SLC
4E
46
AFS
ECC-
39A
The Bahamas
La Habana
Cuba
México
Ciudad
de México
República
Dominican


KSC

Interactive Visual Analysis

TASK 3: Calculate the distances between a launch site to its proximities

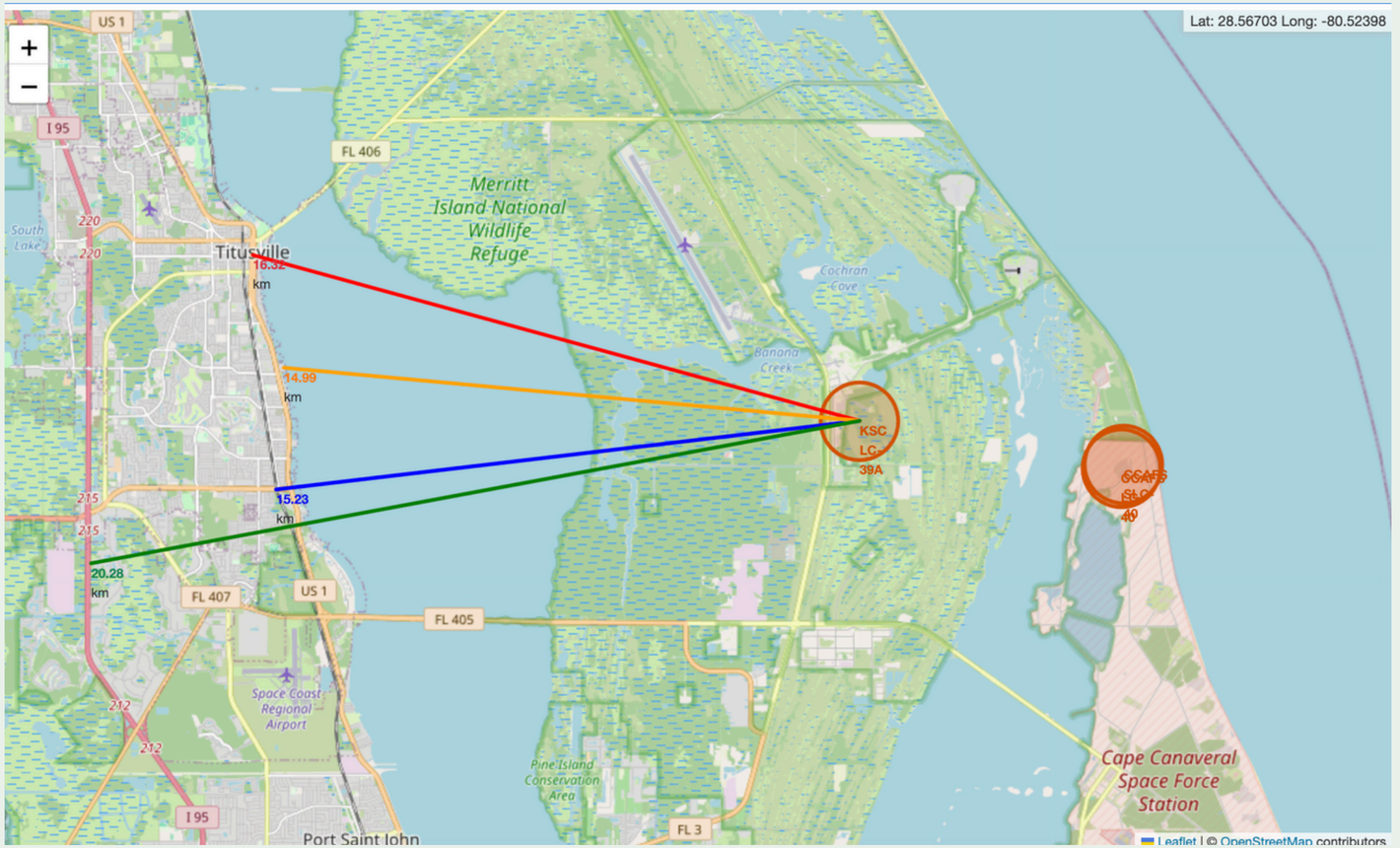
```
from math import sin, cos, sqrt, atan2, radians

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
```

```
railway_marker = [28.55752, -80.80155]
launch_coordinate = [28.57337, -80.64669]
distance_railway = calculate_distance(railway_marker[0], railway_marker[1], launch_coordinate[0], launch_coordinate[1])
distance_railway
```

```
15.230651245141603
```

```
# create and add a folium.Marker on your selected closest raiwaly point on the map
marker = folium.map.Marker(
        railway_marker,
        # Create an icon as a text label
        icon=DivIcon(
            icon_size=(400, 400),
            icon_anchor=(0, 0),
            html='<div style="font-size:400; color:#0c10f2;"><b>%s</b></div>' % str(round(distance_railway, 2))+' km',
            )
    )
marker.add_to(site_map)

site_map
```

```
# Create a marker with distance to a closest city, railway, highway, etc.
# Draw a line between the marker to the launch site
city    = [28.61200, -80.80788]
coastline = [28.5858, -80.79952]
highway   = [28.5402, -80.85079]

city_distance = calculate_distance(city[0], city[1], launch_coordinate[0], launch_coordinate[1])
coastline_distance = calculate_distance(coastline[0], coastline[1], launch_coordinate[0], launch_coordinate[1])
highway_distance = calculate_distance(highway[0], highway[1], launch_coordinate[0], launch_coordinate[1])

colors = ['red','orange','green']
html_colors = ['#dc3545','#fd7e14','#198754']

for coordinate ,distance, color, html_color in zip([city, coastline, highway], [city_distance, coastline_distance, highway_distance], colors
    marker = folium.map.Marker(
            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:'+html_color+';"><b>%s</b></div>' % str(round(distance, 2)) + 'km',
                )
            )
    marker.add_to(site_map)
    folium.PolyLine([coordinate, launch_coordinate], color=color).add_to(site_map)
site_map
```


Lat: 28.56703 Long: -80.52398
US 1
I 95
FL 406
Merritt Island National Wildlife Refuge
South Lake
220
220
Titusville
16.32 km
14.99 km
15.23 km
20.28 km
Cochran Cove
Banana Creek
KSC LC 39A
SSAFS SLC 40
215
215
FL 407
US 1
FL 405
Space Coast Regional Airport
212
212
Pine Island Conservation Area
Cape Canaveral Space Force Station
I 95
Port Saint John
FL 3
Leaflet | © OpenStreetMap contributors

Predictive Analysis

Now, we can develop a machine learning model to predict whether the first stage of the Falcon 9 will land.

For this, we will develop different models for classification to determine which model is the most accurate predictor.

The classification models we will test are:

1. Logistic regression
2. Suport vector machine
3. Decision tree
4. k-Nearest neighbour

Predictive Analysis

1. Logistic Regression Model

```
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, param_grid=parameters,scoring='accuracy', cv=10)
logreg_cv.fit(X_train, Y_train)
logreg_cv.best_params_
```

```
{'C': 0.01, 'penalty': 'l2', 'solver': 'lbfgs'}
```

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

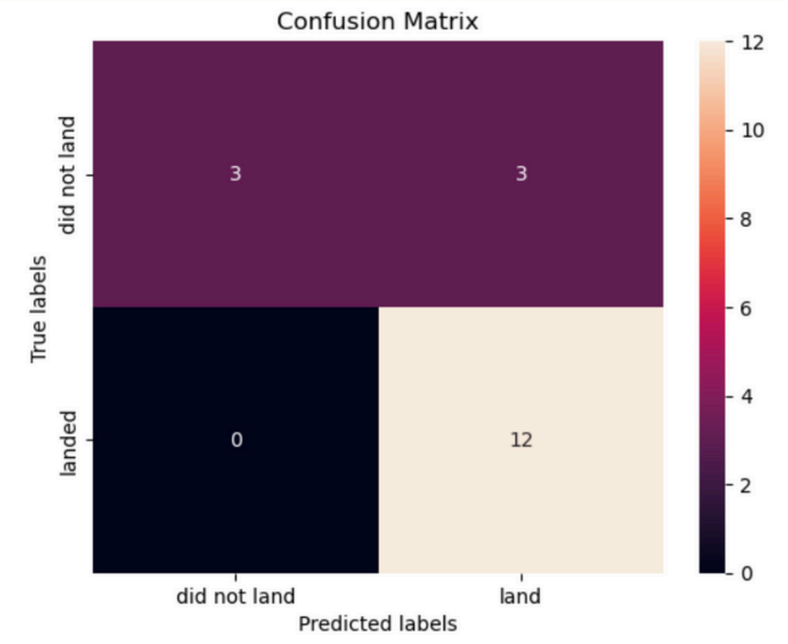
```
tuned hpyerparameters :(best parameters)  {'C': 0.01, 'penalty': 'l2', 'solver': 'lbfgs'}
accuracy : 0.8464285714285713
```

Task 5: Calculate the accuracy on the test data using the method score.

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

```
0.8333333333333334
```

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



Predictive Analysis

2. Support Vector Machine

```
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, param_grid=parameters,scoring='accuracy', cv=10)
svm_cv.fit(X_train, Y_train)
svm_cv.best_params_
```

```
{'C': 1.0, 'gamma': 0.03162277660168379, 'kernel': 'sigmoid'}
```

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

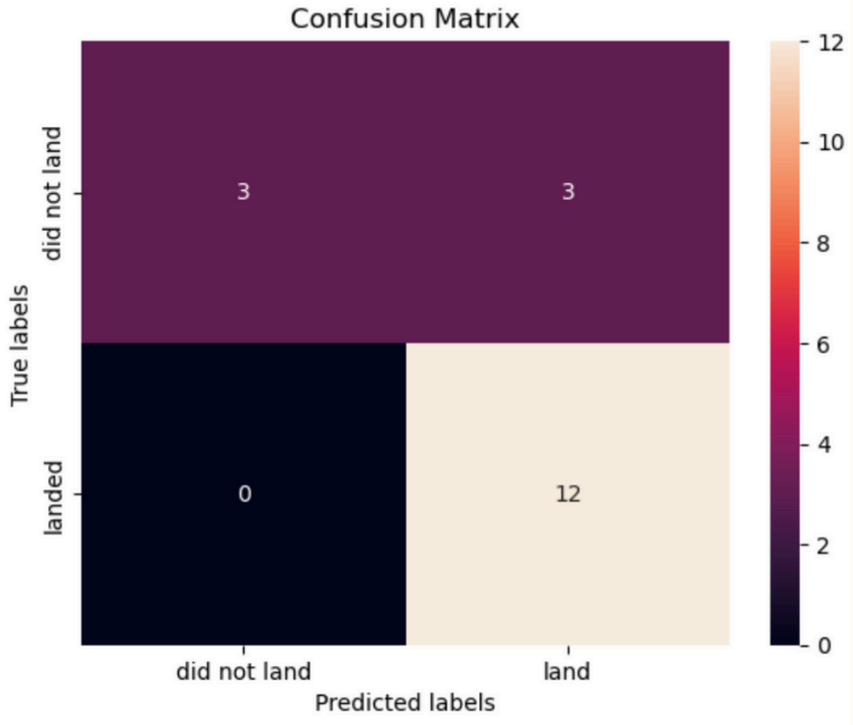
```
tuned hpyerparameters :(best parameters)  {'C': 1.0, 'gamma': 0.03162277660168379, 'kernel': 'sigmoid'}
accuracy : 0.8482142857142856
```

Task 7: Calculate the accuracy on the test data using the method score.

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

```
0.8333333333333334
```

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



Predictive Analysis

3. Decision Tree Classifier

```
parameters = {'criterion': ['gini', 'entropy'],
     'splitter': ['best', 'random'],
     'max_depth': [3, 5, 7, 10],
     'max_features': ['auto', 'sqrt'],
     'min_samples_leaf': [1, 2, 4],
     'min_samples_split': [2, 5, 10]}

tree = DecisionTreeClassifier()
```

```
tree_cv = GridSearchCV(tree, param_grid=parameters,scoring='accuracy', cv=10)
tree_cv = grid_search_tree.fit(X_train, Y_train)
```

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

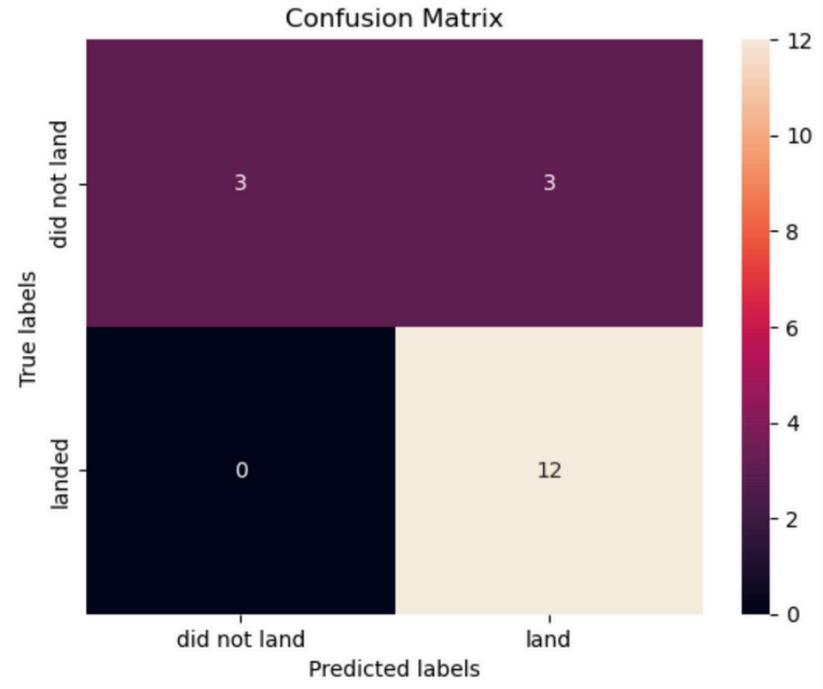
```
tuned hpyerparameters :(best parameters)  {'criterion': 'gini', 'max_depth': 2, 'max_features': 'sqrt', 'min_samples_leaf': 1, 'min_samples
_split': 10, 'splitter': 'best'}
accuracy : 0.875
```

Task 9: Calculate the accuracy of tree_cv on the test data using the method score.

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

```
0.8333333333333334
```

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



Predictive Analysis

4. k-nearest neighbour

```
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, param_grid=parameters,scoring='accuracy', cv=10)
knn_cv.fit(X_train, Y_train)
knn_cv.best_params_
```

```
{'algorithm': 'auto', 'n_neighbors': 10, 'p': 1}
```

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

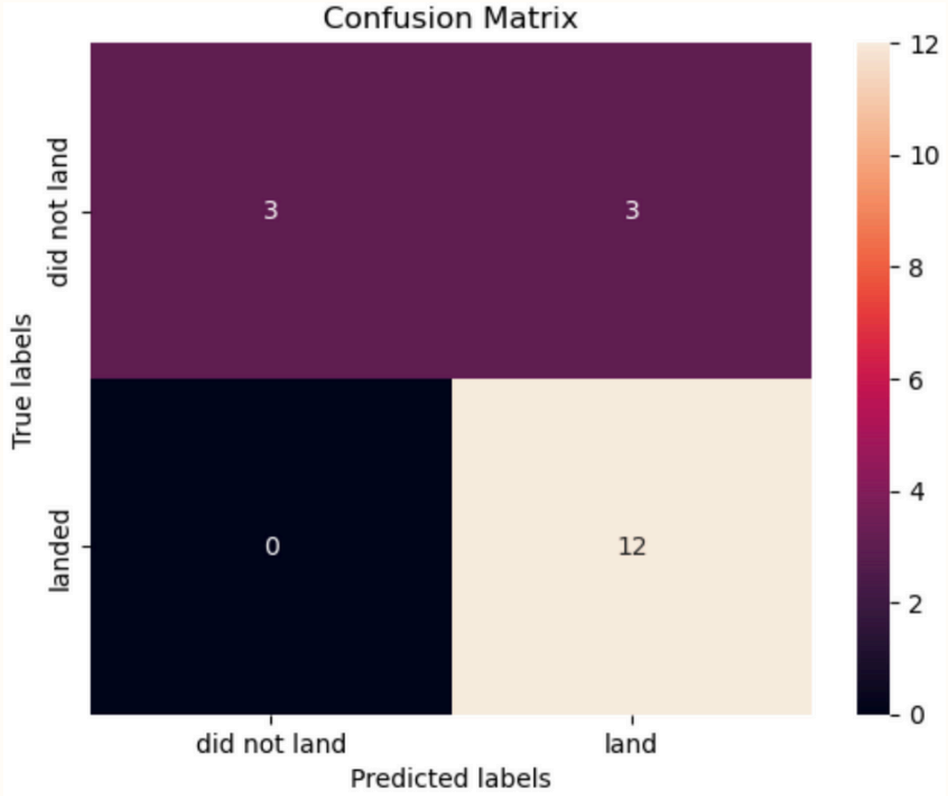
```
tuned hpyerparameters :(best parameters)  {'algorithm': 'auto', 'n_neighbors': 10, 'p': 1}
accuracy : 0.8482142857142858
```

Task 11: Calculate the accuracy of knn_cv on the test data using the method score.

```
knn_cv.score(X_test, Y_test)
```

```
0.8333333333333334
```

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



Predictive Analysis

Attribute	Logistic Regression	Support Vector Mechanism	Decision Tree Classifier	k-nearest neighbour
Accuracy on trained data	0.846	0.848	0.875	0.848
Accuracy on test data	0.833	0.833	0.833	0.833

```
predictor_models = {
              'LogisticRegression':logreg_cv.best_score_,
              'SupportVectorMachine': svm_cv.best_score_,
              'DecisionTree':tree_cv.best_score_,
              'KNeighbors':knn_cv.best_score_,
}

bestmodel = max(predictor_models, key=predictor_models.get)
print('The best model for prediction is', bestmodel,'with a score of', predictor_models[bestmodel],'.')
```

```
The best model for prediction is DecisionTree with a score of 0.875 .
```

We can conclude that the Decision Tree Classification model is the best predictive model with an accuracy of 87.5%.

Conclusion

KSC LC-39A launch site records the highest number of successful launches.

Launch Success Rates Have Consistently Improved Since 2013

Missions targeting orbits ES-L1, GEO, HEO, SSO, and VLEO had the highest rates of success.

Decision tree classifier was the best model to predict the launch outcomes.

Thank you