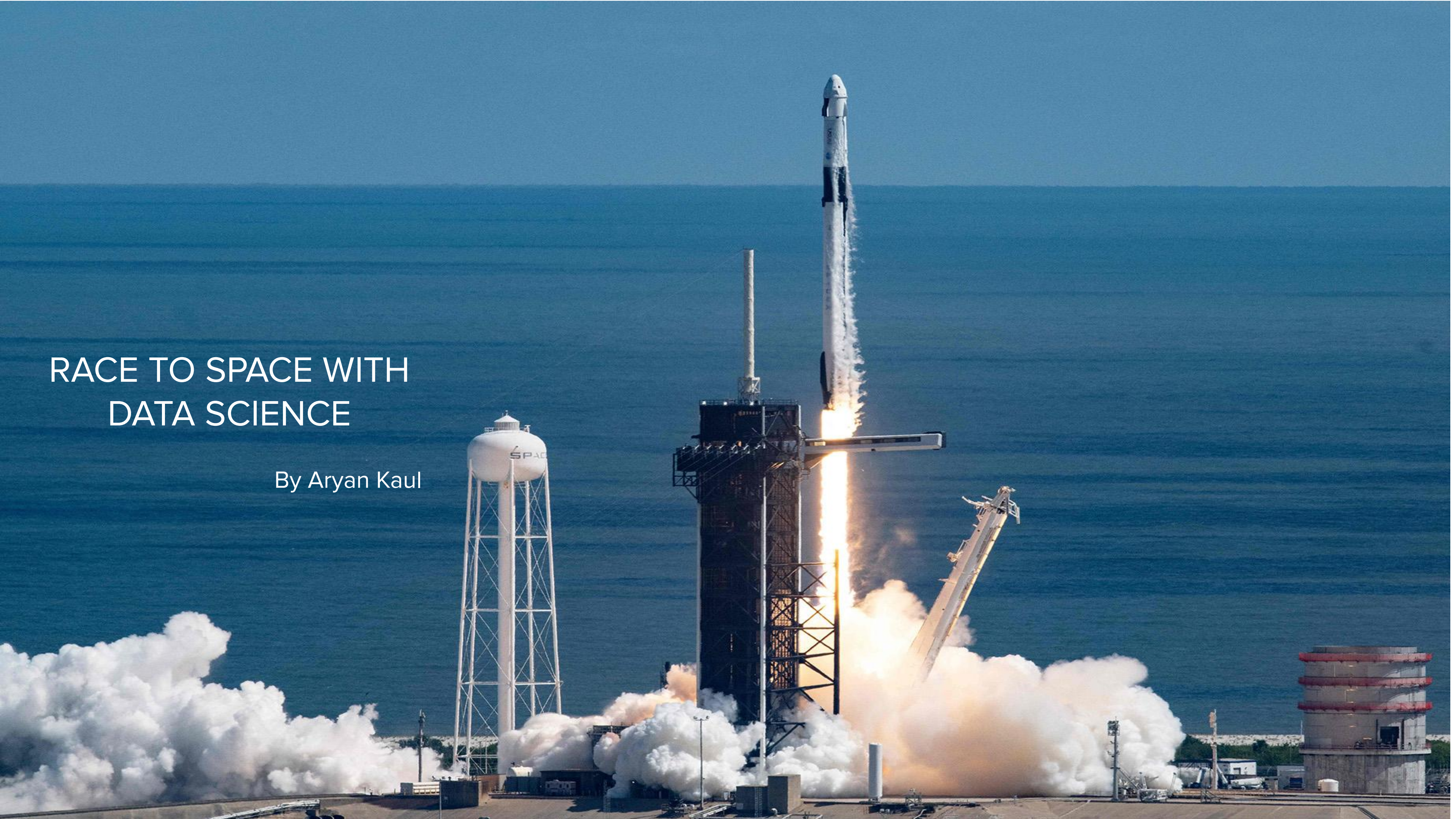


RACE TO SPACE WITH DATA SCIENCE

By Aryan Kaul

Outline

Executive Summary

.Summary Of Methodology

- Data Collection Through API
- Data Collection Through Web scraping
- Data Wrangling
- Exploratory Data Analysis With SQL
- Exploratory Data Analysis With Data Visualization
- Machine Learning Prediction

. Summary Of Result

- Exploratory Data Analysis Results
- Results Of Prediction

Introduction

Project background and context

SpaceX advertises Falcon 9 rocket launches on its website with a cost of 62 million dollars; other providers cost upward of 165 million dollars each, much of the savings is because SpaceX can reuse the first stage. Therefore, if we can determine if the first stage will land, we can determine the cost of a launch. This information can be used if an alternate company wants to bid against spaceX for a rocket launch. This goal of the project is to create a machine learning pipeline to predict if the first stage will land successfully.

Problems you want to find answers

-What factors determine if the rocket will land successfully?

-The interaction amongst various features that determine the success rate of a successful landing.

-What operating conditions needs to be in place to ensure a successful landing program.

Data Collection

- The data was collected using various methods

-Data collection was done using get request to the SpaceX API.

-Next, we decoded the response content as a Json using .json() function call and turn it into a pandas dataframe using .json_normalize().

-We then cleaned the data, checked for missing values and fill in missing values where necessary.

-In addition, we performed web scraping from Wikipedia for Falcon 9 launch records with BeautifulSoup.

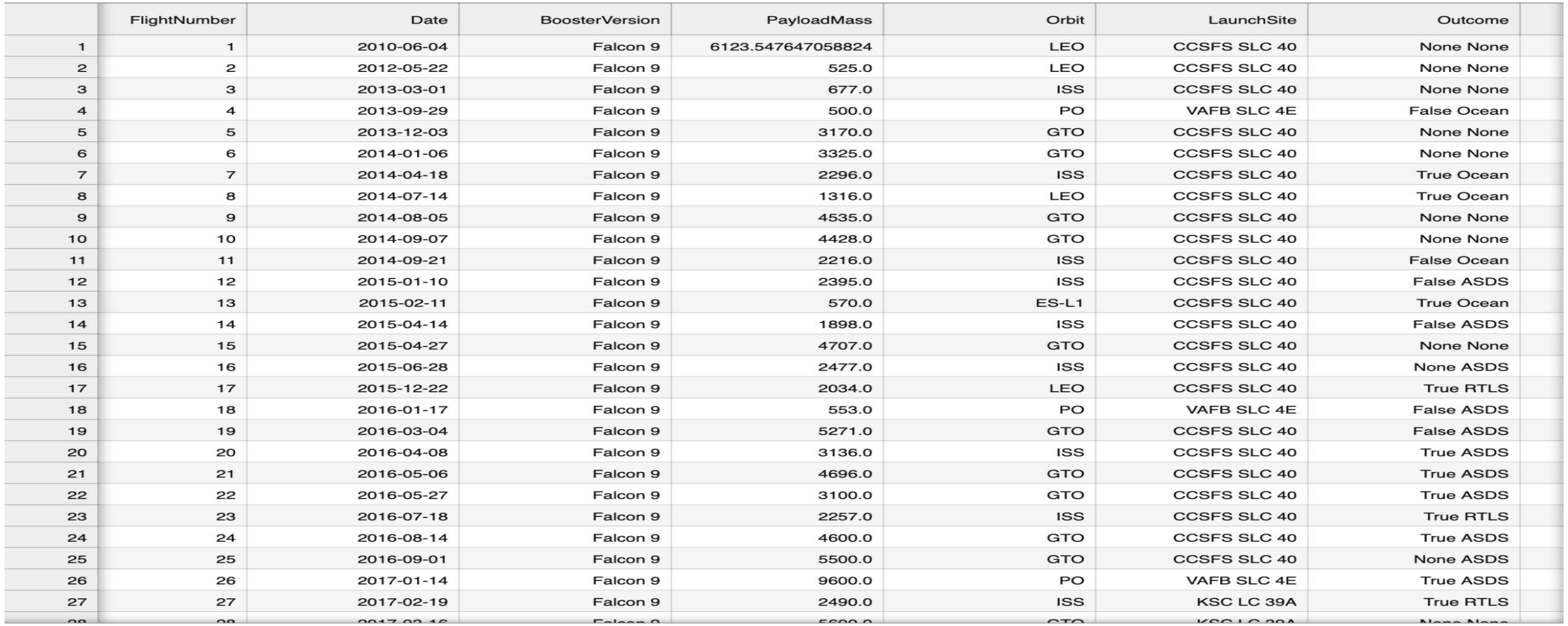
-The objective was to extract the launch records as HTML table, parse the table and convert it to a pandas dataframe for future analysis.

Data Collection With API

Steps For Data Collection:

1. From the url of the API was retrieved all the data through requests library in python and received it as a json file.
2. Then we converted that data to a python dataframe so that we can perform suitable operations on it.
3. Further we started cleaning the data since we needed data related to only Falcon9 we filtered the data for Falcon9 in a new dataframe.
4. After that as a final step of data cleaning we has to handle the empty data in ordered to deal with the empty data in the payload column we replaced the empty data with the mean value of the column but leveraging pandas library.
5. Then at last we save the dataframe as a .csv file for further data analysis

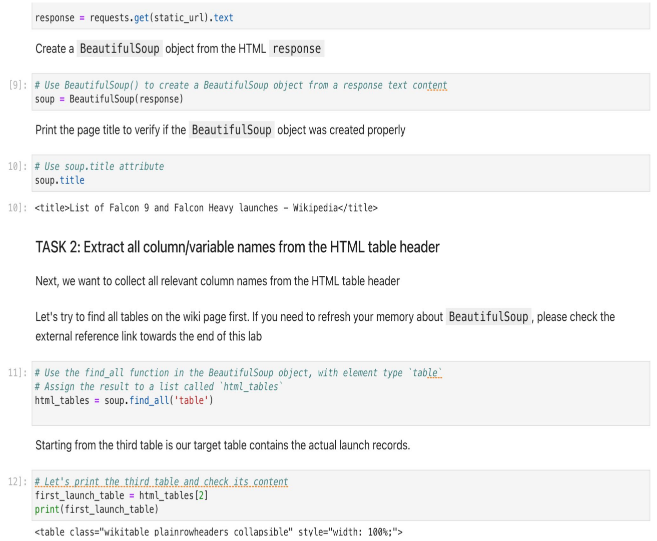
Data Collection Through API Output



	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome
1	1	2010-06-04	Falcon 9	6123.547647058824	LEO	CCSFS SLC 40	None None
2	2	2012-05-22	Falcon 9	525.0	LEO	CCSFS SLC 40	None None
3	3	2013-03-01	Falcon 9	677.0	ISS	CCSFS SLC 40	None None
4	4	2013-09-29	Falcon 9	500.0	PO	VAFB SLC 4E	False Ocean
5	5	2013-12-03	Falcon 9	3170.0	GTO	CCSFS SLC 40	None None
6	6	2014-01-06	Falcon 9	3325.0	GTO	CCSFS SLC 40	None None
7	7	2014-04-18	Falcon 9	2296.0	ISS	CCSFS SLC 40	True Ocean
8	8	2014-07-14	Falcon 9	1316.0	LEO	CCSFS SLC 40	True Ocean
9	9	2014-08-05	Falcon 9	4535.0	GTO	CCSFS SLC 40	None None
10	10	2014-09-07	Falcon 9	4428.0	GTO	CCSFS SLC 40	None None
11	11	2014-09-21	Falcon 9	2216.0	ISS	CCSFS SLC 40	False Ocean
12	12	2015-01-10	Falcon 9	2395.0	ISS	CCSFS SLC 40	False ASDS
13	13	2015-02-11	Falcon 9	570.0	ES-L1	CCSFS SLC 40	True Ocean
14	14	2015-04-14	Falcon 9	1898.0	ISS	CCSFS SLC 40	False ASDS
15	15	2015-04-27	Falcon 9	4707.0	GTO	CCSFS SLC 40	None None
16	16	2015-06-28	Falcon 9	2477.0	ISS	CCSFS SLC 40	None ASDS
17	17	2015-12-22	Falcon 9	2034.0	LEO	CCSFS SLC 40	True RTLS
18	18	2016-01-17	Falcon 9	553.0	PO	VAFB SLC 4E	False ASDS
19	19	2016-03-04	Falcon 9	5271.0	GTO	CCSFS SLC 40	False ASDS
20	20	2016-04-08	Falcon 9	3136.0	ISS	CCSFS SLC 40	True ASDS
21	21	2016-05-06	Falcon 9	4696.0	GTO	CCSFS SLC 40	True ASDS
22	22	2016-05-27	Falcon 9	3100.0	GTO	CCSFS SLC 40	True ASDS
23	23	2016-07-18	Falcon 9	2257.0	ISS	CCSFS SLC 40	True RTLS
24	24	2016-08-14	Falcon 9	4600.0	GTO	CCSFS SLC 40	True ASDS
25	25	2016-09-01	Falcon 9	5500.0	GTO	CCSFS SLC 40	None ASDS
26	26	2017-01-14	Falcon 9	9600.0	PO	VAFB SLC 4E	True ASDS
27	27	2017-02-19	Falcon 9	2490.0	ISS	KSC LC 39A	True RTLS

Data Collection Through Web Scraping

1. To extract more data on Falcon 9 launch records we need to web scrap some data from wikipedia
2. We do so by using the BeautifulSoup Library by finding all the tables in the page and selecting data from the one that we want
3. Then we create a new dataframe with the respective columns is the table and append the values to that column



```
response = requests.get(static_url).text
```

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

```
[9]: # Use BeautifulSoup() to create a BeautifulSoup object from a response text content
soup = BeautifulSoup(response)
```

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

```
10]: # Use soup.title attribute
soup.title
```

```
10]: <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

Let's try to find all tables on the wiki page first. If you need to refresh your memory about `BeautifulSoup`, please check the external reference link towards the end of this lab

```
11]: # Use the find_all function in the BeautifulSoup object, with element type `table`
# Assign the result to a list called `html_tables`
html_tables = soup.find_all('table')
```

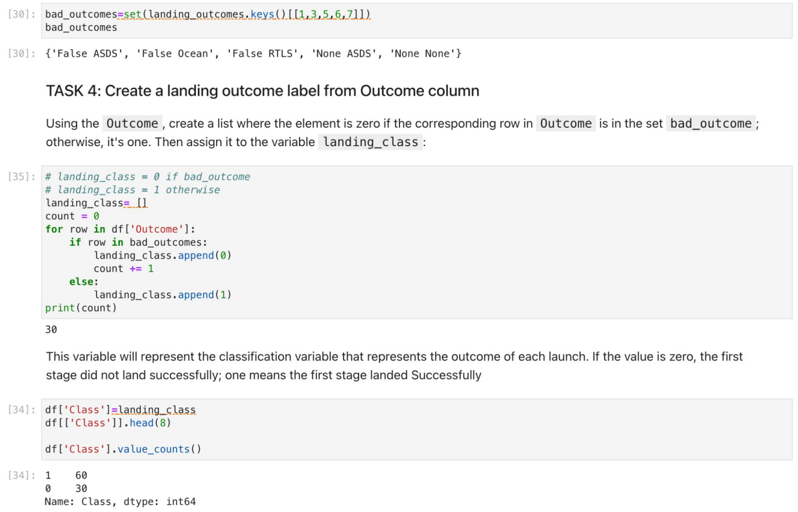
Starting from the third table is our target table contains the actual launch records.

```
12]: # Let's print the third table and check its content
first_launch_table = html_tables[2]
print(first_launch_table)
```

```
<table class="wikitable plainrowheaders collapsible" style="width: 100%;">
```

Data Wrangling

1. During Data Wrangling we look at the distribution of different launch sites and orbits of the spacecraft
2. Then we create a new column that shows if the spacecraft landed successfully or not in form of numerical data so that we can use it in further data analysis and prediction



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

[30]: {'False ASDS', 'False Ocean', 'False RTLS', 'None ASDS', 'None None'}
```

TASK 4: Create a landing outcome label from Outcome column

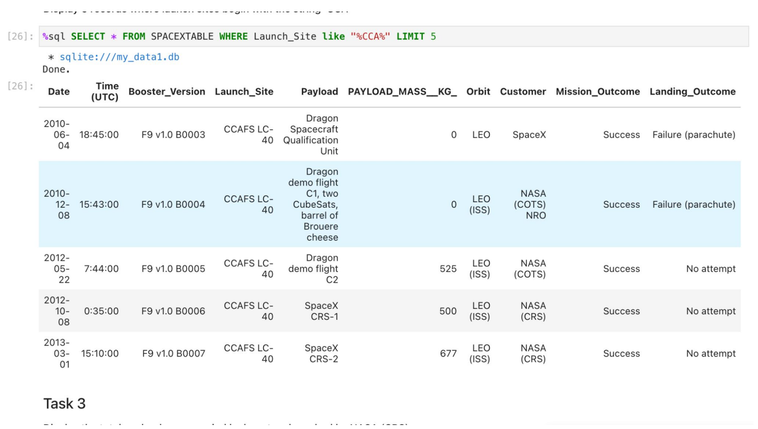
Using the `Outcome`, create a list where the element is zero if the corresponding row in `Outcome` is in the set `bad_outcome`; otherwise, it's one. Then assign it to the variable `landing_class`:

```
[35]: # landing_class = 0 if bad_outcome
      # landing_class = 1 otherwise
      landing_class= []
      count = 0
      for row in df['Outcome']:
          if row in bad_outcomes:
              landing_class.append(0)
              count += 1
          else:
              landing_class.append(1)
      print(count)

      30
```

This variable will represent the classification variable that represents the outcome of each launch. If the value is zero, the first stage did not land successfully; one means the first stage landed Successfully

```
[34]: df['Class']=landing_class
      df[['Class']].head(8)

      df['Class'].value_counts()

[34]: 1    60
      0    30
      Name: Class, dtype: int64
```

Exploratory Data Analysis with SQL



```
[26]: %sql SELECT * FROM SPACEXTABLE WHERE Launch_Site like "%CCA%" LIMIT 5

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

[26]:

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

Task 3

•We loaded the SpaceX dataset into a PostgreSQL database without leaving the jupyter notebook.

•We applied EDA with SQL to get insight from the data. We wrote queries to find out for instance:

-The names of unique launch sites in the space mission.

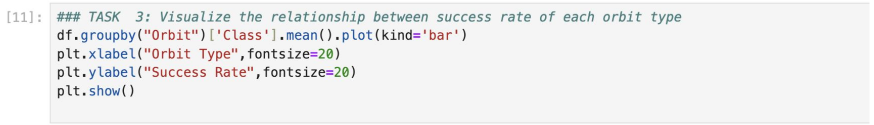
-The total payload mass carried by boosters launched by NASA (CRS)

-The average payload mass carried by booster version F9 v1.1

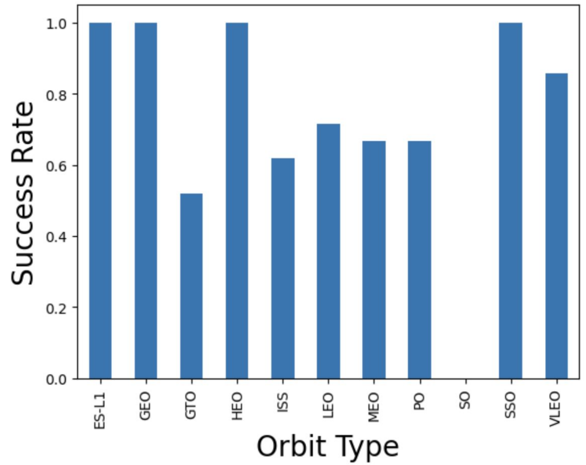
-The total number of successful and failure mission outcomes

Exploratory Data Analysis Through Visualization

•We explored the data by visualizing the relationship between flight number and launch Site, payload and launch site, success rate of each orbit type, flight number and orbit type, the launch success yearly trend.



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



Plotly Dashboard For Data Visualization

•We built an interactive dashboard with Plotly dash

•We plotted pie charts showing the total launches by a certain sites

•We plotted scatter graph showing the relationship with Outcome and Payload Mass (Kg) for the different booster version.

Machine Learning Prediction With Data

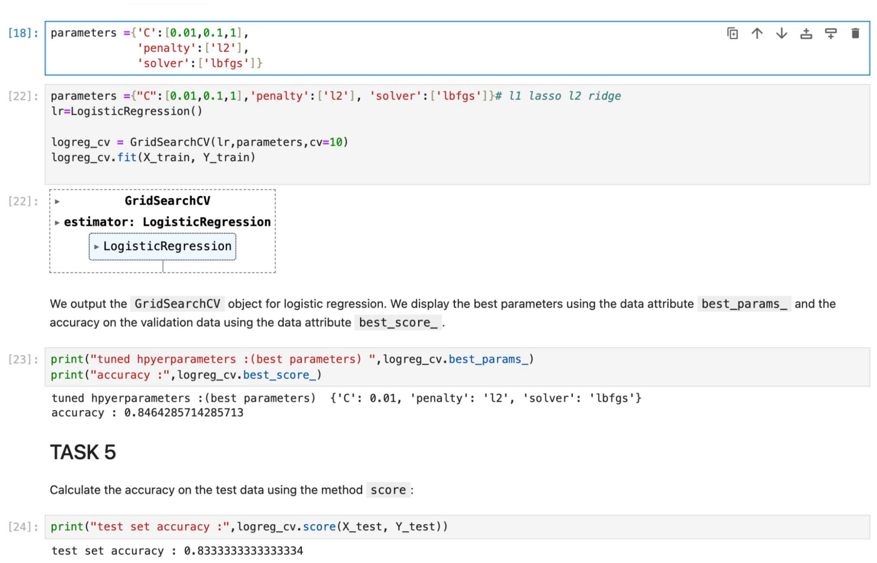
1. Since we are working on finding if a successful landing will take place ,our instinctive approach should be to start with a classification model and in the course we have learnt and implemented several classification machine learning algorithms like logistic regression, support vector machine, decision tree and K nearest neighbour.
2. We go through all this classification models and find out one one has the best accuracy on the test set for out data.

Machine Learning Classification With Logistic Regression

1.By using the scilearn-kit we import the logistic regression object and use it to predict the landing outcome.

2. By using the logistic regression model we get an accuracy of 83.3% on the test set.

3. Finding the best parameters for Logistic Regression with grid search.



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

```
[22]: 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)
```

```
[22]: ▸ GridSearchCV
      ▸ estimator: LogisticRegression
          ▸ LogisticRegression
```

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_`.

```
[23]: 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

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

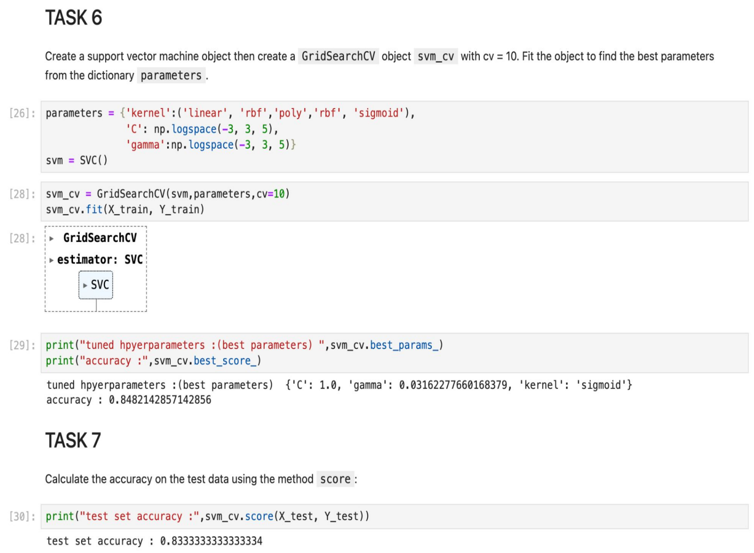
```
[24]: print("test set accuracy :",logreg_cv.score(X_test, Y_test))

      test set accuracy : 0.8333333333333334
```

Machine learning classification with SVM(support vector machine)

1.By using the scilearn-kit we import the SVM object and use it to predict the landing outcome.

2. By using the SVM model we get an accuracy of 83.3% on the test set.

3. Finding the best parameters for SVM with grid search.



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` .

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

```
[28]: svm_cv = GridSearchCV(svm,parameters,cv=10)
      svm_cv.fit(X_train, Y_train)
```

```
[28]: ▸ GridSearchCV
      ▸ estimator: SVC
            ▸ SVC
```

```
[29]: 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
```

TASK 7

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

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

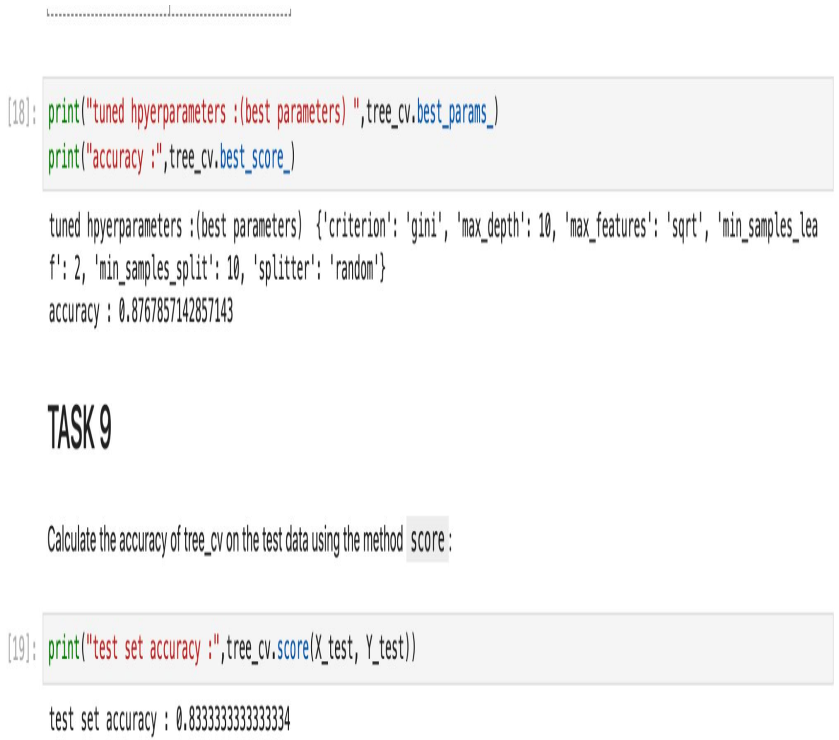
```
test set accuracy : 0.8333333333333334
```

Machine Learning Classification With Decision Tree

1.By using the scilearn-kit we import the Decision Tree object and use it to predict the landing outcome.

2. By using the Decision Tree model we get an accuracy of 83.3% on the test set.

3. Finding the best parameters for Decision Tree with grid search.



```
[18]: 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_lea
f': 2, 'min_samples_split': 10, 'splitter': 'random'}
accuracy : 0.8767857142857143
```

TASK 9

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

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

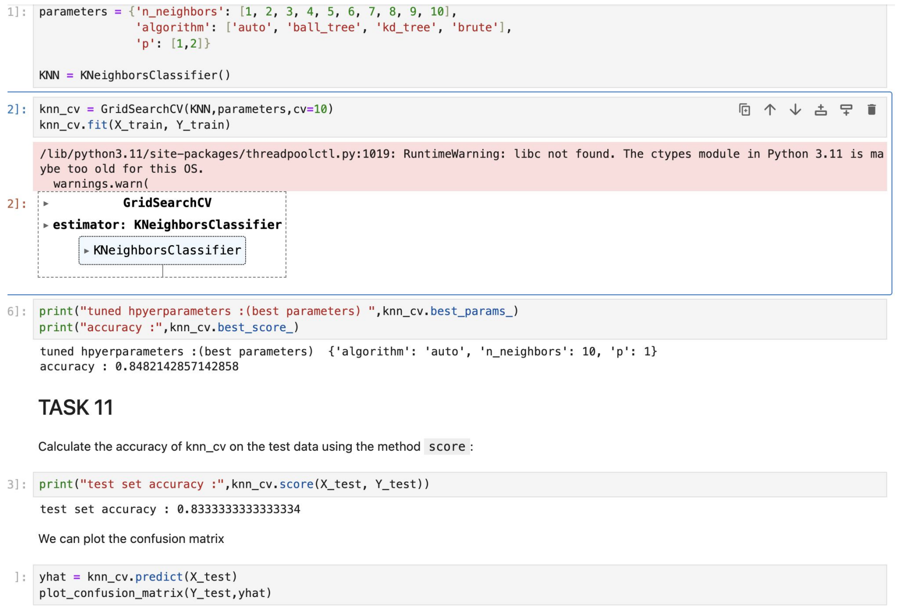
```
test set accuracy : 0.8333333333333334
```

Machine Learning Classification With KNN

1.By using the scilearn-kit we import the KNN object and use it to predict the landing outcome.

2. By using the KNN model we get an accuracy of 83.3% on the test set.

3. Finding the best parameters for KNN with grid search.



```
1]: 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()
```

```
2]: 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 is ma
ybe too old for this OS.
  warnings.warn(
```

```
2]: ▸ GridSearchCV
    ▸ estimator: KNeighborsClassifier
        ▸ KNeighborsClassifier
```

```
6]: 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
```

TASK 11

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

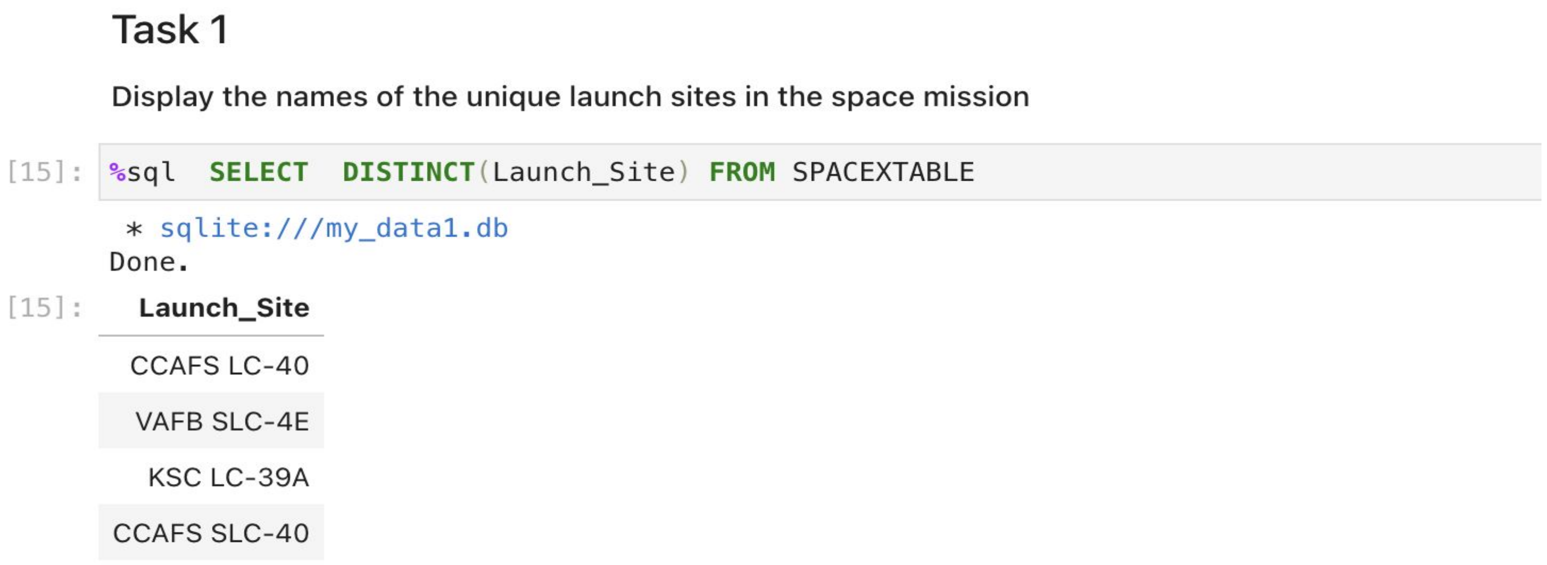
```
3]: print("test set accuracy :",knn_cv.score(X_test, Y_test))
```

```
test set accuracy : 0.8333333333333334
```

We can plot the confusion matrix

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

DISTINCT LAUNCH SITES



Task 1

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

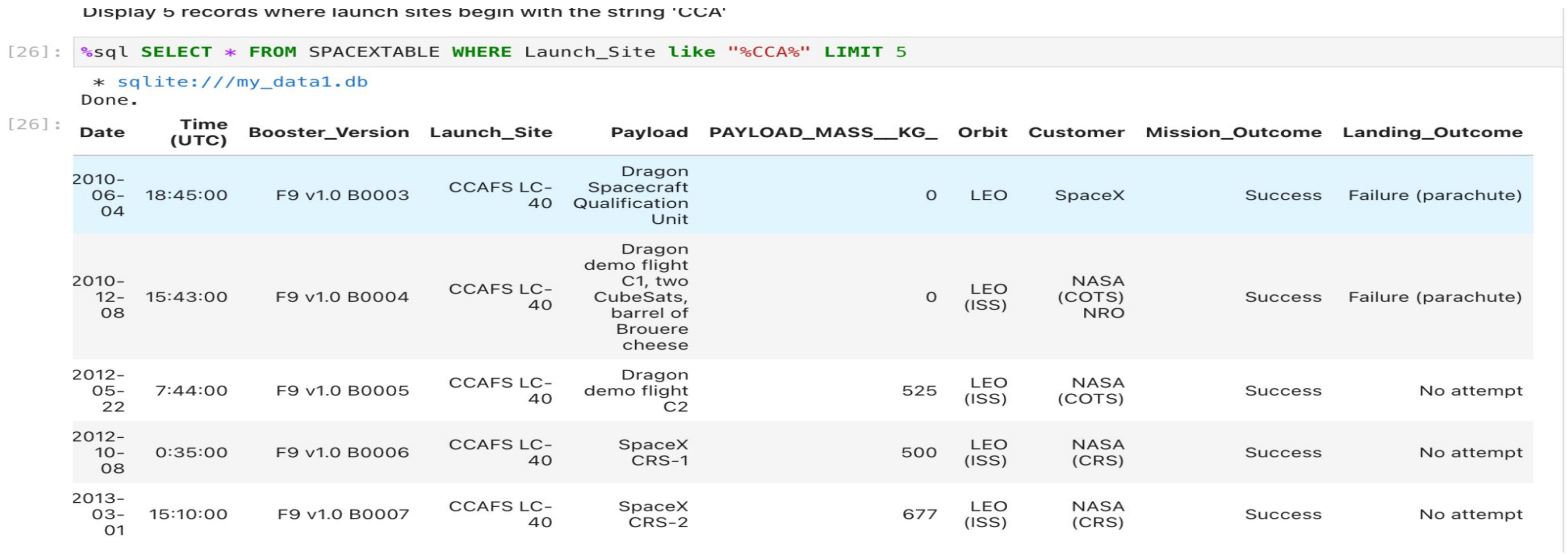
```
[15]: %sql  SELECT  DISTINCT(Launch_Site) FROM SPACEXTABLE
```

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

[15]:

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

CCA LAUNCH SITES



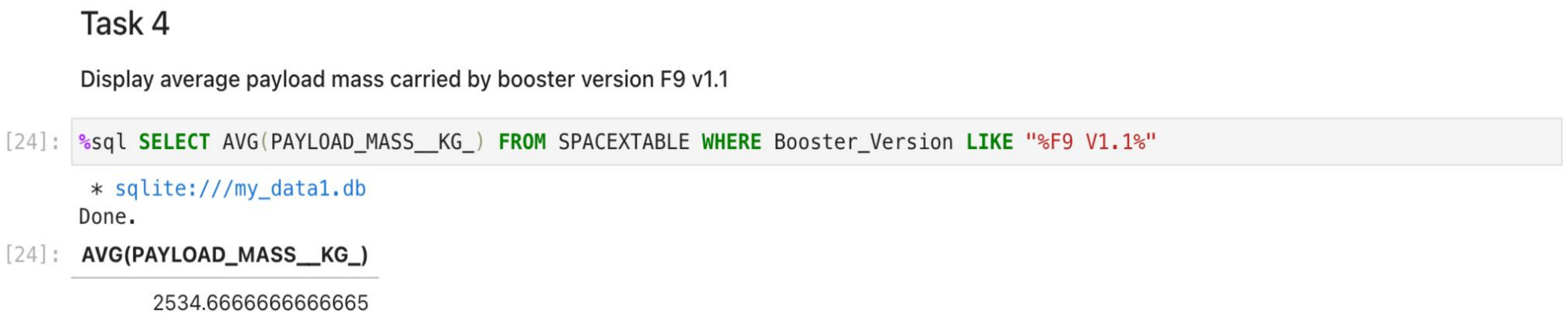
Display 5 records where launch sites begin with the string 'CCA'

```
[26]: %sql SELECT * FROM SPACEXTABLE WHERE Launch_Site like "%CCA%" LIMIT 5

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

[26]:

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

AVG PAYLOAD OF BOOSTER VERSION F9 V1.1



Task 4

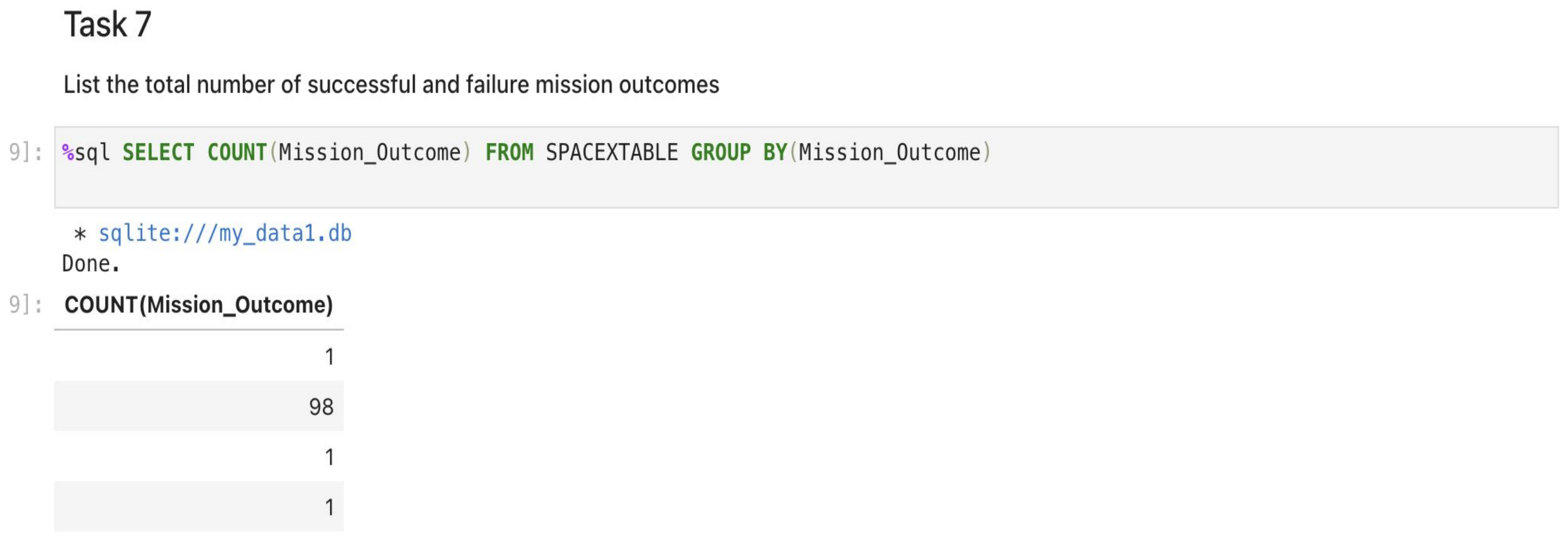
Display average payload mass carried by booster version F9 v1.1

```
[24]: %sql SELECT AVG(PAYLOAD_MASS__KG_) FROM SPACEXTABLE WHERE Booster_Version LIKE "%F9 V1.1%"
```

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

```
[24]: AVG(PAYLOAD_MASS__KG_)
      2534.6666666666665
```

TOTAL NUMBER OF SUCCESSFUL MISSION OUTMCOMES



Task 7

List the total number of successful and failure mission outcomes

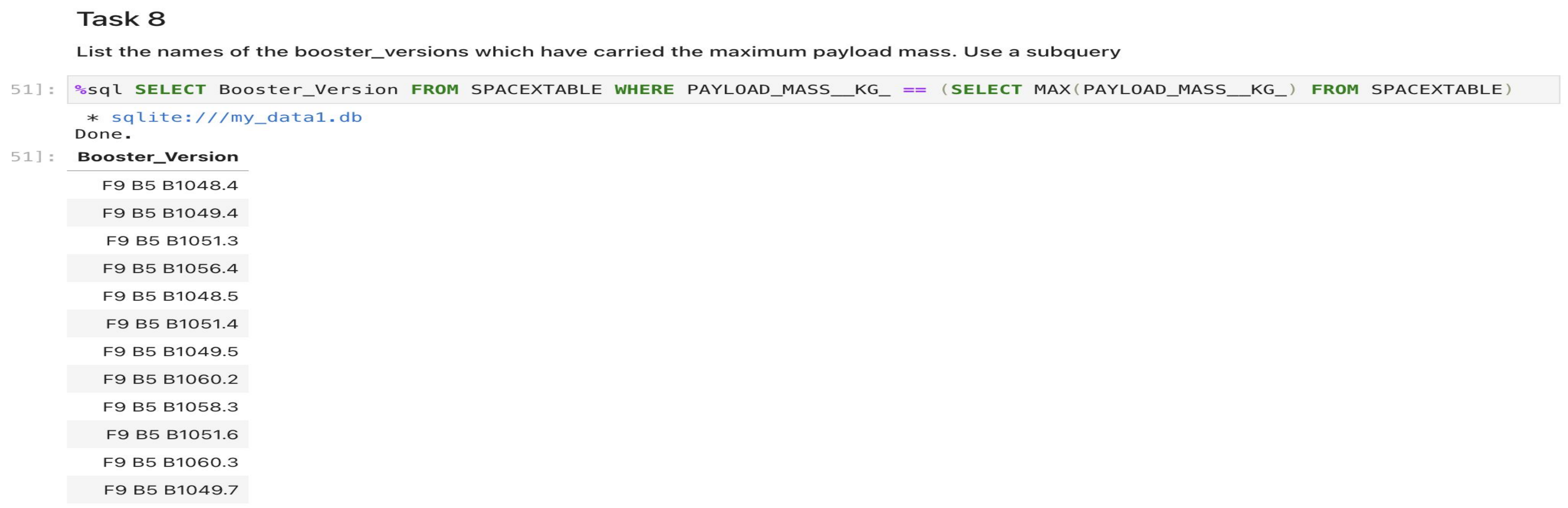
```
9]: %sql SELECT COUNT(Mission_Outcome) FROM SPACEXTABLE GROUP BY(Mission_Outcome)
```

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

9]:

COUNT(Mission_Outcome)
1
98
1
1

BOOST VERSION OF MAX PAYLOAD CRAFT



Task 8

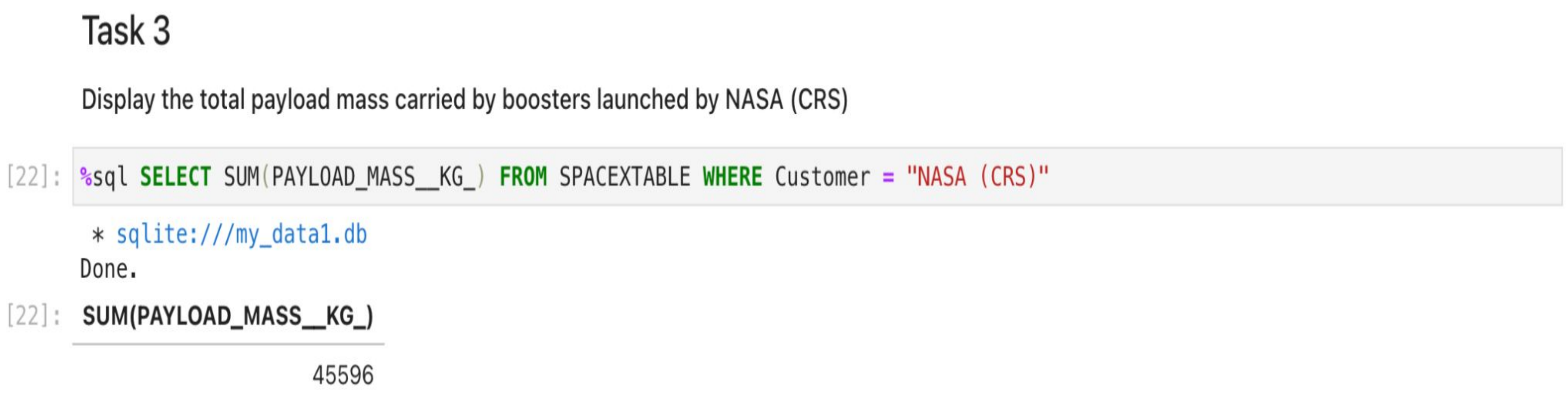
List the names of the booster_versions which have carried the maximum payload mass. Use a subquery

```
51]: %sql SELECT Booster_Version FROM SPACEXTABLE WHERE PAYLOAD_MASS__KG_ == (SELECT MAX(PAYLOAD_MASS__KG_) FROM SPACEXTABLE)

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

51]:

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

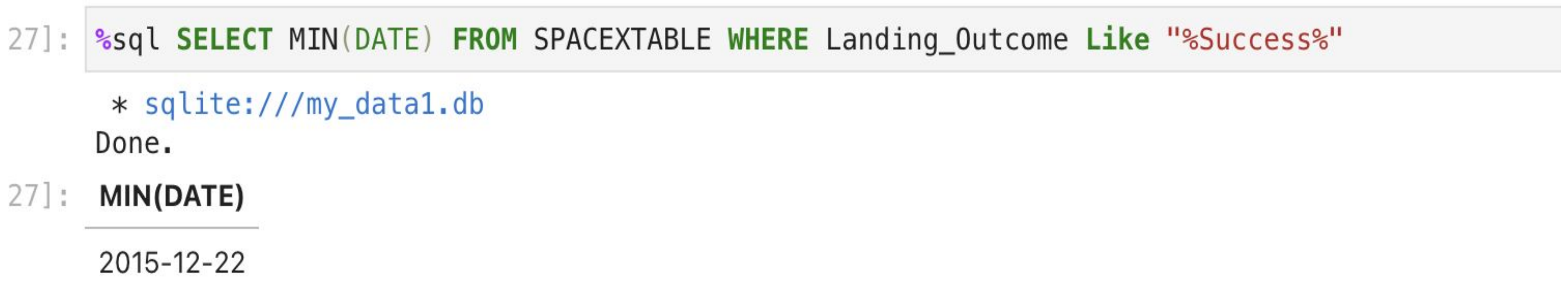
TOTAL PAYLOAD OF NASA BOOSTER LAUNCH


Task 3

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

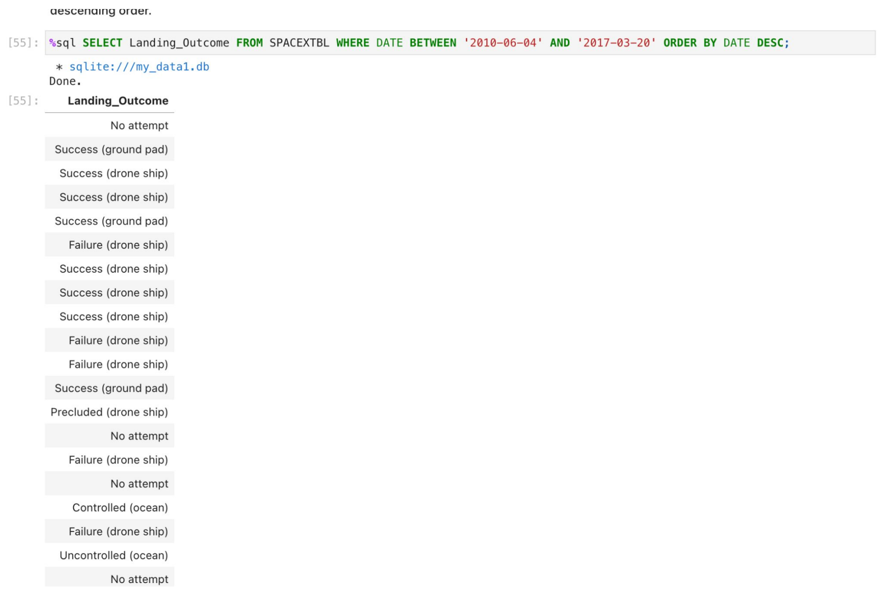
```
[22]: %sql SELECT SUM(PAYLOAD_MASS__KG_) FROM SPACEXTABLE WHERE Customer = "NASA (CRS)"

 * sqlite:///my_data1.db
Done.

[22]: SUM(PAYLOAD_MASS__KG_)
                     45596
```

DATE OF FIRST SUCCESSFUL LAUNCH



```
27]: %sql SELECT MIN(DATE) FROM SPACEXTABLE WHERE Landing_Outcome Like "%Success%"

 * sqlite:///my_data1.db
Done.

27]: MIN(DATE)
     2015-12-22
```

DESCENDING DATE OF LAUNCHES



descending order.

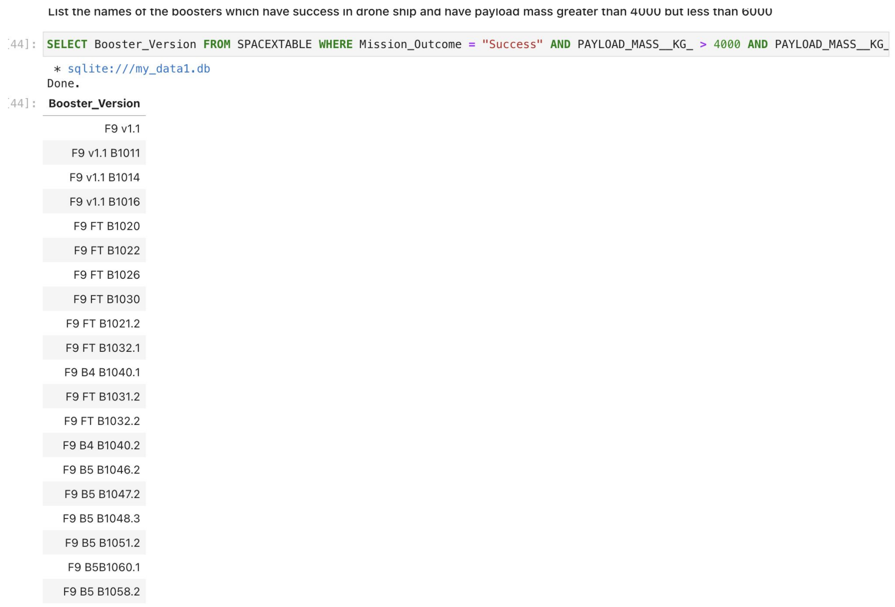
```
[55]: %sql SELECT Landing_Outcome FROM SPACEXTBL WHERE DATE BETWEEN '2010-06-04' AND '2017-03-20' ORDER BY DATE DESC;
```

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

[55]:

Landing_Outcome
No attempt
Success (ground pad)
Success (drone ship)
Success (drone ship)
Success (ground pad)
Failure (drone ship)
Success (drone ship)
Success (drone ship)
Success (drone ship)
Failure (drone ship)
Failure (drone ship)
Success (ground pad)
Precluded (drone ship)
No attempt
Failure (drone ship)
No attempt
Controlled (ocean)
Failure (drone ship)
Uncontrolled (ocean)
No attempt

BOOSTERS THAT HAS SUCCESSFUL LANDING WITH PAYLOAD BETWEEN 4000KG TO 6000KG



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

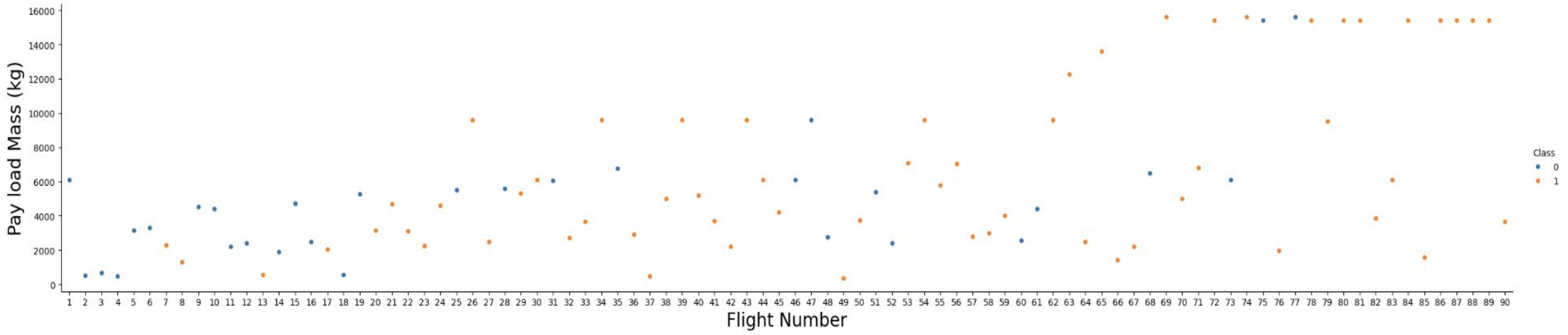
```
[44]: SELECT Booster_Version FROM SPACEXTABLE WHERE Mission_Outcome = "Success" AND PAYLOAD_MASS__KG_ > 4000 AND PAYLOAD_MASS__KG_
```

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

[44]:

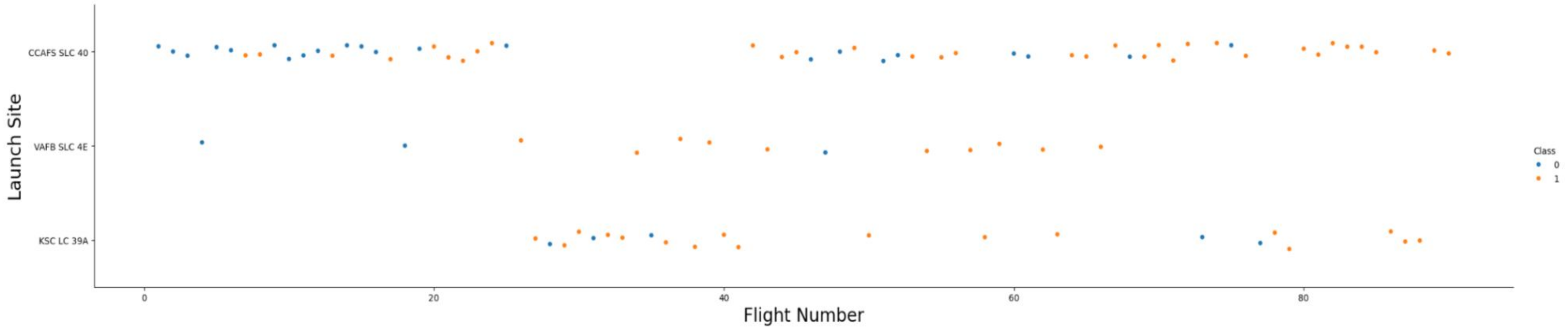
Booster_Version
F9 v1.1
F9 v1.1 B1011
F9 v1.1 B1014
F9 v1.1 B1016
F9 FT B1020
F9 FT B1022
F9 FT B1026
F9 FT B1030
F9 FT B1021.2
F9 FT B1032.1
F9 B4 B1040.1
F9 FT B1031.2
F9 FT B1032.2
F9 B4 B1040.2
F9 B5 B1046.2
F9 B5 B1047.2
F9 B5 B1048.3
F9 B5 B1051.2
F9 B5B1060.1
F9 B5 B1058.2

FLIGHT NUMBER VS PAYLOAD MASS



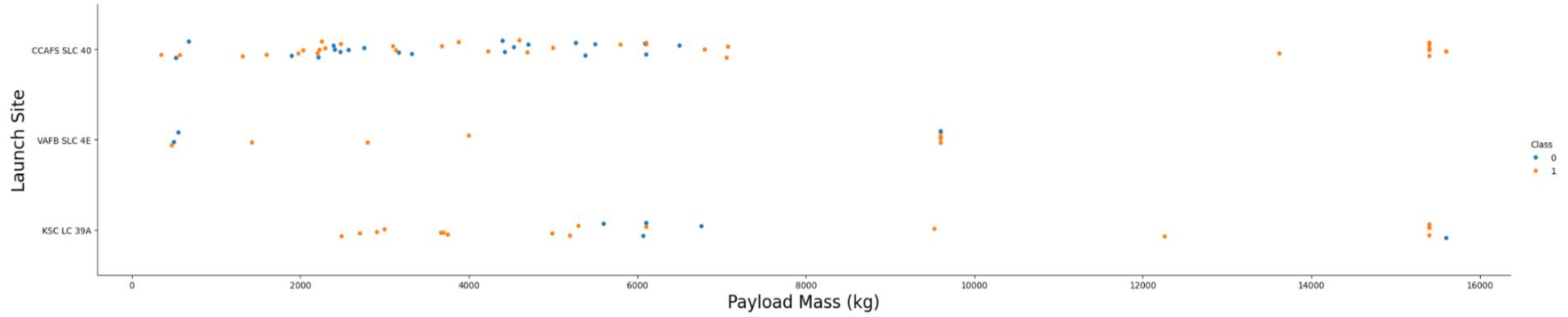
Launch Site VS Flight Number

1.With the increase in flight number there is an increase in successful launches



Launch Site Vs PayLoad Mass

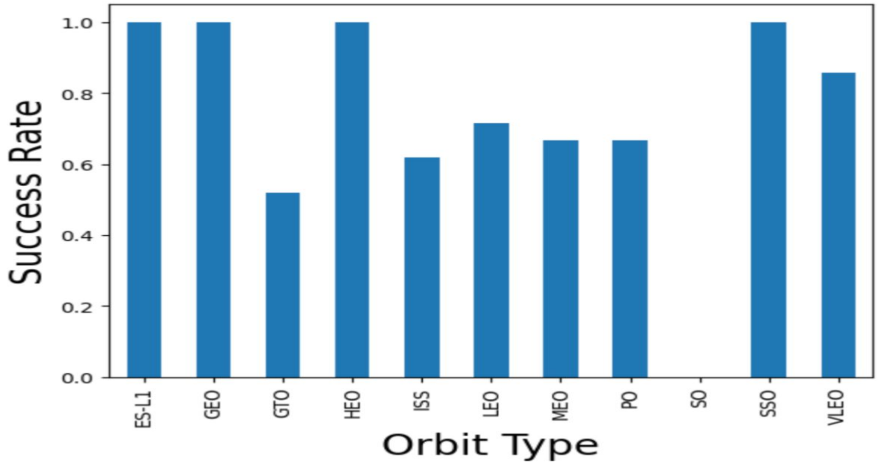
```
[6]: ### TASK 2: Visualize the relationship between Payload and Launch Site
     sns.catplot(y="LaunchSite", x="PayloadMass", hue="Class", data=df, aspect = 5)
     plt.xlabel("Payload Mass (kg)",fontsize=20)
     plt.ylabel("Launch Site",fontsize=20)
     plt.show()
```



Orbit Type Vs Success Rate

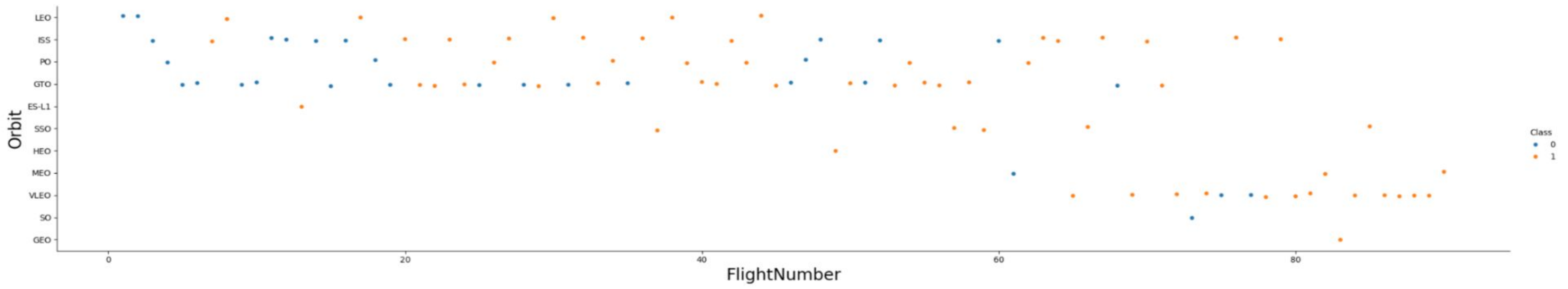
1. ES-L1,GEO,HEO,SSO show the best success rates

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



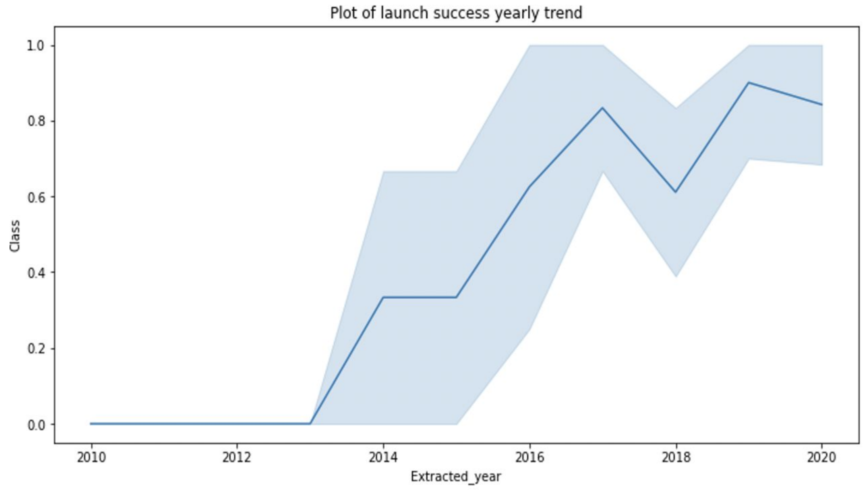
Flight Number Vs Orbit

```
[12]: ### TASK  4: Visualize the relationship between FlightNumber and Orbit type
      sns.catplot(y="Orbit", x="FlightNumber", hue="Class", data=df, aspect = 5)
      plt.xlabel("FlightNumber",fontsize=20)
      plt.ylabel("Orbit",fontsize=20)
      plt.show()
```



Class Vs Year

- Launch success rate started to increase in 2013 till 2020.



Conclusion

We can conclude that:

•With the increase in flight number there is an increase in successful launches

•Launch success rate started to increase in 2013 till 2020.

•Orbits ES-L1, GEO, HEO, SSO, VLEO had the most success rate.

•KSC LC-39A had the most successful launches of any sites.

•The Decision tree classifier is the best machine learning algorithm for this task because of its better train accuracy