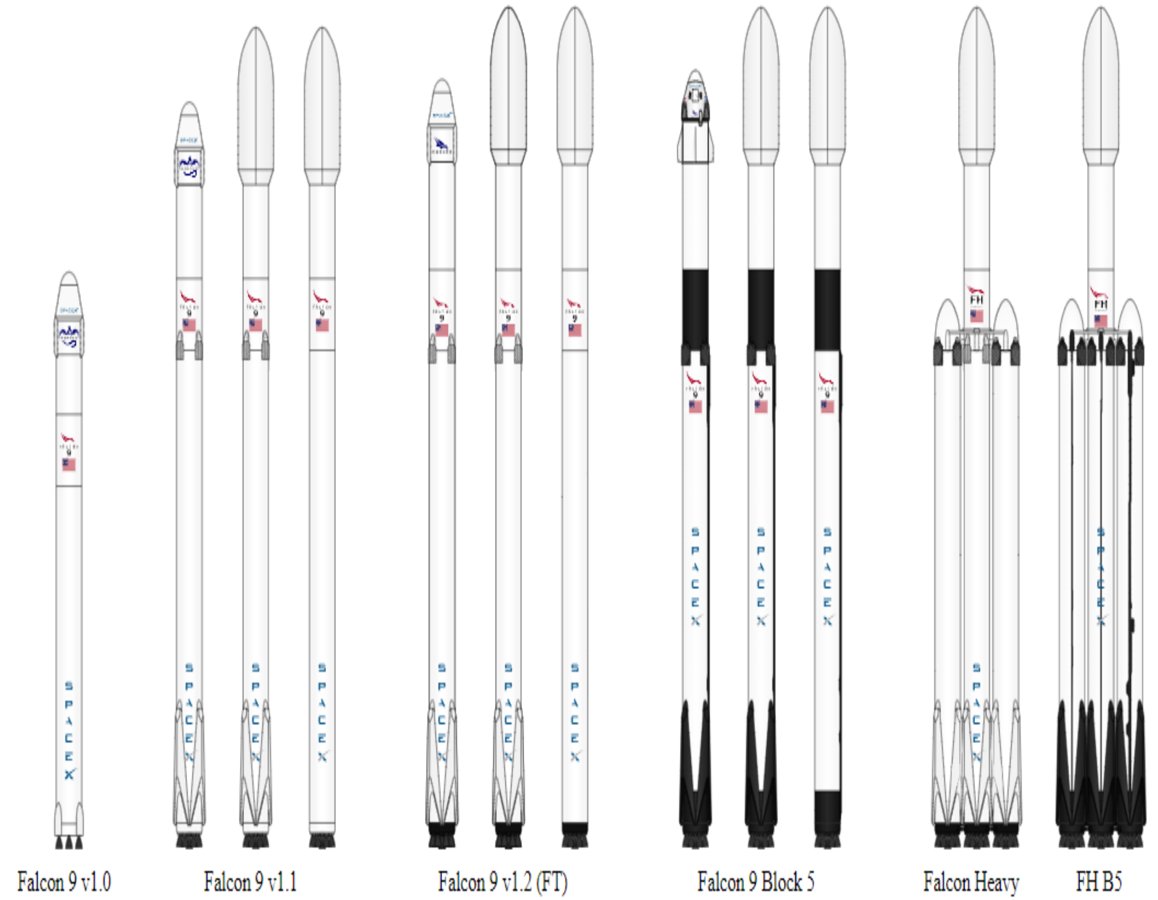
SpaceX: An Analysis of Falcon Rocket Launch



NAME:
Phetole Ramatapa

DATE:
January 2024

IBM Developer

SKILLS NETWORK

OUTLINE

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

EXECUTIVE SUMMARY

- An Outline Of Methodologies
  - Data Collection
  - Data Wrangling
  - Exploratory Data Analysis with Data Visualisation
  - Exploratory Data Analysis with SQL
  - Create an Interactive Folium Map
  - Create a Dashboard using Plotly Dash
  - Predictive Analysis

- Summary of Results from:
  - Exploratory Data Analysis
  - Interactive Analysis
  - Predictive Analysis

INTRODUCTION

- **Point 1: Initial phase**
  - Collect and Clean Data
  - Scrap the web for data using Beautiful Soup
  - Data Wrangling – Exploratory Data Analysis
- **Point 2: Exploratory Data Analysis using SQL**
  - Create datasets to sort out success and failures
  - Payloads masse and Site Locations
- **Point 3: Using Folium**
  - Show launch sites using markers (different colors)
  - Use markers to distinguish fail/success launches
- **Point 4: Using Bash**
  - Plot Dashboard – Pie Chart
  - Plot Dashboard – Drop Downs
- **Point 5: Predictive Analysis**
  - Data Standardization and Train/Test method
  - Plot Confusion Matrix

METHODOLOGY

- Data Collection from the SpaceX Table
- Wrangling data
- Data Exploration with SQL and Data Visualization
- Folium and Plotly Dash are used as essential tools
- Predictive Analysis modeling to determine possible launch and landing outcomes

Data Collection

(Data Scraping)

IBM Developer

SKILLS NETWORK

Web Scraping: Step by Step *

- Load needed Software & Libraries
- Getting Response from HTML
- Creating BeautifulSoup Object
- Finding Tables
- Getting Column Names
- Create Dictionary
- Append Data to Keys
- Convert Dictionary to Dataframe
- Dataframe to .CSV file

TASK 1: Request the Falcon9 Launch Wiki page from its URL

First, let's perform an HTTP GET method to request the Falcon9 Launch HTML page, as an HTTP response.

```
# use requests.get() method with the provided static_url
# assign the response to a object
data  = requests.get(static_url).text
```

Create a `BeautifulSoup` object from the HTML `response`

```
# Use BeautifulSoup() to create a BeautifulSoup object from a response text content
soup = BeautifulSoup(data, 'html5lib')
```

Print the page title to verify if the `BeautifulSoup` object was created properly

```
# Use soup.title attribute
print(soup.title)
```

```
<title>List of Falcon 9 and Falcon Heavy launches - Wikipedia</title>
```

TASK 2: Extract all column/variable names from the HTML table header

Next, we want to collect all relevant column names from the HTML table header

```
column_names = []

labels = first_launch_table.find_all('th')
for label in labels:
    name = extract_column_from_header(label)
   # header = str(label.text).strip()
    #header = str(header).split("($)Footnote", 1)[0]
    if name != None:
       if len(name) > 0:
           column_names.append(name)
```

```
print(column_names)
```

['Flight No.', 'Date and time ( )', 'Launch site', 'Payload', 'Payload mass', 'Orbit', 'Customer', 'Launch outcome']

```
#### TASK 3: Create a data frame by parsing the launch HTML tables
```

```python
launch_dict= dict.fromkeys(column_names)

# Remove an irrelvant column
del launch_dict['Date and time ( )']

# Let's initial the launch_dict with each value to be an empty list
launch_dict['Flight No.'] = []
launch_dict['Launch site'] = []
launch_dict['Payload'] = []
launch_dict['Payload mass'] = []
launch_dict['Orbit'] = []
launch_dict['Customer'] = []
launch_dict['Launch outcome'] = []
# Added some new columns
launch_dict['Version Booster']=[]
launch_dict['Booster landing']=[]
launch_dict['Date']=[]
launch_dict['Time']=[]
```

Task 3: The CSV File: Bottom Half

...	...	...	...	...	...	...	...	...	...	...	...
116	117	CCSFS	Starlink	15,600 kg	LEO	SpaceX	Success\n	F9 B5B1051.10	Success	9 May 2021	06:42
117	118	KSC	Starlink	~14,000 kg	LEO	SpaceX Capella Space and Tyvak	Success\n	F9 B5B1058.8	Success	15 May 2021	22:56
118	119	CCSFS	Starlink	15,600 kg	LEO	SpaceX	Success\n	F9 B5B1063.2	Success	26 May 2021	18:59
119	120	KSC	SpaceX CRS-22	3,328 kg	LEO	NASA (CRS)	Success\n	F9 B5B1067.1	Success	3 June 2021	17:29
120	121	CCSFS	SXM-8	7,000 kg	GTO	Sirius XM	Success\n	F9 B5	Success	6 June 2021	04:26

121 rows × 11 columns

```
df.to_csv('spacex_web_scraped.csv', index=False)
```

Data Wrangling

IBM Developer

SKILLS NETWORK

```
### TASK 1: Calculate the number of launches on each site
```

```python
# Apply value_counts() on column LaunchSite
df.LaunchSite.value_counts()
```

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

```
### TASK 2: Calculate the number and occurrence of each orbit
```

```python
# Apply value_counts on Orbit column
df.Orbit.value_counts()
```

```
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
```

```
### TASK 3: Calculate the number and occurence of mission outcome of the orbits
```

```python
# landing_outcomes = values on Outcome column
landing_outcomes = df.Outcome.value_counts()
landing_outcomes
```

```
True ASDS      41
None None      19
True RTLS      14
False ASDS      6
True Ocean      5
False Ocean     2
None ASDS       2
False RTLS      1
Name: Outcome, dtype: int64
```

```python
for i,outcome in enumerate(landing_outcomes.keys()):
    print(i,outcome)
```

```
0 True ASDS
1 None None
2 True RTLS
3 False ASDS
4 True Ocean
5 False Ocean
6 None ASDS
7 False RTLS
```

```python
bad_outcomes=set(landing_outcomes.keys()[[1,3,5,6,7]])
bad_outcomes
```

```
{'False ASDS', 'False Ocean', 'False RTLS', 'None ASDS', 'None None'}
```

RESULTS

•CCAFS SLL 40 site experienced a higher volume of launches as time progresses compared to other launch site
•CCAFS SLL 40 has a significantly higher success rate with a lighter payload
•KSL LC 39A is singularly more successful as a launch site
•VLEO has more launches in later years compared to other orbits
•The GEO, HEO, SSO and ES-LL have more mission launches as more launches take place and years went by
•SVM, KNN & Logistics Regression models are the best in terms of prediction accuracy for the given dataset

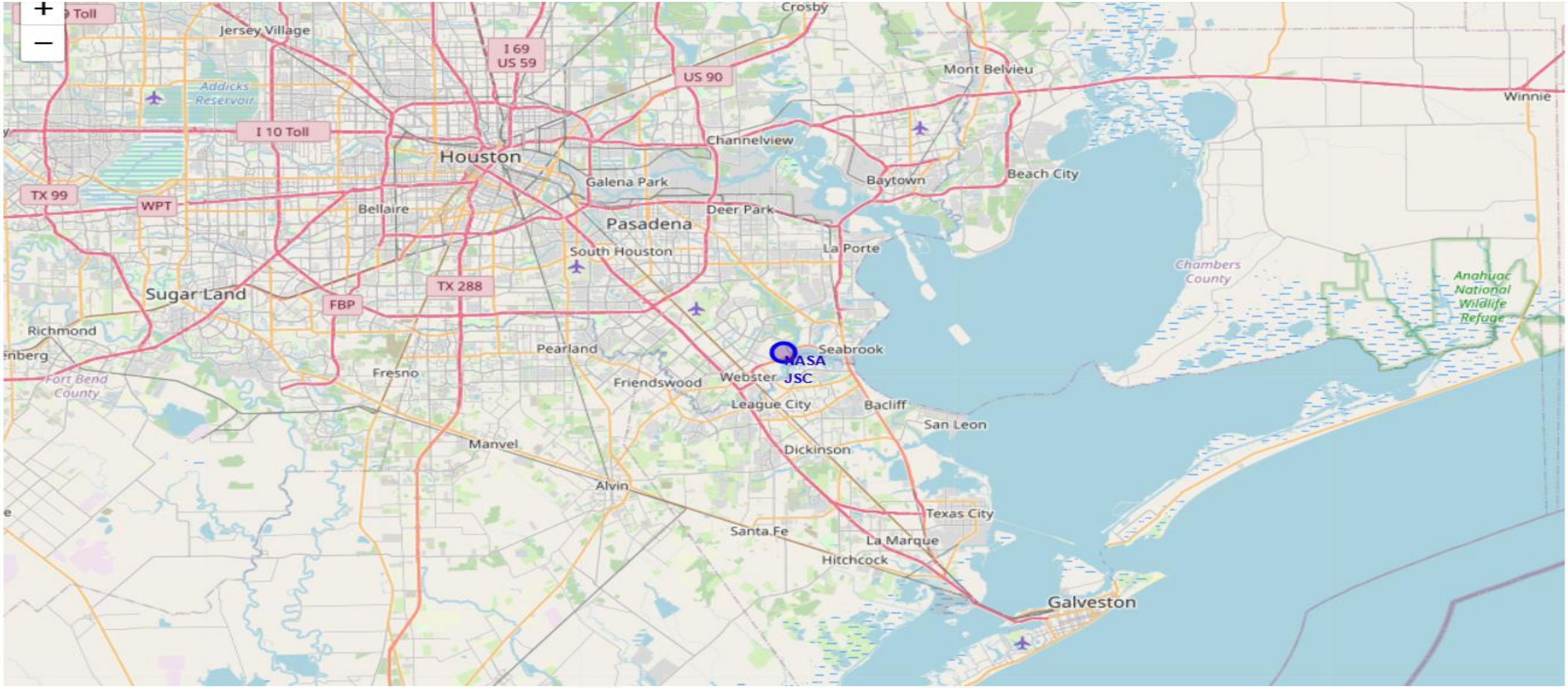
EDA with Data Visualisation (Folium Maps)

IBM Developer

SKILLS NETWORK


80%
75%
50%
25%
A
B
C

Houston launch site (marked in BLUE). Strategically located closer to the ocean and on the east of the country.

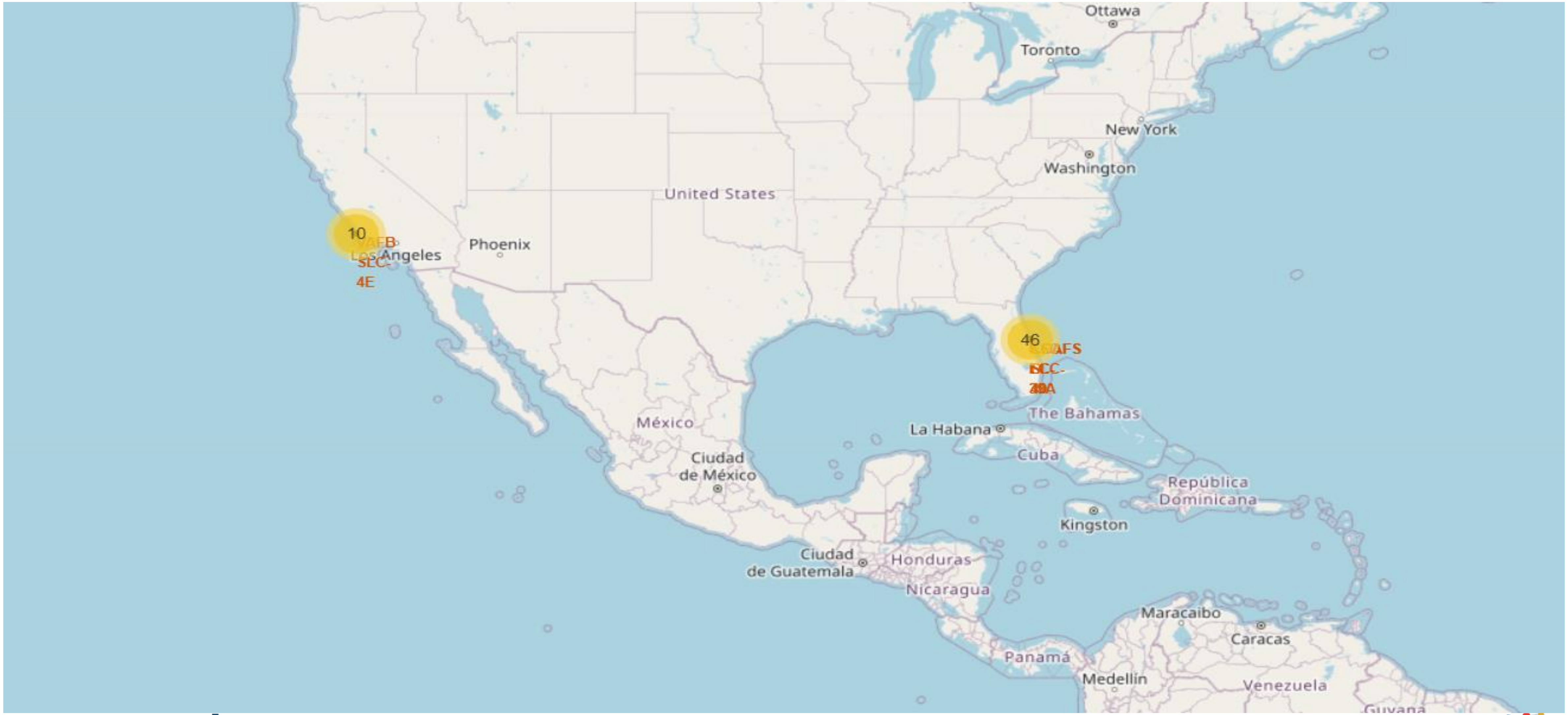


Markers indicating the successful and failed missions. Also showing basic infrastructure, e.g. roads & railway lines.

Launch Site Success Rate

- **Green** markers for successful launches
- **Red** markers for unsuccessful launches
- Launch site **CCAFS SLC-40** has a **3/7 success rate** (**42.9%**)

Number of launches and coastal locations on opposite sites of continental USA, showing number of Launches per site



IBM Developer

SKILLS NETWORK

Folium Map showing infrastructure, number of launches per site and distance between launch sites

Exploratory Data Analysis with SQL

IBM Developer

SKILLS NETWORK


albums
id
artist_id
name
cover
created_at
updated_at
SQL
songs
id
album_id
artist_id
title
artists
id
name

SQL as an Exploratory Data Analysis Tool

The Following Are Some of the Discoveries made from Using SQL:

- Sum Total of All launched Payloads
- A List of All Unique Launch Sites
- List of All Successful Launches
- Successful Launches within a prescribed Period

  Typical Example:

  %sql SELECT * FROM SPACEXTBL WHERE Mission_Outcome LIKE / ‘Success%’ AND (DATE BETWEEN’2015-01-01 AND ‘2015-12-31’) /

  ORDER BY Date DESC

Removing NULLS from the Table

```
[8]: ### code is added to remove blank rows from table

%sql create table SPACEXTABLE as select * from SPACEXTBL where Date is not null
```

```
 * sqlite:///my_data1.db
(sqlite3.OperationalError) table SPACEXTABLE already exists
[SQL: create table SPACEXTABLE as select * from SPACEXTBL where Date is not null]
(Background on this error at: http://sqlalche.me/e/e3q8)
```

[8]:
```
### Task 1: Display the names of the unique launch sites in the space mission
```

[29]:
```
%sql SELECT DISTINCT(LAUNCH_SITE) FROM SPACEXTBL;
```

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

[29]:

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'
```

```
%sql SELECT * FROM SPACEXTBL WHERE (LAUNCH_SITE) LIKE 'CCA%' LIMIT 5;
```

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

Date	Time (UTC)	Booster_Version	Launch_Site	Payload	PAYLOAD_MASS__KG_	Orbit	Customer	Mission_Outcome	Landing_Outcome
2010-06-04	18:45:00	F9 v1.0 B0003	CCAFS LC-40	Dragon Spacecraft Qualification Unit	0	LEO	SpaceX	Success	Failure (parachute)
2010-12-08	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)
2012-05-22	7:44:00	F9 v1.0 B0005	CCAFS LC-40	Dragon demo flight C2	525	LEO (ISS)	NASA (COTS)	Success	No attempt
2012-10-08	0:35:00	F9 v1.0 B0006	CCAFS LC-40	SpaceX CRS-1	500	LEO (ISS)	NASA (CRS)	Success	No attempt
2013-03-01	15:10:00	F9 v1.0 B0007	CCAFS LC-40	SpaceX CRS-2	677	LEO (ISS)	NASA (CRS)	Success	No attempt

[20]:

```
### Task 3: Display the total payload mass carried by boosters launched by NASA (CRS)
```

[30]:

```
%sql SELECT SUM(PAYLOAD_MASS__KG_) AS payloadmass FROM SPACEXTBL;
```

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

[30]:

payloadmass
619967

[ ]:

[ ]:

[22]: ### Task 4: Display average payload mass carried by booster version F9 v1.1

[31]:
```
%sql SELECT AVG(PAYLOAD_MASS__KG_) AS payload_mass FROM SPACEXTBL;
```

* sqlite:///my_data1.db
Done.

[31]:

payload_mass
6138.287128712871

[25]:
```
### Task 5: List the date when the first succesful landing outcome in ground pad was acheived.
```

[32]:
```
%sql SELECT MIN(Date) FROM SPACEXTBL;
```

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

[32]:

MIN(Date)
2010-06-04

TASK 6

[27]:
```
### 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
```

[7]:
```
%sql SELECT BOOSTER_VERSION, PAYLOAD_MASS__KG_ FROM SPACEXTBL WHERE Mission_Outcome = 'Success' \
AND PAYLOAD_MASS__KG_ BETWEEN 4000 AND 6000;
```

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

[7]:

Booster_Version	PAYLOAD_MASS__KG_
F9 v1.1	4535
F9 v1.1 B1011	4428
F9 v1.1 B1014	4159
F9 v1.1 B1016	4707
F9 FT B1020	5271
F9 FT B1022	4696
F9 FT B1026	4600
F9 FT B1030	5600
F9 FT B1021.2	5300
F9 FT B1032.1	5300
F9 B4 B1040.1	4990
F9 FT B1031.2	5200
F9 FT B1032.2	4230
F9 B4 B1040.2	5384
F9 B5 B1046.2	5800
F9 B5 B1047.2	5300
F9 B5 B1046.3	4000
F9 B5 B1048.3	4850
F9 B5 B1051.2	4200
F9 B5B1060.1	4311
F9 B5 B1058.2	5500
F9 B5B1062.1	4311

[28]:

```
### Task 7: List the total number of successful and failure mission outcomes
```

[11]:

```
%sql SELECT COUNT(Mission_Outcome) AS Mission_Outcomes FROM SPACEXTBL WHERE "Mission_Outcome" = 'Success'
```

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

[11]:

Mission_Outcomes
98

[33]:

```
[ ]: ### Task 8: List the names of the booster_versions which have carried the maximum payload mass use a subquery
```

```
[57]: %sql SELECT BOOSTER_VERSION FROM SPACEXTBL \
WHERE PAYLOAD_MASS__KG_ = (SELECT MAX(PAYLOAD_MASS__KG_) FROM SPACEXTBL);
```

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

[57]:

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
F9 B5 B1060.3
F9 B5 B1049.7

[35]:
```
### 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.
```

[12]:
```
%sql SELECT * FROM SPACEXTBL WHERE Mission_Outcome LIKE 'Success%' AND (DATE BETWEEN '2015-01-01' AND '2015-12-31');
```

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

[12]:

Date	Time (UTC)	Booster_Version	Launch_Site	Payload	PAYLOAD_MASS__KG_	Orbit	Customer	Mission_Outcome	Landing_Outcome
2015-01-10	9:47:00	F9 v1.1 B1012	CCAFS LC-40	SpaceX CRS-5	2395	LEO (ISS)	NASA (CRS)	Success	Failure (drone ship)
2015-02-11	23:03:00	F9 v1.1 B1013	CCAFS LC-40	DSCOVR	570	HEO	U.S. Air Force NASA NOAA	Success	Controlled (ocean)
2015-03-02	3:50:00	F9 v1.1 B1014	CCAFS LC-40	ABS-3A Eutelsat 115 West B	4159	GTO	ABS Eutelsat	Success	No attempt
2015-04-14	20:10:00	F9 v1.1 B1015	CCAFS LC-40	SpaceX CRS-6	1898	LEO (ISS)	NASA (CRS)	Success	Failure (drone ship)
2015-04-27	23:03:00	F9 v1.1 B1016	CCAFS LC-40	Turkmen 52 / MonacoSAT	4707	GTO	Turkmenistan National Space Agency	Success	No attempt
2015-12-22	1:29:00	F9 FT B1019	CCAFS LC-40	OG2 Mission 2 11 Orbcomm-OG2 satellites	2034	LEO	Orbcomm	Success	Success (ground pad)

Task 10

```
%sql SELECT [Landing_Outcome], count(*) AS count_outcomes FROM SPACEXTBL \
WHERE DATE between '04-06-2010' and '20-03-2017' group by [Landing_Outcome] order by count_outcomes DESC;
```

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

Out[37]:

Landing _Outcome	count_outcomes
Success	20
No attempt	10
Success (drone ship)	8
Success (ground pad)	6
Failure (drone ship)	4
Failure	3
Controlled (ocean)	3
Failure (parachute)	2
No attempt	1

Create a Dashboard using Plotly Dash

IBM Developer

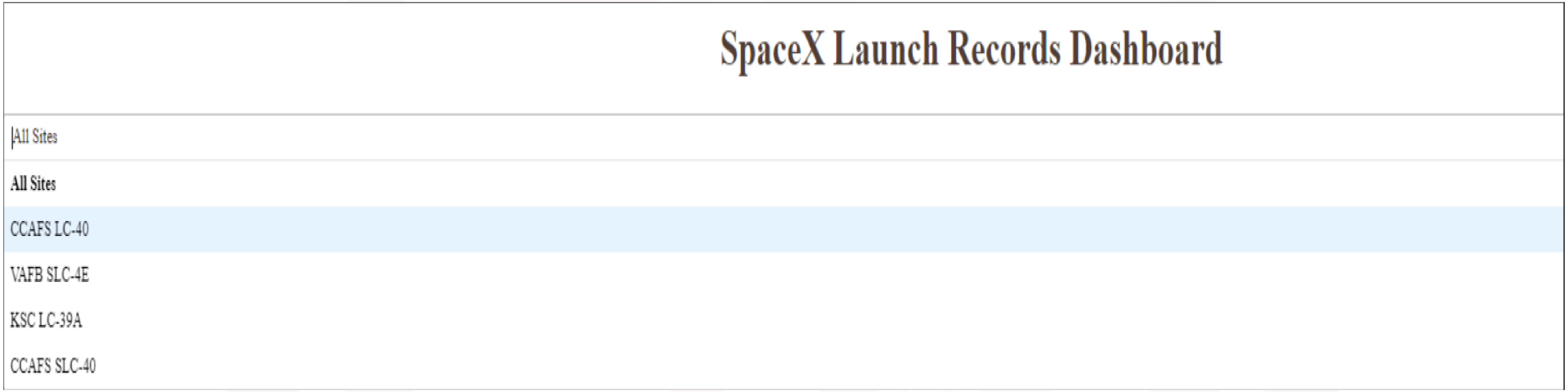
SKILLS NETWORK


PREDICTIVE MAINTENANCE DASHBOARD FOR WIND TURBINES
This Dashboard is focused on estimating the Remaining Useful Life (RUL) in wind turbines. RUL is defined as the time until the next fault.
plotly | Dash
System RUL Estimation (days)
15
RUL is estimated based on the readings for the last week.
WEC: ava windspeed
Start Period
End Period
Get New Data
Predict
Active Power [kW]
Active Power Available from Wind [kW]
Reactive Power [kVAR]
Wind Speed [m/s]
Blade Angle
3070.0
296.0
11.8
2.9

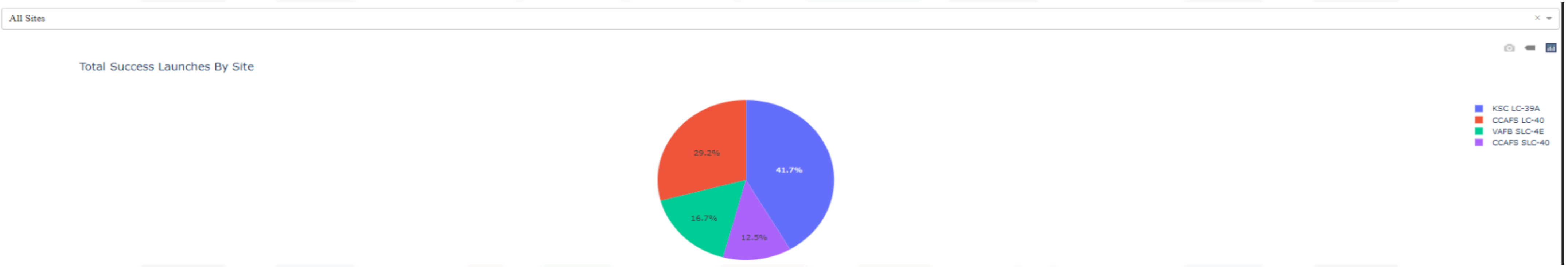
DASHBOARD

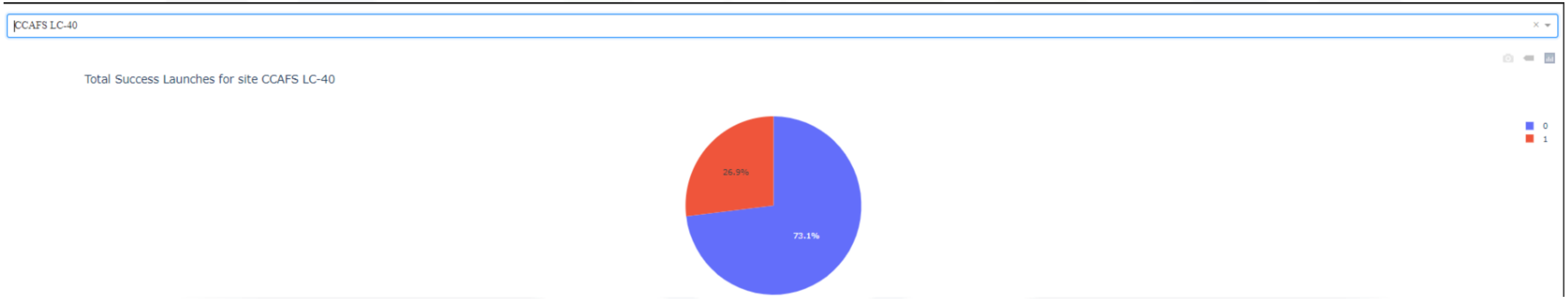
https://github.com/Pilwana/Dashboard.git

DASHBOARD TAB 1



Total Launches for All Site & Total Launches for the CCAFS LC-40 Site





Slider

The Correlation Between Sites & Success Rate

Predictive Analysis

IBM Developer

SKILLS NETWORK


MACHINE
LEARNING

```
### TASK 1
```

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

```
Y
```

```
array([0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1,
       1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1,
       1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1], dtype=int64)
```

```
### TASK 2
# Standardize the data in X then reassign it to the variable X
```

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

```
X = transform.fit_transform(X)
```

```
X= preprocessing.StandardScaler().fit(X).transform(X)
```

```
X[0:5]
```

```
array([[-1.71291154e+00, -5.29526321e-17, -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
```

```
### TASK 3
# X_train, X_test, Y_train, Y_test
```

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

```
Y_test.shape
```

```
(18,)
```

```
X_test.shape
```

```
(18, 83)
```

```
### TASK 4
```

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

```
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)
```

▸ GridSearchCV

▸ estimator: LogisticRegression

▸ LogisticRegression

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

```
tuned hpyerparameters :(best parameters)  {'C': 0.01, 'penalty': 'l2', 'solver': 'lbfgs'}
accuracy : 0.8464285714285713
```

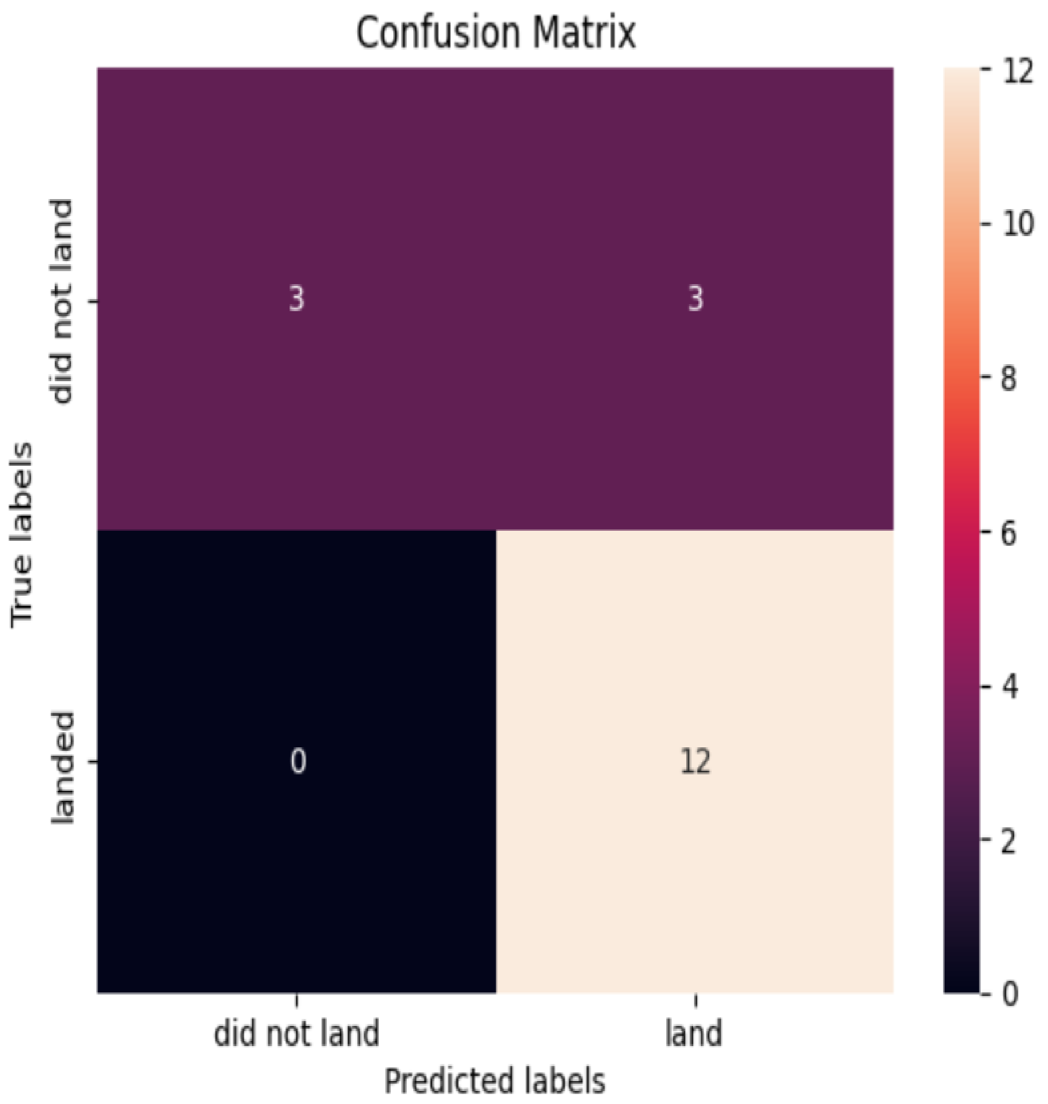
```
### TASK 5
```

```
## accuracy on the test data using the method score
("test set accuracy :",logreg_cv.score(X_test, Y_test))
```

```
('test set accuracy :', 0.8333333333333334)
```

```
## Lets look at the confusion matrix:


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



```
### TASK 6
```

```
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
▸ estimator: SVC
    ▸ SVC
```

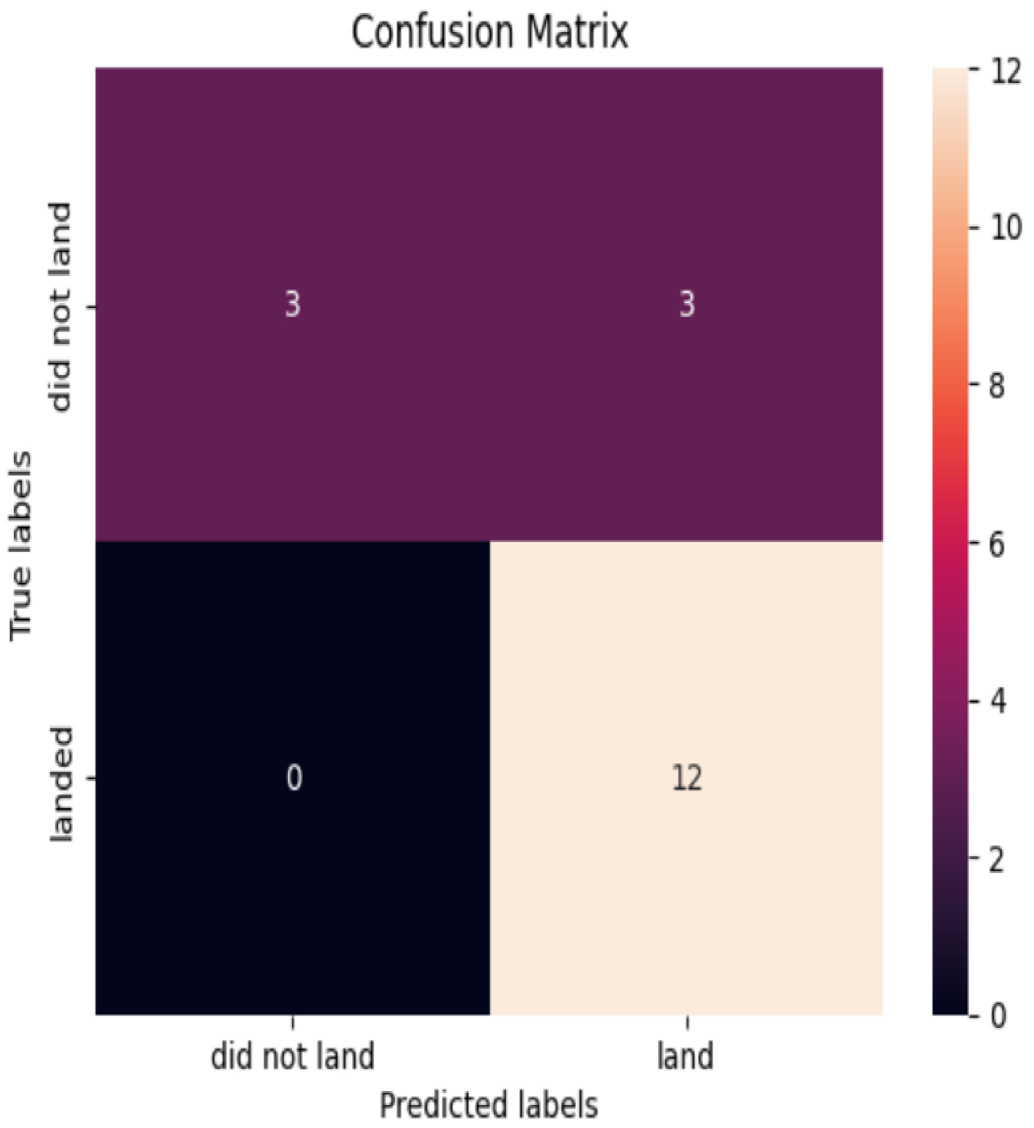
```
### TASK 7
```

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

print("test set accuracy :",svm_cv.score(X_test, Y_test))
```

```
test set accuracy : 0.8333333333333334
```

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



```
### TASK 8
```

```python
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()
```

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

```
▸ GridSearchCV
▸ estimator: DecisionTreeClassifier
    ▸ DecisionTreeClassifier
```

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

```
tuned hpyerparameters :(best parameters)  {'criterion': 'gini', 'max_depth': 18, 'max_features': 'sqrt', 'min_samples_leaf': 4, 'min_samples_split': 2, 'splitter': 'random'}
accuracy : 0.875
```

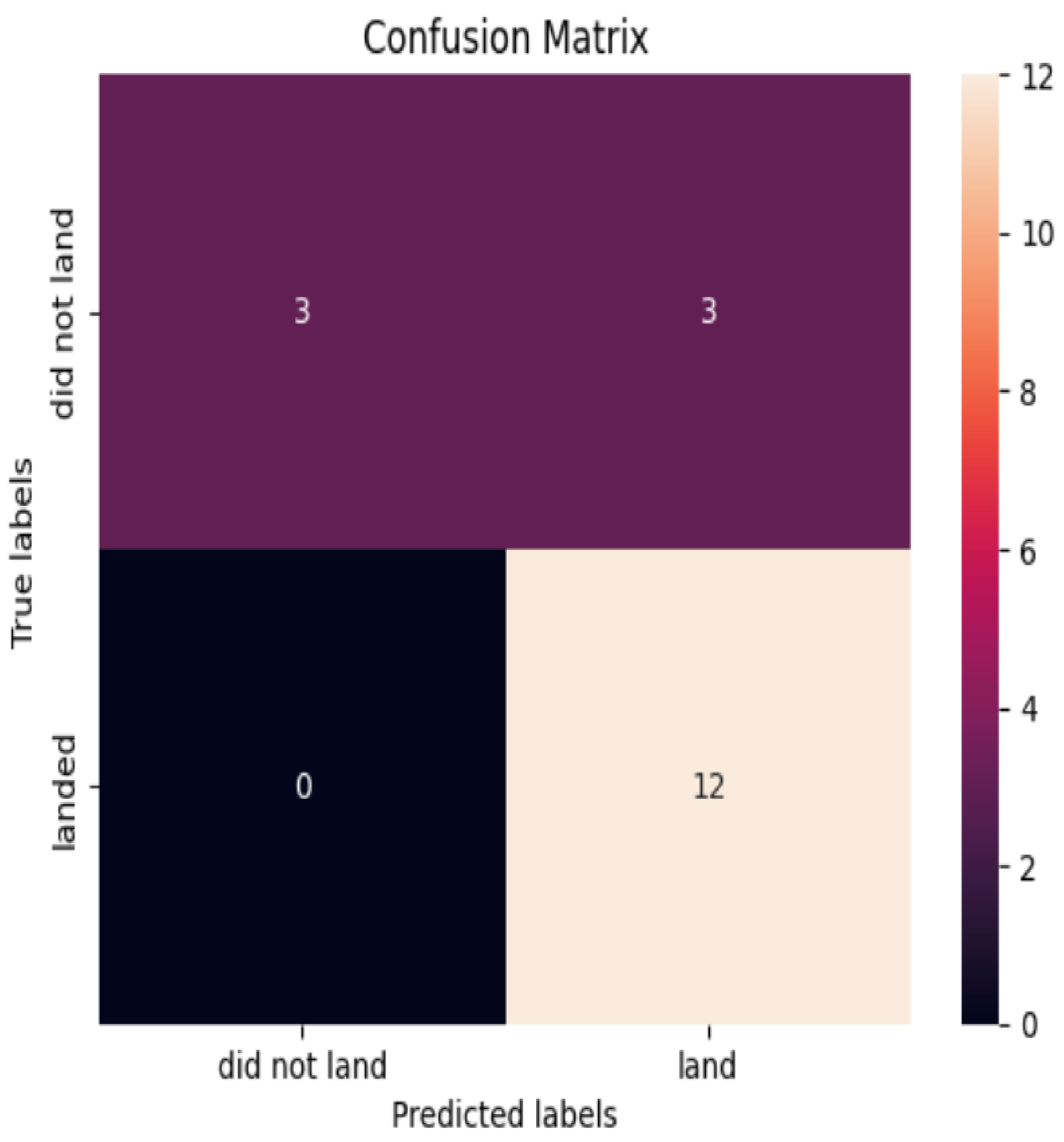
```
### TASK 9
```

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

print("test set accuracy :",tree_cv.score(X_test, Y_test))
```

```
test set accuracy : 0.8888888888888888
```

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



TASK 10

```
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
```

```
▸ GridSearchCV
▸ estimator: DecisionTreeClassifier
    ▸ DecisionTreeClassifier
```

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

```
tuned hpyerparameters :(best parameters)  {'criterion': 'gini', 'max_depth': 18, 'max_features': 'sqrt', 'min_samples
_leaf': 4, 'min_samples_split': 2, 'splitter': 'random'}
accuracy : 0.875
```

TASK 11

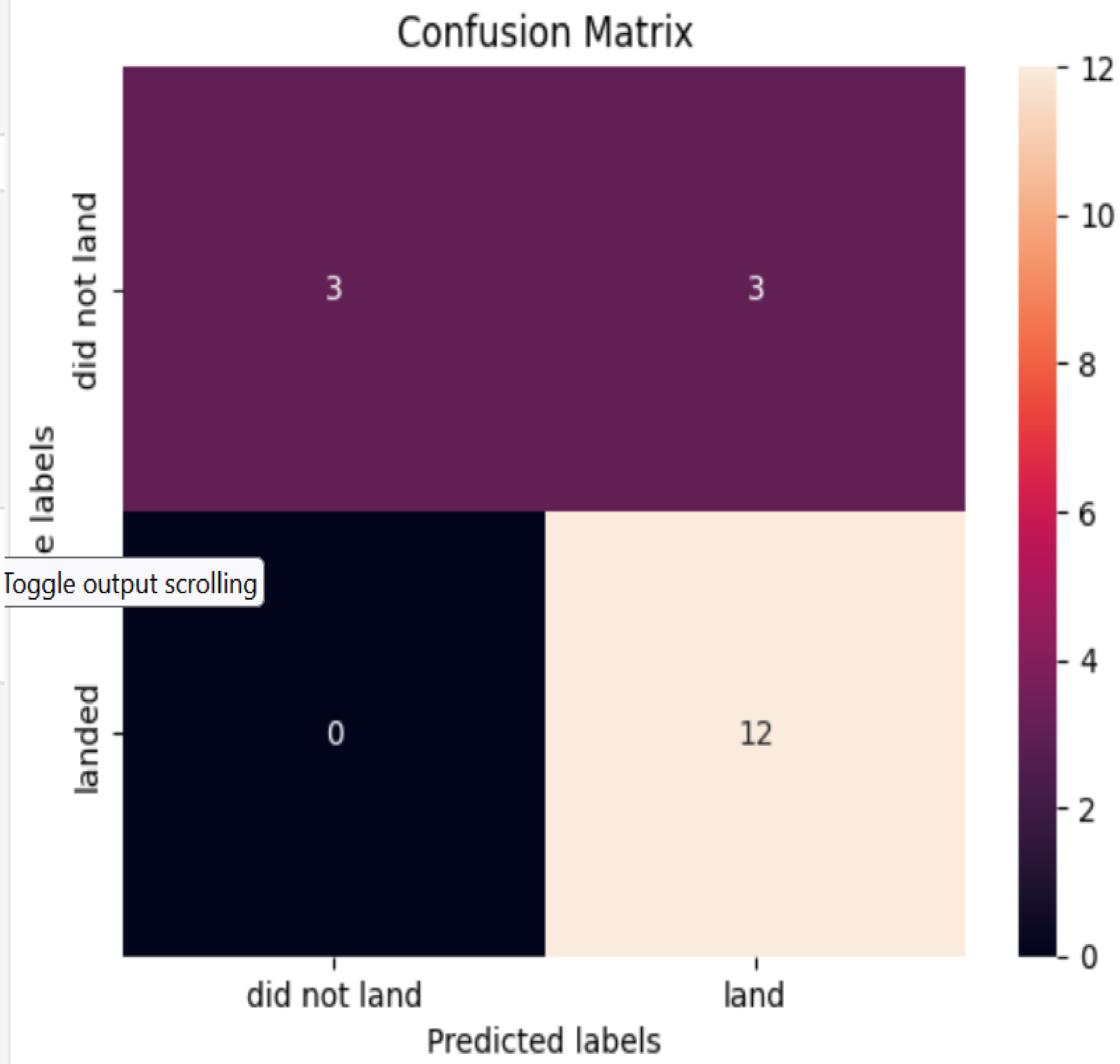
Calculate the accuracy of knn_cv on the test data using the method score:

```
print("test set accuracy :",knn_cv.score(X_test, Y_test))
```

```
test set accuracy : 0.8333333333333334
```

Plot the confusion matrix

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



```
### TASK 12
```

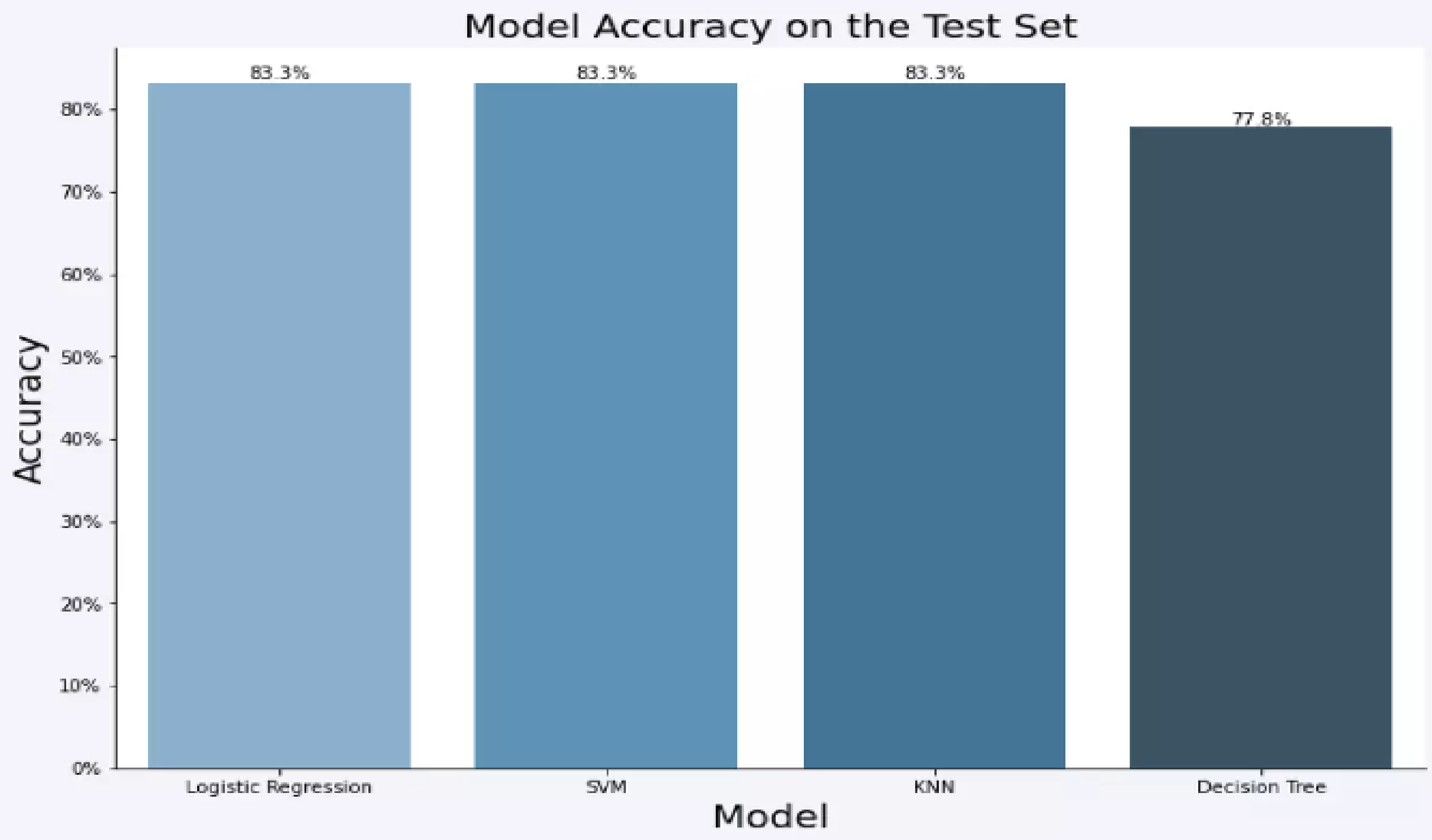
```
# Find the method performs best:


# Comparison of the acuuracy of both methods reveal that they give similar results.
# Tree methods works better for train data, however, test data not as good.
```

```
print('Accuracy for Logistics Regression method:', logreg_cv.score(X_test, Y_test))
print( 'Accuracy for Support Vector Machine method:', svm_cv.score(X_test, Y_test))
print('Accuracy for Decision tree method:', tree_cv.score(X_test, Y_test))
print('Accuracy for K nearsdt neighbors method:', knn_cv.score(X_test, Y_test))
```

```
Accuracy for Logistics Regression method: 0.8333333333333334
Accuracy for Support Vector Machine method: 0.8333333333333334
Accuracy for Decision tree method: 0.8333333333333334
Accuracy for K nearsdt neighbors method: 0.8333333333333334
```

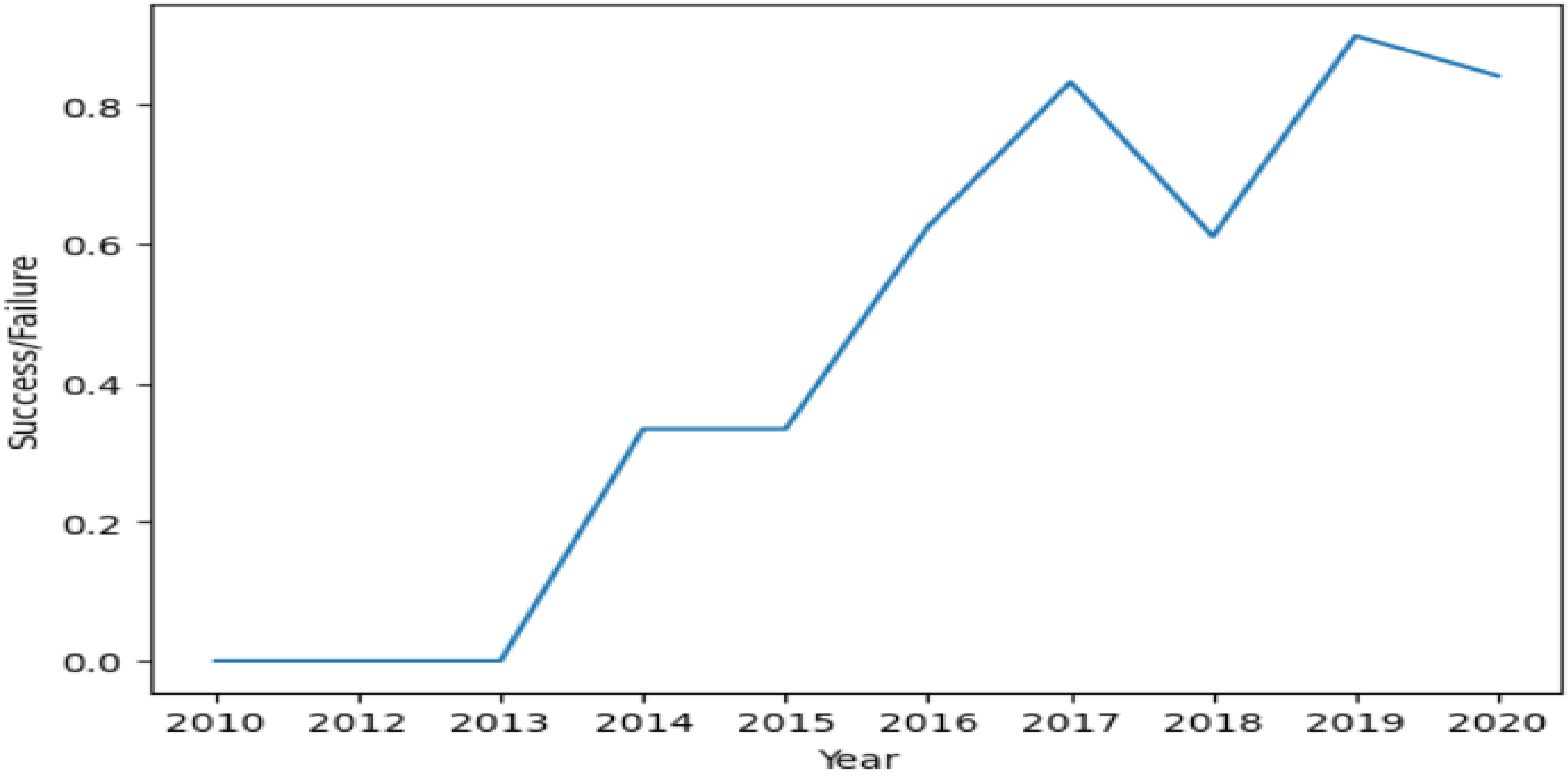
Model Accuracy



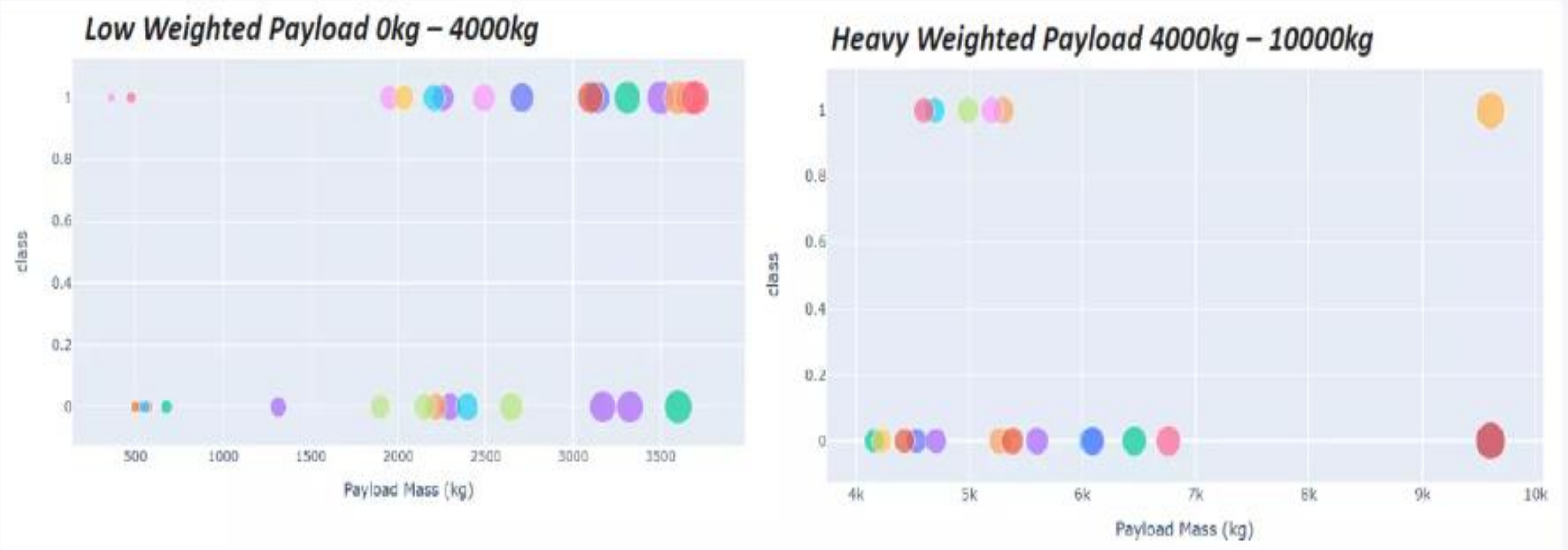
DISCUSSION

- The entire experiment seem to depend on the number of launches and learning from the previous launching lessons to enable positive outcome.
- Payload weight plays a big part in the Mission Outcome. This is causality not Correlation
- Some orbits have a higher success rate
- One Launch Site has a high success rate

Success Rate goes Up as Experience is Acquired



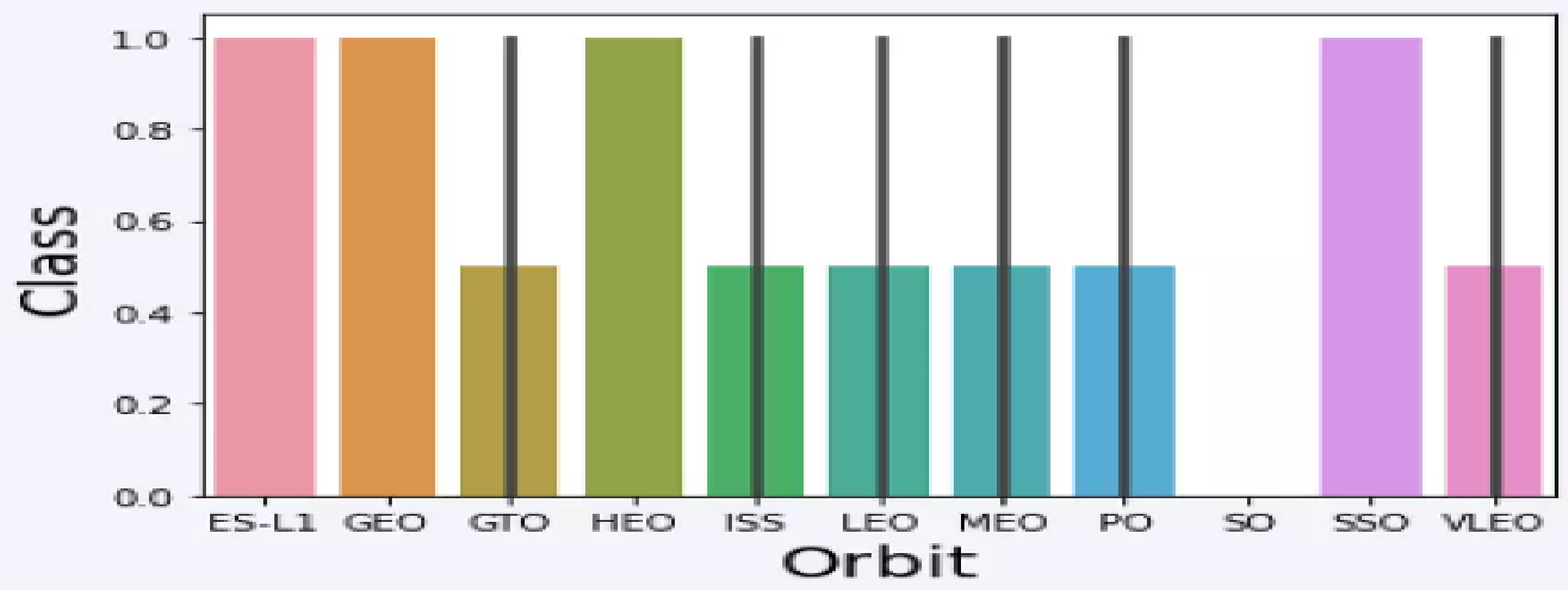
The Payload Weight to Success Rate



IBM Developer

SKILLS NETWORK

Comparison of Orbital Success Rate



OVERALL FINDINGS & IMPLICATIONS

Findings

- Launch sites are not located close to cities and densely populated residential areas
- Launch sites are located close to highways, railways and coastline
- Launch sites are near or around the equatorial region and are also in warmer areas

Implications

- This ensures public safety in case of a launch mishap
- This makes the logistics of moving people and cargo easier
- In case of a mishap and an intentional abort, the unit can be crashed into the ocean

CONCLUSION

- Point 1: Technological advancement leads to a higher launch success rate
- Point 2: Experience and lessons learned affects rate of success
- Point 3: Enough critical variables are known to guarantee a successful launch
- Point 4: At the current rate of success, it can be predicted that a 100% launching success can be guaranteed in the near future

APPENDIX

- Web Scraping: Step by Step Source Wikipedia
- https://www.youtube.com/watch?v=bargNl2WeN4&t=8s Data Wrangling. Source: YouTube
- https://www.youtube.com/watch?v=Ma8tS4p27JI&list=PLH6mU1kedUy8fCzkTTJlwsf2EnV_UvOV- Web Scraping. Source: YouTube
- Understanding Data Science. https://en.Wikipedia.org>wiki>Data Science Source: Wikipedia