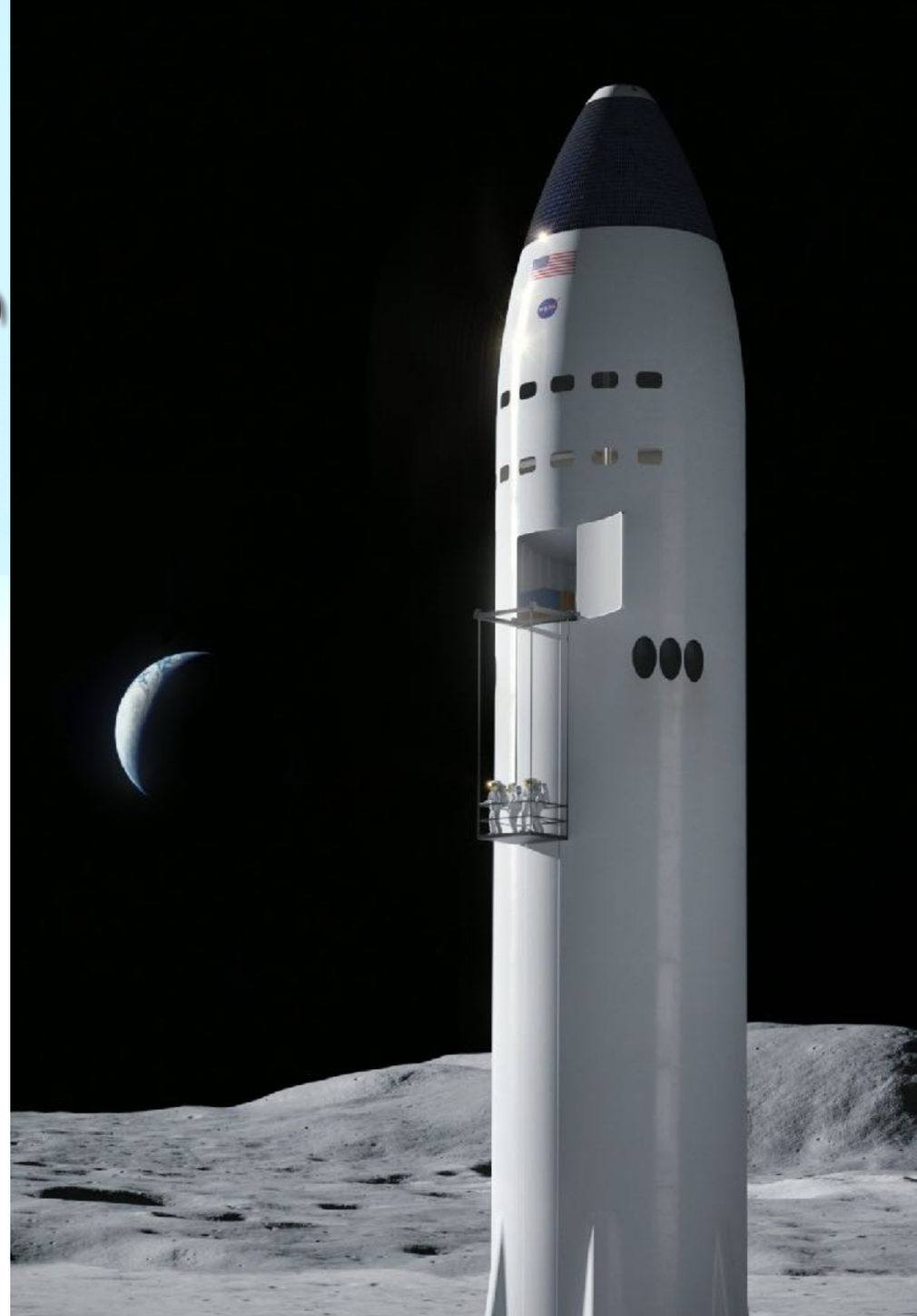
Rocket launch

First Stage Reuse Project

Ujjwal

23/04/2023



Contents

- Executive Summary
- Introduction
- Methodologies
- Data slides
- EDA with SQl results
- Folium result slides
- Predictive analysis
- EDA with visualization
- Conclusion

Executive Summary

- The purpose of this research is to findout and elaborate upon the major factors responsible for a sucessfull rocket landing. the steps undertook to establish these findings are as follows:
- ***Data collection*** which can be done through with SpaceX REST API & web scraping.
- ***Data Wrangling*** to make variables based on success/fail outcomes.
- ***Exploration*** through various visualization methods to observe the key findings.
- ***Data Analysis*** by using tools such as SQL to run various statistical operations.
- ***Model building*** to predict landing outcomes using algorithms such as logistic regression,SVM and KNN.

- **Results**
- ***Data analysis:-*** 1. Sucess rate increases overtime. 2. KCS LC-39A has the best success rate.
- ***Visualization analysis***:- Most launch lines are near the coastlines and equator.
- ***Model analysis***:- Similar results are found in all models (decision tree perform slightly better).

Introduction

background

- SpaceX is one of the biggest names in space sector today, which stand out even among state backed bahemoth such as NASA and ESA, and a huge part of credit for that goes to it's relatively inexpensive cost of space launches, which it able to achieve due to it's Falcon 9 rocket that comes with reusable first stage boosters which reduces the cost per launch to a mere $62 million from that of $165 million dollars.

- Our goal within this project is to use the publically available data regarding the the Falcon 9 to find and explore the possiblities within which our presumed startup "Space Y" or spaceX itself can be able to reuse the first stage in their future launches.

Methodology

Steps

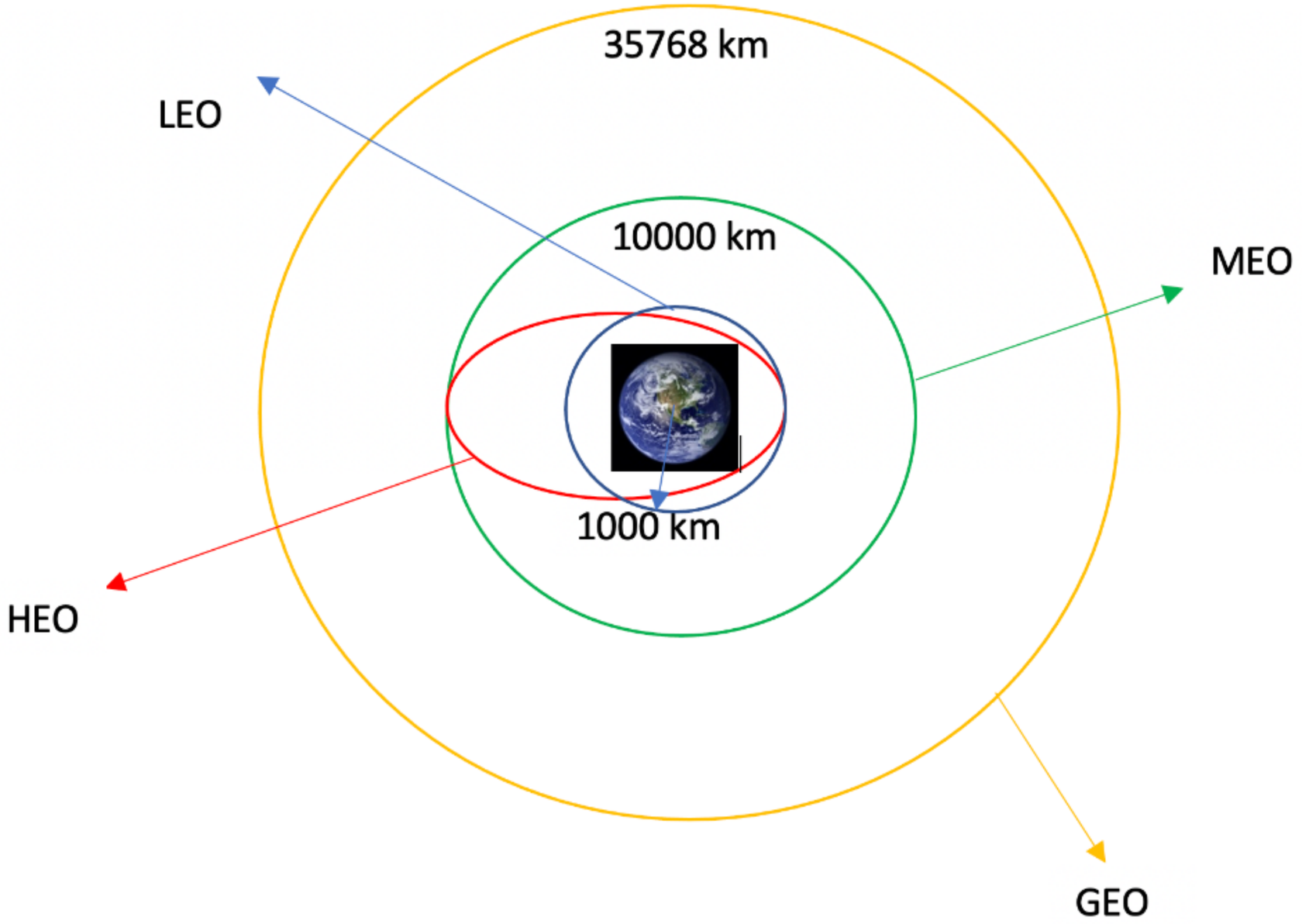
- ***Collect*** the data through webscrapping from the url.
- ***Data wrangling*** helps process and filter the data and prepare it for the next steps.
- ***Explore*** it with tools like SQL to find the insights.
- ***Visualize*** it through folium.
- ***Build*** the suitable model to predict the landing outcome.

Data collection through Web Scraping

Steps

- ***Request*** Falcon 9 launch data from wikipedia
- Make a ***BeautifulSoup*** object from HTML response
- Extract ***column names*** from HTML table header
- Collect data from ***parsing*** HTMl table
- make a ***dictionary***
- From the dictionary create a ***dataframe***
- Export it to a ***csv*** file

Data wrangling result slides


35768 km
LEO
10000 km
MEO
1000 km
HEO
GEO

TASK 2: Calculate the number and occurrence of each orbit

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

```
# 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
```

EDA with SQL slides

Task 1

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

```
%sql SELECT distinct(LAUNCH_SITE) FROM SPACEXTBL;
```

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

Task 2

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

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

[25]

```
%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
2010-04-06	18:45:00	F9 v1.0 B0003	CCAFS LC-40	Dragon Spacecraft Qualification Unit	0	LEO	SpaceX	Success	Failure (parachute)
2010-08-12	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	07:44:00	F9 v1.0 B0005	CCAFS LC-40	Dragon demo flight C2	525	LEO (ISS)	NASA (COTS)	Success	No attempt
2012-08-10	00:35:00	F9 v1.0 B0006	CCAFS LC-40	SpaceX CRS-1	500	LEO (ISS)	NASA (CRS)	Success	No attempt
2013-01-03	15:10:00	F9 v1.0 B0007	CCAFS LC-40	SpaceX CRS-2	677	LEO (ISS)	NASA (CRS)	Success	No attempt

Task 3

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

```
%sql SELECT SUM(PAYLOAD_MASS__KG_) \
    FROM SPACEXTBL \
    WHERE CUSTOMER = 'NASA (CRS)';
```

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

SUM(PAYLOAD_MASS__KG_)
45596

Task 4

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

[27]

```
%sql SELECT AVG(PAYLOAD_MASS__KG_) \
    FROM SPACEXTBL \
    WHERE BOOSTER_VERSION = 'F9 v1.1';
```

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

AVG(PAYLOAD_MASS__KG_)
2928.4

Task 5

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

Hint:Use min function

[35]

```
%sql SELECT MIN(DATE) \
FROM SPACEXTBL \
WHERE Landing_Outcome = 'Success (ground pad)'
```

```
 * 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

[37]

```
%sql SELECT PAYLOAD \
FROM SPACEXTBL \
WHERE Landing_Outcome = 'Success (drone ship)' \
AND PAYLOAD_MASS__KG_ BETWEEN 4000 AND 6000;
```

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

Payload
JCSAT-14
JCSAT-16
SES-10
SES-11 / EchoStar 105

Task 7

List the total number of successful and failure mission outcomes

[38]
```
%sql SELECT Mission_outcome, COUNT(*) as total_number \
FROM SPACEXTBL \
GROUP BY Mission_outcome;
```

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

Mission_Outcome	total_number
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

[39]
```
%sql SELECT BOOSTER_VERSION \
FROM SPACEXTBL \
WHERE PAYLOAD_MASS__KG_ = (SELECT MAX(PAYLOAD_MASS__KG_) FROM SPACEXTBL);
```

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

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

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, 4, 2) as month to get the months and substr(Date,7,4)='2015' for year.

[75]

```
%sql SELECT substr(Date,4,2) as month, Date, Booster_Version, LAUNCH_SITE, Landing_Outcome \
FROM SPACEXTBL \
where Landing_Outcome = 'Failure (drone_ship)';
```

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

month	Date	Booster_Version	Launch_Site	Landing_Outcome

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.

[64]

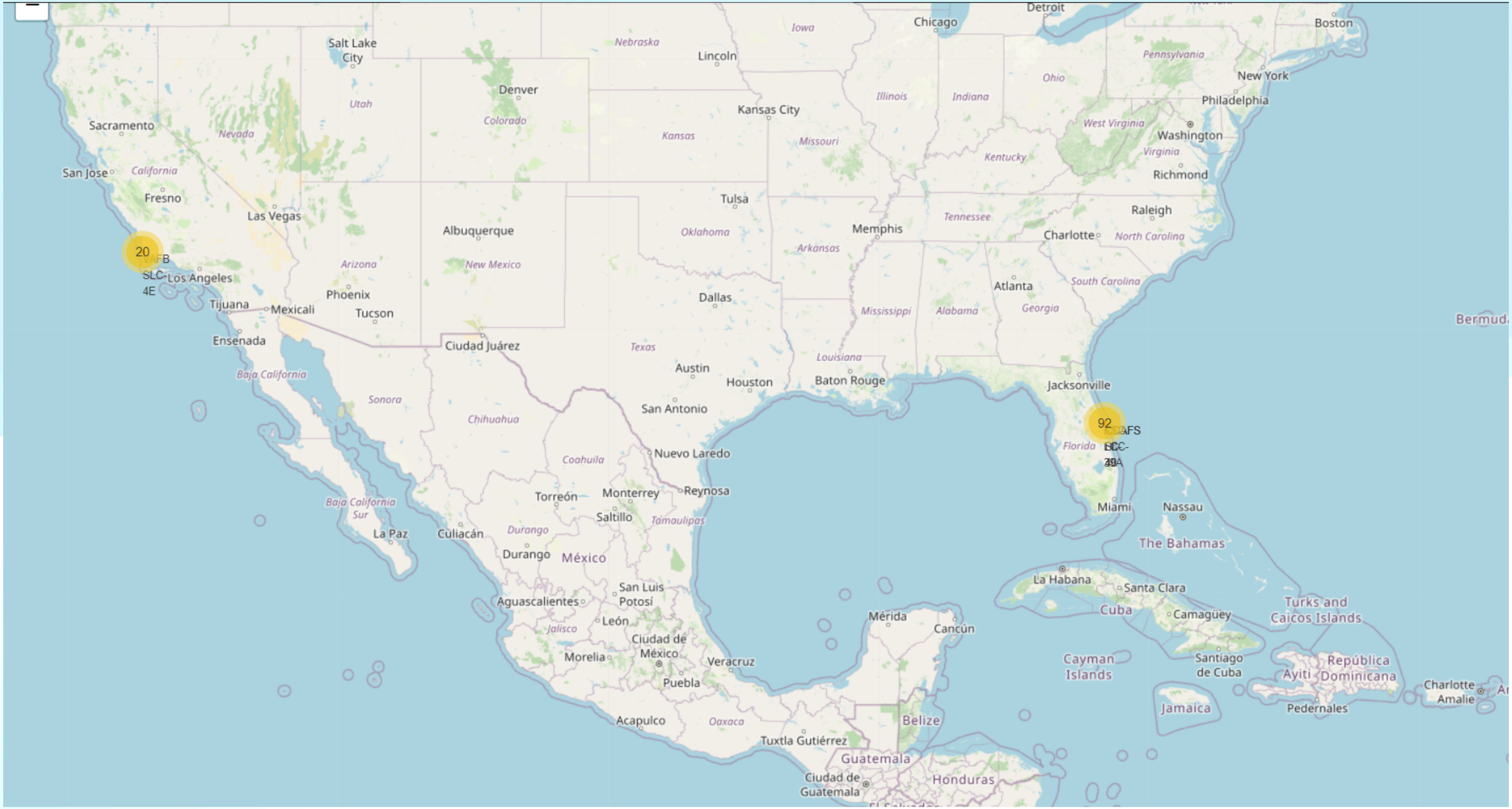
```
%sql SELECT Landing_Outcome, count(*) as count_outcomes \
FROM SPACEXTBL \
WHERE Date between '04-06-2010' and '20-03-2017' group by [Landing_Outcome] order by count_outcomes DESC;
```

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

Landing_Outcome	count_outcomes

Folium result slides


Washington
North Dakota
Montana
Minnesota
Québec
Idaho
Ottawa
Montréal
Portland
Saint Paul
Wisconsin
Michigan
Maine
Lake Huron
Oregon
South Dakota
Boise
Lake Michigan
Toronto
Milwaukee
Hamilton
Wyoming
New York
Detroit
Iowa
Chicago
Boston
Salt Lake City
Nebraska
Lincoln
Pennsylvania
Ohio
New York
Denver
Illinois
Indiana
Philadelphia
Utah
Kansas City
Sacramento
Colorado
West Virginia
Washington
Nevada
Kansas
Missouri
Virginia
Kentucky
San Jose
California
Richmond
Fresno
Tulsa
Las Vegas
Tennessee
Raleigh
Albuquerque
Oklahoma
Memphis
Charlotte
North Carolina
Arkansas
VAFB
SLC-4E
Los Angeles
Arizona
New Mexico
Atlanta
South Carolina
Phoenix
Dallas
Tijuana
Mexicali
Mississippi
Alabama
Georgia
Tucson
Ensenada
Ciudad Juárez
Texas
Louisiana
Austin
Baja California
Houston
Baton Rouge
Jacksonville
Sonora
San Antonio
Chihuahua
CCSFS
SLC-
Florida
Coahuila
Nuevo Laredo
Torreón
Monterrey
Reynosa
Miami
Nassau
Baja California Sur
Saltillo
Tamaulipas
The Bahamas
La Paz
Culiacán
Durango
México
La Habana
Santa Clara
Cuba
San Luis Potosí
Aguascalientes
Camagüey
Turks and Caicos Islands
Mérida
León
Cancún
Jalisco
Ciudad de


20
92
Detroit
Chicago
Boston
Iowa
Nebraska
Lincoln
Salt Lake City
Pennsylvania
New York
Ohio
Denver
Illinois
Indiana
Philadelphia
Utah
Kansas City
Colorado
Sacramento
Nevada
Kansas
Missouri
West Virginia
Washington
Virginia
Kentucky
San Jose
California
Richmond
Fresno
Tulsa
Raleigh
Las Vegas
Tennessee
Albuquerque
Memphis
Oklahoma
Charlotte
North Carolina
Arkansas
Arizona
New Mexico
Atlanta
South Carolina
Los Angeles
Phoenix
Dallas
Tijuana
Mexicali
Tucson
Mississippi
Alabama
Georgia
Ensenada
Ciudad Juárez
Texas
Louisiana
Austin
Houston
Baton Rouge
Jacksonville
Baja California
Sonora
San Antonio
Chihuahua
Florida
Nuevo Laredo
Coahuila
Torreón
Monterrey
Reynosa
Miami
Nassau
Baja California Sur
Saltillo
Tamaulipas
The Bahamas
La Paz
Culiacán
Durango
Durango
México
La Habana
Santa Clara
San Luis Potosí
Cuba
Aguascalientes
Mérida
Camagüey
Turks and Caicos Islands
León
Cancún
Jalisco
Ciudad de México
Cayman Islands
Santiago de Cuba
Morelia
Veracruz
República Dominicana
Haití
Puebla
Charlotte Amalie
Jamaica
Pedernales
Acapulco
Oaxaca
Belize
Tuxtla Gutiérrez
Guatemala
Ciudad de Guatemala
Honduras

[38]

```
print("City Distance", city_distance)

print("Railway Distance", railway_distance)

print("Highway Distance", highway_distance)

print("Coastline Distance", distance_coastli
```

```
City Distance 23.234752126023245
Railway Distance 21.961465676043673
Highway Distance 26.880385696814 92
Coastline Distance 0.8627671182499878
```

Predictive analysis

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']).

[8]
```
Y = data['Class'].to_numpy()
```

TASK 2

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

[9]
```
# students get this
transform = preprocessing.StandardScaler()
X = transform.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`

[10]
```
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.

[11]
```
Y_test.shape
```

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`.

[12]
```
parameters ={'C':[0.01,0.1,1],
             'penalty':['l2'],
             'solver':['lbfgs']}
```

[13]
```
parameters ={"C":[0.01,0.1,1],'penalty':['l2'], 'solver':['lbfgs']}# l1 lasso l2 ridge
lr=LogisticRegression()
logreg_cv = GridSearchCV(estimator=lr, cv=10, param_grid=parameters).fit(X_train, Y_train)
```

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_`.

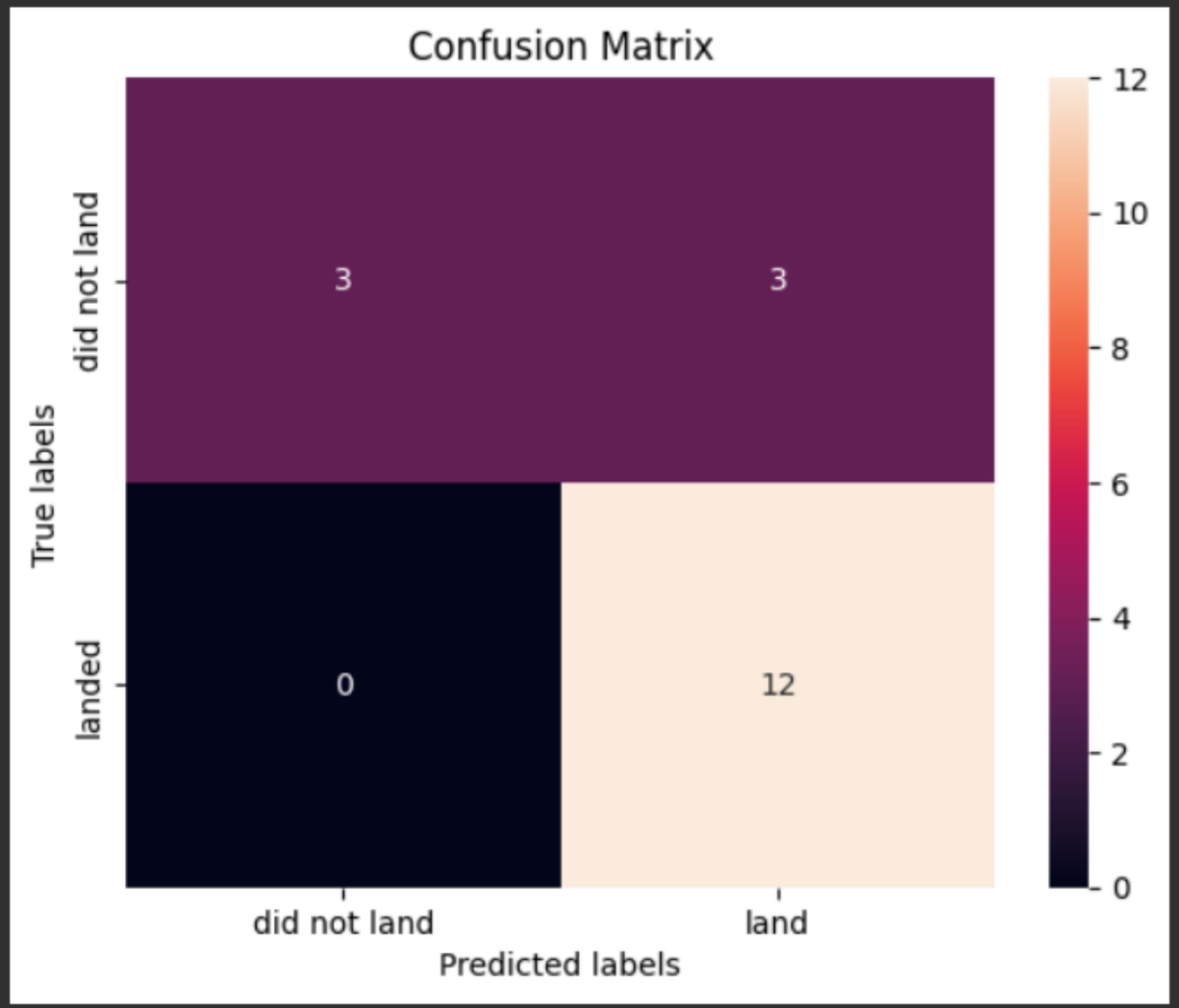
[14]
```
print("tuned hpyerparameters :(best parameters) ",logreg_cv.best_params_)
print("accuracy :",logreg_cv.best_score_)
```

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

Lets look at the confusion matrix:

[16]

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



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`.

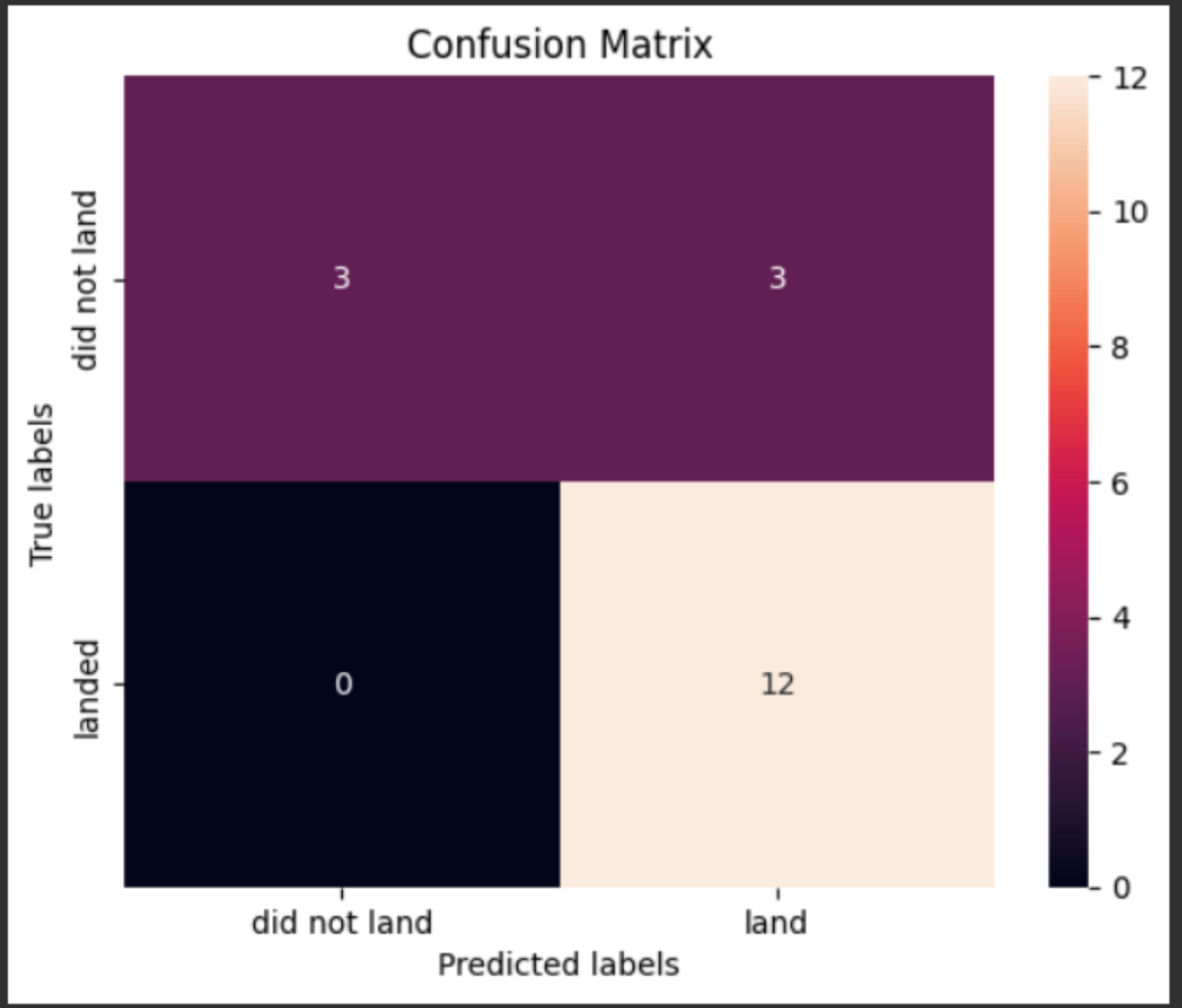
[17]
```
parameters = {'kernel':('linear', 'rbf','poly','rbf', 'sigmoid'),
              'C': np.logspace(-3, 3, 5),
              'gamma':np.logspace(-3, 3, 5)}
svm = SVC()
svm_cv = GridSearchCV(estimator=svm, cv=10, param_grid=parameters).fit(X_train, Y_train)
```

[18]
```
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 : 0.8482142857142856
```

[23]

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



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`.

22]

```
parameters = {'criterion': ['gini', 'entropy'],
     'splitter': ['best', 'random'],
     'max_depth': [2*n for n in range(1,10)],
     'max_features': ['auto', 'sqrt'],
     'min_samples_leaf': [1, 2, 4],
     'min_samples_split': [2, 5, 10]}

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

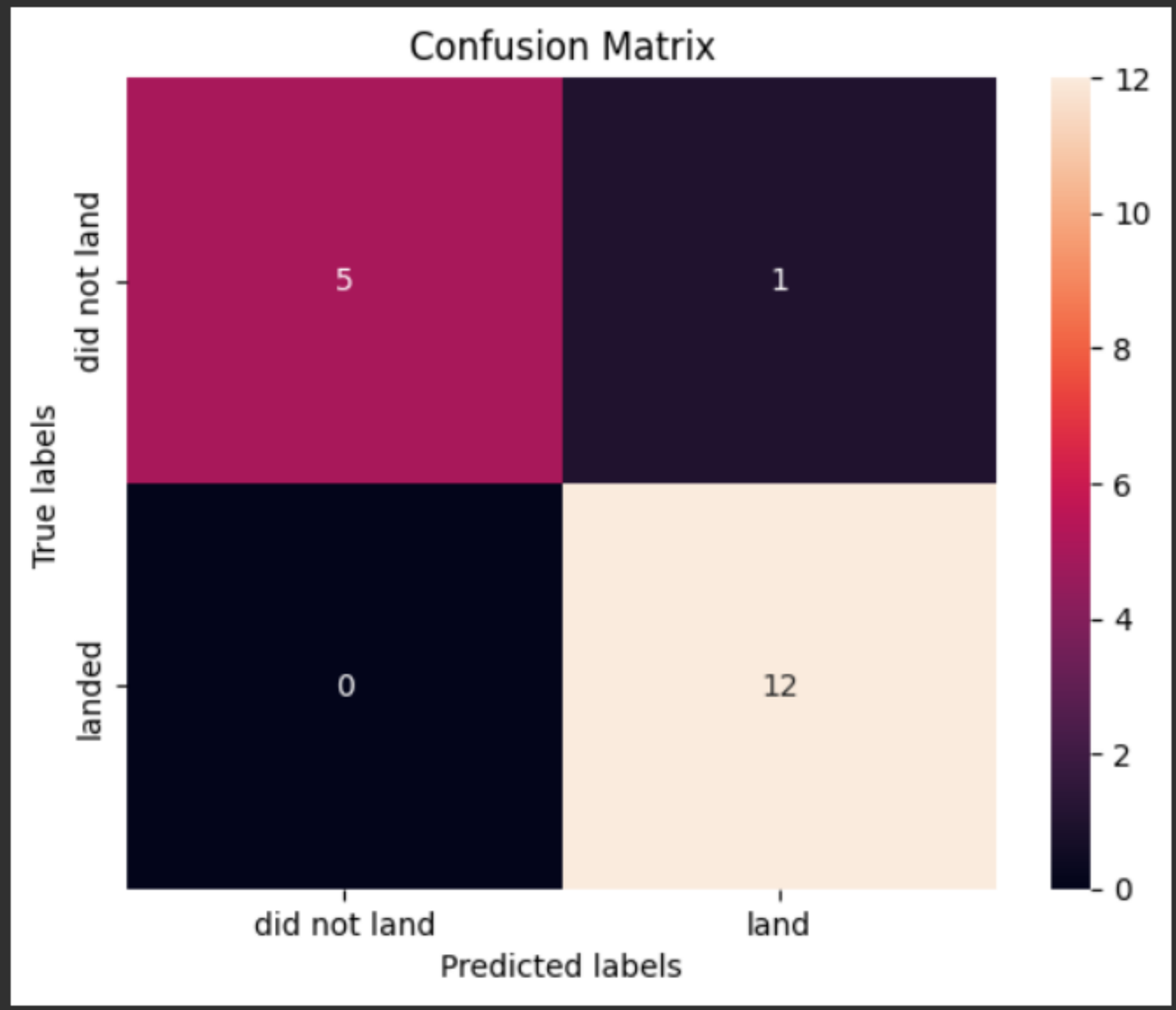
24]

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

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

[26]

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



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`.

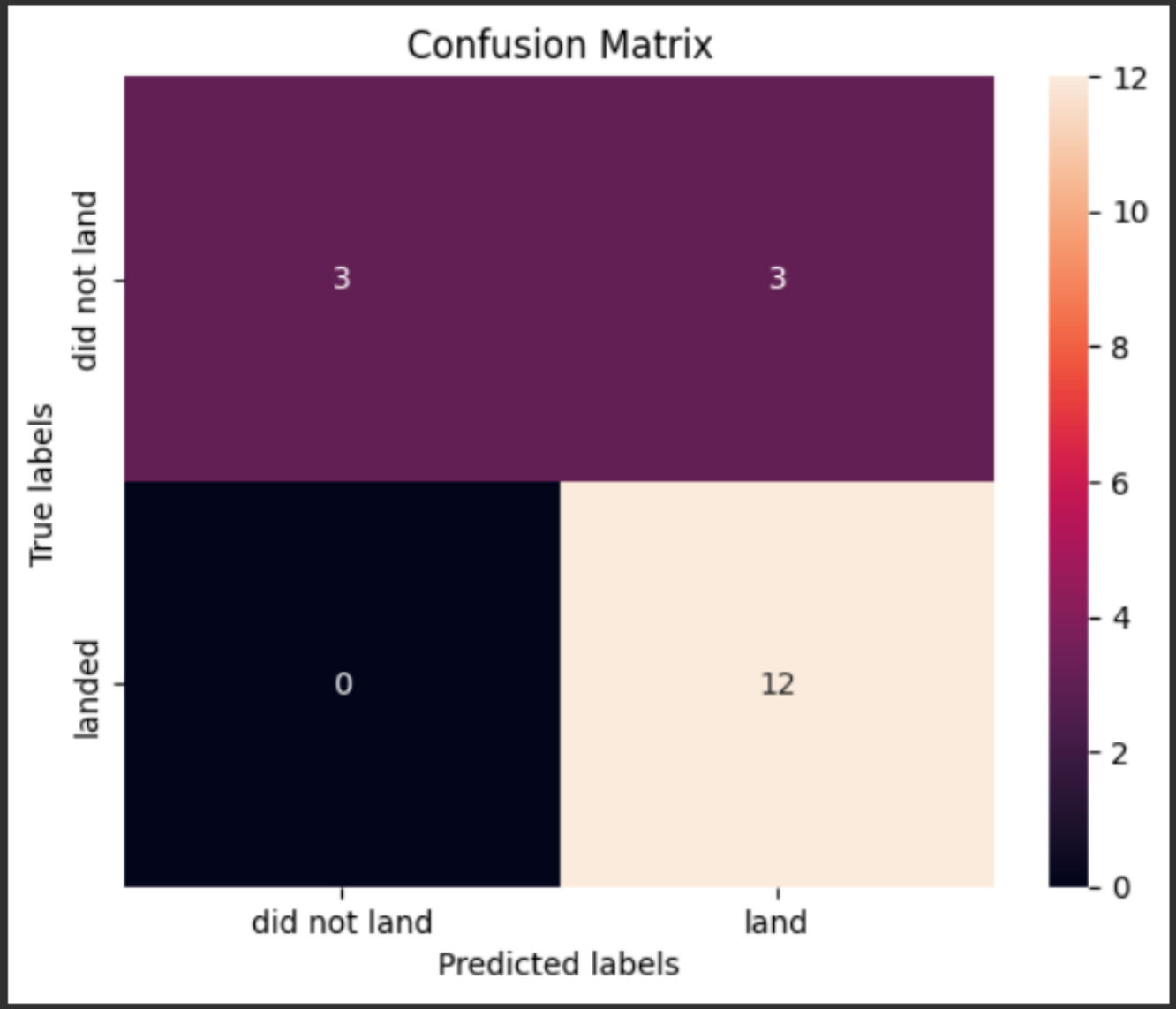
```
parameters = {'n_neighbors': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
              'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute'],
              'p': [1,2]}

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

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

```
tuned hpyerparameters :(best parameters)  {'algorithm': 'auto', 'n_neighbors': 10, 'p': 1}
accuracy : 0.8482142857142858
```

[30]

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



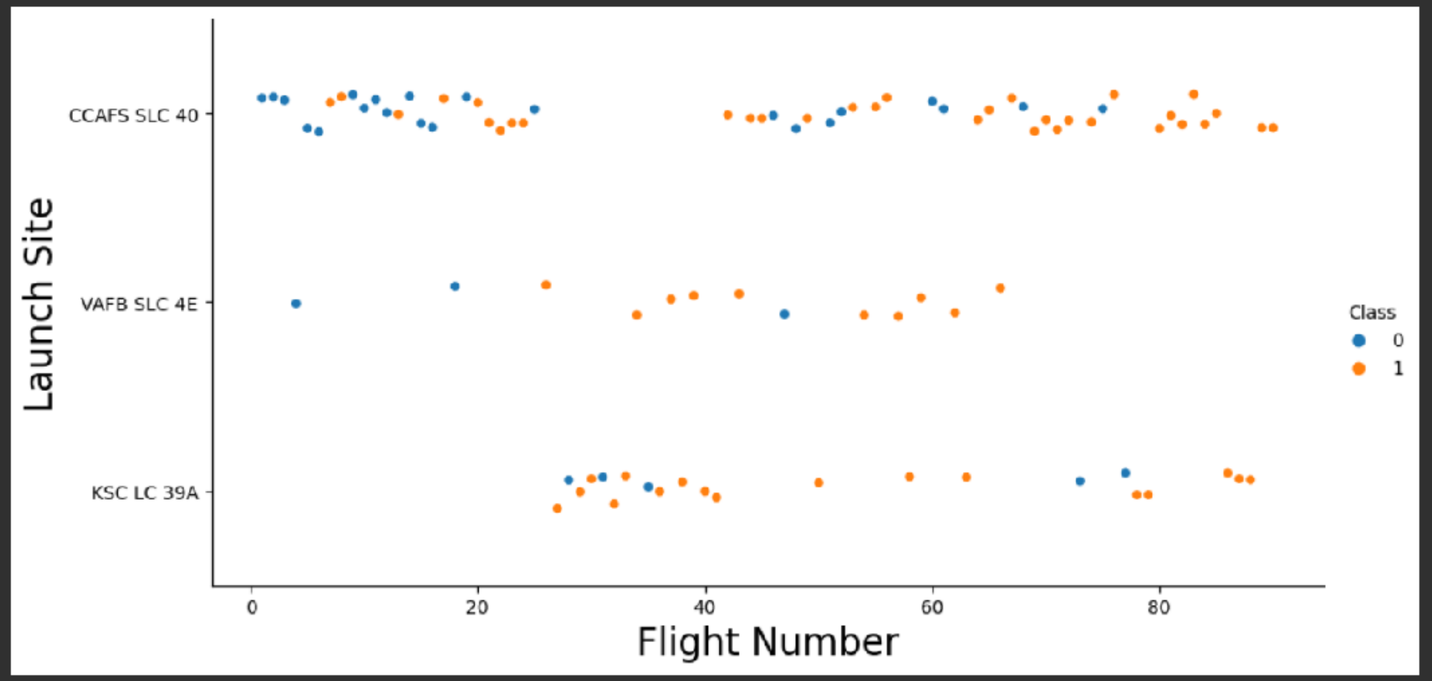
TASK 12

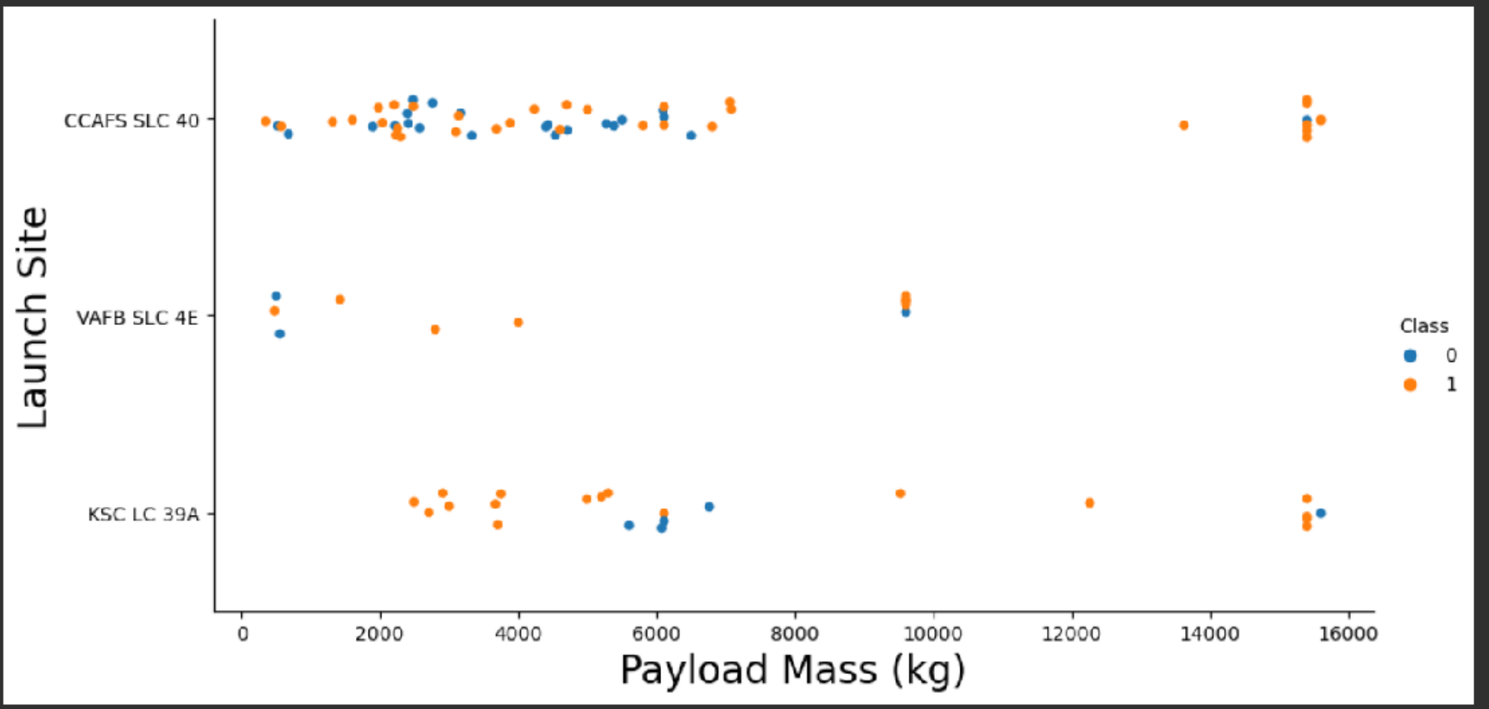
Find the method performs best:

```
accuracy = [svm_cv_score, logreg_score, knn_cv_score, tree_cv_score]
accuracy = [i * 100 for i in accuracy]

method = ['Support Vector Machine', 'Logistic Regression', 'K Nearest Neighbour', 'Decision Tree']
models = {'ML Method':method, 'Accuracy Score (%)':accuracy}

ML_df = pd.DataFrame(models)
ML_df
```

	ML Method	Accuracy Score (%)
0	Support Vector Machine	83.333333
1	Logistic Regression	83.333333
2	K Nearest Neighbour	83.333333
3	Decision Tree	83.333333

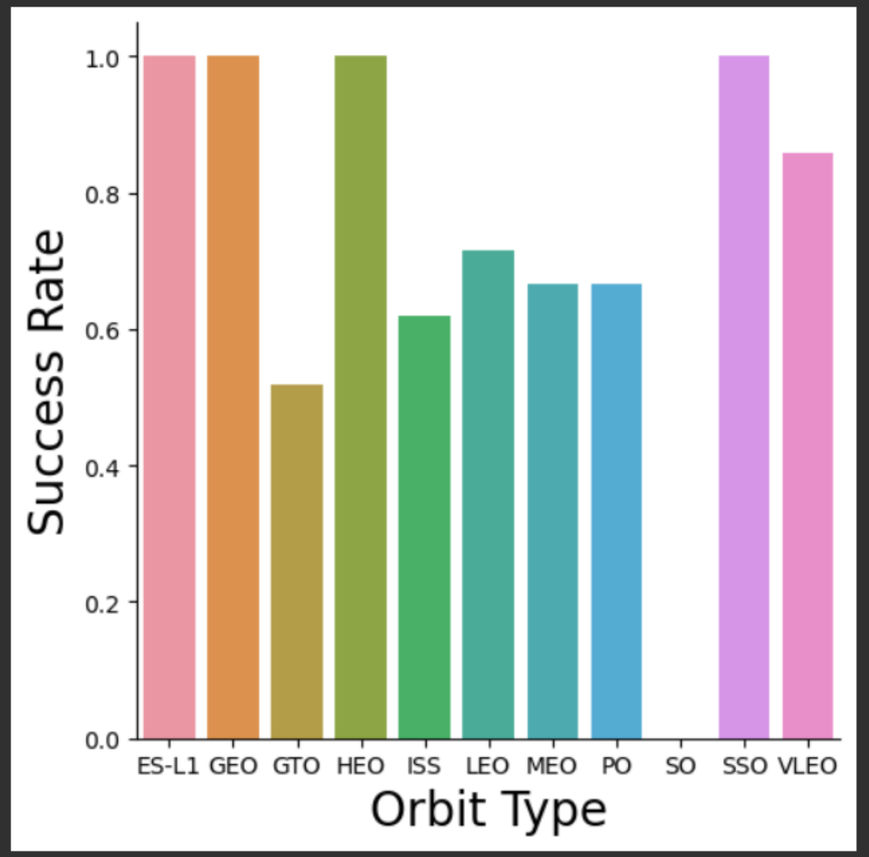
EDA with visualization result slides

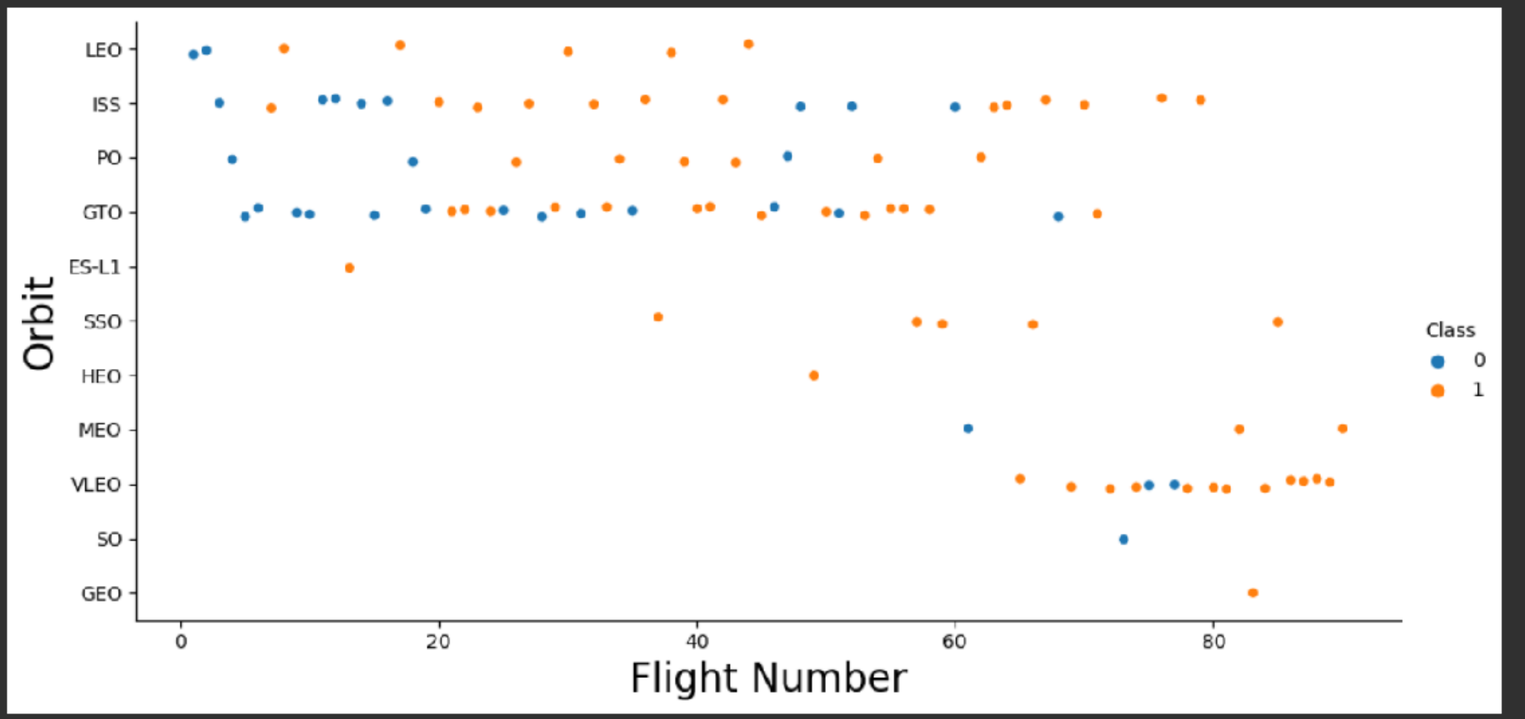

CCAFS SLC 40
VAFB SLC 4E
KSC LC 39A
Launch Site
0
20
40
60
80
Flight Number
Class
0
1

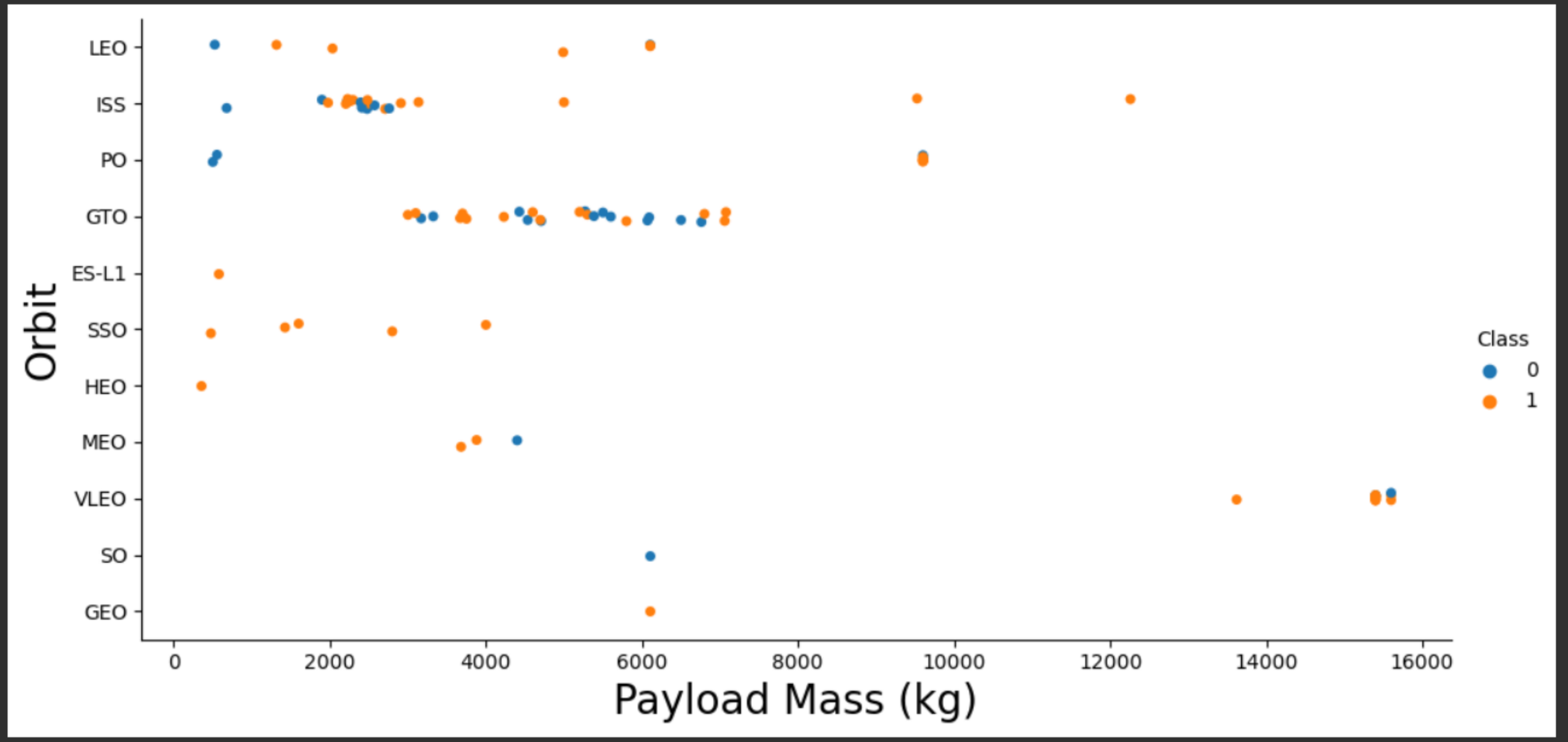

CCAFS SLC 40
VAFB SLC 4E
KSC LC 39A
Launch Site
0
2000
4000
6000
8000
10000
12000
14000
16000
Payload Mass (kg)
Class
0
1

[10]

```
### TASK  3: Visualize the relationship between success rate of each orbit type
sns.catplot(x= 'Orbit', y = 'Class', data = df.groupby('Orbit')['Class'].mean().reset_index(), kind = 'bar')
plt.xlabel('Orbit Type',fontsize=20)
plt.ylabel('Success Rate',fontsize=20)
plt.show()
```







	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins	Reused	Legs	LandingPad	Block	ReusedCount	Serial	Longitude	Latitude	Class
0	1	2010	Falcon 9	6104.959412	LEO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B0003	-80.577366	28.561857	0
1	2	2012	Falcon 9	525.000000	LEO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B0005	-80.577366	28.561857	0
2	3	2013	Falcon 9	677.000000	ISS	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B0007	-80.577366	28.561857	0
3	4	2013	Falcon 9	500.000000	PO	VAFB SLC 4E	False Ocean	1	False	False	False	NaN	1.0	0	B1003	-120.610829	34.632093	0
4	5	2013	Falcon 9	3170.000000	GTO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B1004	-80.577366	28.561857	0

```
### 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.

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

	FlightNumber	PayloadMass	Flights	GridFins	Reused	Legs	Block	ReusedCount	Orbit_ES-L1	Orbit_GEO	...	Serial_B1048	Serial_B1049	Serial_B1050	Serial_B1051	Serial_B1054	Serial_B1056	Serial_B1058	Serial_B1059	Serial_B1060	Serial_B1062
0	1	6104.959412	1	False	False	False	1.0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
1	2	525.000000	1	False	False	False	1.0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
2	3	677.000000	1	False	False	False	1.0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
3	4	500.000000	1	False	False	False	1.0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
4	5	3170.000000	1	False	False	False	1.0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

5 rows × 80 columns

```
### TASK  8: Cast all numeric columns to `float64`
features_one_hot.astype(float)
```

	FlightNumber	PayloadMass	Flights	GridFins	Reused	Legs	Block	ReusedCount	Orbit_ES-L1	Orbit_GEO	...	Serial_B1048	Serial_B1049	Serial_B1050	Serial_B1051	Serial_B1054	Serial_B1056	Serial_B1058	Serial_B1059	Serial_B1060	Serial_B1062
0	1.0	6104.959412	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	2.0	525.000000	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	3.0	677.000000	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	4.0	500.000000	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	5.0	3170.000000	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
85	86.0	15400.000000	2.0	1.0	1.0	1.0	5.0	2.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0
86	87.0	15400.000000	3.0	1.0	1.0	1.0	5.0	2.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0
87	88.0	15400.000000	6.0	1.0	1.0	1.0	5.0	5.0	0.0	0.0	...	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0
88	89.0	15400.000000	3.0	1.0	1.0	1.0	5.0	2.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0
89	90.0	3681.000000	1.0	1.0	0.0	1.0	5.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0

90 rows × 80 columns

Conclusion

- The size and quality plays a pivotal in the effienciency of the results.
- Coastal area (near the equator) are preferable spot for launch sites.
- KSC LC-39A has the best launch results overall.
- In terms of orbits GEO,HEO and SSO have a 100% success rate.