Data Science Capstone Project

Othmane BLIAL

outline

- Executive Summary
- Introduction
- Methodology
- Results
- Conclusion

Executive Summary

- Data was sourced from the public SpaceX API and its Wikipedia page. A 'class' column was created to indicate successful landings. The data was analyzed using SQL, visualizations, folium maps, and dashboards. Key columns were selected as features and categorical variables were converted to binary format using one-hot encoding. Data standardization was followed by employing GridSearchCV to optimize parameters for machine learning models. The performance of four models—Logistic Regression, Support Vector Machine, Decision Tree Classifier, and K Nearest Neighbors—was visualized, each yielding an accuracy of approximately 83.33%. All models tended to overestimate successful landings, indicating a need for more data to enhance model accuracy and effectiveness.

Methodology

- Data collection methodology: Combined data from SpaceX public API and SpaceX Wikipedia page
- Perform data wrangling: Classifying true landings as successful and unsuccessful otherwise
- Perform exploratory data analysis (EDA) using visualization and SQL
- Perform interactive visual analytics using Folium and Plotly Dash
- Perform predictive analysis using classification models: Tuned models using GridSearchCV

Methodology

Data Collection Process

- The data collection process utilized API requests from SpaceX's public API and scraped data from SpaceX's Wikipedia page. The following slides will detail the data collection flowcharts for both API and web scraping. SpaceX API data includes columns such as FlightNumber, Date, BoosterVersion, PayloadMass, Orbit, and others. The data scraped from Wikipedia includes Flight No., Launch Site, Payload, and additional details.

Data collection from spaceX API

- The data collection process for SpaceX API involves requesting data, filtering for Falcon 9 launches, and handling missing PayloadMass values. Data is extracted from JSON files, converted into a DataFrame, and relevant dictionary data is cast to a DataFrame.
- Link Github: https://github.com/OthmaneBlial/DataScience/blob/main/Lab%201-1%3A%20Collecting%20the%20data%20-%20solution.ipynb

Data collection web scrapping

- The web scraping data collection process includes requesting Wikipedia's HTML, parsing it with BeautifulSoup using the html5lib parser, identifying the launch information table, extracting data into a dictionary, and finally converting that dictionary into a DataFrame.
- Link Github: https://github.com/OthmaneBlial/DataScience/blob/main/Lab%201-2%3A%20Data%20Collection%20with%20Web%20Scraping%20lab%20-%20solution.ipynb

Data wrangling

- Develop a training label 'class' for landing outcomes: assign 1 for successful landings ('True ASDS', 'True RTLS', 'True Ocean') and 0 for failures ('None None', 'False ASDS', 'None ASDS', 'False Ocean', 'False RTLS'). The label reflects the 'Mission Outcome'.
- Link Github: https://github.com/OthmaneBlial/DataScience/blob/main/Lab%202%3A%20Data%20wrangling%20-%20solution.ipynb

EDA with Data Visualization

- Exploratory Data Analysis examined variables like Flight Number, Payload Mass, Launch Site, Orbit, Class, and Year using scatter plots, line charts, and bar plots to analyze relationships. Plots included comparisons of Flight Number vs. Payload Mass, and Orbit vs. Success Rate, among others, to inform machine learning model training.
- Link Github: https://github.com/OthmaneBlial/DataScience/blob/main/Lab%203%3A%20EDA%20with%20Visualization%20-%20solution.ipynb

EDA with SQL

- Data was loaded into an IBM DB2 Database and queried using SQL Python integration to understand key aspects like launch sites, mission outcomes, payload sizes, booster versions, and landing outcomes.
- Link Github: https://github.com/OthmaneBlial/DataScience/blob/main/Lab%204%3A%20EDA%20with%20SQL%20-%20solution.ipynb

Build an interactive map with Folium

- Folium maps display Launch Sites, marking successful and unsuccessful landings and proximity to railways, highways, coasts, and cities. This visualization aids in understanding the strategic placement of launch sites and the impact of location on landing success.
- Link Github: https://github.com/OthmaneBlial/DataScience/blob/main/Lab%205%3A%20Interactive%20Visual%20Analytics%20with%20Folium%20lab%20-%20solution.ipynb

Build a Dashboard with Plotly Dash

- The dashboard features a pie chart and scatter plot. The pie chart displays the distribution of successful landings by launch site, either collectively or individually. The scatter plot compares success across launch sites, payload masses (0-10,000 kg), and booster versions, allowing for detailed analysis.
- Link Github: https://github.com/OthmaneBlial/DataScience/blob/main/spacex_dash_app.py

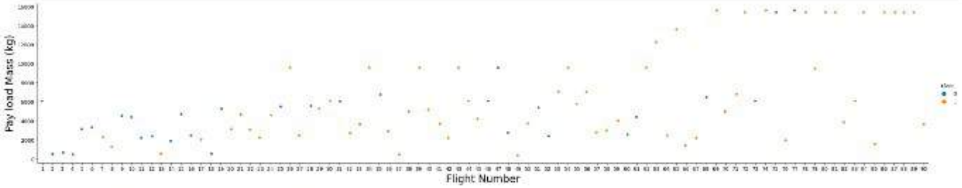
Machine Learning Prediction

- This focuses on predicting the landing outcome of SpaceX's Falcon 9 rocket's first stage using machine learning, aiming to assess launch costs effectively. SpaceX, boasting launches at $62 million versus competitors' $165 million, attributes savings to reusability. The project involves exploratory data analysis, training label creation, data standardization, and model training/testing with SVM, Classification Trees, and Logistic Regression to identify the most accurate prediction method.
- Link Github: https://github.com/OthmaneBlial/DataScience/blob/main/SpaceX_Machine%20Learning%20Prediction_Part_5_othmane.ipynb

Results

EDA with Visualization: Results

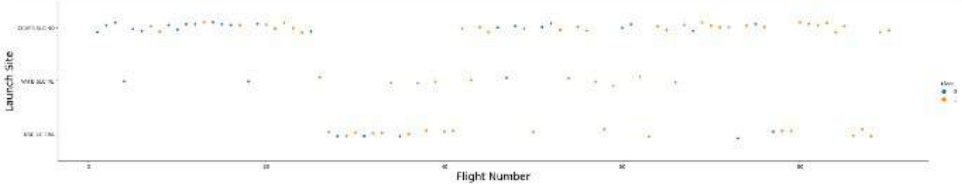
```
sns.catplot(y="PayloadMass", x="FlightNumber", hue="Class", data=df, aspect = 5)
plt.xlabel("Flight Number",fontsize=20)
plt.ylabel("Pay load Mass (kg)",fontsize=20)
plt.show()
```



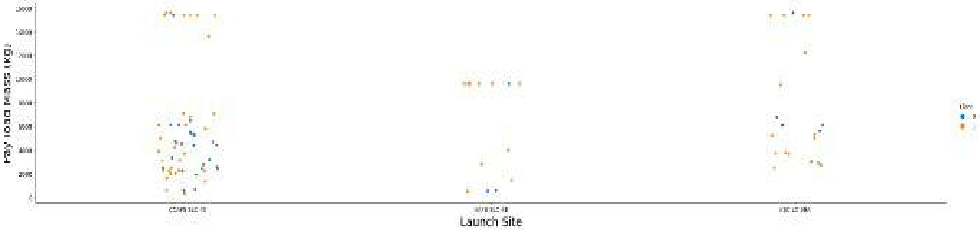
We see that different launch sites have different success rates. `CCAFS LC-40`, has a success rate of 60 %, while `KSC LC-39A` and `VAFB SLC 4E` has a success rate of 77%.

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

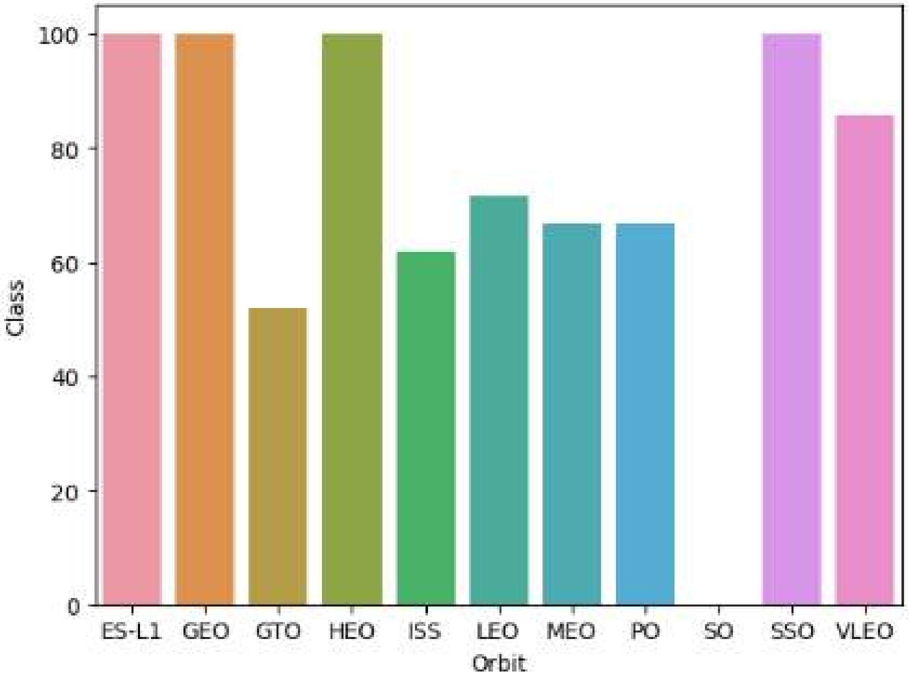
```
### TASK 1: Visualize the relationship between Flight Number and Launch Site
sns.catplot(y="LaunchSite", x="FlightNumber", hue="Class", data=df, aspect = 5)
plt.xlabel("Flight Number",fontsize=20)
plt.ylabel("Launch Site",fontsize=20)
plt.show()
```



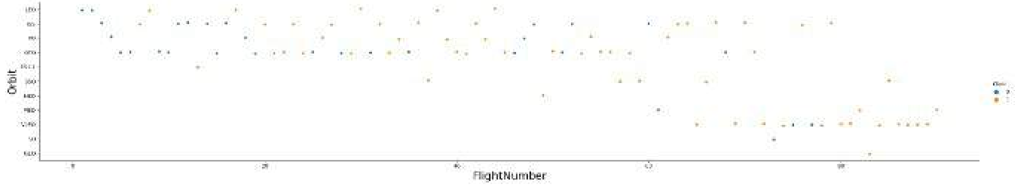

```
### TASK 2: Visualize the relationship between Payload and Launch Site
sns.catplot(y="PayloadMass", x="LaunchSite", hue="Class", data=df, aspect = 5)
plt.xlabel("Launch Site",fontsize=20)
plt.ylabel("Pay load Mass (kg)",fontsize=20)
plt.show()
```



EDA with Visualization: Results



[5]:
```
### TASK  4: Visualize the relationship between FlightNumber and Orbit type
sns.catplot(y="Orbit", x="FlightNumber", hue="Class", data=df, aspect = 5)
plt.xlabel("FlightNumber",fontsize=20)
plt.ylabel("Orbit",fontsize=20)
plt.show()
```



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

[ ]:
```
# Plot a scatter point chart with x axis to be FlightNumber and y axis to be the Orbit, and hue to be the class
```

You should see that in the LEO orbit the Success appears related to the number of flights; on the other hand, there seems to be no relationship between flight number when in GTO orbit.

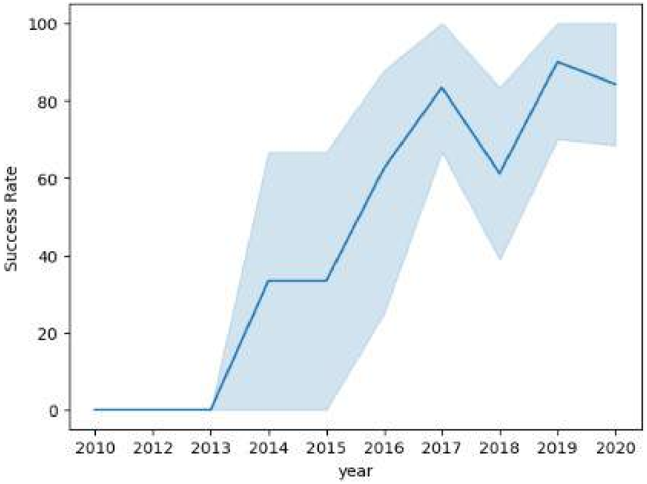
[6]:
```
### TASK  5: Visualize the relationship between Payload and Orbit type
sns.catplot(y="Orbit", x="PayloadMass", hue="Class", data=df, aspect = 5)
plt.xlabel("PayloadMass",fontsize=20)
plt.ylabel("Orbit",fontsize=20)
plt.show()
```

EDA with Visualization: Results

```python
# A function to Extract years from the date
def Extract_year(year):
    for i in df["Date"]:
        year.append(i.split("-")[0])
    return year
```

```python
# Plot a line chart with x axis to be the extracted year and y axis to be the success rate
year = []
df["year"] = Extract_year(year)
df["Success Rate"] = df["Class"] * 100
sns.lineplot(data = df, x = "year", y = "Success Rate")
```

<AxesSubplot:xlabel='year', ylabel='Success Rate'>




```python
### TASK  7: Create dummy variables to categorical columns
oh_orbit = pd.get_dummies(features["Orbit"])
oh_launch = pd.get_dummies(features["LaunchSite"])
oh_landing = pd.get_dummies(features["LandingPad"])
oh_serial = pd.get_dummies(features["Serial"])
remainder = features[["FlightNumber","PayloadMass", "Flights", "GridFins", "Reused", "Legs", "Block","ReusedCount
features_one_hot = pd.concat([oh_launch, oh_landing, oh_serial, oh_orbit], axis=1)
features_one_hot.head()
```

	CCAFS SLC 40	KSC LC 39A	VAFB SLC 4E	5e9e3032383ecb267a34e7c7	5e9e3032383ecb554034e7c9	5e9e3032383ecb6bb234e7ca	5e9e3032383ecb761634e
0	1	0	0	0	0	0	
1	1	0	0	0	0	0	
2	1	0	0	0	0	0	
3	0	0	1	0	0	0	
4	1	0	0	0	0	0	

5 rows × 72 columns

```python
### TASK  8: Cast all numeric columns to `float64`
features_one_hot.astype('float64')
```

	CCAFS SLC 40	KSC LC 39A	VAFB SLC 4E	5e9e3032383ecb267a34e7c7	5e9e3032383ecb554034e7c9	5e9e3032383ecb6bb234e7ca	5e9e3032383ecb761634
0	1.0	0.0	0.0	0.0	0.0	0.0	
1	1.0	0.0	0.0	0.0	0.0	0.0	
2	1.0	0.0	0.0	0.0	0.0	0.0	
3	0.0	0.0	1.0	0.0	0.0	0.0	
4	1.0	0.0	0.0	0.0	0.0	0.0	
...	...	...	...	...	...	...	
85	0.0	1.0	0.0	0.0	0.0	1.0	
86	0.0	1.0	0.0	0.0	0.0	1.0	
87	0.0	1.0	0.0	0.0	0.0	1.0	
88	1.0	0.0	0.0	0.0	0.0	0.0	
89	1.0	0.0	0.0	0.0	0.0	1.0	

90 rows × 72 columns

EDA with SQL: Results

Task 1

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

```
%load_ext sql
%sql sqlite:///my_data1.db

%sql SELECT DISTINCT LAUNCH_SITE FROM SPACEXTBL
```

```
The sql extension is already loaded. To reload it, use:
  %reload_ext sql
 * sqlite:///my_data1.db
Done.
```

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

Task 2

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

```
%load_ext sql
%sql sqlite:///my_data1.db

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

```
The sql extension is already loaded. To reload it, use:
  %reload_ext sql
 * sqlite:///my_data1.db
Done.
```

Date	Time_(UTC)	Booster_Version	Launch_Site	Payload	PAYLOAD_MASS__KG_	Orbit	Customer	Mission_Outcome	Landing_Ou
2010-06-04	18:45:00	F9 v1.0 B0003	CCAFS LC-40	Dragon Spacecraft Qualification Unit	0	LEO	SpaceX	Success	Failure (par
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 (par

Task 3

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

```
%load_ext sql
%sql sqlite:///my_data1.db

%sql SELECT SUM(PAYLOAD_MASS__KG_) as Total_Payload_Mass FROM SPACEXTBL WHERE Customer LIKE '%NASA (CRS)%'
```

```
The sql extension is already loaded. To reload it, use:
  %reload_ext sql
 * sqlite:///my_data1.db
Done.
```

Total_Payload_Mass
48213

Task 4

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

```
%load_ext sql
%sql sqlite:///my_data1.db

%sql SELECT AVG(PAYLOAD_MASS__KG_) as Average_Payload_Mass FROM SPACEXTBL WHERE Booster_Version = 'F9 v1.1'
```

```
The sql extension is already loaded. To reload it, use:
  %reload_ext sql
 * sqlite:///my_data1.db
Done.
```

Average_Payload_Mass
2928.4

Task 5

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

Hint:Use min function

```
%load_ext sql
%sql sqlite:///my_data1.db

%sql SELECT MIN(Date) as First_Successful_Ground_Pad_Landing FROM SPACEXTBL WHERE Landing_Outcome = 'Success (gro
```

```
The sql extension is already loaded. To reload it, use:
  %reload_ext sql
 * sqlite:///my_data1.db
Done.
```

First_Successful_Ground_Pad_Landing
2015-12-22

EDA with SQL: Results

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

```
%load_ext sql
%sql sqlite:///my_data1.db

%sql SELECT DISTINCT Booster_Version FROM SPACEXTBL WHERE Landing_Outcome = 'Success (drone ship)' AND PAYLOAD_MA
```

```
The sql extension is already loaded. To reload it, use:
  %reload_ext sql
 * sqlite:///my_data1.db
Done.
```

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

Task 7

List the total number of successful and failure mission outcomes

```
%load_ext sql
%sql sqlite:///my_data1.db

%sql SELECT Mission_Outcome, COUNT(*) as Total FROM SPACEXTBL GROUP BY Mission_Outcome HAVING Mission_Outcome LIK
```

```
The sql extension is already loaded. To reload it, use:
  %reload_ext sql
 * sqlite:///my_data1.db
Done.
```

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

Task 8

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

```
%load_ext sql
%sql sqlite:///my_data1.db

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

```
The sql extension is already loaded. To reload it, use:
  %reload_ext sql
 * 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
F9 B5 B1049.7

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.

Note: SQLLite does not support monthnames. So you need to use substr(Date, 6,2) as month to get the months and substr(Date,0,5)='2015' for year.

```
%load_ext sql
%sql sqlite:///my_data1.db

%sql SELECT substr(Date, 6, 2) AS Month, Booster_Version, Launch_Site, Landing_Outcome FROM SPACEXTBL WHERE subst
```

```
The sql extension is already loaded. To reload it, use:
  %reload_ext sql
 * sqlite:///my_data1.db
Done.
```

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

Interactive Map with Folium: Results

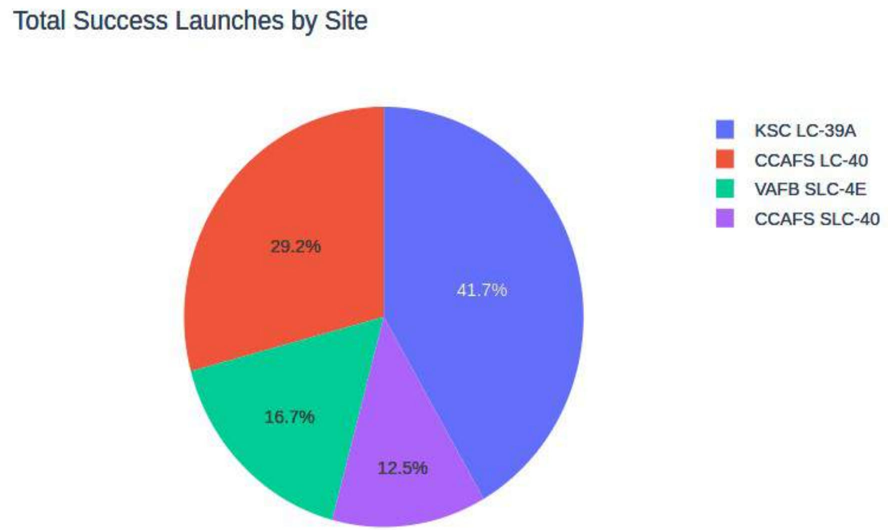
```
response = requests.get(URL)

spacex_csv_file = io.BytesIO(response.content)
spacex_df = pd.read_csv(spacex_csv_file)
print(spacex_df.head())
```

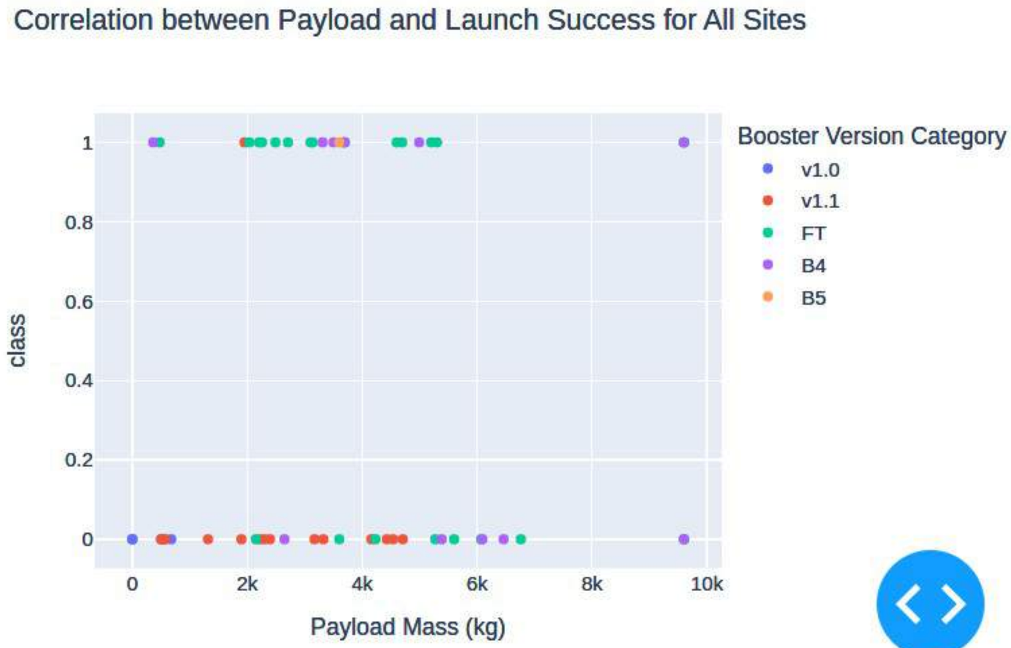
```
  Flight Number        Date Time (UTC) Booster Version  Launch Site  \
0             1  2010-06-04   18:45:00  F9 v1.0  B0003  CCAFS LC-40
1             2  2010-12-08   15:43:00  F9 v1.0  B0004  CCAFS LC-40
2             3  2012-05-22    7:44:00  F9 v1.0  B0005  CCAFS LC-40
3             4  2012-10-08    0:35:00  F9 v1.0  B0006  CCAFS LC-40
4             5  2013-03-01   15:10:00  F9 v1.0  B0007  CCAFS LC-40

                                        Payload  Payload Mass (kg)  \
0            Dragon Spacecraft Qualification Unit                0.0
1  Dragon demo flight C1, two CubeSats,  barrel o...             0.0
2                            Dragon demo flight C2+            525.0
3                                    SpaceX CRS-1            500.0
4                                    SpaceX CRS-2            677.0

      Orbit         Customer       Landing Outcome  class        Lat  \
0       LEO           SpaceX  Failure  (parachute)      0  28.562302
1  LEO (ISS)  NASA (COTS) NRO  Failure  (parachute)     0  28.562302
2  LEO (ISS)      NASA (COTS)           No attempt      0  28.562302
3  LEO (ISS)       NASA (CRS)           No attempt      0  28.562302
4  LEO (ISS)       NASA (CRS)           No attempt      0  28.562302

       Long
0 -80.577356
1 -80.577356
2 -80.577356
3 -80.577356
4 -80.577356
```

```python
def func(item):
    if item == 'Site1':
        return 'green'
    else:
        return 'red'

launch_sites_df['marker_color'] = launch_sites_df['Launch Site'].apply(func)
launch_sites_df
```

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

Build a Dashboard with Plotly Dash: Results





Machine Learning Prediction: Results

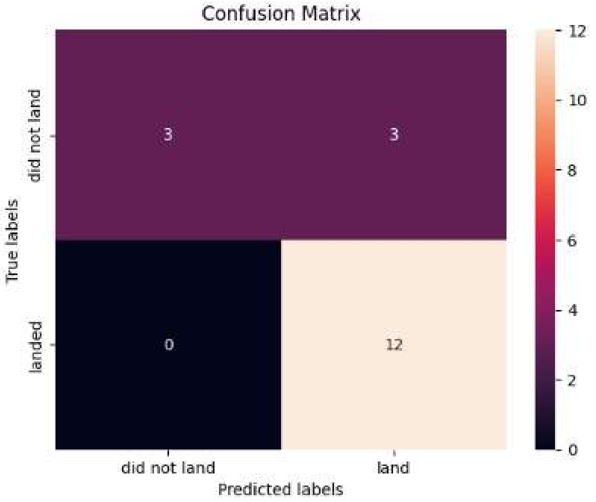
Calculate the accuracy on the test data using the method `score` :

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

```
0.8333333333333334
```

Lets look at the confusion matrix:

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



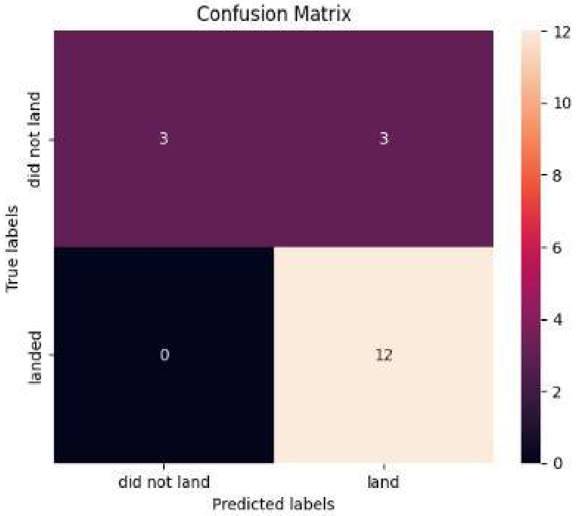
Calculate the accuracy on the test data using the method `score` :

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

```
0.8333333333333334
```

We can plot the confusion matrix

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



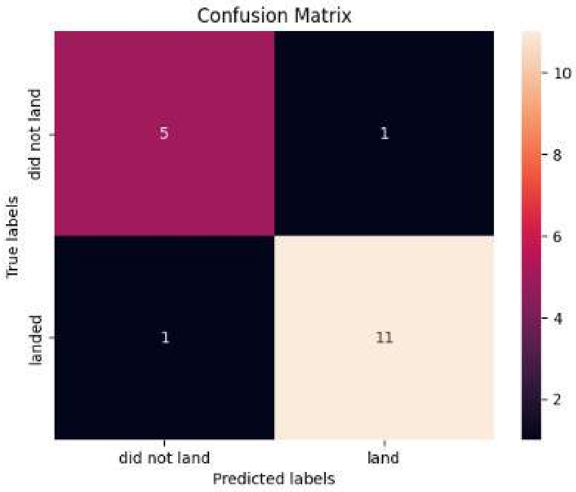
Calculate the accuracy of tree_cv on the test data using the method `score` :

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

```
0.7222222222222222
```

We can plot the confusion matrix

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



Machine Learning Prediction: Results

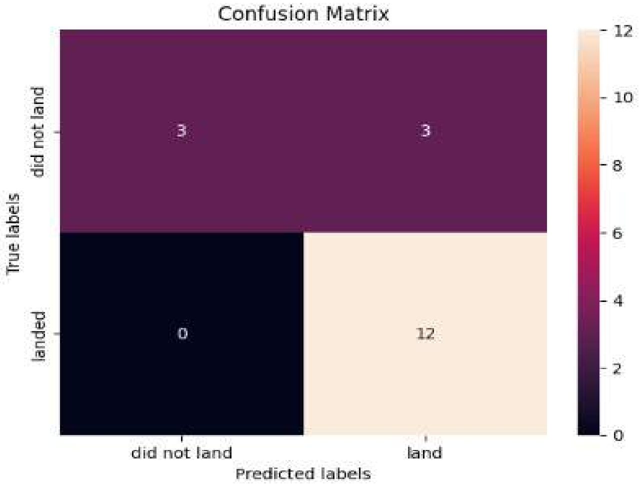
Calculate the accuracy of knn_cv on the test data using the method `score` :

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

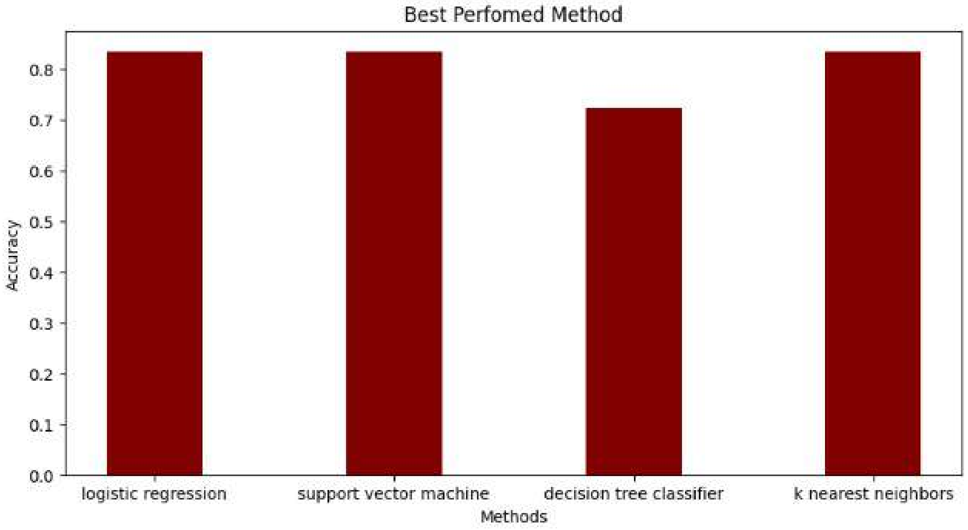
```
t[37]: 0.8333333333333334
```

We can plot the confusion matrix

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




```
import numpy as np
import matplotlib.pyplot as plt


fig = plt.figure(figsize = (10, 5))

plt.bar(methods, accu, color ='maroon',
        width = 0.4)

plt.xlabel("Methods")
plt.ylabel("Accuracy")
plt.title("Best Perfomed Method")
plt.show()
```



Conclusion

Our task was to create a machine learning model for Space Y to compete with SpaceX, aiming to predict successful Stage 1 landings and potentially save about $100 million USD. We sourced data via a public SpaceX API and from SpaceX's Wikipedia page, labeling and storing it in a DB2 SQL database. A dashboard was created for visualization. The model achieved an 83% accuracy, enabling Space Y's Allon Mask to predict successful landings pre-launch, thus aiding launch decisions. Additional data collection is recommended to enhance model accuracy.