APPLIED DATA SCIENCE CAPSTONE

SUMMARY

Introduction to Data Science
Data Collection
Data Preprocessing
Exploratory Data Analysis
Modeling and Machine Learning

Introduction to Data Science:

In this capstone, we will predict if the Falcon 9 first stage will land successfully. 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. In this module, you will be provided with an overview of the problem and the tools you need to complete the course.

Learning Objectives:

- Develop Python code to manipulate data in a Pandas data frame
- Convert a JSON file into a Create a Python Pandas data frame by converting a JSON file
- Create a Jupyter notebook and make it sharable using GitHub
- Utilize data science methodologies to define and formulate a real-world business problem
- Utilize your data analysis tools to load a dataset, clean it, and find out interesting insights from it

Data Collection :

In this capstone, we will predict if the Falcon 9 first stage will land successfully. 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. In this lab, you will collect and make sure the data is in the correct format from an API. The following is an example of a successful and launch.

Result:

[30]:

	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins	Reused	Legs	LandingPad	Block	ReusedCount	Serial	Longitude	Latitude
4	1	2010-06-04	Falcon 9	NaN	LEO	CCSFS SLC 40	None None	1	False	False	False	None	1.0	0	B0003	-80.577366	28.561857
5	2	2012-05-22	Falcon 9	525.0	LEO	CCSFS SLC 40	None None	1	False	False	False	None	1.0	0	B0005	-80.577366	28.561857
6	3	2013-03-01	Falcon 9	677.0	ISS	CCSFS SLC 40	None None	1	False	False	False	None	1.0	0	B0007	-80.577366	28.561857
7	4	2013-09-29	Falcon 9	500.0	PO	VAFB SLC 4E	False Ocean	1	False	False	False	None	1.0	0	B1003	-120.610829	34.632093
8	5	2013-12-03	Falcon 9	3170.0	GTO	CCSFS SLC 40	None None	1	False	False	False	None	1.0	0	B1004	-80.577366	28.561857
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
89	86	2020-09-03	Falcon 9	15600.0	VLEO	KSC LC 39A	True ASDS	2	True	True	True	5e9e3032383ecb6bb234e7ca	5.0	12	B1060	-80.603956	28.608058
90	87	2020-10-06	Falcon 9	15600.0	VLEO	KSC LC 39A	True ASDS	3	True	True	True	5e9e3032383ecb6bb234e7ca	5.0	13	B1058	-80.603956	28.608058
91	88	2020-10-18	Falcon 9	15600.0	VLEO	KSC LC 39A	True ASDS	6	True	True	True	5e9e3032383ecb6bb234e7ca	5.0	12	B1051	-80.603956	28.608058
92	89	2020-10-24	Falcon 9	15600.0	VLEO	CCSFS SLC 40	True ASDS	3	True	True	True	5e9e3033383ecbb9e534e7cc	5.0	12	B1060	-80.577366	28.561857
93	90	2020-11-05	Falcon 9	3681.0	MEO	CCSFS SLC 40	True ASDS	1	True	False	True	5e9e3032383ecb6bb234e7ca	5.0	8	B1062	-80.577366	28.561857

90 rows × 17 columns

Data Wrangling:

In the data set, there are several different cases where the booster did not land successfully. Sometimes a landing was attempted but failed due to an accident; for example, True Ocean means the mission outcome was successfully landed to a specific region of the ocean while False Ocean means the mission outcome was unsuccessfully landed to a specific region of the ocean. True RTLS means the mission outcome was successfully landed to a ground pad False RTLS means the mission outcome was unsuccessfully landed to a ground pad.True ASDS means the mission outcome was successfully landed on a drone ship False ASDS means the mission outcome was unsuccessfully landed on a drone ship.

[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)
```

[2]:

	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins	Reused	Legs	LandingPad	Block	ReusedCount	Serial	Longitude	Latitude
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
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
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
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
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
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	28.561857
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	28.561857
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	28.561857
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	28.561857
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	28.561857

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

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

```
[4]: 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:

```
[5]: df.dtypes
```

```
[5]: 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
```

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

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

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

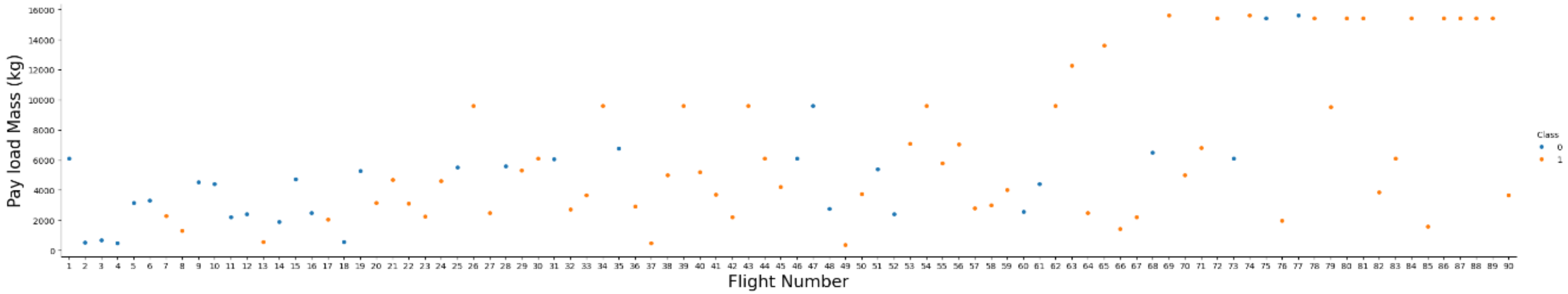
```
[6]: # Apply value_counts() on column LaunchSite
     df['LaunchSite'].value_counts()
```

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

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

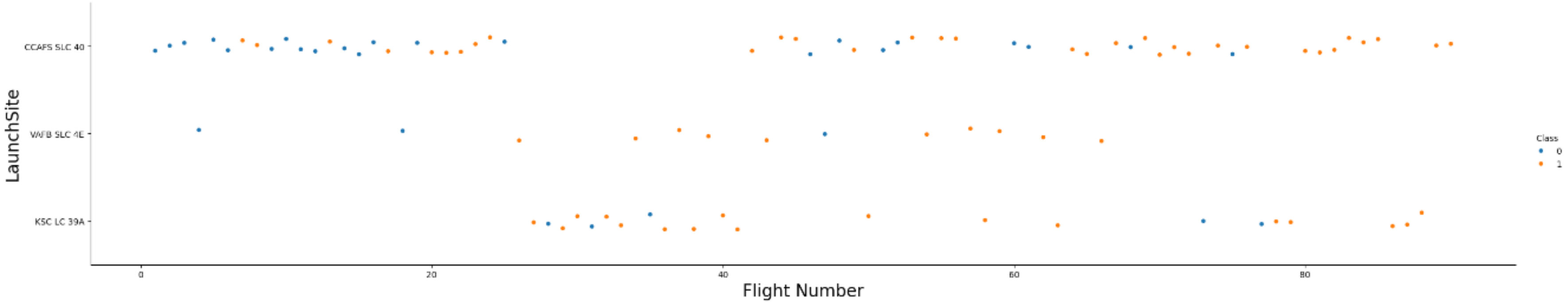
Exploring and Preparing Data:

```
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()
```

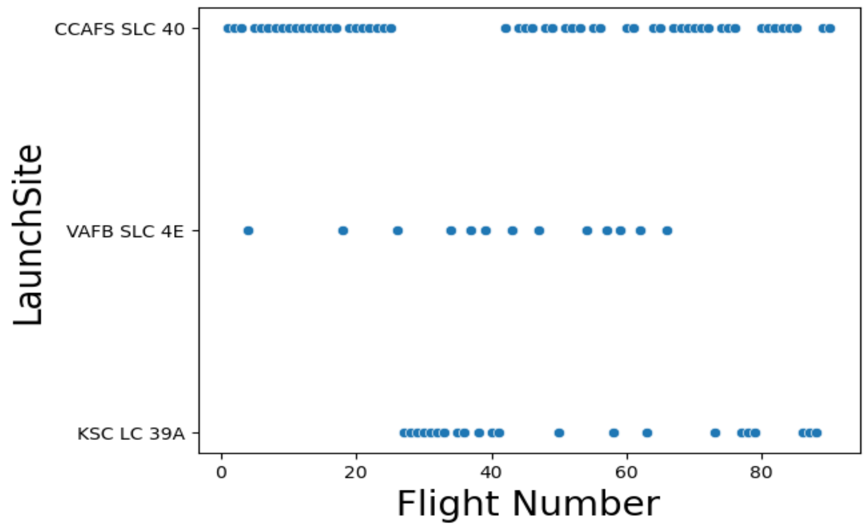


We see that different launch sites have different success rates. `CCAFS LC-40`, has a success rate of 60 %, while `KSC LC-39A` and `VAFB SLC 4E` has a success rate of 77%.

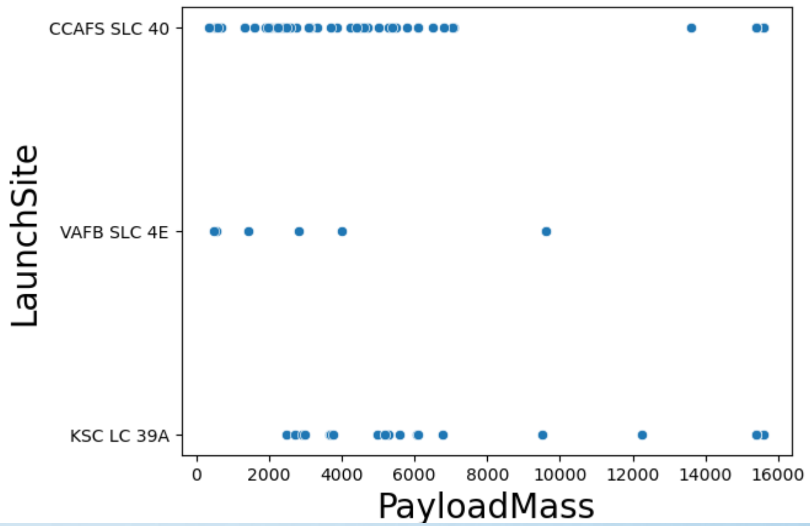
```
[5]: ### TASK 1: Visualize the relationship between Flight Number and Launch Site
sns.catplot(y="LaunchSite", x="FlightNumber", hue="Class", data=df, aspect = 5)
plt.xlabel("Flight Number",fontsize=20)
plt.ylabel("LaunchSite",fontsize=20)
plt.show()
```



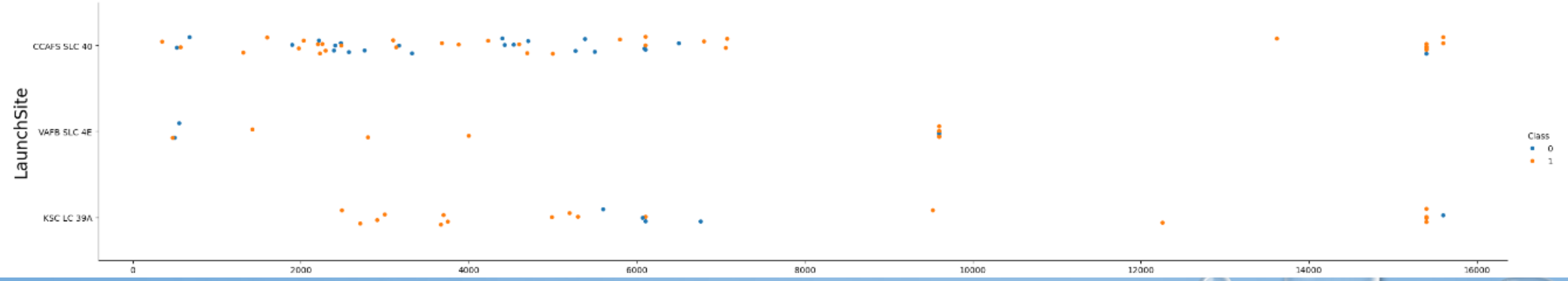
```python
# Plot a scatter point chart with x axis to be Flight Number and y axis to be the launch site, and hue to be the class value
sns.scatterplot(y="LaunchSite", x="FlightNumber", data=df)
plt.xlabel("Flight Number",fontsize=20)
plt.ylabel("LaunchSite",fontsize=20)
plt.show()
```



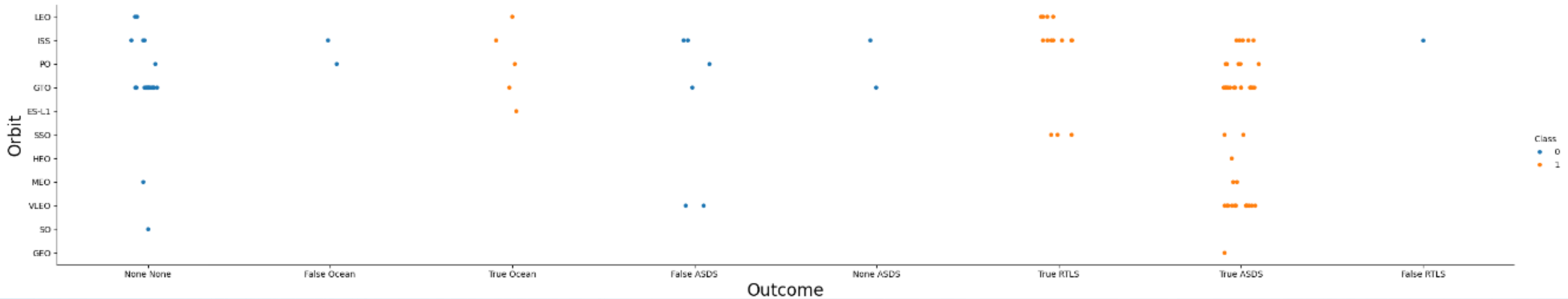

```python
# Plot a scatter point chart with x axis to be Pay Load Mass (kg) and y axis to be the launch site, and hue to be the class value
sns.scatterplot(y="LaunchSite", x="PayloadMass", data=df)
plt.xlabel("PayloadMass",fontsize=20)
plt.ylabel("LaunchSite",fontsize=20)
plt.show()
```

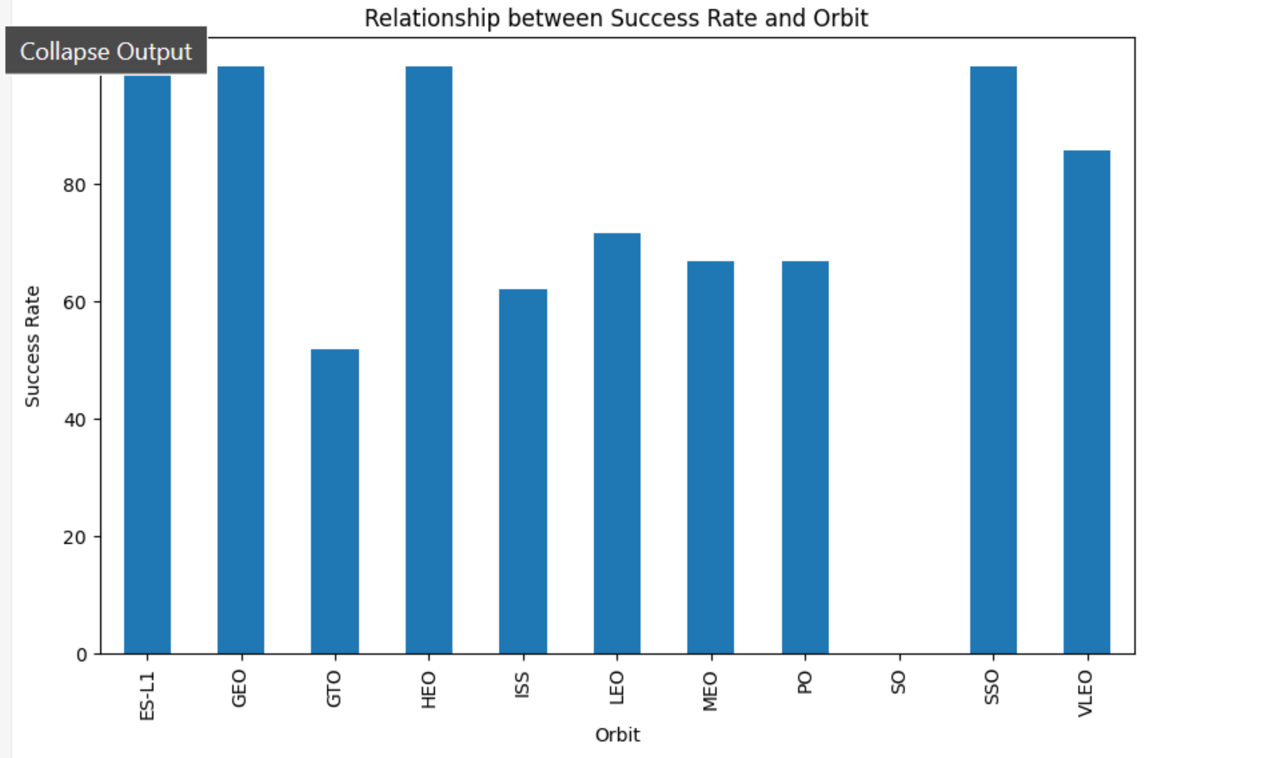


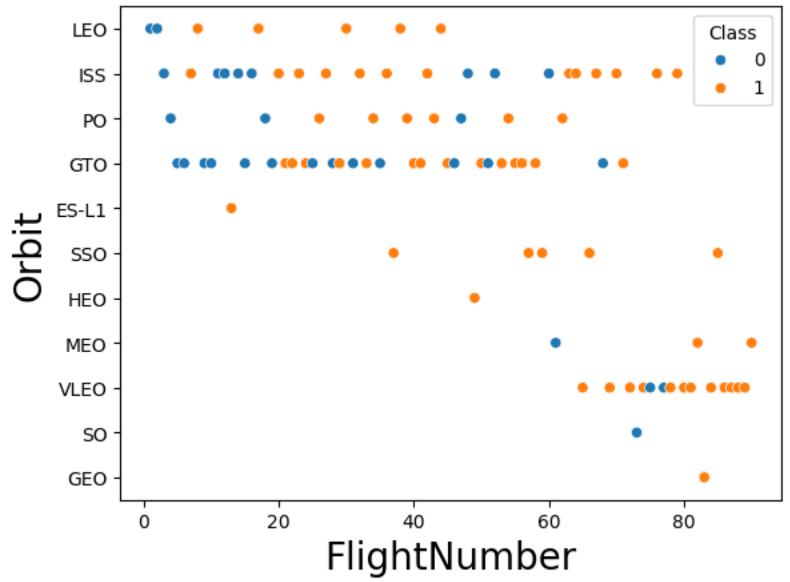

```python
### TASK 2: Visualize the relationship between Payload and Launch Site
sns.catplot(y="LaunchSite", x="PayloadMass", hue="Class", data=df, aspect = 5)
plt.xlabel("PayloadMass",fontsize=20)
plt.ylabel("LaunchSite",fontsize=20)
plt.show()
```

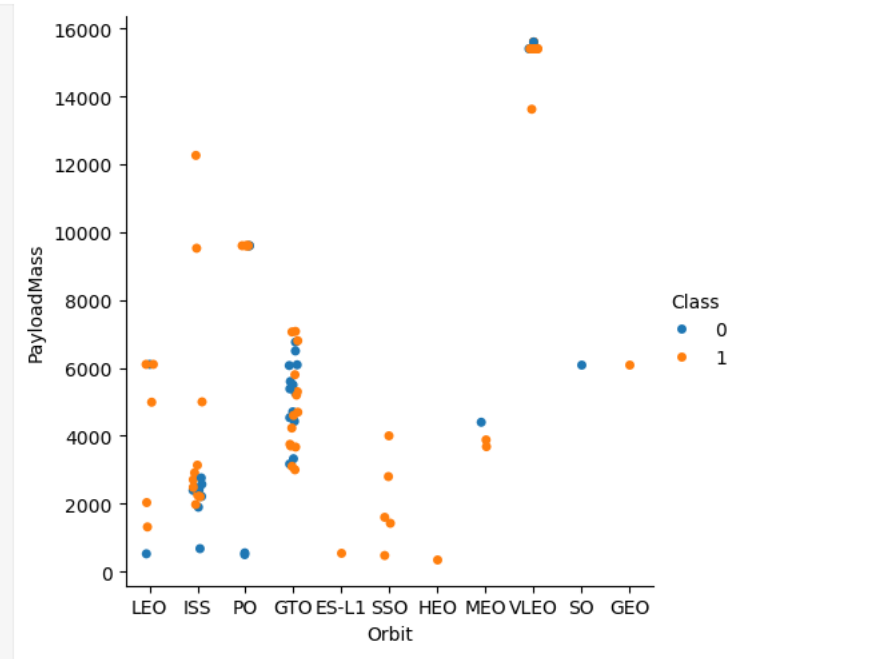


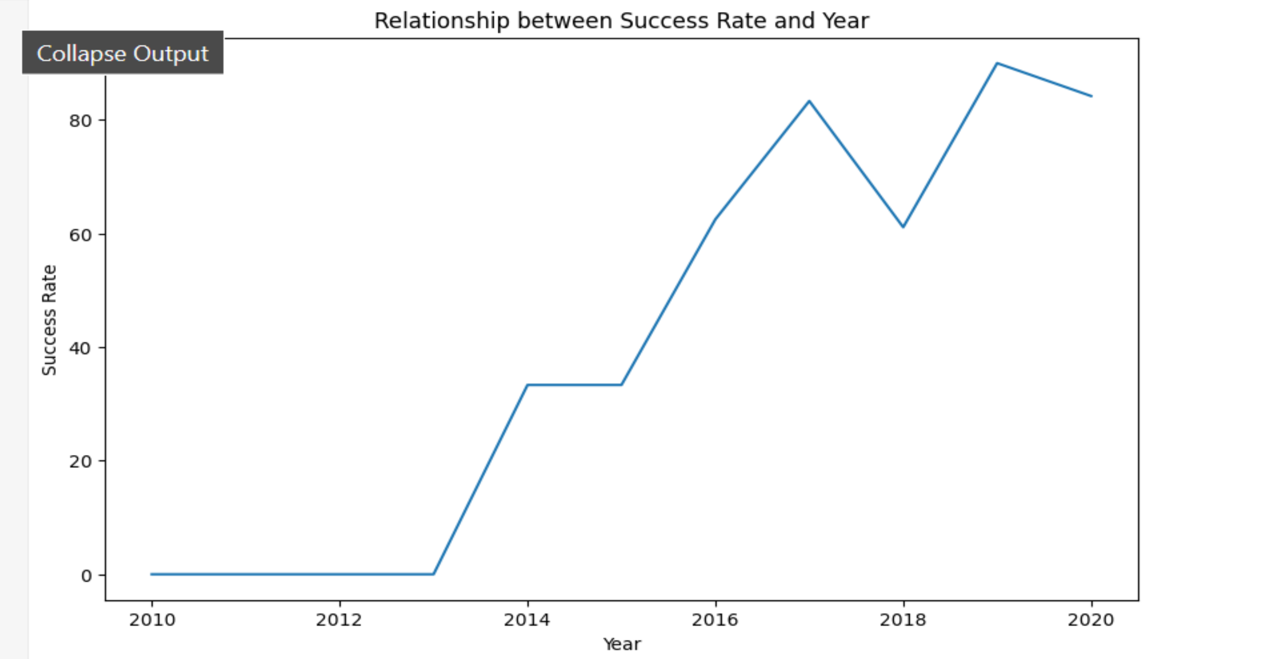
```python
### TASK  3: Visualize the relationship between success rate of each orbit type
sns.catplot(y="Orbit", x="Outcome", hue="Class", data=df, aspect = 5)
plt.xlabel("Outcome",fontsize=20)
plt.ylabel("Orbit",fontsize=20)
plt.show()
```









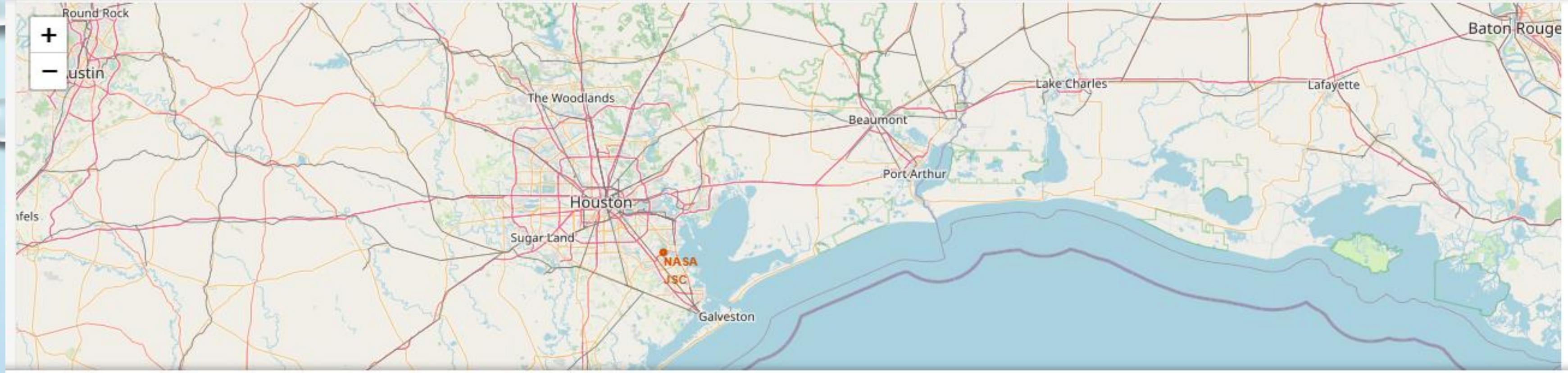


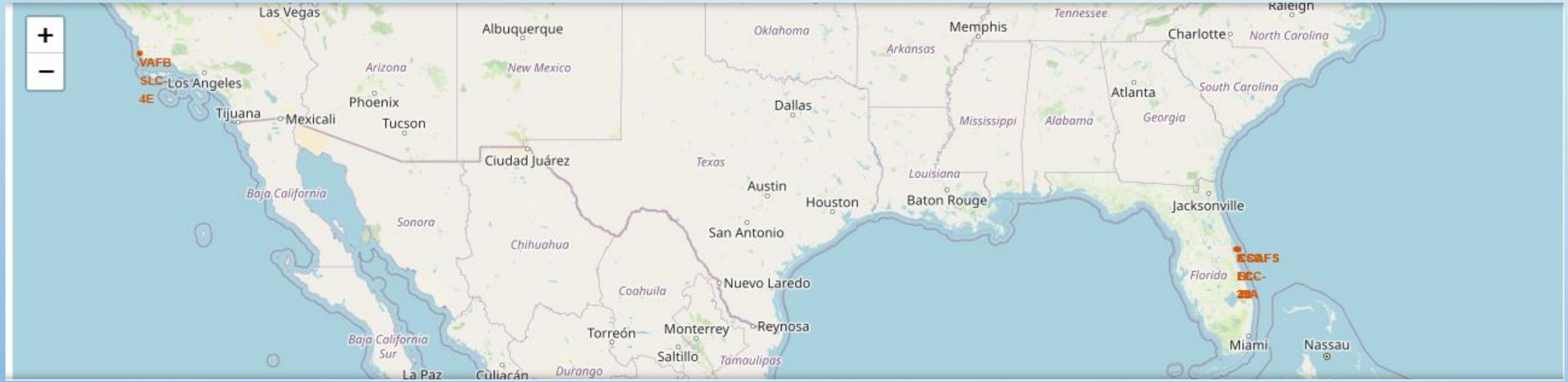
[28]:

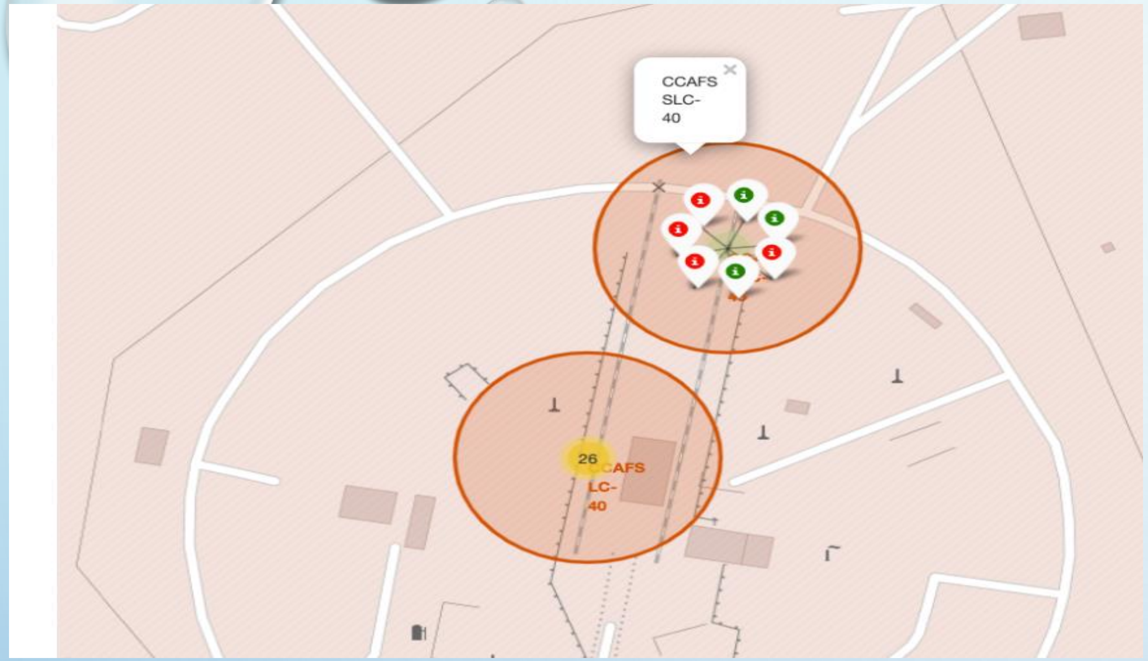
	FlightNumber	PayloadMass	Flights	GridFins	Reused	Legs	Block	ReusedCount	Orbits_ES-L1	Orbits_GEO	...	Serial_B1048	Serial_B1049	Serial_B1050	Serial_B1051	Serial_B1054	Serial_B1056	Seri
0	1.0	6104.959412	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	
1	2.0	525.000000	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	
2	3.0	677.000000	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	
3	4.0	500.000000	1.0	0.0	0.0	0.0	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	0.0	0.0	0.0	1.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
85	86.0	15400.000000	2.0	1.0	1.0	1.0	5.0	2.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	
86	87.0	15400.000000	3.0	1.0	1.0	1.0	5.0	2.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	
87	88.0	15400.000000	6.0	1.0	1.0	1.0	5.0	5.0	0.0	0.0	...	0.0	0.0	0.0	1.0	0.0	0.0	
88	89.0	15400.000000	3.0	1.0	1.0	1.0	5.0	2.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	
89	90.0	3681.000000	1.0	1.0	0.0	1.0	5.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	

90 rows × 80 columns

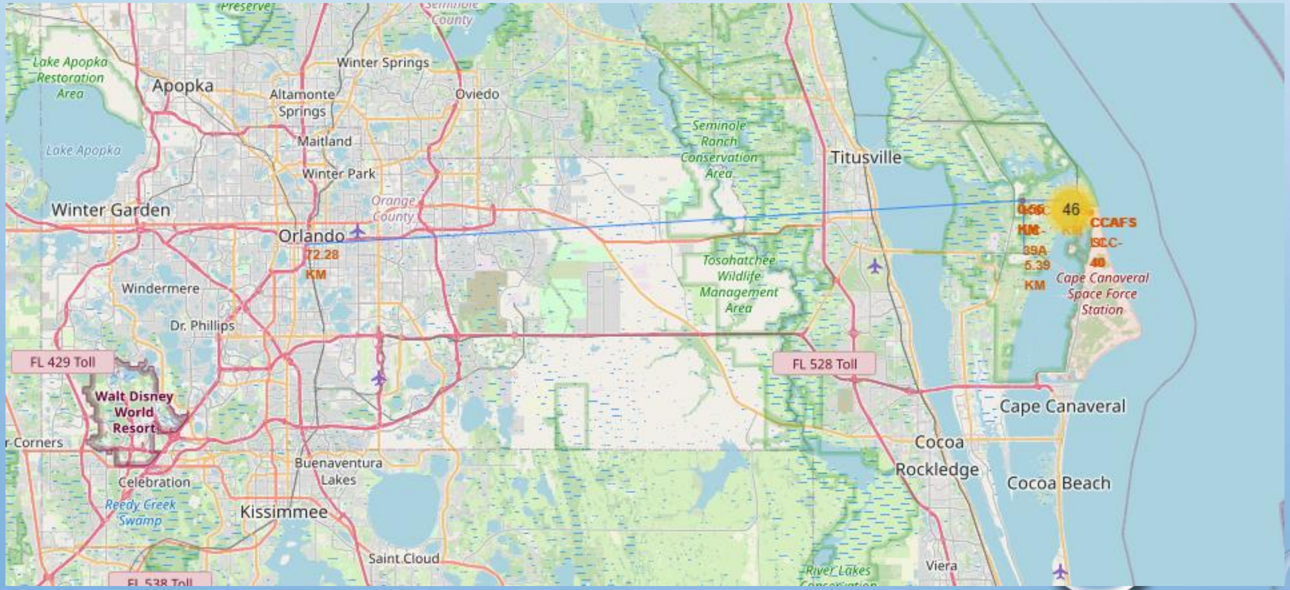
Launch Sites Locations Analysis with Folium






CCAFS
SLC-
40
26
LC-
40


P
P
P
7
SLC-
40
26
LC-
40
0.90
KM
Centaur Raod
Samuel C Phillips Pkwy


Lake Apopka Restoration Area
Apopka
Lake Apopka
Winter Springs
Altamonte Springs
Oviedo
Maitland
Winter Park
Seminole Ranch Conservation Area
Titusville
Winter Garden
Orange County
Orlando
72.28
KM
46
CCAFS
SLC-
40
39A
5.39
KM
Tosohatchee Wildlife Management Area
Cape Canaveral Space Force Station
Windermere
Dr. Phillips
FL 429 Toll
FL 528 Toll
Walt Disney World Resort
Cape Canaveral
Cocoa
Rockledge
Cocoa Beach
Celebration
Reedy Creek Swamp
Buenaventura Lakes
Kissimmee
Saint Cloud
Viera

Dashboard in Plotly

```python
app.layout = html.Div(children=[html.H1('SpaceX Launch Records Dashboard',
                                        style={'textAlign': 'center', 'color': '#503D36',
                                               'font-size': 40}),
                                # TASK 1: Add a dropdown list to enable Launch Site selection
                                # The default select value is for ALL sites
                                # dcc.Dropdown(id='site-dropdown',...)
                                dcc.Dropdown(id='site-dropdown',
                                            options=[
                                                        {'label': 'ALL SITES', 'value': 'ALL'},
                                                        {'label': 'CCAFS LC-40', 'value': 'CCAFS LC-40'},
                                                        {'label': 'VAFB SLC-4E', 'value': 'VAFB SLC-4E'},
                                                        {'label': 'KSC LC-39A', 'value': 'KSC LC-39A'},
                                                        {'label': 'CCAFS SLC-40', 'value': 'CCAFS SLC-40'}
                                                    ],
                                            value='ALL',
                                            placeholder="Select a Launch Site here",
                                            searchable=True),
                                html.Br(),

                                # TASK 2: Add a pie chart to show the total successful launches count for all sites
                                # If a specific launch site was selected, show the Success vs. Failed counts for the site
                                html.Div(dcc.Graph(id='success-pie-chart')),
                                html.Br(),

                                html.P("Payload range (Kg):"),
                                # TASK 3: Add a slider to select payload range
                                #dcc.RangeSlider(id='payload-slider',...)
                                dcc.RangeSlider(id='payload-slider',
                                                min=0,max=10000,step=1000,
                                                value=[min_payload,max_payload],
                                                marks={0: '0', 2500:'2500',5000:'5000',
                                                7500:'7500', 10000: '10000'}),
```

```python
# TASK 2:
# Add a callback function for `site-dropdown` as input, `success-pie-chart` as output
@app.callback(
    Output(component_id='success-pie-chart', component_property='figure'),
    Input(component_id='site-dropdown', component_property='value'))

def build_graph(site_dropdown):
    if site_dropdown == 'ALL':
        piechart = px.pie(data_frame = spacex_df, names='Launch Site', values='class' ,title='Total Launches for All Sites
        return piechart
    else:
        #specific_df = spacex_df['Launch Site']
        specific_df=spacex_df.loc[spacex_df['Launch Site'] == site_dropdown]
        piechart = px.pie(data_frame = specific_df, names='class',title='Total Launch for a Specific Site')
        return piechart

# TASK 4:
# Add a callback function for `site-dropdown` and `payload-slider` as inputs, `success-payload-scatter-chart` as output
@app.callback(
    Output(component_id='success-payload-scatter-chart', component_property='figure'),
    [Input(component_id='site-dropdown', component_property='value'),
    Input(component_id='payload-slider', component_property='value')])

def update_graph(site_dropdown, payload_slider):
    if site_dropdown == 'ALL':
        filtered_data = spacex_df[(spacex_df['Payload Mass (kg)']>=payload_slider[0])
        &(spacex_df['Payload Mass (kg)']<=payload_slider[1])]
        scatterplot = px.scatter(data_frame=filtered_data, x="Payload Mass (kg)", y="class",
        color="Booster Version Category")
        return scatterplot
```

Prediction using Machine Learning

Logistic Regression

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`.

11]:
```
parameters ={'C':[0.01,0.1,1],
             'penalty':['l2'],
             'solver':['lbfgs']}
```

12]:
```
parameters ={"C":[0.01,0.1,1],'penalty':['l2'], 'solver':['lbfgs']}# l1 lasso l2 ridge
lr=LogisticRegression()
```

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_`.

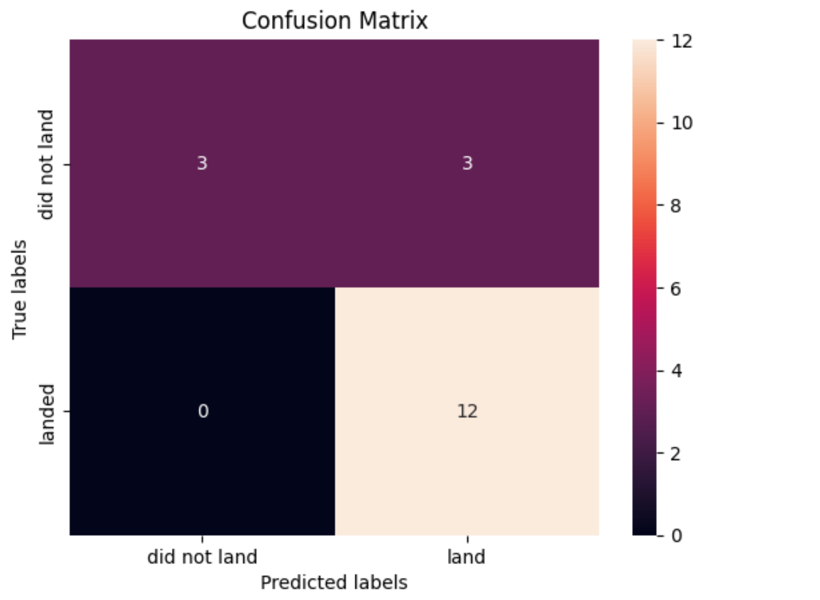
13]:
```
logreg_cv=GridSearchCV(lr,parameters, cv=10)
logreg_cv.fit(X_train,Y_train)
print("tuned hpyerparameters :(best parameters) ",logreg_cv.best_params_)
print("accuracy :",logreg_cv.best_score_)
```

```
accuracy = logreg_cv.score(X_test, Y_test)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.8333333333333334
```

Lets look at the confusion matrix:

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



SVM

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

```
svm_cv=GridSearchCV(svm, parameters, cv=10)
svm_cv.fit(X_train,Y_train)
```

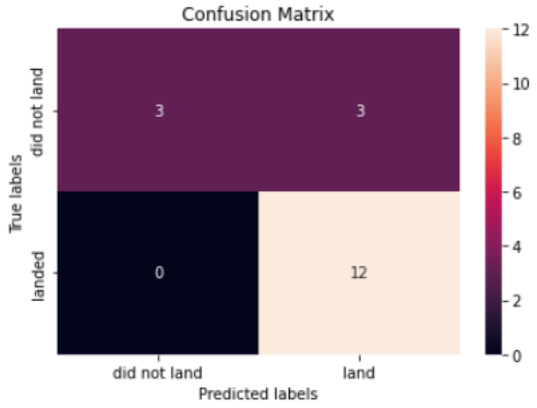
```
GridSearchCV(cv=10, estimator=SVC(),
             param_grid={'C': array([1.00000000e-03, 3.16227766e-02, 1.00000000e+00, 3.16227766e+01,
       1.00000000e+03]),
                         'gamma': array([1.00000000e-03, 3.16227766e-02, 1.00000000e+00, 3.16227766e+01,
       1.00000000e+03]),
                         'kernel': ('linear', 'rbf', 'poly', 'rbf', 'sigmoid')})
```

```
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 : 0.8482142857142858
```

Confusion Matrix for SVM

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



KNN

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`.

```
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()
```

```
knn_cv = GridSearchCV(KNN , parameters , cv = 10)
knn_cv.fit(X_train , Y_train)
```

```
/lib/python3.11/site-packages/threadpoolctl.py:1019: RuntimeWarning: libc not found. The ctypes module in Python 3.11 is maybe too old for this OS.
  warnings.warn(
```

```
▸ GridSearchCV
▸ estimator: KNeighborsClassifier
  ▸ KNeighborsClassifier
```

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

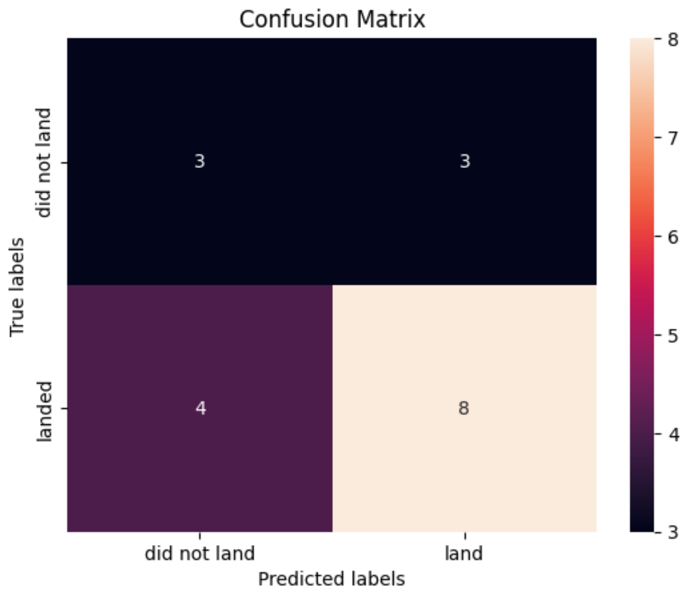
```
tuned hpyerparameters :(best parameters)  {'algorithm': 'auto', 'n_neighbors': 3, 'p': 1}
accuracy : 0.6642857142857143
```

```
accuracy = knn_cv.score(X_test ,Y_test)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.6111111111111112
```

We can plot the confusion matrix

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



Decision Tree Classifier

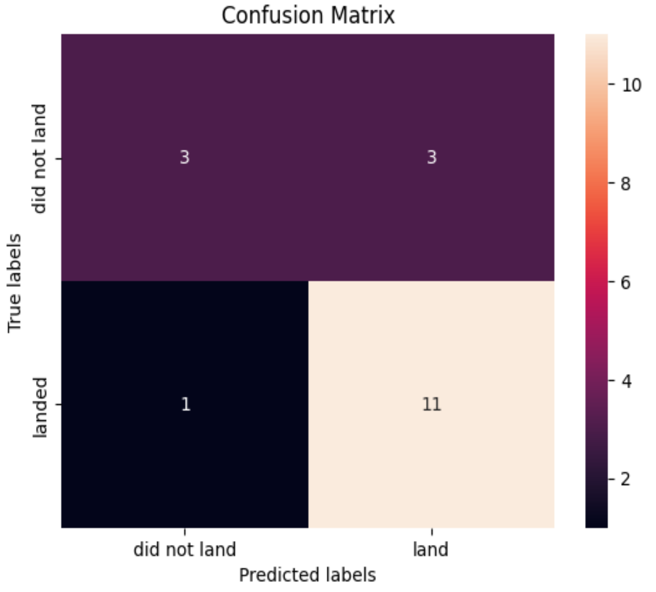
Calculate the accuracy of tree_cv on the test data using the method `score` :

```
accuracy = tree_cv.score(X_test ,Y_test)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.7777777777777778
```

We can plot the confusion matrix

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




```
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()
```

```
tree_cv = GridSearchCV(tree, parameters , cv = 10)
tree_cv.fit(X_train , Y_train)
```

```
The score on these train-test partitions for these parameters will be set to nan.
If these failures are not expected, you can try to debug them by setting error_score='raise'.

Below are more details about the failures:
--------------------------------------------------------------------------------
3240 fits failed with the following error:
Traceback (most recent call last):
  File "/lib/python3.11/site-packages/sklearn/model_selection/_validation.py", line 729, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "/lib/python3.11/site-packages/sklearn/base.py", line 1145, in wrapper
    estimator._validate_params()
  File "/lib/python3.11/site-packages/sklearn/base.py", line 638, in _validate_params
    validate_parameter_constraints(
  File "/lib/python3.11/site-packages/sklearn/utils/_param_validation.py", line 95, in validate_parameter_constraints
    raise InvalidParameterError(
sklearn.utils._param_validation.InvalidParameterError: The 'max_features' parameter of DecisionTreeClassifier must be an int in the range [1, inf), a float in the range (0.0, 1.0], a str among {'log2', 'sqrt'} or None. Got 'auto' instead.
```

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

```
tuned hpyerparameters :(best parameters)  {'criterion': 'entropy', 'max_depth': 10, 'max_features': 'sqrt', 'min_samples_leaf': 1, 'min_samples_split': 2, 'splitter': 'random'}
accuracy : 0.9053571428571429
```

Conclusion:

From the above models we can conclude that the Decision tree classifier gives the best performance compared to the other models.