SPACEX LAUNCH SUCCESS STORY

CH Sweta

21-Jan-2024

OUTLINE

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

EXECUTIVE SUMMARY

- SpaceX is the most successful Rocket launcher
- SpaceX Falcon 9 rocket launch is less expensive to other providers because of the below points
- SpaceX reuses first stage of rocket launch
- First stage is large and expensive unlike other Rocket launchers, SpaceX can reuse its first rocket launch

- Instead of using rocket science to determine if the first stage will land successfully, we have trained a machine learning model and use public information to predict if SpaceX will reuse the first stage.

INTRODUCTION

- Data Collection and Data Wrangling
- Exploratory Data Analyzing
- Interactive Visual Analytics and Dashboard building
- Predictive Analysis
  - Splitting data in tarin and test
  - Applying different models i.e. Logistic regression ,Decision Tree, Support Vector Machine, K-means models

Data Collection and Data Wrangling Methodology

- Developed Python code to manipulate data in a Pandas data frame
- Converted a JSON file into a Create a Python Pandas data frame by converting a JSON file
- Created a Jupyter notebook and made it sharable using GitHub
- Objectives accomplished
  - Request and parse the SpaceX launch data using the GET request
  - Filtered the data frame to only include Falcon 9 launches
  - Relace None values in the Payload Mass with the mean

EDA and interactive visual analytics methodology

- Created scatter plots and bar charts by writing Python code to analyze data in a Pandas data frame
- Written Python code to conduct exploratory data analysis by manipulating data in a Pandas data frame
- Created and executed SQL queries to select and sort data
- Objectives accomplished
  - Visualize the relationship between different parameters
  - Visualize the launch success yearly trend
  - Created dummy variables to categorical columns to standardize the data

EDA and interactive visual analytics methodology

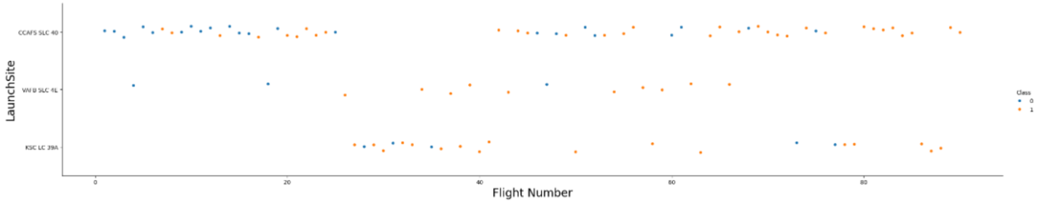
- Built an interactive dashboard that contains pie charts and scatter plots to analyze data with the Plotly Dash Python library
- Calculated distances on an interactive map by writing Python code using the Folium library
- Generated interactive maps, plot coordinates, and mark clusters by writing Python code using the Folium library
- Objectives accomplished
  - Marked all launch sites on a Folium map
  - Marked the success/failed launches for each site
  - Calculated the distances between a launch site to its proximities
  - Added a launch site drop-down input component and added a range slider to choose payload
  - Added a callback function to render the required success outcome pie chart and added callback function to render the payload-outcome scatter plot

Predictive Analysis Methodology

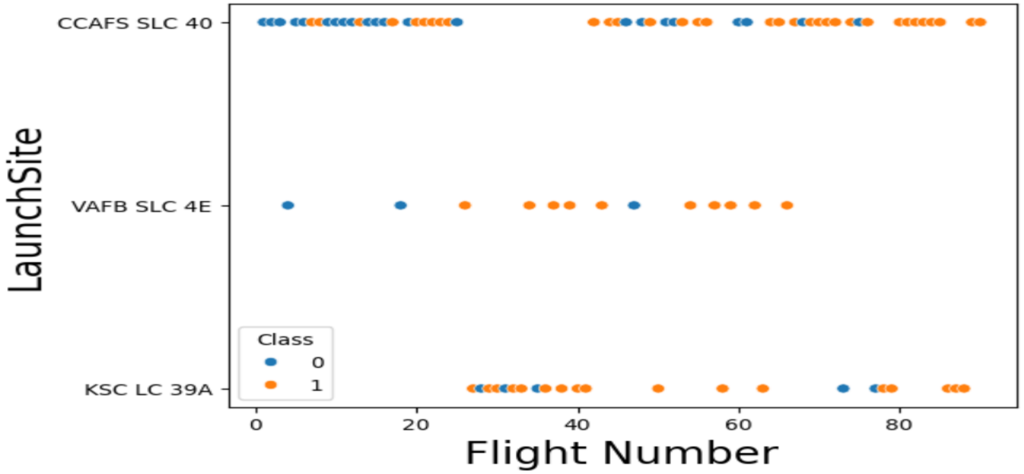
- Split the data into training testing data
- Trained different classification models
- Optimized the Hyperparameter grid search
- Utilized machine learning skills to build a predictive model to help a business function more efficiently
- Objectives accomplished
  - Standardized the data
  - Split the data into train and test set with test size of 20% and random state of 2
  - For each model found the best hyperparameters using GridSearchCV parameters and setting CV value of 10
  - Performed analyzing using Logistic regression, Decision Tree clustering, K means clustering and Support Vector Machine models
  - Calculate the accuracy of the test data by applying the Score function which is derived by fitting train data on above models.
  - Picked Decision Tree clustering model with accuracy of 83% as the best model params to perform predictive analysis on success rate of First stage launch by FALCON9

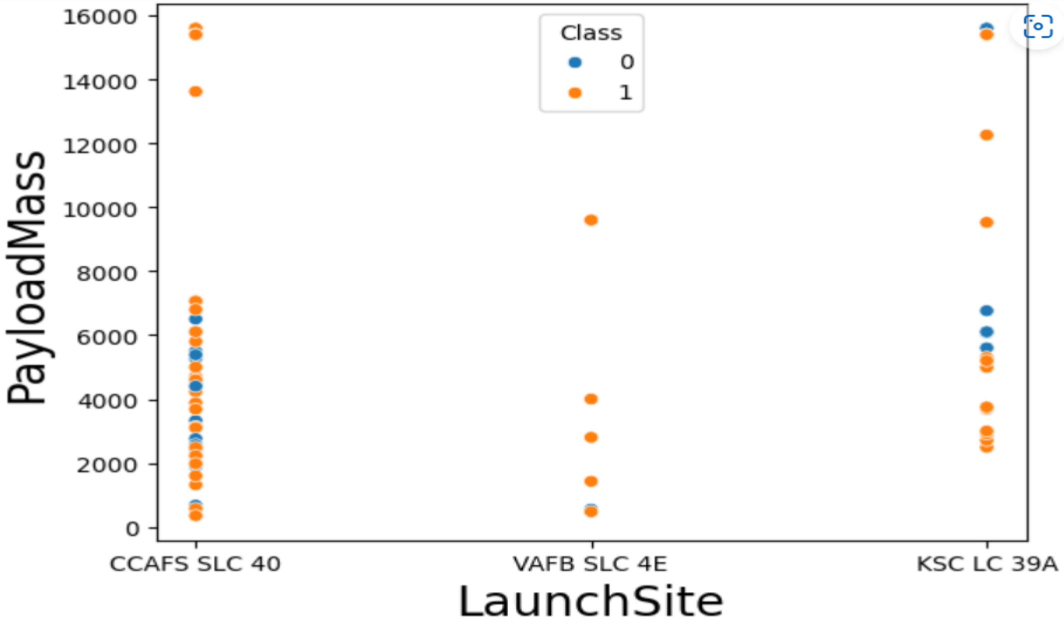
EDA with visualization results

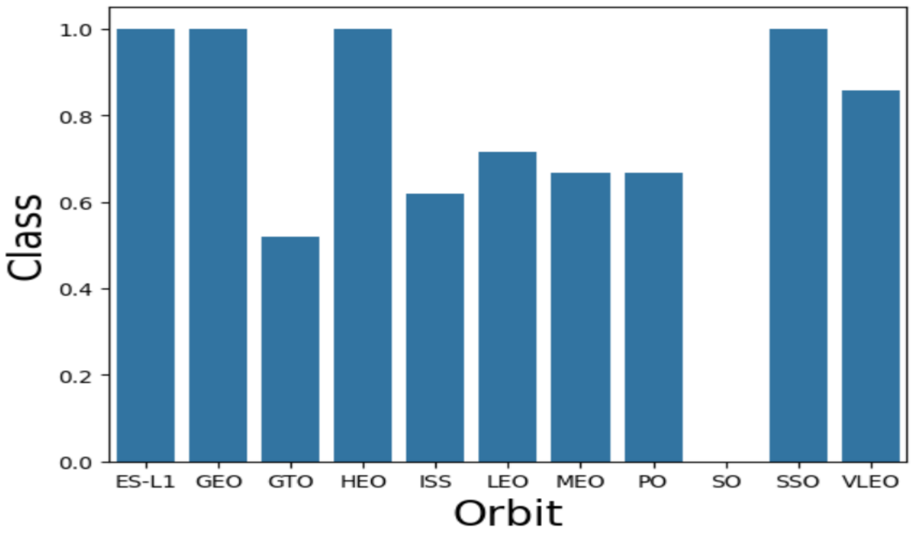
```
[5]: ### TASK 1: Visualize the relationship between Flight Number and Launch Site
sns.catplot(y="LaunchSite", x="FlightNumber", hue="Class", data=df, aspect = 5)
plt.xlabel("Flight Number",fontsize=20)
plt.ylabel("LaunchSite",fontsize=20)
plt.show()
```



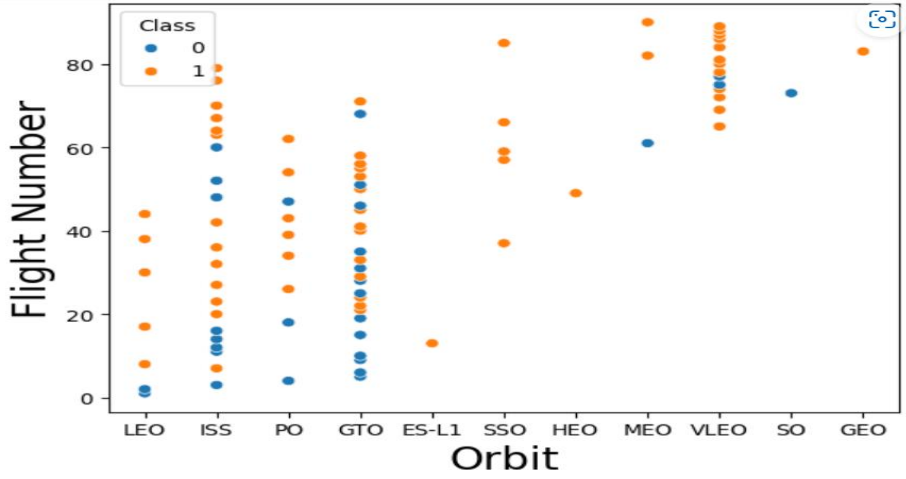
Use the function `catplot` to plot `FlightNumber` vs `LaunchSite`, set the parameter `x` parameter to `FlightNumber`,set the `y` to `Launch Site` and set the parameter `hue` to `'class'`

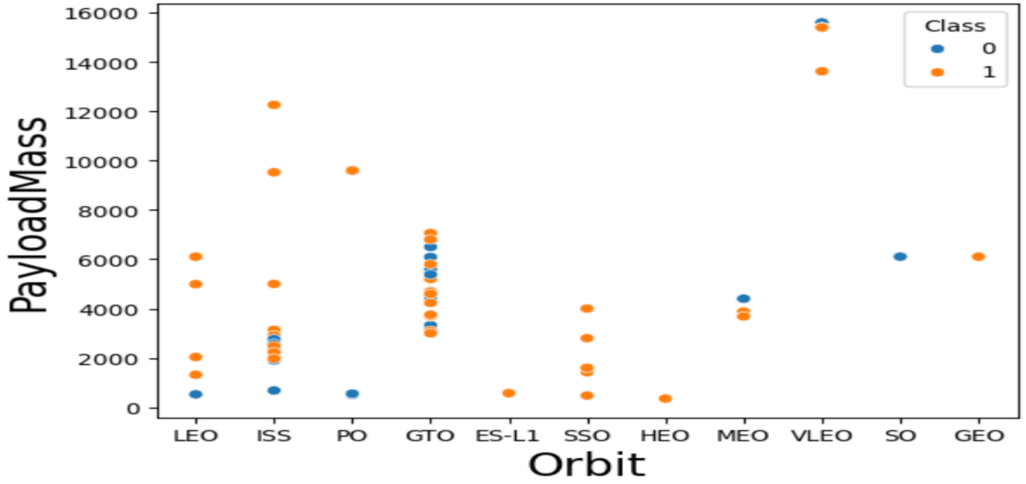


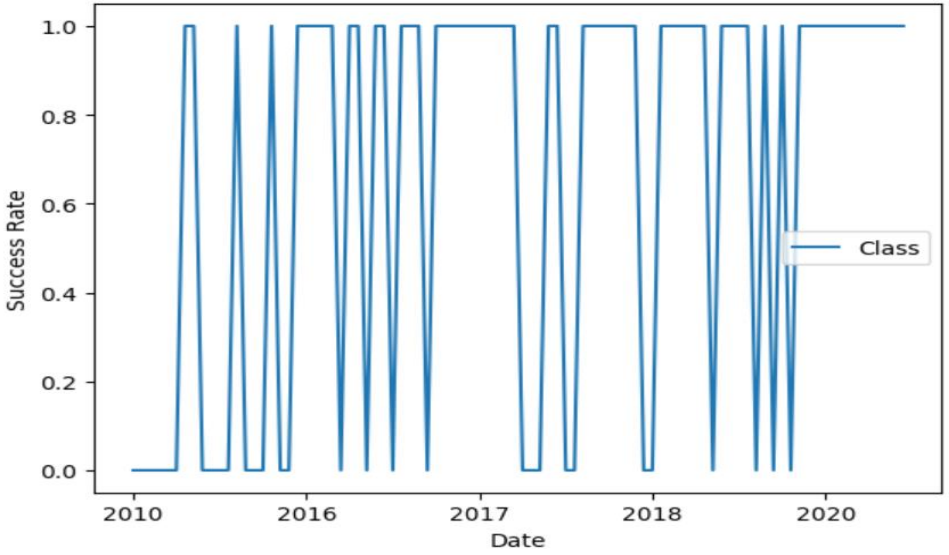




EDA with visualization results








```
dummy_variable_orbits = pd.get_dummies(features['Orbit'])
dummy_variable_LaunchSite = pd.get_dummies(features['LaunchSite'])
dummy_variable_LandingPad = pd.get_dummies(features['LandingPad'])
dummy_variable_Serial = pd.get_dummies(features['Serial'])
features_one_hot = pd.concat([features,dummy_variable_orbits,dummy_variable_LaunchSite,dummy_variable_LandingPad,du
features_one_hot.head()
```

	FlightNumber	PayloadMass	Orbit	LaunchSite	Flights	GridFins	Reused	Legs	LandingPad	Block	...	B1048	B1049	B1
0	1	6104.959412	LEO	CCAFS SLC 40	1	False	False	False	NaN	1.0	...	0	0	
1	2	525.000000	LEO	CCAFS SLC 40	1	False	False	False	NaN	1.0	...	0	0	
2	3	677.000000	ISS	CCAFS SLC 40	1	False	False	False	NaN	1.0	...	0	0	
3	4	500.000000	PO	VAFB SLC 4E	1	False	False	False	NaN	1.0	...	0	0	
4	5	3170.000000	GTO	CCAFS SLC 40	1	False	False	False	NaN	1.0	...	0	0	

5 rows × 84 columns

EDA with SQL results

Task 1

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

[8]:
```
%sql select distinct Launch_Site from SPACEXTABLE
```

 * sqlite:///my_data1.db
Done.

[8]:

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

Task 2

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

[9]:
```
%sql select Launch_Site from SPACEXTABLE where Launch_Site like 'CCA%' limit 5
```

 * sqlite:///my_data1.db
Done.

[9]:

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

Task 3

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

[10]:
```
%sql select Booster_Version,sum(PAYLOAD_MASS__KG_) as Total_Payload_Mass from SPACEXTABLE group by Booster
```

 * sqlite:///my_data1.db
Done.

[10]:

Booster_Version	Total_Payload_Mass
F9 B4 B1039.2	2647
F9 B4 B1040.2	5384
F9 B4 B1041.2	9600
F9 B4 B1043.2	6460
F9 B4 B1039.1	3310
F9 B4 B1040.1	4990

Task 4

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

```
%sql select Booster_Version,avg(PAYLOAD_MASS__KG_) as average_Payload_Mass from SPACEXTABLE group by Boost
```

 * sqlite:///my_data1.db
Done.

Booster_Version	average_Payload_Mass
F9 v1.1	2928.4

EDA with SQL results

Task 5

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

Hint:Use min function

```
%sql select min(date) from SPACEXTABLE where Landing_Outcome like 'Success%ground%'
```

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

min(date)
2015-12-22

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

```
.3]: %sql select distinct Booster_Version from SPACEXTABLE where Landing_Outcome like 'Success%drone ship%' an
```

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

.3]:

Booster_Version
F9 FT B1022
F9 FT B1026
F9 FT B1021.2
F9 FT B1031.2

Task 7

List the total number of successful and failure mission outcomes

```
[14]: %sql select Mission_Outcome,count(*) total from SPACEXTABLE group by Mission_Outcome
```

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

[14]:

Mission_Outcome	total
Failure (in flight)	1
Success	98
Success	1
Success (payload status unclear)	1

Task 8

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

```
[22]: %sql select Booster_Version from SPACEXTABLE where PAYLOAD_MASS__KG_ = (select max(PAYLOAD_MASS__KG_) from
```

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

[22]:

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

EDA with SQL results

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.

Note: SQLLite does not support monthnames. So you need to use substr(Date, 6,2) as month to get the months and substr(Date,0,5)='2015' for year.

```
:3]: ,Launch_Site from SPACEXTABLE where substr(Date,0,5)='2015' and Landing_Outcome like 'Failure%drone ship%'
```

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

:3]:

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

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.

```
[24]: tcome,count(*) from SPACEXTABLE group by Landing_Outcome having Date between '2010=06-04' and '2017-03-20'
```

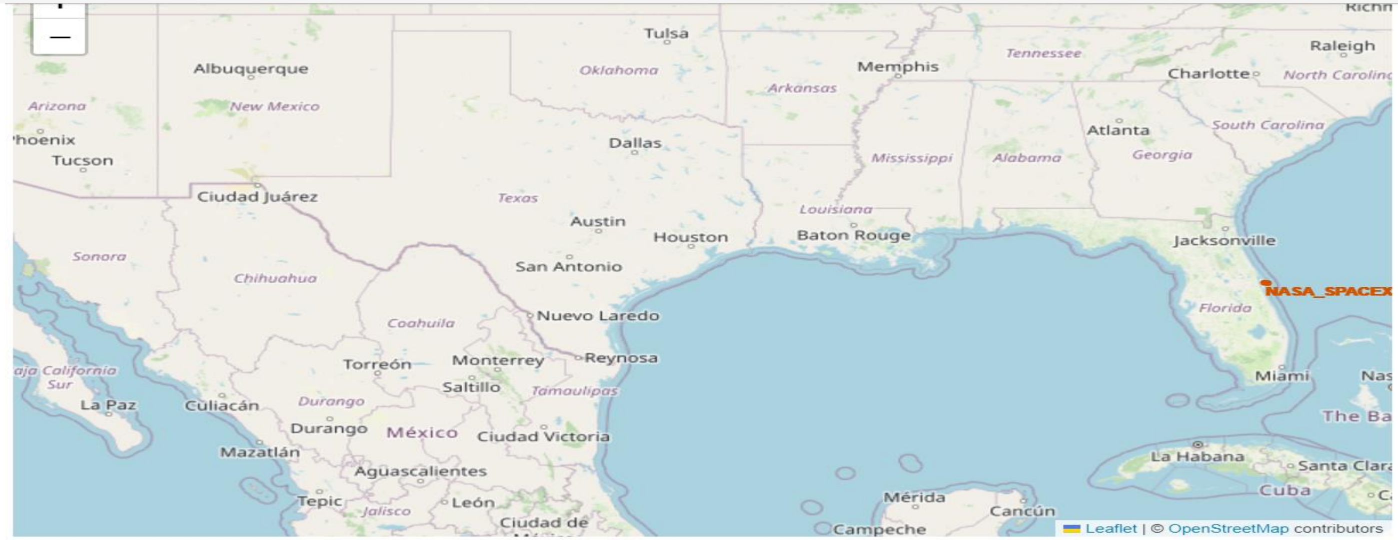
```
 * sqlite:///my_data1.db
Done.
```

[24]:

Landing_Outcome	count(*)
Controlled (ocean)	5
Failure (drone ship)	5
No attempt	21
Precluded (drone ship)	1
Success (drone ship)	14
Success (ground pad)	9
Uncontrolled (ocean)	2

Interactive map with Folium results

```python
# 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 si

incidents = folium.map.FeatureGroup()
for lat,lng in zip(launch_sites_df.Lat,launch_sites_df.Long):
    Circle = folium.Circle([lat,lng], radius = 1000,color = '#d35400', fill =True).add_child(folium.Popup("NASA Joh
    Marker = folium.map.Marker([lat,lng], icon=DivIcon(icon_size=(20,20),icon_anchor=(0,0), html='<div style="font-
    site_map.add_child(Circle)
    site_map.add_child(Marker)
site_map
```



Interactive map with Folium results

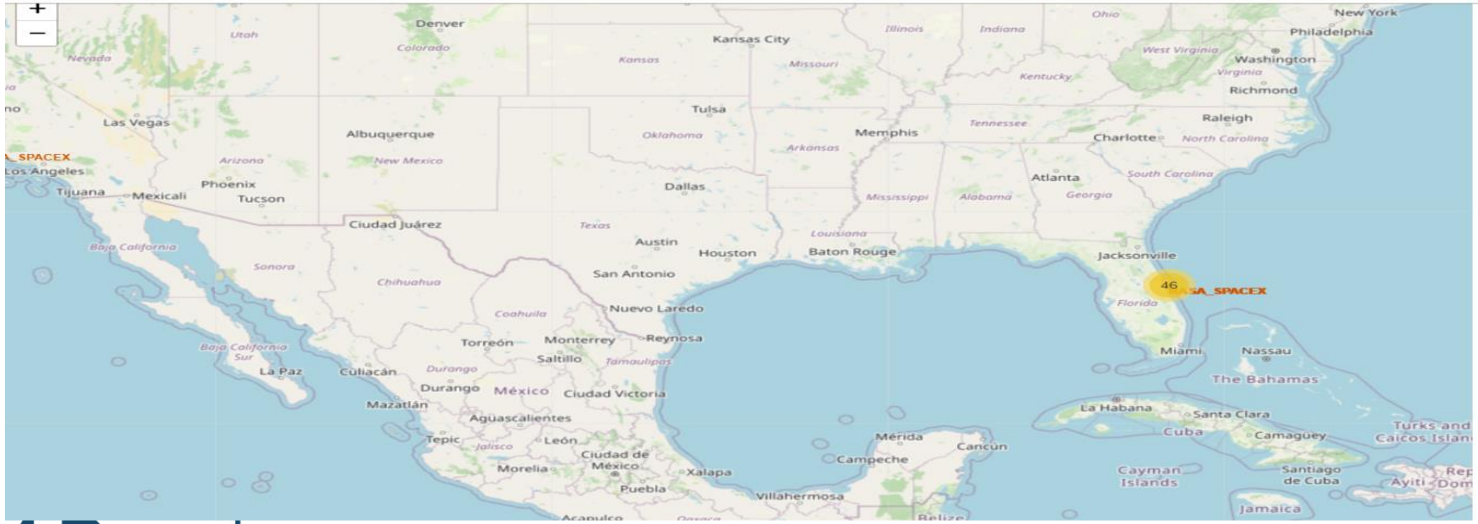
```
[21]: marker_cluster = MarkerCluster()
```

TODO: Create a new column in `launch_sites` dataframe called `marker_color` to store the marker colors based on the `class` value

```
[41]: # 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
marker_color = []
for i in spacex_df['class']:
    if i == 0:
        marker_color.append('red')
    else:
        marker_color.append("green")
spacex_df['marker_color']=marker_color
```

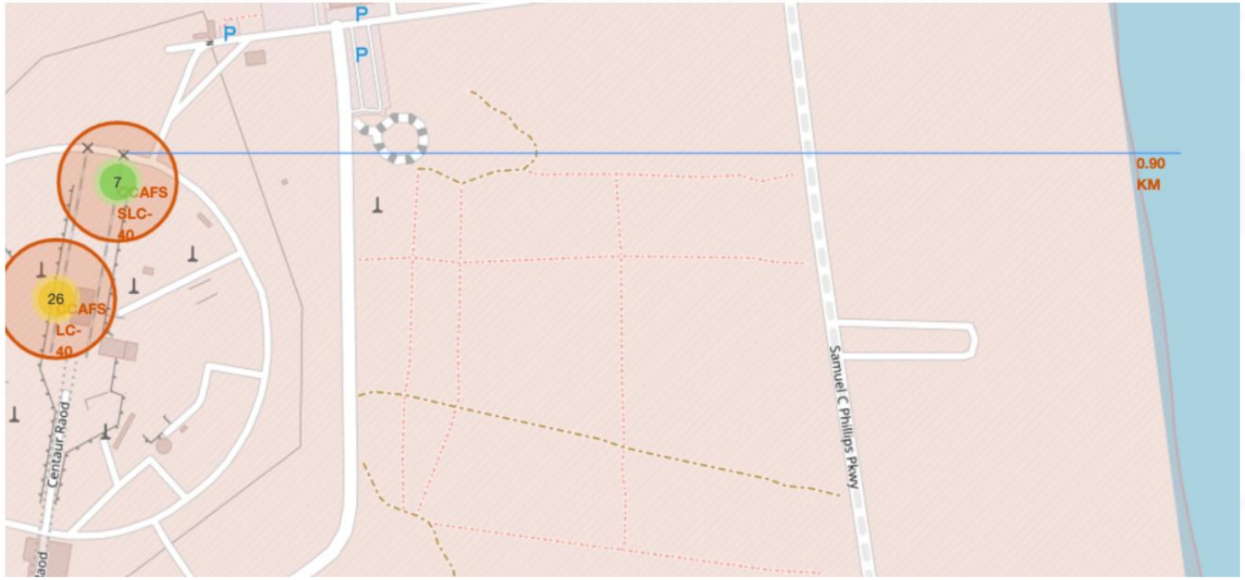
TODO: For each launch result in `spacex_df` data frame, add a `folium.Marker` to `marker_cluster`

```
[47]: # 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 lat,lng,c in zip(spacex_df.Lat,spacex_df.Long,spacex_df.marker_color):
    # TODO: Create and add a Marker cluster to the site map
    marker = folium.map.Marker(
        [lat,lng], icon=folium.Icon(color='white', icon_color=c)
    )
    marker_cluster.add_child(marker)

site_map.add_child(marker_cluster)
```



Interactive map with Folium results

```
# Add Mouse Position to get the coordinate (Lat, Long) for a mouse over on the map
formatter = "function(num) {return L.Util.formatNum(num, 5);};"
mouse_position = MousePosition(
    position='topright',
    separator=' Long: ',
    empty_string='NaN',
    lng_first=False,
    num_digits=20,
    prefix='Lat:',
    lat_formatter=formatter,
    lng_formatter=formatter,
)

site_map.add_child(mouse_position)
site_map
```



Plotly Dash dashboard results

SpaceX Launch Records Dashboard

All Sites

All Sites

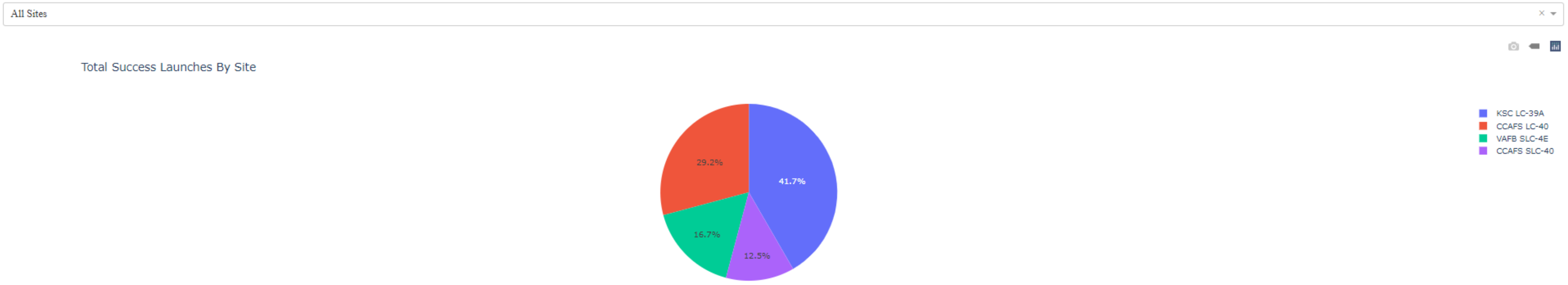
CCAFS LC-40

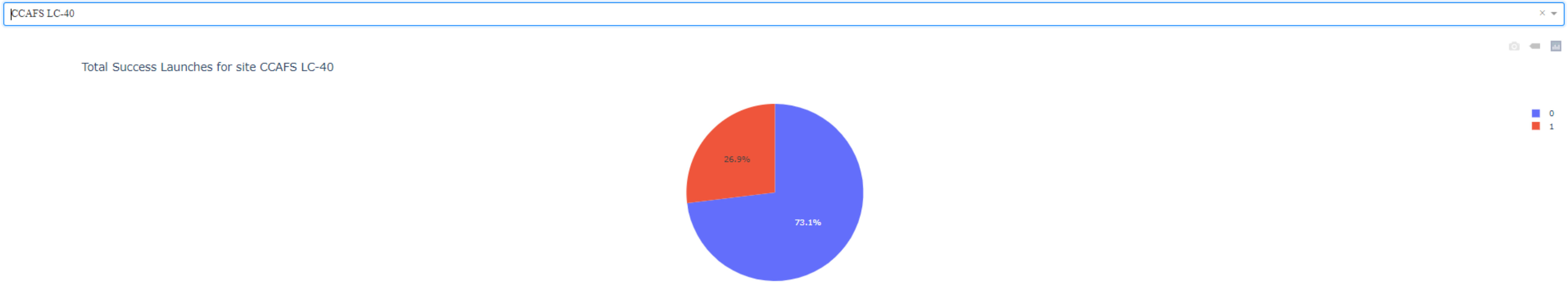
VAFB SLC-4E

KSC LC-39A

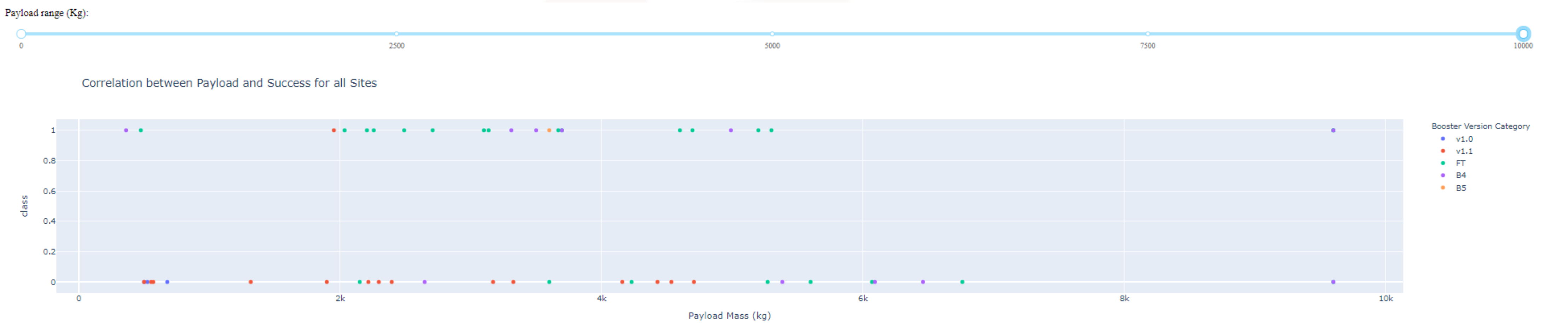
CCAFS SLC-40

Plotly Dash dashboard results

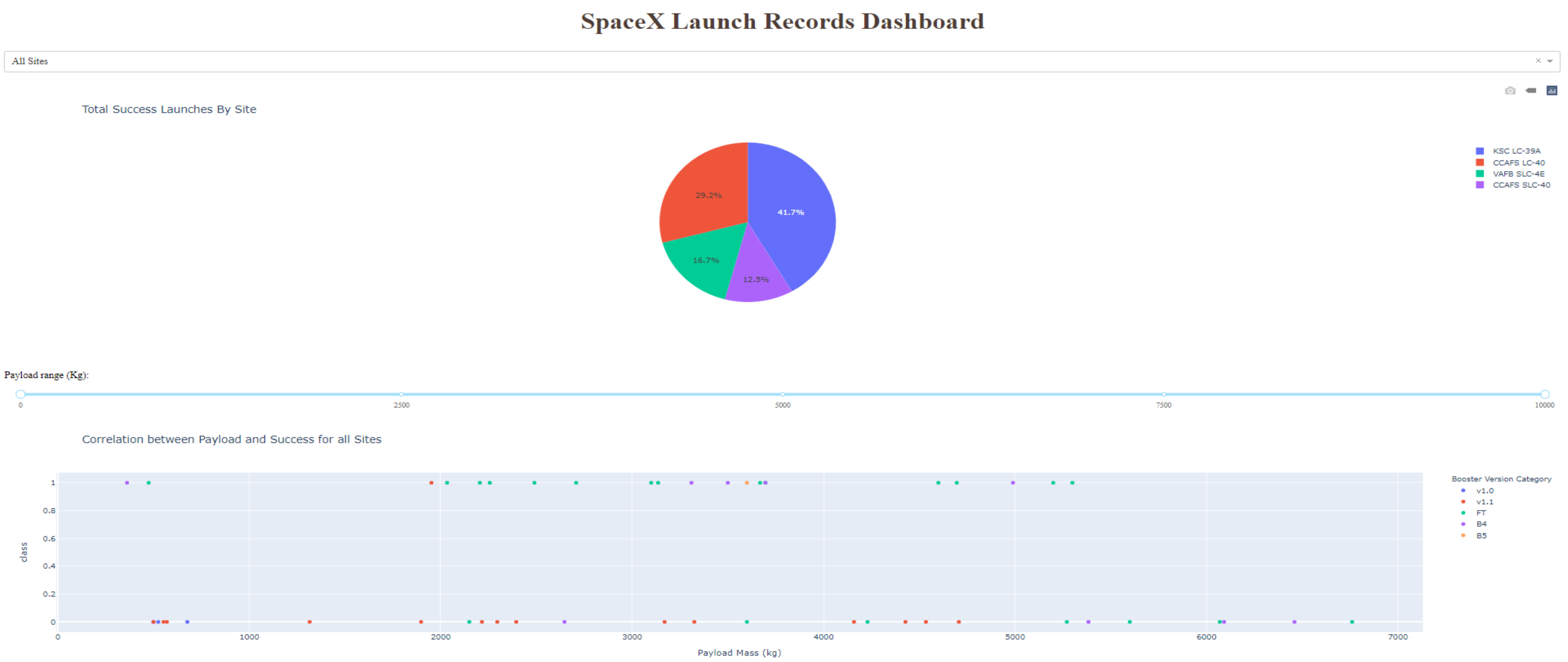




Plotly Dash dashboard results



Plotly Dash dashboard results



Predictive analysis results

[9]:
```
Y = data['Class'].to_numpy()
```

TASK 2

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

[10]:
```
# students get this
transform = preprocessing.StandardScaler()
```

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`

[11]:
```
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.

[12]:
```
Y_test.shape
```

[12]:
```
(18,)
```

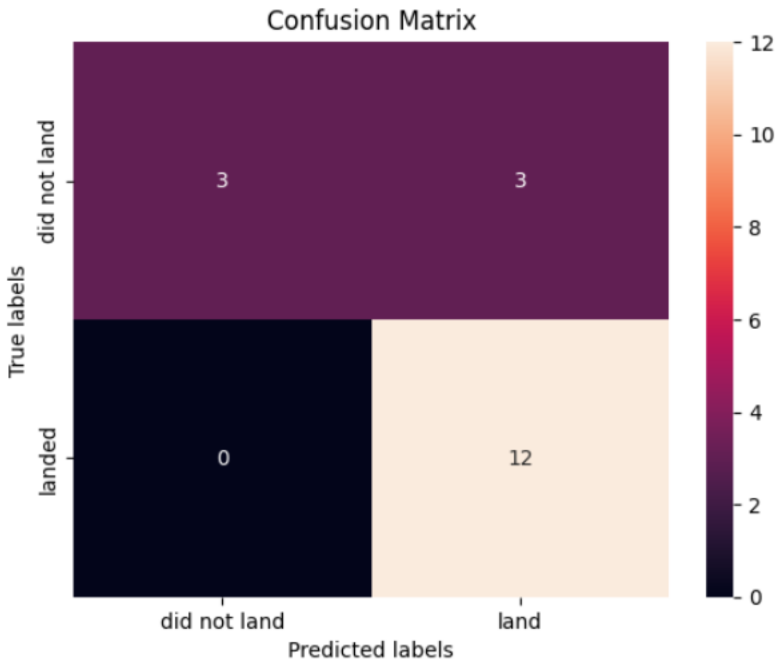
Predictive analysis results

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`.

[13]:
```
parameters ={'C':[0.01,0.1,1],
             'penalty':['l2'],
             'solver':['lbfgs']}
```

[19]:
```
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)
```



[20]:
```
print("tuned hpyerparameters :(best parameters) ",logreg_cv.best_params_)
print("accuracy :",logreg_cv.best_score_)
```

```
tuned hpyerparameters :(best parameters)  {'C': 0.1, 'penalty': 'l2', 'solver': 'lbfgs'}
accuracy : 0.8196428571428571
```

TASK 5

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

[21]:
```
logreg_cv.score(X_test,Y_test)
```

[21]: 0.8333333333333334

Lets look at the confusion matrix:

[22]:
```
yhat=logreg_cv.predict(X_test)
plot_confusion_matrix(Y_test,yhat)
```

Predictive analysis results

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`.

```
32]: 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)
```

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

TASK 7

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

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

We can plot the confusion matrix

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

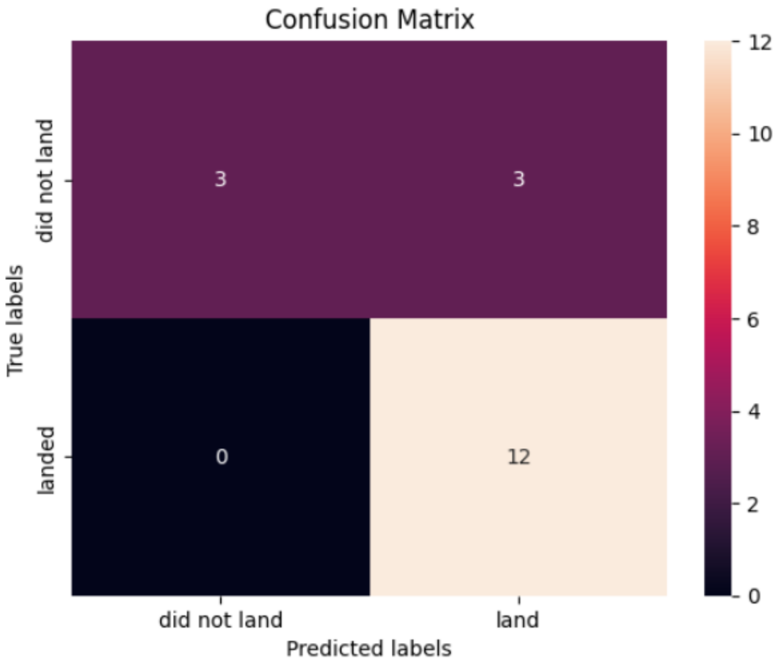
Predictive analysis results

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`.

17]:
```
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()
```

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




```
 0.78928571 0.83392857 0.83392857 0.79107143 0.73928571 0.83392857
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
 0.77857143 0.80357143 0.74821429 0.78928571 0.79107143 0.80535714
 0.75       0.79285714 0.75178571 0.77678571 0.75       0.79107143
 0.77321429 0.74642857 0.81785714 0.78928571 0.71785714 0.80535714]
  warnings.warn(
```

21]:
```
GridSearchCV
estimator: DecisionTreeClassifier
DecisionTreeClassifier
```

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

```
tuned hpyerparameters :(best parameters)  {'criterion': 'gini', 'max_depth': 12, 'max_features': 'sqrt', 'min_samples_leaf': 4, 'min_samples_split': 2, 'splitter': 'random'}
accuracy : 0.8625
```

TASK 9

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

23]:
```
tree_cv.score(X_test,Y_test)
```

23]: 0.8333333333333334

We can plot the confusion matrix

24]:
```
yhat = tree_cv.predict(X_test)
```

Predictive analysis results

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`.

```
[27]: 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()
```

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

```
[28]: GridSearchCV
      estimator: KNeighborsClassifier
      KNeighborsClassifier
```

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

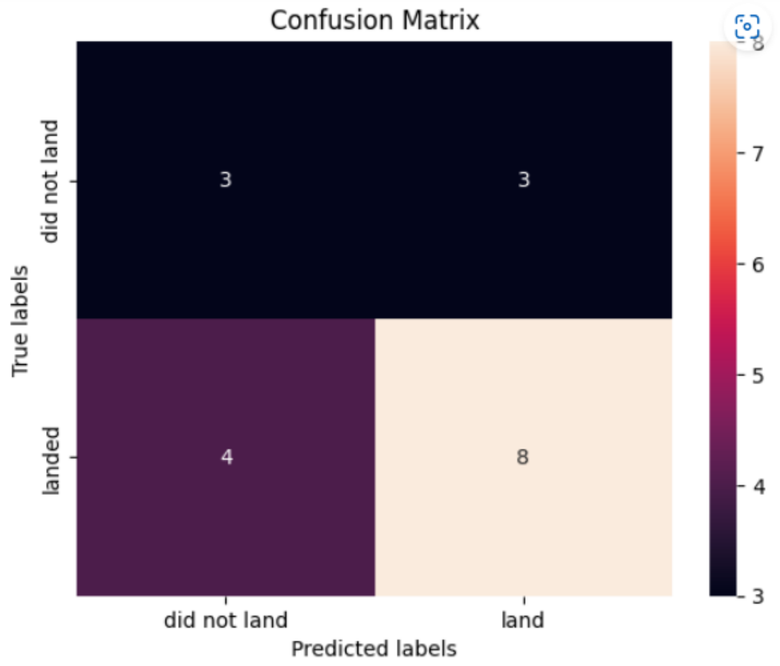
```
tuned hpyerparameters :(best parameters)  {'algorithm': 'auto', 'n_neighbors': 3, 'p': 1}
accuracy : 0.6642857142857143
```

TASK 11

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

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

```
[30]: 0.6111111111111112
```



CONCLUSION

- We have successfully collected the data from SPACEX REST API and performed data wrangling by removing nulls in required columns
- We have performed Exploratory data analysis by connecting to SQL lite server and performed different data operations on SPACEX TABLE
- Using Folium & Map library functions we have successfully plotted the geographical locations of launching sites, clubbed all the launching sites using Marker cluster and using Mouse option we have successfully displayed the distance of city Railway station from Launch Site
- Using Dash and Plotly Library of Python we are build the web analytical dashboard that contains a ‘success-pie-chart’ built by the input taken on Launch Site dropdown and added a Range slider for Payload mass and then displayed the Payload mass and Launch site dependency on success rate by scatterplot
- We have split the data into Train and Test and then applied different regression models by adjusting the hyper parameters and found the best params that displays the best score for each model and found the best model that fits on the train data and predicts the launch of FALCON 9 First Stage

Findings and Insights

- 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
- 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%
- AFB-SLC launch site there are no rockets launched for heavy payload mass(greater than 10000)
- LEO orbit the Success appears related to the number of flights; on the other hand, there seems to be no relationship between flight number when in GTO orbit
- With heavy payloads the successful landing or positive landing rate are more for Polar, LEO and ISS.
- However, for GTO we cannot distinguish this well as both positive landing rate and negative landing(unsuccessful mission) are both there
- Observed that the success rate since 2013 kept increasing till 2020
- Examining the confusion matrix, we see that logistic regression can distinguish between the different classes. We see that the major problem is false positives.
- With an accuracy of 83.33% both Decision Tree and logistic regression models with cv=10 are the best performing models