Space X Falcon 9

Scientist Analyzing

M Sadewa Wicaksana W

11 July 2024

Outline

- Executive Summary
- Introduction
- Methodology
- Results
- Conclusion

Executive Summary

- Summary of methodologies

1. Data Collection SpaceX API
2. Data Collection with Web Scraping
3. Data Wrangling
4. Data Exploratory Data Analysis
5. EDA DataViz

- Summary of all results

1. EDA Results
2. Interactive Visual Analytics and Dashboards

Introduction

SpaceX 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 SpaceX 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 SpaceX for a rocket launch

Data Collection

Data collected from public website starlink space X Falcon 9. To get the data clear without noise data is normalize using *json_normalize()*

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

To make the requested JSON results more consistent, we will use the following static response object for this project:

[8]:
```
static_json_url='https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/datasets/API_call_spacex_api.json'
```

We should see that the request was successfull with the 200 status response code

[9]:
```
response.status_code
```

[9]: 200

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

[10]:
```
# Use json_normalize meethod to convert the json result into a dataframe
import pandas as pd

# Assuming `response` is the variable that holds the response object from requests.get()
resJson = response.json()

# Convert the JSON data into a Pandas DataFrame
data = pd.json_normalize(resJson)
# response.json_normalize()
```

Using the dataframe `data` print the first 5 rows

[11]:
```
# Get the head of the dataframe
data.head(5)
```

[11]:

	static_fire_date_utc	static_fire_date_unix	net	window	rocket	success	failures	details	crew	ships	capsules	payloads	
0	2006-03-17T00:00:00.000Z	1.142554e+09	False	0.0	5e9d0d95eda69955f709d1eb	False	[{'time': 33, 'altitude': None, 'reason': 'merlin engine failure'}]	Engine failure at 33 seconds and loss of vehicle	[]	[]	[]	[5eb0e4b5b6c3bb0006eeb1e1]	5e9e45
							[{'time': 301,	Successful first stage burn and transition to second stage, maximum					

Data Wrangling

With data wrangling, exploratory data is started with check available nullable data and check data type in each columns.

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

[3]: FlightNumber       0.000000
     Date               0.000000
     BoosterVersion     0.000000
     PayloadMass        0.000000
     Orbit              0.000000
     LaunchSite         0.000000
     Outcome            0.000000
     Flights            0.000000
     GridFins           0.000000
     Reused             0.000000
     Legs               0.000000
     LandingPad        28.888889
     Block              0.000000
     ReusedCount        0.000000
     Serial             0.000000
     Longitude          0.000000
     Latitude           0.000000
     dtype: float64
```

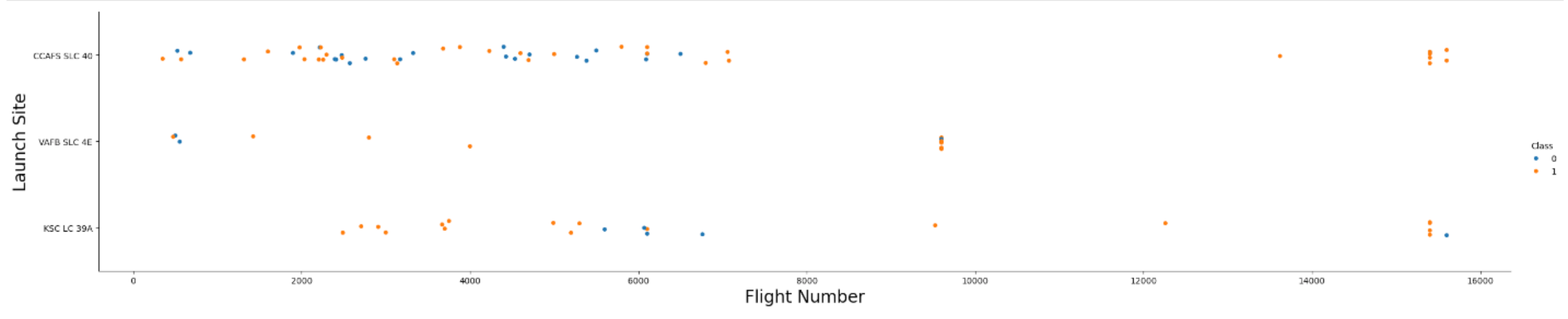
Identify which columns are numerical and categorical:

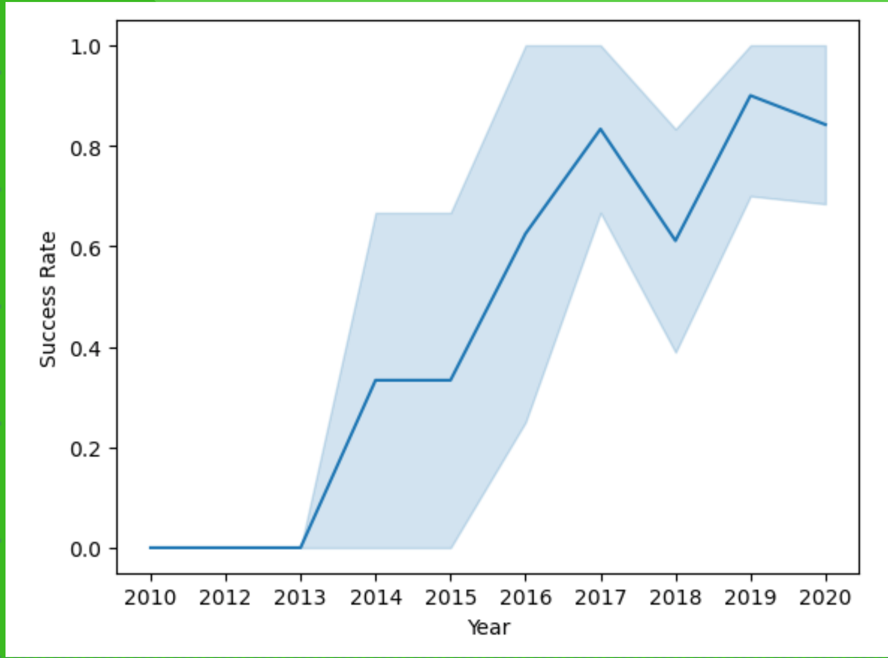
```
[4]: df.dtypes

[4]: FlightNumber        int64
     Date               object
     BoosterVersion     object
     PayloadMass       float64
     Orbit              object
     LaunchSite         object
     Outcome            object
     Flights             int64
     GridFins             bool
     Reused               bool
     Legs                 bool
     LandingPad         object
     Block             float64
     ReusedCount         int64
     Serial             object
     Longitude         float64
     Latitude          float64
     dtype: object
```

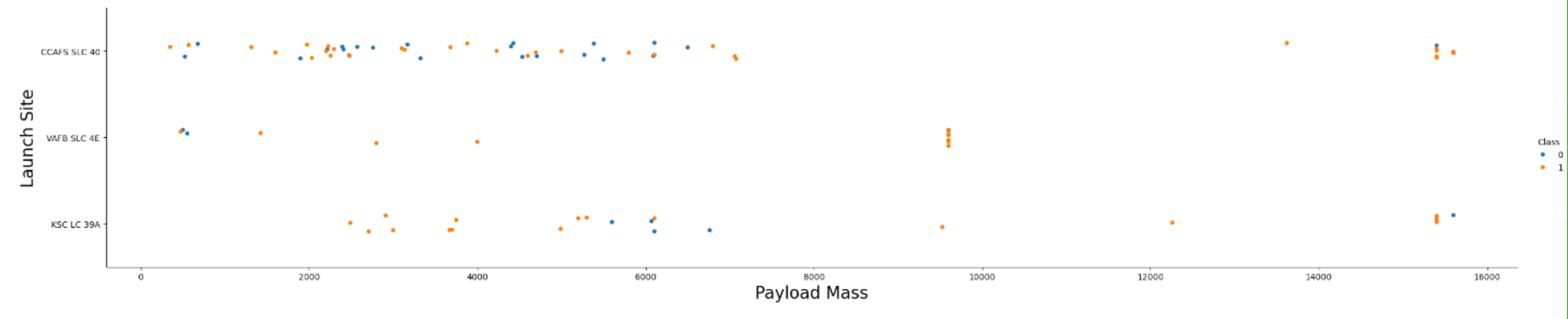
EDA with Data Visualization

1. Performed data Analysis and Feature Engineering using Pandas and Matplotlib i.e *Exploratory Data Analysis* and *Preparing Data Feature Engineering*
2. Used scatter plots to Visualize the relationship between Flight Number and Launch Site, Payload and Launch Site, FlightNumber and Orbit type, Payload and Orbit type.
3. Used Bar chart to Visualize the relationship between success rate of each orbit type
4. Line plot to Visualize the launch success yearly trend

EDA with Data Visualization (Contd..)







EDA with SQL

Display the names of unique launch sites

```
[10]: %sql SELECT DISTINCT ST.Launch_Site FROM SPACEXTABLE ST

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

[10]:

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

Display 5 records begin with string CCA

```
[11]: %sql SELECT * FROM SPACEXTABLE ST WHERE ST.Launch_Site LIKE 'CCA%' LIMIT 5

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

[11]:

Date	Time (UTC)	Booster_Version	Launch_Site	Payload	PAYLOAD_MASS__KG_	Orbit	Customer	Mission_Outcome	Landing_Outcome
2010-06-04	18:45:00	F9 v1.0 B0003	CCAFS LC-40	Dragon Spacecraft Qualification Unit	0	LEO	SpaceX	Success	Failure (parachute)
2010-12-08	15:43:00	F9 v1.0 B0004	CCAFS LC-40	Dragon demo flight C1, two CubeSats, barrel of Brouere cheese	0	LEO (ISS)	NASA (COTS) NRO	Success	Failure (parachute)
2012-05-22	7:44:00	F9 v1.0 B0005	CCAFS LC-40	Dragon demo flight C2	525	LEO (ISS)	NASA (COTS)	Success	No attempt
2012-10-08	0:35:00	F9 v1.0 B0006	CCAFS LC-40	SpaceX CRS-1	500	LEO (ISS)	NASA (CRS)	Success	No attempt
2013-03-01	15:10:00	F9 v1.0 B0007	CCAFS LC-40	SpaceX CRS-2	677	LEO (ISS)	NASA (CRS)	Success	No attempt

EDA With SQL (Contd..)

List names of the boosters with payload mass greater than 4000 and less than 6000

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

[25]:
```
%sql SELECT * FROM SPACEXTABLE ST WHERE ST.Landing_Outcome = 'Success (drone ship)' AND ST.PAYLOAD_MASS__KG_ >= 4000 AND ST.PAYLOAD_MASS__KG_ <= 6000
```

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

[25]:

Date	Time (UTC)	Booster_Version	Launch_Site	Payload	PAYLOAD_MASS__KG_	Orbit	Customer	Mission_Outcome	Landing_Outcome
2016-05-06	5:21:00	F9 FT B1022	CCAFS LC-40	JCSAT-14	4696	GTO	SKY Perfect JSAT Group	Success	Success (drone ship)
2016-08-14	5:26:00	F9 FT B1026	CCAFS LC-40	JCSAT-16	4600	GTO	SKY Perfect JSAT Group	Success	Success (drone ship)
2017-03-30	22:27:00	F9 FT B1021.2	KSC LC-39A	SES-10	5300	GTO	SES	Success	Success (drone ship)
2017-10-11	22:53:00	F9 FT B1031.2	KSC LC-39A	SES-11 / EchoStar 105	5200	GTO	SES EchoStar	Success	Success (drone ship)

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

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

[20]:
```
%sql SELECT AVG(PAYLOAD_MASS__KG_) FROM SPACEXTABLE WHERE Booster_Version = 'F9 v1.1'
```

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

[20]:

AVG(PAYLOAD_MASS__KG_)
2928.4

Build an interactive map with folium

Calculate Distance between the coastline point and the launch site

```
coast_coordinates = [coastline_lat, coastline_lon]
distance_marker = folium.Marker(
    coast_coordinates,
    icon=DivIcon(
        icon_size=(20,20),
        icon_anchor=(0,0),
        html='%s' % "{:10.2f} KM".format(distance_coastline),)
    )
distance_marker.add_to(site_map)
site_map
```

Build a Dashboard with Plotly Dash

Built an interactive dashboard application with Plotly dash by:

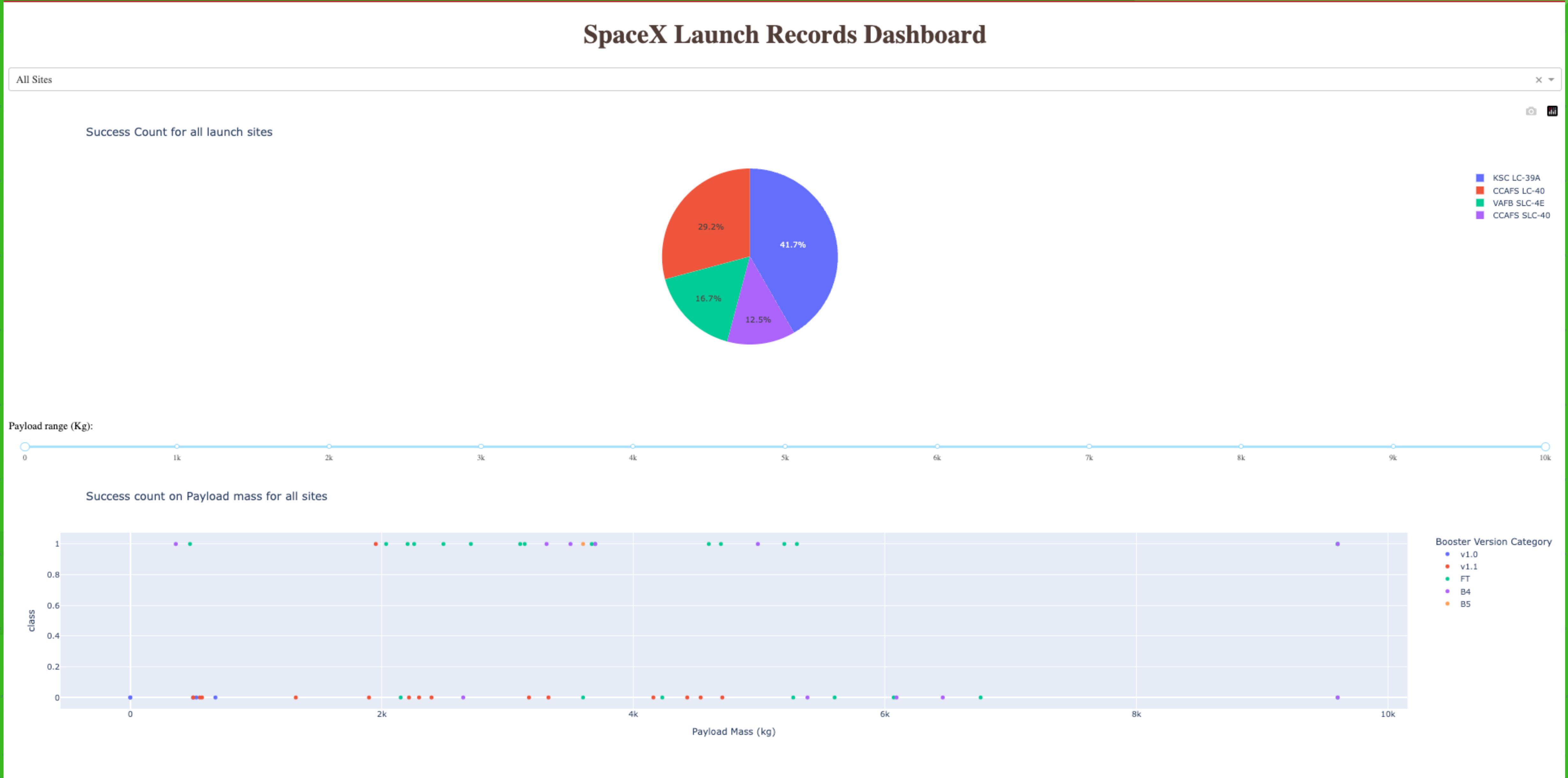
a. Adding a Launch Site Drop-down Input Component

b. Adding a callback function to render success-pie-chart based on selected site dropdown

c. Adding a Range Slider to Select Payload

d. Adding a callback function to render the success-payload-scatter-chart scatter plot

SpaceX Dash App



Predictive Analysis

In order to find the best ML model/ method that would performs best using the test data between SVM, Classification Trees, k nearest neighbors and Logistic Regression;

1. First created an object for each of the algorithms then created a GridSearchCV object and assigned them a set of parameters for each model.
2. For each of the models under evaluation, the GridsearchCV object was created with cv=10, then fit the training data into the GridSearch object for each to Find best Hyperparameter.
3. After fitting the training set, we output GridSearchCV object for each of the models, then displayed the best parameters using the data attribute best_params_ and the accuracy on the validation data using the data attribute best_score_.
4. Finally using the method score to calculate the accuracy on the test data for each model and plotted a confussion matrix for each using the test and predicted outcomes.

Summary

The table below shows the test data accuracy score for each of the methods comparing them to show which performed best using the test data between SVM, Classification Trees, k nearest neighbors and Logistic Regression;

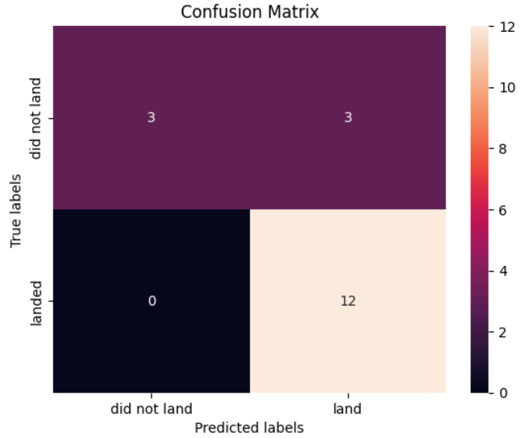
Calculate the accuracy of knn_cv on the test data using the method `score` :

```
[35]: knn_cv.score(X_test, Y_test)
```

```
[35]: 0.8333333333333334
```

We can plot the confusion matrix

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




```
[37]: Report = pd.DataFrame({'Method' : ['Test Data Accuracy']})

      knn_accuracy=knn_cv.score(X_test, Y_test)
      Decision_tree_accuracy=tree_cv.score(X_test, Y_test)
      SVM_accuracy=svm_cv.score(X_test, Y_test)
      Logistic_Regression=logreg_cv.score(X_test, Y_test)

      Report['Logistic_Reg'] = [Logistic_Regression]
      Report['SVM'] = [SVM_accuracy]
      Report['Decision Tree'] = [Decision_tree_accuracy]
      Report['KNN'] = [knn_accuracy]




      Report.transpose()
```

[37]:

	0
Method	Test Data Accuracy
Logistic_Reg	0.833333
SVM	0.833333
Decision Tree	0.833333
KNN	0.833333

Thanks