Import Libraries and Define Auxiliary Functions

We will import the following libraries for the lab

In [1]:
```
# Pandas is a software library written for the Python programming language for data manipulation and analysis.
import pandas as pd
# NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a la
import numpy as np
# Matplotlib is a plotting library for python and pyplot gives us a MatLab like plotting framework. We will use this in our plotter funct.
import matplotlib.pyplot as plt
#Seaborn is a Python data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and inform
import seaborn as sns
# Preprocessing allows us to standarsize our data
from sklearn import preprocessing
# Allows us to split our data into training and testing data
from sklearn.model_selection import train_test_split
# Allows us to test parameters of classification algorithms and find the best one
from sklearn.model_selection import GridSearchCV
# Logistic Regression classification algorithm
from sklearn.linear_model import LogisticRegression
# Support Vector Machine classification algorithm
from sklearn.svm import SVC
# Decision Tree classification algorithm
from sklearn.tree import DecisionTreeClassifier
# K Nearest Neighbors classification algorithm
from sklearn.neighbors import KNeighborsClassifier
```

This function is to plot the confusion matrix.

In [2]:
```
def plot_confusion_matrix(y,y_predict):
    "this function plots the confusion matrix"
    from sklearn.metrics import confusion_matrix

    cm = confusion_matrix(y, y_predict)
    ax= plt.subplot()
    sns.heatmap(cm, annot=True, ax = ax); #annot=True to annotate cells
    ax.set_xlabel('Predicted labels')
    ax.set_ylabel('True labels')
    ax.set_title('Confusion Matrix');
    ax.xaxis.set_ticklabels(['did not land', 'land']); ax.yaxis.set_ticklabels(['did not land', 'landed'])
```

Load the dataframe

Load the data

In [3]:
```
data = pd.read_csv("https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/datasets/dataset_part_2.

# If you were unable to complete the previous lab correctly you can uncomment and load this csv

#data = pd.read_csv('https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-DS0701EN-SkillsNetwork/a

data.head()
```

Out[3]:

	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins	Reused	Legs	LandingPad	Block	ReusedCount	Serial	Longit
0	1	2010-06-04	Falcon 9	6104.959412	LEO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B0003	-80.577
1	2	2012-05-22	Falcon 9	525.000000	LEO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B0005	-80.577
2	3	2013-03-01	Falcon 9	677.000000	ISS	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B0007	-80.577
3	4	2013-09-29	Falcon 9	500.000000	PO	VAFB SLC 4E	False Ocean	1	False	False	False	NaN	1.0	0	B1003	-120.610
4	5	2013-12-03	Falcon 9	3170.000000	GTO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B1004	-80.577

In [4]:
```
data.shape
```

Out[4]:
```
(90, 18)
```

In [5]:
```
#X = pd.read_csv('https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/datasets/dataset_part_3.c

# If you were unable to complete the previous lab correctly you can uncomment and load this csv

X = pd.read_csv('https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-DS0701EN-SkillsNetwork/api/d

X.head(100)
```

,

Out[5]:

	FlightNumber	PayloadMass	Flights	Block	ReusedCount	Orbit_ES-L1	Orbit_GEO	Orbit_GTO	Orbit_HEO	Orbit_ISS	...	Serial_B1058	Serial_B1059	Serial_B1060
0	1.0	6104.959412	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0
1	2.0	525.000000	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0
2	3.0	677.000000	1.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	...	0.0	0.0	0.0
3	4.0	500.000000	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0
4	5.0	3170.000000	1.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	...	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
85	86.0	15400.000000	2.0	5.0	2.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	1.0
86	87.0	15400.000000	3.0	5.0	2.0	0.0	0.0	0.0	0.0	0.0	...	1.0	0.0	0.0
87	88.0	15400.000000	6.0	5.0	5.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0
88	89.0	15400.000000	3.0	5.0	2.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	1.0
89	90.0	3681.000000	1.0	5.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0

90 rows × 83 columns

In [6]:

```
X.shape
```

Out[6]:

```
(90, 83)
```

In [7]:

```
X.dtypes
```

Out[7]:

```
FlightNumber     float64
PayloadMass      float64
Flights          float64
Block            float64
ReusedCount      float64
                  ...
GridFins_True    float64
Reused_False     float64
Reused_True      float64
Legs_False       float64
Legs_True        float64
Length: 83, dtype: object
```

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 [8]:

```
data['Class'].dtypes
data.dtypes
```

Out[8]:

```
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
Class               int64
dtype: object
```

In [9]:

```
Y = data["Class"].to_numpy()
Y[0:5]
```

Out[9]:

```
array([0, 0, 0, 0, 0])
```

TASK 2

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

In [10]:

```
# students get this
transform = preprocessing.StandardScaler()
```

In [11]:

```
X = preprocessing.StandardScaler().fit(X).transform(X)
#X = transform.fit(X).transform(X)
X[0:5]
```

```
Out[11]: array([[-1.71291154e+00, -1.94814463e-16, -6.53912840e-01,
                 -1.57589457e+00, -9.73440458e-01, -1.05999788e-01,
                 -1.05999788e-01, -6.54653671e-01, -1.05999788e-01,
                 -5.51677284e-01,  3.44342023e+00, -1.85695338e-01,
                 -3.33333333e-01, -1.05999788e-01, -2.42535625e-01,
                 -4.29197538e-01,  7.97724035e-01, -5.68796459e-01,
                 -4.10890702e-01, -4.10890702e-01, -1.50755672e-01,
                 -7.97724035e-01, -1.50755672e-01, -3.92232270e-01,
                  9.43398113e+00, -1.05999788e-01, -1.05999788e-01,
                 -1.05999788e-01, -1.05999788e-01, -1.05999788e-01,
                 -1.05999788e-01, -1.05999788e-01, -1.05999788e-01,
                 -1.05999788e-01, -1.05999788e-01, -1.05999788e-01,
                 -1.05999788e-01, -1.05999788e-01, -1.05999788e-01,
                 -1.05999788e-01, -1.05999788e-01, -1.05999788e-01,
                 -1.05999788e-01, -1.50755672e-01, -1.05999788e-01,
                 -1.05999788e-01, -1.05999788e-01, -1.05999788e-01,
                 -1.05999788e-01, -1.50755672e-01, -1.05999788e-01,
                 -1.50755672e-01, -1.50755672e-01, -1.05999788e-01,
                 -1.50755672e-01, -1.50755672e-01, -1.05999788e-01,
                 -1.05999788e-01, -1.50755672e-01, -1.50755672e-01,
                 -1.50755672e-01, -1.05999788e-01, -1.05999788e-01,
                 -1.05999788e-01, -1.50755672e-01, -2.15665546e-01,
                 -1.85695338e-01, -2.15665546e-01, -2.67261242e-01,
                 -1.05999788e-01, -2.42535625e-01, -1.05999788e-01,
                 -2.15665546e-01, -1.85695338e-01, -2.15665546e-01,
                 -1.85695338e-01, -1.05999788e-01,  1.87082869e+00,
                 -1.87082869e+00,  8.35531692e-01, -8.35531692e-01,
                  1.93309133e+00, -1.93309133e+00],
                [-1.67441914e+00, -1.19523159e+00, -6.53912840e-01,
                 -1.57589457e+00, -9.73440458e-01, -1.05999788e-01,
                 -1.05999788e-01, -6.54653671e-01, -1.05999788e-01,
                 -5.51677284e-01,  3.44342023e+00, -1.85695338e-01,
                 -3.33333333e-01, -1.05999788e-01, -2.42535625e-01,
                 -4.29197538e-01,  7.97724035e-01, -5.68796459e-01,
                 -4.10890702e-01, -4.10890702e-01, -1.50755672e-01,
                 -7.97724035e-01, -1.50755672e-01, -3.92232270e-01,
                 -1.05999788e-01,  9.43398113e+00, -1.05999788e-01,
                 -1.05999788e-01, -1.05999788e-01, -1.05999788e-01,
                 -1.05999788e-01, -1.05999788e-01, -1.05999788e-01,
                 -1.05999788e-01, -1.05999788e-01, -1.05999788e-01,
                 -1.05999788e-01, -1.05999788e-01, -1.05999788e-01,
                 -1.05999788e-01, -1.05999788e-01, -1.05999788e-01,
                 -1.05999788e-01, -1.50755672e-01, -1.05999788e-01,
                 -1.05999788e-01, -1.05999788e-01, -1.05999788e-01,
                 -1.05999788e-01, -1.50755672e-01, -1.05999788e-01,
                 -1.50755672e-01, -1.50755672e-01, -1.05999788e-01,
                 -1.50755672e-01, -1.50755672e-01, -1.05999788e-01,
                 -1.05999788e-01, -1.50755672e-01, -1.50755672e-01,
                 -1.50755672e-01, -1.05999788e-01, -1.05999788e-01,
                 -1.05999788e-01, -1.50755672e-01, -2.15665546e-01,
                 -1.85695338e-01, -2.15665546e-01, -2.67261242e-01,
                 -1.05999788e-01, -2.42535625e-01, -1.05999788e-01,
                 -2.15665546e-01, -1.85695338e-01, -2.15665546e-01,
                 -1.85695338e-01, -1.05999788e-01,  1.87082869e+00,
                 -1.87082869e+00,  8.35531692e-01, -8.35531692e-01,
                  1.93309133e+00, -1.93309133e+00],
```

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 [12]:

```
X_train, X_test, Y_train, Y_test = (train_test_split(X, Y,  test_size=0.2, random_state=2))
```

we can see we only have 18 test samples.

In [13]:

```
Y_test.shape
```

Out[13]:

```
(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 [14]:

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

In [15]:

```
#parameters ={"C":[0.01,0.1,1],'penalty':['l2'], 'solver':['lbfgs']}  # l1 lasso l2 ridge
lr=LogisticRegression()
logreg_cv = GridSearchCV(lr, parameters, cv = 10)
logreg_cv.fit(X_train, Y_train)
```

Out[15]:

```
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 [16]:

```
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

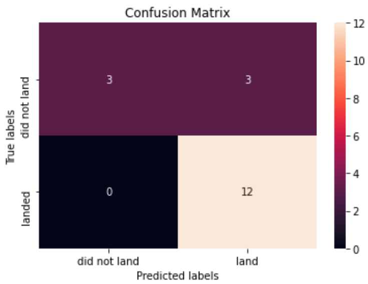
Calculate the accuracy on the test data using the method `score` :

In [17]:
```
logreg_cv.score(X_test, Y_test)
```

Out[17]:
```
0.8333333333333334
```

Lets look at the confusion matrix:

In [18]:
```
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 [19]:
```
parameters = {'kernel':('linear', 'rbf','poly','rbf', 'sigmoid'),
              'C': np.logspace(-3, 3, 5),
              'gamma':np.logspace(-3, 3, 5)}
svm = SVC()
```

In [20]:
```
svm_cv = GridSearchCV(svm, parameters, cv=10).fit(X_train, Y_train)
```

In [21]:
```
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

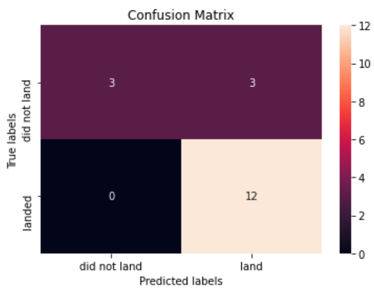
Calculate the accuracy on the test data using the method `score` :

In [22]:
```
svm_cv.score(X_test, Y_test)
```

Out[22]:
```
0.8333333333333334
```

We can plot the confusion matrix

In [23]:
```
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 [24]:
```
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 [25]:
```
tree_cv = GridSearchCV(tree, parameters, cv = 10).fit(X_train, Y_train)
```

In [26]:
```
print("tuned hpyerparameters :(best parameters) ",tree_cv.best_params_)
print("accuracy :",tree_cv.best_score_)
```

```
tuned hpyerparameters :(best parameters)  {'criterion': 'entropy', 'max_depth': 14, 'max_features': 'auto', 'min_samples_leaf': 2, 'min_sa
mples_split': 10, 'splitter': 'best'}
accuracy : 0.9142857142857143
```

TASK 9

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

In [27]:
```
tree_cv.score(X_test, Y_test)
```

Out[27]:
```
0.9444444444444444
```

We can plot the confusion matrix

In [28]:
```
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 [29]:
```
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 [30]:
```
knn_cv = GridSearchCV(KNN, parameters, cv=10).fit(X_train, Y_train)
```

In [31]:
```
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 11

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

In [32]:
```
knn_cv.score(X_test, Y_test)
```

Out[32]:
```
0.8333333333333334
```

We can plot the confusion matrix

In [33]:
```
yhat = knn_cv.predict(X_test)
plot_confusion_matrix(Y_test,yhat)
```

TASK 12

Find the method performs best:

Model Evaluation Using Alternative Test Sets

In [49]:
```
from sklearn.metrics import f1_score
from sklearn.metrics import log_loss
from sklearn.metrics import jaccard_score
```

In [36]:
```
# np arrays to store intermediate result
Jaccard = np.full(4, np.nan)
F1 = np.full(4, np.nan)
LogLoss = np.full(4, np.nan)
Algorithm = np.array(4)
```

In [53]:

```
Algorithm = ["lr", "svm", "tree", "KNN"]

Jaccard[0] = jaccard_score(Y_test, logreg_cv.predict(X_test), average='weighted')
Jaccard[1] = jaccard_score(Y_test, svm_cv.predict(X_test), average='weighted')
Jaccard[2] = jaccard_score(Y_test, tree_cv.predict(X_test), average='weighted')
Jaccard[3] = jaccard_score(Y_test, knn_cv.predict(X_test), average='weighted')

F1[0] = f1_score(Y_test, logreg_cv.predict(X_test))
F1[1] = f1_score(Y_test, svm_cv.predict(X_test))
F1[2] = f1_score(Y_test, tree_cv.predict(X_test))
F1[3] = f1_score(Y_test, knn_cv.predict(X_test))

LogLoss[0] = log_loss(Y_test, logreg_cv.predict(X_test))

The_Method_Perform_Best = pd.DataFrame({"Jaccard-Score":Jaccard, "F1-Score":F1, "LogLoss-Score":LogLoss}, index=Algorithm)
The_Method_Perform_Best
```

Out[53]:

	Jaccard-Score	F1-Score	LogLoss-Score
lr	0.700000	0.888889	5.756596
svm	0.700000	0.888889	NaN
tree	0.893162	0.960000	NaN
KNN	0.700000	0.888889	NaN

In [64]:

```
df = {'Methods': ['Logreg', 'SVM', 'Tree', 'KNN'],
      'Test1_Accuracy': [f1_score(Y_test, logreg_cv.predict(X_test)),
      f1_score(Y_test, svm_cv.predict(X_test)),
      f1_score(Y_test, tree_cv.predict(X_test)),
      f1_score(Y_test, knn_cv.predict(X_test))],
      'Test2_Accuracy': [jaccard_score(Y_test, logreg_cv.predict(X_test), average='weighted'),
      jaccard_score(Y_test, svm_cv.predict(X_test), average='weighted'),
      jaccard_score(Y_test, tree_cv.predict(X_test), average='weighted'),
      jaccard_score(Y_test, knn_cv.predict(X_test), average='weighted')],
      'Best_Accuracy': [logreg_cv.best_score_,
                        svm_cv.best_score_,
                        tree_cv.best_score_,
                        knn_cv.best_score_]

     }

df=pd.DataFrame(df)
```

In [65]:

```
df
```

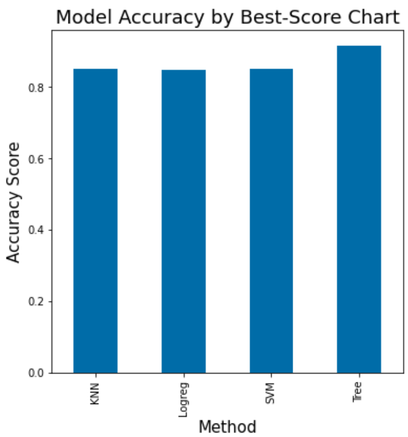
Out[65]:

	Methods	Test1_Accuracy	Test2_Accuracy	Best_Accuracy
0	Logreg	0.888889	0.700000	0.846429
1	SVM	0.888889	0.700000	0.848214
2	Tree	0.960000	0.893162	0.914286
3	KNN	0.888889	0.700000	0.848214

In [66]:

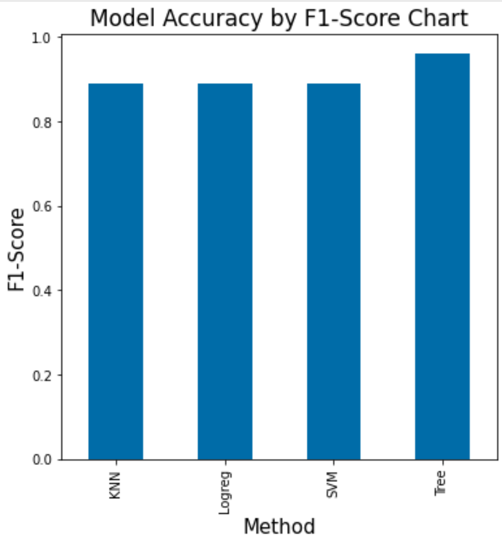
```
Accuracy_df= df.groupby(['Methods'])['Best_Accuracy'].mean()


Accuracy_df.plot(kind='bar', figsize=(6,6))
plt.xlabel('Method', fontsize=15)
plt.ylabel('Accuracy Score', fontsize=15)
plt.title('Model Accuracy by Best-Score Chart', fontsize=18)
plt.show()
```



In [67]:

```
F1_Score_df = df.groupby(['Methods'])['Test1_Accuracy'].mean()
F1_Score_df.plot(kind='bar', figsize=(6,6))
plt.xlabel('Method', fontsize=15)
plt.ylabel('F1-Score', fontsize=15)
plt.title('Model Accuracy by F1-Score Chart', fontsize=17)
plt.show()
```



In [68]:
```
F1_Score_df = df.groupby(['Methods'])['Test2_Accuracy'].mean()
F1_Score_df.plot(kind='bar', figsize=(6,6))
plt.xlabel('Method', fontsize=15)
plt.ylabel('Jaccard-Score', fontsize=15)
plt.title('Model Accuracy by Jaccard-Score Chart', fontsize=17)
plt.show()
```