SpaceX Data Exploration And Predictions

Ejale Emmanuel Osobase

September 2022

OUTLINE

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

EXECUTIVE SUMMARY

- Summary of Methodology
  - Data collection with API and Web Scraping
  - Data Wrangling
  - Exploratory Data Analysis with SQL
  - Exploratory Data Analysis with Data Visualization
  - Interactive Visual Analysis with Folium
  - Interactive Visual Analysis with Plotly Dash
- Predictive Analysis
  - Machine Learning
- Result Summary
  - Exploratory Data Analysis
  - Visuals
  - Predictive Analysis

INTRODUCTION

- Project Scenario Overview

SpaceX Falcon 9 launches with a cost of 62 million dollars while other providers cost upwards of 165 million dollars each, this is due to SpaceX being able to reuse the first stage of the rocket launched. Therefore, if we can predict successful landing of the first stage we can determine the cost of a launch.

- Questions
    - Determine if SpaceX will reuse the first atage
    - Determine the price of each Launch
    - Train ML models and use public information to predict if SpaceX will reuse the first stage

METHODOLOGY

- Data Collection:
  - Using SpaceX API
  - Web Scraping from Wikipedia with Beautiful Soup
- Data Wrangling
  - Exploratory Data Analysis (EDA) with SQL and Visuals
  - Interactive visual analysis with folium and Plotly Dash
- Predictive Analysis:
  - Train classification models for predictions
  - Using folium world map to calculate proximity to city, coastline etc.

Data Collection- SpaceX API

- The data was collected with get request from SpaceX API
- Use the API to get information about the launches
- Apply accurate names to each columns
- Combine the columns into a dictionary
- Create a pandas data frame from the dictionary
- Filter the data frame to only include falcon 9 launches

Data Collection- Web Scraping

- Request the Falcon 9 launch Wikipedia page from its URL
- Extract Falcon 9 records HTML tables from Wikipedia with BeautifulSoup
- Parse the table and convert it into a Pandas Data frame

Data Wrangling

- Remove null values
- Identify numerical and categorical columns
- Late the number of launches on each site
- Calculate the number of occurrences of each orbit
- Calculate occurrences of mission outcome per orbit type
- Create a landing Outcome column

EDA with SQL Analysis

- Space X dataset is Loaded and formatted within IBM Watson DB2 database and explored using jupyter notebook.
  - Exploration caried out using SQL to derive insights. These queries include:
    1. Names of unique Launch sites in the space mission
    2. Display 5 records where launch sites begin with the string 'CCA'
    3. Display the total payload mass carried by booster launched by NASA (CRS)
    4. Display average payload mass carried by booster version F9 v1.1
    5. List the date when the first successful landing outcome in ground pad was achieved

Link to the Notebook: https://github.com/EmmaOsobase/coursera-Capstone-Project/blob/main/jupyter-labs-eda-sql-coursera_sqllite.ipynb

EDA And Interactive Visual Analytics

- Display Unique Launch sites
- Total Mass carried by NASA (CRS)
- Average Payload carried by F9 v1.1
- First successful ground pad landing date
- Names of Booster versions which have carried maximum payload mass.
- How Flight Number and Payload affects the landing outcome
- Relationship between Flight Number and Launch site
- Relationship between Payload and Launch site
- How the Orbit type affects the success rate
- Yearly landing success rate

Plotly Dash dashboard Analysis

- Created an interactive webpage dashboard using plotly dash
- Plot a pie chart for success launches for selected launch sites
- Scattered chart for success count to show how payload mass and class for selected sites, and colour-label points for booster-versions. This chart is to show how payload mass affects the mission outcome for the selected site.

Interactive map with Folium

- Launch Sites Location Analysis using Folium
- Marked all Launch sites on the map using NASA Johnson Space center coordinates as the start location
- Add Circles and Markers to the map to pin point and label launch site using launch sites coordinates.
- Marked successful/failed launch es for each site on the map
- Use marker cluster to highlight successful and failed launches based on colour.
- Calculate launch site distance from various proximities; such as coastline, Railway tracks, city and Highway.

Predictive Analysis

- Import the necessary Libraries
- Create a function to plot the confusion matrix
- Create numpy array from the column class and assign as my dependent variable
- Standardize fit and transform the independent variable
- Split the data to train and test the datasets
- Create classification objects, using LR, SVM, and decision tree, then create a GridSearch and fit to find the best parameter
- Calculate objects accuracy on the test dataset using method score

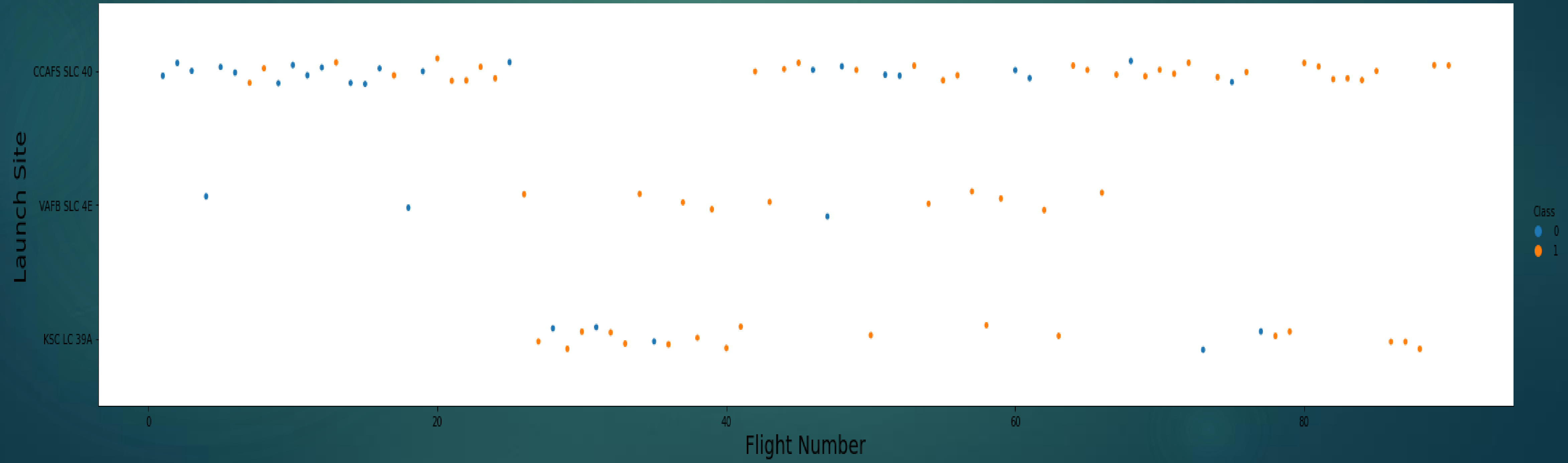
RESULTS

- EDA with visualization
- EDA with SQL
- Interactive map with Folium
- Plotly Dash dashboard
- Predictive analysis

EDA with Visualization

Flight Number Vs Launch Site

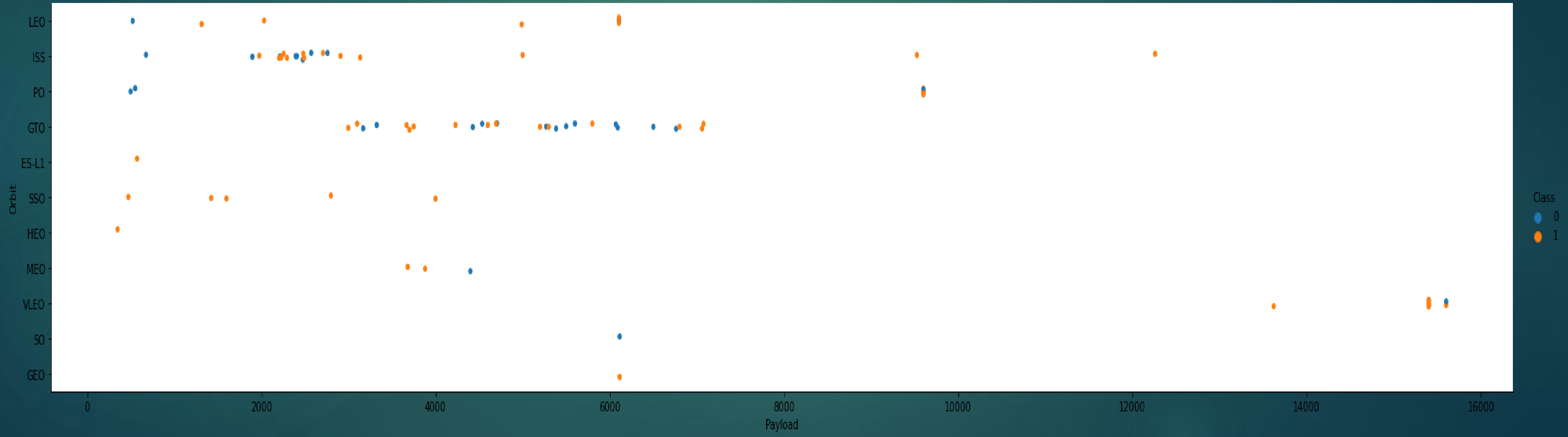
From this scattered plot we can see that irrespective of the launch site, the more flights are made the higher the success rate



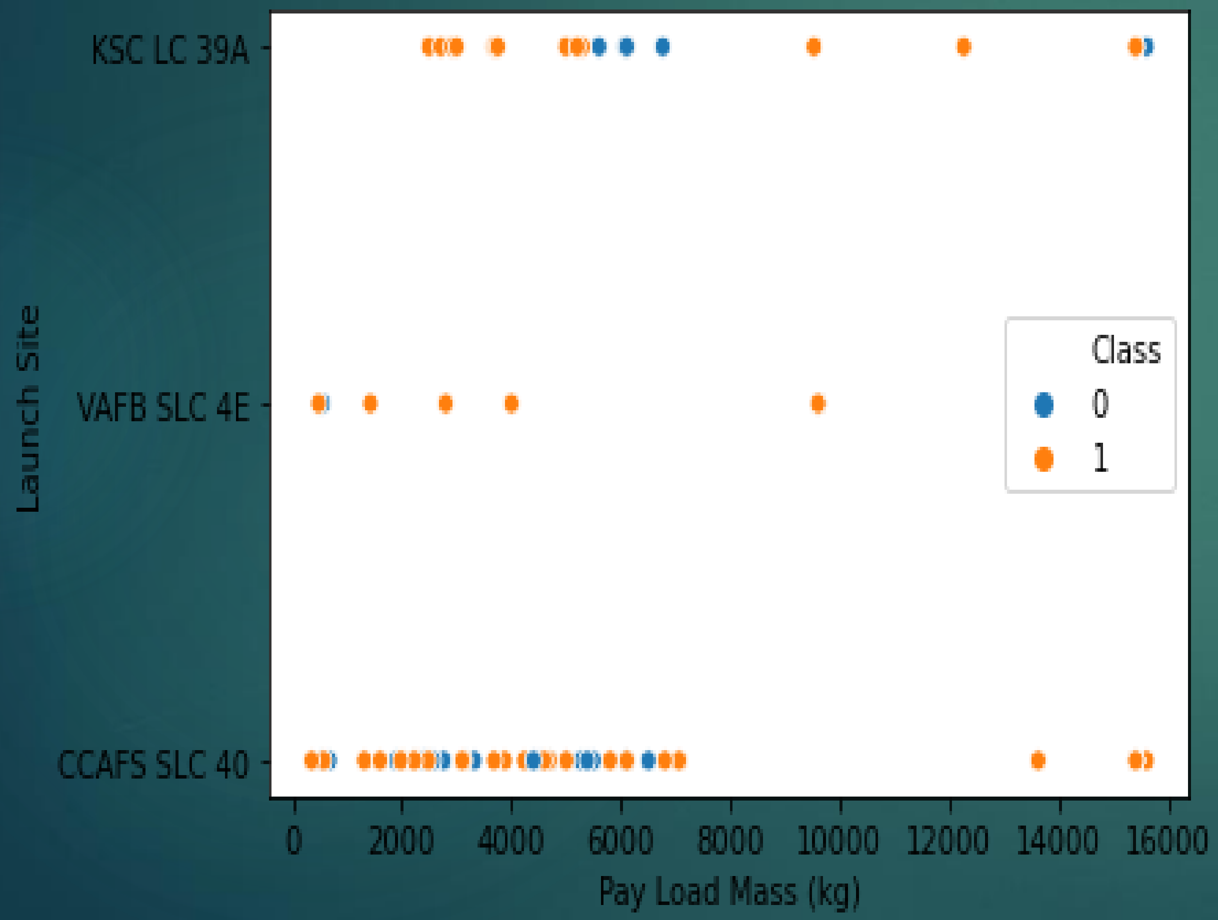
EDA with Visualization

Payload Vs Orbit

From this scattered plot we can see that with heavier payload the successful landing rate are more for Polar, LEO and ISS. However GTO has no distinguishable difference as both successful and failed landing outcomes are there.



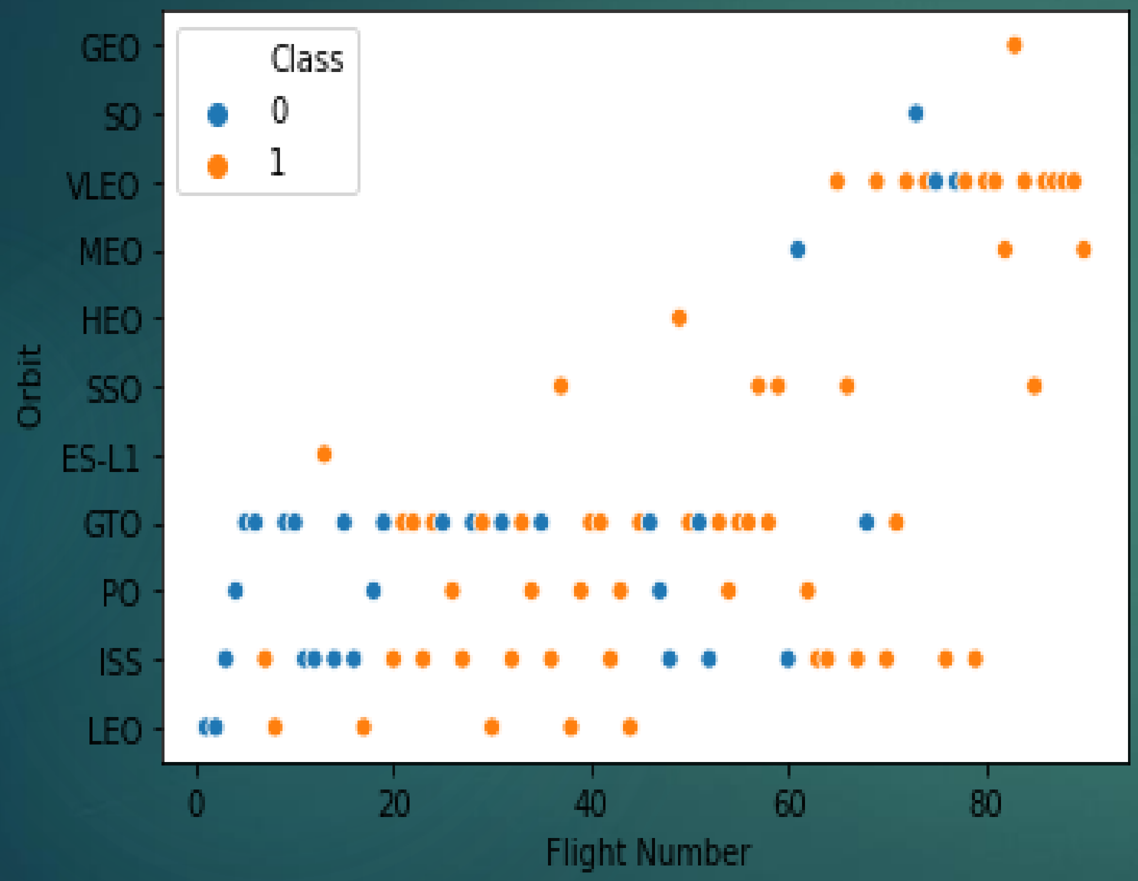
EDA with Visualization



Payload Vs Launch Site

In this scattered chart of Payload Vs launch site we can see that irrespective of the launch sites, rockets with payload > 4000kg and <6000kg have a higher failure rate.

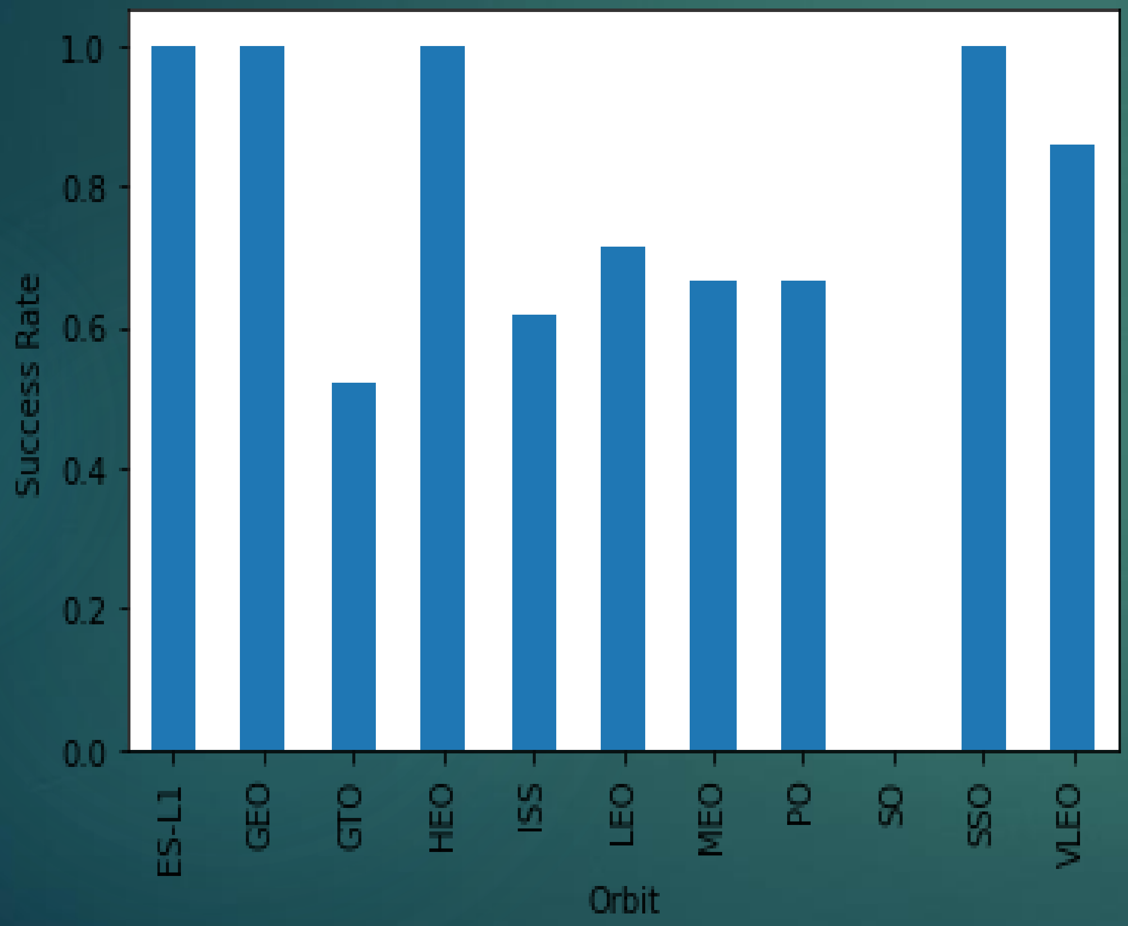
EDA with Visualization



Flight Number Vs Orbit

In this scattered chart of number of flights vs orbit we can see that In the LEO orbit the Success appears to be related to the number of flights; on the other hand, there seem to be no relationship between flight number when in GTO orbit.

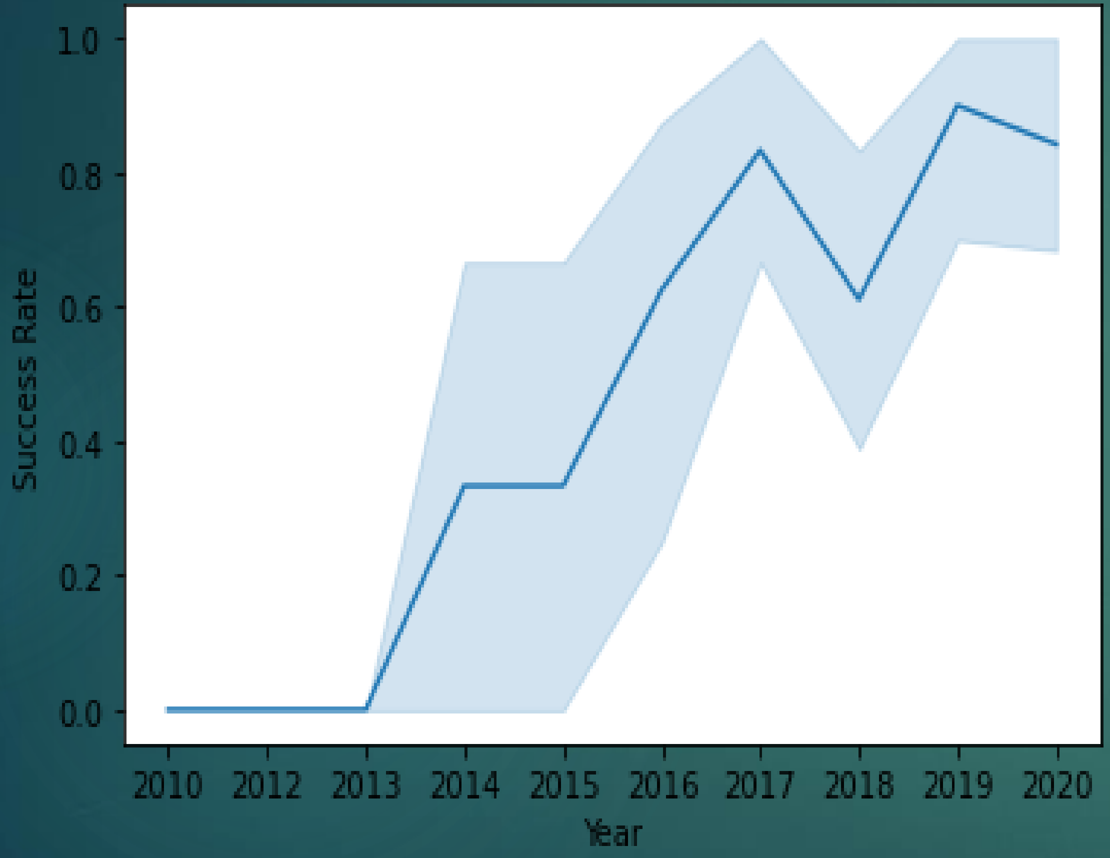
EDA with Visualization



Success Rate Vs Orbit

This bar chart shows that launches aimed at ES-L1, GEO, HEO, and SSO, have a higher success rate than most with SO orbit having the lowest success rate

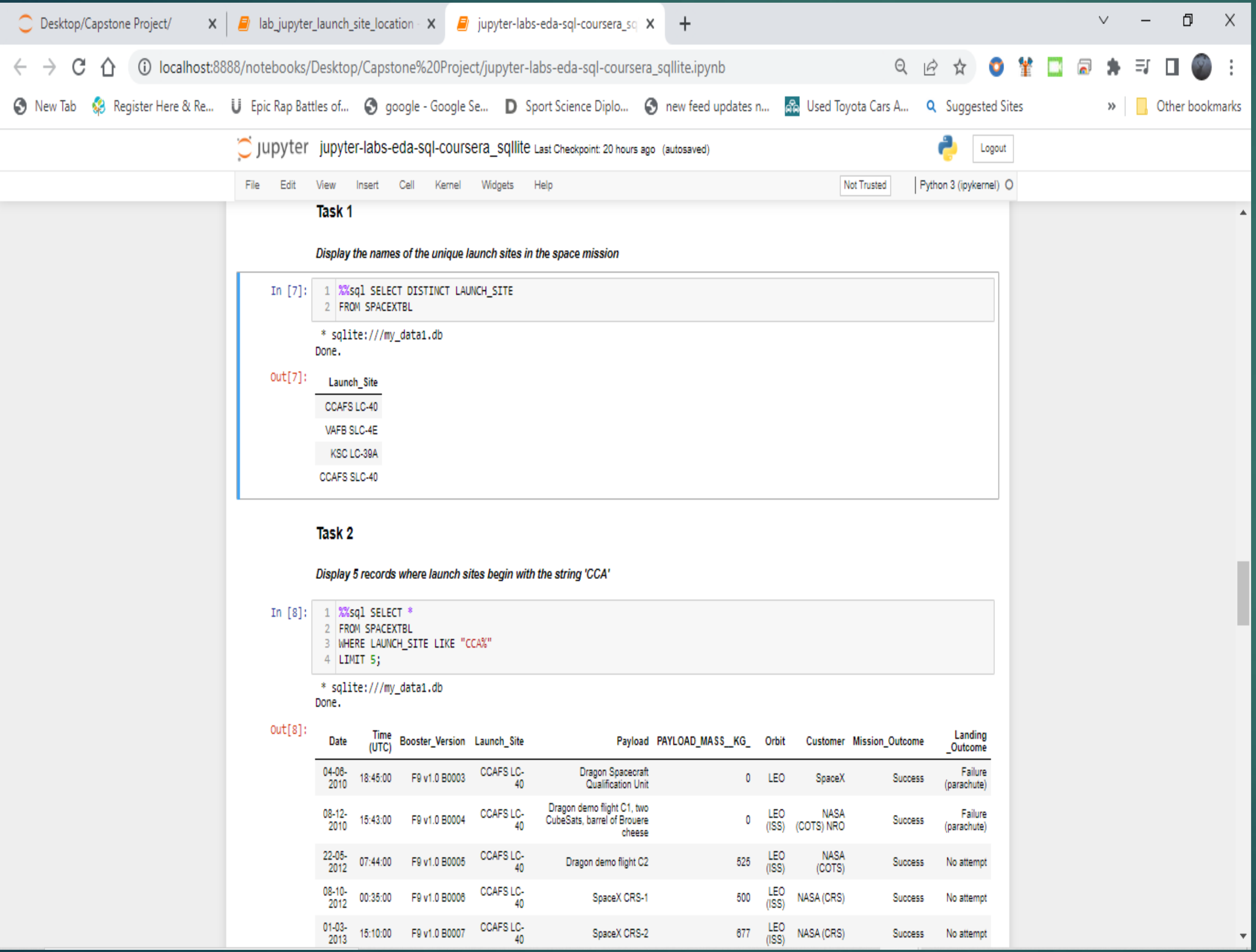
EDA with Visualization



Success Rate Vs Year

In this chart we can observe that the success rate since 2013 kept increasing until a dip in 2018, and a spike from there till 2020.

EDA with SQL



Task 1

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

In [7]:
```
%%sql SELECT DISTINCT LAUNCH_SITE
FROM SPACEXTBL
```

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

Out[7]:

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'

In [8]:
```
%%sql SELECT *
FROM SPACEXTBL
WHERE LAUNCH_SITE LIKE "CCA%"
LIMIT 5;
```

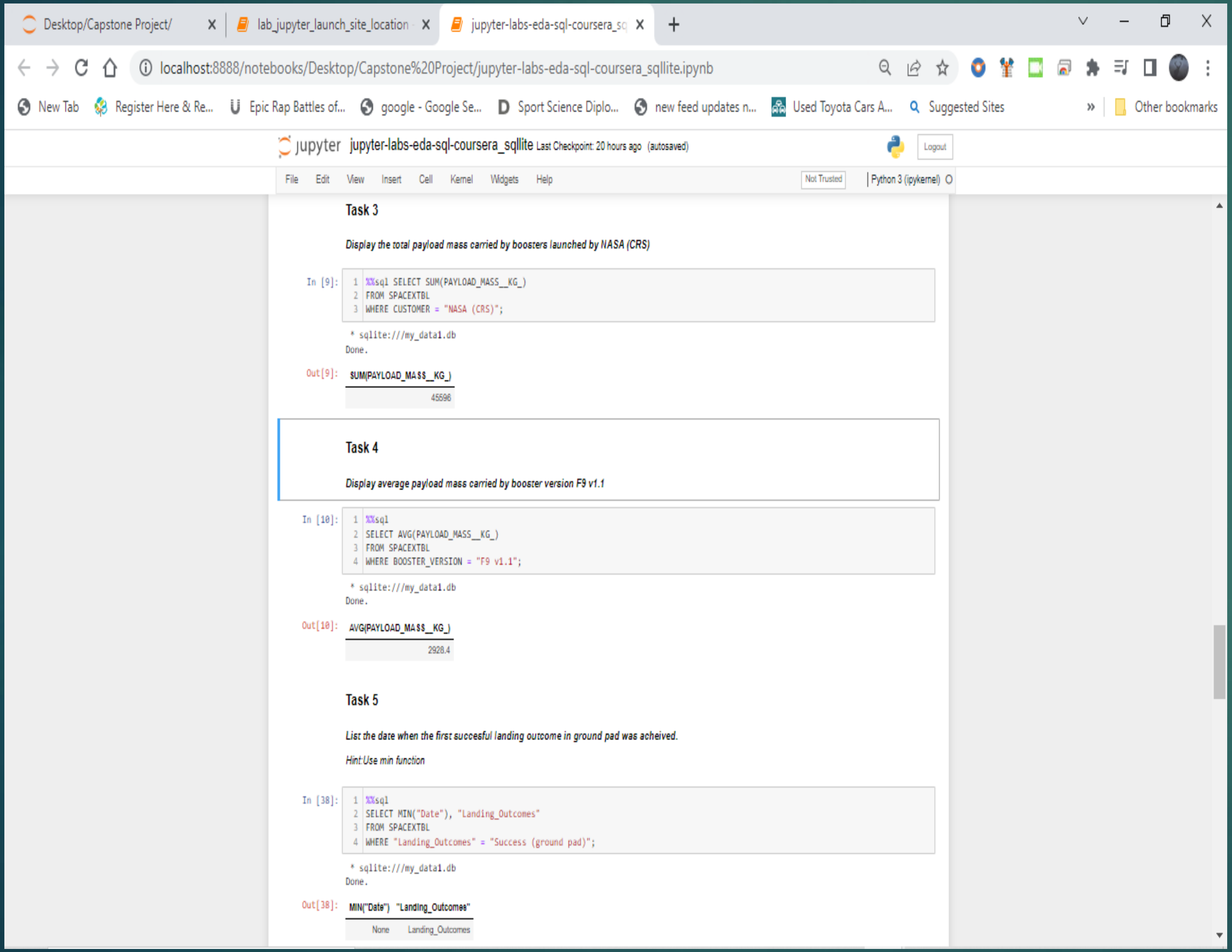
```
 * sqlite:///my_data1.db
Done.
```

Out[8]:

Date	Time (UTC)	Booster_Version	Launch_Site	Payload	PAYLOAD_MASS__KG_	Orbit	Customer	Mission_Outcome	Landing _Outcome
04-06-2010	18:45:00	F9 v1.0 B0003	CCAFS LC-40	Dragon Spacecraft Qualification Unit	0	LEO	SpaceX	Success	Failure (parachute)
08-12-2010	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)
22-05-2012	07:44:00	F9 v1.0 B0005	CCAFS LC-40	Dragon demo flight C2	525	LEO (ISS)	NASA (COTS)	Success	No attempt
08-10-2012	00:35:00	F9 v1.0 B0006	CCAFS LC-40	SpaceX CRS-1	500	LEO (ISS)	NASA (CRS)	Success	No attempt
01-03-2013	15:10:00	F9 v1.0 B0007	CCAFS LC-40	SpaceX CRS-2	677	LEO (ISS)	NASA (CRS)	Success	No attempt

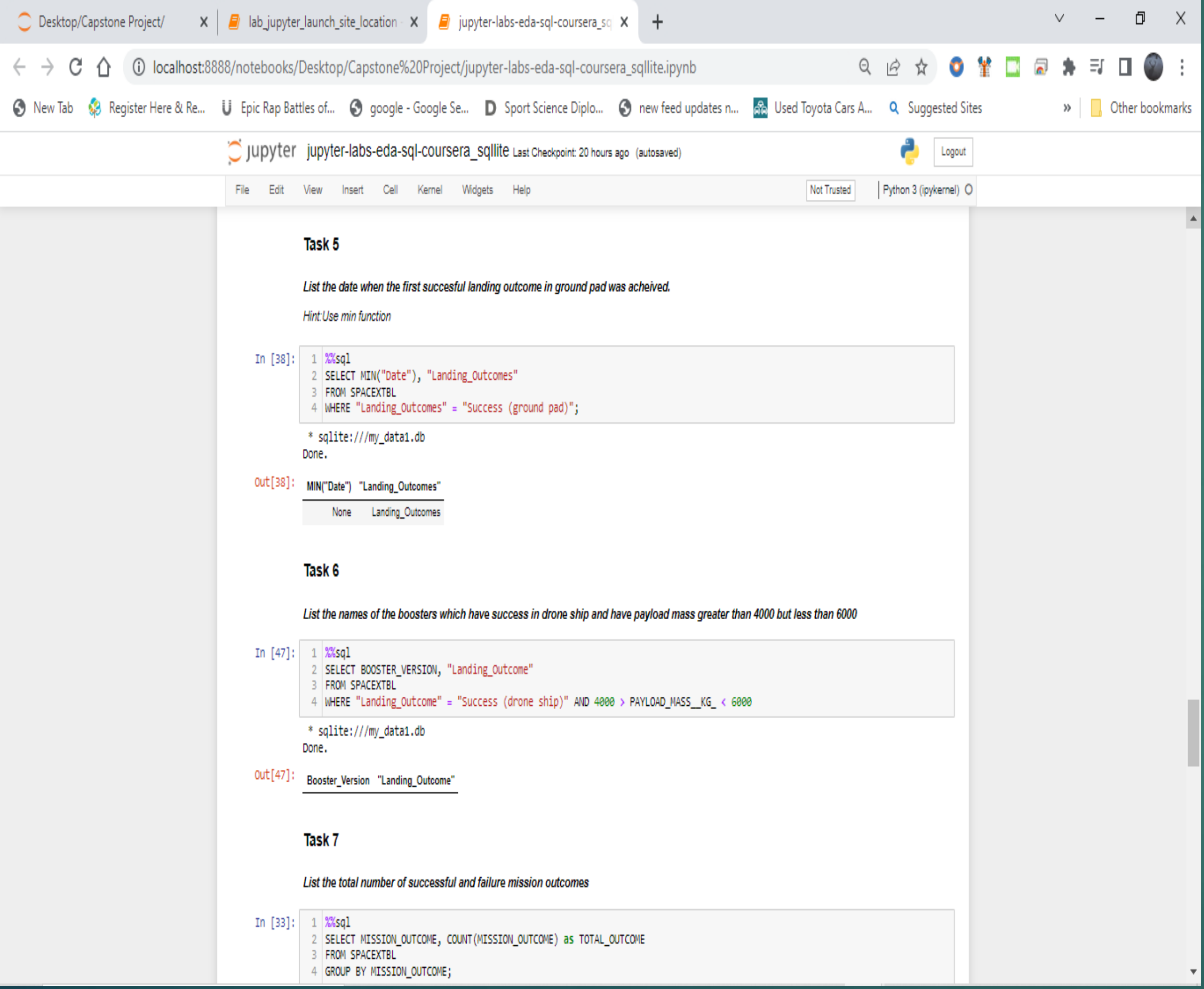
- Display the unique launch sites names
- 5 records where Launch sites names begins with "CCA"

EDA with SQL



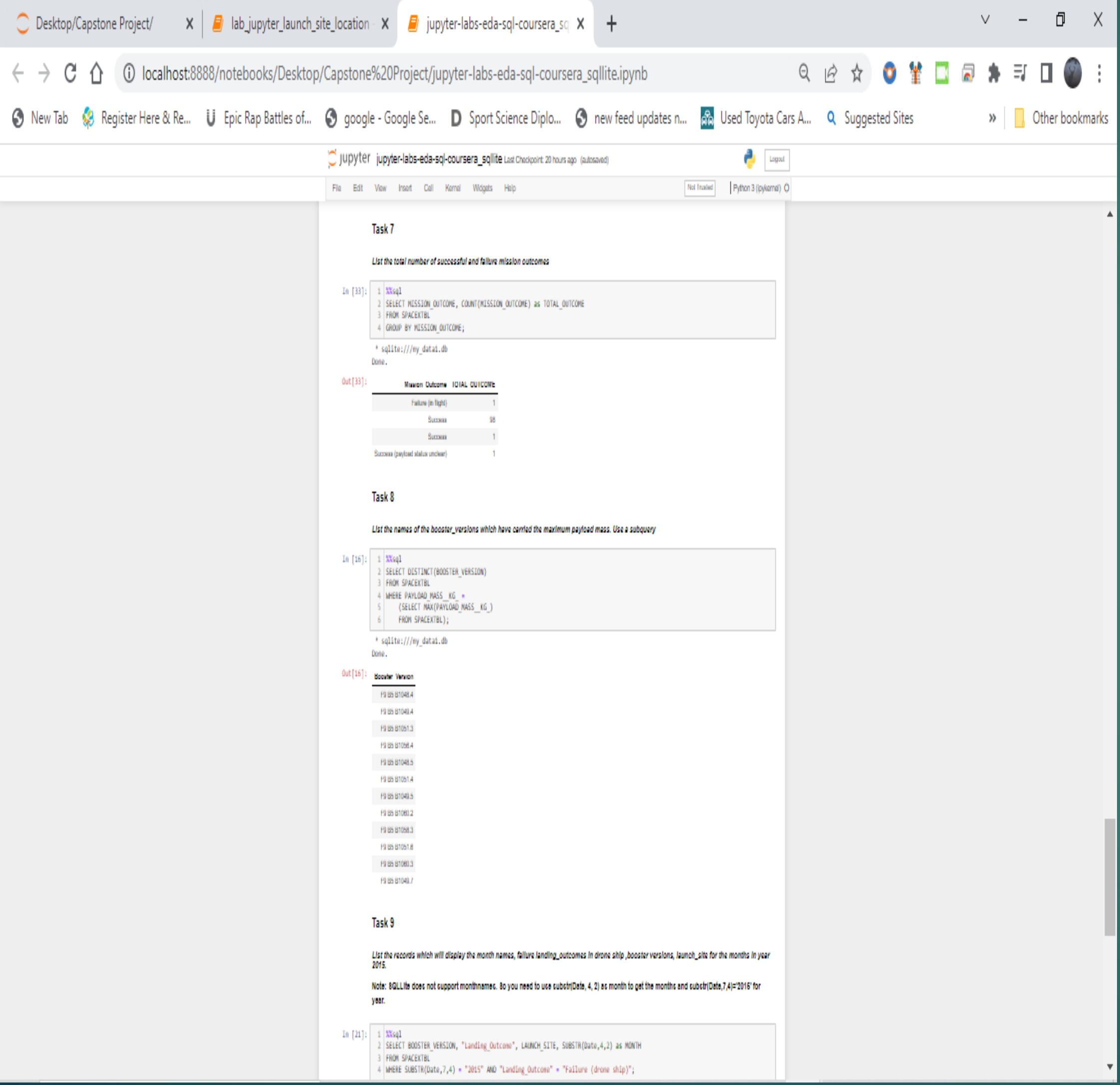
- Total Payload carried by boosters launched by NASA (CRS)
- Average payload carried by booster version F9 v1.1
- Date when the first successful landing outcome in ground pad was achieved

EDA with SQL



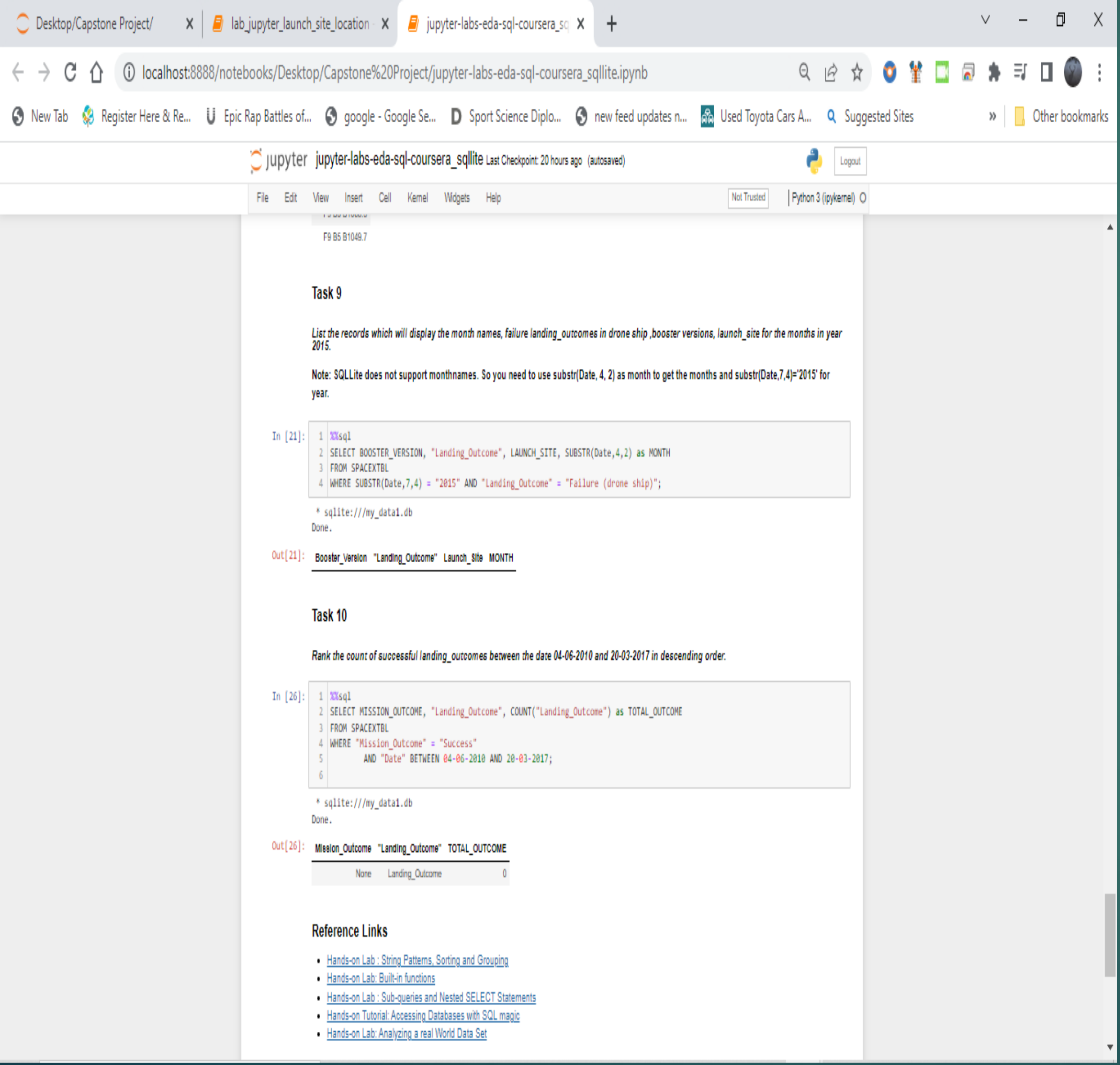
- Names of booster with success in drone ship having payload mass greater than 4000 and less than 6000

EDA with SQL



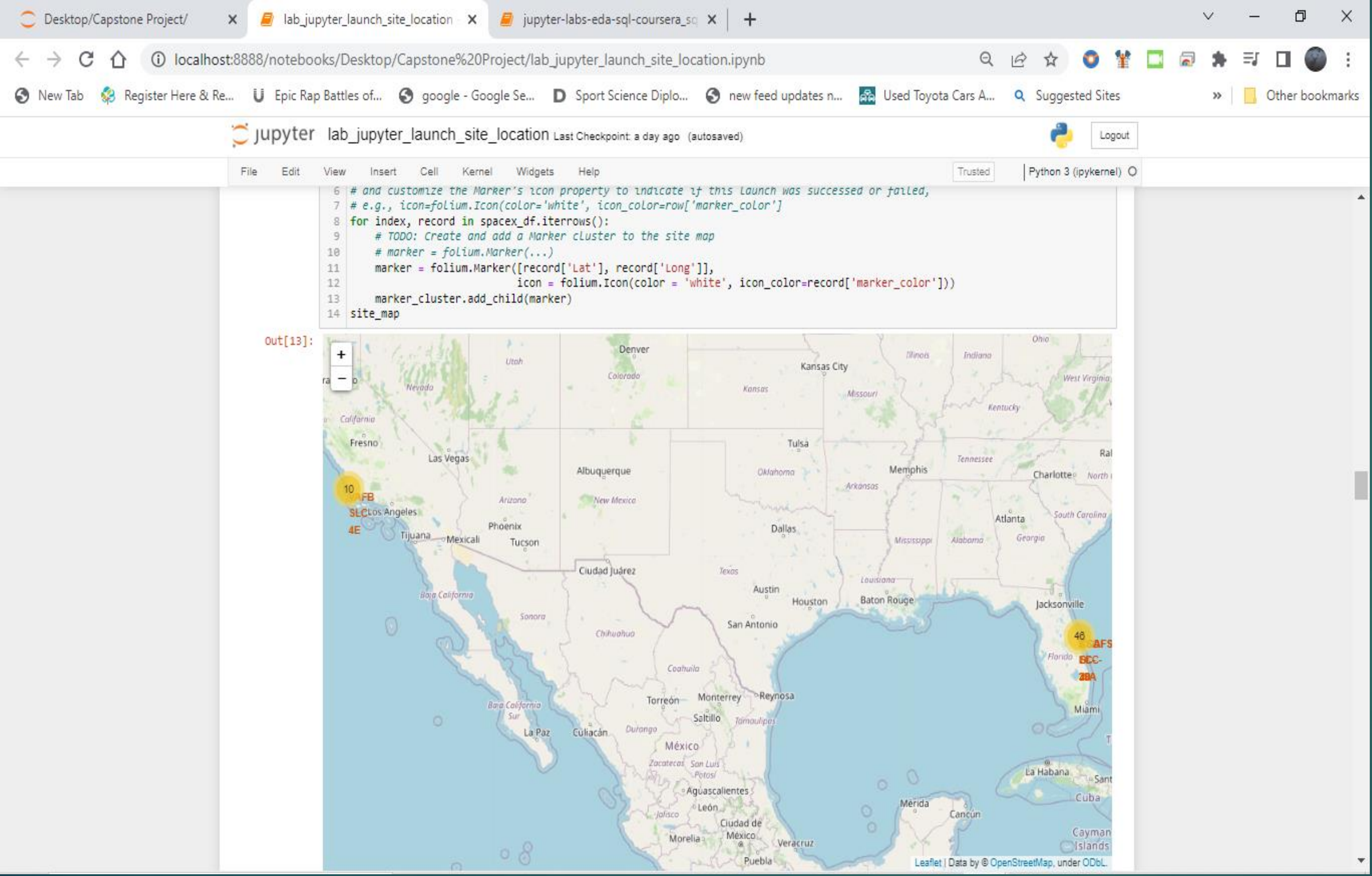
- List of total number of successful and failure mission outcome.
- Names of booster versions which have carried the maximum payload mass.

EDA with SQL



- Records which will display the names, failure landing outcome in drone ship, booster versions launch _size for the months in year 2015
- Rank the count of successful landing outcomes between the date 04-06-2010 and 20-03-2017 in descending order

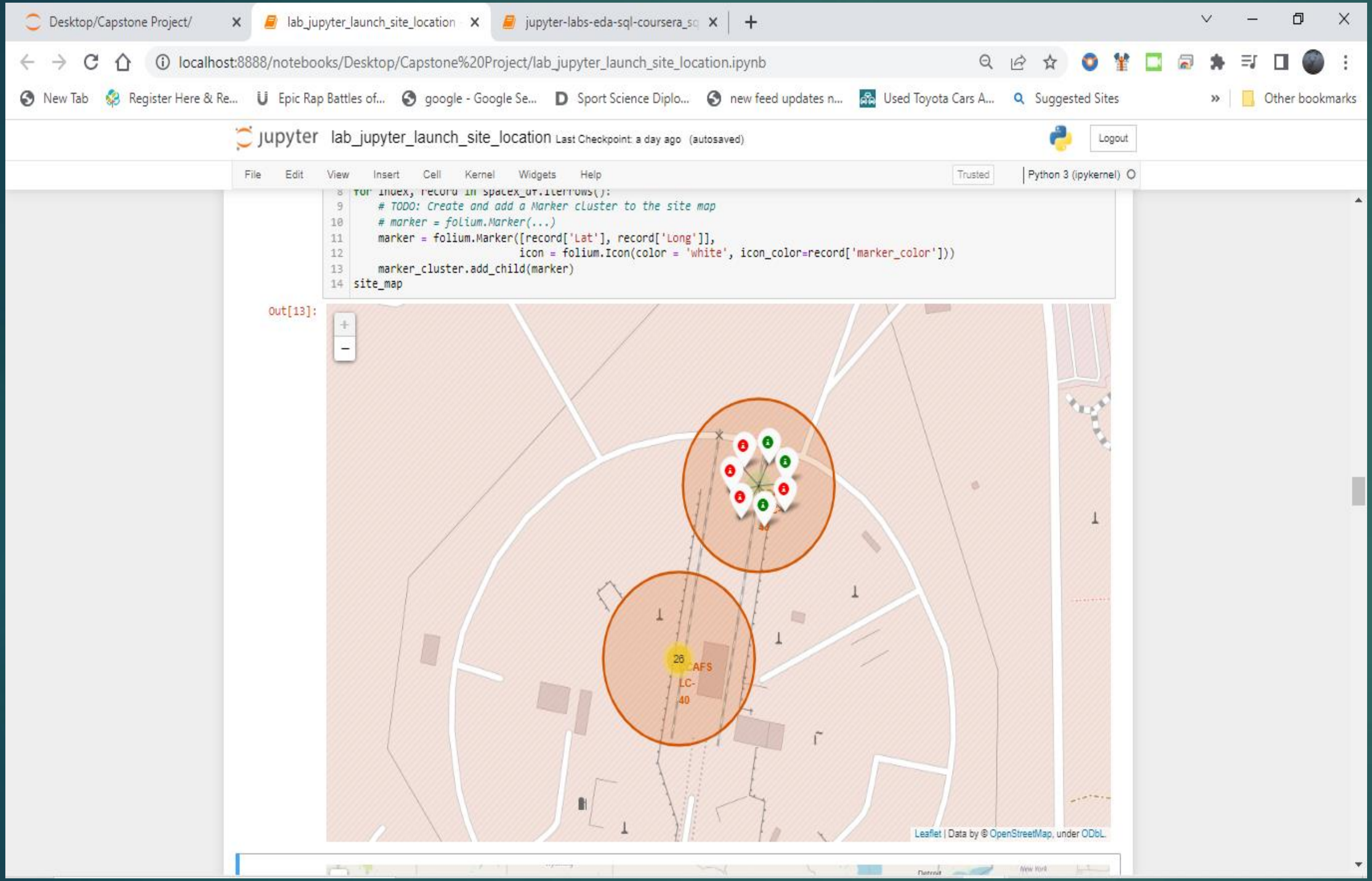
Interactive Map with folium



Folium circle and label for each launch site on the map.

This shows the major coordinates on the map where launches took place.

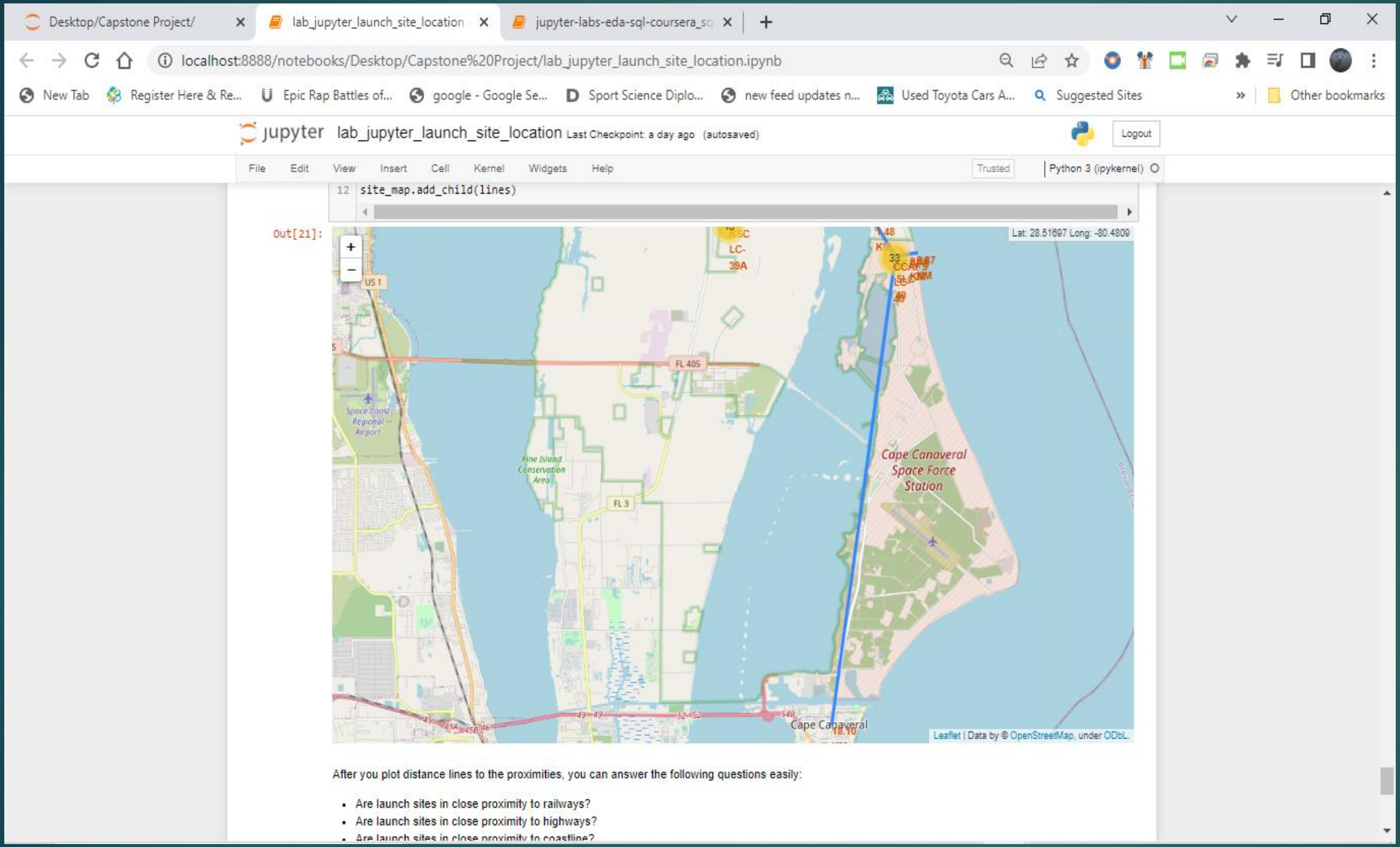
Interactive Map with folium



Marker cluster for each launch result

This shows the various launches successful (green), failed(red) at CCAFS SLC-40 launch site

Interactive Map with folium

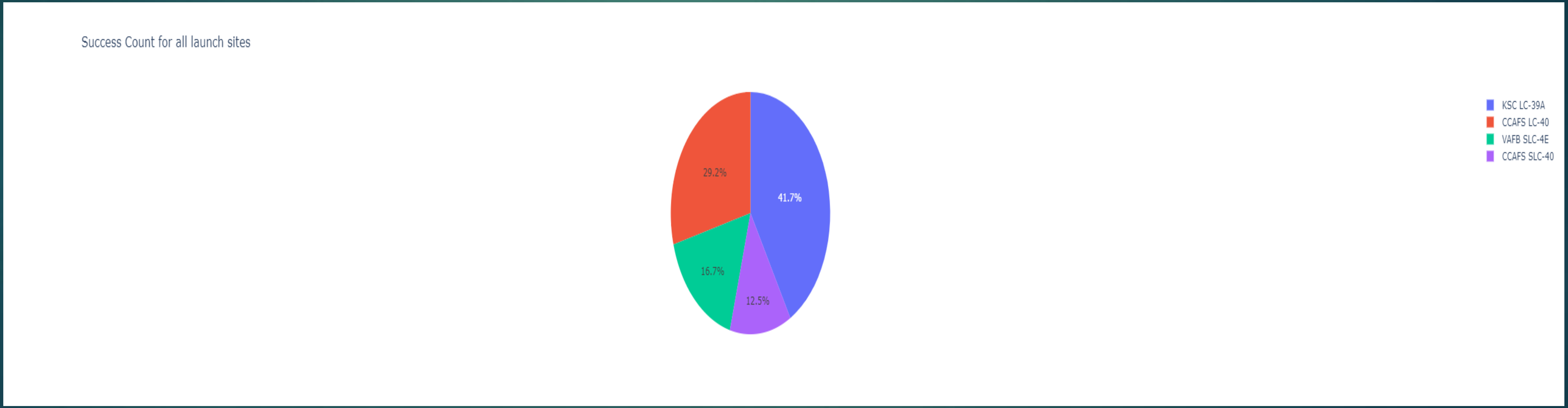


Proximity Map

This map shows lines with distance from launch site to major city, highway, coastline, and railway.

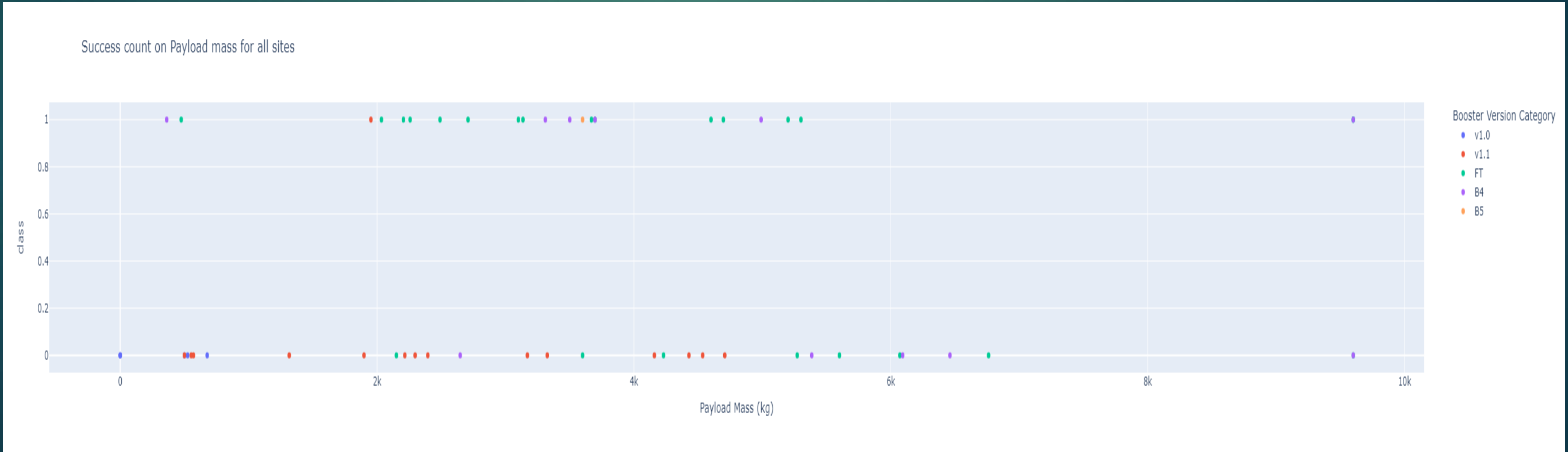
Interactive Visual with Plotly Dash

The pie chart shows that KSC LC-39A has the largest successful launches with a 41% launch success count



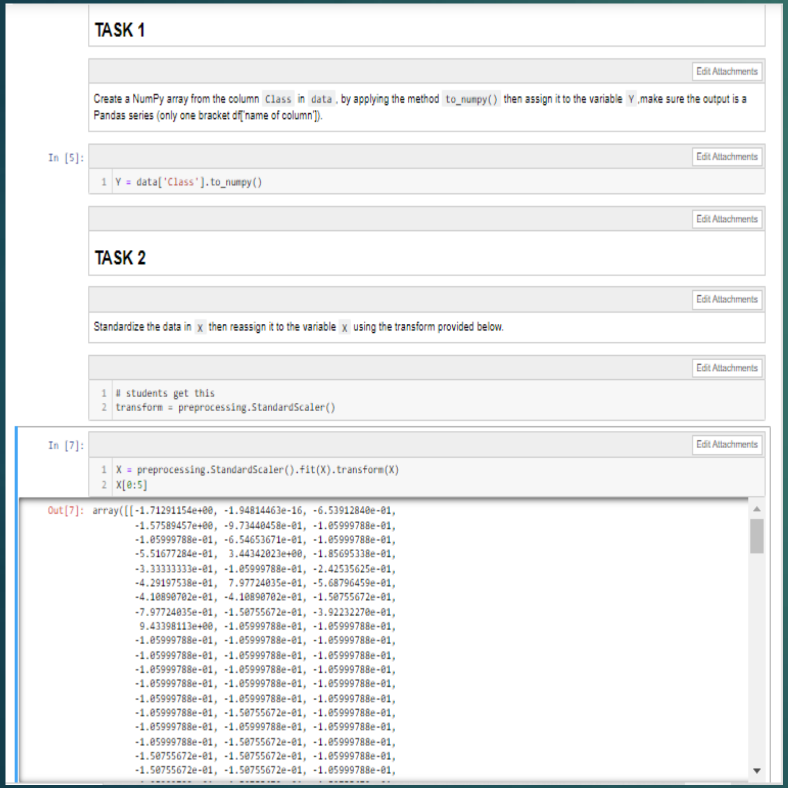
Interactive Visual with Plotly Dash

- The scattered plot shows that payload within 2k and 4k range has the highest success range
- Payload within 6k and 8k range has the lowest success rate
- F9 Booster Version FT has the highest launch success rate



Predictive analysis Result Accuracy and confusion matrix

Creating a numpy array from column class as our dependent variable. Standardizing x data and assigning it to our independent variable after transforming it



TASK 1

Edit Attachments

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]:

Edit Attachments

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

Edit Attachments

TASK 2

Edit Attachments

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

Edit Attachments

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

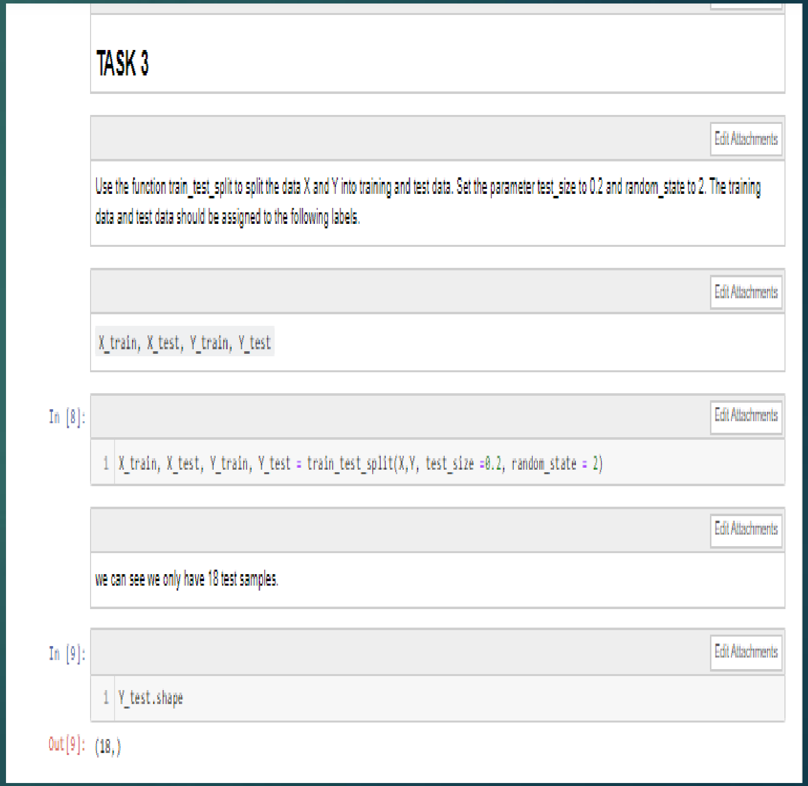
In [7]:

Edit Attachments

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

```
Out[7]: 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,
```

Splitting our data into train and test data set.



TASK 3

Edit Attachments

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.

Edit Attachments

`X_train, X_test, Y_train, Y_test`

In [8]:

Edit Attachments

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

Edit Attachments

we can see we only have 18 test samples.

In [9]:

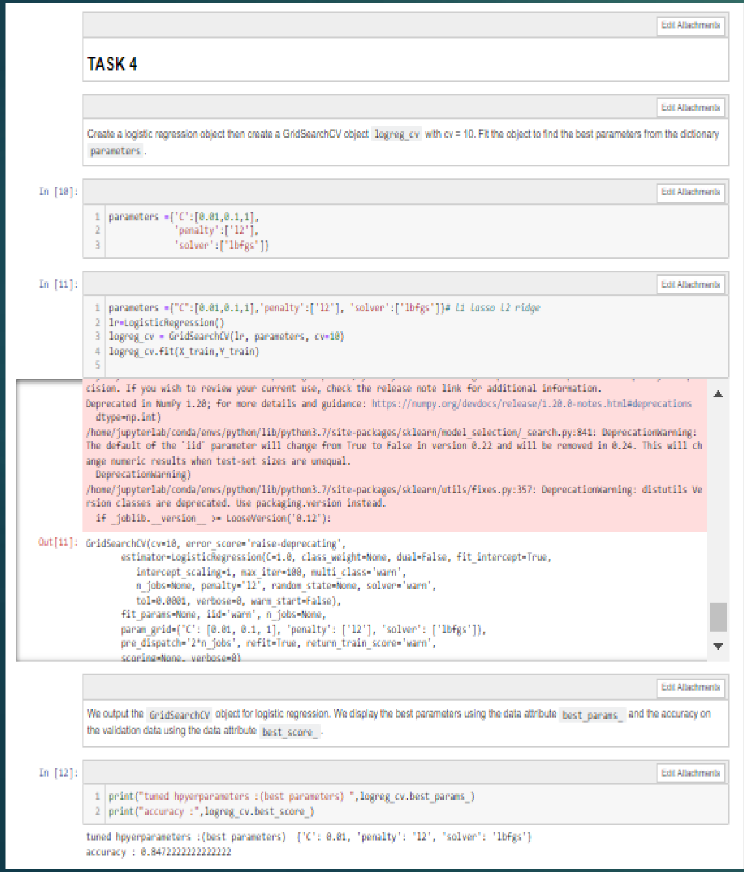
Edit Attachments

```
Y_test.shape
```

Out[9]: (18,)

Logistic Regression

Creation of a LR object, fit the object to find the best parameter from the dictionary parameters.



Edit Attachments

TASK 4

Edit Attachments

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]: Edit Attachments

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

In [11]: Edit Attachments

```
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)
```

```
cision. If you wish to review your current use, check the release note link for additional information.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  dtype=np.int)
/home/jupyterlab/conda/envs/python/lib/python3.7/site-packages/sklearn/model_selection/_search.py:841: DeprecationWarning:
The default of the `iid` parameter will change from True to False in version 0.22 and will be removed in 0.24. This will ch
ange numeric results when test-set sizes are unequal.
  DeprecationWarning)
/home/jupyterlab/conda/envs/python/lib/python3.7/site-packages/sklearn/utils/fixes.py:357: DeprecationWarning: distutils Ve
rsion classes are deprecated. Use packaging.version instead.
  if _joblib.__version__ >= LooseVersion('0.12'):
```

Out[11]:

```
GridSearchCV(cv=10, error_score='raise-deprecating',
       estimator=LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None, solver='warn',
          tol=0.0001, verbose=0, warm_start=False),
       fit_params=None, iid='warn', n_jobs=None,
       param_grid={'C': [0.01, 0.1, 1], 'penalty': ['l2'], 'solver': ['lbfgs']},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring=None, verbose=0)
```

Edit Attachments

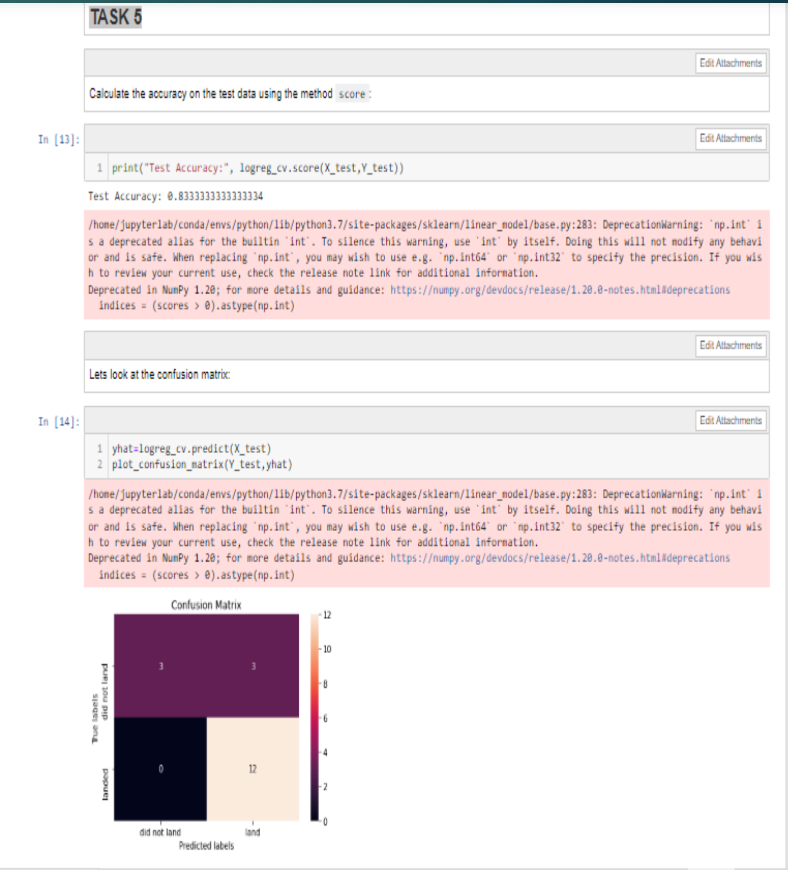
We output the `GridSearchCV` object for logistic regression. We display the best parameters using the data attribute `best_params_` and the accuracy on the validation data using the data attribute `best_score_`.

In [12]: Edit Attachments

```
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 : 0.8472222222222222
```

Calculating the accuracy, and plotting the confusion matrix



TASK 5

Edit Attachments

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

In [13]: Edit Attachments

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

```
Test Accuracy: 0.8333333333333334
/home/jupyterlab/conda/envs/python/lib/python3.7/site-packages/sklearn/linear_model/base.py:283: DeprecationWarning: `np.int` i
s a deprecated alias for the builtin `int`. To silence this warning, use `int` by itself. Doing this will not modify any behavi
or and is safe. When replacing `np.int`, you may wish to use e.g. `np.int64` or `np.int32` to specify the precision. If you wis
h to review your current use, check the release note link for additional information.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  indices = (scores > 0).astype(np.int)
```

Edit Attachments

Lets look at the confusion matrix:

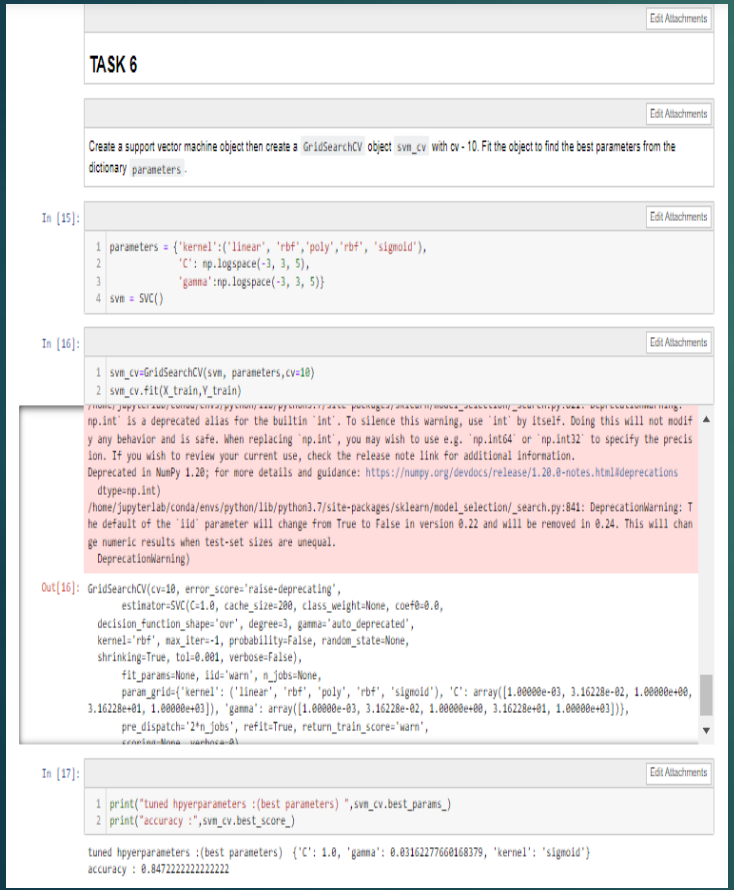
In [14]: Edit Attachments

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

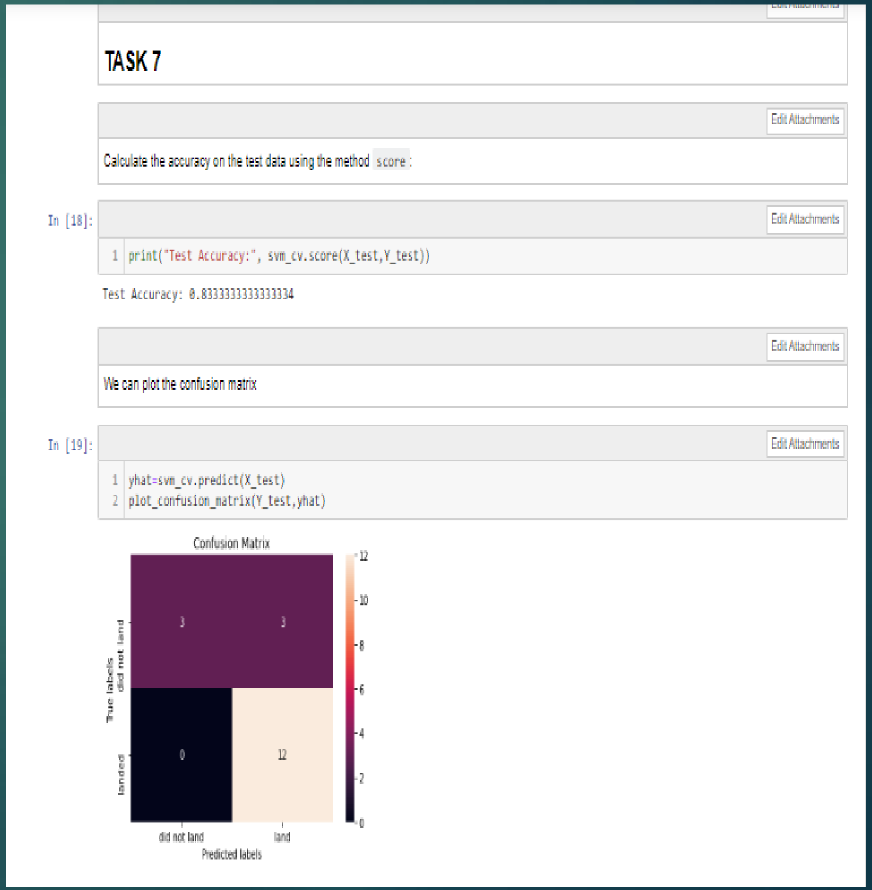
```
/home/jupyterlab/conda/envs/python/lib/python3.7/site-packages/sklearn/linear_model/base.py:283: DeprecationWarning: `np.int` i
s a deprecated alias for the builtin `int`. To silence this warning, use `int` by itself. Doing this will not modify any behavi
or and is safe. When replacing `np.int`, you may wish to use e.g. `np.int64` or `np.int32` to specify the precision. If you wis
h to review your current use, check the release note link for additional information.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  indices = (scores > 0).astype(np.int)
```

Support Vector Machine

Create a SVM object, fit and search for the best parameter from the parameter dictionary

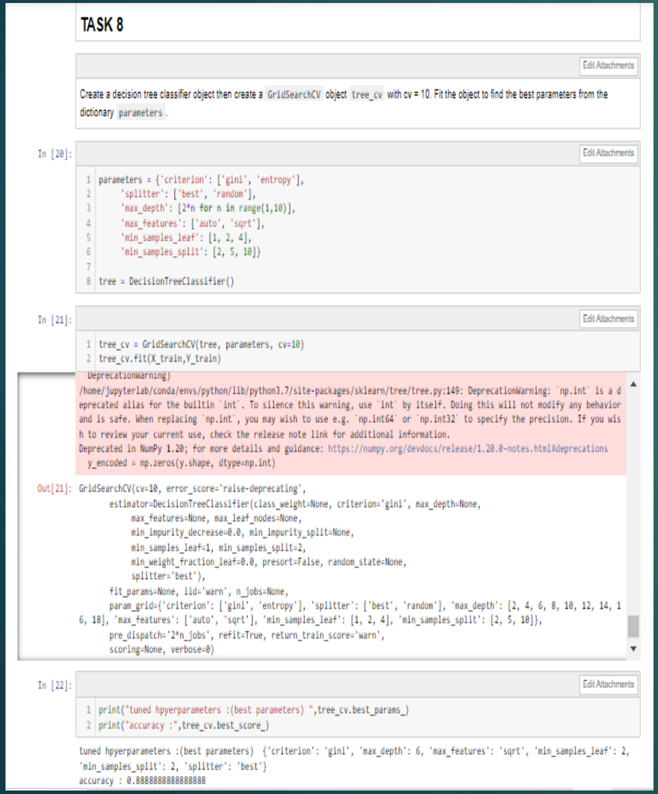


Calculate accuracy and plot confusion matrix



Decision Tree Classifier

Create a decision tree classifier object, fit to search for the best parameter from the parameter dictionary.



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)
```

```
  DeprecationWarning)
/home/jupyterlab/conda/envs/python/lib/python3.7/site-packages/sklearn/tree/tree.py:149: DeprecationWarning: `np.int` is a d
eprecated alias for the builtin `int`. To silence this warning, use `int` by itself. Doing this will not modify any behavior
and is safe. When replacing `np.int`, you may wish to use e.g. `np.int64` or `np.int32` to specify the precision. If you wis
h to review your current use, check the release note link for additional information.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  y_encoded = np.zeros(y.shape, dtype=np.int)
```

Out[21]:

```
GridSearchCV(cv=10, error_score='raise-deprecating',
       estimator=DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best'),
       fit_params=None, iid='warn', n_jobs=None,
       param_grid={'criterion': ['gini', 'entropy'], 'splitter': ['best', 'random'], 'max_depth': [2, 4, 6, 8, 10, 12, 14, 1
6, 18], 'max_features': ['auto', 'sqrt'], 'min_samples_leaf': [1, 2, 4], 'min_samples_split': [2, 5, 10]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring=None, verbose=0)
```

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': 6, 'max_features': 'sqrt', 'min_samples_leaf': 2,
'min_samples_split': 2, 'splitter': 'best'}
accuracy : 0.8888888888888888
```

Calculate accuracy and plot confusion matrix

TASK 9

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

In [23]:

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

```
Test Accuracy: 0.7777777777777778
```

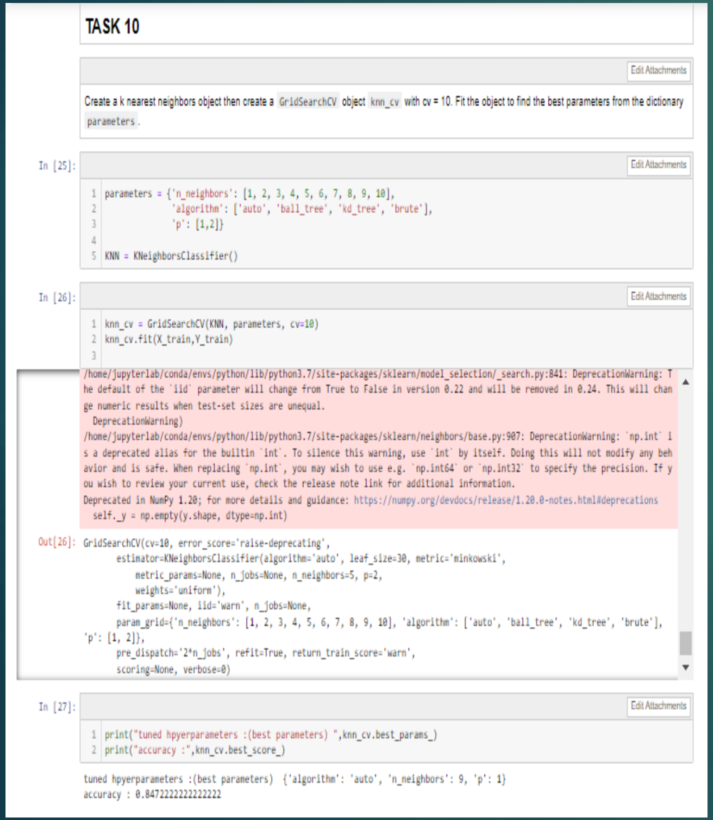
We can plot the confusion matrix

In [24]:

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

K Nearest Neighbour

Create a KNN object, fit to find the best parameter from the parameter dictionary.



TASK 10

Edit Attachments

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]: Edit Attachments

```
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]: Edit Attachments

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

```
/home/jupyterlab/conda/envs/python/lib/python3.7/site-packages/sklearn/model_selection/_search.py:841: DeprecationWarning: The default of the `iid` parameter will change from True to False in version 0.22 and will be removed in 0.24. This will change numeric results when test-set sizes are unequal.
  DeprecationWarning)
/home/jupyterlab/conda/envs/python/lib/python3.7/site-packages/sklearn/neighbors/base.py:907: DeprecationWarning: `np.int` is a deprecated alias for the builtin `int`. To silence this warning, use `int` by itself. Doing this will not modify any behavior and is safe. When replacing `np.int`, you may wish to use e.g. `np.int64` or `np.int32` to specify the precision. If you wish to review your current use, check the release note link for additional information.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  self._y = np.empty(y.shape, dtype=np.int)
```

Out[26]:

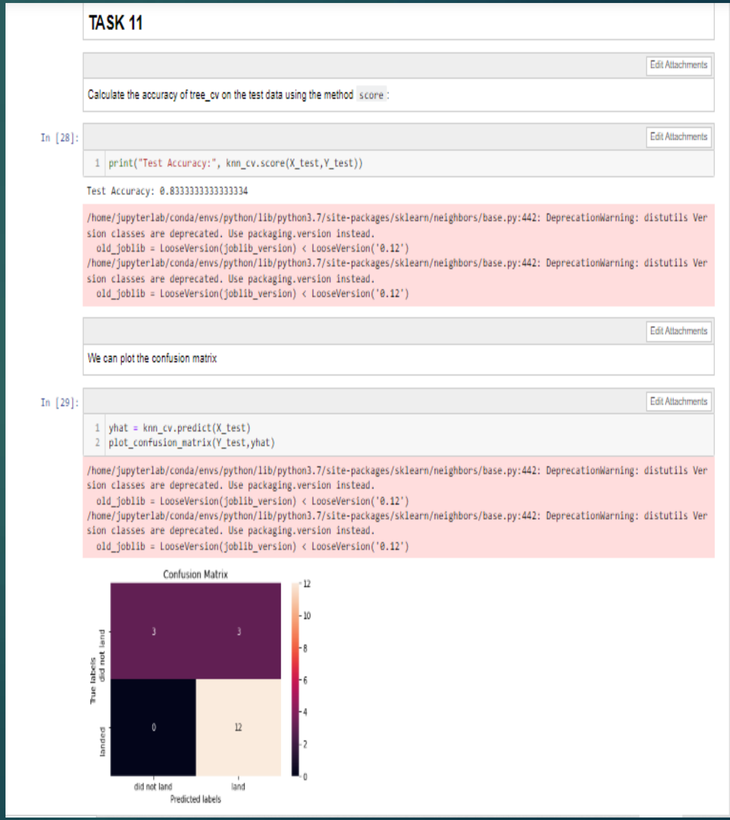
```
GridSearchCV(cv=10, error_score='raise-deprecating',
       estimator=KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=None, n_neighbors=5, p=2,
           weights='uniform'),
       fit_params=None, iid='warn', n_jobs=None,
       param_grid={'n_neighbors': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10], 'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute'],
'p': [1, 2]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring=None, verbose=0)
```

In [27]: Edit Attachments

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

```
tuned hpyerparameters :(best parameters)  {'algorithm': 'auto', 'n_neighbors': 9, 'p': 1}
accuracy : 0.8472222222222222
```

Calculate accuracy on the test data, and plot confusion matrix



TASK 11

Edit Attachments

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

In [28]: Edit Attachments

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

```
Test Accuracy: 0.8333333333333334
/home/jupyterlab/conda/envs/python/lib/python3.7/site-packages/sklearn/neighbors/base.py:442: DeprecationWarning: distutils Version classes are deprecated. Use packaging.version instead.
  old_joblib = LooseVersion(joblib_version) < LooseVersion('0.12')
/home/jupyterlab/conda/envs/python/lib/python3.7/site-packages/sklearn/neighbors/base.py:442: DeprecationWarning: distutils Version classes are deprecated. Use packaging.version instead.
  old_joblib = LooseVersion(joblib_version) < LooseVersion('0.12')
```

Edit Attachments

We can plot the confusion matrix

In [29]: Edit Attachments

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

```
/home/jupyterlab/conda/envs/python/lib/python3.7/site-packages/sklearn/neighbors/base.py:442: DeprecationWarning: distutils Version classes are deprecated. Use packaging.version instead.
  old_joblib = LooseVersion(joblib_version) < LooseVersion('0.12')
/home/jupyterlab/conda/envs/python/lib/python3.7/site-packages/sklearn/neighbors/base.py:442: DeprecationWarning: distutils Version classes are deprecated. Use packaging.version instead.
  old_joblib = LooseVersion(joblib_version) < LooseVersion('0.12')
```

CONCLUSION

- Irrespective of the launch site rockets with pay load within 4000kg and 6000kg had a higher failure rate
- Launch success rate climbed as from 2013
- Launch sites has lowest proximity to cities than any Highways, Rail tracks and even coastlines
- KSC LC-39A has the highest success launch rate than any other site
- Orbit SO has the lowest success rate
- Decision tree classifier is the best machine learning model for the task with a higher classification fit accuracy.


Thank you