Applied Data-Science Capstone Project

Treasure Okafor

24th October,2022

IBM Developer

SKILLS NETWORK

OUTLINE

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

EXECUTIVE SUMMARY

- Data Science journey:

  Data collection, wrangling, exploratory data analysis with sequel and python, interactive visual analytics and predictive analytics.

- Summary of all results.

INTRODUCTION

A Data Science project is a project that provides the basis for performing subsequent analysis and processing of data.

In this report, I'll help you understand my Data science journey so far with the little skills I covered.

Let get started!

METHODOLOGY

- Executive Summary
  - Data collection
  - Data wrangling
  - Exploratory data analysis
  - Data visualization
  - Model Development
  - Reporting

Data Collection

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.

Data Collection API

Now let's start requesting rocket launch data from SpaceX API with the following URL:

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

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

Check the content of the response

```
In [8]: print(response.content)
```


```
b'[{"fairings":{"reused":false,"recovery_attempt":false,"recovered":false,"ships":[]},"links":{"patch":{"sma
h45r_o.png","large":"https://images2.imgbox.com/5b/02/QcxHUb5V_o.png"},"reddit":{"campaign":null,"launch":nu
{"small":[],"original":[]},"presskit":null,"webcast":"https://www.youtube.com/watch?v=0a_00nJ_Y88","youtube_
pace.com/2196-spacex-inaugural-falcon-1-rocket-lost-launch.html","wikipedia":"https://en.wikipedia.org/wiki/
17T00:00:00.000Z","static_fire_date_unix":1142553600,"net":false,"window":0,"rocket":"5e9d0d95eda69955f709d1
3,"altitude":null,"reason":"merlin engine failure"}],"details":"Engine failure at 33 seconds and loss of veh
[],"payloads":["5eb0e4b5b6c3bb0006eeb1e1"],"launchpad":"5e9e4502f5090995de566f86","flight_number":1,"name":"
0.000Z","date_unix":1143239400,"date_local":"2006-03-25T10:30:00+12:00","date_precision":"hour","upcoming":f
b2623","flight":1,"gridfins":false,"legs":false,"reused":false,"landing_attempt":false,"landing_success":nul
to_update":true,"tbd":false,"launch_library_id":null,"id":"5eb87cd9ffd86e000604b32a"},{"fairings":{"reused":
d":false,"ships":[]},"links":{"patch":{"small":"https://images2.imgbox.com/f9/4a/ZboXReNb_o.png","large":"ht
png"},"reddit":{"campaign":null,"launch":null,"media":null,"recovery":null},"flickr":{"small":[],"original":
```


Data Collection with webscraping with beautiful soup

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

Next, request the HTML page from the above URL and get a `response` object

[redacted]

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

```
In [7]:
# use requests.get() method with the provided static_url
response = requests.get(static_url)
# assign the response to a object
```

Create a `BeautifulSoup` object from the HTML `response`

```
In [12]:
# Use BeautifulSoup() to create a BeautifulSoup object from a response text content
soup = BeautifulSoup(response.text, "html.parser")
```

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

```
In [10]:
# Use soup.title attribute
print(html_file.prettify())
```

```
<!DOCTYPE html>
<html class="client-nojs" dir="ltr" lang="en">
 <head>
  <meta charset="utf-8"/>
  <title>
   List of Falcon 9 and Falcon Heavy launches - Wikipedia
  </title>
  <script>
   document.documentElement.className="client-js";RLCONF={"wgBreakFrames":false,"wgSeparatorTransformTable":["",""],"wgDigitTransform
gDefaultDateFormat":"dmy","wgMonthNames":["","January","February","March","April","May","June","July","August","September","October",
ber"],"wgRequestId":"2f2cb346-bf0a-4a7e-b44d-6a31ede8feb4","wgCSPNonce":false,"wgCanonicalNamespace":"","wgCanonicalSpecialPageName":
```

Data wrangling

- Calculate the number of launches on each site.
- Calculate the number and occurrence of each orbit.
- https://github.com/Teekafy/Treasure_proj/blob/97cce462a14e1bd6bc35d7f05705b941e0acf056/Data%20Wrangling.ipynb

Load Space X dataset, from last section.

In [2]:
```
df=pd.read_csv("https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/datasets/dataset_part_1.csv")
df.head(10)
```

Out[2]:

	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins	Reused	Legs	LandingPad	Block	ReusedCount	Serial	Longitude	Latit
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	28.561
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	28.561
2	3	2013-03-01	Falcon 9	677.000000	ISS	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B0007	-80.577366	28.561
3	4	2013-09-29	Falcon 9	500.000000	PO	VAFB SLC 4E	False Ocean	1	False	False	False	NaN	1.0	0	B1003	-120.610829	34.632
4	5	2013-12-03	Falcon 9	3170.000000	GTO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B1004	-80.577366	28.561
5	6	2014-01-06	Falcon 9	3325.000000	GTO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B1005	-80.577366	28.561
6	7	2014-04-18	Falcon 9	2296.000000	ISS	CCAFS SLC 40	True Ocean	1	False	False	True	NaN	1.0	0	B1006	-80.577366	28.561
7	8	2014-07-14	Falcon 9	1316.000000	LEO	CCAFS SLC 40	True Ocean	1	False	False	True	NaN	1.0	0	B1007	-80.577366	28.561
8	9	2014-08-05	Falcon 9	4535.000000	GTO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B1008	-80.577366	28.561
9	10	2014-09-07	Falcon 9	4428.000000	GTO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B1011	-80.577366	28.561

Identify and calculate the percentage of the missing values in each attribute

In [3]:
```
df.isnull().sum()/df.count()*100
```

Out[3]:
```
FlightNumber      0.000
Date              0.000
BoosterVersion    0.000
PayloadMass       0.000
Orbit             0.000
LaunchSite        0.000
Outcome           0.000
```

[redacted] 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:

In [8]:
```
# Apply value_counts() on column LaunchSite
df.LaunchSite.value_counts()
```

Out[8]:
```
CCAFS SLC 40    55
KSC LC 39A      22
VAFB SLC 4E     13
Name: LaunchSite, dtype: int64
```

In [9]:
```
df.value_counts()
```

Out[9]:
```
FlightNumber  Date        BoosterVersion  PayloadMass  Orbit  LaunchSite    Outcome     Flights  GridFins  Reused  Legs  LandingPad                Blo
ck  ReusedCount  Serial  Longitude    Latitude
12            2015-01-10  Falcon 9        2395.000000  ISS    CCAFS SLC 40  False ASDS  1        True      False   True  5e9e3032383ecb761634e7cb  1.0
0             B1012   -80.577366   28.561857    1
14            2015-04-14  Falcon 9        1898.000000  ISS    CCAFS SLC 40  False ASDS  1        True      False   True  5e9e3032383ecb761634e7cb  1.0
0             B1015   -80.577366   28.561857    1
58            2018-11-15  Falcon 9        3000.000000  GTO    KSC LC 39A    True ASDS   2        True      True    True  5e9e3032383ecb6bb234e7ca  5.0
2             B1047   -80.603956   28.608058    1
59            2018-12-03  Falcon 9        4000.000000  SSO    VAFB SLC 4E   True ASDS   3        True      True    True  5e9e3033383ecbb9e534e7cc  5.0
3             B1046   -120.610829  34.632093    1
60            2018-12-05  Falcon 9        2573.000000  ISS    CCAFS SLC 40  False RTLS  1        True      False   True  5e9e3032383ecb267a34e7c7  5.0
0             B1050   -80.577366   28.561857    1
```

EDA and interactive visual analytics

https://github.com/Teekafy/Treasure_proj/blob/c1fba45b132d661b31ac248b66a5355003838975/eda%20visualization.ipynb

Exploratory Data Analysis

First, let's read the SpaceX dataset into a Pandas dataframe and print its summary

[27]:
```
df=pd.read_csv("https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/datasets/dataset_part_2.csv")

df
```

[27]:

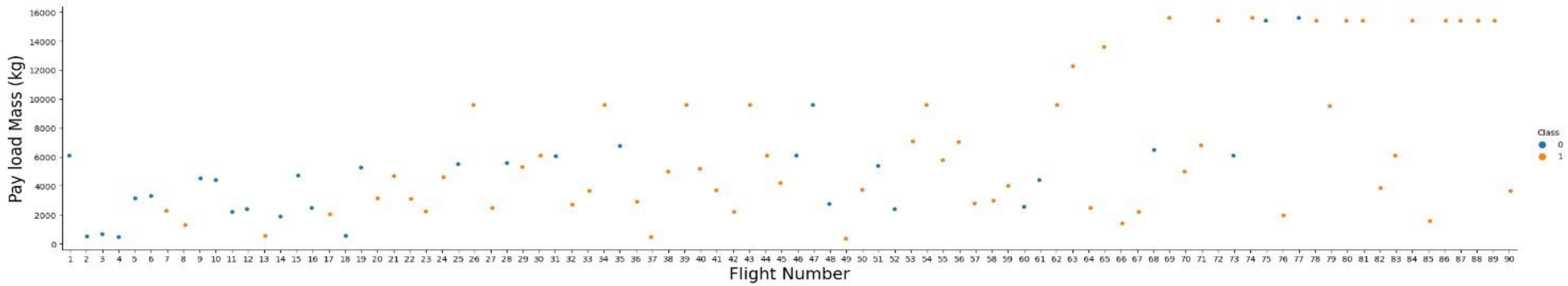
	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins	Reused	Legs	LandingPad	Block	ReusedCount	Serial	Longitude	Latitude	Class
0	1	2010-06-04	Falcon 9	6104.959412	LEO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B0003	-80.577366	28.561857	0
1	2	2012-05-22	Falcon 9	525.000000	LEO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B0005	-80.577366	28.561857	0
2	3	2013-03-01	Falcon 9	677.000000	ISS	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B0007	-80.577366	28.561857	0
3	4	2013-09-29	Falcon 9	500.000000	PO	VAFB SLC 4E	False Ocean	1	False	False	False	NaN	1.0	0	B1003	-120.610829	34.632093	0
4	5	2013-12-03	Falcon 9	3170.000000	GTO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B1004	-80.577366	28.561857	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
85	86	2020-09-03	Falcon 9	15400.000000	VLEO	KSC LC 39A	True ASDS	2	True	True	True	5e9e3032383ecb6bb234e7ca	5.0	2	B1060	-80.603956	28.608058	1
86	87	2020-10-06	Falcon 9	15400.000000	VLEO	KSC LC 39A	True ASDS	3	True	True	True	5e9e3032383ecb6bb234e7ca	5.0	2	B1058	-80.603956	28.608058	1
87	88	2020-10-18	Falcon 9	15400.000000	VLEO	KSC LC 39A	True ASDS	6	True	True	True	5e9e3032383ecb6bb234e7ca	5.0	5	B1051	-80.603956	28.608058	1
88	89	2020-10-24	Falcon 9	15400.000000	VLEO	CCAFS SLC 40	True ASDS	3	True	True	True	5e9e3033383ecbb9e534e7cc	5.0	2	B1060	-80.577366	28.561857	1
89	90	2020-11-05	Falcon 9	3681.000000	MEO	CCAFS SLC 40	True ASDS	1	True	False	True	5e9e3032383ecb6bb234e7ca	5.0	0	B1062	-80.577366	28.561857	1

90 rows × 18 columns

First, let's try to see how the `FlightNumber` (indicating the continuous launch attempts.) and `Payload` variables would affect the launch outcome.

We can plot out the `FlightNumber` vs. `PayloadMass` and overlay the outcome of the launch. We see that as the flight number increases, the first stage is more likely to land successfully. The payload mass is also important; it seems the more massive the payload, the less likely the first stage will return.

[3]:
```
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()
```



Predictive analysis

Features Engineering

By now, you should obtain some preliminary insights about how each important variable would affect the success rate, we will select the features that will be used in success prediction in the future module.

[17]:
```
features = df[['FlightNumber', 'PayloadMass', 'Orbit', 'LaunchSite', 'Flights', 'GridFins', 'Reused', 'Legs', 'LandingPad', 'Block', 'ReusedCount', 'Serial']]
features.head()
```

[17]:

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

[redacted] 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.

[26]:
```
# HINT: Use get_dummies() function on the categorical columns
features_one_hot = pd.get_dummies(df, columns=['Orbit', 'LaunchSite', 'LandingPad', 'Serial'])
features_one_hot
```

[26]:

	FlightNumber	Date	BoosterVersion	PayloadMass	Outcome	Flights	GridFins	Reused	Legs	Block	...	Serial_B1048	Serial_B1049	Serial_B1050	Serial_B1051	Serial_B1054	Serial_B1056	Serial_B1058	Serial_B1059	Serial_B1060	Serial_B1062
0	1	2010-06-04	Falcon 9	6104.959412	None None	1	False	False	False	1.0	...	0	0	0	0	0	0	0	0	0	0
1	2	2012-05-22	Falcon 9	525.000000	None None	1	False	False	False	1.0	...	0	0	0	0	0	0	0	0	0	0
2	3	2013-03-01	Falcon 9	677.000000	None None	1	False	False	False	1.0	...	0	0	0	0	0	0	0	0	0	0
3	4	2013-09-29	Falcon 9	500.000000	False Ocean	1	False	False	False	1.0	...	0	0	0	0	0	0	0	0	0	0
4	5	2013-12-03	Falcon 9	3170.000000	None None	1	False	False	False	1.0	...	0	0	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
85	86	2020-09-03	Falcon 9	15400.000000	True ASDS	2	True	True	True	5.0	...	0	0	0	0	0	0	0	0	1	0
86	87	2020-10-06	Falcon 9	15400.000000	True ASDS	3	True	True	True	5.0	...	0	0	0	0	0	0	1	0	0	0
87	88	2020-10-18	Falcon 9	15400.000000	True ASDS	6	True	True	True	5.0	...	0	0	0	1	0	0	0	0	0	0
88	89	2020-10-24	Falcon 9	15400.000000	True ASDS	3	True	True	True	5.0	...	0	0	0	0	0	0	0	0	1	0
89	90	2020-11-05	Falcon 9	3681.000000	True ASDS	1	True	False	True	5.0	...	0	0	0	0	0	0	0	0	0	1

90 rows × 86 columns

[redacted] Cast all numeric columns to `float64`

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

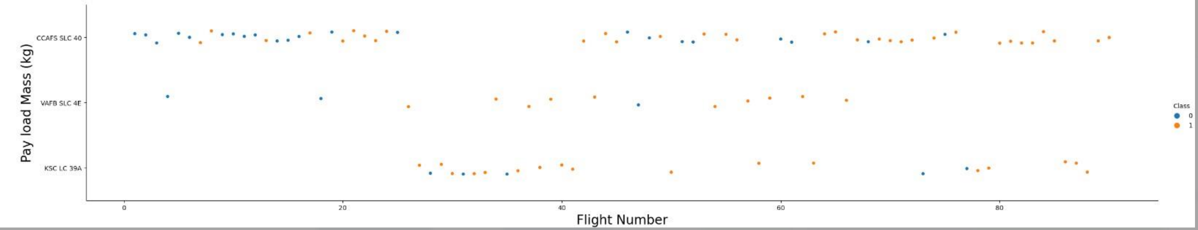
[24]:
```
features_one_hot = features_one_hot.astype('float64')
features_one_hot.dtypes
```

EDA with Visualization

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'`

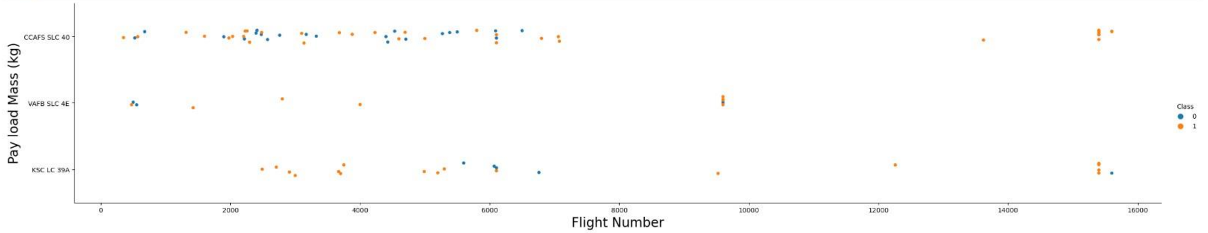
```
[4]: # Plot a scatter point chart with x axis to be Flight Number and y axis to be the launch site, and hue to be the class value
sns.catplot(y="LaunchSite", x="FlightNumber", hue="Class", data=df, aspect = 5)
plt.xlabel("Flight Number",fontsize=20)
plt.ylabel("Pay load Mass (kg)",fontsize=20)
plt.show()
```



[redacted] 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.

```
[5]: # 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 value
# 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="PayloadMass", hue="Class", data=df, aspect = 5)
plt.xlabel("Flight Number",fontsize=20)
plt.ylabel("Pay load Mass (kg)",fontsize=20)
plt.show()
```

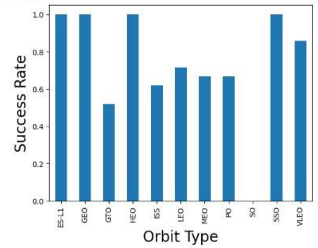


[redacted]: Visualize the relationship between success rate of each orbit type

Next, we want to visually check if there are any relationship between success rate and orbit type.

Let's create a `bar chart` for the sucess rate of each orbit

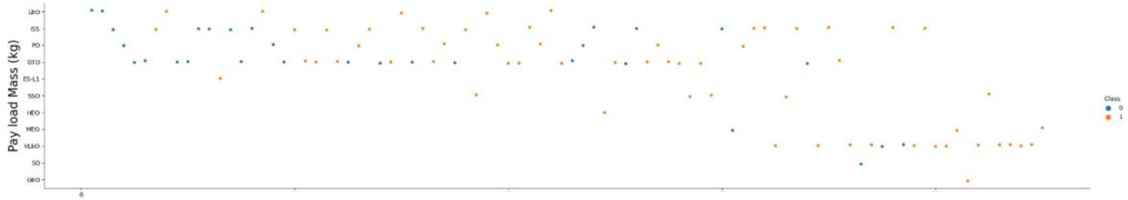
```
[9]: # HINT use groupby method on Orbit column and get the mean of Class column
df.groupby("Orbit").mean()['Class'].plot(kind='bar')
plt.xlabel("Orbit Type",fontsize=20)
plt.ylabel("Success Rate",fontsize=20)
plt.show()
```



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



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

```
[7]: # 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
# 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="Orbit", x="PayloadMass", hue="Class", data=df, aspect = 5)
plt.xlabel("Flight Number",fontsize=20)
plt.ylabel("Pay load Mass (kg)",fontsize=20)
plt.show()
```

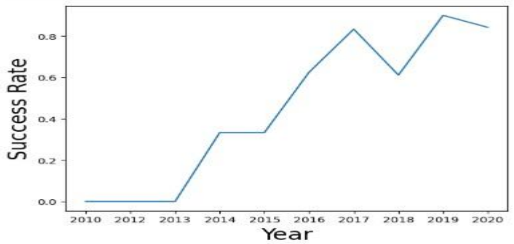
TASK 6: Visualize the launch success yearly trend

You can plot a line chart with x axis to be `Year` and y axis to be average success rate, to get the average launch success trend.

The function will help you get the year from the date:

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

```
[16]: # Plot a line chart with x axis to be the extracted year and y axis to be the success rate
df1 = pd.DataFrame(Extract_year(df['Date']), columns = ['year'])
df1['Class'] = df['Class']
sns.lineplot(x=np.unique(Extract_year(df['Date'])), y=df1.groupby('year')['Class'].mean(), data=df1)
plt.xlabel("Year", fontsize=20)
plt.ylabel("Success Rate", fontsize=20)
plt.show()
```



IBM Developer

EDA with Sequel (SQL)

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

In [29]:
```
%sql select distinct "Launch_Site" from SPACEXTBL1
```
```
 * sqlite:///my_data1.db
Done.
```
Out[29]:

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

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

In [37]:
```
%sql select AVG("PAYLOAD_MASS__KG_"), "BOOSTER_VERSION" \
    from SPACEXTBL1 \
    where "BOOSTER_VERSION" = 'F9 v1.1' \
    group by "BOOSTER_VERSION"
```
```
 * sqlite:///my_data1.db
Done.
```
Out[37]:

AVG("PAYLOAD_MASS__KG_")	Booster_Version
2928.4	F9 v1.1

List the total number of successful and failure mission outcomes

In [54]:
```
%sql select COUNT(MISSION_OUTCOME) \
    from SPACEXTBL1 \
    where MISSION_OUTCOME in ('Success', 'Failure')
```
```
 * sqlite:///my_data1.db
Done.
```
Out[54]:

COUNT(MISSION_OUTCOME)
98

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

In [61]:
```
%sql select "PAYLOAD_MASS__KG_", BOOSTER_VERSION from SPACEXTBL1 \
    where "PAYLOAD_MASS__KG_" = (select max(PAYLOAD_MASS__KG_) from SPACEXTBL1)
```
```
 * sqlite:///my_data1.db
Done.
```
Out[61]:

PAYLOAD_MASS__KG_	Booster_Version
15600	F9 B5 B1048.4
15600	F9 B5 B1049.4
15600	F9 B5 B1051.3
15600	F9 B5 B1056.4
15600	F9 B5 B1048.5
15600	F9 B5 B1051.4
15600	F9 B5 B1049.5
15600	F9 B5 B1060.2
15600	F9 B5 B1058.3
15600	F9 B5 B1051.6
15600	F9 B5 B1060.3
15600	F9 B5 B1049.7

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

In [71]:
```
%sql select count("LANDING _OUTCOME"), "DATE" \
    from SPACEXTBL1 \
    where "DATE" between '04-06-2010' and '20-03-2017' = ("LANDING _OUTCOME" like 'Success%') \
    group by "LANDING _OUTCOME" \
    order by count("LANDING _OUTCOME") DESC
```
```
 * sqlite:///my_data1.db
Done.
```
Out[71]:

count("LANDING _OUTCOME")	Date
20	07-08-2018
11	22-05-2012
8	08-04-2016
6	18-07-2016
2	29-09-2013
2	23-12-2017
1	28-06-2015
1	04-03-2016

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

In [30]:
```
%sql select "Launch_Site" from SPACEXTBL1 \
    where "Launch_Site" like 'CCA%' \
    limit 5
```
```
 * sqlite:///my_data1.db
Done.
```
Out[30]:

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

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

Hint:Use min function

In [43]:
```
%sql select min("DATE") \
    from SPACEXTBL1 \
    where "LANDING _OUTCOME" = "Success (ground pad)"
```
```
 * sqlite:///my_data1.db
Done.
```
Out[43]:

min("DATE")
01-05-2017

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

In [44]:
```
%sql select "BOOSTER_VERSION", "LANDING _OUTCOME", "PAYLOAD_MASS__KG_" \
    from SPACEXTBL1 \
    where "LANDING _OUTCOME" = "Success (drone ship)" \
    and "PAYLOAD_MASS__KG_" between 4000 and 6000
```
```
 * sqlite:///my_data1.db
Done.
```
Out[44]:

Booster_Version	Landing _Outcome	PAYLOAD_MASS__KG_
F9 FT B1022	Success (drone ship)	4696
F9 FT B1026	Success (drone ship)	4600
F9 FT B1021.2	Success (drone ship)	5300
F9 FT B1031.2	Success (drone ship)	5200

List the records which will display the month

Note: SQLLite does not support monthnames. So you need to use substr(Date, 4, 2) as month to get the months and substr(Date,7,4)='2015' for year.

In [73]:
```
%sql select substr(Date, 4, 2) as 'Month_Number', substr(Date,7,4) as 'Year', \
    "BOOSTER_VERSION", "LAUNCH_SITE" \
    from SPACEXTBL1 \
    where substr(Date,7,4)='2015' and "LANDING _OUTCOME" = 'Failure (drone ship)'
```
```
 * sqlite:///my_data1.db
Done.
```
Out[73]:

Month_Number	Year	Booster_Version	Launch_Site
01	2015	F9 v1.1 B1012	CCAFS LC-40
04	2015	F9 v1.1 B1015	CCAFS LC-40

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

In [32]:
```
%sql select SUM("PAYLOAD_MASS__KG_"), "Customer"\
    from SPACEXTBL1 \
    where Customer = 'NASA (CRS)' \
    group by Customer
```
```
 * sqlite:///my_data1.db
Done.
```
Out[32]:

SUM("PAYLOAD_MASS__KG_")	Customer
45596	NASA (CRS)

SKILLS NETWORK

Interactive map with Folium

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

```python
[15]:  #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():
    # TODO: Create and add a Marker cluster to the site map
    lat = record['Lat']
    long = record['Long']
    coordinate = [lat, long]
    marker = folium.Marker(
        coordinate,
        icon=folium.Icon(color='white', icon_color=record['marker_color'])
        )
    marker_cluster.add_child(marker)

site_map
```

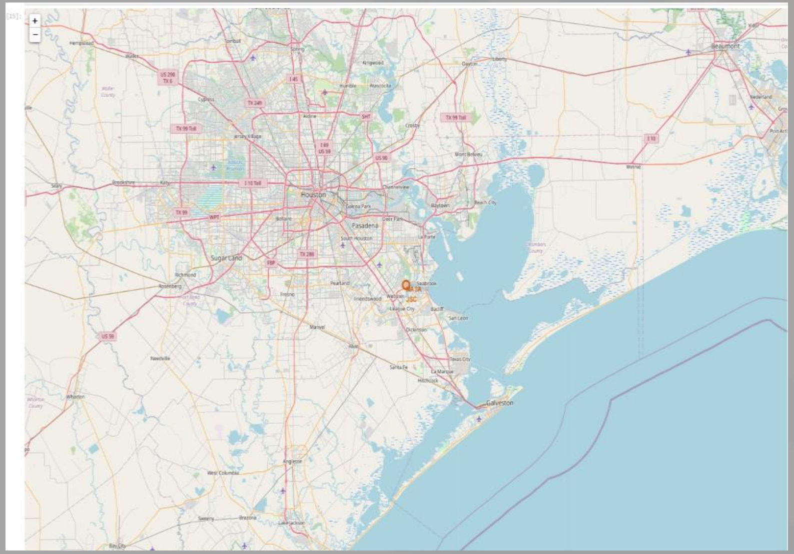
```python
folium.map.Marker(coordinate, icon=DivIcon(icon_size=(20,20),icon_anchor=(0,0), html='<div style="font-size: 12; color:#d35400;"><b>%s</b></div>' % 'label', ))
```

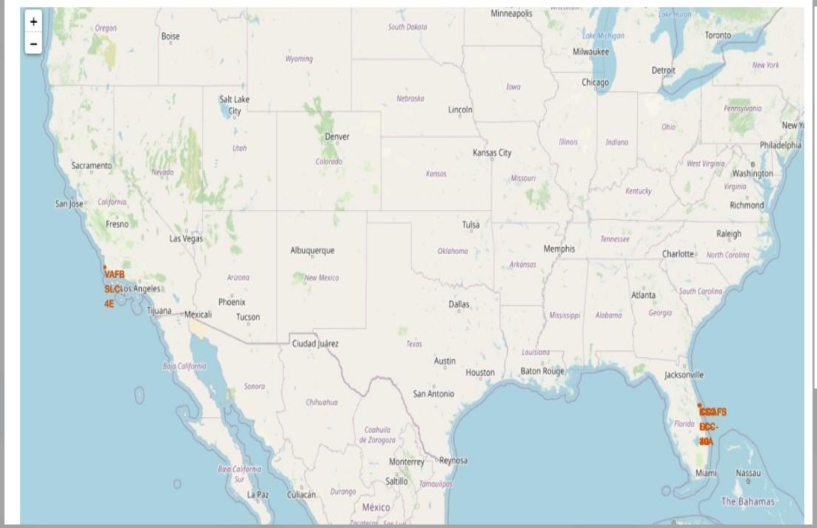
```python
[8]: # For each launch site, add a Circle object based on its coordinate (Lat, Long) values. In addition, add Launch site name as a popup label
for index,site in launch_sites_df.iterrows():
    lat = site['Lat']
    long = site['Long']
    coordinate = [lat, long]
    circle = folium.Circle(
        location=coordinate,
        radius=5000,
        color='#000000',
        fill=True).add_child(folium.Popup(site['Launch Site']))
    marker = folium.map.Marker(
        coordinate,
        icon=DivIcon(
            icon_size=(20,20),
            icon_anchor=(0,0),
            html='<div style="font-size: 12; color:#d35400;"><b>%s</b></div>' % site['Launch Site'],
        )
    )
    site_map.add_child(circle)
    site_map.add_child(marker)

site_map
```

TODO: Draw a `PolyLine` between a launch site to the selected coastline point

```python
# Create a `folium.PolyLine` object using the coastline coordinates and launch site coordinate
# lines=folium.PolyLine(locations=coordinates, weight=1)
# Create a `folium.PolyLine` object using the coastline coordinates and launch site coordinate
# lines=folium.PolyLine(locations=coordinates, weight=1)
coast_coordinate = [coastline_lat, coastline_lon]
launch_coordinate = [launch_site_lat, launch_site_lon]
coordinates = [coast_coordinate, launch_coordinate]
lines = folium.PolyLine(locations=coordinates, weight=1)
site_map.add_child(lines)
site_map
```





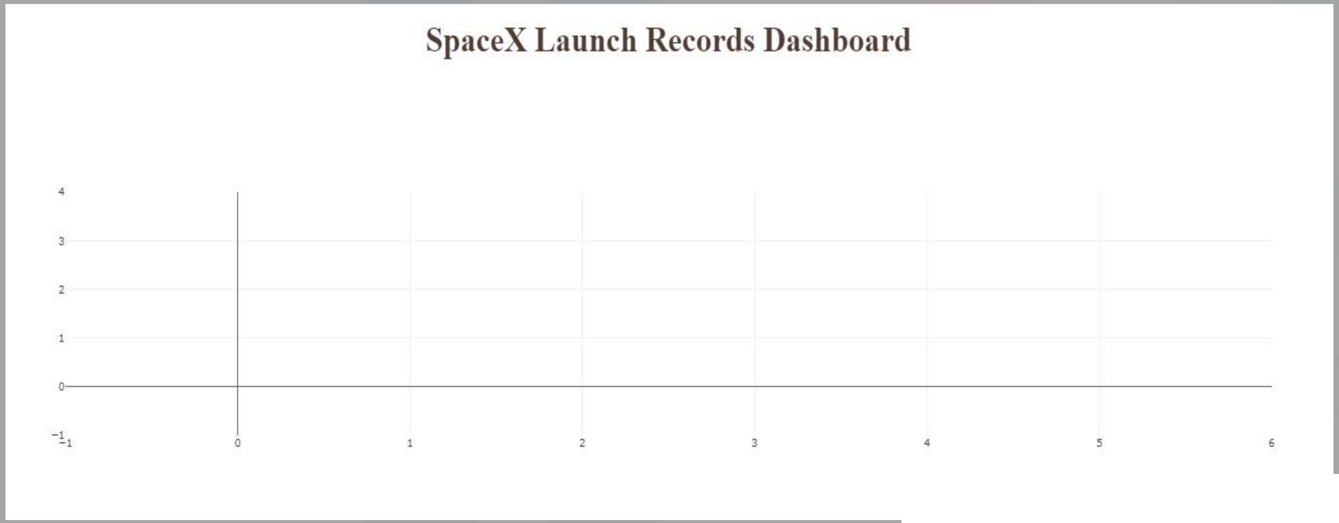
IBM Developer

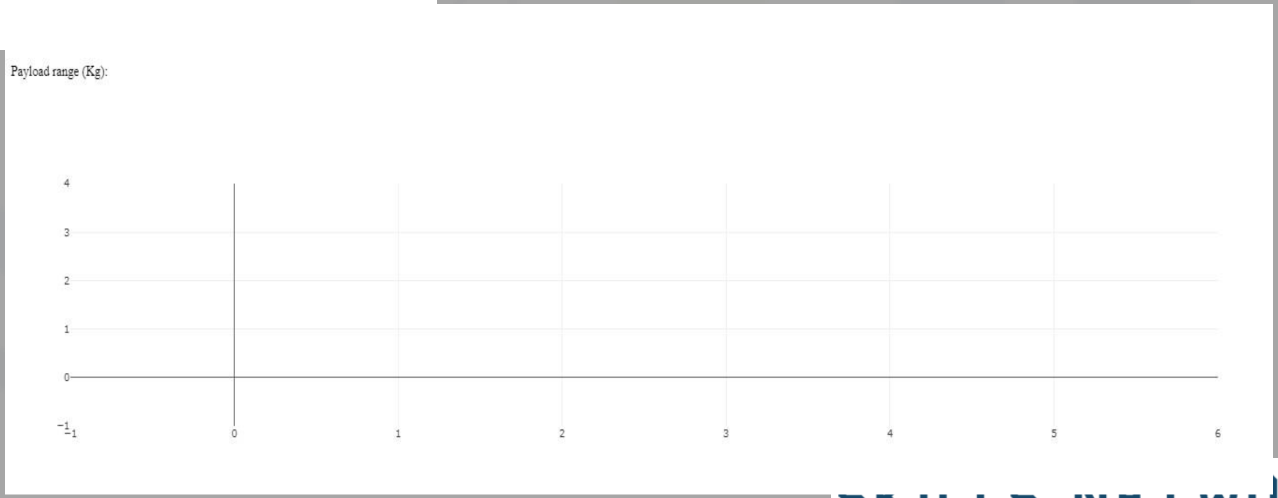
SKILLS NETWORK

Plotly Dash dashboard

Link to Code :

https://github.com/Teekafy/Treasure_proj/blob/c1fba45b132d661b31ac248b66a5355003838975/spacex_dash_app.py





Predictive Analysis

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 = data['Class'].to_numpy()
Y
```

```
Out[5]:
array([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])
```

TASK 2

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

```
In [6]:
# students get this
transform = preprocessing.StandardScaler()
```

```
In [7]:
X = transform.fit_transform(X)
```

We split the data into training and testing data using the function `train_test_split`. The training data is divided into validation data, a second set used for training data; then the models are trained and hyperparameters are selected using the function `GridSearchCV`.

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

`X_train, X_test, Y_train, Y_test`

```
In [8]:
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.

```
In [9]:
Y_test.shape
```

```
Out[9]: (18,)
```

TASK 4

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

```
In [10]:
parameters ={'C':[0.01,0.1,1],
             'penalty':['l2'],
             'solver':['lbfgs']}

lr=LogisticRegression()
```

```
In [11]:
logreg_cv = GridSearchCV(lr,parameters,cv=10)
logreg_cv.fit(X_train, Y_train)
```

```
Out[11]:
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 [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
```

TASK 5

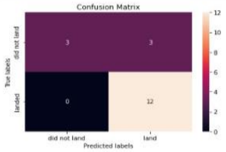
Calculate the accuracy on the test data using the method `score`:

```
In [13]:
print("test set accuracy :",logreg_cv.score(X_test, Y_test))
```

```
test set accuracy : 0.8333333333333334
```

Lets look at the confusion matrix:

```
In [14]:
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.

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`.

```
In [15]:
parameters = {'kernel':('linear', 'rbf','poly','rbf', 'sigmoid'),
              'C': np.logspace(-3, 3, 5),
              'gamma':np.logspace(-3, 3, 5)}
svm = SVC()
```

```
In [16]:
svm_cv = GridSearchCV(svm,parameters,cv=10)
svm_cv.fit(X_train, Y_train)
```

```
Out[16]:
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')})
```

```
In [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
```

TASK 7

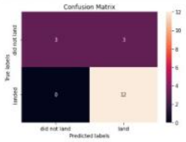
Calculate the accuracy on the test data using the method `score`:

```
In [18]:
print("test set accuracy :",svm_cv.score(X_test, Y_test))
```

```
test set accuracy : 0.8333333333333334
```

We can plot the confusion matrix

```
In [19]:
yhat=svm_cv.predict(X_test)
plot_confusion_matrix(Y_test,yhat)
```



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 [20]:
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 [21]:
tree_cv = GridSearchCV(tree,parameters,cv=10)
tree_cv.fit(X_train, Y_train)
```

```
Out[21]:
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 [22]:
print("tuned hpyerparameters :(best parameters) ",tree_cv.best_params_)
print("accuracy :",tree_cv.best_score_)
```

```
tuned hpyerparameters :(best parameters)  {'criterion': 'gini', 'max_depth': 16, 'max_features': 'auto', 'min_samples_leaf': 4, 'min_samples_split': 5, 'splitter': 'random'}
accuracy : 0.8767857142857143
```

continued

TASK 9

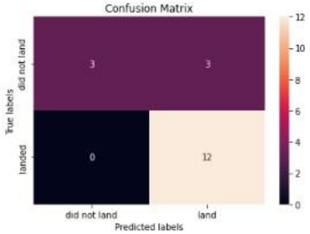
Calculate the accuracy of tree_cv on the test data using the method `score` :

In [23]:
```
print("test set accuracy :",tree_cv.score(X_test, Y_test))
```

```
test set accuracy : 0.8333333333333334
```

We can plot the confusion matrix

In [24]:
```
yhat = svm_cv.predict(X_test)
plot_confusion_matrix(Y_test,yhat)
```



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` .

In [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()
```

In [26]:
```
knn_cv = GridSearchCV(KNN,parameters,cv=10)
knn_cv.fit(X_train, Y_train)
```

Out[26]:
```
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]})
```

In [27]:
```
print("tuned hpyerparameters :(best parameters) ",knn_cv.best_params_)
print("accuracy :",knn_cv.best_score_)
```

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

TASK 12

Find the method performs best:

In [25]:
```
models = {'KNeighbors':knn_cv.best_score_,
              'DecisionTree':tree_cv.best_score_,
              'LogisticRegression':logreg_cv.best_score_,
              'SupportVector': svm_cv.best_score_}

bestalgorithm = max(models, key=models.get)
print('Best model is', bestalgorithm,'with a score of', models[bestalgorithm])
if bestalgorithm == 'DecisionTree':
    print('Best params is :', tree_cv.best_params_)
if bestalgorithm == 'KNeighbors':
    print('Best params is :', knn_cv.best_params_)
if bestalgorithm == 'LogisticRegression':
    print('Best params is :', logreg_cv.best_params_)
if bestalgorithm == 'SupportVector':
    print('Best params is :', svm_cv.best_params_)
```

```
Best model is DecisionTree with a score of 0.8732142857142856
Best params is : {'criterion': 'gini', 'max_depth': 6, 'max_features': 'auto', 'min_samples_leaf': 2, 'min_samples_split': 5, 'splitter': 'random'}
```

TASK 11

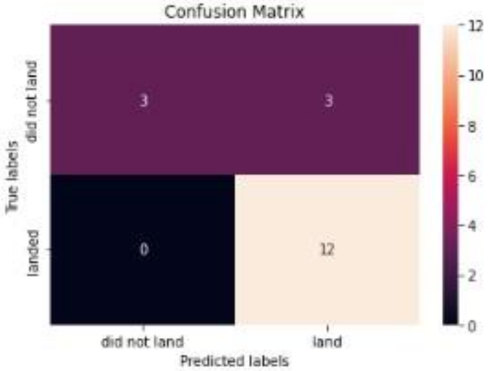
Calculate the accuracy of tree_cv on the test data using the method `score` :

In [28]:
```
print("test set accuracy :",knn_cv.score(X_test, Y_test))
```

```
test set accuracy : 0.8333333333333334
```

We can plot the confusion matrix

In [29]:
```
yhat = knn_cv.predict(X_test)
plot_confusion_matrix(Y_test,yhat)
```



Conclusion and Insights

- The SVM, KNN and Logistic Regression models are the best in terms of prediction accuracy for this dataset.
- Low weighted payloads perform better than heavier payloads.
- KSC LC 39A had the most successful launches from all the sites.
- Orbit GEO, HEO, SSO, ES l1 have the best success rate.

APPENDICES

https://en.wikipedia.org/wiki/SpaceX

https://www.youtube.com/c/spacex