SPACE-Y REUSABLE ROCKET PROGRAM

Cost Optimization by Prediction Analysis

Skills Network

OUTLINE

- Executive Summary
- Introduction
- Methodology
- Results
  - Codes
  - Visualization – Charts
  - Dashboard
- Conclusion

EXECUTIVE SUMMARY

- In this capstone, The aim is to predict if the Falcon 9 first stage will land successfully with other relevant information
- The Methodology involves Data Collection, Wrangling, EDA, Intractive Visual Analytics and Machine learning.
- The following were key findings:
  - ES-L1, GEO, HEO and SSO are ideal orbit candidates with 100% success rate.
  - Successful launches generally improve over time.
  - Generally, there is an estimated 80% chances of successfully recovering the first stage across all launch sites.

INTRODUCTION

- The commercial space age is here, companies are making space travel affordable for everyone. Space X is successful because their rocket launches are relatively inexpensive
- Space X launches cost $65m compared to the industry standard of $165m
- Space Y wants to compete with Space X, cost competition can be achieved by predicting if the first stage will land
- In this presentation, we seek to achieve this by
    - Apply knowledge of data science and machine learning in this scenario.
    - Analyze and visualize mined data using Python.
    - Build and validate a predictive machine learning model using Python.
    - Create and share actionable insights found.

METHODOLOGY
(Data Collection and Wrangling)

- Create a Jupyter notebook and make it sharable using GitHub.
- Use an API to extract information from a web service.
- Write Python code to manipulate data in a Pandas data frame.
- Convert a JSON file into a Pandas data frame.
- Load a dataset into a database.

METHODOLOGY
(EDA and Interactive Visual Analytics)

- Write and execute SQL queries to select and sort data.
- Write Python code to conduct exploratory data analysis by manipulating data in a Pandas data frame.
- Visualize the data and extract meaningful patterns to guide the modeling process.
- Create scatter plots and bar charts to analyze data in a Pandas data frame.
- Build an interactive map to analyze the launch site proximity with Folium.
- Calculate distances on an interactive map by writing Python code using the Folium library.
- Build an interactive dashboard that contains pie charts and scatter plots to analyze data with the Plotly Dash Python library.

METHODOLOGY
(Predictive Analysis)

- Train different classification models.
- Split the data into training testing data.
- Perform grid search to find the hyperparameters that allow a given algorithm to perform best.
- Use machine learning skills to build a predictive model.

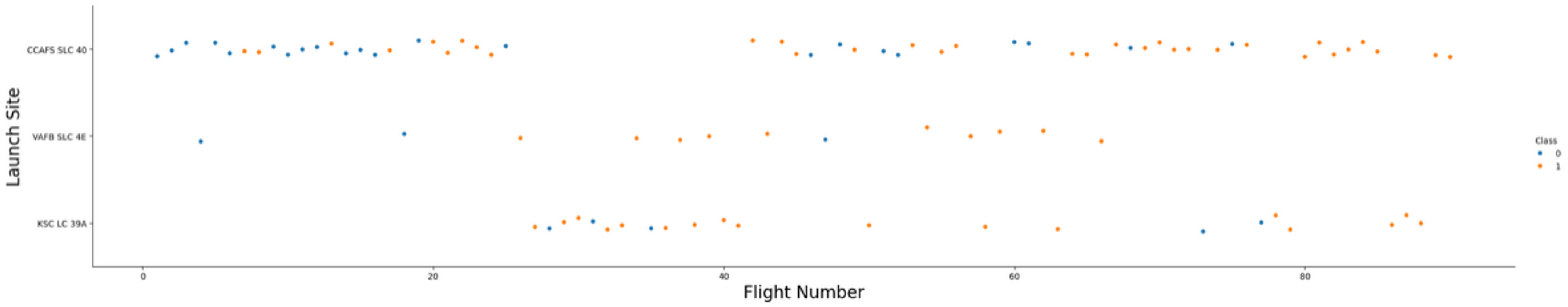
RESULTS
(EDA with Visualization)

TASK 1: Visualize the relationship between Flight Number and Launch Site

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'`

[5]:

```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.catplot(y="LaunchSite", x="FlightNumber", hue="Class", data=df, aspect = 5)
plt.xlabel("Flight Number",fontsize=20)
plt.ylabel("Launch Site",fontsize=20)
plt.show()
```



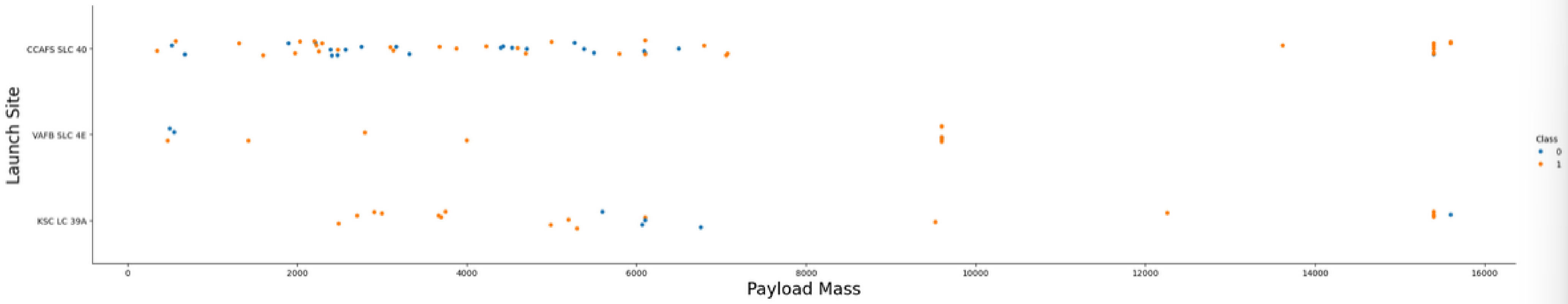
Now try to explain the patterns you found in the Flight Number vs. Launch Site scatter point plots.

RESULTS
(EDA with Visualization)

TASK 2: Visualize the relationship between Payload and Launch Site

We also want to observe if there is any relationship between launch sites and their payload mass.

[6]:
```
# 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.catplot(x="PayloadMass", y="LaunchSite", data=df, hue="Class", aspect=5)
plt.xlabel("Payload Mass", fontsize=20)
plt.ylabel("Launch Site", fontsize=20)
plt.show()
```



Now if you observe Payload Vs. Launch Site scatter point chart you will find for the VAFB-SLC launchsite there are no rockets launched for heavypayload mass(greater than 10000).

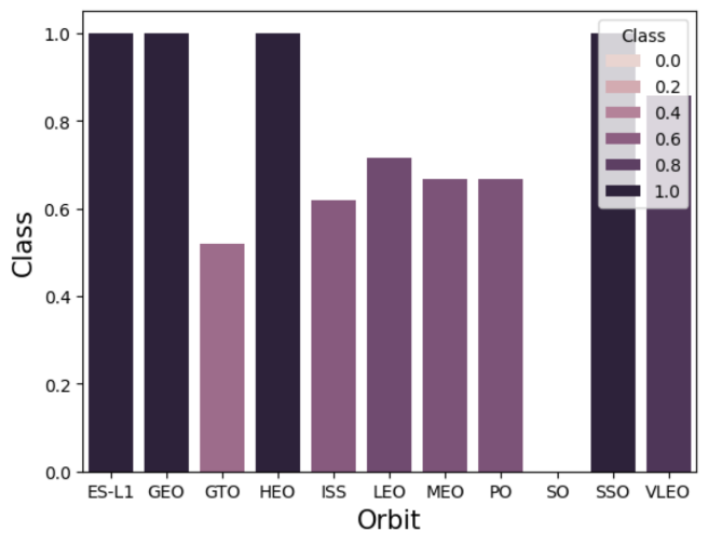
RESULTS
(EDA with Visualization)

TASK 3: Visualize the relationship between success rate of each orbit type

Next, we want to visually check if there are any relationship between success rate and orbit type.

Let's create a `bar chart` for the sucess rate of each orbit

[7]:
```
# HINT use groupby method on Orbit column and get the mean of Class column
df_group = df.groupby('Orbit').mean('Class').reset_index()
df_groupmean = df_group[['Orbit', 'Class']]
df_groupmean

# Plot a bar chart with x axis to be Orbit and y axis to be the Class, and hue to be the class value
sns.barplot(x="Orbit", y="Class", data=df_groupmean, hue="Class")
plt.xlabel("Orbit", fontsize=15)
plt.ylabel("Class", fontsize=15)
plt.show()
```



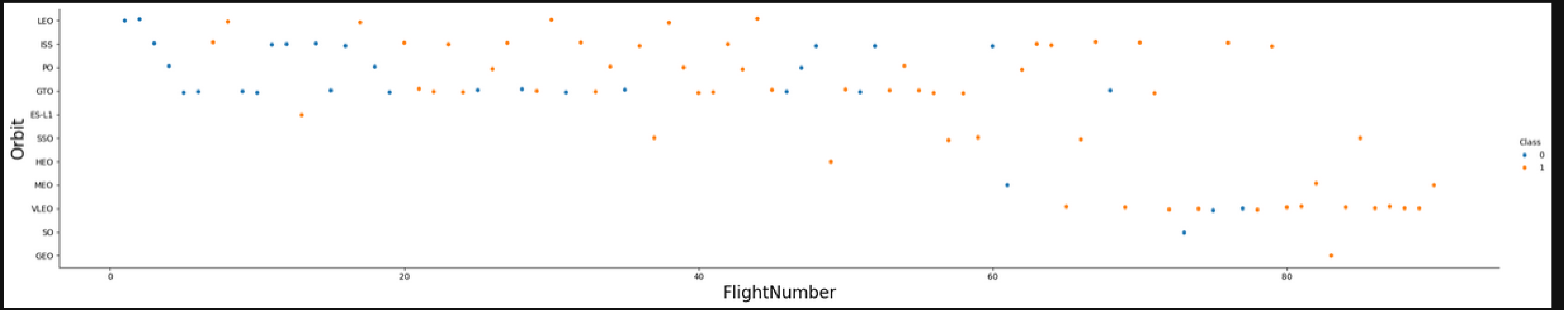
Analyze the ploted bar chart try to find which orbits have high sucess rate.

RESULTS
(EDA with Visualization)

TASK 4: Visualize the relationship between FlightNumber and Orbit type

For each orbit, we want to see if there is any relationship between FlightNumber and Orbit type.

[8]:
```
# Plot a scatter point chart with x axis to be FlightNumber and y axis to be the Orbit, and hue to be the class value
sns.catplot(x="FlightNumber", y="Orbit", data=df, hue="Class", aspect=5)
plt.xlabel("FlightNumber", fontsize=20)
plt.ylabel("Orbit", fontsize=20)
plt.show()
```



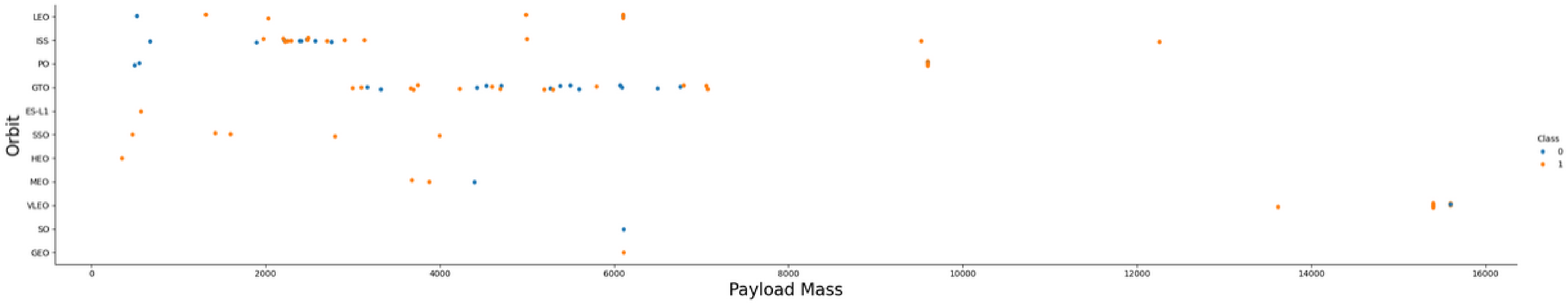
You should see that in the LEO orbit the Success appears related to the number of flights; on the other hand, there seems to be no relationship between flight number when in GTO orbit.

RESULTS
(EDA with Visualization)

TASK 5: Visualize the relationship between Payload and Orbit type

Similarly, we can plot the Payload vs. Orbit scatter point charts to reveal the relationship between Payload and Orbit type

[9]:
```
# Plot a scatter point chart with x axis to be Payload and y axis to be the Orbit, and hue to be the class value
sns.catplot(x="PayloadMass", y="Orbit", data=df, hue="Class", aspect=5)
plt.xlabel("Payload Mass", fontsize=20)
plt.ylabel("Orbit", fontsize=20)
plt.show()
```



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 here.

RESULTS
(EDA with Visualization)

TASK 6: Visualize the launch success yearly trend

You can plot a line chart with x axis to be Year and y axis to be average success rate, to get the average launch success trend.

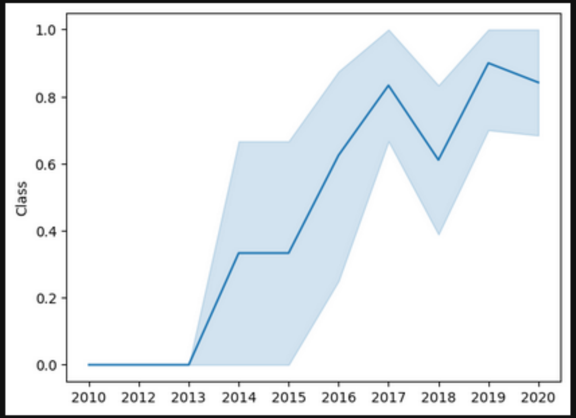
The function will help you get the year from the date:

[10]:
```
# A function to Extract years from the date
year=[]
def Extract_year():
    for i in df["Date"]:
        year.append(i.split("-")[0])
    return year
```

[22]:
```
#Execute function Extract Years to get the years
if len(year)>90:
    year.clear()
print ('list cleared')

Extract_year()

#df['Year'] = year
#df.head(5)

# Plot a line chart with x axis to be the extracted year and y axis to be the success rate
sns.lineplot(data=df, x=year, y="Class")
```

```
list cleared
```

[22]: <Axes: ylabel='Class'>



You can observe that the success rate since 2013 kept increasing till 2017 (stable in 2014) and after 2015 it started increasing.

RESULTS
(EDA with Visualization)

TASK 7: Create dummy variables to categorical columns

Use the function `get_dummies` and `features` dataframe to apply OneHotEncoder to the column `Orbits`, `LaunchSite`, `LandingPad`, and `Serial`. Assign the value to the variable `features_one_hot`, display the results using the method head. Your result dataframe must include all features including the encoded ones.

[26]:
```
# HINT: Use get_dummies() function on the categorical columns
features_one_hot = pd.get_dummies(features[['Orbit', 'LaunchSite', 'LandingPad', 'Serial']])
features_one_hot.head(5)
```

[26]:

	Orbit_ES-L1	Orbit_GEO	Orbit_GTO	Orbit_HEO	Orbit_ISS	Orbit_LEO	Orbit_MEO	Orbit_PO	Orbit_SO	Orbit_SSO	...	Serial_B1048	Serial_B1049	Serial_B1050	Serial_B1051	Seria
0	False	False	False	False	False	True	False	False	False	False	...	False	False	False	False	
1	False	False	False	False	False	True	False	False	False	False	...	False	False	False	False	
2	False	False	False	False	True	False	False	False	False	False	...	False	False	False	False	
3	False	False	False	False	False	False	False	True	False	False	...	False	False	False	False	
4	False	False	True	False	False	False	False	False	False	False	...	False	False	False	False	

5 rows × 72 columns

[27]:
```
features_two_hot = pd.concat([features, features_one_hot], axis = 1)
features_two_hot.head(5)
```

[27]:

	FlightNumber	PayloadMass	Orbit	LaunchSite	Flights	GridFins	Reused	Legs	LandingPad	Block	...	Serial_B1048	Serial_B1049	Serial_B1050	Serial_B1051	Serial_B1054
0	1	6104.959412	LEO	CCAFS SLC 40	1	False	False	False	NaN	1.0	...	False	False	False	False	False
1	2	525.000000	LEO	CCAFS SLC 40	1	False	False	False	NaN	1.0	...	False	False	False	False	False
2	3	677.000000	ISS	CCAFS SLC 40	1	False	False	False	NaN	1.0	...	False	False	False	False	False
3	4	500.000000	PO	VAFB SLC 4E	1	False	False	False	NaN	1.0	...	False	False	False	False	False
4	5	3170.000000	GTO	CCAFS SLC 40	1	False	False	False	NaN	1.0	...	False	False	False	False	False

5 rows × 84 columns

RESULTS
(EDA with Visualization)

TASK 8: Cast all numeric columns to `float64`

Now that our `features_one_hot` dataframe only contains numbers cast the entire dataframe to variable type `float64`

[29]:
```
# HINT: use astype function
features_one_hot = features_one_hot.astype(float)
features_one_hot.head(5)
```

[29]:

	Orbit_ES-L1	Orbit_GEO	Orbit_GTO	Orbit_HEO	Orbit_ISS	Orbit_LEO	Orbit_MEO	Orbit_PO	Orbit_SO	Orbit_SSO	...	Serial_B1048	Serial_B1049	Serial_B1050	Seria
0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	
1	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	
2	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	...	0.0	0.0	0.0	
4	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	

5 rows × 72 columns

RESULTS
(EDA with SQL)

Task 1

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

[24]:
```
%config SqlMagic.style = '_DEPRECATED_DEFAULT'
```

[25]:
```
%sql SELECT DISTINCT "Launch_Site" from SPACEXTABLE
```

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

[25]:

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 'KSC'

[34]:
```
%sql SELECT "Launch_Site" from SPACEXTABLE WHERE "Launch_Site" LIKE 'KSC%' LIMIT 5
```

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

[34]:

Launch_Site
KSC LC-39A
KSC LC-39A
KSC LC-39A
KSC LC-39A
KSC LC-39A

RESULTS
(EDA with SQL)

Task 3

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

[42]:
```
%sql SELECT SUM("PAYLOAD_MASS__KG_") FROM SPACEXTABLE WHERE Customer LIKE 'NASA (CRS)'
```

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

[42]:

SUM(PAYLOAD_MASS__KG_)
45596

Task 4

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

[46]:
```
%sql SELECT AVG("PAYLOAD_MASS__KG_") FROM SPACEXTABLE WHERE "Booster_Version" LIKE 'F9 v1.1'
```

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

[46]:

AVG(PAYLOAD_MASS__KG_)
2928.4

Task 5

List the date where the succesful landing outcome in drone ship was acheived.

Hint:Use min function

[48]:
```
%sql SELECT MIN(Date) FROM SPACEXTABLE WHERE "Landing_Outcome" == 'Success (drone ship)'
```

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

[48]:

MIN(Date)
2016-04-08

RESULTS (EDA with SQL)

Task 6

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

[50]:
```
%sql SELECT Payload FROM SPACEXTABLE WHERE "Landing_Outcome" == 'Success (ground pad)' AND "PAYLOAD_MASS__KG_" BETWEEN 4000 AND 6000
```

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

[50]:

Payload
NROL-76
Boeing X-37B OTV-5
Zuma

Task 7

List the total number of successful and failure mission outcomes

[53]:
```
%sql SELECT COUNT("Mission_Outcome") FROM SPACEXTABLE
```

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

[53]:

COUNT(Mission_Outcome)
101

RESULTS
(EDA with SQL)

Task 8

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

```
[57]: %sql SELECT Payload FROM SPACEXTABLE WHERE "PAYLOAD_MASS__KG_" == (SELECT MAX("PAYLOAD_MASS__KG_") FROM SPACEXTABLE)

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

[57]:

Payload
Starlink 1 v1.0, SpaceX CRS-19
Starlink 2 v1.0, Crew Dragon in-flight abort test
Starlink 3 v1.0, Starlink 4 v1.0
Starlink 4 v1.0, SpaceX CRS-20
Starlink 5 v1.0, Starlink 6 v1.0
Starlink 6 v1.0, Crew Dragon Demo-2
Starlink 7 v1.0, Starlink 8 v1.0
Starlink 11 v1.0, Starlink 12 v1.0
Starlink 12 v1.0, Starlink 13 v1.0
Starlink 13 v1.0, Starlink 14 v1.0
Starlink 14 v1.0, GPS III-04
Starlink 15 v1.0, SpaceX CRS-21

RESULTS
(EDA with SQL)

Task 9

List the records which will display the month names, succesful landing_outcomes in ground pad ,booster versions, launch_site for the months in year 2017

Note: SQLLite does not support monthnames. So you need to use substr(Date,6,2) for month, substr(Date,9,2) for date, substr(Date,0,5),='2017' for year.

```
[68]: %sql SELECT substr(Date,6,2) as Month, "Landing_Outcome", "Booster_Version", "Launch_Site" FROM SPACEXTABLE WHERE "Landing_Outcome" == 'Success (g
```

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

[68]:

Month	Landing_Outcome	Booster_Version	Launch_Site
02	Success (ground pad)	F9 FT B1031.1	KSC LC-39A
05	Success (ground pad)	F9 FT B1032.1	KSC LC-39A
06	Success (ground pad)	F9 FT B1035.1	KSC LC-39A
08	Success (ground pad)	F9 B4 B1039.1	KSC LC-39A
09	Success (ground pad)	F9 B4 B1040.1	KSC LC-39A
12	Success (ground pad)	F9 FT B1035.2	CCAFS SLC-40

RESULTS
(EDA with SQL)

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

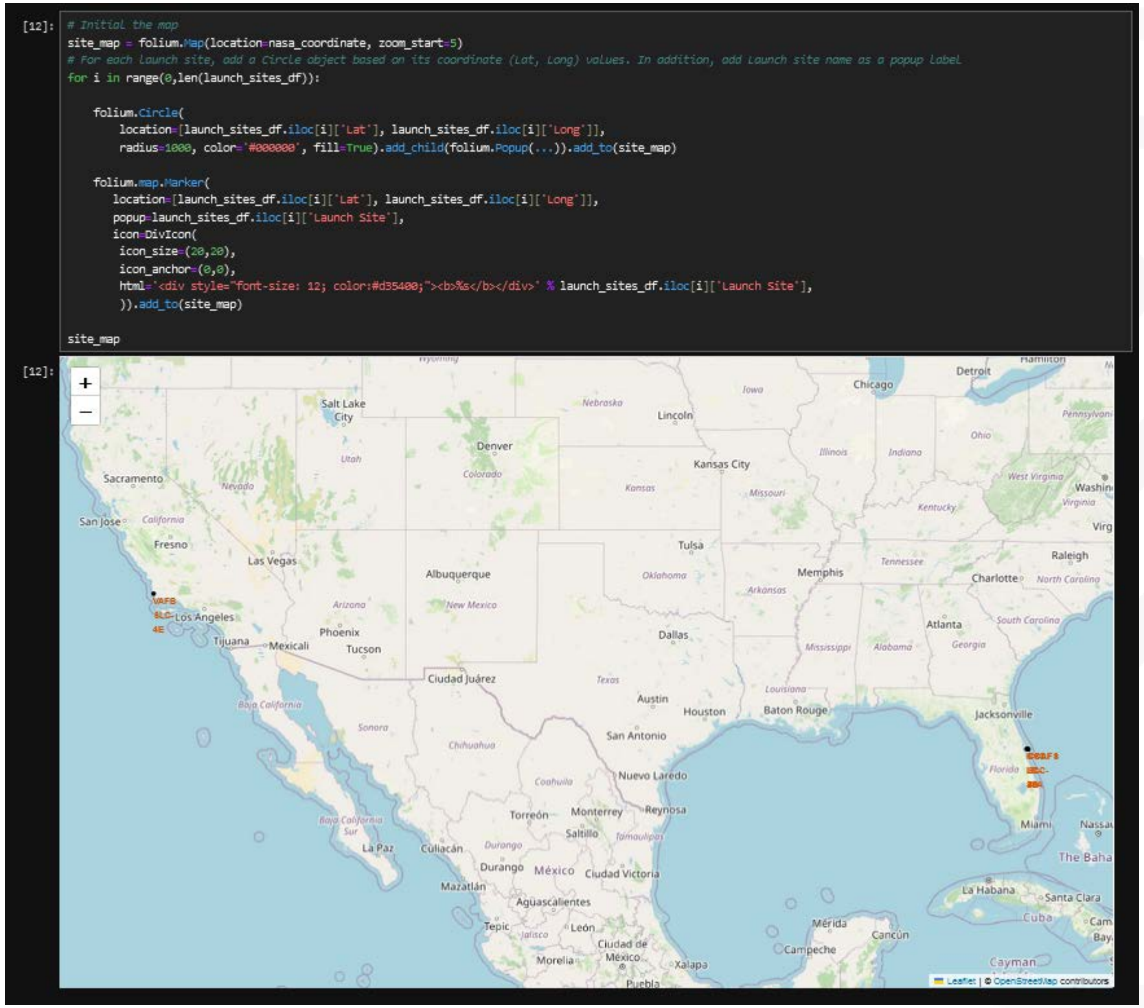
[91]:
```
%%sql
SELECT "Landing_Outcome", COUNT("Landing_Outcome") as Count FROM SPACEXTABLE
WHERE Date BETWEEN '2010-06-04' AND '2017-03-20'
GROUP BY "Landing_Outcome"
ORDER BY Count Desc
```

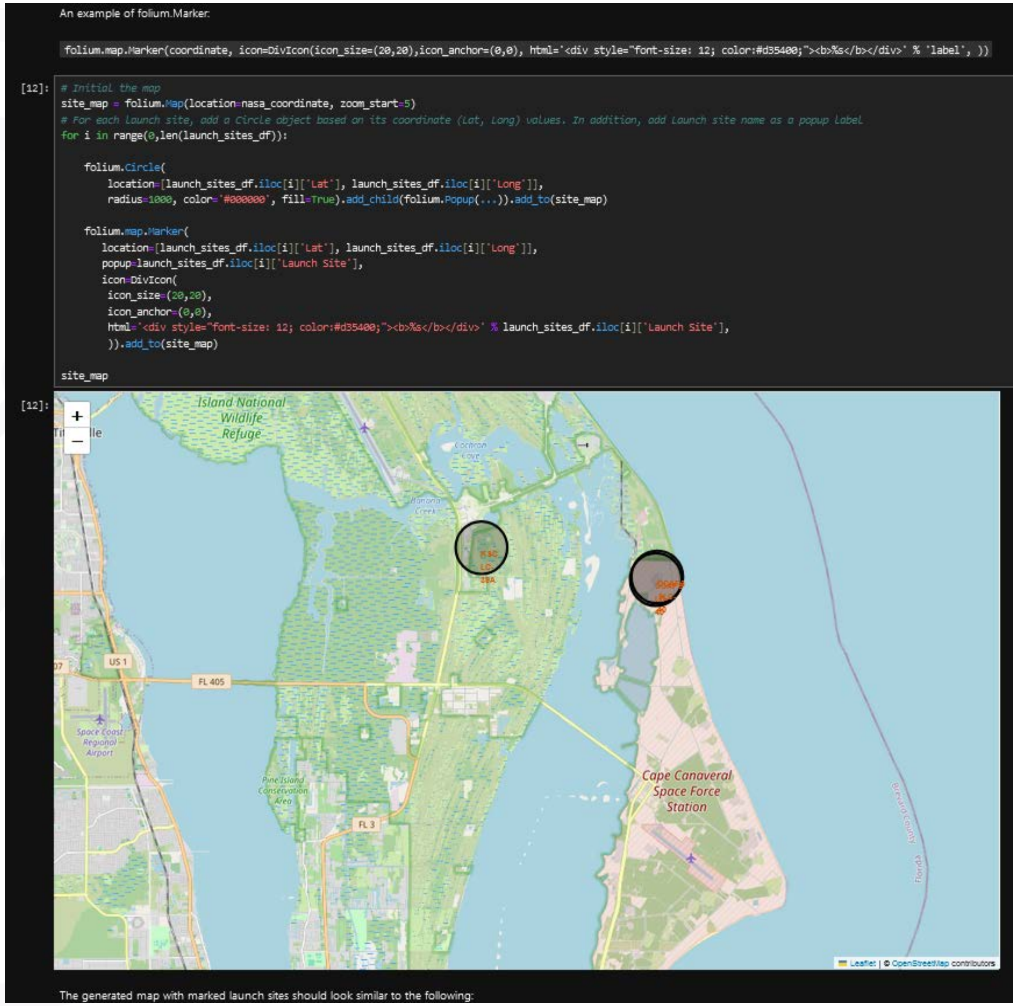
```
 * sqlite:///my_data1.db
Done.
```

[91]:

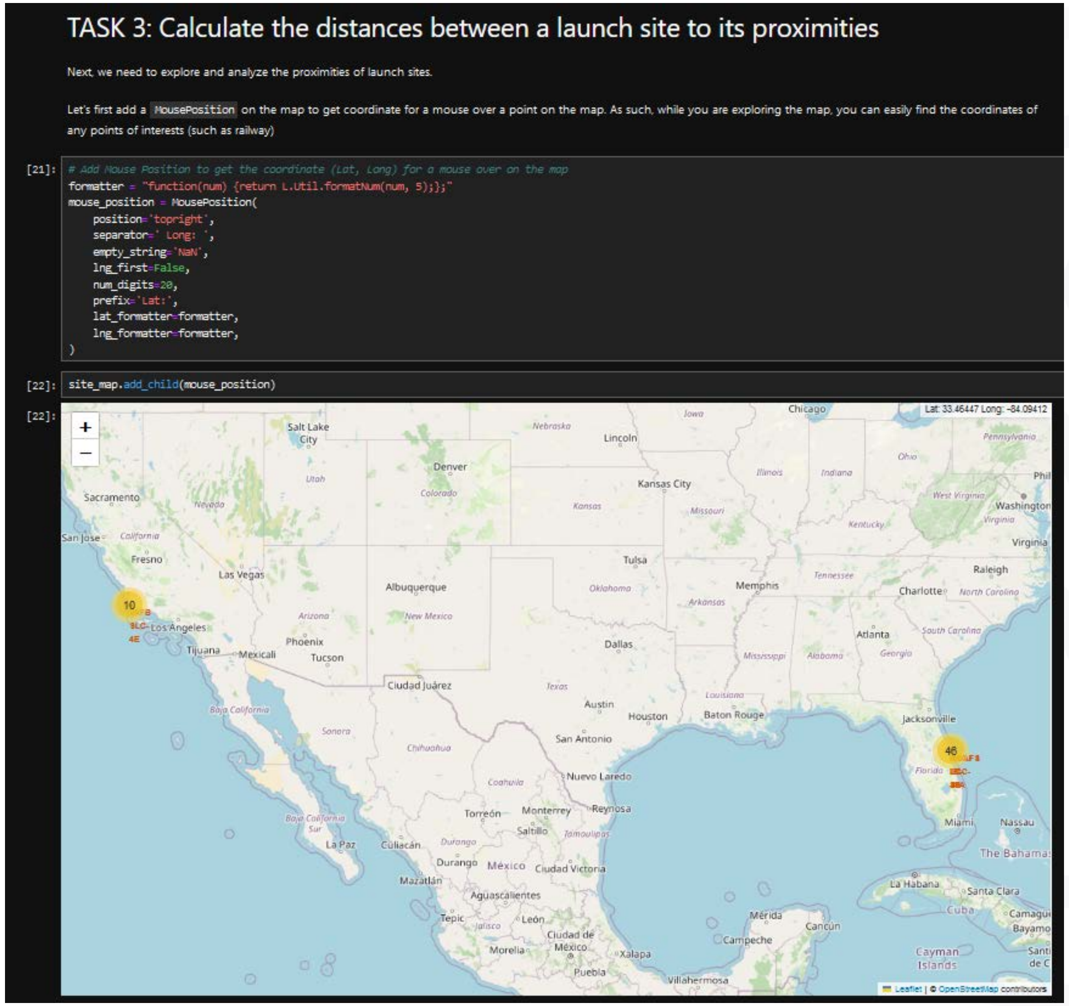
Landing_Outcome	Count
No attempt	10
Success (drone ship)	5
Failure (drone ship)	5
Success (ground pad)	3
Controlled (ocean)	3
Uncontrolled (ocean)	2
Failure (parachute)	2
Precluded (drone ship)	1

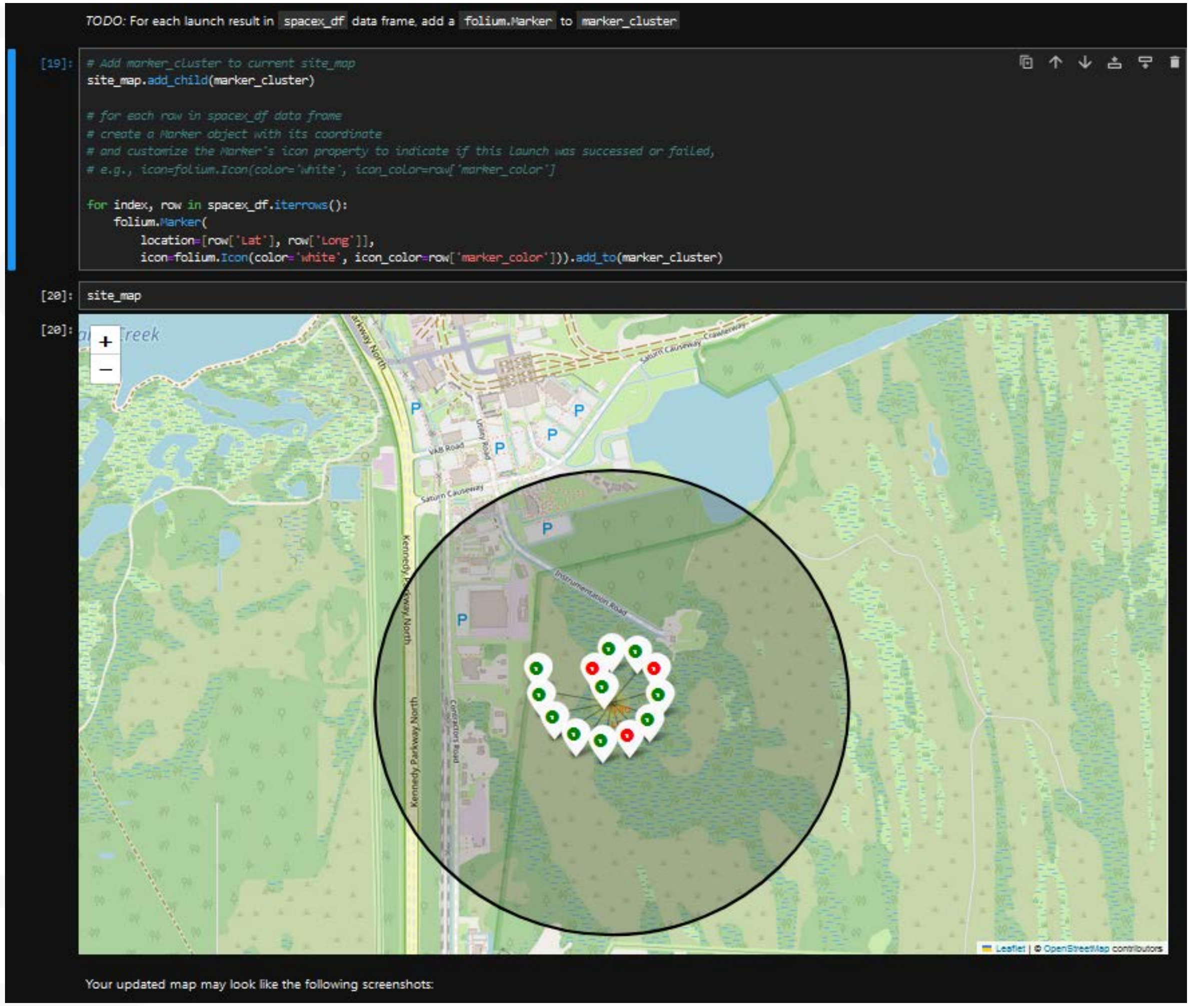
RESULTS
(Interactive Map with Folium)





RESULTS
(Interactive Map with Folium)





RESULTS
(Interactive Map with Folium)

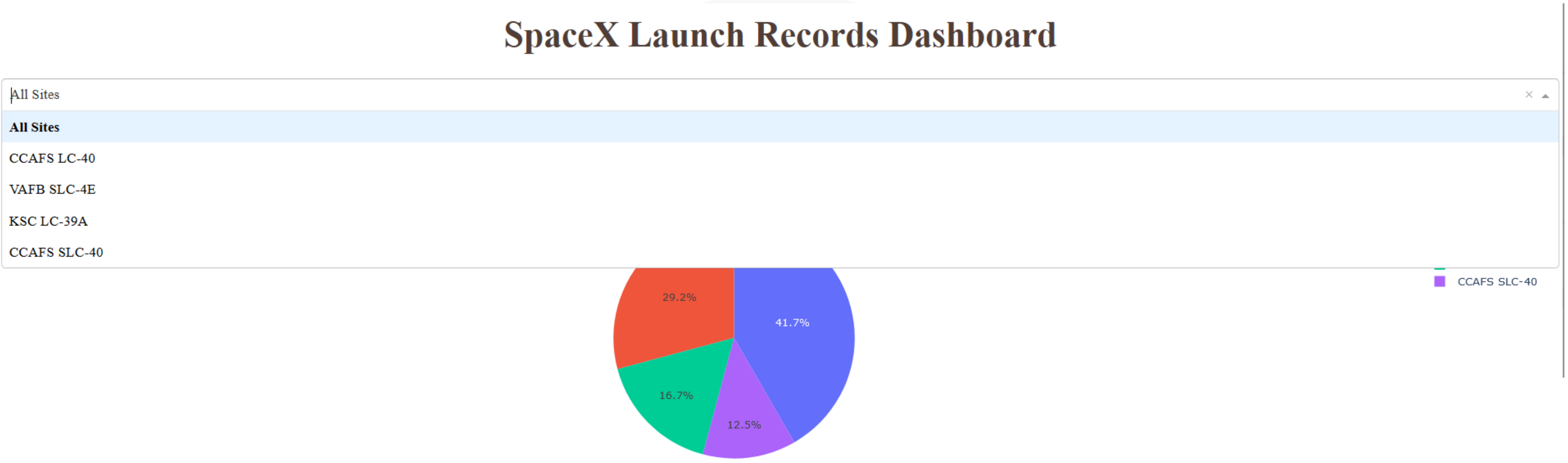
TODO: Draw a `PolyLine` between a launch site to the selected coastline point

```
[29]: distance_coordinates = [[spacex_df.loc[0]['Lat'], spacex_df.loc[0]['Long']], coast_line]
      distance_coordinates

[29]: [[28.56230197, -80.57735648], [28.56392, -80.56758]]

[30]: # Create a `folium.PolyLine` object using the coastline coordinates and launch site coordinate

      lines=folium.PolyLine(locations=distance_coordinates, weight=1)
      site_map.add_child(lines)

[30]:
```

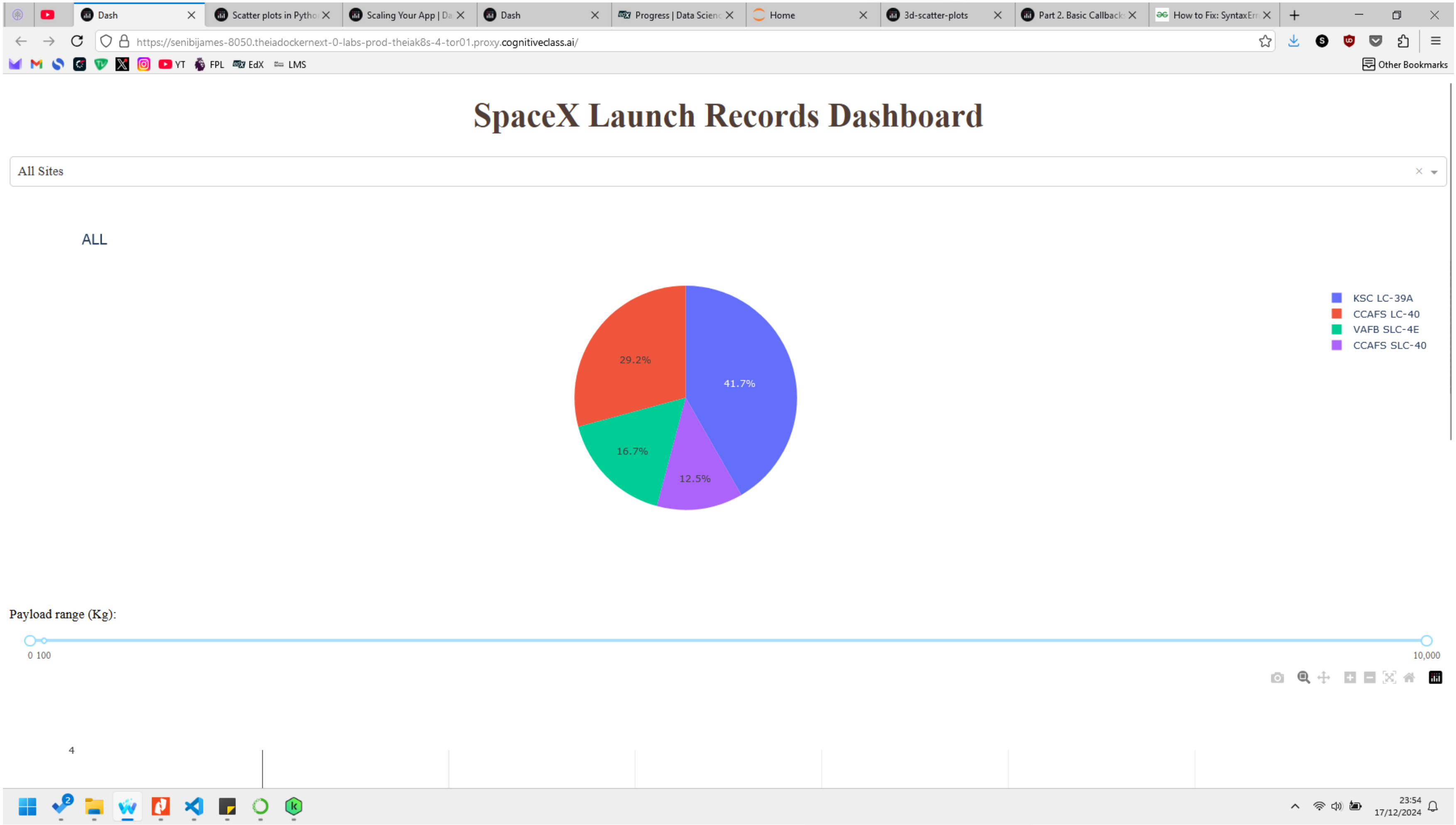

```
[32]: # Create a marker with distance to a closest city, railway, highway, etc.
      titusville_city = [28.60984, -80.81268]
      railway = [28.57338, -80.654]
      highway = [28.57357, -80.65553]

[33]: #Distance from Launch Site to Titusville City
      distance_titusville_city = calculate_distance(unique_df.loc[2]['Lat'], spacex_df.loc[2]['Long'], titusville_city[0], titusville_city[1])
      print ('Titusville City is ', distance_titusville_city, 'km away!')

      Titusville City is  23.34052876286819 km away!

[34]: #Distance from Launch Site to Railway
      distance_railway = calculate_distance(unique_df.loc[2]['Lat'], unique_df.loc[2]['Long'], railway[0], railway[1])
      print ('Railway is ', distance_railway, 'km away!')

      Railway is  0.6941473717803225 km away!

[35]: # Draw a line between the highway marker to the launch site
      highway_dist_coord = [[unique_df.loc[2]['Lat'], unique_df.loc[2]['Long']], highway]

      lines=folium.PolyLine(locations=highway_dist_coord, weight=1)
      site_map.add_child(lines)

[35]:
```

After you plot distance lines to the proximities, you can answer the following questions easily:

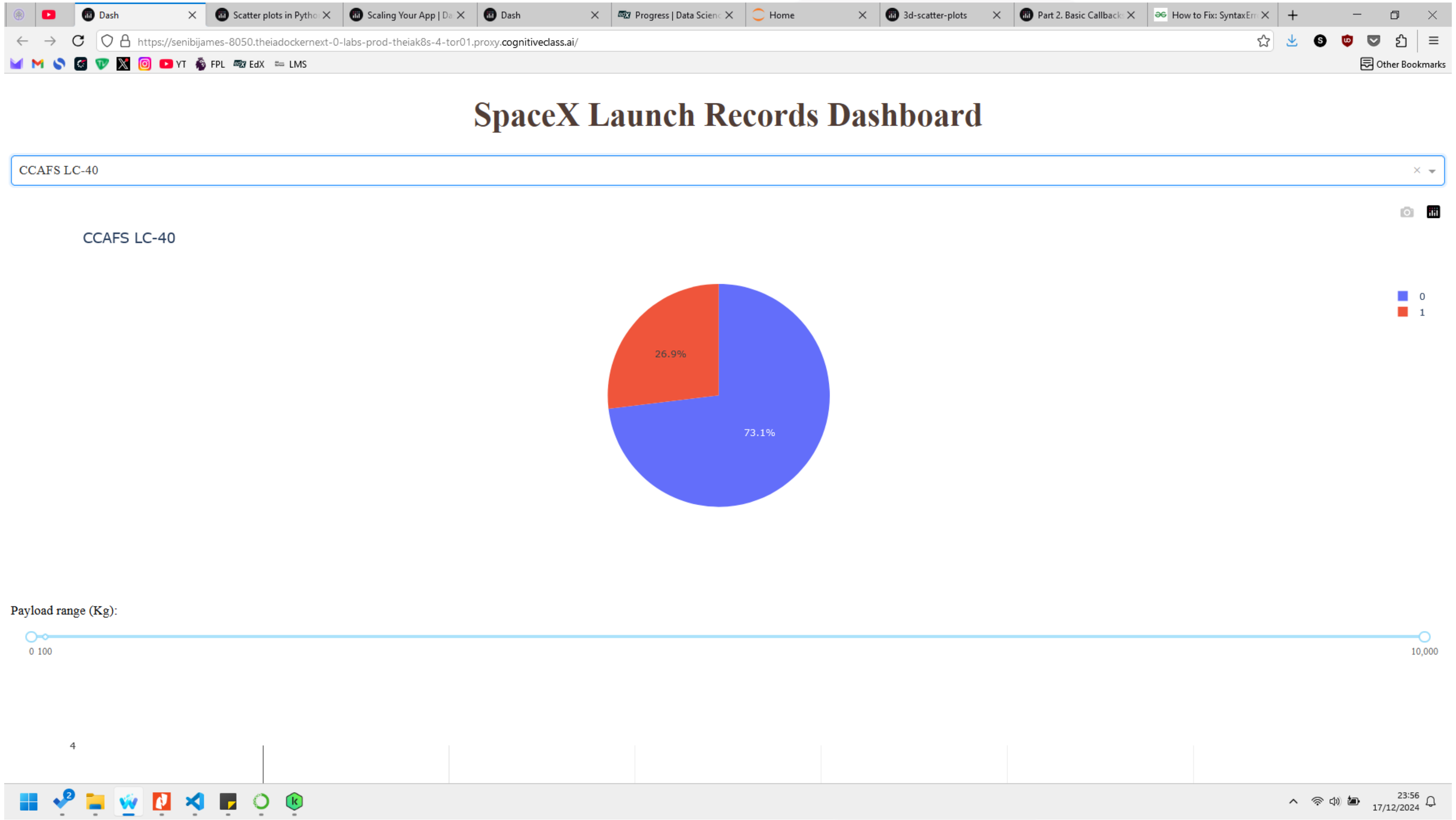
RESULTS
(Plotly Dash Dashboard)



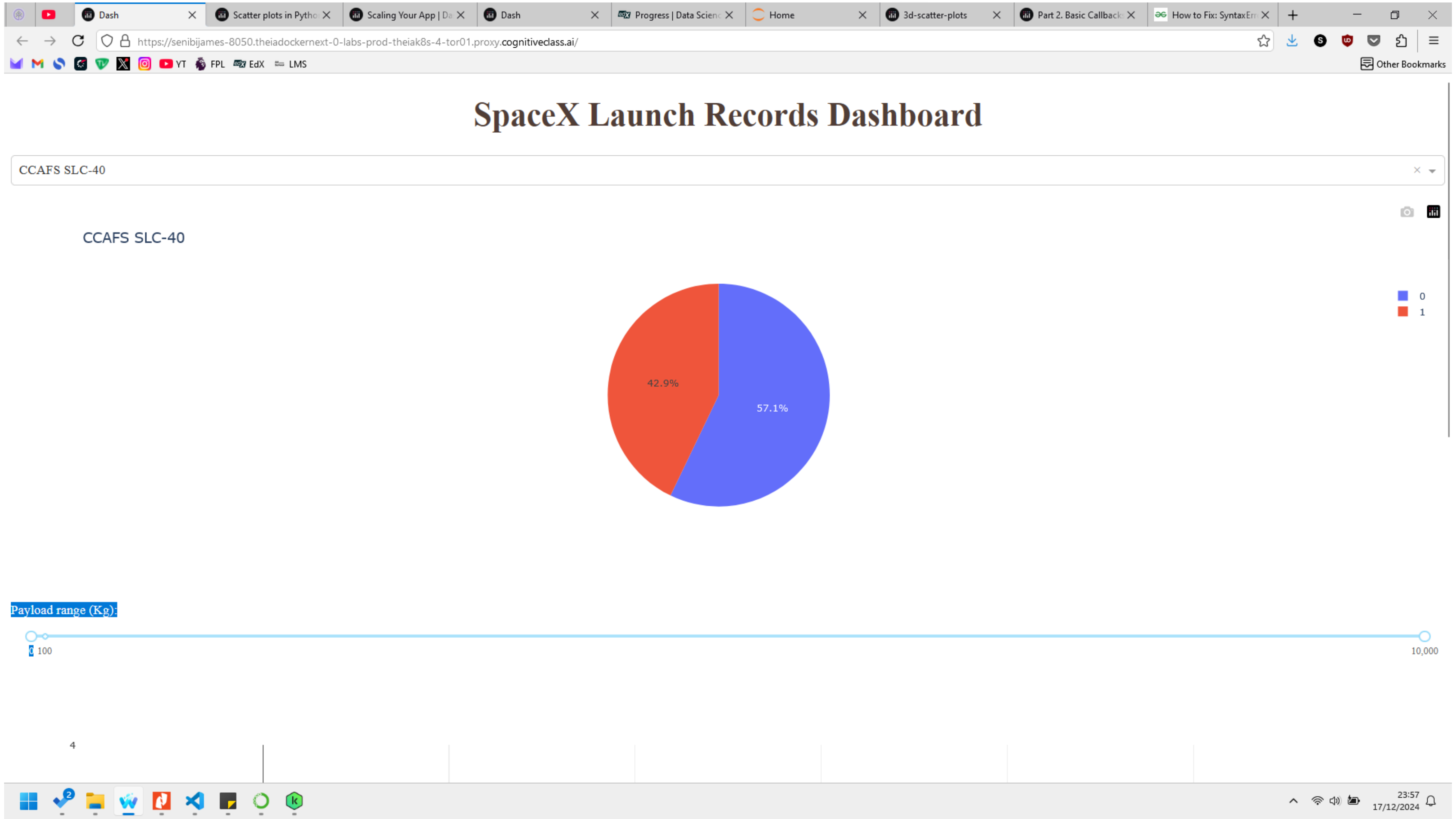
RESULTS
(Plotly Dash Dashboard)



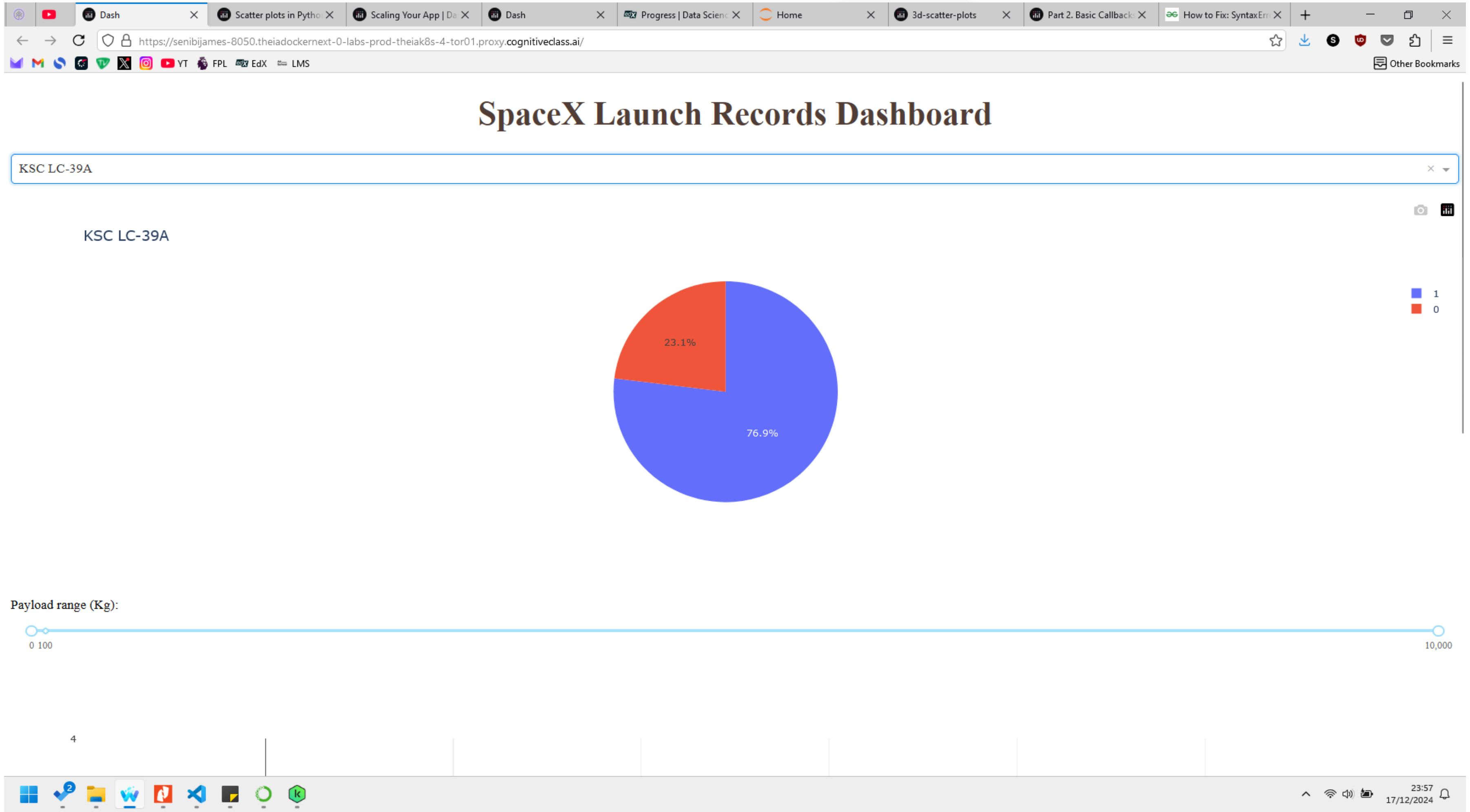
RESULTS
(Plotly Dash Dashboard)



RESULTS
(Plotly Dash Dashboard)



RESULTS
(Plotly Dash Dashboard)



RESULTS
(Predictive Analysis - Classification)

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']).

[35]:
```
p = data['Class'].to_numpy()
Y = pd.Series(p)
```

TASK 2

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

[38]:
```
# students get this
X = preprocessing.StandardScaler().fit(X).transform(X)
```

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`

[42]:
```
X_train, X_test, Y_train, Y_test = train_test_split( X, Y, test_size=0.2, random_state=4)
print ('Train set:', X_train.shape,  Y_train.shape)
print ('Test set:', X_test.shape,  Y_test.shape)
```

```
Train set: (72, 83) (72,)
Test set: (18, 83) (18,)
```

we can see we only have 18 test samples.

[43]:
```
Y_test.shape
```

[43]: (18,)

RESULTS
(Predictive Analysis - Logistics Regression)

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`.

[45]:
```
parameters ={'C':[0.01,0.1,1],
             'penalty':['l2'],
             'solver':['lbfgs']}
```

[59]:
```
parameters ={"C":[0.01,0.1,1],'penalty':['l2'], 'solver':['lbfgs']}# l1 lasso l2 ridge

#Create a logistic regression object
lr=LogisticRegression()
#Create a GridSearchCV object
logreg_cv  = GridSearchCV(lr, parameters, cv=10)
#Fit the object to the GridSearch with dictionary
bestLR =  logreg_cv.fit(X_train, Y_train)
bestLR
```

[59]:
```
GridSearchCV
best_estimator_: LogisticRegression
LogisticRegression
```

[60]:
```
logreg_cv.best_estimator_
```

[60]:
```
LogisticRegression
LogisticRegression(C=0.01)
```

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_`.

[61]:
```
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 : 0.8357142857142857
```

RESULTS
(Predictive Analysis – Logistics Regression)

TASK 5

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

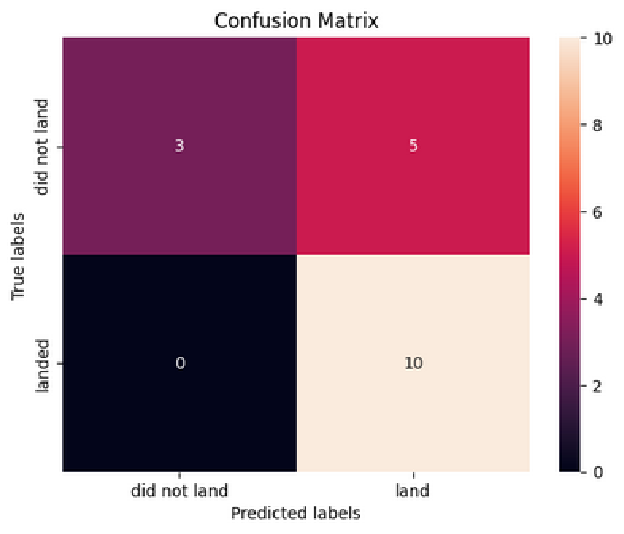
[67]:
```
logreg_cv.score(X_test, Y_test)
```

[67]: 0.7222222222222222

bestLR.score

Lets look at the confusion matrix:

[68]:
```
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 problem is false positives.

RESULTS
(Predictive Analysis - Support Vector Machine)

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`.

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

```
[83]: #Create a GridSearchCV object
      svm_cv  = GridSearchCV(svm, parameters, cv=10)
      #Fit the object to the GridSearch with dictionary
      best_svm = svm_cv.fit(X_train, Y_train)
      #svm_cv
```

```
[84]: 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 : 0.8625
```

RESULTS
(Predictive Analysis – Support Vector Machine)

TASK 7

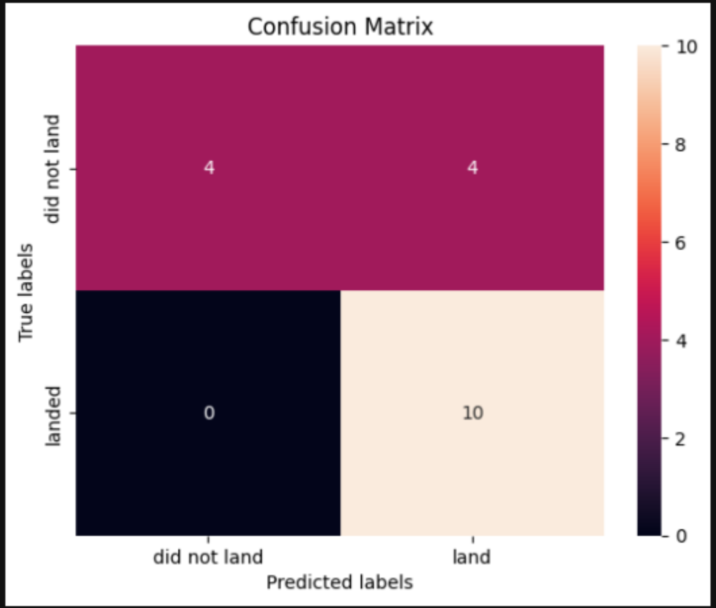
Calculate the accuracy on the test data using the method `score` :

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

```
[85]: 0.7777777777777778
```

We can plot the confusion matrix

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



RESULTS
(Predictive Analysis – Decision Tree)

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`.

[89]:
```
parameters1 = {'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()
```

[1]:
```
#Create a GridSearchCV object
#tree_cv  = GridSearchCV(tree, parameters1, cv=10)
#Fit the object to the GridSearch with dictionary
#best_tree = tree_cv.fit(X_train, Y_train)
#tree_cv
```

[91]:
```
print("tuned hpyerparameters :(best parameters) ",tree_cv.best_params_)
print("accuracy :",tree_cv.best_score_)
```

```
tuned hpyerparameters :(best parameters)  {'criterion': 'gini', 'max_depth': 10, 'max_features': 'sqrt', 'min_samples_leaf': 1, 'min_samples_split': 10, 'sp
litter': 'best'}
accuracy : 0.9053571428571429
```

RESULTS
(Predictive Analysis – Decision Tree)

TASK 9

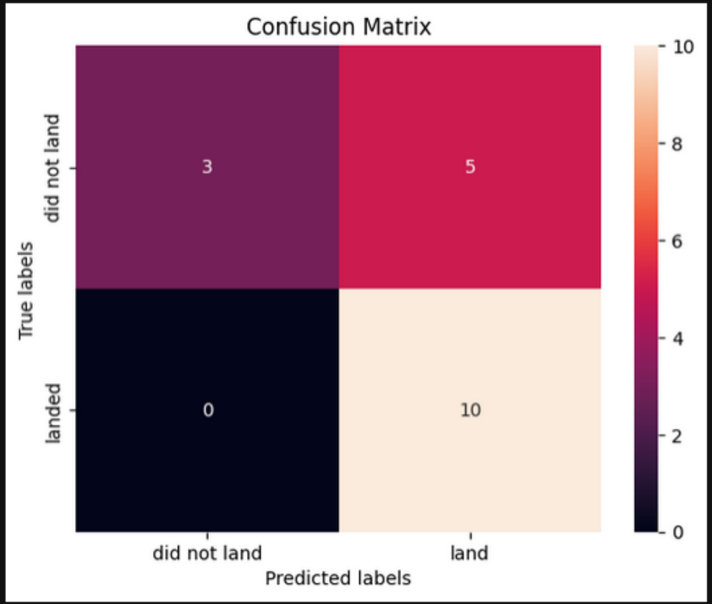
Calculate the accuracy of tree_cv on the test data using the method `score`:

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

```
[92]: 0.7222222222222222
```

We can plot the confusion matrix

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



RESULTS
(Predictive Analysis - K-Nearest Neighbor)

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`.

[100]:
```
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()
```

[101]:
```
#Create a GridSearchCV object
knn_cv  = GridSearchCV(KNN, parameters, cv=10)
#Fit the object to the GridSearch with dictionary
best_knn = knn_cv.fit(X_train, Y_train)
knn_cv
```

[101]:
```
GridSearchCV
best_estimator_: KNeighborsClassifier
KNeighborsClassifier
```

[102]:
```
print("tuned hpyerparameters :(best parameters) ",knn_cv.best_params_)
print("accuracy :",knn_cv.best_score_)
```

```
tuned hpyerparameters :(best parameters)  {'algorithm': 'auto', 'n_neighbors': 4, 'p': 1}
accuracy : 0.8767857142857143
```

RESULTS
(Predictive Analysis – K-Nearest Neighbor)

TASK 11

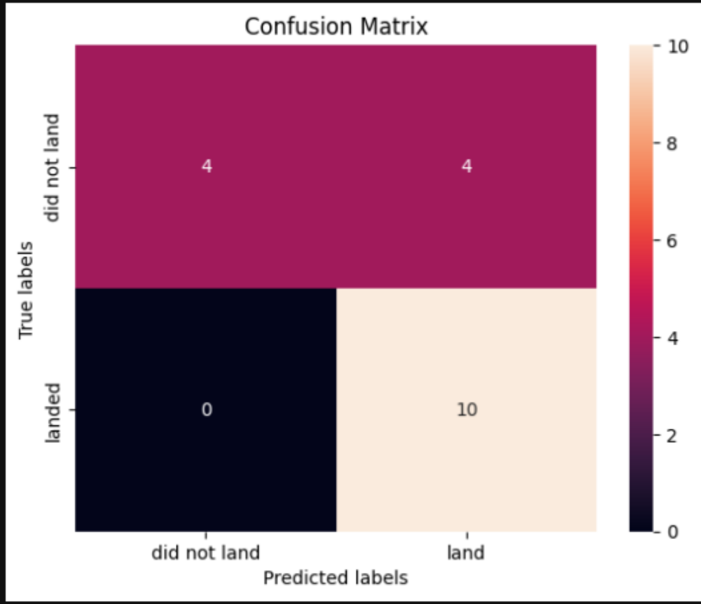
Calculate the accuracy of knn_cv on the test data using the method `score` :

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

```
[103]: 0.7777777777777778
```

We can plot the confusion matrix

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



CONCLUSION

- There is a higher chance of success with increased number of flights per launch site.
- There is a 100% chance of success at launch site CCAFS SLC-40 for Payload mass over 10,000kg.
- KSC LC-39A was the most used Space-X launch site while CCAFS SLC-40 was the least used.

CONCLUSION

- ES-L1, GEO, HEO and SSO are ideal orbit candidates with 100% success rate.
- For Flights number over 80, all orbits records 100% success rates. This might be due to re-adjusted parameters over time.
- Successful launches generally improve over time.

CONCLUSION

- There is close proximity between launch sites and coastal areas, highways and railways, but little or no proximity to cities.
- Decision Trees Prediction Model is the most accurate model for this scenario with an accuracy of over 80%
- Generally there is an estimated 80% chances of successfully recovering the first stage across all launch sites.