IBM Developer
SKILLS NETWORK

Winning Space Race with Data Science

Rohit Kumar

Dec. 31, 2023

Outline

- Executive Summary
- Introduction
- Methodology
- Results
- Conclusion
- Appendix

Executive Summary

- Data Collection using API
- Data Collection using Web Scraping
- Data Wrangling
- Exploratory Data Analysis using SQL
- Exploratory Data Analysis using Data Visualization
- Machine Learning
- Results from Data Analysis
- Results from Machine Learning

Introduction

The purpose of this project is to predict whether the rocket will land successfully or not using machine learning. If we can determine it, we can predict the cost of launch of Falcon 9 rockets of Space X.

Problems that will be answered are:

- Factors that affect landing of a rocket
- What are the needs for a successful landing

Section 1

Methodology

Methodology

Executive Summary

- Data collection methodology:
  - Space X API and web scraping from Wikipedia
- Perform data wrangling
  - Filtering data, handling null values
- Perform exploratory data analysis (EDA) using visualization and SQL
- Perform predictive analysis using classification models
  - How to build, tune, evaluate classification models

Data Collection

- Data was collected by using many different techniques.
- Firstly; I used a get request to the SpaceX API.
- I also performed web scraping from Wikipedia for the Falcon 9 launch records.

Data Collection – SpaceX API

- We used get request to the SpaceX API to collect the data.
- Add the GitHub URL of the completed SpaceX API calls notebook (must include completed code cell and outcome cell), as an external reference and peer-review purpose

In [3]:
```python
# Takes the dataset and uses the launchpad column to call the API and append the data to the list
def getLaunchSite(data):
    for x in data['launchpad']:
       if x:
         response = requests.get("https://api.spacexdata.com/v4/launchpads/"+str(x)).json()
         Longitude.append(response['longitude'])
         Latitude.append(response['latitude'])
         LaunchSite.append(response['name'])
```

From the `payload` we would like to learn the mass of the payload and the orbit that it is going to.

In [4]:
```python
# Takes the dataset and uses the payloads column to call the API and append the data to the lists
def getPayloadData(data):
    for load in data['payloads']:
       if load:
        response = requests.get("https://api.spacexdata.com/v4/payloads/"+load).json()
        PayloadMass.append(response['mass_kg'])
        Orbit.append(response['orbit'])
```

From `cores` we would like to learn the outcome of the landing, the type of the landing, number of flights with that core, whether gridfins were used, wheter the core is reused, wheter legs were used, the landing pad used, the block of the core which is a number used to seperate version of cores, the number of times this specific core has been reused, and the serial of the core.

In [5]:
```python
# Takes the dataset and uses the cores column to call the API and append the data to the lists
def getCoreData(data):
    for core in data['cores']:
            if core['core'] != None:
                response = requests.get("https://api.spacexdata.com/v4/cores/"+core['core']).json()
                Block.append(response['block'])
                ReusedCount.append(response['reuse_count'])
                Serial.append(response['serial'])
            else:
                Block.append(None)
                ReusedCount.append(None)
                Serial.append(None)
            Outcome.append(str(core['landing_success'])+' '+str(core['landing_type']))
            Flights.append(core['flight'])
            GridFins.append(core['gridfins'])
            Reused.append(core['reused'])
            Legs.append(core['legs'])
            LandingPad.append(core['landpad'])
```

Now let's start requesting rocket launch data from SpaceX API with the following URL:

In [6]:
```python
spacex_url="https://api.spacexdata.com/v4/launches/past"
```

In [7]:
```python
response = requests.get(spacex_url)
```

Data Collection - Scraping

- In order to have Falcon 9 data, we applied web scraping.

- Add the GitHub URL of the completed web scraping notebook, as an external reference and peer-review purpose

In [5]:
```python
# use requests.get() method with the provided static_url
htmlData = requests.get(static_url)
# assign the response to a object
htmlData.status_code
```

Out[5]: 200

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

In [7]:
```python
# Use BeautifulSoup() to create a BeautifulSoup object from a response text content
soup = BeautifulSoup(htmlData.text)
```

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

In [8]:
```python
# Use soup.title attribute
soup.title
```

Out[8]: <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

In [9]:
```python
# 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.

In [10]:
```python
# Let's print the third table and check its content
first_launch_table = html_tables[2]
print(first_launch_table)
```

Data Wrangling

- We applied exploratory data analysis and we calculated the number of launches at each site.

Add the GitHub URL of your completed data wrangling related notebooks, as an external reference and peer-review purpose

EDA with Data Visualization

- We performed exploratory data analysis by using many different plots in order to visualize the relationship between data.
- Add the GitHub URL of your completed EDA with data visualization notebook, as an external reference and peer-review purpose

EDA with SQL

- We applied many different SQL commands in order to understand the data better. We used SQL for:
  - 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
  - The total number of successful and failure mission outcomes
  - The failed landing outcomes in drone ship, their booster version and launch site names.

- Add the GitHub URL of your completed EDA with SQL notebook, as an external reference and peer-review purpose

Predictive Analysis (Classification)

- We transformed data using numpy and pandas and we split data into train test sets.
- We tried many different classification models and evaluated their accuracies
- And we found the best model for our case.
- Add the GitHub URL of your completed predictive analysis lab, as an external reference and peer-review purpose

Results

- Exploratory data analysis results
- Interactive analytics demo in screenshots
- Predictive analysis results

Section 2

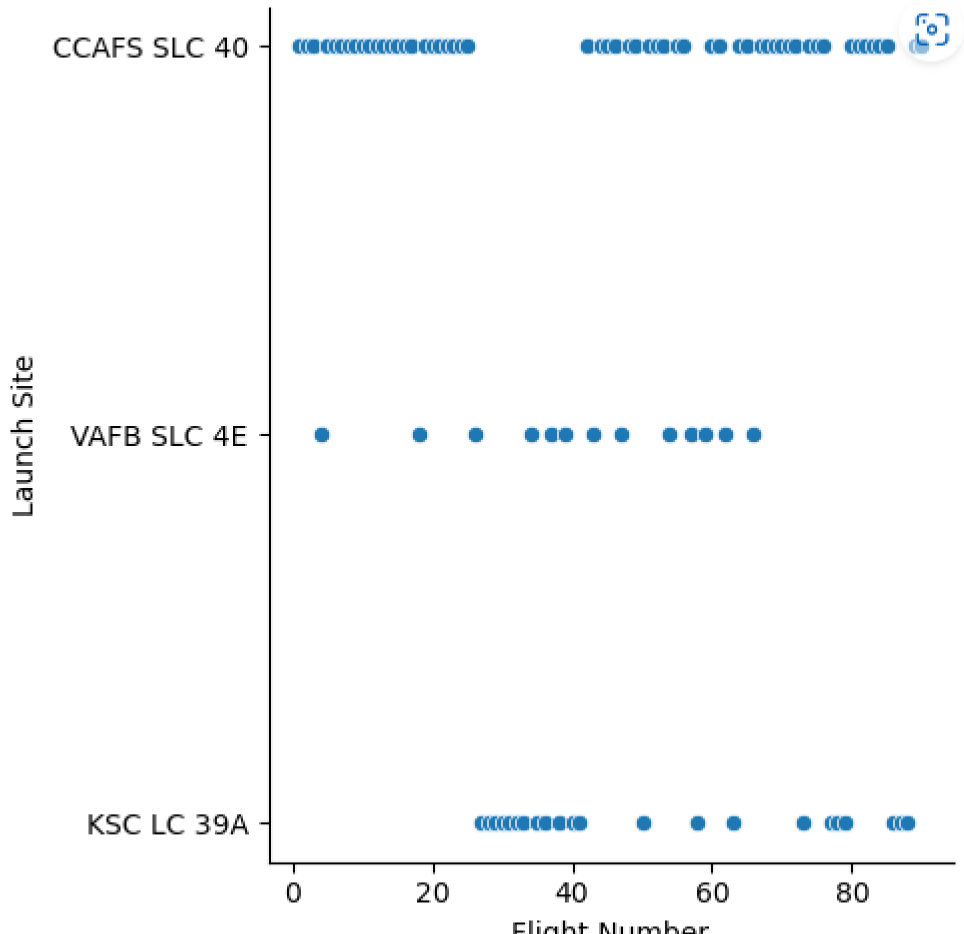
Insights drawn from EDA

Flight Number vs. Launch Site

When the flight amount increase,

Success rate increases as well.

[5]:
```
### TASK 1: Visualize the relationship between Flight Number and Launch Site
sns.relplot(y="LaunchSite", x="FlightNumber", data=df)
plt.xlabel("Flight Number")
plt.ylabel("Launch Site")
plt.show()
```

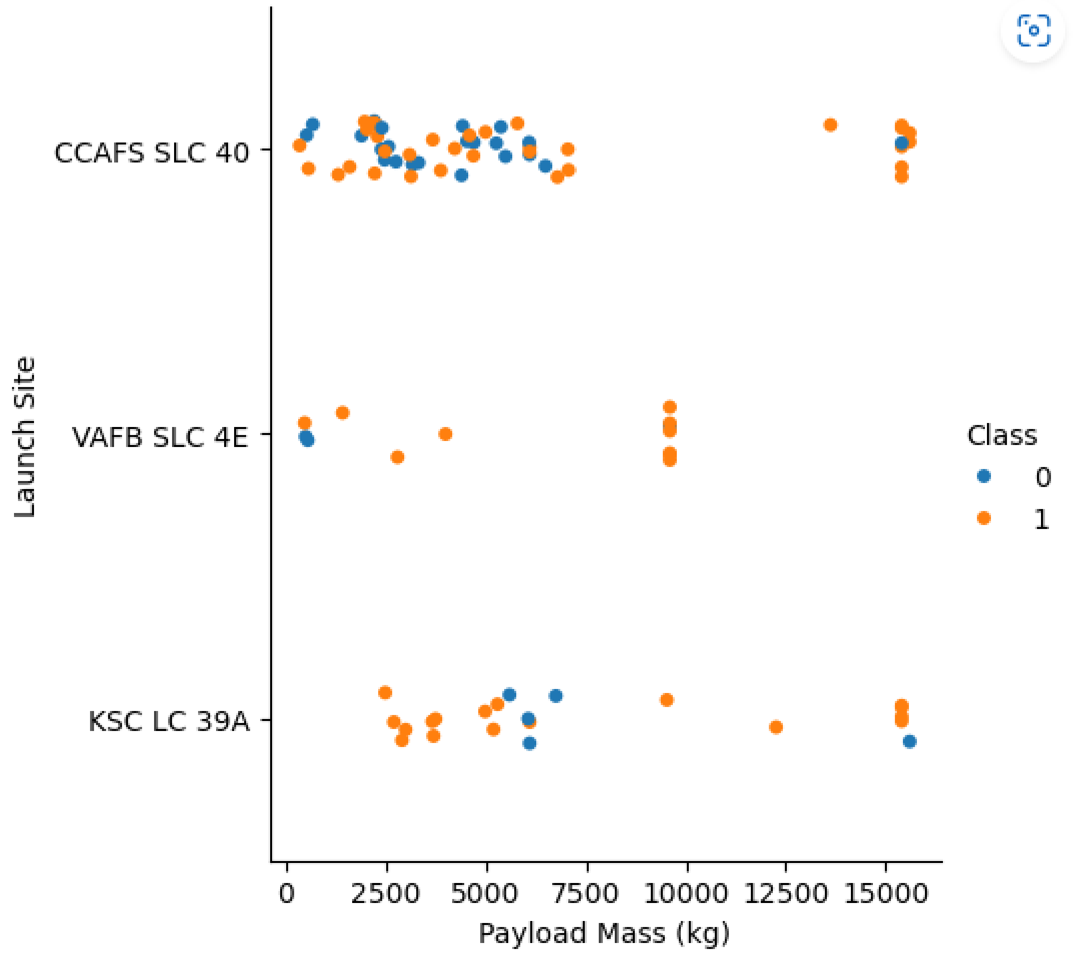


Payload vs. Launch Site

When the payload mass increase,

Success rate increases as well.

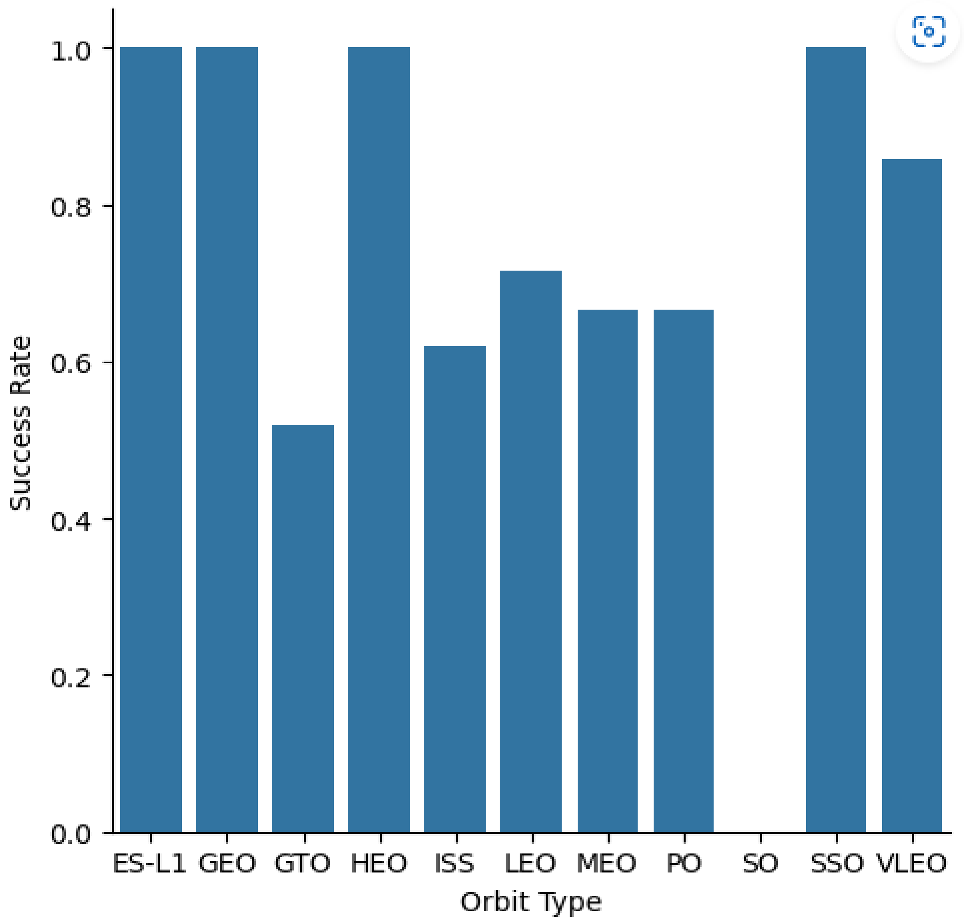
[8]:
```
# Plot a scatter point chart with x axis to be Pay Load Mass (kg) and y axis to be the L
sns.catplot(y="LaunchSite", x="PayloadMass", hue="Class", data=df)
plt.xlabel("Payload Mass (kg)")
plt.ylabel("Launch Site")
plt.show()
```



Success Rate vs. Orbit Type

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

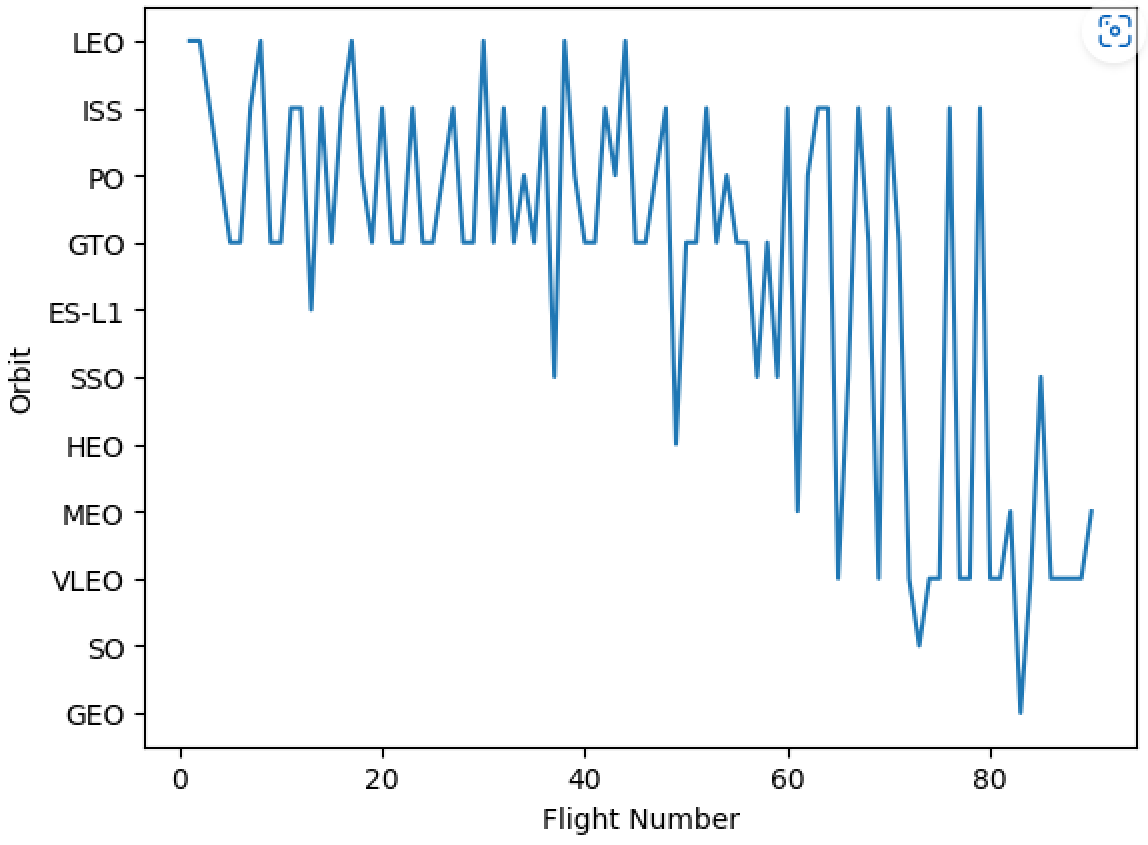
```
[9]: ### TASK  3: Visualize the relationship between success rate of each orbit type
sns.catplot(x= 'Orbit', y = 'Class', data = df.groupby('Orbit')['Class'].mean().reset_index(), kind = 'bar')
plt.xlabel('Orbit Type')
plt.ylabel('Success Rate')
plt.show()
```



Flight Number vs. Orbit Type

In the LEO orbit, success is related to the number of flights whereas in the GTO orbit, there is no relationship between flight number and the orbit.

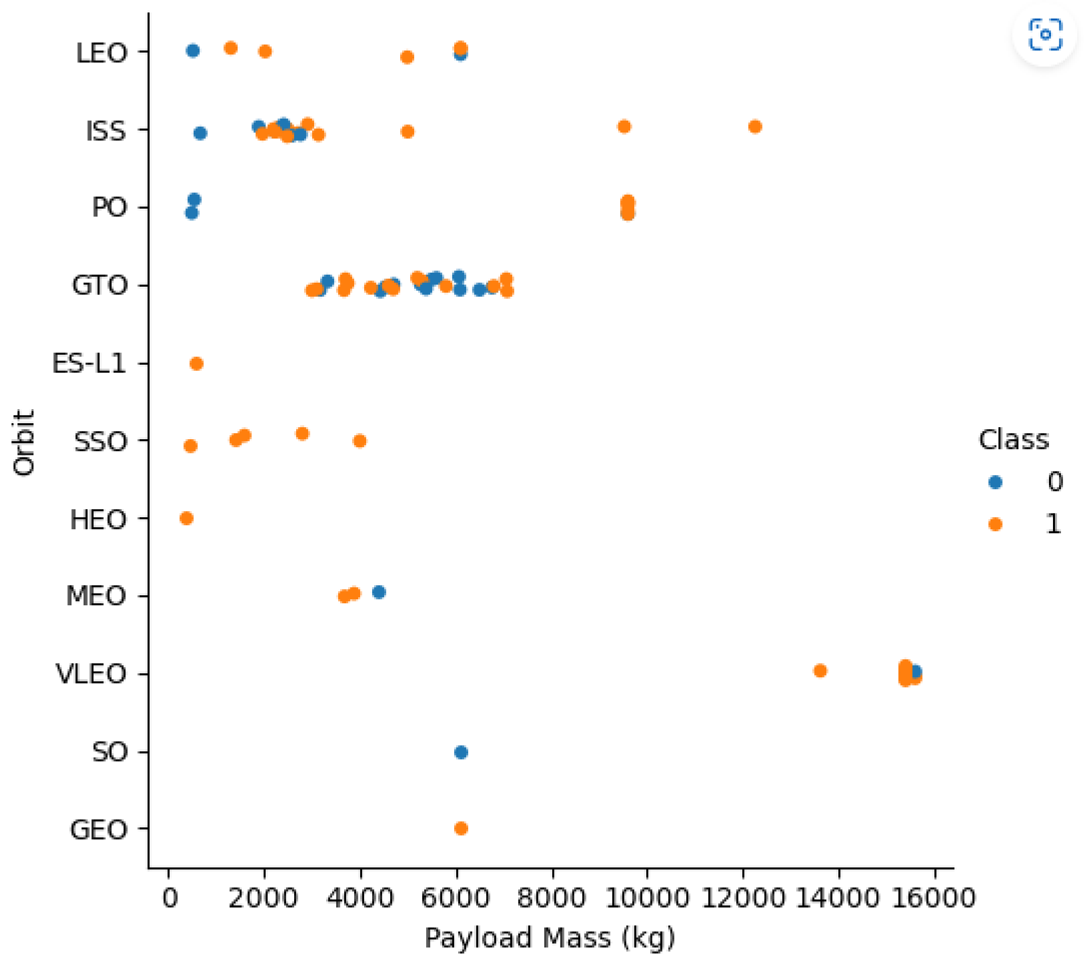
[10]:
```
### TASK  4: Visualize the relationship between FlightNumber and Orbit type
sns.lineplot(y="Orbit", x="FlightNumber",data=df)
plt.xlabel("Flight Number")
plt.ylabel("Orbit")
plt.show()
```



Payload vs. Orbit Type

The successful landing are more for PO, LEO and ISS orbits.

[12]:
```
### TASK  5: Visualize the relationship between Payload and Orbit type
sns.catplot(y="Orbit", x="PayloadMass", hue="Class", data=df)
plt.xlabel("Payload Mass (kg)")
plt.ylabel("Orbit")
plt.show()
```

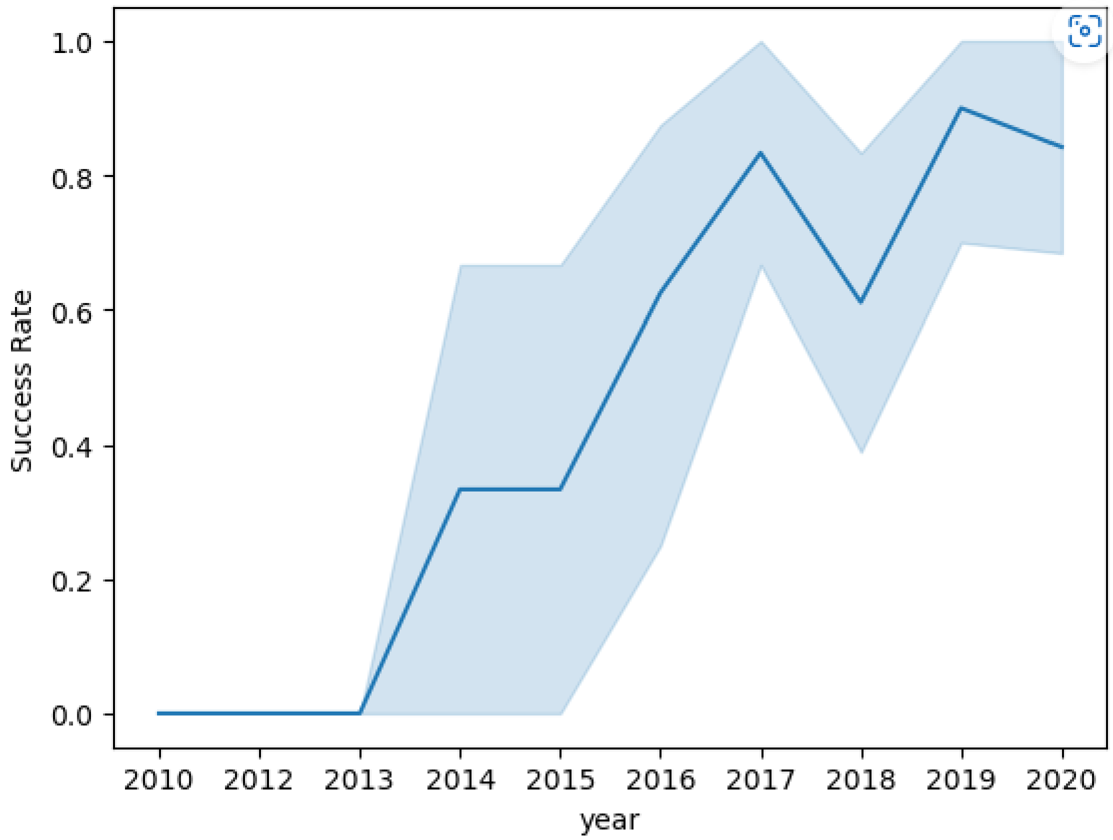


Launch Success Yearly Trend

- Success rate increased from 2013 until 2019.

[15]:

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



All Launch Site Names

We used DISTINCT key Word to show unique names

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

In [10]:
```
task_1 = '''
        SELECT DISTINCT LaunchSite
        FROM SpaceX
'''
create_pandas_df(task_1, database=conn)
```

Out[10]:

	launchsite
0	KSC LC-39A
1	CCAFS LC-40
2	CCAFS SLC-40
3	VAFB SLC-4E

Launch Site Names Begin with 'CCA'

- The following query performed for the task.

```
[10]: %sql SELECT * \
          FROM SPACEXTBL \
          WHERE LAUNCH_SITE LIKE'CCA%' LIMIT 5;

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

[10]:

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

Total Payload Mass

The total payload carried by boosters from NASA is 45596

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

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

[15]:

SUM(PAYLOAD_MASS__KG_)
45596

Average Payload Mass by F9 v1.1

- The average payload mass carried by booster version F9 v1.1 is 2928.4

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

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

[16]:

AVG(PAYLOAD_MASS__KG_)
2928.4

First Successful Ground Landing Date

- The date of the first successful landing is 2015-12-22

```
In [14]:
task_5 = '''
        SELECT MIN(Date) AS FirstSuccessfull_landing_date
        FROM SpaceX
        WHERE LandingOutcome LIKE 'Success (ground pad)'
        '''
create_pandas_df(task_5, database=conn)
```

Out[14]:

	firstsuccessfull_landing_date
0	2015-12-22

Successful Drone Ship Landing with Payload between 4000 and 6000

- The result of the current task is provided here:

In [15]:
```
task_6 = '''
        SELECT BoosterVersion
        FROM SpaceX
        WHERE LandingOutcome = 'Success (drone ship)'
            AND PayloadMassKG > 4000
            AND PayloadMassKG < 6000
        '''
create_pandas_df(task_6, database=conn)
```

Out[15]:

	boosterversion
0	F9 FT B1022
1	F9 FT B1026
2	F9 FT B1021.2
3	F9 FT B1031.2

Total Number of Successful and Failure Mission Outcomes

[19]:
```
%sql SELECT MISSION_OUTCOME, COUNT(*) as total_number \
FROM SPACEXTBL \
GROUP BY MISSION_OUTCOME;
```

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

[19]:

Mission_Outcome	total_number
Failure (in flight)	1
Success	98
Success	1
Success (payload status unclear)	1

Boosters Carried Maximum Payload

Using WHERE and MAX(), we calculated the query as following:

[20]:

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

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

[20]:

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

2015 Launch Records

- We used following query to list launch records of 2015

List the failed landing_outcomes in drone ship, their booster versions, and launch site names for in year 2015

In [18]:
```
task_9 = '''
        SELECT BoosterVersion, LaunchSite, LandingOutcome
        FROM SpaceX
        WHERE LandingOutcome LIKE 'Failure (drone ship)'
            AND Date BETWEEN '2015-01-01' AND '2015-12-31'
        '''
create_pandas_df(task_9, database=conn)
```

Out[18]:

	boosterversion	launchsite	landingoutcome
0	F9 v1.1 B1012	CCAFS LC-40	Failure (drone ship)
1	F9 v1.1 B1015	CCAFS LC-40	Failure (drone ship)

Rank Landing Outcomes Between 2010-06-04 and 2017-03-20

- Ranks of landings between the given years are as following:

Rank the count of landing outcomes (such as Failure (drone ship) or Success (ground pad))

In [19]:

```
task_10 = '''
        SELECT LandingOutcome, COUNT(LandingOutcome)
        FROM SpaceX
        WHERE DATE BETWEEN '2010-06-04' AND '2017-03-20'
        GROUP BY LandingOutcome
        ORDER BY COUNT(LandingOutcome) DESC
        '''
create_pandas_df(task_10, database=conn)
```

Out[19]:

	landingoutcome	count
0	No attempt	10
1	Success (drone ship)	6
2	Failure (drone ship)	5
3	Success (ground pad)	5
4	Controlled (ocean)	3
5	Uncontrolled (ocean)	2
6	Precluded (drone ship)	1
7	Failure (parachute)	1

Section 5

Predictive Analysis (Classification)

Classification Accuracy

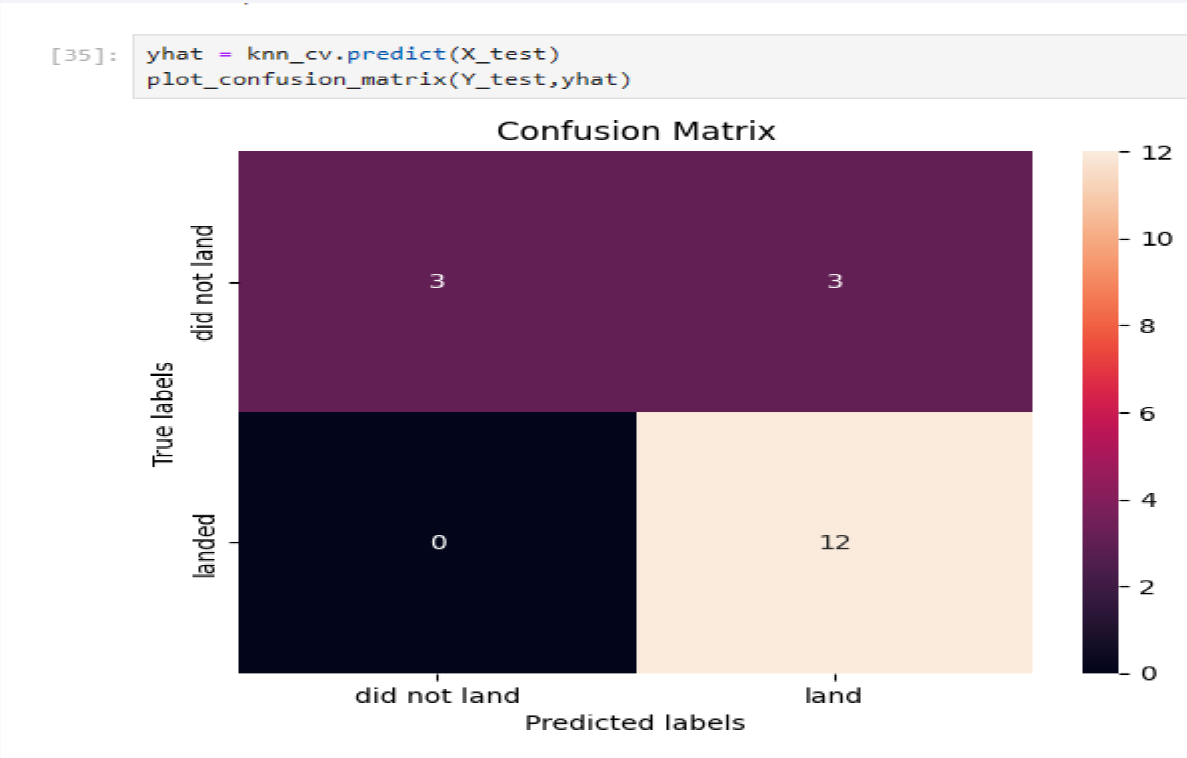
- We used K Nearest Neighbors for our model.

```
[32]: 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)
```

Confusion Matrix

The confusion matrix for the K Nearest Neighbors is like that:



Conclusions

When the flight amount increase, success rate increases as well.

When the payload mass increase, success rate increases as well

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

K Neares Neighbors is the most successful model

Thank you!