Capstone Project

Skills Network

IBM

OUTLINE

- Executive Summary
- Introduction
- Methodology
- Results
- Conclusion

EXECUTIVE SUMMARY

- SpaceX advertises that Falcon9 rocket with a cost of 62M, which is much more cost saving, compared to other providers with a cost upward 165M each.
- Key point:
  - Analysis of Falcon9 in terms of launch side, payload mass, class, orbit
  - Objective: Determine the cost launch
- Summary Findings:
  - Majority payload mass of Falcon 9 is fallen under 3500, which is medium payload mass
  - The yearly success rate shows a increase from 2013
- Recommendation:
  - Acquire the “Cost” value for each booster version to verify that the cost for running the Falcon9 rocket provided by SpaceX is really lower than other providers.

INTRODUCTION

Space X advertises Falcon 9 rocket launches on its website with a cost of 62 million dollars; other providers cost upward of 165 million dollars each, much of the savings is because Space X can reuse the first stage. Therefore if we can determine if the first stage will land, we can determine the cost of a launch. This information can be used if an alternate company wants to bid against space X for a rocket launch.

METHODOLOGY

- **CRISP DM:**
    - Data Collection: API, IBM dataset, Web Scrapping
    - Data Preparation:
        - Filter: Only filter rows which booster version is Falcon9 only
        - Missing Value (Payload Mass): Replace missing value with mean of payload mass
        - Data Conversion (to numpy): Convert the target column (e.g., Class) into numpy format
        - Data Transformation: Standardize all features data for building machine learning models
    - EDA
    - Data Modelling:
        - Logistic Regression
        - Support Vector Machine (SVM)
        - Decision Tree
        - K-nearest neighborhood (KNN)
        - GridSearch CV with value of 10
    - Model Evaluation
        - Cross Validation
        - Confusion Matrix
        - Accuracy

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

```
spacex_url="https://api.spacexdata.com/v4/launches/past"
```

```
response = requests.get(spacex_url)
```

```
response.status_code
```

```
200
```

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

```
# Use json_normalize meethod to convert the json result into a dataframe
# Decode the response content as JSON
data = response.json()

# Convert the JSON to a Pandas DataFrame
df = pd.json_normalize(data)
```

Using the dataframe `data` print the first 5 rows

```
# Get the head of the dataframe
df.head()
```

Data Collection: Task 2 - Filter the dataframe to only include Falcon 9 launches

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

```
# Hint data['BoosterVersion']!='Falcon 1'
data_falcon9 = df2[df2['BoosterVersion'] != 'Falcon 1']
```

```
data_falcon9.head().reset_index()
```

	index	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins	Reused	Legs	LandingPad	Blocl
0	4	6	2010-06-04	Falcon 9	NaN	LEO	CCSFS SLC 40	None None	1	False	False	False	None	1.(
1	5	8	2012-05-22	Falcon 9	525.0	LEO	CCSFS SLC 40	None None	1	False	False	False	None	1.(
2	6	10	2013-03-01	Falcon 9	677.0	ISS	CCSFS SLC 40	None None	1	False	False	False	None	1.(
3	7	11	2013-09-29	Falcon 9	500.0	PO	VAFB SLC 4E	False Ocean	1	False	False	False	None	1.(
4	8	12	2013-12-03	Falcon 9	3170.0	GTO	CCSFS SLC 40	None None	1	False	False	False	None	1.(

Data Wrangling: Task 3 - Dealing with Missing Values

```
# Replace the np.nan values with its mean value
data_falcon9['PayloadMass'] = data_falcon9['PayloadMass'].replace(np.nan, data_falcon9['PayloadMass'].mean())
```

```
/home/jupyterlab/conda/envs/python/lib/python3.7/site-packages/ipykernel_launcher.py:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy
  This is separate from the ipykernel package so we can avoid doing imports until
```

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

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

```
FlightNumber       0
Date               0
BoosterVersion     0
PayloadMass        0
Orbit              0
LaunchSite         0
Outcome            0
Flights            0
GridFins           0
Reused             0
Legs               0
LandingPad        26
Block              0
ReusedCount        0
Serial             0
Longitude          0
```

Web Scrapping: Task 1 - Request the Falcon9 Launch Wiki page from its URL

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

```
# use requests.get() method with the provided static_url
# assign the response to a object
# Use requests.get() to fetch the URL and assign the response to an object
response = requests.get(static_url)

# Print the response status code to check if the request was successful
print("Status Code:", response.status_code)

# Optionally, print the content of the response
print("Response Content:", response.text)
```

```
nned-clientpref-1 vector-feature-night-mode-enabled skin-theme-clientpref-day vector-toc-available" lang="en"
dir="ltr">
<head>
<meta charset="UTF-8">
<title>List of Falcon 9 and Falcon Heavy launches - Wikipedia</title>
```

Web Scrapping: Task 2 - Extract all column/variable names from the HTML table header

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

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

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

```
<table class="wikitable plainrowheaders collapsible" style="width: 100%;">
<tbody><tr>
<th scope="col">Flight No.
</th>
<th scope="col">Date and<br/>time (<a href="/wiki/Coordinated_Universal_Time" title="Coordinated Universal Tim
e">UTC</a>)
</th>
```

Web Scrapping: Task 3 - Create a data frame by parsing the launch HTML tables

```
    for rows in table.find_all("tr"):
        #check to see if first table heading is as number corresponding to launch a number
        if rows.th:
            if rows.th.string:
                flight_number=rows.th.string.strip()
                flag=flight_number.isdigit()
        else:
            flag=False
        #get table element
        row=rows.find_all('td')
        #if it is number save cells in a dictonary
        if flag:
            extracted_row += 1
            # Flight Number value
            # TODO: Append the flight_number into launch_dict with key `Flight No.`
            #print(flight_number)
            datatimelist=date_time(row[0])

            # Date value
```

Skills Network

IBM

EDA: Task 1 - Calculate the number of launches on each site

The data contains several Space X launch facilities: Cape Canaveral Space Launch Complex 40 **VAFB SLC 4E** , Vandenberg Air Force Base Space Launch Complex 4E **(SLC-4E)**, Kennedy Space Center Launch Complex 39A **KSC LC 39A** .The location of each Launch Is placed in the column `LaunchSite`

Next, let's see the number of launches for each site.

Use the method `value_counts()` on the column `LaunchSite` to determine the number of launches on each site:

```
# Apply value_counts() on column LaunchSite
df['LaunchSite'].value_counts()
```

```
LaunchSite
CCAFS SLC 40    55
KSC LC 39A      22
VAFB SLC 4E     13
Name: count, dtype: int64
```

Each launch aims to an dedicated orbit, and here are some common orbit types:

EDA: Task 2 - Calculate the number of launches on each orbit

Use the method `.value_counts()` to determine the number and occurrence of each orbit in the column `Orbit`

```
# Apply value_counts on Orbit column
df['Orbit'].value_counts()
```

```
Orbit
GTO      27
ISS      21
VLEO     14
PO        9
LEO       7
SSO       5
MEO       3
ES-L1     1
HEO       1
SO        1
GEO       1
Name: count, dtype: int64
```

EDA: Task 3 - Calculate the number and occurence of mission outcome of the orbits¶

```
# Group by 'orbit' and 'outcome', and count occurrences
grouped = df.groupby(['orbit', 'outcome']).size().reset_index(name='count')

# Display the result
print(grouped)
```

EDA: Task 4 - Create a landing outcome label from Outcome column

```
# landing_class = 0 if bad_outcome
# landing_class = 1 otherwise
# Count the unique values in the 'Outcome' column
landing_outcomes = df['Outcome'].value_counts()

# Display the index and outcome for inspection
for i, outcome in enumerate(landing_outcomes.keys()):
    print(i, outcome)

# Create the set of bad outcomes using specific indices
bad_outcomes = set(landing_outcomes.keys()[[1, 3, 5, 6, 7]])

# Create the landing_class list based on the 'Outcome' column
landing_class = [0 if outcome in bad_outcomes else 1 for outcome in df['Outcome']]

# Print the bad_outcomes set and a preview of landing_class
print("Bad outcomes:", bad_outcomes)
print("Landing class:", landing_class[:10])  # Display the first 10 values for preview
```

```
Landing class: [1, 1, 1, 1, 1, 1, 1, 1]
```

SQL: Task 1 - Display the names of the unique launch sites in the space mission

[29]:
```
%sql  SELECT DISTINCT LAUNCH_SITE FROM SPACEXTABLE;
```

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

[29]:

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

SQL: Task 2 - Display 5 records where launch sites begin with the string 'CCA'

```
[45]: %sql  SELECT LAUNCH_SITE FROM SPACEXTABLE WHERE LAUNCH_SITE LIKE 'CCA%' LIMIT 5;

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

[45]:

Launch_Site
CCAFS LC-40
CCAFS LC-40
CCAFS LC-40
CCAFS LC-40
CCAFS LC-40

SQL: Task 3 - Display the total payload mass carried by boosters launched by NASA (CRS)

```
[65]: %sql SELECT BOOSTER_VERSION, SUM(PAYLOAD_MASS__KG_) AS 'TOTAL PAYLOAD MASS(KG)' FROM SPACEXTABLE WHERE CUSTOMER L
```

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

[65]:

Booster_Version	TOTAL PAYLOAD MASS(KG)
F9 B5B1058.1	12530
F9 B5B1061.1	12500
F9 B5B1051.1	12055
F9 B5 B1046.4	12050
F9 B4 B1039.1	3310
F9 FT B1021.1	3136
F9 B5 B1058.4	2972
F9 FT B1035.1	2708
F9 B4 B1045.2	2697
F9 B4 B1039.2	2647

(Partial Result Shown)

SQL: Task 4 - Display average payload mass carried by booster version F9 v1.1

[69]:
```
%sql SELECT BOOSTER_VERSION, AVG(PAYLOAD_MASS__KG_) AS 'AVERAGE PAYLOAD MASS(KG)' FROM SPACEXTABLE WHERE BOOSTER_
```

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

[69]:

Booster_Version	AVERAGE PAYLOAD MASS(KG)
F9 v1.1	2928.4
F9 v1.1 B1016	4707.0
F9 v1.1 B1011	4428.0
F9 v1.1 B1014	4159.0
F9 v1.1 B1012	2395.0
F9 v1.1 B1010	2216.0
F9 v1.1 B1018	1952.0
F9 v1.1 B1015	1898.0
F9 v1.1 B1013	570.0
F9 v1.1 B1017	553.0
F9 v1.1 B1003	500.0

SQL: Task 5 - List the date when the first successful landing outcome in ground pad was achieved.

```
[79]: %sql SELECT MIN(DATE) AS 'FIRST SUCCESSFUL' FROM SPACEXTABLE WHERE LANDING_OUTCOME LIKE 'Success%' ORDER BY DATE;
```

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

[79]:

FIRST SUCCESSFUL
2015-12-22

SQL: 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 BOOSTER_VERSION FROM SPACEXTABLE WHERE PAYLOAD_MASS__KG_ BETWEEN 4000 AND 6000 AND MISSION_OUTCOME = 'Success'; •••
```

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

[82]:

Booster_Version
F9 v1.1
F9 v1.1 B1011
F9 v1.1 B1014
F9 v1.1 B1016
F9 FT B1020
F9 FT B1022

(Partial Result Shown)

SQL: Task 7 - List the total number of successful and failure mission outcomes

```
%sql SELECT MISSION_OUTCOME, COUNT(MISSION_OUTCOME) AS 'TOTAL' FROM SPACEXTABLE WHERE MISSION_OUTCOME like 'F
```

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

Mission_Outcome	TOTAL
Failure (in flight)	1

SQL: Task 8 - List the names of the booster_versions which have carried the maximum payload mass. Use a subquery

```
%sql SELECT BOOSTER_VERSION FROM SPACEXTABLE WHERE PAYLOAD_MASS__KG_ = (SELECT MIN(PAYLOAD_MASS__KG_) FROM SPACEX
```

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

Booster_Version
F9 v1.0 B0003
F9 v1.0 B0004

SQL: 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 SUBSTR(DATE,6,2) AS 'MONTH', LANDING_OUTCOME, MISSION_OUTCOME, BOOSTER_VERSION, LAUNCH_SITE FROM SPAC
```

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

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

SQL: Task 10 - Rank the count of landing outcomes (such as Failure (drone ship) or Success (ground pad)) between the date 2010-06-04 and 2017-03-20, in descending order.

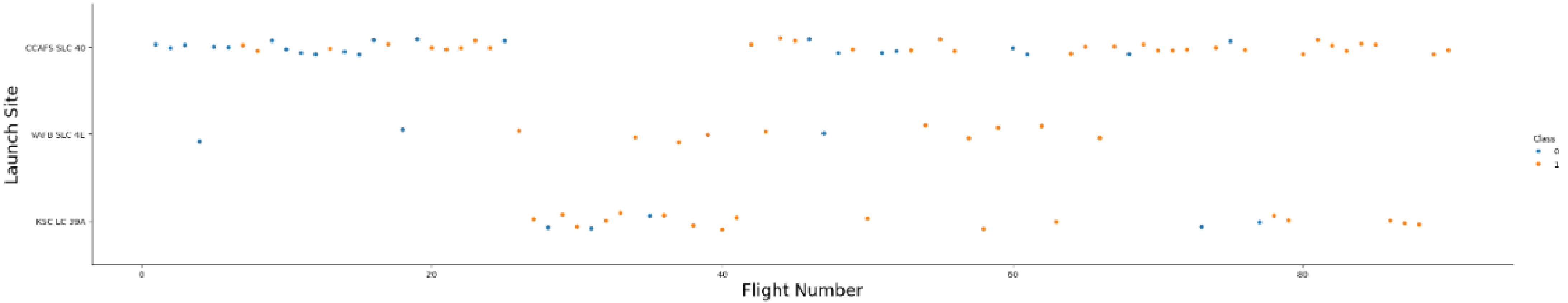
```
%sql SELECT LANDING_OUTCOME, COUNT(LANDING_OUTCOME) FROM SPACEXTABLE WHERE DATE BETWEEN '2010-06-04' AND '2017-03
```

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

Landing_Outcome	COUNT(LANDING_OUTCOME)
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

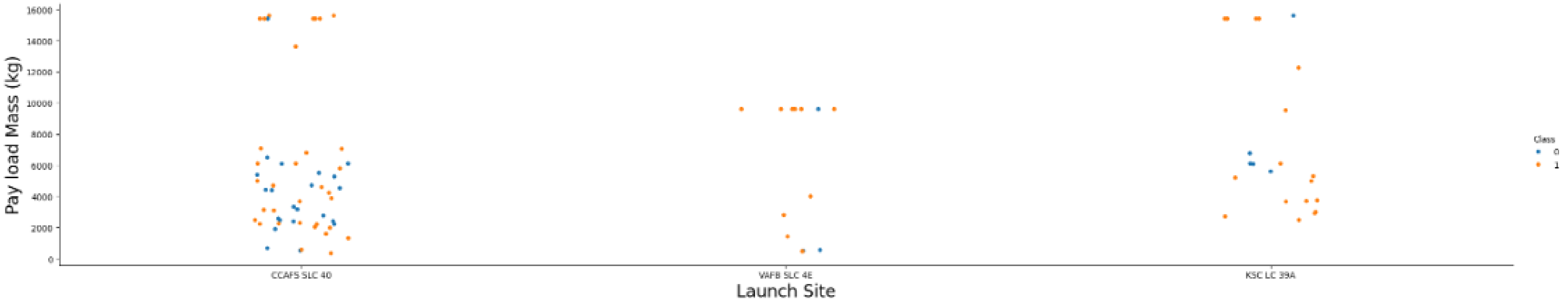
Visualization: Task 1 - Visualize the relationship between Flight Number and Launch Site

```python
# 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
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()
```



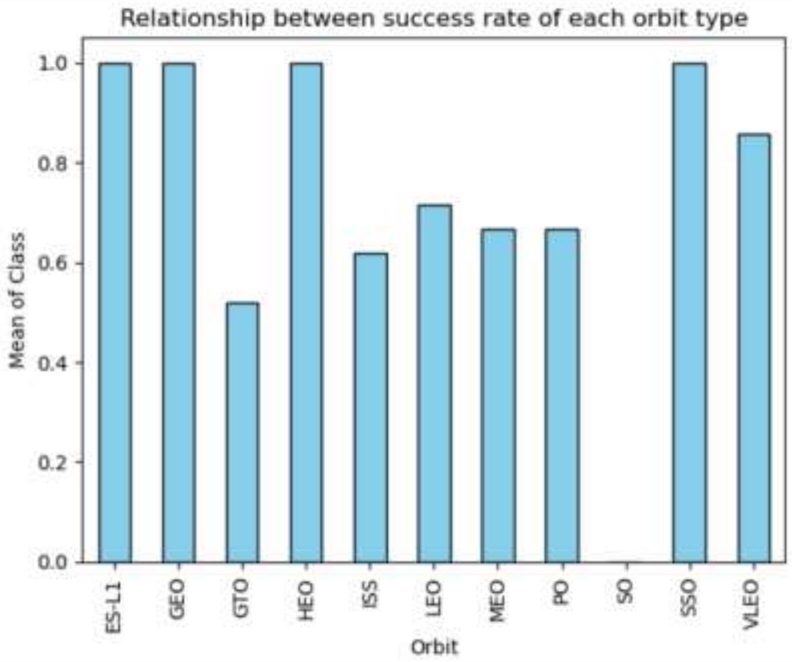
Visualization: Task 2 - Visualize the relationship between PayLoad Mass and Launch Site

```
# 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
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()
```



Visualization: Task 3 - Visualize the relationship between success rate of each orbit type between success rate of each orbit type

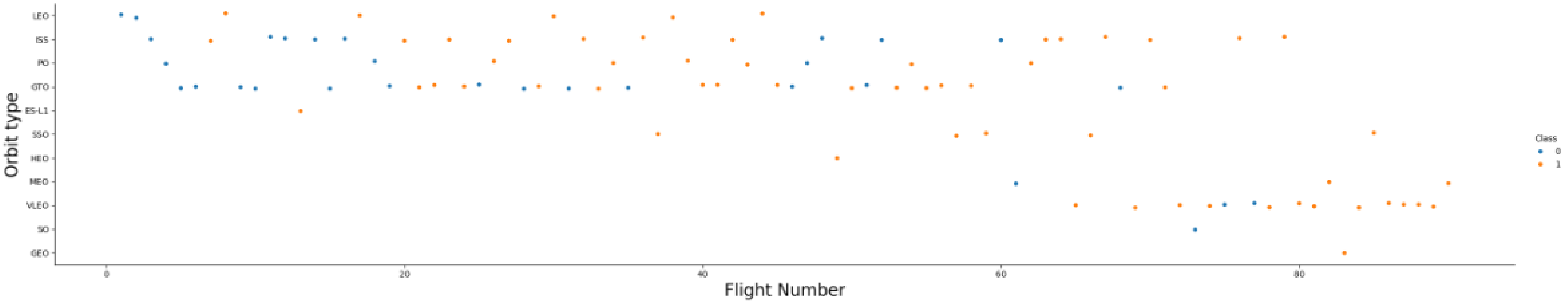
```
grouped_df.plot(kind='bar', color='skyblue', edgecolor='black')

# Set the title and labels for the chart
plt.title('Relationship between success rate of each orbit type')
plt.xlabel('Orbit')
plt.ylabel('Mean of Class')

# Show the plot
plt.show()
```



Visualization: Task 4 - Visualize the relationship between FlightNumber and Orbit type

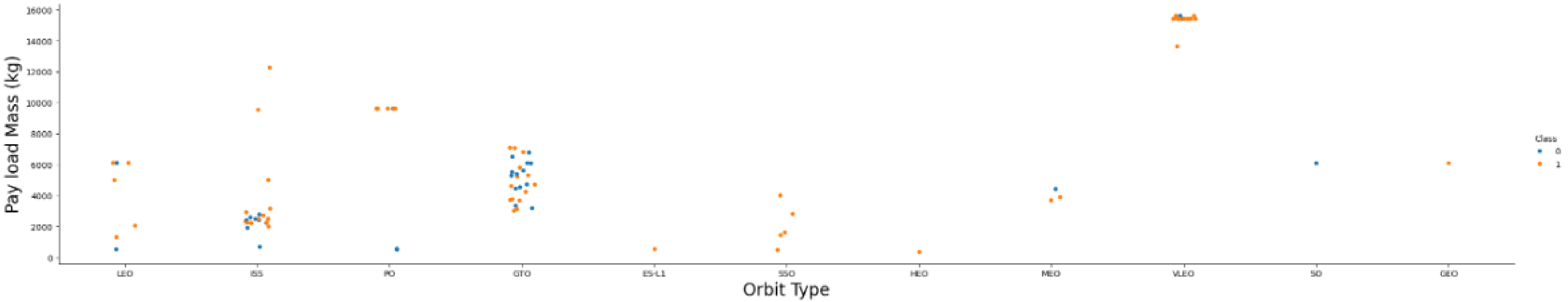
For each orbit, we want to see if there is any relationship between FlightNumber and Orbit type.

```
# Plot a scatter point chart with x axis to be FlightNumber and y axis to be the Orbit, and hue to be the class value
sns.catplot(y="Orbit", x="FlightNumber", hue="Class", data=df, aspect = 5)
plt.xlabel("Flight Number",fontsize=20)
plt.ylabel("Orbit type",fontsize=20)
plt.show()
```



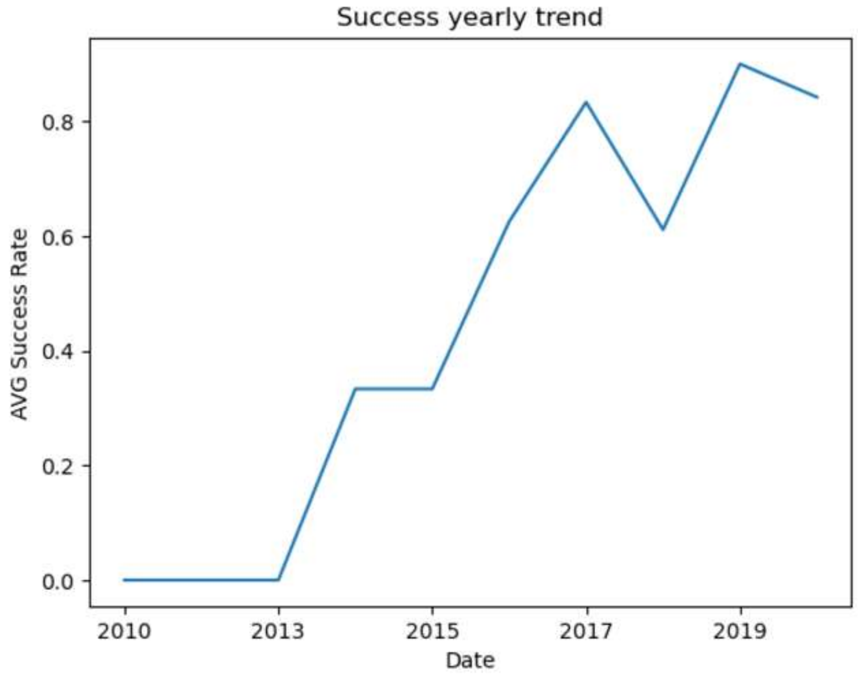
Visualization: Task 5 - Visualize the relationship between Payload Mass and Orbit type

```python
# Plot a scatter point chart with x axis to be Payload Mass and y axis to be the Orbit, and hue to be the class value
sns.catplot(y="PayloadMass", x="Orbit", hue="Class", data=df, aspect = 5)
plt.xlabel("Orbit Type",fontsize=20)
plt.ylabel("Pay load Mass (kg)",fontsize=20)
plt.show()
```



Visualization: Task 6 - Visualize the launch success yearly trend

```
# Plot a line chart with x axis to be the extracted year and y axis to be the success rate
grouped_df2 = df.groupby('Date')['Class'].mean()
grouped_df2.plot(kind= 'line')
plt.xlabel('Date')
plt.ylabel('AVG Success Rate')
plt.title('Success yearly trend')
plt.show()
```



Map: 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
for idx, row in launch_sites_df.iterrows():
    coordinate = (row['Lat'], row['Long'])
    site_name = row['Launch Site']

    # Add a circle at each launch site with a radius of 1000 meters
    folium.Circle(coordinate, radius=1000, color='#000000', fill=True, fill_color='#000000').add_child(
        folium.Popup(f'Launch Site: {site_name}')
    ).add_to(site_map)

    # Add a marker with a label at each launch site
    folium.Marker(coordinate, icon=DivIcon(icon_size=(20, 20), icon_anchor=(0, 0),
                                           html=f'<div style="font-size: 12; color:#d35400;"><b>{site_name}</b></div>')).add_t
```

```
site_map
```

Map: Task 2 - Mark the success/failed launches for each site on the map

```
# 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
spacex_df['marker_color'] = spacex_df['class'].apply(lambda x: 'green' if x == 1 else 'red')
```

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

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

# for 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,
# e.g., icon=folium.Icon(color='white', icon_color=row['marker_color']
for index, record in spacex_df.iterrows():
    # Get the coordinates and site name
    coordinate = (record['Lat'], record['Long'])
    site_name = record['Launch Site']

    # Get the marker color for the current row (based on the 'Class' column)
    marker_color2 = record['marker_color']

    # Create a Marker with a customized icon (color based on success or failure)
    marker = folium.Marker(
        coordinate,
        icon=folium.Icon(color=marker_color2, icon_color='white', icon='info-sign')
    )

    # Add a Popup with additional information (e.g., site name, class)
    marker.add_child(folium.Popup(f"Launch Site: {site_name} - {'Success' if record['class'] == 1 else 'Failure'}"))
```

Map: Task 3 - Calculate the distances between a launch site to its proximities

```python
from geopy.distance import geodesic

# Function to calculate distance using geodesic from geopy
def calculate_distance(coord1, coord2):
    return geodesic(coord1, coord2).km  # distance in kilometers

# Create a dictionary to store distances
distances = {}

# Loop through the DataFrame to calculate distances
for i, site1 in spacex_df.iterrows():
    distances[site1['Launch Site']] = []
    for j, site2 in spacex_df.iterrows():
        if i != j:  # Don't calculate the distance to itself
            coord1 = (site1['Lat'], site1['Long'])
            coord2 = (site2['Lat'], site2['Long'])
            dist = calculate_distance(coord1, coord2)
            distances[site1['Launch Site']].append((site2['Launch Site'], dist))

# Print out the distances for each site
for site, dists in distances.items():
    print(f"Distances from {site}:")
    for target_site, dist in dists:
        print(f"  - {target_site}: {dist:.2f} km")
```

```
Distances from CCAFS LC-40:
  - CCAFS LC-40: 0.00 km
  - CCAFS LC-40: 0.00 km
  - CCAFS LC-40: 0.00 km
  - CCAFS LC-40: 0.00 km
  - CCAFS LC-40: 0.00 km
  - CCAFS LC-40: 0.00 km
```

ML: Task 1 – Target format conversion to numpy format

```python
# Assuming 'df' is your DataFrame and it has a 'Class' column
Y = data['Class'].to_numpy()

# Output to verify
print(Y)
```

```
[0 0 0 0 0 0 1 1 0 0 0 0 1 0 0 0 1 0 0 1 1 1 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1
 1 1 1 1 1 1 1 1 0 0 0 1 1 0 0 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 0 1 1 1 1 0 1
 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1]
```

ML: Task 2 – Transform data

```python
# Assuming 'X' is your input data (a pandas DataFrame or NumPy array)
transform = preprocessing.StandardScaler()

# Fit and transform the data
X = transform.fit_transform(X)

# Output to verify
print(X)
```

```
[[-1.71291154e+00 -1.94814463e-16 -6.53912840e-01 ... -8.35531692e-01
   1.93309133e+00 -1.93309133e+00]
 [-1.67441914e+00 -1.19523159e+00 -6.53912840e-01 ... -8.35531692e-01
   1.93309133e+00 -1.93309133e+00]
 [-1.63592675e+00 -1.16267307e+00 -6.53912840e-01 ... -8.35531692e-01
   1.93309133e+00 -1.93309133e+00]
 ...
 [ 1.63592675e+00  1.99100483e+00  3.49060516e+00 ...  1.19684269e+00
  -5.17306132e-01  5.17306132e-01]
 [ 1.67441914e+00  1.99100483e+00  1.00389436e+00 ...  1.19684269e+00
  -5.17306132e-01  5.17306132e-01]
 [ 1.71291154e+00 -5.19213966e-01 -6.53912840e-01 ... -8.35531692e-01
  -5.17306132e-01  5.17306132e-01]]
```

ML: Task 3 – Cross Validation (Train-Test split)

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

`X_train, X_test, Y_train, Y_test`

```
# Step 1: Split the data into training and testing sets (80% training, 20% testing)
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=42)
```

we can see we only have 18 test samples.

```
Y_test.shape
```

```
(18,)
```

ML: Task 4 – Logistic Regression

```
# Step 1: Create the Logistic Regression object
lr=LogisticRegression()

# Step 2: Define the hyperparameters to tune in a dictionary
parameters ={'C':[0.01,0.1,1],
             'penalty':['l2'],
             'solver':['lbfgs']}

# Step 3: Create the GridSearchCV object with 10-fold cross-validation
lr_cv = GridSearchCV(lr, param_grid=parameters, cv=10, scoring='accuracy')

# Step 4: Fit the GridSearchCV to find the best parameters
lr_cv.fit(X_train, Y_train)
```

```
# Step 5: Display the best parameters found
print("tuned hpyerparameters :(best parameters) ",logreg_cv.best_params_)
```

```
tuned hpyerparameters :(best parameters)  {'C': 0.1, 'penalty': 'l1', 'solver': 'liblinear'}
```

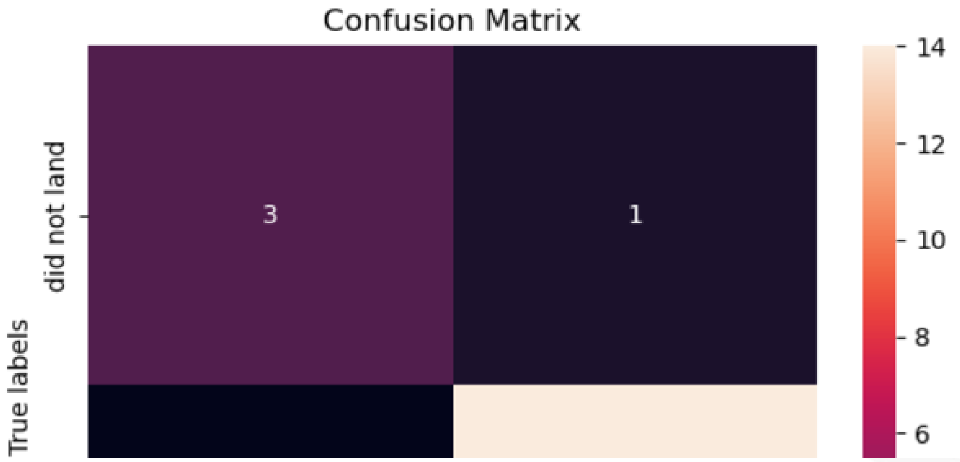
ML: Task 5 – Logistic Regression – Accuracy & Confusion Matrix

```
best_lr_model = lr_cv.best_estimator_

# Evaluate the best model on the test data
test_accuracy = best_lr_model.score(X_test, y_test)
print("accuracy :",test_accuracy)
```

```
accuracy : 0.8333333333333334
```

Lets look at the confusion matrix:

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



ML: Task 6 – SVM

```python
# Step 1: Create the Support Vector Machine (SVM) object
svm = SVC()

# Step 2: Define the hyperparameters to tune in a dictionary
parameters = {
    'kernel':('linear', 'rbf','poly','rbf', 'sigmoid'),  # Kernel type
    'C': np.logspace(-3, 3, 5),  # Regularization strength
    'gamma':np.logspace(-3, 3, 5)  # Kernel coefficient
    #'degree': [3, 4, 5],  # Degree of the polynomial kernel function
}

# Step 3: Create the GridSearchCV object with 10-fold cross-validation
svm_cv = GridSearchCV(svm, param_grid=parameters, cv=10, scoring='accuracy')

# Step 4: Fit the GridSearchCV to find the best parameters
svm_cv.fit(X_train, Y_train)

# Step 5: Display the best parameters found
print("tuned hpyerparameters :(best parameters) ",svm_cv.best_params_)
```

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

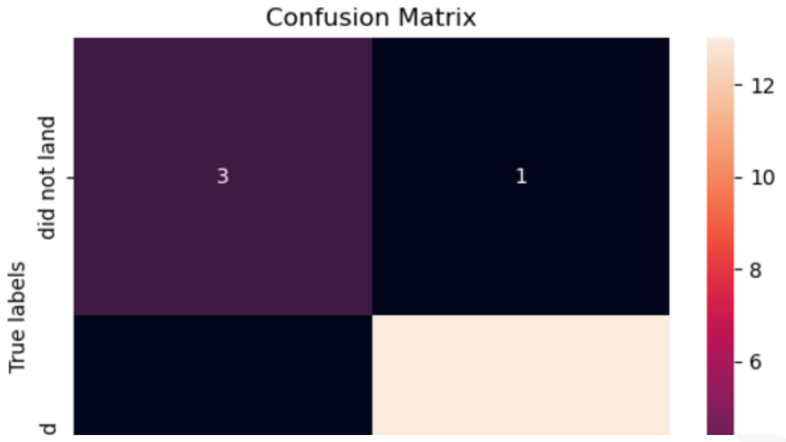
ML: Task 7 – SVM – Accuracy & Confusion Matrix

```
best_svm_model = svm_cv.best_estimator_

# Evaluate the best model on the test data
test_accuracy = best_svm_model.score(X_test, y_test)
print("accuracy :",test_accuracy)
```

```
accuracy : 0.8888888888888888
```

We can plot the confusion matrix

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



ML: Task 8 – Decision Tree – Accuracy & Confusion Matrix

```
# Step 1: Create the Decision Tree Classifier object
tree = DecisionTreeClassifier()

# Step 2: Define the hyperparameters to tune in a dictionary
parameters = {'criterion': ['gini', 'entropy'], # Function to measure the quality of a split
      'splitter': ['best', 'random'], # Strategy used to split at each node
      'max_depth': [2*n for n in range(1,10)], # Maximum depth of the tree
      'max_features': ['auto', 'sqrt'], # Number of features to consider for the best split
      'min_samples_leaf': [1, 2, 4], # Minimum samples required to be at a leaf node
      'min_samples_split': [2, 5, 10] # Minimum samples required to split an internal node
}

# Step 3: Create the GridSearchCV object with 10-fold cross-validation
tree_cv = GridSearchCV(tree, param_grid=parameters, cv=10, scoring='accuracy')

# Step 4: Fit the GridSearchCV to find the best parameters
tree_cv.fit(X_train, Y_train)

# Step 5: Display the best parameters found
print("tuned hpyerparameters :(best parameters) ",tree_cv.best_params_)
```

```
tuned hpyerparameters :(best parameters)  {'criterion': 'gini', 'max_depth': 6, 'max_features': 'sqrt', 'min_samples_leaf':
```

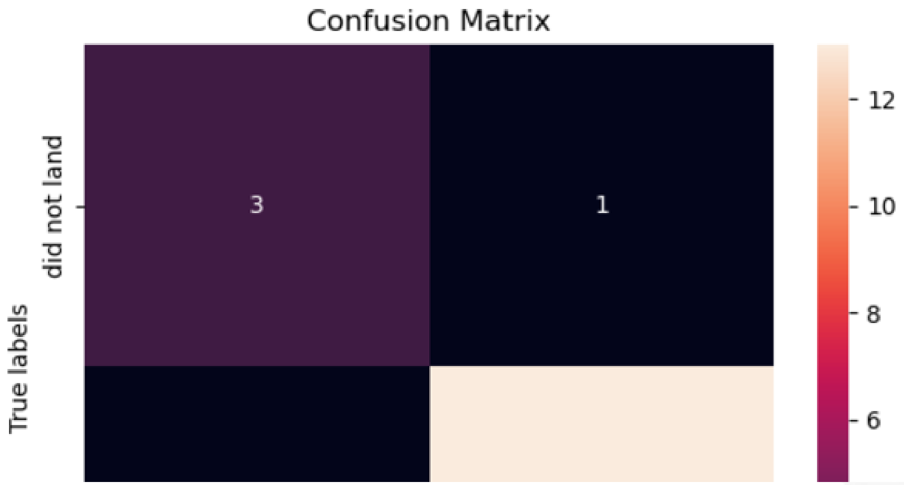
ML: Task 9 – Decision Tree – Accuracy & Confusion Matrix

```
best_tree_model = tree_cv.best_estimator_

# Evaluate the best model on the test data
test_accuracy = best_tree_model.score(X_test, y_test)
print("accuracy :",test_accuracy)
```

```
accuracy : 0.8888888888888888
```

We can plot the confusion matrix

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



ML: Task 10 – KNN

```
# Step 1: Create the K-Nearest Neighbors (KNN) classifier object
KNN = KNeighborsClassifier()

# Step 2: Define the hyperparameters to tune in a dictionary
parameters = {
    'n_neighbors': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],  # Number of neighbors
    'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute'],  # Algorithm for finding nearest neighbors
    'p': [1, 2],  # Power parameter for the Minkowski distance (1 = Manhattan, 2 = Euclidean)
}

# Step 3: Create the GridSearchCV object with 10-fold cross-validation
knn_cv = GridSearchCV(KNN, param_grid=parameters, cv=10, scoring='accuracy')

# Step 4: Fit the GridSearchCV to find the best parameters
knn_cv.fit(X_train, Y_train)

# Step 5: Display the best parameters found
print("tuned hpyerparameters :(best parameters) ",knn_cv.best_params_)
```

```
tuned hpyerparameters :(best parameters)  {'algorithm': 'auto', 'n_neighbors': 6, 'p': 1}
```

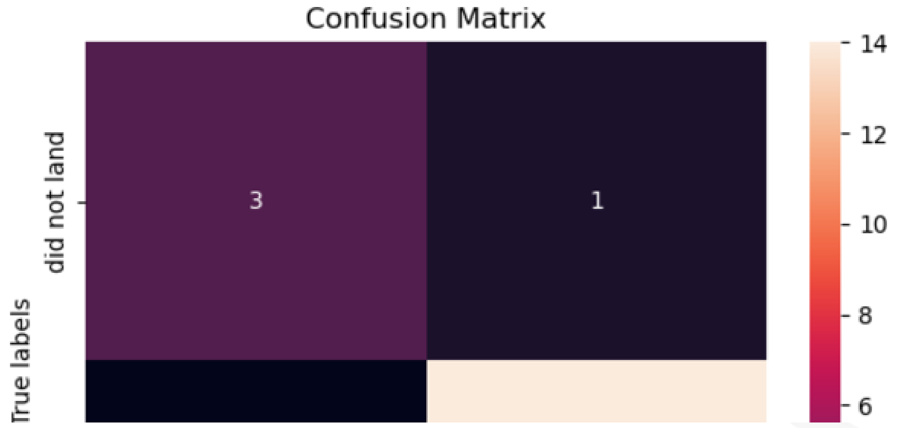
ML: Task 11 – KNN – Accuracy & Confusion Matrix

```
best_knn_model = knn_cv.best_estimator_

# Evaluate the best model on the test data
test_accuracy = best_knn_model.score(X_test, y_test)
print("accuracy :",test_accuracy)
```

```
accuracy : 0.9444444444444444
```

We can plot the confusion matrix

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



ML: Task 12 – Find the best model

```python
    # Evaluate the best model on the test data
    best_model = grid_search.best_estimator_
    test_accuracy = best_model.score(X_test, y_test)

    # Append test results (test accuracy) to test_results
    test_results.append({
        'Model': model_name,
        'Test Accuracy': test_accuracy
    })

# Create DataFrames for both results
parameter_results_df = pd.DataFrame(parameter_results)
best_params_df = pd.DataFrame(best_params_results)
test_results_df = pd.DataFrame(test_results)

# Display the results
print("Parameter Results DataFrame:")
print(parameter_results_df)

print("\nBest Parameters DataFrame:")
print(best_params_df)

print("\nTest Results DataFrame:")
print(test_results_df)
```

```
Best Parameters DataFrame:
                 Model                                      Best Parameters  \
0  Logistic Regression                                         {'C': 0.1}
1                  SVM          {'C': 1, 'gamma': 'scale', 'kernel': 'rbf'}
2        Decision Tree  {'criterion': 'entropy', 'max_depth': None, 'm...
3                  KNN    {'algorithm': 'auto', 'n_neighbors': 6, 'p': 1}

   Best Accuracy (CV)
0            0.803571
1            0.816071
2            0.762500
3            0.844643

Test Results DataFrame:
                 Model  Test Accuracy
0  Logistic Regression       0.944444
1                  SVM       0.888889
2        Decision Tree       0.888889
3                  KNN       0.944444
```

Best: Either Logistic Regression or KNN

CONCLUSION

Falcon9 rocket provided by SpaceX has the high success rate. The line chart has indicated that the success rate increased from 2013.