IBM Data Science

You are currently enrolled in **10 of the 10 courses** in this Professional Certificate.

- Data Science Methodology
- Data Analysis with Python
- Machine Learning with Python
- Python for Data Science, AI & Development
- Data Visualization with Python
- Tools for Data Science
- Python Project for Data Science
- Applied Data Science Capstone
- What is Data Science?
- Databases and SQL for Data Science with Python

Applied Data Science Capstone Report

Wilson Ashimwe

06 August 2021

OUTLINE

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

EXECUTIVE SUMMARY

This report discusses summarizes the results of project aiming to provide students who doing the IBM Data Science Professional Certificate skills and knowledge to take on a real-world data science project. This project exploited data on space lockets. Falcon 9 is advertised by SpaceX a lower cost (62 million dollars) compared to other providers because SpaceX can reuse the first stage resulting in a lower cost. Therefore, determining if the first stage will land, allows to determine the cost of a launch. This information can be used if an alternate company wants to bid against SpaceX for a rocket launch. In this project, I determined the price of each launch gathered information about Space X and created dashboards to visualize the data. In addition, I determined if SpaceX would reuse the first stage. Various machine learning algorithms were used to train a machine learning model and use public information to predict if SpaceX will reuse the first stage.

INTRODUCTION

- The final course of the Data Science Professional Certificate consists of a capstone project where in all the skills and relevant knowledge that one has gathered from these 9 intense courses has to be applied on a final capstone project.
- During the capstone project, different data analysis, data visualization and machine learning packages were used to train a machine learning model and use public information to predict if SpaceX will reuse the first stage. These include Pandas, Matplotlib, Seaborn, Folium, Plotly, and Sklearn packages. This report discusses summarizes the results of the project.

INTRODUCTION 2

- The aim of the project was to predict if the Falcon 9 first stage will land successfully. Falcon 9 is advertised by SpaceX a lower cost (62 million dollars) compared to other providers because SpaceX can reuse the first stage resulting in a lower cost. Therefore, determining if the first stage will land, allows to determine the cost of a launch. This information can be used if an alternate company wants to bid against SpaceX for a rocket launch.
- In this project, I determined the price of each launch gathered information about Space X and created dashboards to visualize the data. In addition, I determined if SpaceX would reuse the first stage. Various machine learning algorithms were used to train a machine learning model and use public information to predict if SpaceX will reuse the first stage.

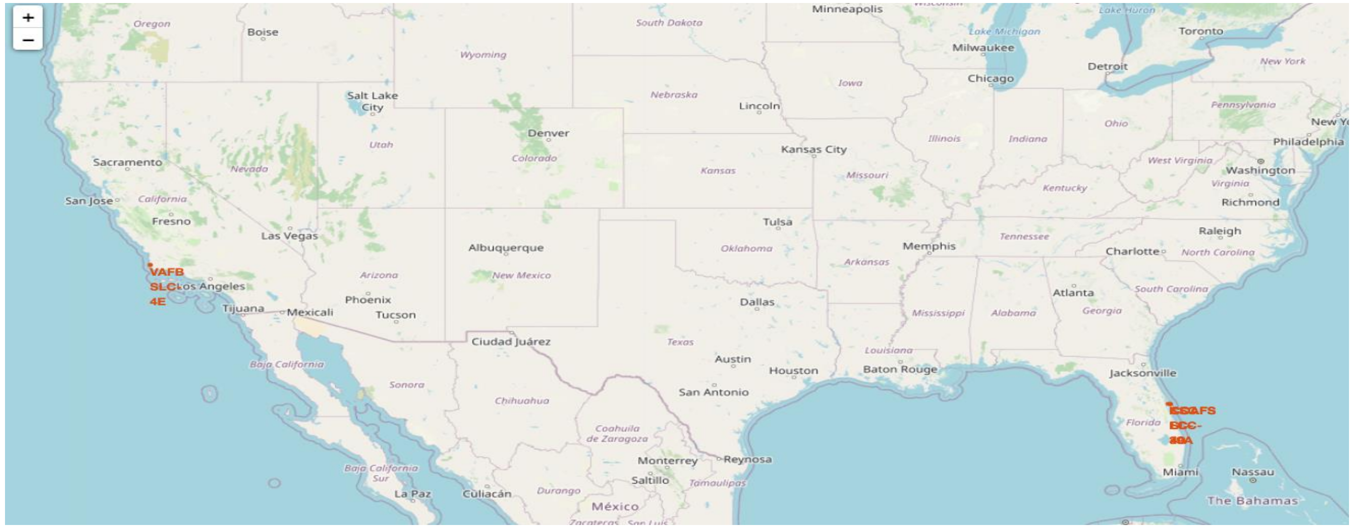
METHODOLOGY : Data Wrangling

- An in-depth research of the dataset has been done and a thorough analysis of the various features and methods have been investigated to ensure the maximum accuracy of the model as possible.
- Data wrangling was performed to get a dataset that could be used for further analysis. In the data set, there are several different cases where the booster did not land successfully. Those outcomes were converted into training Labels with 1 meaning the booster successfully landed and meaning it was unsuccessful.
- See the Class column below

	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins	Reused	Legs	LandingPad	Block	ReusedCount	Serial	Longitude	Latitude	Class
0	1	2010-06-04	Falcon 9	6104.959412	LEO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B0003	-80.577366	28.561857	0
1	2	2012-05-22	Falcon 9	525.000000	LEO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B0005	-80.577366	28.561857	0

METHODOLOGY: interactive visual analytics

- An exploratory analysis was done using SQL and then interactive visual analytics were performed using packages including Folium and Plotly.
- Folium makes it easy to visualize geolocation data .
- Please see below a map with marked launch sites



METHODOLOGY: predictive analysis

- Various machine learning algorithms were used to train a machine learning model and use public information to predict if SpaceX will reuse the first stage.
- The figure below shows on of the machine learning model

Create a logistic regression object using then create a GridSearchCV object `logreg_cv` with cv = 10. Fit the object to find the best parameters from the dictionary `parameters`.

```
In [29]: parameters ={'C':[0.01,0.1,1],
                      'penalty':['l2'],
                      'solver':['lbfgs']}
```

```
In [30]: parameters ={"C":[0.01,0.1,1],'penalty':['l2'], 'solver':['lbfgs']}# l1 lasso l2 ridge
         lr=LogisticRegression()
         logreg_cv = GridSearchCV(lr, param_grid=parameters, cv=10)
         logreg_cv.fit(X_train, Y_train)

Out[30]: 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\_`.

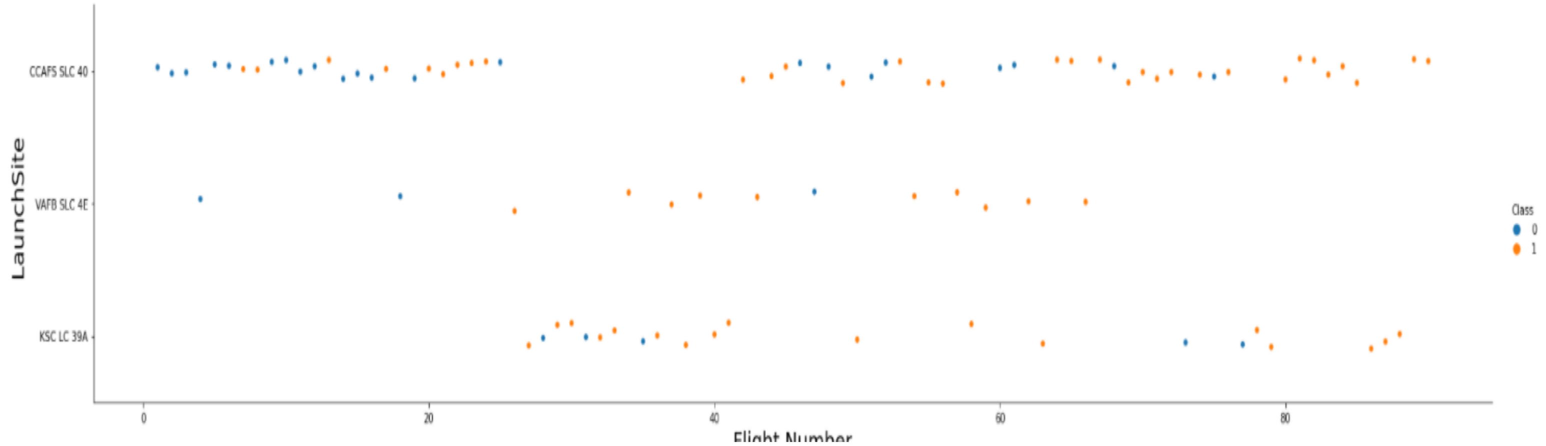
```
In [31]: print("tuned hpyerparameters :(best parameters) ",logreg_cv.best_params_)
         print("accuracy :",logreg_cv.best_score_)

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

RESULT: EDA with visualization results: Task 1

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



RESULT: EDA with visualization results: Task 2

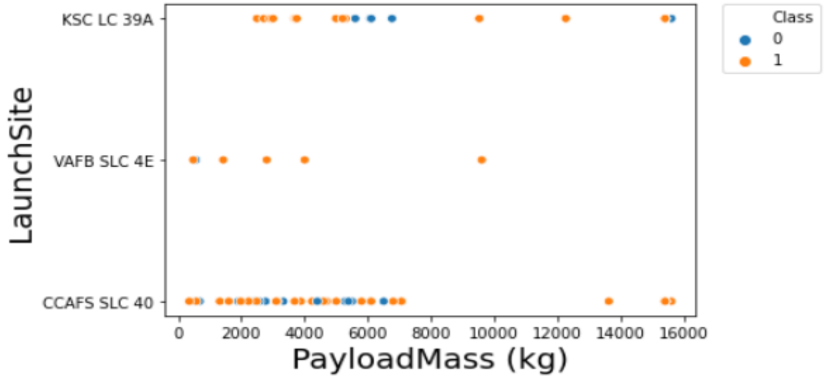
TASK 2: Visualize the relationship between Payload and Launch Site

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

In [5]:
```
df.columns
```

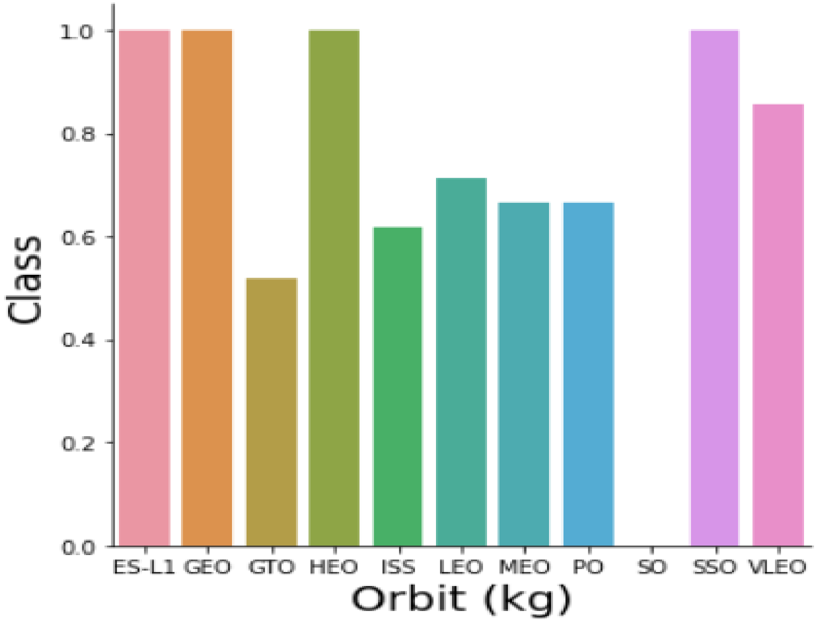
Out[5]:
```
Index(['FlightNumber', 'Date', 'BoosterVersion', 'PayloadMass', 'Orbit',
       'LaunchSite', 'Outcome', 'Flights', 'GridFins', 'Reused', 'Legs',
       'LandingPad', 'Block', 'ReusedCount', 'Serial', 'Longitude', 'Latitude',
       'Class'],
      dtype='object')
```

[10]:
```
# 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 valu
e
sns.scatterplot(y="LaunchSite", x="PayloadMass", hue="Class", data=df)
plt.xlabel("PayloadMass (kg)",fontsize=20)
plt.ylabel("LaunchSite",fontsize=20)
plt.legend(bbox_to_anchor=(1.05, 1), borderaxespad=0)
plt.show()
```



RESULT: EDA with visualization results: Task 3

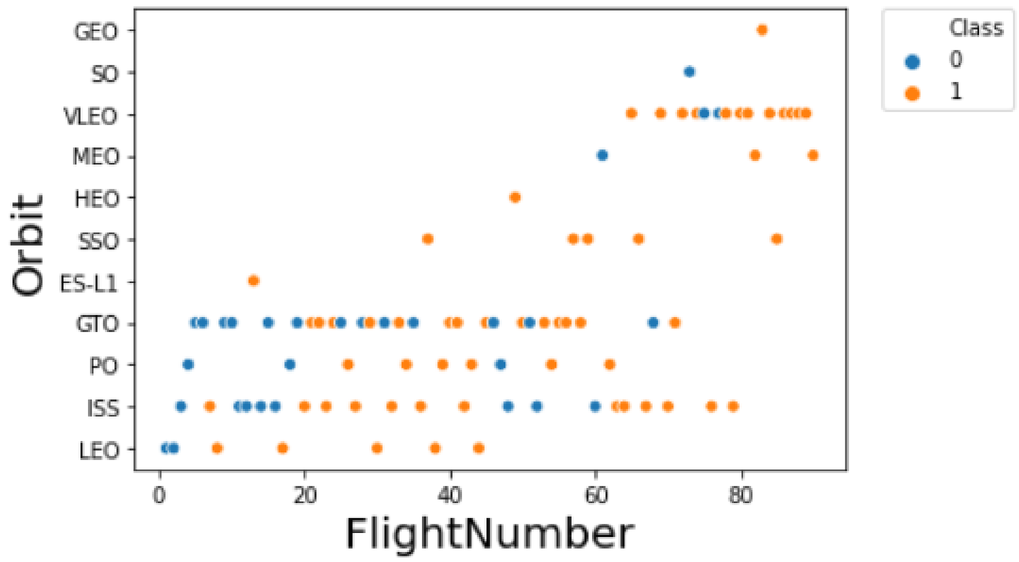
```
[17]: sns.catplot(y="Class", x="Orbit", kind="bar", data=newdf)
      plt.xlabel("Orbit (kg)",fontsize=20)
      plt.ylabel("Class",fontsize=20)
      plt.show()
```



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

RESULT: EDA with visualization results: Task 4

```
# 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.scatterplot(y="Orbit", x="FlightNumber", hue="Class", data=df)
plt.xlabel("FlightNumber",fontsize=20)
plt.ylabel("Orbit",fontsize=20)
plt.legend(bbox_to_anchor=(1.05, 1), borderaxespad=0)
plt.show()
```

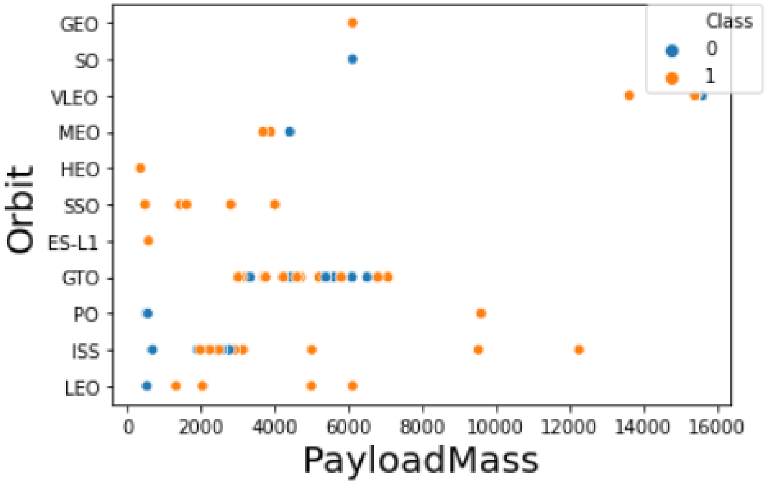


RESULT: EDA with visualization results: Task 5

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

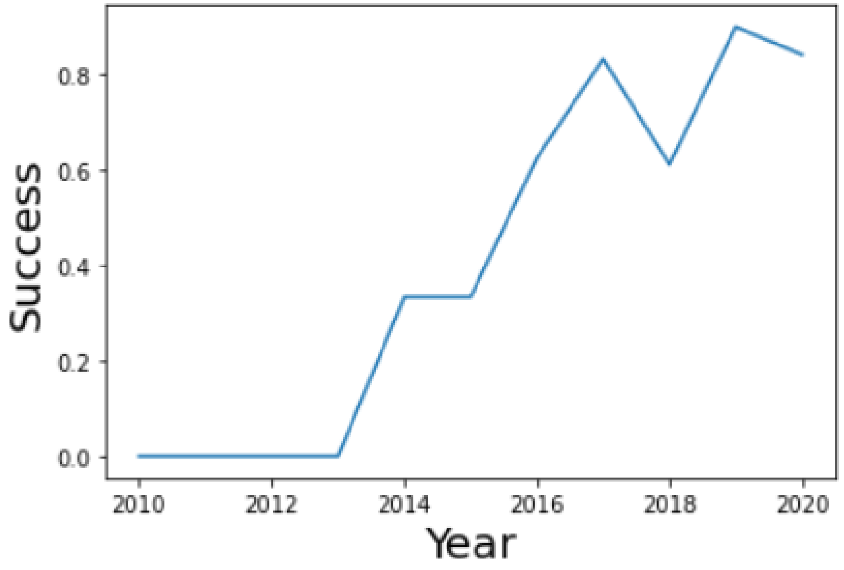
```
]: # Plot a scatter point chart with x axis to be Payload and y axis to be the Orbit, and hue to be the class value
sns.scatterplot(y="Orbit", x="PayloadMass", hue="Class", data=df)
plt.xlabel("PayloadMass",fontsize=20)
plt.ylabel("Orbit",fontsize=20)
plt.legend(bbox_to_anchor=(1.05, 1), borderaxespad=0)
plt.show()
```



You should observe that Heavy payloads have a negative influence on GTO orbits and positive on GTO and Polar LEO (ISS) orbits.

RESULT: EDA with visualization results: Task 6

```
]: # Plot a line chart with x axis to be the extracted year and y axis to be the success rate
sns.lineplot(y="Success", x="Year", data=dfnew1)
plt.xlabel("Year",fontsize=20)
plt.ylabel("Success",fontsize=20)
plt.show()
```



RESULT: EDA with visualization results: Task 7 and 8

TASK 7: Create dummy variables to categorical columns

Use the function `get_dummies` and `features` dataframe to apply OneHotEncoder to the column `Orbits`, `LaunchSite`, `LandingPad`, and `Serial`. Assign the value to the variable `features_one_hot`, display the results using the method head. Your result dataframe must include all features including the encoded ones.

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

```
features_one_hot.head()
```

TASK 8: Cast all numeric columns to `float64`

Now that our `features_one_hot` dataframe only contains numbers cast the entire dataframe to variable type `float64`

```
# HINT: use astype function
features_one_hot=features_one_hot.astype('float64')
```

We can now export it to a **CSV** for the next section,but to make the answers consistent, in the next lab we will provide data in a pre-selected date range.

```
features_one_hot.to_csv('dataset_part\_3.csv', index=False)
```

RESULT: EDA with SQL results slides: Task 1 and 2

Task 1

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

```
2]: %sql SELECT DISTINCT LAUNCH_SITE FROM SPACEDATA;
```

 * ibm_db_sa://mqy07600:***@0c77d6f2-5da9-48a9-81f8-86b520
Done.

2]:

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

Task 2

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

```
%sql SELECT * FROM SPACEDATA WHERE LAUNCH_SITE LIKE 'CCA%' LIMIT 5;
```

 * ibm_db_sa://mqy07600:***@0c77d6f2-5da9-48a9-81f8-86b520b87518.bs2io90l08kqb1od8lcg.databases.appdomain.cloud:31198/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

RESULT: EDA with SQL results slides: Task 1 and 2

Task 1

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

```
2]: %sql SELECT DISTINCT LAUNCH_SITE FROM SPACEDATA;
```

```
 * ibm_db_sa://mqy07600:***@0c77d6f2-5da9-48a9-81f8-86b520
Done.
```

2]:

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

Task 2

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

```
%sql SELECT * FROM SPACEDATA WHERE LAUNCH_SITE LIKE 'CCA%' LIMIT 5;
```

```
 * ibm_db_sa://mqy07600:***@0c77d6f2-5da9-48a9-81f8-86b520b87518.bs2io90l08kqb1od8lcg.databases.appdomain.cloud:31198/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

RESULT: EDA with SQL results slides: Task 3 and 4

Task 3

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

```
%sql SELECT  SUM(PAYLOAD_MASS__KG_) AS TOTAL_MASS FROM SPACEDATA WHERE CUSTOMER = 'NASA (CRS)';
```

```
 * ibm_db_sa://mqy07600:***@0c77d6f2-5da9-48a9-81f8-86b520b87518.bs2io90l08kqb1od8lcg.databases.appdomain.cloud:31198/bludb
Done.
```

total_mass
45596

Task 4

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

```
%sql SELECT  AVG(PAYLOAD_MASS__KG_) AS TOTAL_MASS FROM SPACEDATA WHERE BOOSTER_VERSION LIKE 'F9 v1.0%';
```

```
 * ibm_db_sa://mqy07600:***@0c77d6f2-5da9-48a9-81f8-86b520b87518.bs2io90l08kqb1od8lcg.databases.appdomain.cloud:31198/bludb
Done.
```

total_mass
340

RESULT: EDA with SQL results slides: Task 3 and 4

Task 3

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

```
%sql SELECT  SUM(PAYLOAD_MASS__KG_) AS TOTAL_MASS FROM SPACEDATA WHERE CUSTOMER = 'NASA (CRS)';
```

```
 * ibm_db_sa://mqy07600:***@0c77d6f2-5da9-48a9-81f8-86b520b87518.bs2io90l08kqb1od8lcg.databases.appdomain.cloud:31198/bludb
Done.
```

total_mass
45596

Task 4

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

```
%sql SELECT  AVG(PAYLOAD_MASS__KG_) AS TOTAL_MASS FROM SPACEDATA WHERE BOOSTER_VERSION LIKE 'F9 v1.0%';
```

```
 * ibm_db_sa://mqy07600:***@0c77d6f2-5da9-48a9-81f8-86b520b87518.bs2io90l08kqb1od8lcg.databases.appdomain.cloud:31198/bludb
Done.
```

total_mass
340

RESULT: EDA with SQL results slides: Task 6

Task 6

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

```
]: %sql SELECT DISTINCT BOOSTER_VERSION FROM SPACEDATA  WHERE MISSION_OUTCOME LIKE '%Success%' AND PAYLOAD_MASS__KG
OAD_MASS__KG_ <6000  ;
```

```
 * ibm_db_sa://mqy07600:***@0c77d6f2-5da9-48a9-81f8-86b520b87518.bs2io90l08kqb1od8lcg.databases.appdomain.cloud:
Done.
```

]:

booster_version
F9 B4 B1040.2
F9 B4 B1040.1
F9 B4 B1043.1
F9 B5 B1046.2
F9 B5 B1047.2
F9 B5 B1048.3
F9 B5 B1051.2
F9 B5 B1058.2
F9 B5B1054
F9 B5B1060.1
F9 B5B1062.1

F9 FT B1021.2
F9 FT B1031.2
F9 FT B1032.2
F9 FT B1020
F9 FT B1022
F9 FT B1026
F9 FT B1030
F9 FT B1032.1
F9 v1.1
F9 v1.1 B1011
F9 v1.1 B1014
F9 v1.1 B1016

RESULT: EDA with SQL results slides: Task 7 and 8

Task 7

List the total number of successful and failure mission outcomes

```
%sql SELECT MISSION_OUTCOME, COUNT(*) AS TOTALN FROM SPACEDATA GROUP BY MISSION_OUTCOME;
```

```
 * ibm_db_sa://mqy07600:***@0c77d6f2-5da9-48a9-81f8-86b520b87518.bs2io90l08kqb1od8lcg.databases.appdomain.cloud:31198/bludb
Done.
```

mission_outcome	totaln
Failure (in flight)	1
Success	99
Success (payload status unclear)	1

Task 8

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

```
%sql SELECT BOOSTER_VERSION, (SELECT SUM(PAYLOAD_MASS__KG_) FROM SPACEDATA GROUP BY BOOSTER_VERSION DESC) AS TOTALMASS FROM SPAC
EDATA LIMIT 5;
```

```
 * ibm_db_sa://mqy07600:***@0c77d6f2-5da9-48a9-81f8-86b520b87518.bs2io90l08kqb1od8lcg.databases.appdomain.cloud:31198/bludb
(ibm_db_dbi.ProgrammingError) ibm_db_dbi::ProgrammingError: Exception('SQLNumResultCols failed: [IBM][CLI Driver][DB2/LINUXX866
4] SQL0104N  An unexpected token "DESC" was found following "P BY BOOSTER_VERSION".  Expected tokens may include:  "<grouping_c
ol_exp_list>".  SQLSTATE=42601 SQLCODE=-104')
[SQL: SELECT BOOSTER_VERSION, (SELECT SUM(PAYLOAD_MASS__KG_) FROM SPACEDATA GROUP BY BOOSTER_VERSION DESC) AS TOTALMASS FROM SP
ACEDATA LIMIT 5;]
(Background on this error at: http://sqlalche.me/e/f405)
```

Task 9

RESULT: EDA with SQL results slides: Task 8 and 9

Task 8

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

]:
```
%sql SELECT BOOSTER_VERSION, (SELECT SUM(PAYLOAD_MASS__KG_) FROM SPACEDATA GROUP BY BOOSTER_VERSION DESC) AS TOTALMASS FROM SPAC
EDATA LIMIT 5;
```

```
 * ibm_db_sa://mqy07600:***@0c77d6f2-5da9-48a9-81f8-86b520b87518.bs2io90l08kqb1od8lcg.databases.appdomain.cloud:31198/bludb
(ibm_db_dbi.ProgrammingError) ibm_db_dbi::ProgrammingError: Exception('SQLNumResultCols failed: [IBM][CLI Driver][DB2/LINUXX866
4] SQL0104N  An unexpected token "DESC" was found following "P BY BOOSTER_VERSION".  Expected tokens may include:  "<grouping_c
ol_exp_list>".  SQLSTATE=42601 SQLCODE=-104')
[SQL: SELECT BOOSTER_VERSION, (SELECT SUM(PAYLOAD_MASS__KG_) FROM SPACEDATA GROUP BY BOOSTER_VERSION DESC) AS TOTALMASS FROM SP
ACEDATA LIMIT 5;]
(Background on this error at: http://sqlalche.me/e/f405)
```

Task 9

List the records which will display the month names, failure landing_outcomes in drone ship ,booster versions, launch_site for the months in year 2015

]:
```
%sql SELECT MONTHNAME(DATE) AS MONTHNAME, LANDING__OUTCOME, BOOSTER_VERSION, LAUNCH_SITE FROM SPACEDATA WHERE YEAR(DATE)=2015 AN
D LANDING__OUTCOME NOT LIKE '%Success%';
```

```
 * ibm_db_sa://mqy07600:***@0c77d6f2-5da9-48a9-81f8-86b520b87518.bs2io90l08kqb1od8lcg.databases.appdomain.cloud:31198/bludb
Done.
```

]:

monthname	landing__outcome	booster_version	launch_site
January	Failure (drone ship)	F9 v1.1 B1012	CCAFS LC-40
February	Controlled (ocean)	F9 v1.1 B1013	CCAFS LC-40
March	No attempt	F9 v1.1 B1014	CCAFS LC-40
April	Failure (drone ship)	F9 v1.1 B1015	CCAFS LC-40
April	No attempt	F9 v1.1 B1016	CCAFS LC-40
June	Precluded (drone ship)	F9 v1.1 B1018	CCAFS LC-40

RESULT: EDA with SQL results slides: Task 10

Task 10

Rank the count of successful landing_outcomes between the date 2010-06-04 and 2017-03-20 in descending order.

[11]:
```
%sql SELECT LANDING__OUTCOME, COUNT(*) AS COUNT FROM SPACEDATA WHERE LANDING__OUTCOME LIKE '%Success%' AND DATE BETWEEN '20
-04'  AND '2017-03-20' GROUP BY LANDING__OUTCOME ORDER BY COUNT DESC;
```

```
 * ibm_db_sa://mqy07600:***@0c77d6f2-5da9-48a9-81f8-86b520b87518.bs2io90l08kqb1od8lcg.databases.appdomain.cloud:31198/bludb
Done.
```

[11]:

landing__outcome	COUNT
Success (drone ship)	5
Success (ground pad)	3

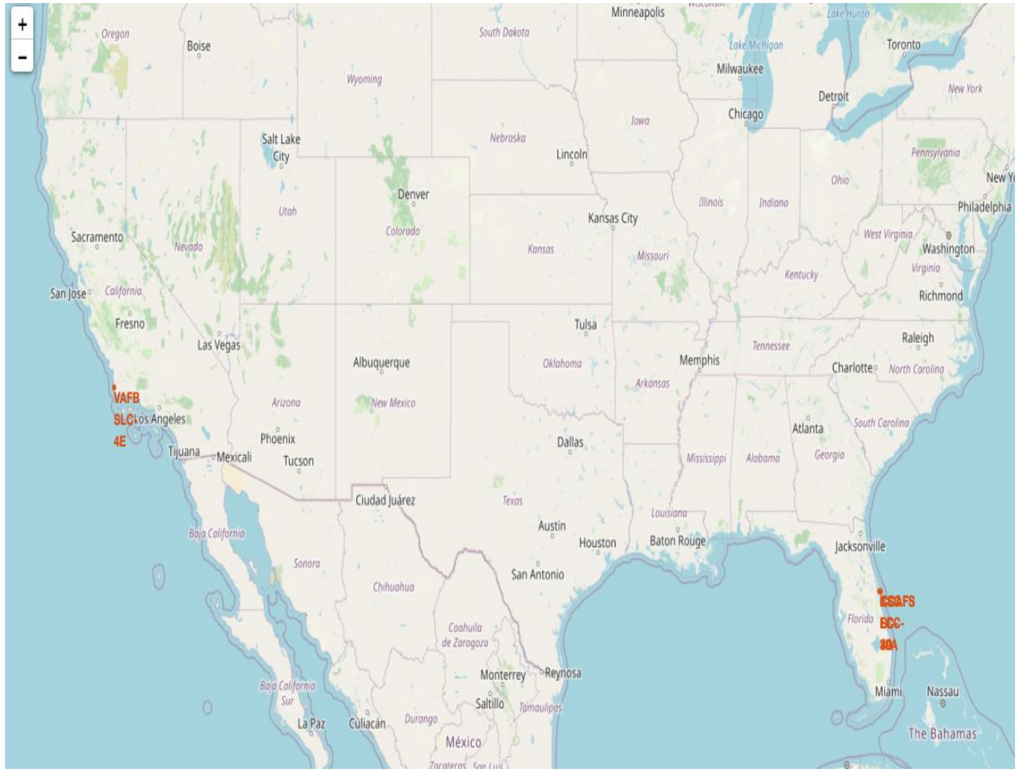
[14]:
```
%sql SELECT PAYLOAD_MASS__KG_ FROM SPACEDATA ORDER BY PAYLOAD_MASS__KG_ LIMIT ;
```

```
 * ibm_db_sa://mqy07600:***@0c77d6f2-5da9-48a9-81f8-86b520b87518.bs2io90l08kqb1od8lcg.databases.appdomain.cloud:31198/bludb
Done.
```

[14]:

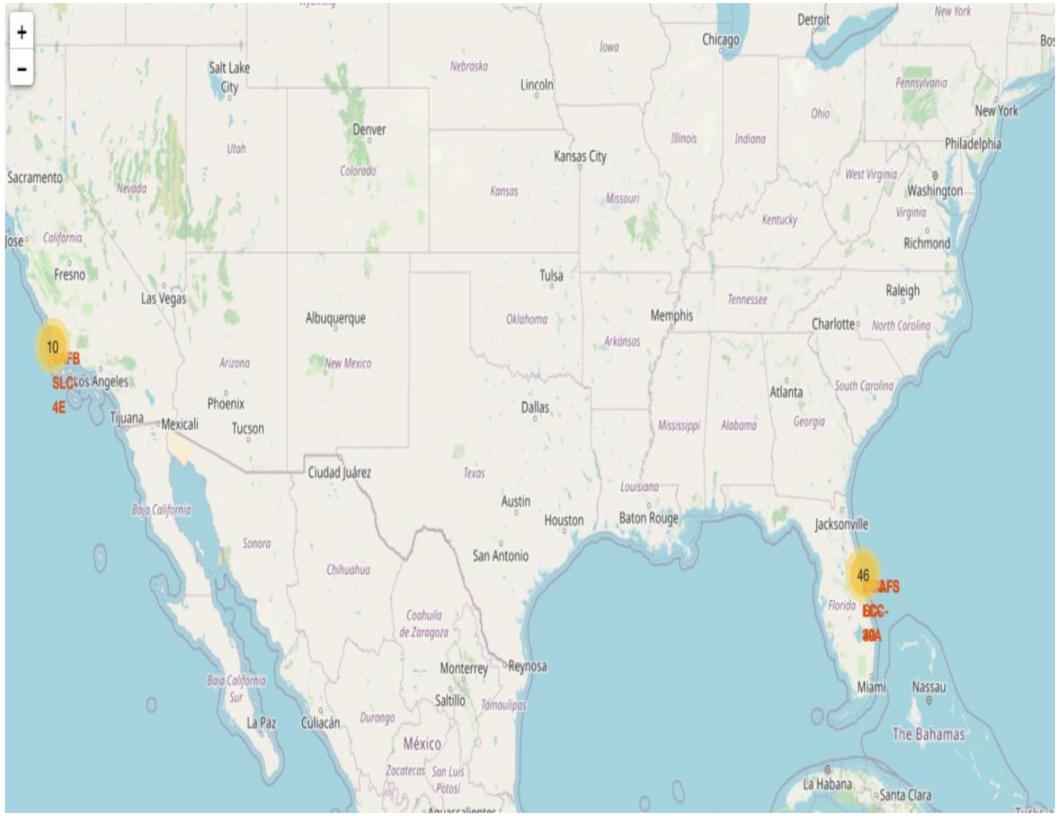
payload_mass__kg_
0
0
362
475

RESULT: interactive map with Folium results : Task 1



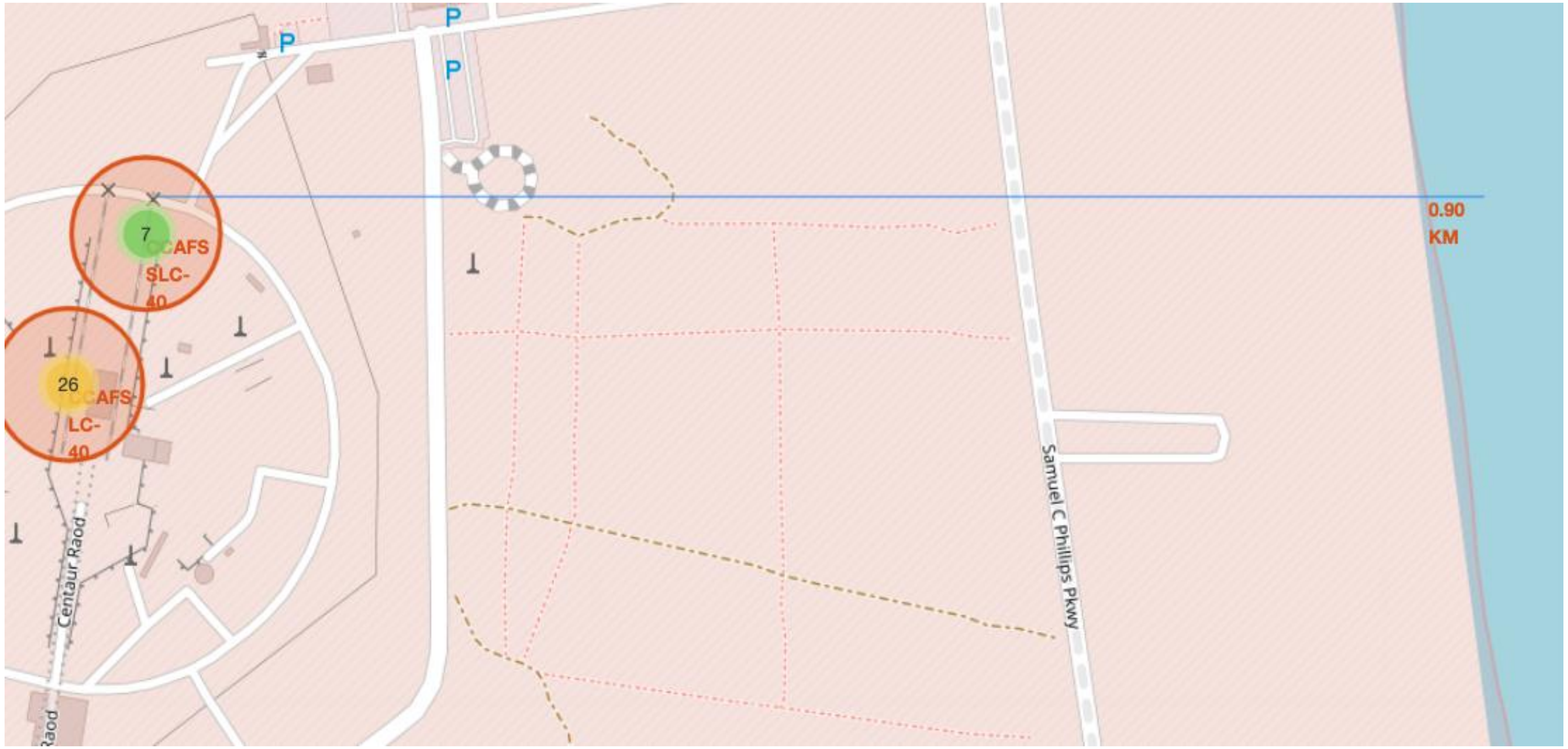

```python
site_map = folium.Map()
for lat,lgt,site in zip(launch_sites_df['Lat'], launch_sites_df['Long'], launch_sites_df['Launch Site']):
    # For each launch site, add a Circle object based on its coordinate (Lat, Long) values. In addition, add Launch site name as a popup label
    circle = folium.Circle([lat,lgt], radius=100, color='#d35400', fill=True).add_child(folium.Popup(str(site)))
    # Create a blue circle at NASA Johnson Space Center's coordinate with a icon showing its name
    marker = folium.map.Marker(
    [lat,lgt],
    # 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>' % str(site),
        ))
    site_map.add_child(circle)
    site_map.add_child(marker)
site_map
```

RESULT: interactive map with Folium results : Task 2

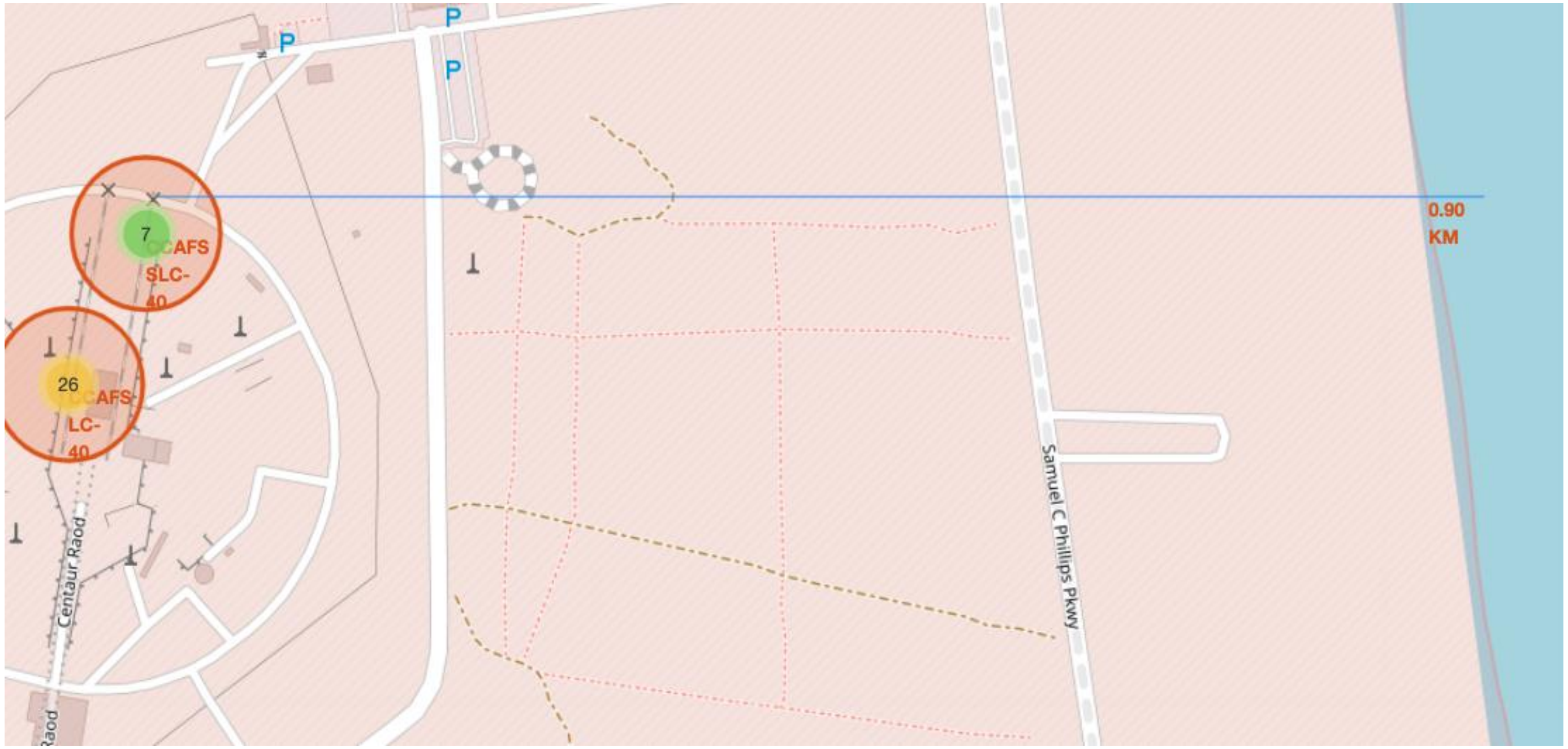



```
icons=[folium.Icon(icon="car",
color='white', icon_color=color,
prefix="fa") for  _ in
range(len(locations))]
marker_cluster=
MarkerCluster(locations=locations,
icons=icons)
site_map.add_child(marker_cluster)
site_map
```

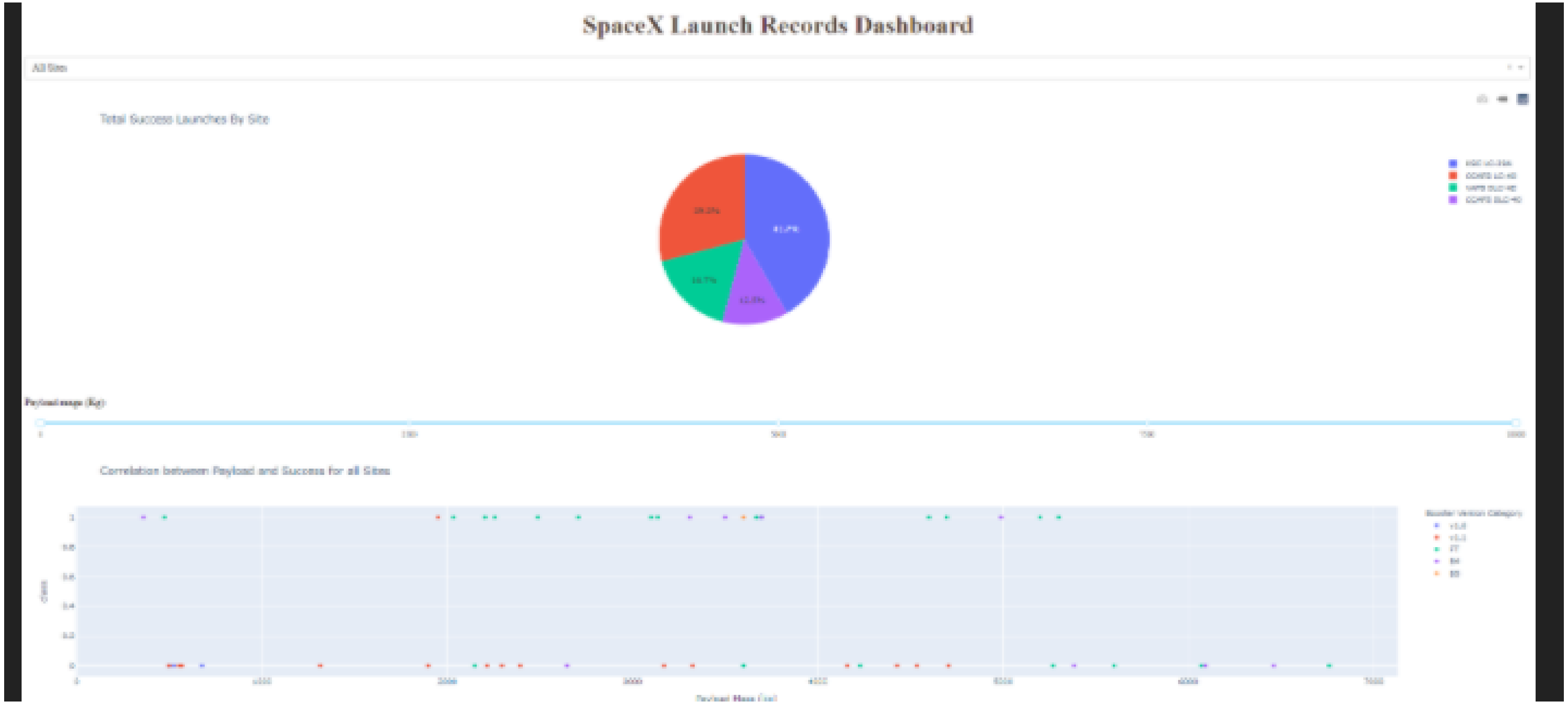
RESULT: interactive map with Folium results : Task 3



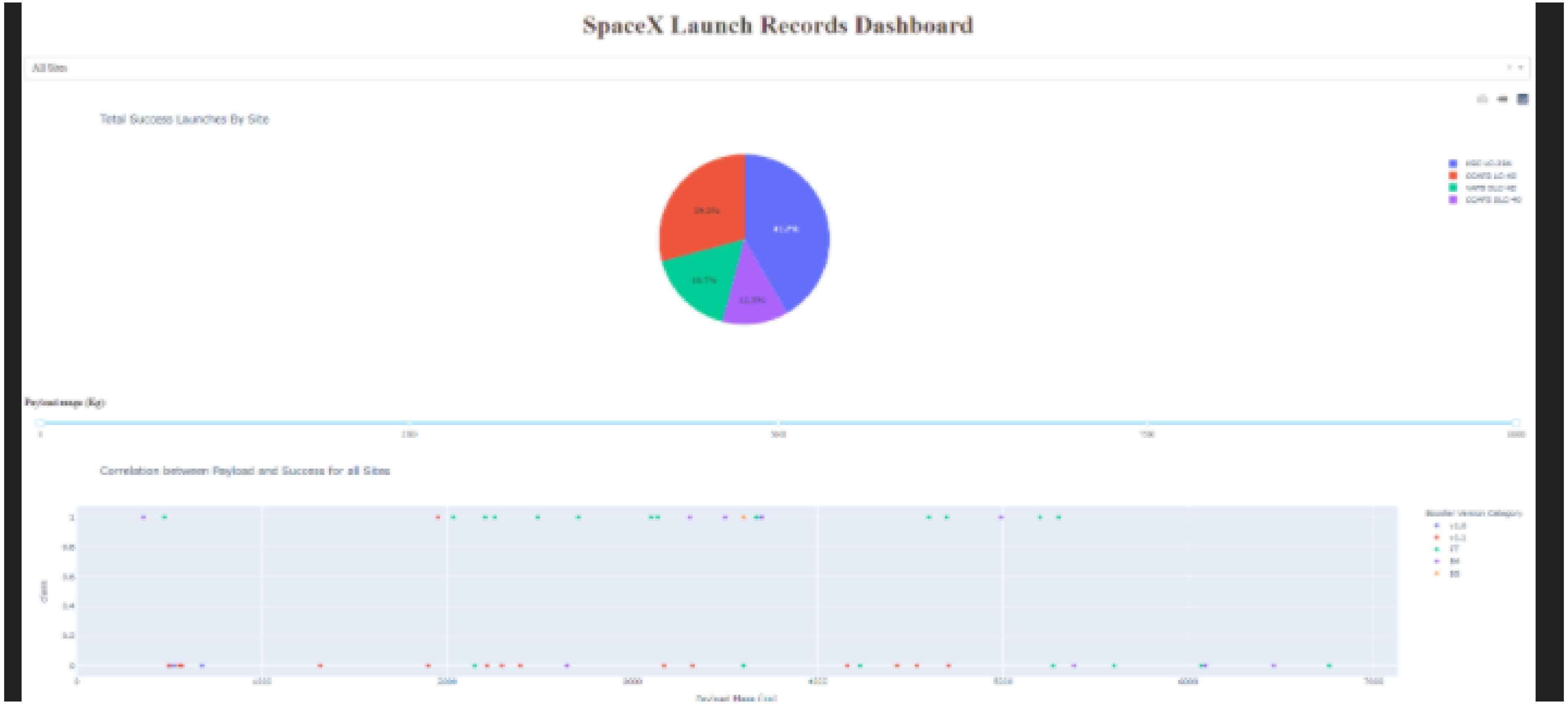
RESULT: interactive map with Folium results : Task 3



RESULT: Plotly Dash dashboard results :



RESULT: Plotly Dash dashboard :



RESULT: predictive analysis (classification) : task 1

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.Series(data['Class'].to_numpy())
```

```
In [6]: Y
```

```
Out[6]: 0     0
        1     0
        2     0
        3     0
        4     0
             ..
        85    1
        86    1
        87    1
        88    1
        89    1
        Length: 90, dtype: int64
```

RESULT: predictive analysis (classification) : task 2

TASK 2

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

7]:
```
# students get this
X= preprocessing.StandardScaler().fit(X).transform(X)
```

8]:
```
X[0:5]
```

8]:
```
array([[-1.71291154e+00, -1.94814463e-16, -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,
        -7.97724035e-01, -1.50755672e-01, -3.92232270e-01,
         9.43398113e+00, -1.05999788e-01, -1.05999788e-01,
        -1.05999788e-01, -1.05999788e-01, -1.05999788e-01,
        -1.05999788e-01, -1.05999788e-01, -1.05999788e-01,
        -1.05999788e-01, -1.05999788e-01, -1.05999788e-01,
        -1.05999788e-01, -1.05999788e-01, -1.05999788e-01,
        -1.05999788e-01, -1.05999788e-01, -1.05999788e-01,
        -1.05999788e-01, -1.50755672e-01, -1.05999788e-01,
        -1.05999788e-01, -1.05999788e-01, -1.05999788e-01,
        -1.05999788e-01, -1.50755672e-01, -1.05999788e-01,
        -1.50755672e-01, -1.50755672e-01, -1.05999788e-01,
        -1.50755672e-01, -1.50755672e-01, -1.05999788e-01,
        -1.05999788e-01, -1.50755672e-01, -1.50755672e-01,
        -1.50755672e-01, -1.05999788e-01, -1.05999788e-01,
        -1.05999788e-01, -1.50755672e-01, -2.15665546e-01,
        -1.85695338e-01, -2.15665546e-01, -2.67261242e-01,
        -1.05999788e-01, -2.42535625e-01, -1.05999788e-01,
        -2.15665546e-01, -1.85695338e-01, -2.15665546e-01,
        -1.85695338e-01, -1.05999788e-01,  1.87082869e+00,
```

RESULT: predictive analysis (classification) : task 3

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 d
and test data should be assigned to the following labels.

`X_train, X_test, Y_train, Y_test`

```
X_train, X_test, Y_train, Y_test = train_test_split( X, Y, test_size=0.2, random_state=2)
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,)
```

we can see we only have 18 test samples.

```
Y_test.shape
```

```
(18,)
```

RESULT: predictive analysis (classification) : task 4

TASK 4

Create a logistic regression object using 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, param_grid=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
```

RESULT: predictive analysis (classification) : task 5

TASK 5

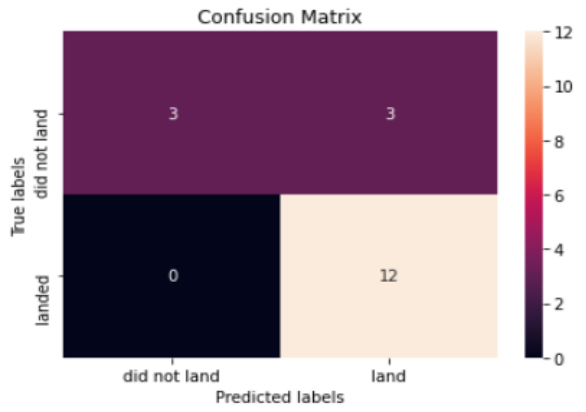
Calculate the accuracy on the test data using the method `score`:

```
33]: result=logreg_cv1.fit(X_train, Y_train)
     result.score(X_test, Y_test)
```

```
33]: 0.8333333333333334
```

Lets look at the confusion matrix:

```
34]: 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.

RESULT: predictive analysis (classification) : task 6

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`.

```
]: 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, 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
```

RESULT: predictive analysis (classification) : task 7

TASK 7

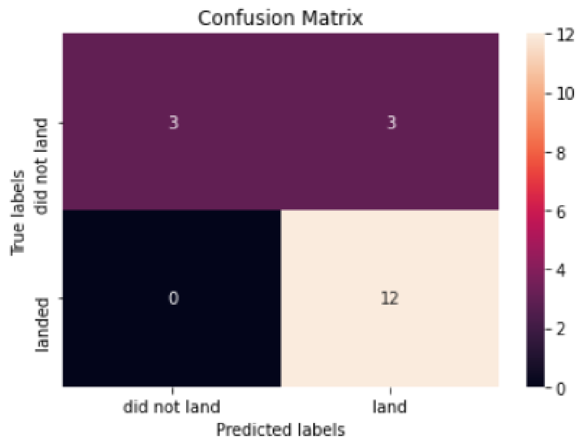
Calculate the accuracy on the test data using the method `score`:

```
37]: svm_cv.score(X_test, Y_test)
```

```
37]: 0.8333333333333334
```

We can plot the confusion matrix

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



RESULT: predictive analysis (classification) : task 8

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 [43]: 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 [44]: tree_cv = GridSearchCV(tree, param_grid=parameters, cv=10)
         tree_cv.fit(X_train, Y_train)

Out[44]: 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']})
```

```
In [46]: print("tuned hpyerparameters :(best parameters) ",tree_cv.best_params_)
         print("accuracy :",tree_cv.best_score_)

         tuned hpyerparameters :(best parameters)  {'criterion': 'gini', 'max_depth': 14, 'max_features': 'sqrt', 'min_samples_leaf': 4,
         'min_samples_split': 5, 'splitter': 'best'}
         accuracy : 0.875
```

RESULT: predictive analysis (classification) : task 9

TASK 9

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

```
]: treemodel= DecisionTreeClassifier(criterion= 'gini', max_depth= 14, max_features= 'sqrt', min_samples_leaf= 4, min_samples_split
   = 5, splitter= 'best')
   treemodel.fit(X_train, Y_train)
   treemodel.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)
```

RESULT: predictive analysis (classification) : task 10

TASK 10

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

```
0]: 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()
```

```
1]: knn_cv = GridSearchCV(KNN, param_grid=parameters, cv=10)
    knn_cv.fit(X_train, Y_train)
```

```
1]: 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]})
```

```
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
```

RESULT: predictive analysis (classification) : task 11 and 12

TASK 11

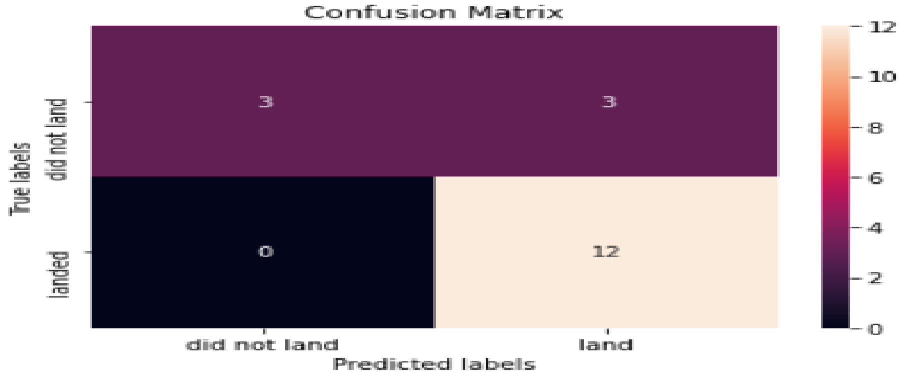
Calculate the accuracy of tree_cv on the test data using the method `score`:

```
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)
```



TASK 12

Find the method performs best:

"the decision tree"

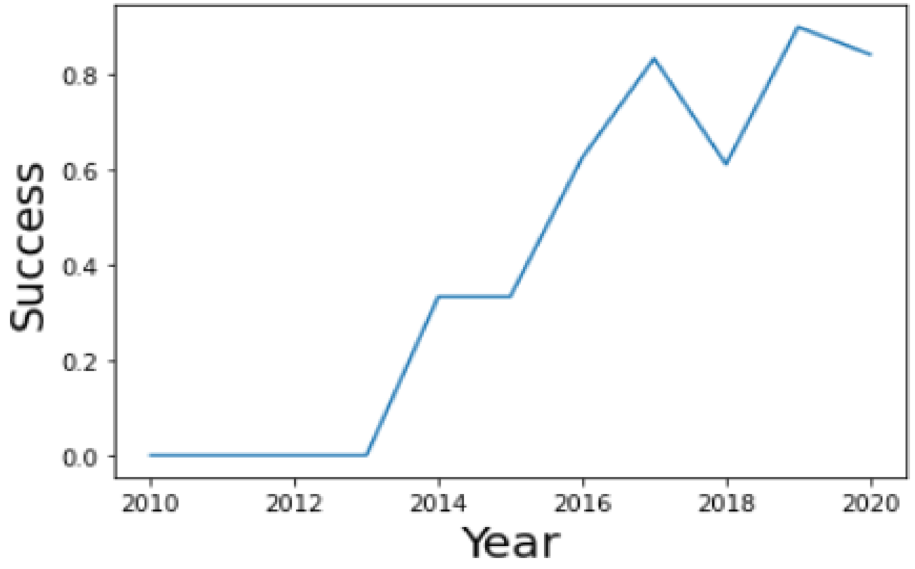
CONCLUSION

- After an indepth review of data many the conclusion is that SpaceX claim that Falcon lower cost (62 million dollars) is because SpaceX can reuse the first stage has been confirmed.
- The outcome of launching rockets has been increasingly successful as a result of reusing the first stage

APPENDIX

Increasing successful outcome over the years

```
]: # Plot a line chart with x axis to be the extracted year and y axis to be the success rate
sns.lineplot(y="Success", x="Year", data=dfnew1)
plt.xlabel("Year",fontsize=20)
plt.ylabel("Success",fontsize=20)
plt.show()
```



GITHUB Link

This is the link to the github site with project material:

https://github.com/Robertboy18/IBM-Data-Science/tree/master/Applied%20Data%20Science%20Capstone