Predicting Falcon 9 First-stage Landin

Aflah Khalaf Al Abri

Oct 9th 2023

OUTLINE

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

EXECUTIVE SUMMARY

- Objective: predict the success of Falcon 9 first stage landings
- Cost Efficiency: SpaceX offers launches at $62 million, a fraction of traditional providers' costs.
- Importance: Understanding first stage landings impacts launch cost and competitiveness in the space industry.
- Data Collection: SpaceX API and Previous launches records on wikipedia
- Methodology: data is collected, cleaned, visualized, and fed into multiple machine algorithms to predict successful landings to be displayed in a confusion matrix
- Key findings:
  - most of the machine learning algorithms used had a similar prediction accuracy
  - Feature such as launch site, payload mass, and year had significant effect on the results

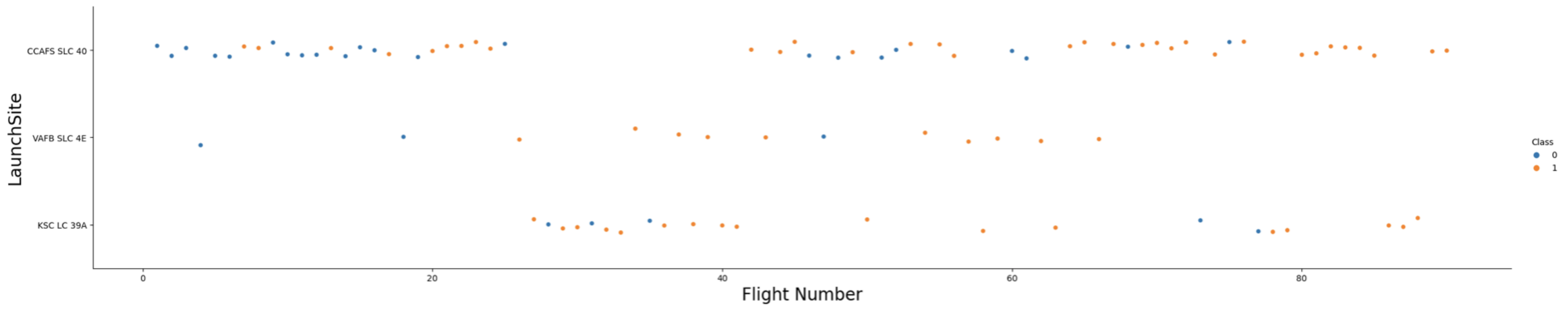
INTRODUCTION

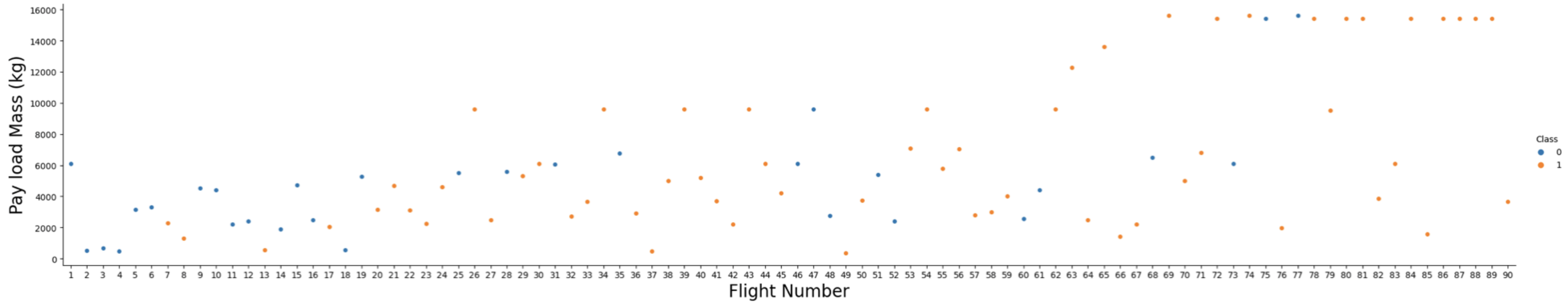
- Welcome to our capstone project, where we explore the captivating realm of space exploration and commercial rocket launches.
- Objective: Our mission is to harness the power of data to predict the success of Falcon 9 first stage landings—a critical element in SpaceX's groundbreaking cost-saving strategy.
- Importance: Why does this matter? It's not just about rockets; it's about economics. Understanding the factors influencing first stage landings can have a profound impact on the cost of space travel.
- SpaceX's Cost-Saving Strategy: SpaceX has redefined the space industry by offering rocket launches at an astonishing $62 million, a fraction of the cost of traditional providers that charge upwards of $165 million per launch. The key to this incredible cost efficiency lies in SpaceX's innovative ability to reuse the Falcon 9's first stage.

METHODOLOGY

- Data was obtained from two sources:
  - SpaceX API
  - Previous launches tables in Wikipedia
- Data wrangling was done to clean up and preprocess the data
- EDA was done through visualization and analysis
- Interactive dashboard was created to further interact with findings
- Finally, multiple machine learning algorithms were applied to predict successful landing of stage one

EDA

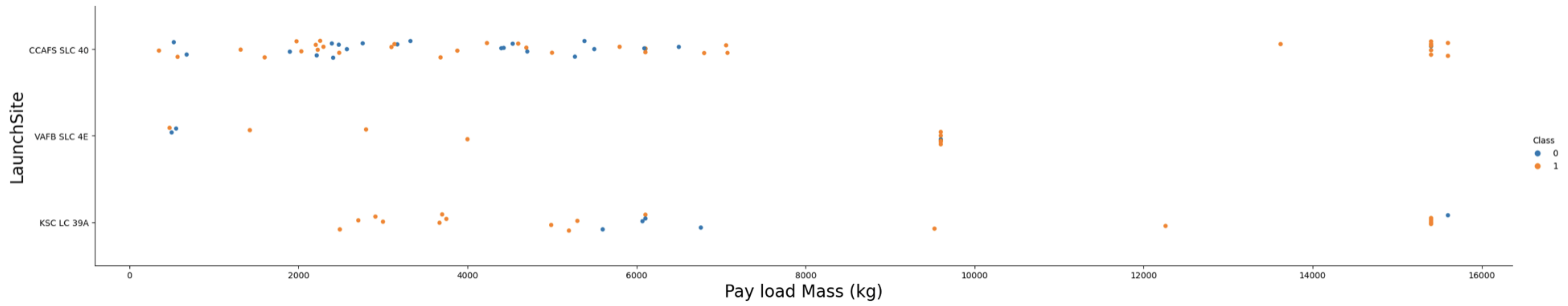


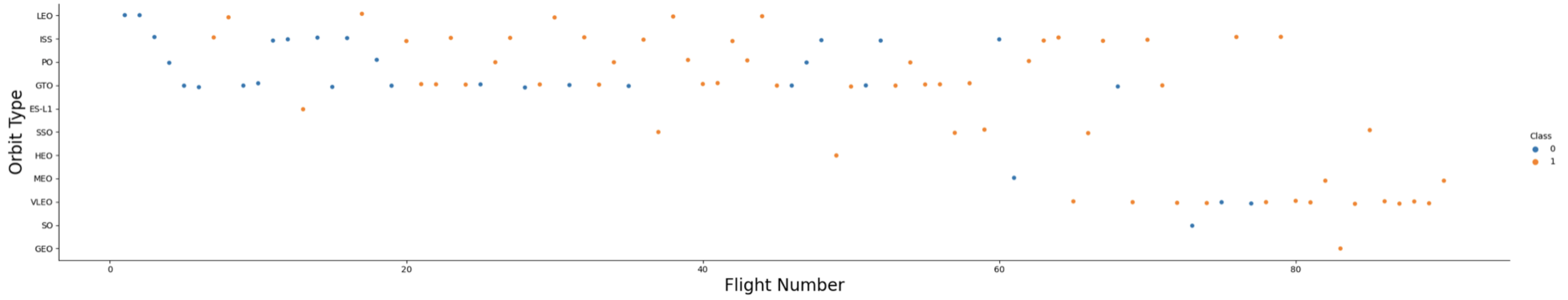


IBM Developer

SKILLS NETWORK

EDA

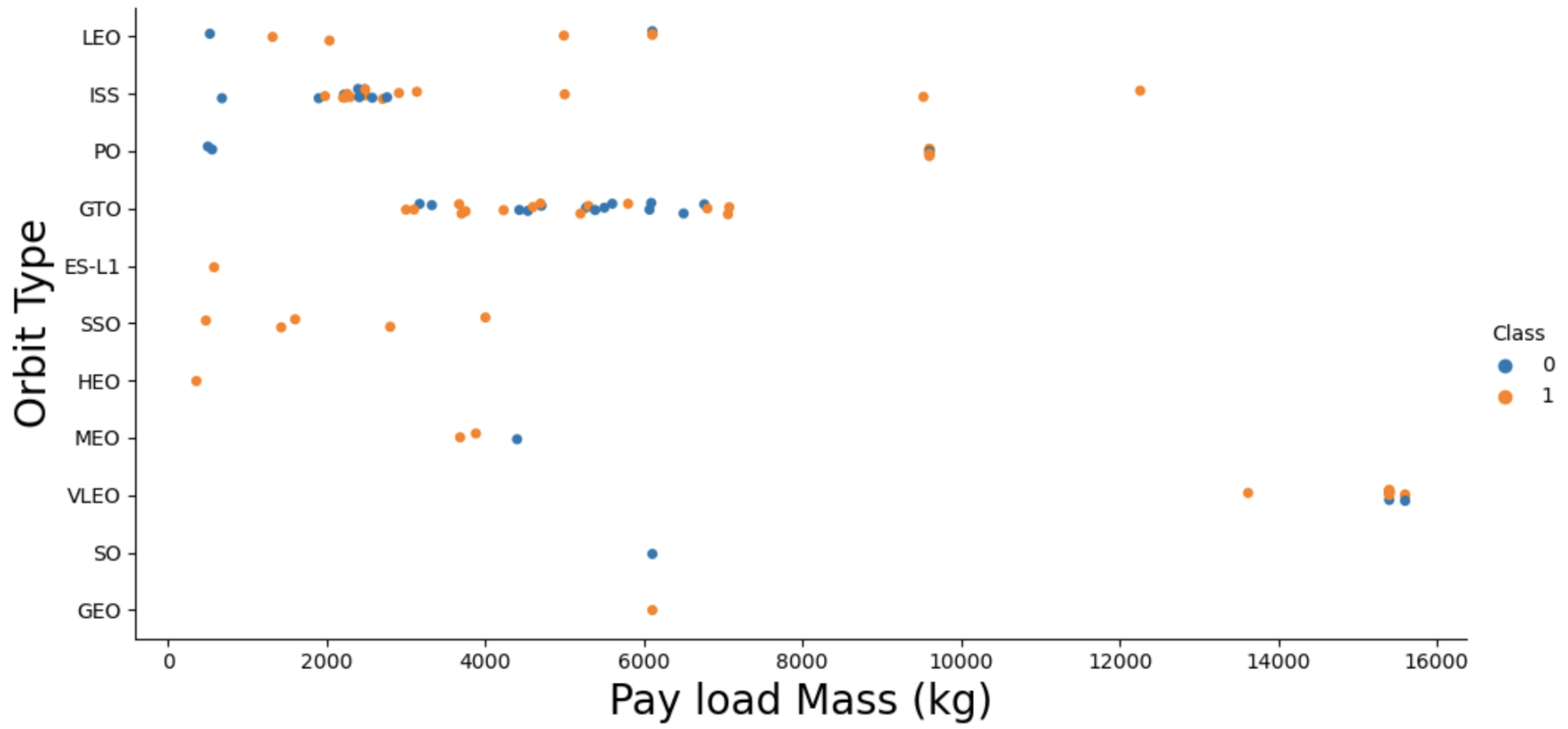




IBM Developer

SKILLS NETWORK

EDA



IBM Developer

SKILLS NETWORK

EDA

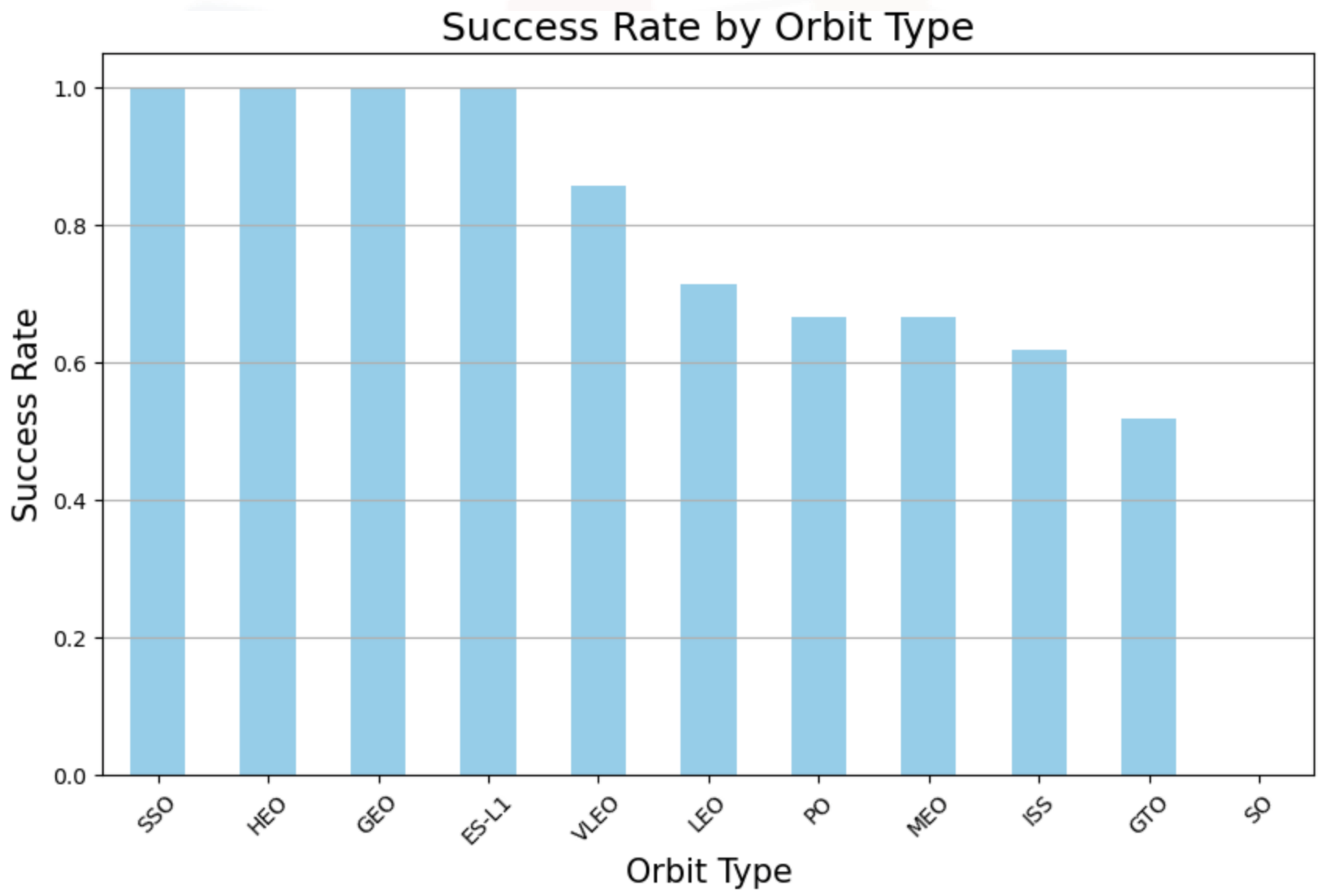
Features Engineering

By now, you should obtain some preliminary insights about how each important variable would affect the success rate, we will select the features that will be used in success prediction in the future module.

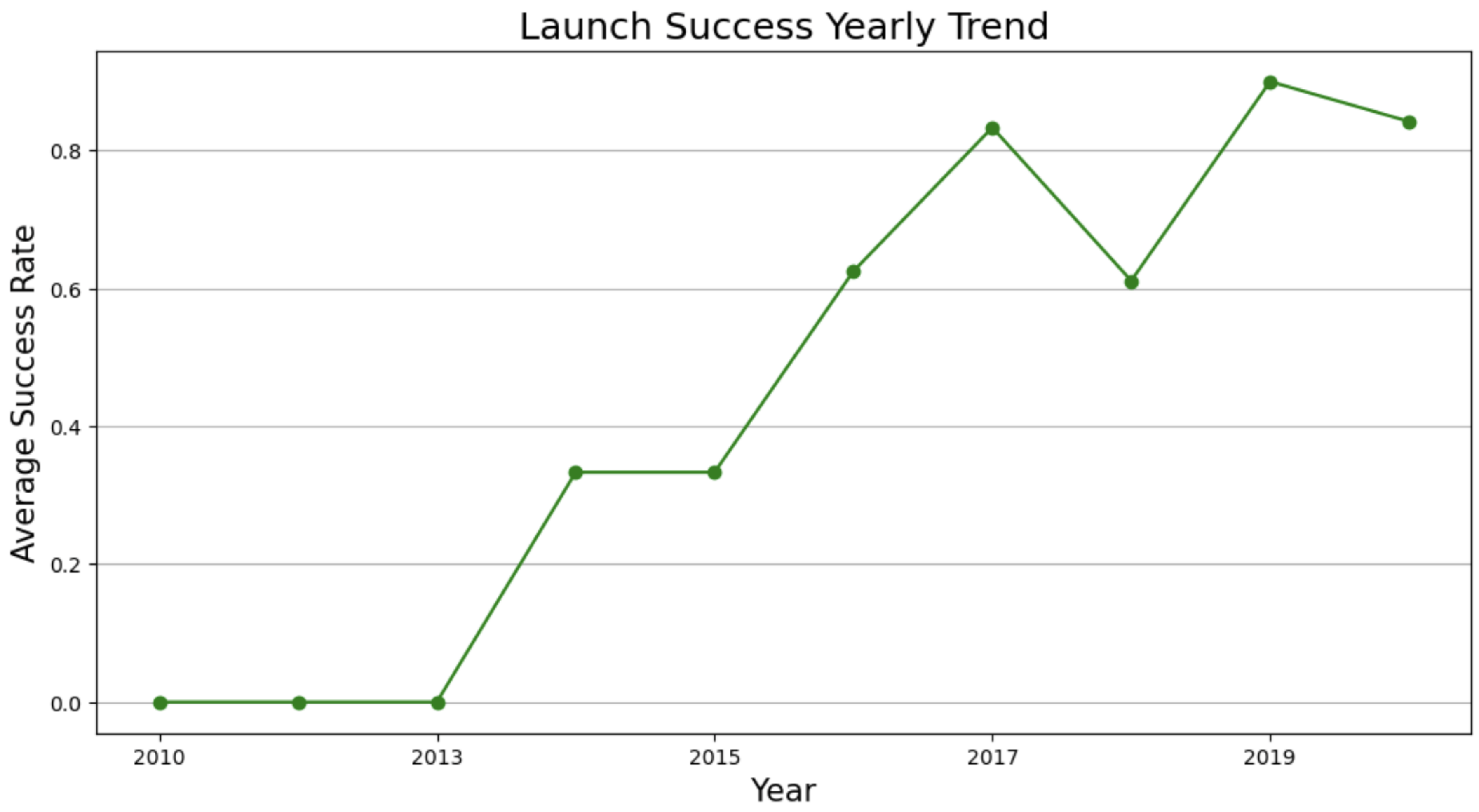
```
features = df[['FlightNumber', 'PayloadMass', 'Orbit', 'LaunchSite', 'Flights', 'GridFins', 'Reused', 'Legs', 'Landi
features.head()
```

	FlightNumber	PayloadMass	Orbit	LaunchSite	Flights	GridFins	Reused	Legs	LandingPad	Block	ReusedCount	Serial
0	1	6104.959412	LEO	CCAFS SLC 40	1	False	False	False	NaN	1.0	0	B0003
1	2	525.000000	LEO	CCAFS SLC 40	1	False	False	False	NaN	1.0	0	B0005
2	3	677.000000	ISS	CCAFS SLC 40	1	False	False	False	NaN	1.0	0	B0007
3	4	500.000000	PO	VAFB SLC 4E	1	False	False	False	NaN	1.0	0	B1003
4	5	3170.000000	GTO	CCAFS SLC 40	1	False	False	False	NaN	1.0	0	B1004

EDA


Success Rate by Orbit Type
Success Rate
1.0
0.8
0.6
0.4
0.2
0.0
SSO
HEO
GEO
ES-L1
VLEO
LEO
PO
MEO
ISS
GTO
SO
Orbit Type

EDA



Launch Success Yearly Trend
Average Success Rate
Year
0.0
0.2
0.4
0.6
0.8
2010
2013
2015
2017
2019

EDA

TASK 7: Create dummy variables to categorical columns

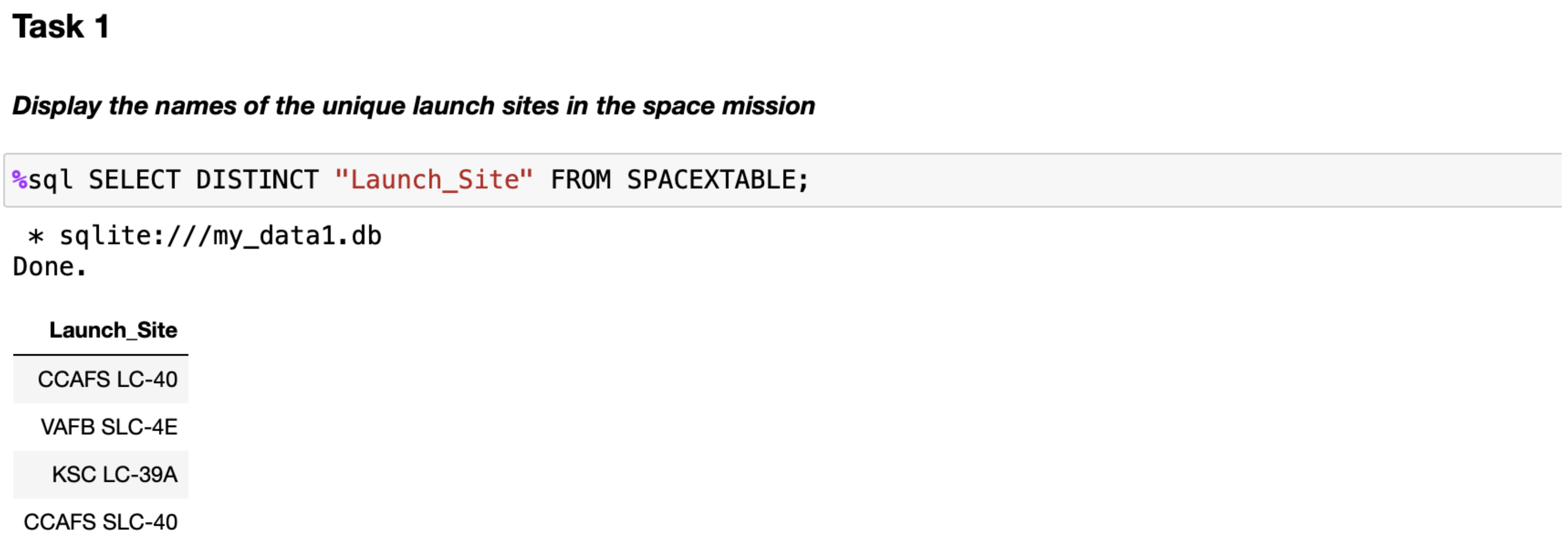
Use the function `get_dummies` and `features` dataframe to apply OneHotEncoder to the column `Orbits`, `LaunchSite`, `LandingPad`, and `Serial`. Assign the value to the variable `features_one_hot`, display the results using the method head. Your result dataframe must include all features including the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

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

# Concatenating the dummy variables with the original features dataframe
features_one_hot = pd.concat([features.drop(['Orbit', 'LaunchSite', 'LandingPad', 'Serial'], axis=1), dummies], axis

# Displaying the resulting dataframe
features_one_hot.head()
```

	FlightNumber	PayloadMass	Flights	GridFins	Reused	Legs	Block	ReusedCount	Orbit_ES-L1	Orbit_GEO	...	Serial_B1048	Serial_B1049	Serial_B1050	Se
0	1	6104.959412	1	False	False	False	1.0	0	0	0	...	0	0	0	
1	2	525.000000	1	False	False	False	1.0	0	0	0	...	0	0	0	
2	3	677.000000	1	False	False	False	1.0	0	0	0	...	0	0	0	
3	4	500.000000	1	False	False	False	1.0	0	0	0	...	0	0	0	
4	5	3170.000000	1	False	False	False	1.0	0	0	0	...	0	0	0	

5 rows × 80 columns

SQL



Task 1

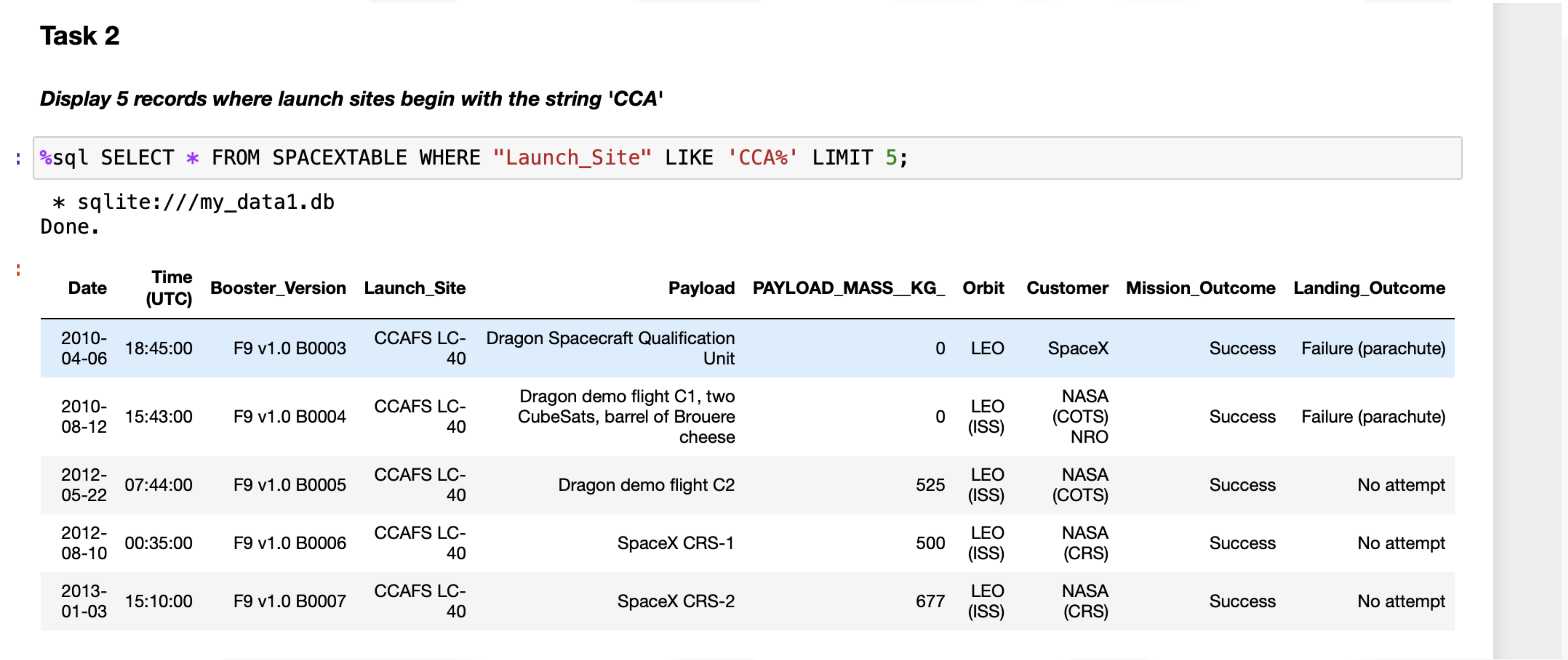
Display the names of the unique launch sites in the space mission

```
%sql SELECT DISTINCT "Launch_Site" FROM SPACEXTABLE;
```

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

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

SQL



Task 2

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

```
%sql SELECT * FROM SPACEXTABLE WHERE "Launch_Site" LIKE 'CCA%' LIMIT 5;
```

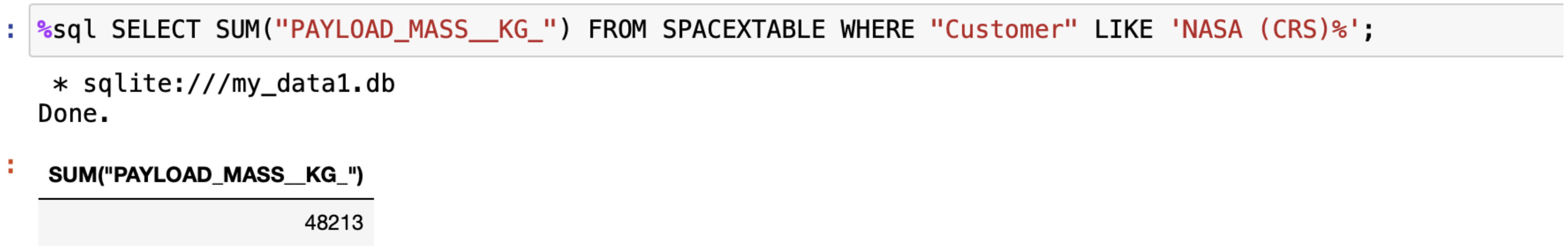
```
 * sqlite:///my_data1.db
Done.
```

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

SQL

Task 3

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

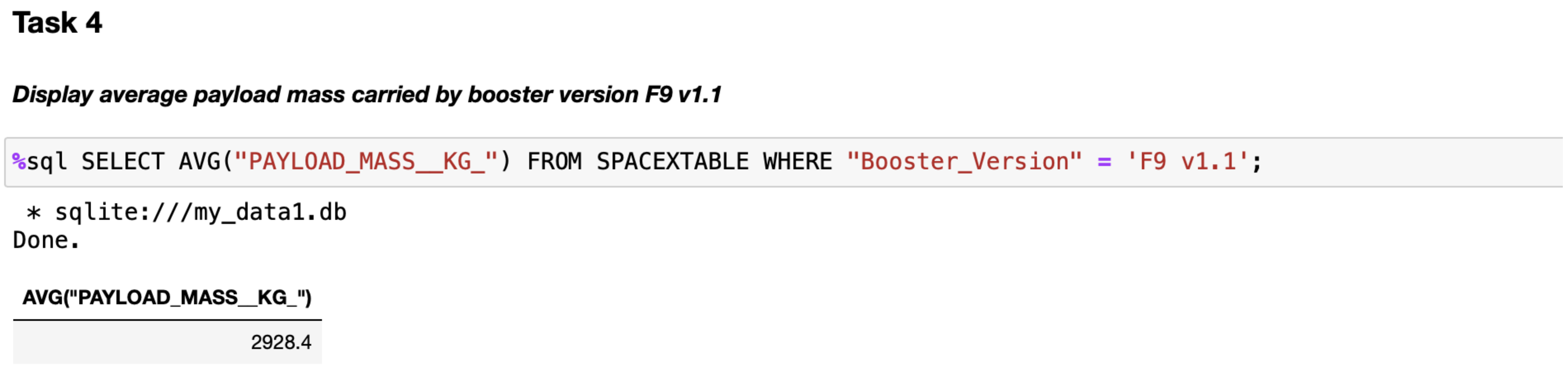


```
%sql SELECT SUM("PAYLOAD_MASS__KG_") FROM SPACEXTABLE WHERE "Customer" LIKE 'NASA (CRS)%';
```

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

SUM("PAYLOAD_MASS__KG_")
48213

SQL



Task 4

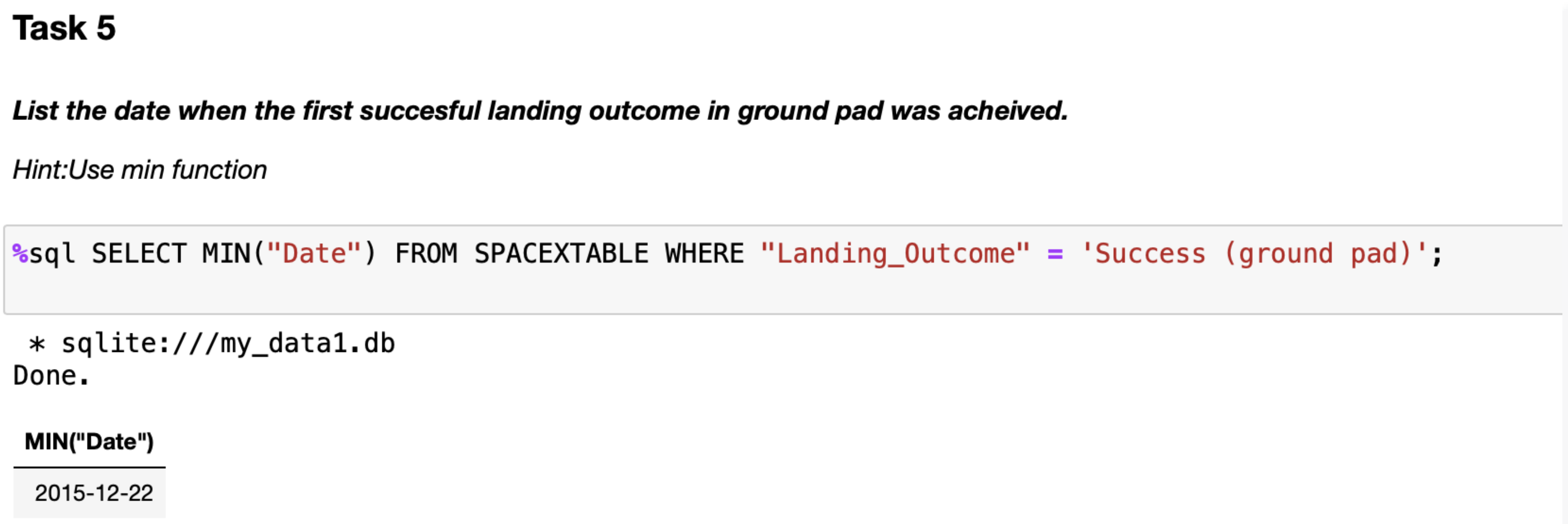
Display average payload mass carried by booster version F9 v1.1

```
%sql SELECT AVG("PAYLOAD_MASS__KG_") FROM SPACEXTABLE WHERE "Booster_Version" = 'F9 v1.1';
```

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

AVG("PAYLOAD_MASS__KG_")
2928.4

SQL



Task 5

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

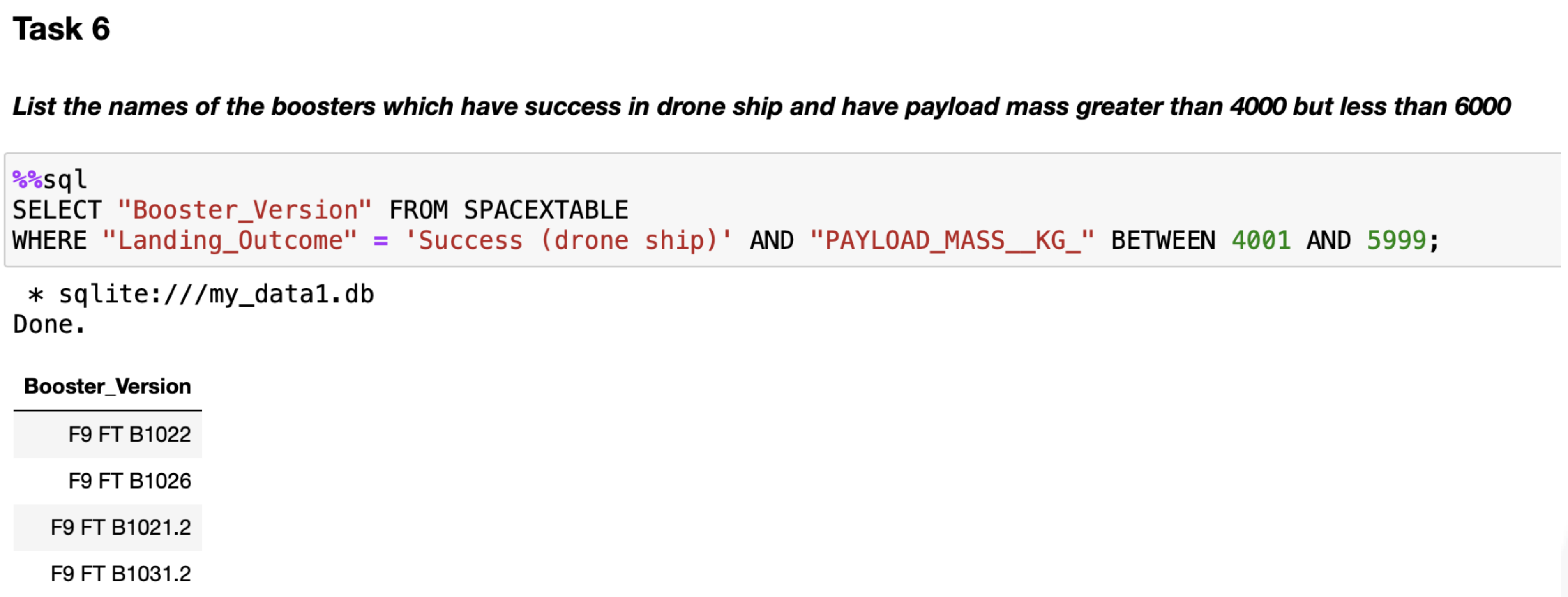
Hint:Use min function

```
%sql SELECT MIN("Date") FROM SPACEXTABLE WHERE "Landing_Outcome" = 'Success (ground pad)';
```

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

MIN("Date")
2015-12-22

SQL



Task 6

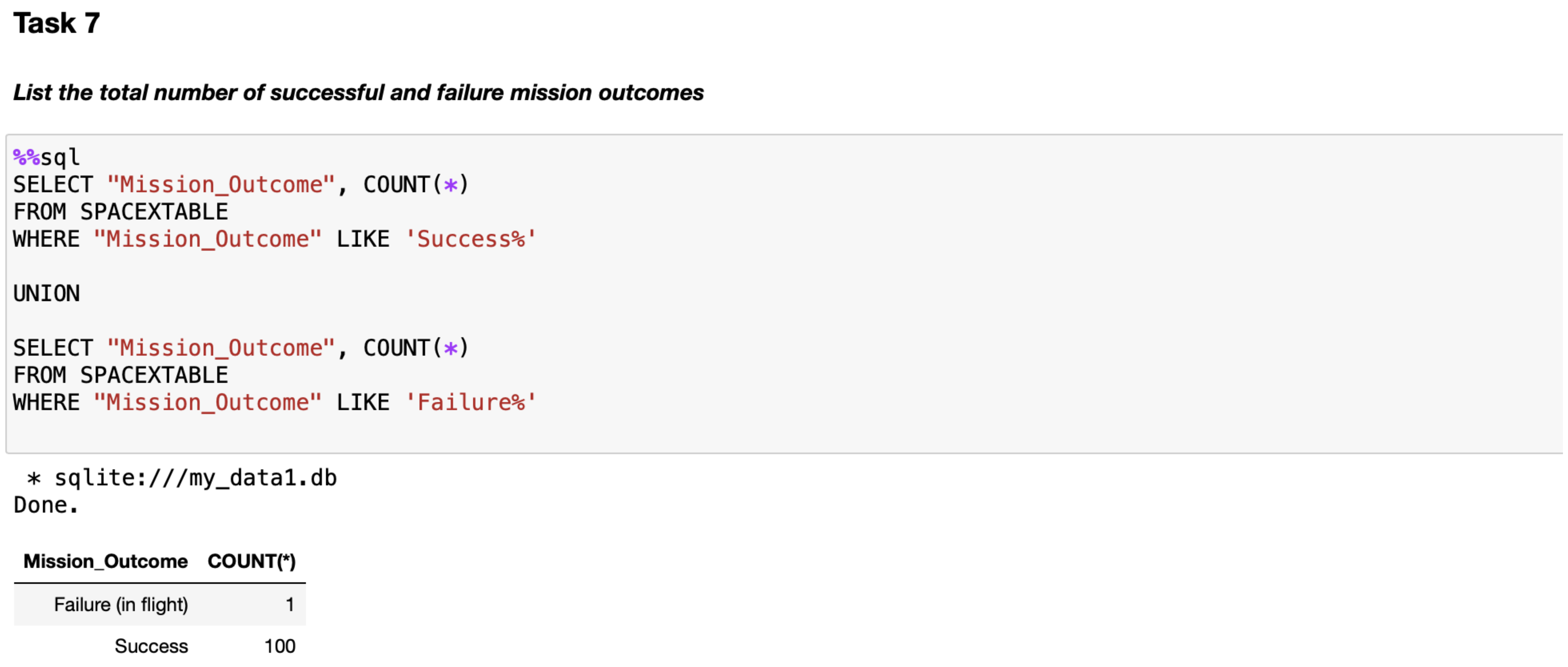
List the names of the boosters which have success in drone ship and have payload mass greater than 4000 but less than 6000

```
%%sql
SELECT "Booster_Version" FROM SPACEXTABLE
WHERE "Landing_Outcome" = 'Success (drone ship)' AND "PAYLOAD_MASS__KG_" BETWEEN 4001 AND 5999;
```

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

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

SQL



Task 7

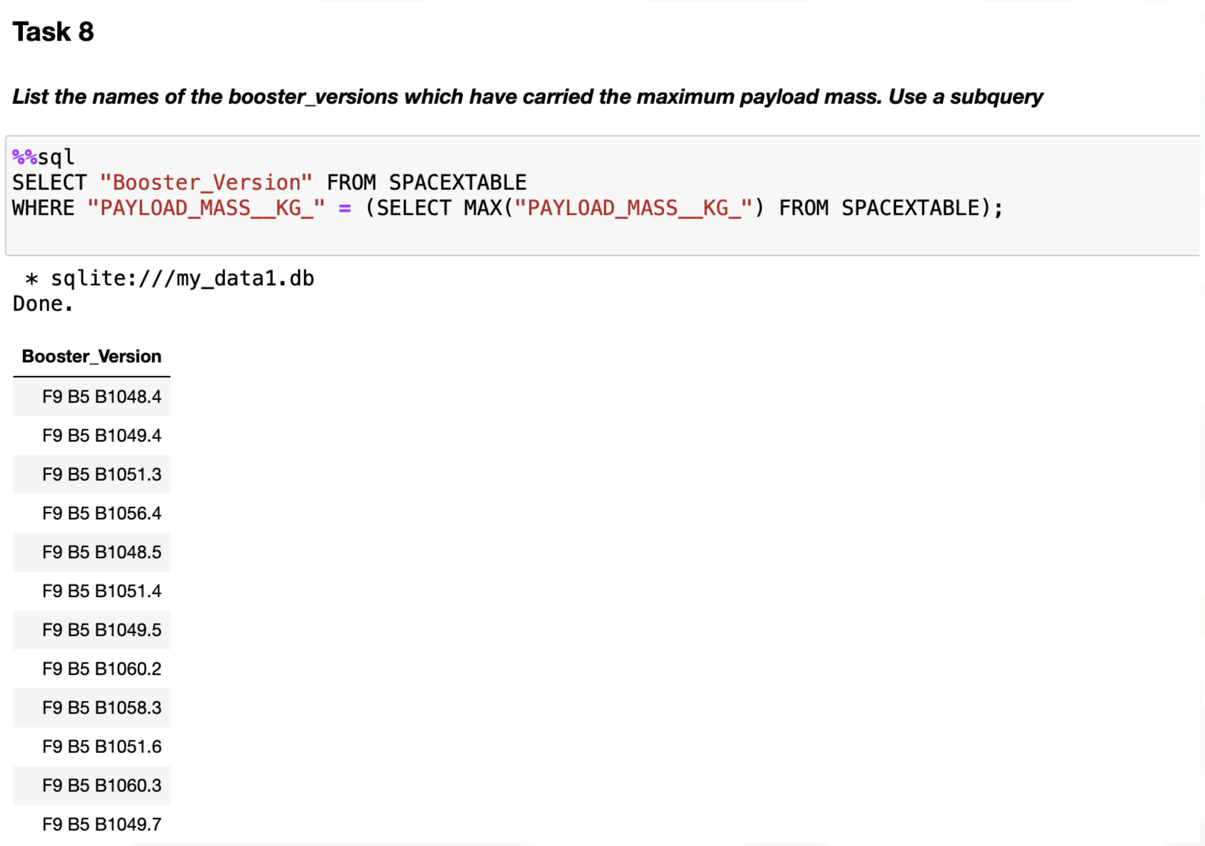
List the total number of successful and failure mission outcomes

```
%%sql
SELECT "Mission_Outcome", COUNT(*)
FROM SPACEXTABLE
WHERE "Mission_Outcome" LIKE 'Success%'

UNION

SELECT "Mission_Outcome", COUNT(*)
FROM SPACEXTABLE
WHERE "Mission_Outcome" LIKE 'Failure%'
```

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

Mission_Outcome	COUNT(*)
Failure (in flight)	1
Success	100

SQL



Task 8

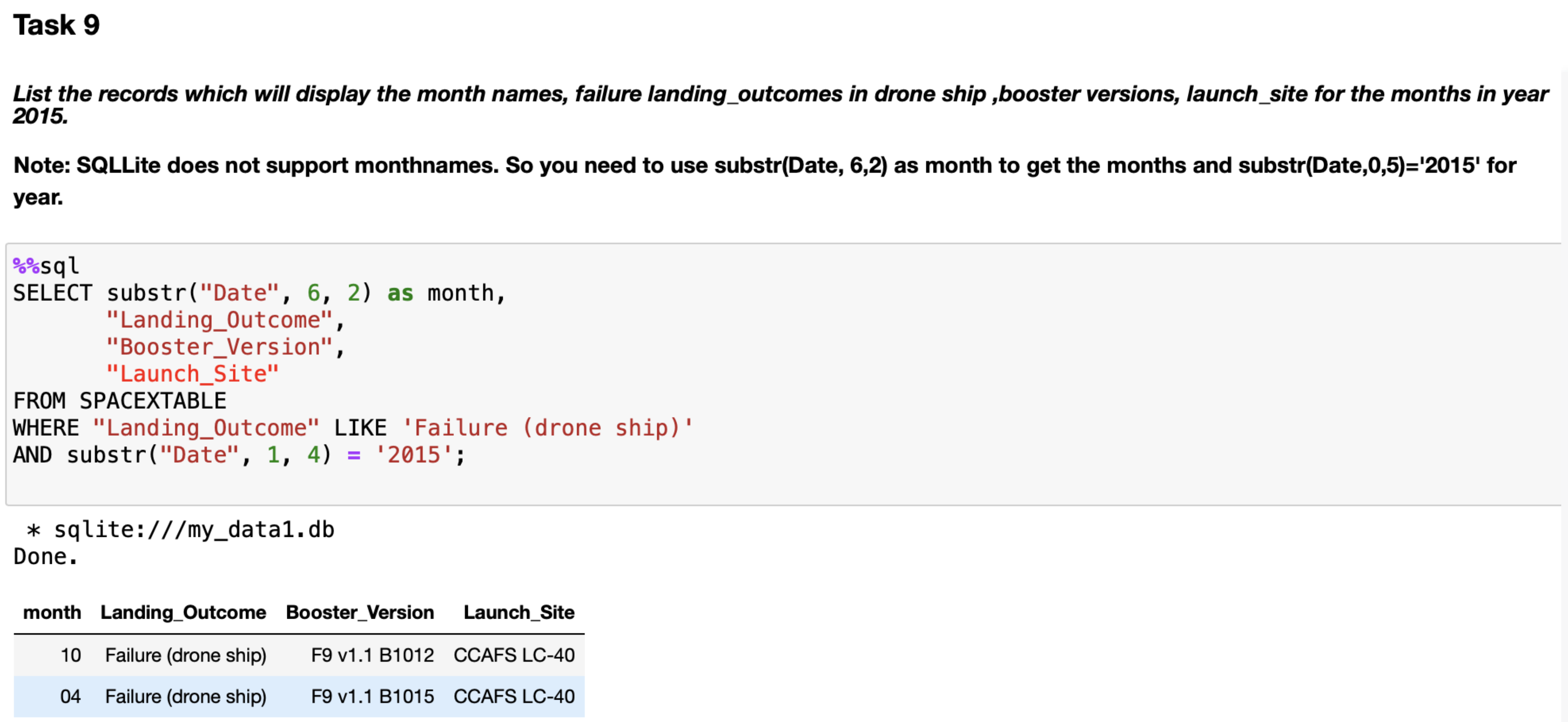
List the names of the booster_versions which have carried the maximum payload mass. Use a subquery

```
%%sql
SELECT "Booster_Version" FROM SPACEXTABLE
WHERE "PAYLOAD_MASS__KG_" = (SELECT MAX("PAYLOAD_MASS__KG_") FROM SPACEXTABLE);
```

```
 * 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

SQL



Task 9

List the records which will display the month names, failure landing_outcomes in drone ship ,booster versions, launch_site for the months in year 2015.

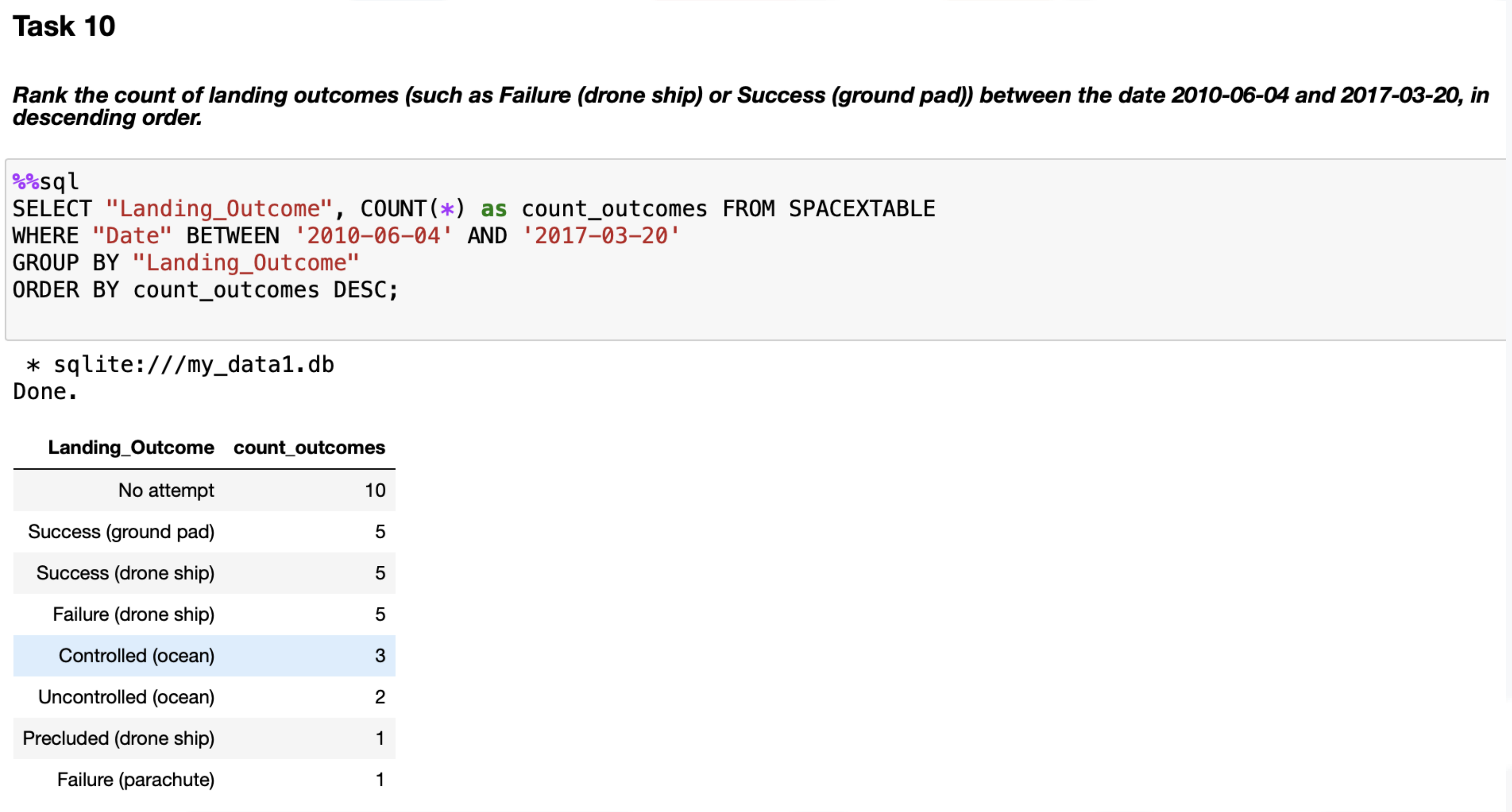
Note: SQLLite does not support monthnames. So you need to use substr(Date, 6,2) as month to get the months and substr(Date,0,5)='2015' for year.

```
%%sql
SELECT substr("Date", 6, 2) as month,
       "Landing_Outcome",
       "Booster_Version",
       "Launch_Site"
FROM SPACEXTABLE
WHERE "Landing_Outcome" LIKE 'Failure (drone ship)'
AND substr("Date", 1, 4) = '2015';
```

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

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

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.

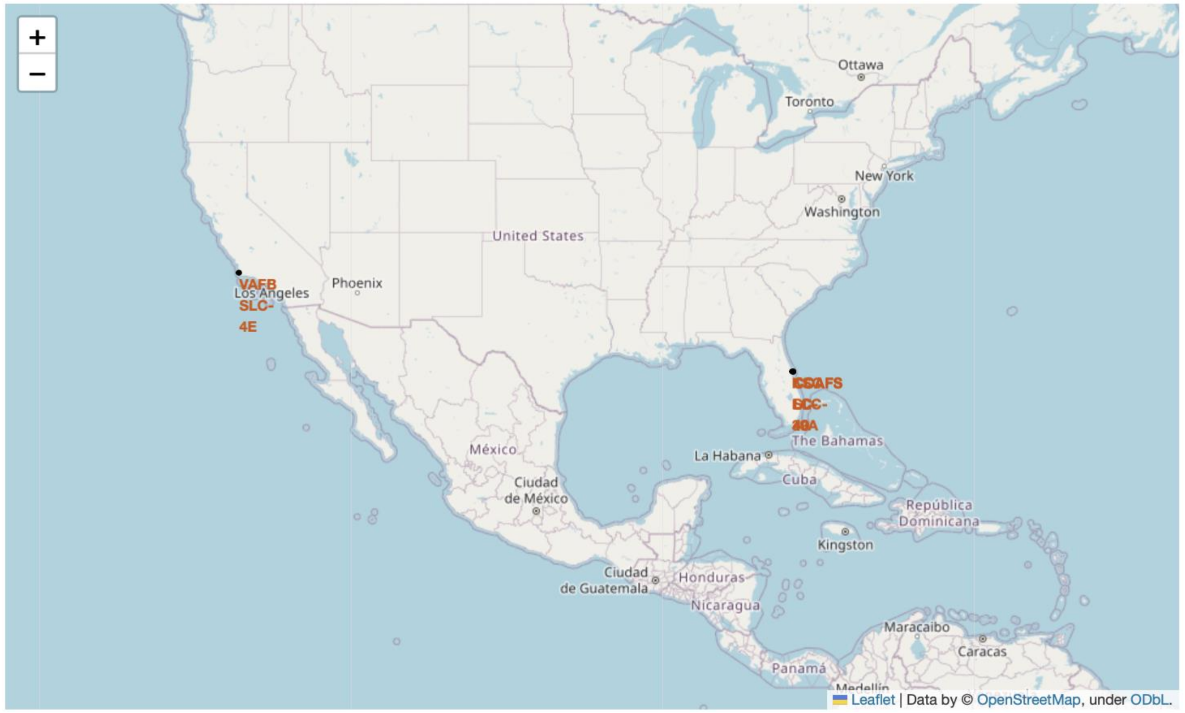
```
%%sql
SELECT "Landing_Outcome", COUNT(*) as count_outcomes FROM SPACEXTABLE
WHERE "Date" BETWEEN '2010-06-04' AND '2017-03-20'
GROUP BY "Landing_Outcome"
ORDER BY count_outcomes DESC;
```

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

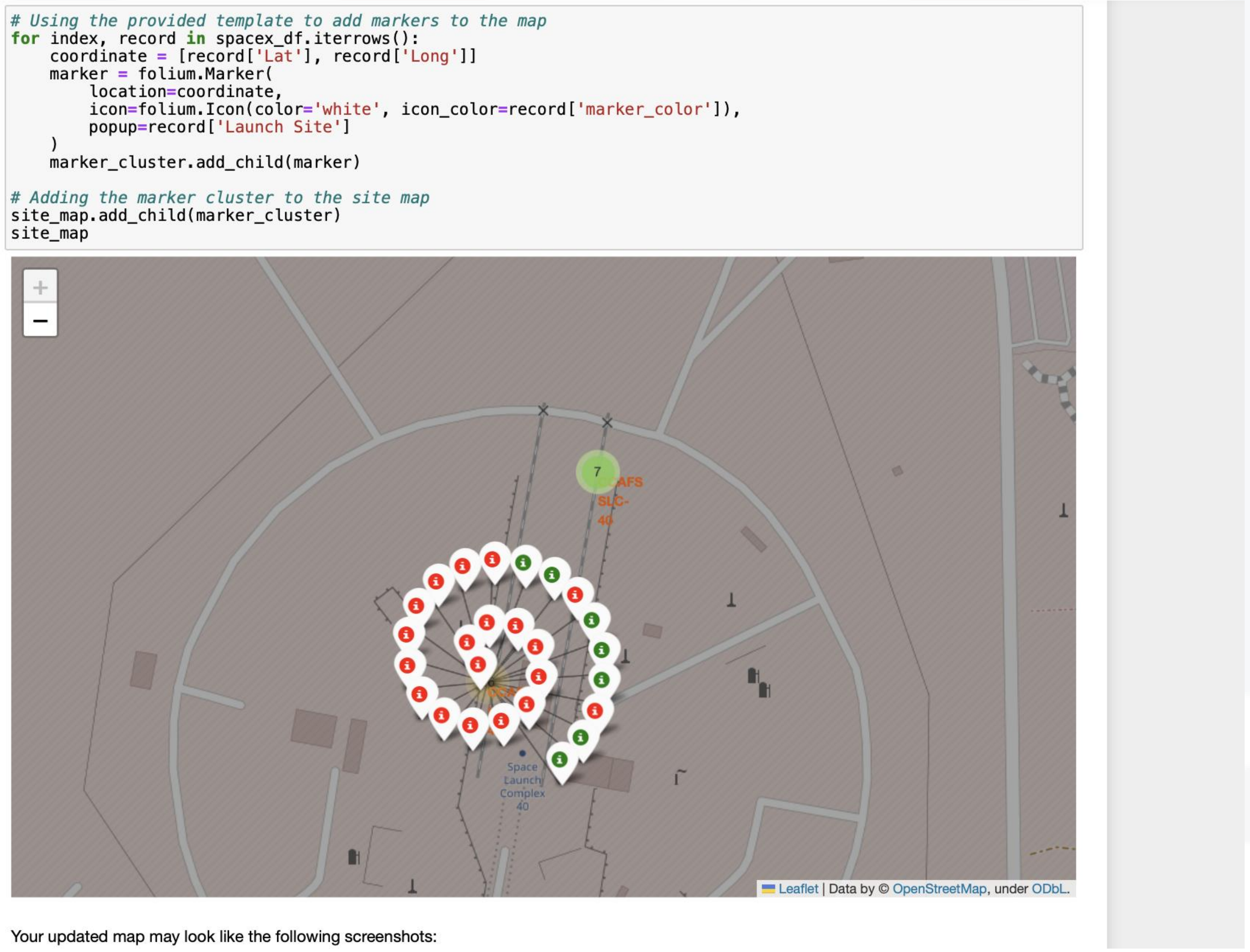
Landing_Outcome	count_outcomes
No attempt	10
Success (ground pad)	5
Success (drone ship)	5
Failure (drone ship)	5
Controlled (ocean)	3
Uncontrolled (ocean)	2
Precluded (drone ship)	1
Failure (parachute)	1

Interactive MAP

```python
# Initial the map
site_map = folium.Map(location=nasa_coordinate, zoom_start=4)
# For each launch site, add a Circle object based on its coordinate (Lat, Long) values. In addition, add Launc
for index, site in launch_sites_df.iterrows():
    coordinate = [site['Lat'], site['Long']]
    circle = folium.Circle(coordinate, radius=1000, color='#000000', fill=True).add_child(folium.Popup(site['L
    marker = folium.map.Marker(
        coordinate,
        icon=DivIcon(
            icon_size=(20,20),
            icon_anchor=(0,0),
            html='<div style="font-size: 12; color:#d35400;"><b>%s</b></div>' % site['Launch Site'],
            )
        )
    site_map.add_child(circle)
    site_map.add_child(marker)

site_map
```



Interactive MAP



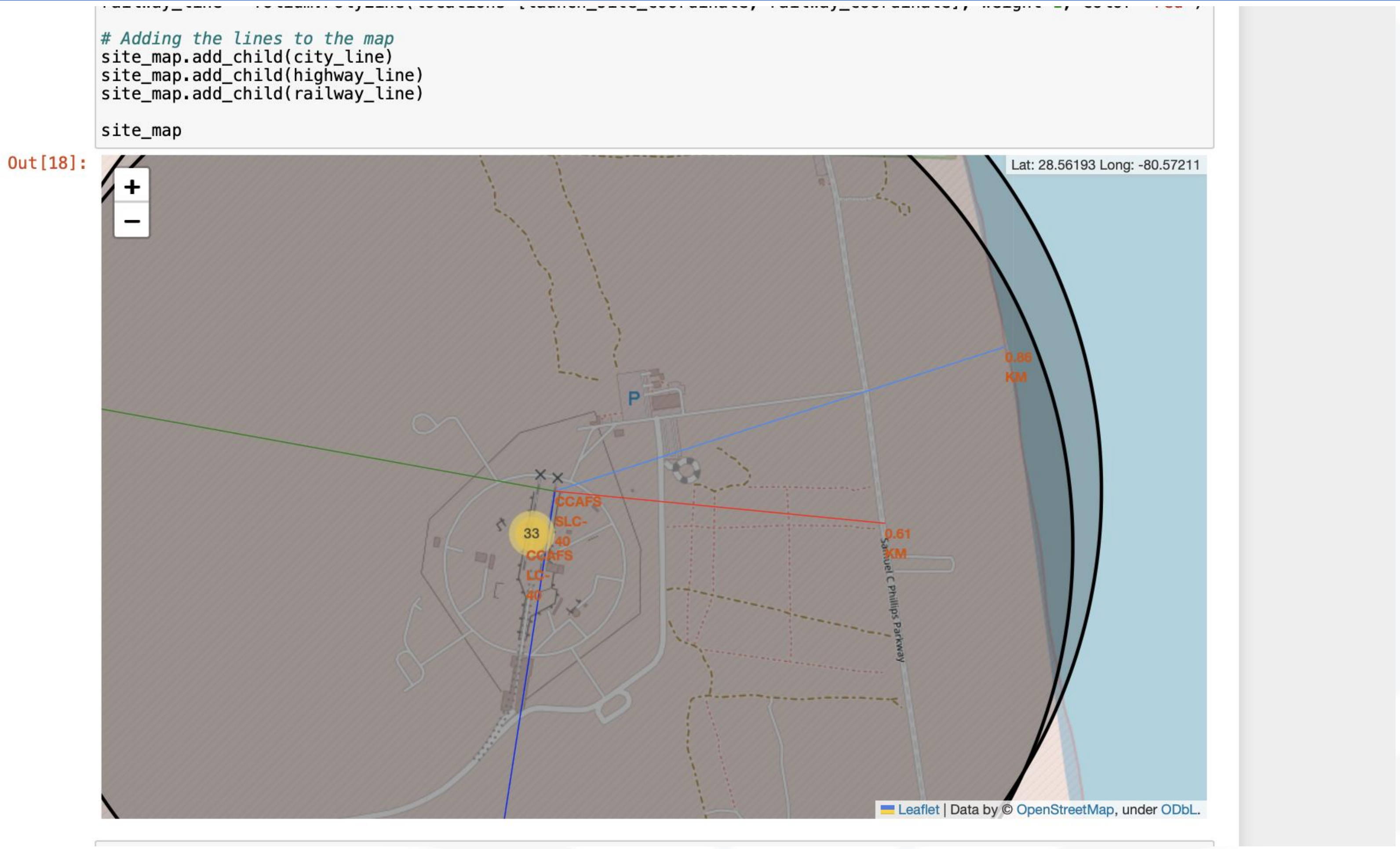
```
# Using the provided template to add markers to the map
for index, record in spacex_df.iterrows():
    coordinate = [record['Lat'], record['Long']]
    marker = folium.Marker(
        location=coordinate,
        icon=folium.Icon(color='white', icon_color=record['marker_color']),
        popup=record['Launch Site']
    )
    marker_cluster.add_child(marker)

# Adding the marker cluster to the site map
site_map.add_child(marker_cluster)
site_map
```

Your updated map may look like the following screenshots:

IBM Developer

SKILLS NETWORK

Interactive MAP

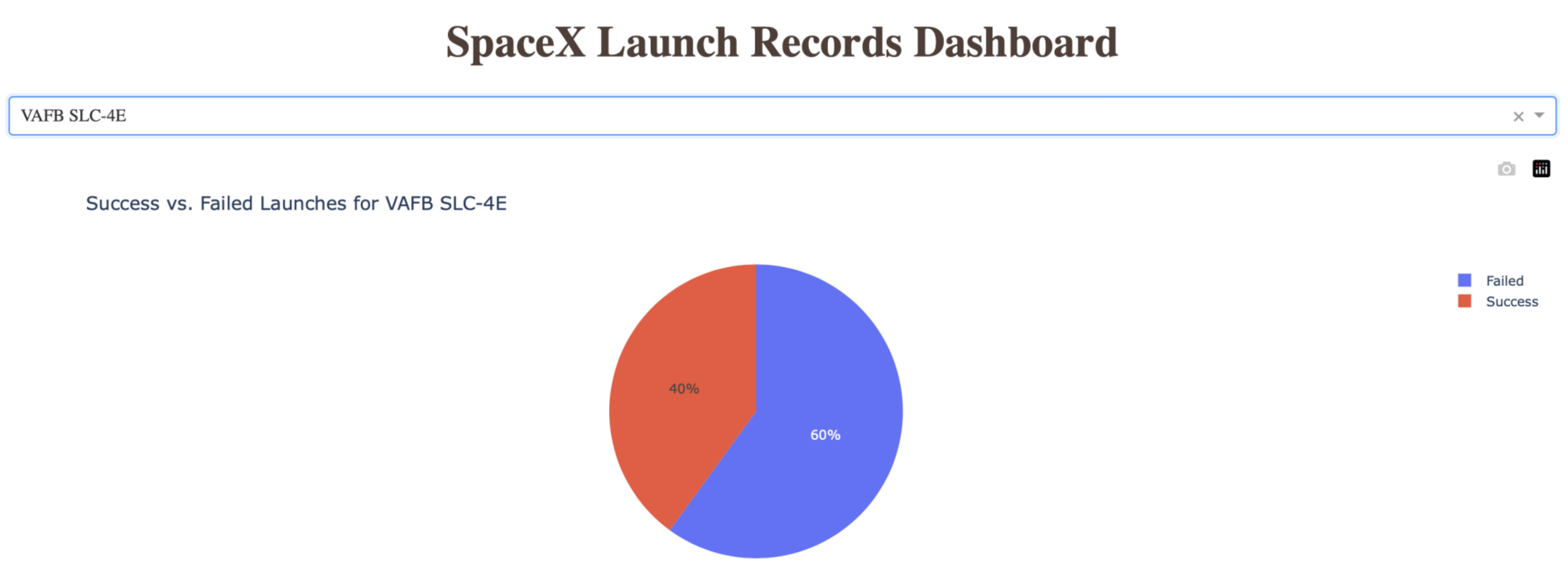


```
# Adding the lines to the map
site_map.add_child(city_line)
site_map.add_child(highway_line)
site_map.add_child(railway_line)

site_map
```

Out[18]:

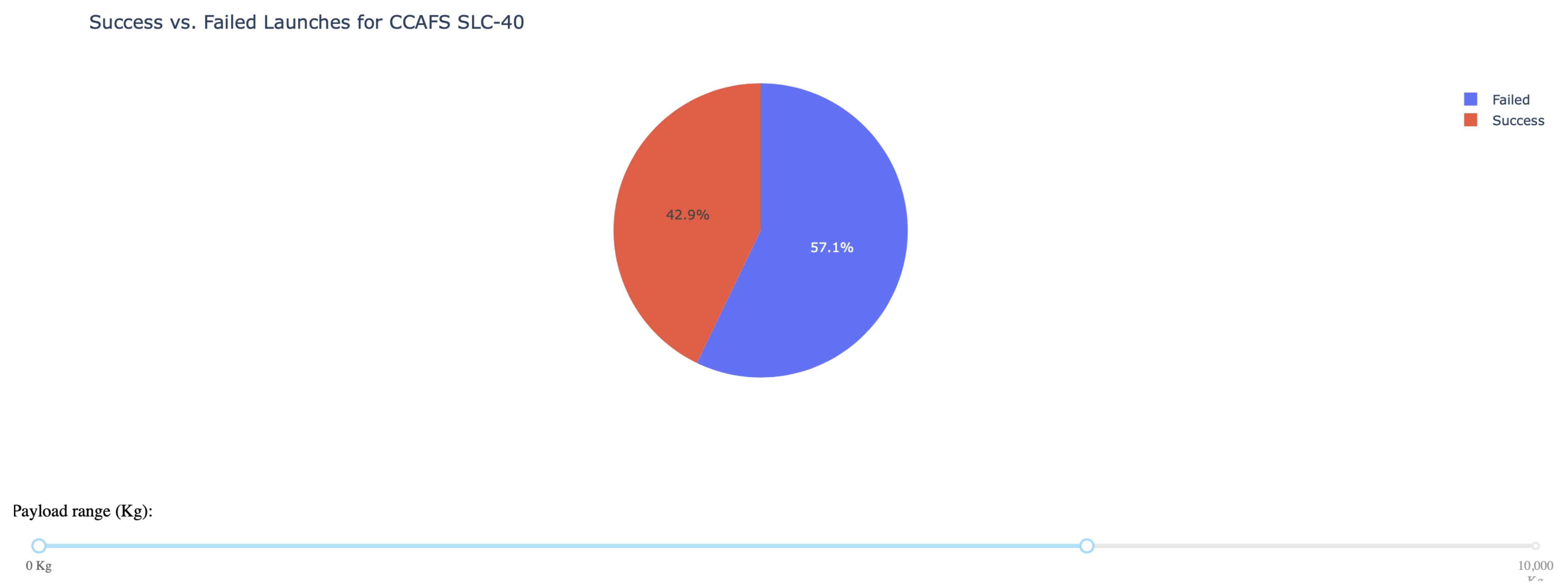
Lat: 28.56193 Long: -80.57211
0.86 KM
0.61 KM
CCAFS SLC-40
CCAFS LC-40
33
Samuel C Phillips Parkway
Leaflet | Data by © OpenStreetMap, under ODbL.

DASHBOARD TABS

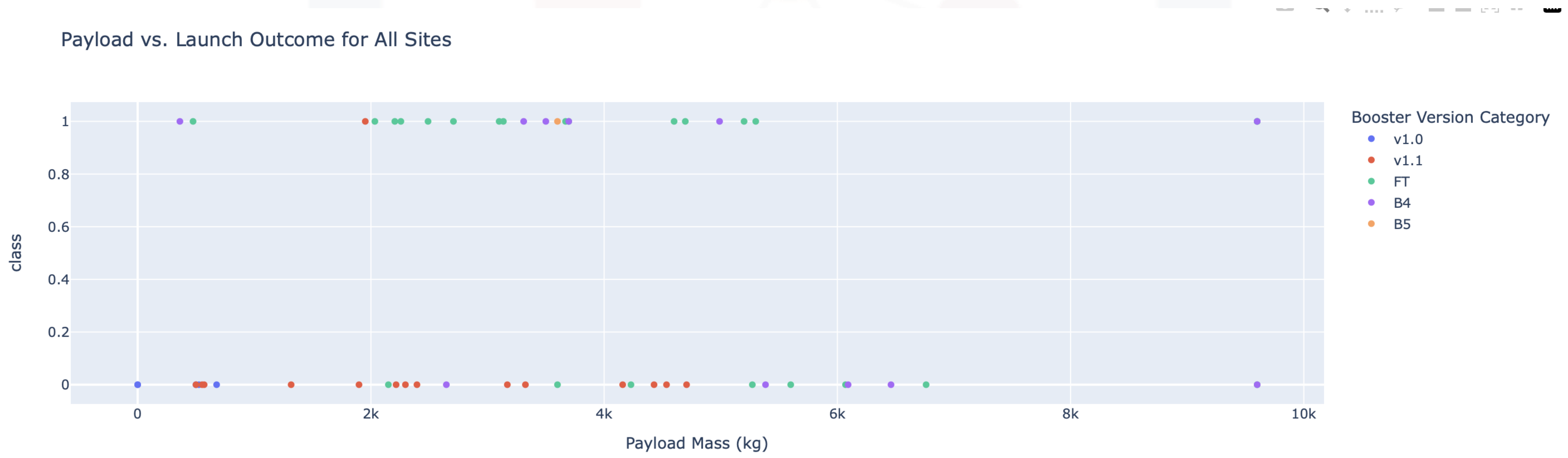

SpaceX Launch Records Dashboard

VAFB SLC-4E

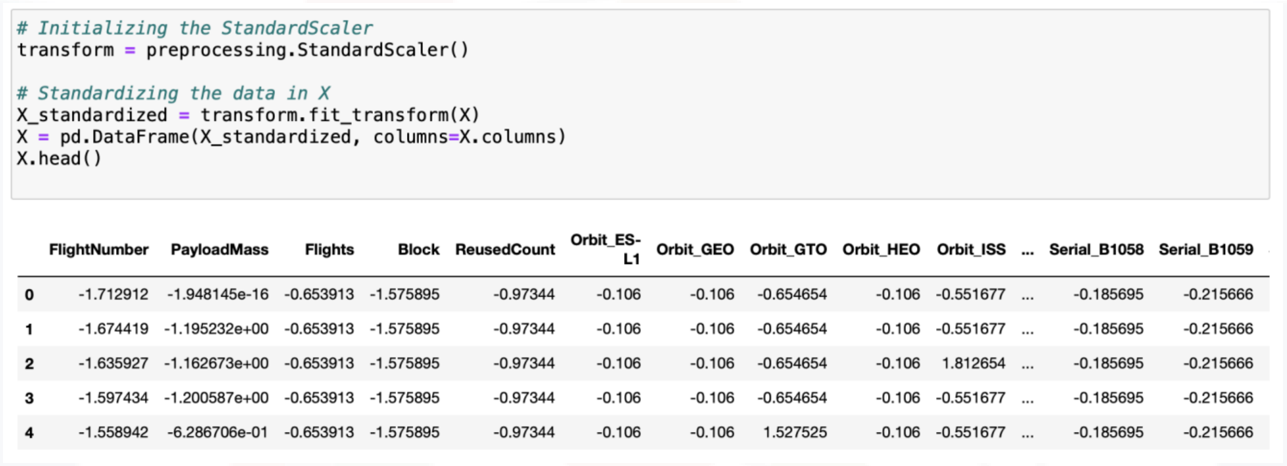
DASHBOARD TABS



DASHBOARD TABS

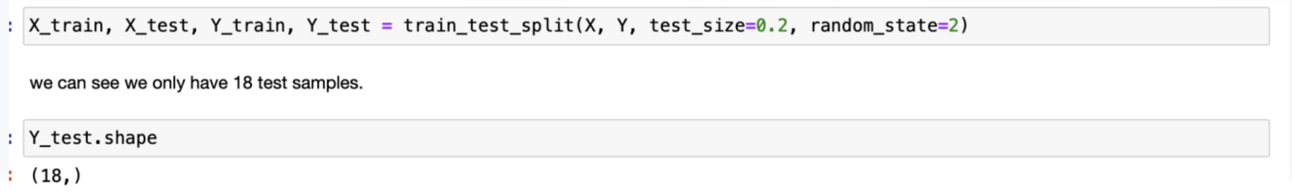


DISCUSSION



```python
# Initializing the StandardScaler
transform = preprocessing.StandardScaler()

# Standardizing the data in X
X_standardized = transform.fit_transform(X)
X = pd.DataFrame(X_standardized, columns=X.columns)
X.head()
```

	FlightNumber	PayloadMass	Flights	Block	ReusedCount	Orbit_ES-L1	Orbit_GEO	Orbit_GTO	Orbit_HEO	Orbit_ISS	...	Serial_B1058	Serial_B1059
0	-1.712912	-1.948145e-16	-0.653913	-1.575895	-0.97344	-0.106	-0.106	-0.654654	-0.106	-0.551677	...	-0.185695	-0.215666
1	-1.674419	-1.195232e+00	-0.653913	-1.575895	-0.97344	-0.106	-0.106	-0.654654	-0.106	-0.551677	...	-0.185695	-0.215666
2	-1.635927	-1.162673e+00	-0.653913	-1.575895	-0.97344	-0.106	-0.106	-0.654654	-0.106	1.812654	...	-0.185695	-0.215666
3	-1.597434	-1.200587e+00	-0.653913	-1.575895	-0.97344	-0.106	-0.106	-0.654654	-0.106	-0.551677	...	-0.185695	-0.215666
4	-1.558942	-6.286706e-01	-0.653913	-1.575895	-0.97344	-0.106	-0.106	1.527525	-0.106	-0.551677	...	-0.185695	-0.215666



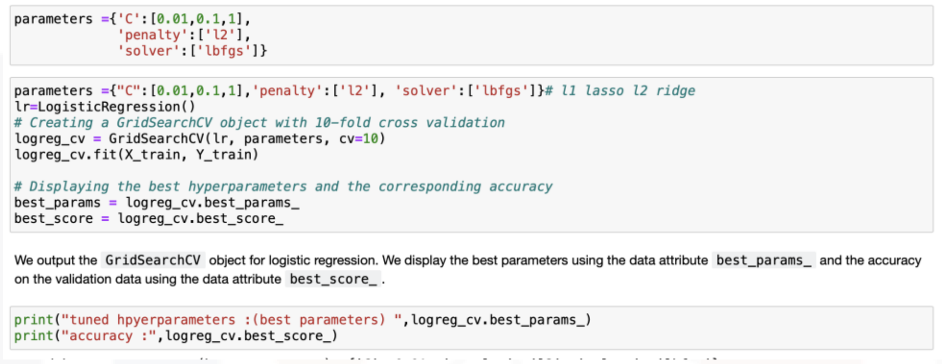
```python
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=2)
```

we can see we only have 18 test samples.

```python
Y_test.shape
```

```
(18,)
```

DISCUSSION

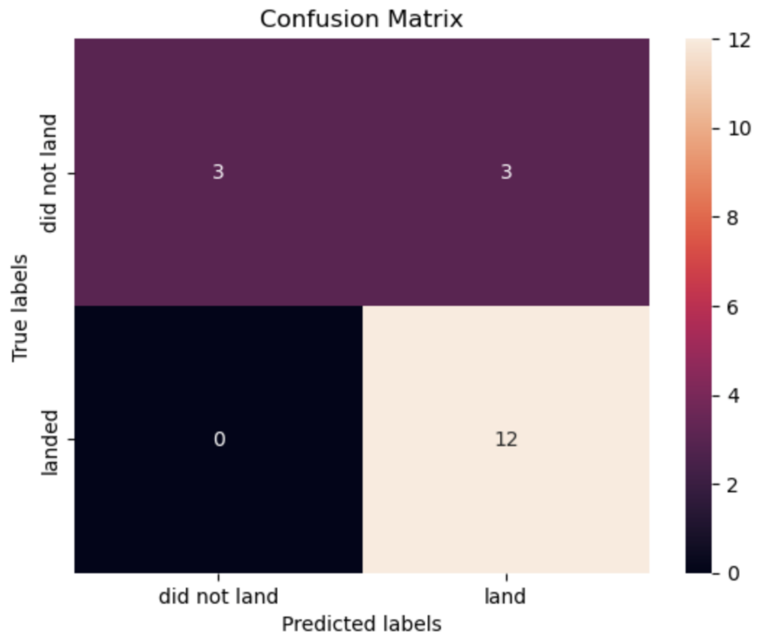


```
parameters ={'C':[0.01,0.1,1],
             'penalty':['l2'],
             'solver':['lbfgs']}
```

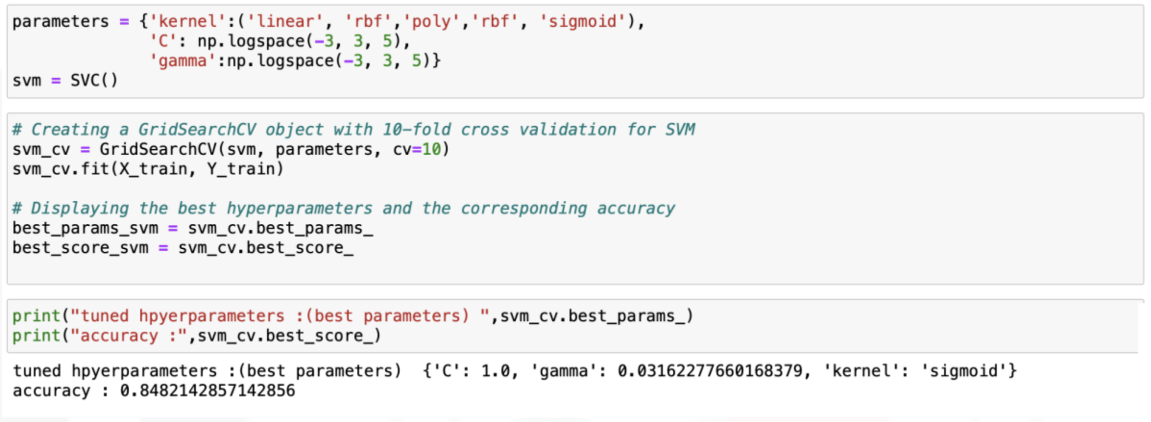
```
parameters ={"C":[0.01,0.1,1],'penalty':['l2'], 'solver':['lbfgs']}# l1 lasso l2 ridge
lr=LogisticRegression()
# Creating a GridSearchCV object with 10-fold cross validation
logreg_cv = GridSearchCV(lr, parameters, cv=10)
logreg_cv.fit(X_train, Y_train)

# Displaying the best hyperparameters and the corresponding accuracy
best_params = logreg_cv.best_params_
best_score = logreg_cv.best_score_
```

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_`.

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



DISCUSSION

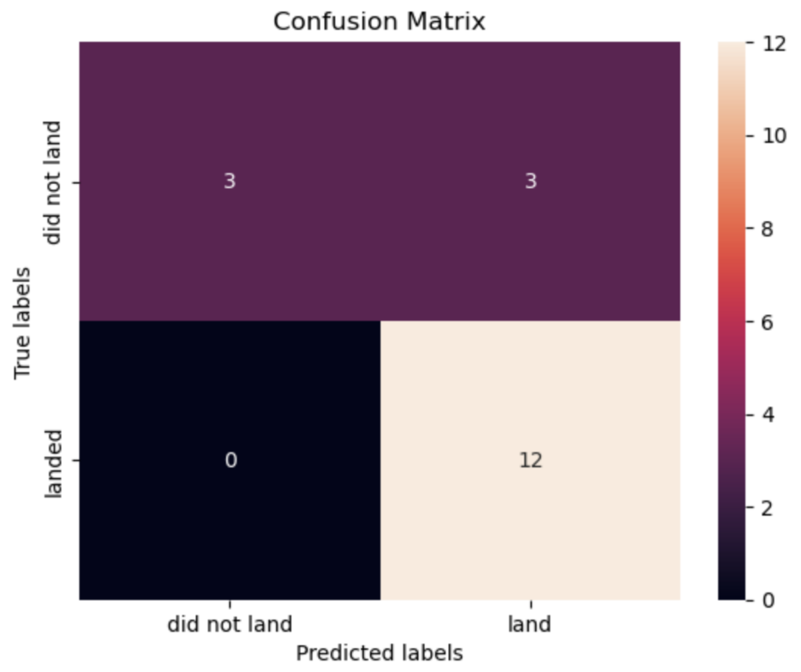


```
parameters = {'kernel':('linear', 'rbf','poly','rbf', 'sigmoid'),
              'C': np.logspace(-3, 3, 5),
              'gamma':np.logspace(-3, 3, 5)}
svm = SVC()
```

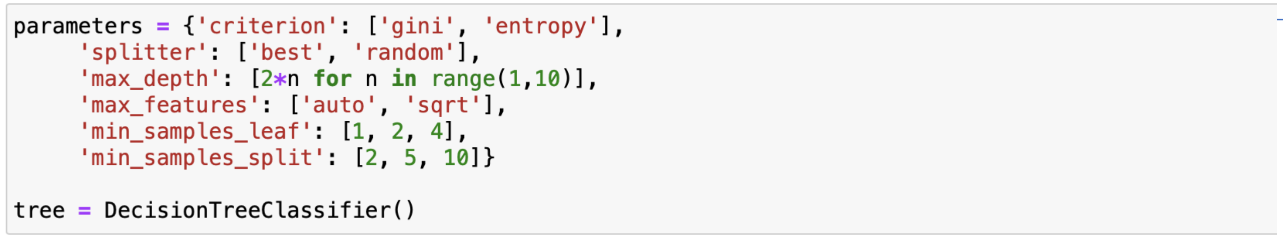
```
# Creating a GridSearchCV object with 10-fold cross validation for SVM
svm_cv = GridSearchCV(svm, parameters, cv=10)
svm_cv.fit(X_train, Y_train)

# Displaying the best hyperparameters and the corresponding accuracy
best_params_svm = svm_cv.best_params_
best_score_svm = svm_cv.best_score_
```

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

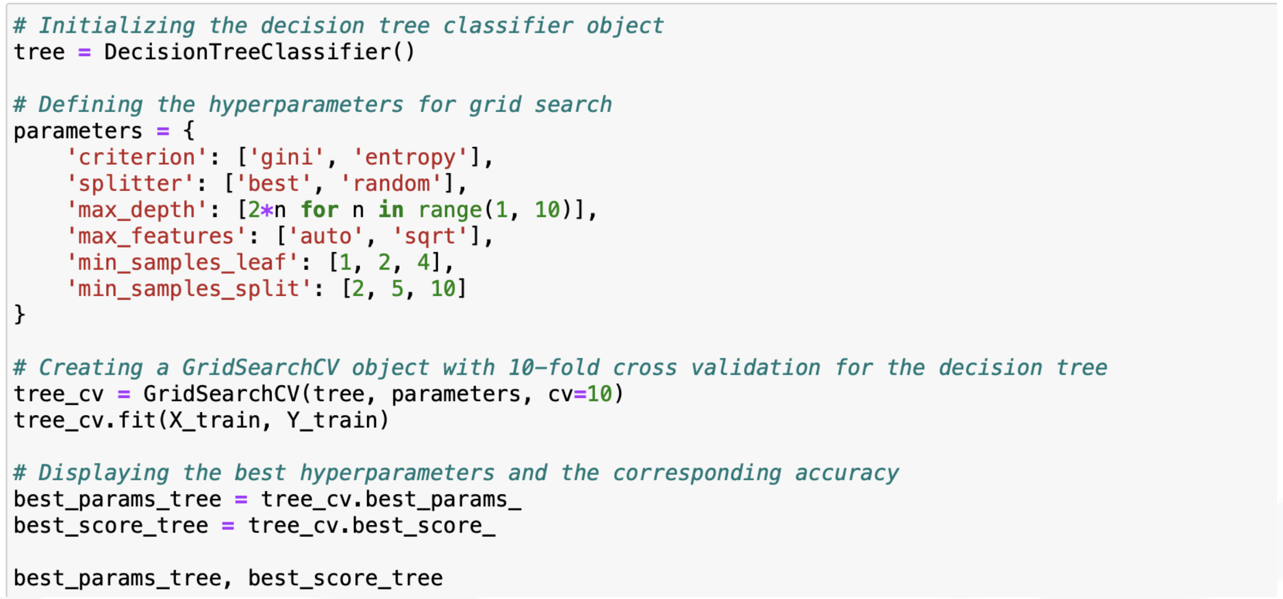
```
tuned hpyerparameters :(best parameters)  {'C': 1.0, 'gamma': 0.03162277660168379, 'kernel': 'sigmoid'}
accuracy : 0.8482142857142856
```



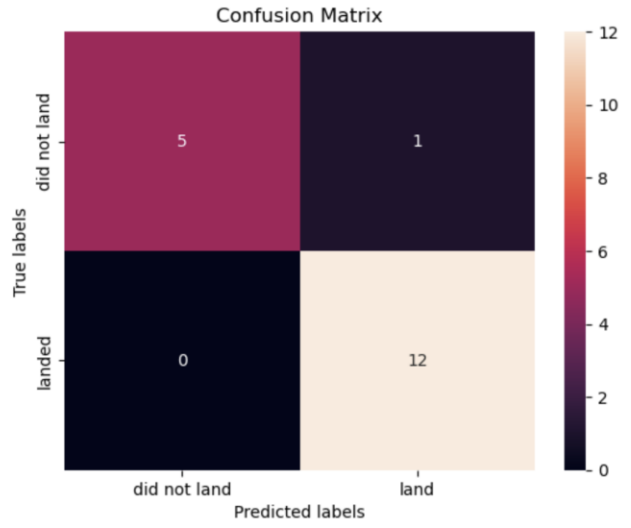
DISCUSSION



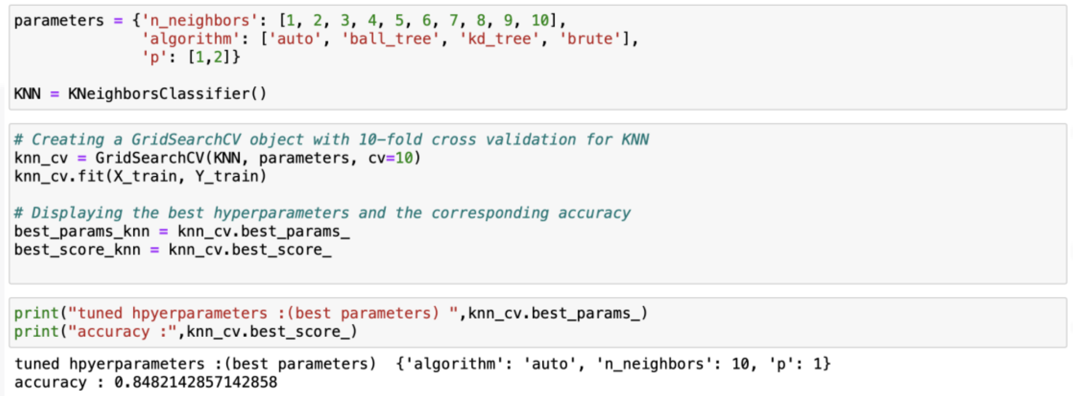
```
parameters = {'criterion': ['gini', 'entropy'],
     'splitter': ['best', 'random'],
     'max_depth': [2*n for n in range(1,10)],
     'max_features': ['auto', 'sqrt'],
     'min_samples_leaf': [1, 2, 4],
     'min_samples_split': [2, 5, 10]}

tree = DecisionTreeClassifier()
```



```
# Initializing the decision tree classifier object
tree = DecisionTreeClassifier()

# Defining the hyperparameters for grid search
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]
}

# Creating a GridSearchCV object with 10-fold cross validation for the decision tree
tree_cv = GridSearchCV(tree, parameters, cv=10)
tree_cv.fit(X_train, Y_train)

# Displaying the best hyperparameters and the corresponding accuracy
best_params_tree = tree_cv.best_params_
best_score_tree = tree_cv.best_score_

best_params_tree, best_score_tree
```



DISCUSSION

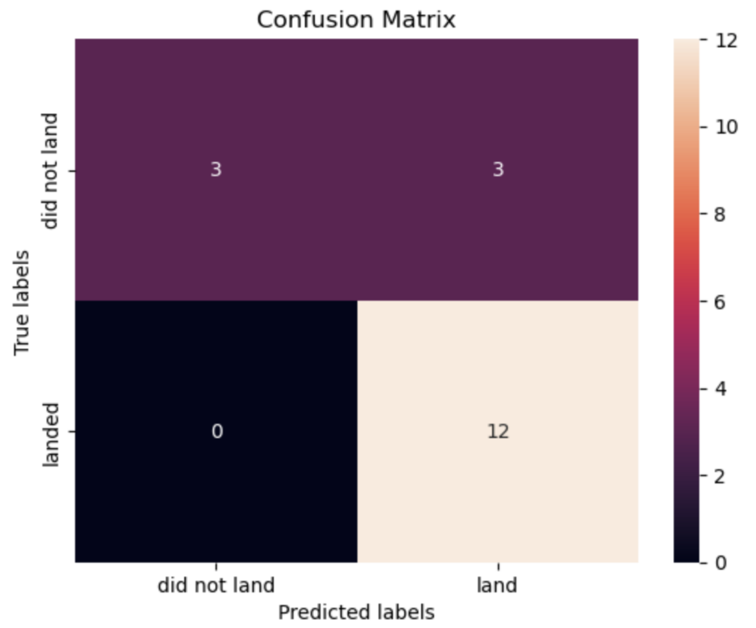


```
parameters = {'n_neighbors': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
              'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute'],
              'p': [1,2]}

KNN = KNeighborsClassifier()
```

```
# Creating a GridSearchCV object with 10-fold cross validation for KNN
knn_cv = GridSearchCV(KNN, parameters, cv=10)
knn_cv.fit(X_train, Y_train)

# Displaying the best hyperparameters and the corresponding accuracy
best_params_knn = knn_cv.best_params_
best_score_knn = knn_cv.best_score_
```

```
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
```



CONCLUSION

- Most machine learnings have similar accuracy
- Decision tree was selected as the best method with a 83.4% accuracy
- Lauch site is a huge factor into the success of landing of first stage

APPENDIX

- Resources:
  - SpaceX API: https://api.spacexdata.com/v4/
  - Wikipedia: https://en.wikipedia.org/wiki/List_of_Falcon_9_and_Falcon_Heavy_launches
  - GitHub repository: https://github.com/akalabri/Capstone-Project-