IBM DS Certificate – Capstone Project

CHENYU WANG

HTTPS://GITHUB.COM/NIAMABIE/IBM-DS-CAPSTONE

Executive Summary

- In this capstone project we collected data through web scrapping and accessing the public API. We subsequently explored the insights of data using SQL and data visualization techniques. The main purpose of this project is to predict success of SpaceX Stage-1 recovery landings. In the main Machine Learning prediction step, we engineered certain features and encoded certain categorical variables. We also standardized data and tuned the ML models

- Four ML models were used (Logistic Regression, SVM, Decision Tree Classifier, KNN), and they all produced very similar results. Additional data will be needed for better modeling

Introduction

- SpaceX has been successful in commercial rocket launches. This is largely because of its competitive pricing. The reason that their business is viable with such competitive pricing is that the Stage-1 propulsion parts of SpaceX rockets are specially designed and can be recovered and reused

- We are tasked to predict success/failure of Stage-1 recovery landings using data analytics and machine learning techniques

Methodology

Summary of Methodology

- Data collection with SpaceX API and web scrapping
- Data wrangling: One-hot encoding applied to convert categorical variables
- EDA with data visualization and SQL
- Predictive analysis with classification models

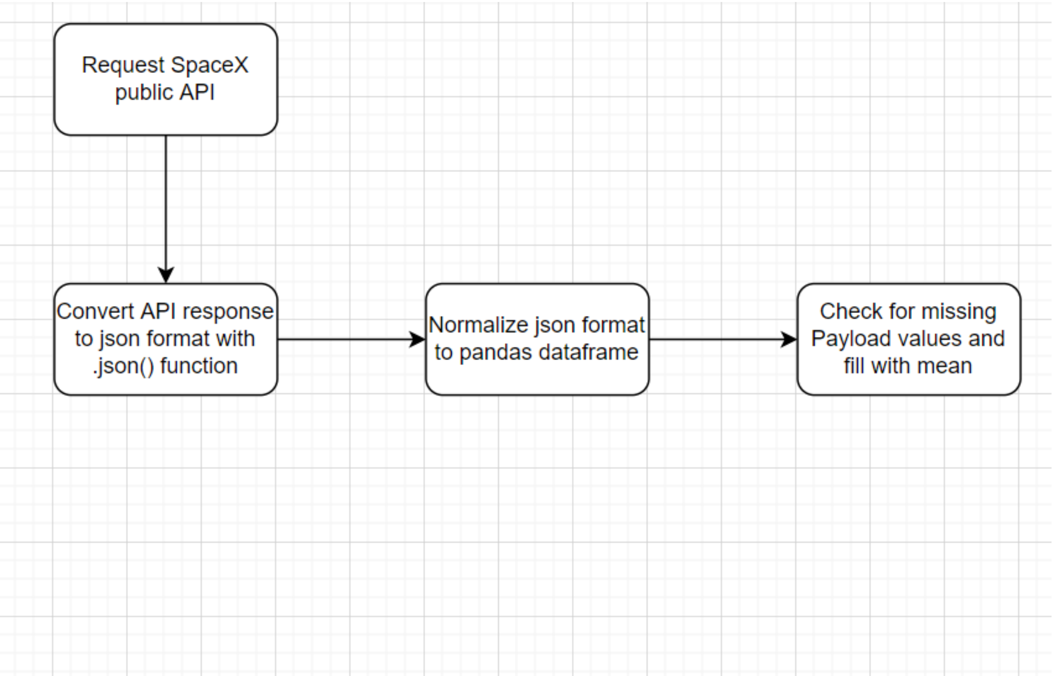
- Folium Map – Did not complete; NOT included in this presentation
- Dashboards – Did not complete; NOT included in this presentation

Data Collection

- The first data collection task was done through access the public SpaceX API
- We then convert the API response to json, and eventually turn it to a pandas dataframe
- We also checked and treated missing values
- The second data collection task was done through web-scrapping the Wikipedia page with BeautifulSoup package
- We parsed the HTML table, and eventually converted it into a pandas dataframe

Data Collection – SpaceX API

https://github.com/niamabie/IBM-DS-Capstone/blob/main/Week1/lab1%20spacex%20api.ipynb



1. Get request for rocket launch data using API

```
In [6]: spacex_url="https://api.spacexdata.com/v4/launches/past"
```

```
In [7]: response = requests.get(spacex_url)
```

2. Use json_normalize method to convert json result to dataframe

```
In [12]: # Use json_normalize method to convert the json result into a dataframe

         # decode response content as json
         static_json_df = res.json()
```

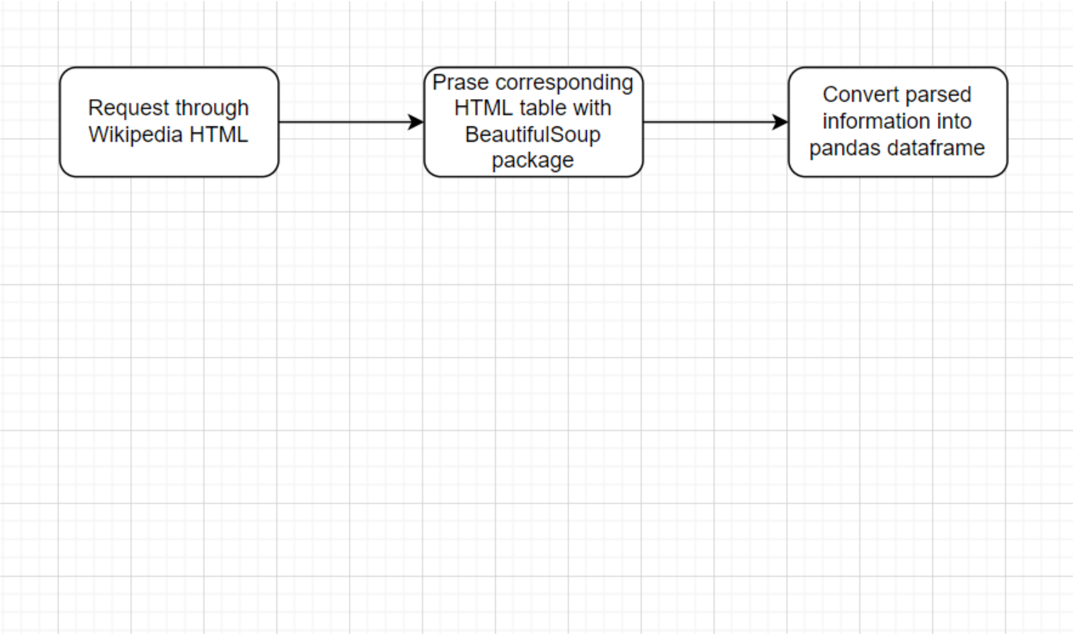
```
In [13]: # apply json_normalize
         data = pd.json_normalize(static_json_df)
```

3. We then performed data cleaning and filling in the missing values

```
In [30]: rows = data_falcon9['PayloadMass'].values.tolist()[0]

         df_rows = pd.DataFrame(rows)
         df_rows = df_rows.replace(np.nan, PayloadMass)

         data_falcon9['PayloadMass'][0] = df_rows.values
         data_falcon9
```

Data Collection – Web Scrapping

https://github.com/niamabie/IBM-DS-Capstone/blob/main/Week1/lab2_data_scrapping.ipynb



1. Apply HTTP Get method to request the Falcon 9 rocket launch page

```
In [4]: static_url = "https://en.wikipedia.org/w/index.php?title=List_of_Falcon_9_and_Falcon_Heavy_launches&oldid=1027686922"
```

```
In [5]: # use requests.get() method with the provided static_url
        # assign the response to a object
        html_data = requests.get(static_url)
        html_data.status_code

Out[5]: 200
```

2. Create a BeautifulSoup object from the HTML response

```
In [6]: # Use BeautifulSoup() to create a BeautifulSoup object from a response text content
        soup = BeautifulSoup(html_data.text, 'html.parser')
```

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

```
In [7]: # Use soup.title attribute
        soup.title

Out[7]: <title>List of Falcon 9 and Falcon Heavy launches - Wikipedia</title>
```

3. Extract all column names from the HTML table header

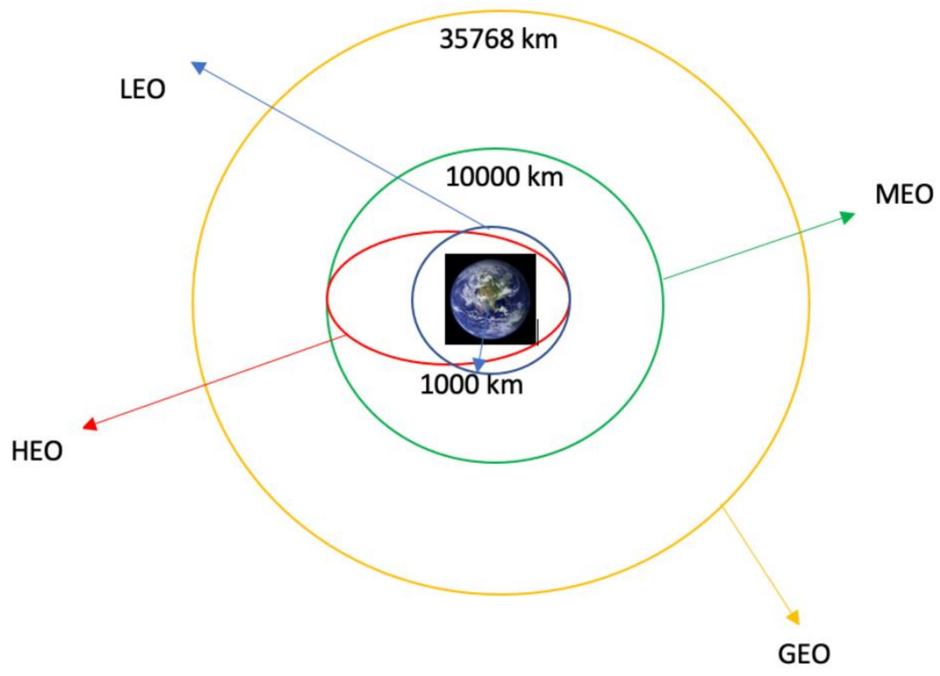
```
In [10]: column_names = []

         # Apply find_all() function with `th` element on first_launch_table
         # Iterate each th element and apply the provided extract_column_from_header() to get a column name
         # Append the Non-empty column name (`if name is not None and len(name) > 0`) into a list called column_names

         element = soup.find_all('th')
         for row in range(len(element)):
             try:
                 name = extract_column_from_header(element[row])
                 if (name is not None and len(name) > 0):
                     column_names.append(name)
             except:
                 pass
```

4. Create a dataframe by parsing the launch HTML tables
5. Export data to csv

Data Wrangling

https://github.com/niamabie/IBM-DS-Capstone/blob/main/Week1/lab3_data_wrangling.ipynb

- We performed EDA on dataset
- We calculated number of launches at each launch site
- We also calculated the number and occurrence of each orbit
- Created landing outcome label
- Saved and exported data to .csv



EDA - Visualization

- We have completed multiple visualization tasks in this step, including scatter plots, bar graphs, and line graphs

- Topics of the plots include Number of Flights vs. Launch Site, Payload Mass vs. Launch Site, Success Rate by Orbit Types, and Yearly Trends, etc.

https://github.com/niamabie/IBM-DS-Capstone/blob/main/Week2/lab2_eda_visualization.ipynb

EDA - SQL

- We wrote SQL queries within Jupyter Notebook
- We applied SQL EDA to get insights regarding several topics, including names of unique launch sites, total payload mass by boosters, average payload mass by v1.1 Falcon 9, total numbers of successful and failed missions, etc.

https://github.com/niamabie/IBM-DS-Capstone/blob/main/Week2/lab1_eda_sql%20(1).ipynb

Interactive Folium Map

- Did not complete; NOT included in this presentation

Dashboard with Plotly Dash

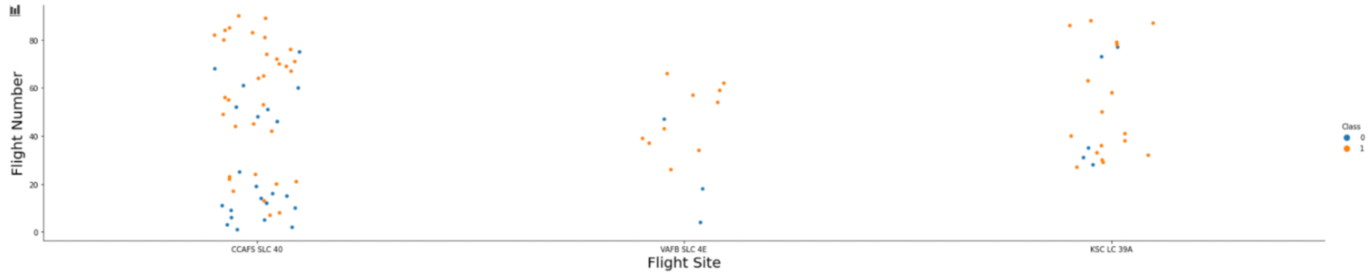
- Did not complete; NOT included in this presentation

Predictive Analysis with ML (Classification)

- We loaded the data using Numpy and pandas packages, transformed the data, and split it into training and test sets (20% vs. 80%)
- We constructed different classification models and tuned hyperparameters using GridSearchCV
- We improved model performance (accuracy) by feature engineering
- We found the best performing model in this project – the Decision Tree Classifier

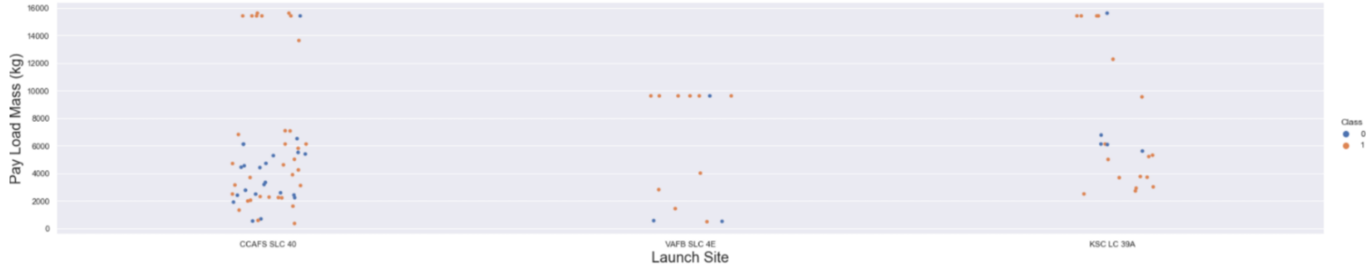
https://github.com/niamabie/IBM-DS-Capstone/blob/main/Week4/lab1_ML_Prediction%20(1).ipynb

Results



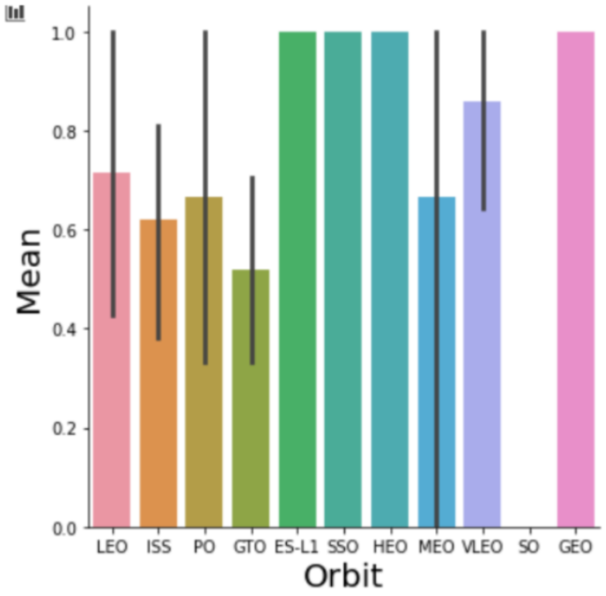
- From the scatter plot, we can see that launch sites with more launches/flights have higher success rate

EDA – Visualization (Number of Flights vs. Launch Site)



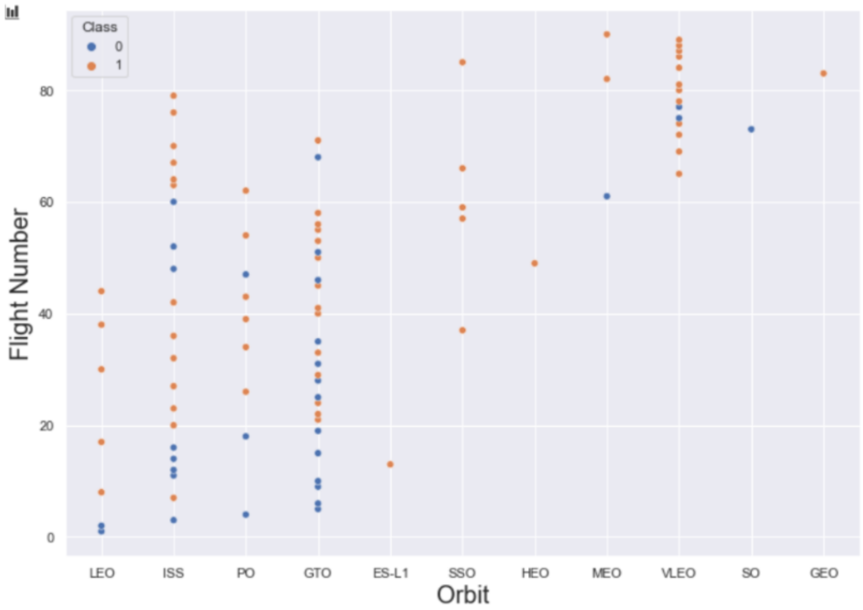
- It is not a clear pattern, but we do see that the greater the payload mass, the higher the success rate at certain launch sites

EDA – Visualization (Payload Mass vs. Launch Site)



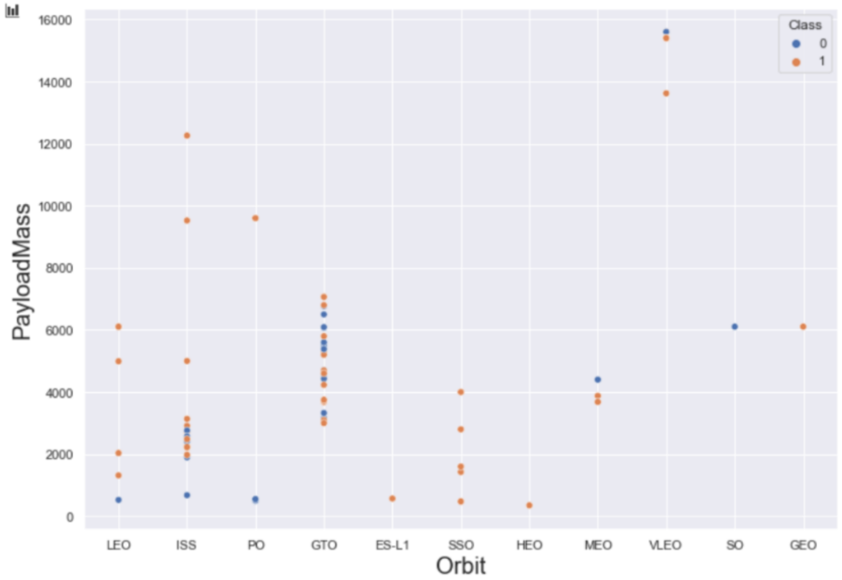
- Orbit ES-L1, SSO, HEO, and GEO have the highest mean success rate

EDA – Visualization (Mean Success Rate by Orbit Type)



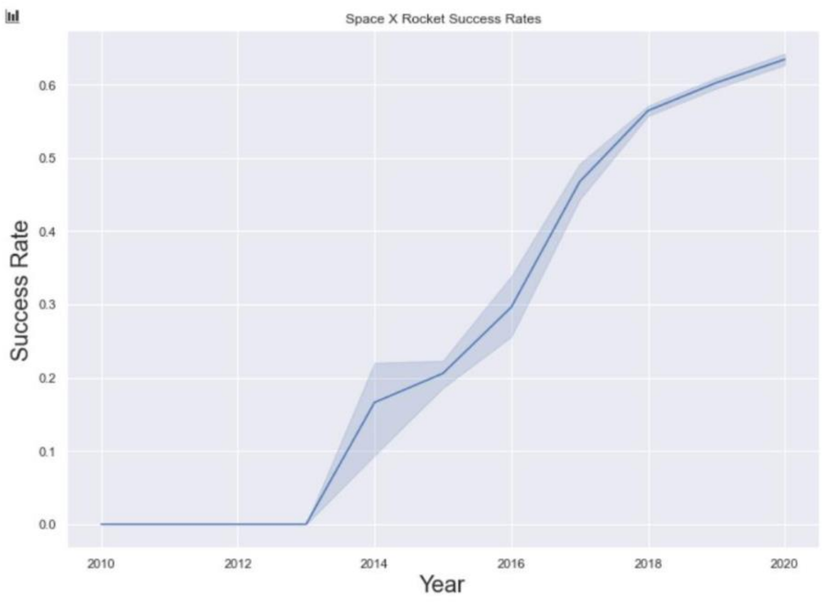
- No clear pattern between number of flights vs. orbit type

EDA – Visualization (Number of Flights vs. Orbit Type)



- We can observe that heavier payload mass has a negative effect on the GTO orbit type

EDA – Visualization (Payload Mass vs. Orbit Type)



- We can observe that the yearly trend of success rate has been going up ever since 2013

EDA – Visualization (Yearly Trend of Success Rate)

EDA –SQL
(All Unique Launch Site Names)

```
'select DISTINCT Launch_Site from tblSpaceX
','Launch_Site'
```

- Using keyword DISTINCT to select unique launch site names from dataset
- The result include: CCAFS LC-40, CCAFS SLC-40, KSC LC-39A, VAFB SLC-4E

```
"select SUM(PAYLOAD_MASS_KG_) TotalPayloadM
ass from tblSpaceX where Customer = 'NASA (
CRS)'",'TotalPayloadMass'
```

- Calculated “SUM” of the group filtered after “WHERE” and “LIKE” keywords

- The result is 45596

EDA – SQL
(Total Payload Mass by NASA)

```
"select AVG(PAYLOAD_MASS_KG_) AveragePayloa
dMass from tblSpaceX where Booster_Version
= 'F9 v1.1'",'AveragePayloadMass'
```

- Similar query, but this time we used AVG calculation instead of SUM

- The result is 2928.4

EDA – SQL
(Average Payload Mass by Falcon 9 v1.1)

```
"select MIN(Date) SLO from tblSpaceX where
Landing_Outcome = 'Success (drone ship)'",'
SLO'
```

- Using MIN function to obtain the first/smallest date of successful mission

- The result is: 06-05-2016

EDA – SQL
(First Successful Landing Date)

```
"select Booster_Version from tblSpaceX wher
e Landing_Outcome = 'Success (ground pad)'
AND Payload_MASS_KG_ > 4000 AND Payload_MAS
S_KG_ < 6000",'Booster_Version'
```

EDA – SQL (Success Landings for Payload Mass 4000-6000)

- Use “WHERE” and “AND” clauses to limit the results
- The result include: F9 FT B1032.1, F9 B4 B1040.1, F9 B4 B1043.1

```
"SELECT(SELECT Count(Mission_Outcome) from
tblSpaceX where Mission_Outcome LIKE '%Succ
ess%') as Successful_Mission_Outcomes,(SELE
CT Count(Mission_Outcome) from tblSpaceX wh
ere Mission_Outcome LIKE '%Failure%') as Fa
ilure_Mission_Outcomes"
```

- Use wildcard search and subquery to return to desired result

- The result is: 100 successes, and 1 failure

EDA – SQL
(Total Number of Success and Failure)

Interactive Folium Map

- Did not complete; NOT included in this presentation

Dashboard with Plotly Dash

- Did not complete; NOT included in this presentation

Predictive Analysis – Logistic Regression

```
[10] 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()
gscv = GridSearchCV(lr,parameters,scoring='accuracy',cv=10)
logreg_cv = gscv.fit(X_train,Y_train)
```

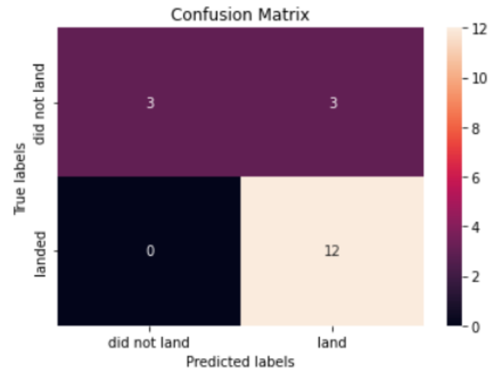
We output the `GridSearchCV` object for logistic regression. We display the best parameters using the data attribute `best_params_` and the accuracy on the validation data using the data attribute `best_score_`.

```
[12] 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
```

```
[13] print('Accuracy=  ',logreg_cv.score(X_test,Y_test))

Accuracy=   0.8333333333333334
```

Lets look at the confusion matrix:

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



Predictive Analysis - SVM

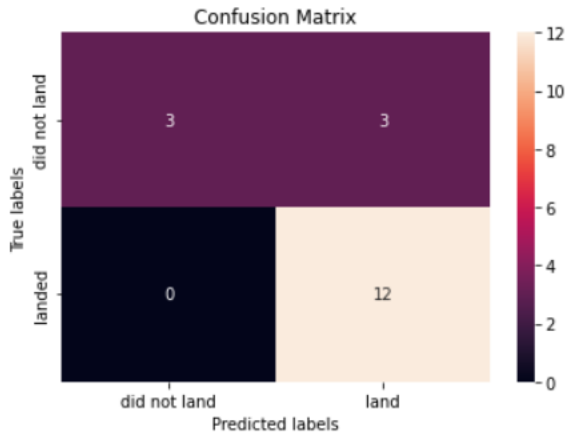
Create a support vector machine object then create a `GridSearchCV` object `svm_cv` with cv - 10. Fit the object to find the l
the dictionary `parameters`.

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

```
[16]
gscv = GridSearchCV(svm,parameters,scoring='accuracy',cv=10)
svm_cv = gscv.fit(X_train,Y_train)
```

```
[17] 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
```

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



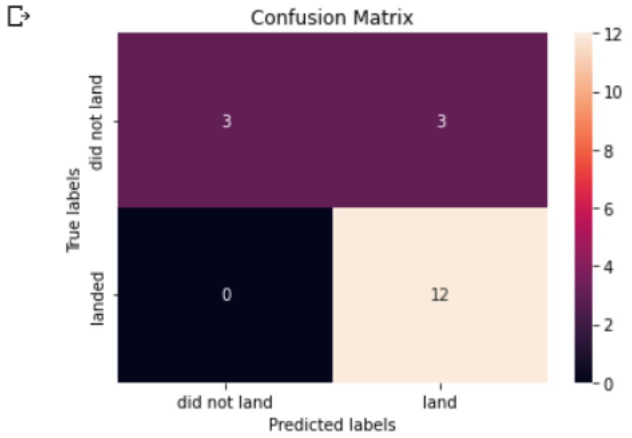
Predictive Analysis – Decision Tree

```
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()
```

```
[21] gscv = GridSearchCV(tree,parameters,scoring='accuracy',cv=10)
     tree_cv = gscv.fit(X_train,Y_train)
```

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

     tuned hpyerparameters :(best parameters)  {'criterion': 'gini', 'max_depth': 6, 'max_features': 'sqrt', 'min_samples_leaf': 2, 'min_samples_split': 5, 'splitter': 'random'}
     accuracy : 0.9035714285714287
```

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



Predictive Analysis - KNN

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

```
[25] 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()
```

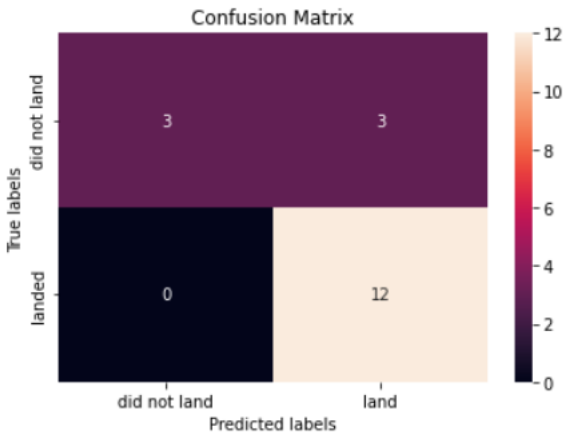
```
[26] gscv = GridSearchCV(KNN,parameters,scoring='accuracy',cv=10)
     knn_cv = gscv.fit(X_train,Y_train)
```

```
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
```

+ Code

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



Predictive Analysis – Best Performing Model

TASK 12

Find the method performs best:

```
algorithms = {'KNN':knn_cv.best_score_,'Tree':tree_cv.best_score_,'LogisticRegression':logreg_cv.best_score_}
bestalgorithm = max(algorithms, key=algorithms.get)
print('Best Algorithm is',bestalgorithm,'with a score of',algorithms[bestalgorithm])
if bestalgorithm == 'Tree':
    print('Best Params is :',tree_cv.best_params_)
if bestalgorithm == 'KNN':
    print('Best Params is :',knn_cv.best_params_)
if bestalgorithm == 'LogisticRegression':
    print('Best Params is :',logreg_cv.best_params_)
```

```
Best Algorithm is Tree with a score of 0.9035714285714287
Best Params is : {'criterion': 'gini', 'max_depth': 6, 'max_features': 'sqrt', 'min_samples_leaf': 2, 'min_samples_split': 5, 'splitter': 'random'}
```

Authors

Joseph Santarcangelo has a PhD in Electrical Engineering, his research focused on using machine learning, signal processing, and computer vision to determine how videos impact human cognition. Joseph has been working for IBM since he completed his PhD.

- Although the four models performed similarly, we do have a winner in this project – the Decision Tree model with a score of 0.90

Conclusion

Project Insights

- If a launch site hosts a higher number of launches, the success rate of landing is higher
- Ever since 2013, the success rate has been increasing steadily
- Orbit types ES-L1, SSO, HEO, and GEO have the highest mean success rate
- In this project, the Decision Tree classifier model performed the best with a score of 0.90