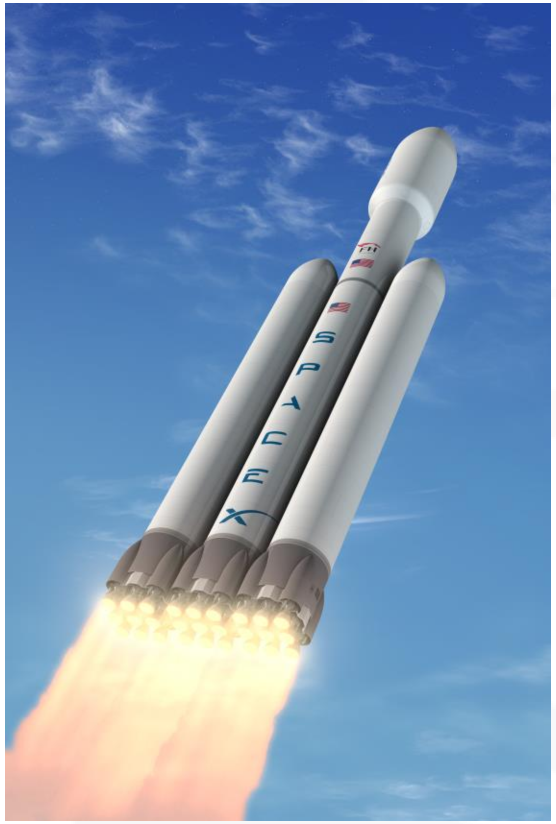
Data Analysis of SpaceX Falcon 9 First Stage Landing

Frederick Tan

03 Oct 2024

OUTLINE



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

EXECUTIVE SUMMARY

- SpaceX Falcon 9 Launch Data obtained from SpaceX API and Web Scraping
- SQL used to clean data
- Python used to visualize data – Folium and Ploty Dash used to create interactive visualizations
- Analysis primarily conducted on Launch Site, Orbit, Payload Mass, Flight Number, and Year to assess how the variables contribute to the success or failure of a launch
- Logistic Regression, Support Vector Machine, Decision Tree, and K Nearest can all be used for predictive classification of future flights with all algorithms obtaining a score of 0.8366 when using the same test/train split on data set

INTRODUCTION

- Project Background
  - SpaceX advertises Falcon 9 rocket launches on its website with a cost of 62 million dollars; other providers cost upwards of 165 million dollars each, much of the savings is because SpaceX can reuse the first stage
  - Analysis completed to determine if the first stage will land

- Problems to Solve
  - Determine the price of each launch. by gathering information about Space X and creating dashboards for your team.
  - Determine if SpaceX will reuse the first stage by training a machine learning model and use of public information to predict if SpaceX will reuse the first stage.

METHODOLOGY

- Data Gathering and Wrangling
  - API and Webscraping utilized to create data set
- Exploratory Analysis
  - SQL to refine dataset and determine initial findings
  - Visualizations
    - Python for initial visualizations
    - Folium and Plotly Dash for interactive visualizations
- Predictive Analysis
  - Logistic Regression, Support Vector Machine, Decision Tree Classifier, and k-Nearest Neighbors to determine best predictive algorithm

Methodology

Data Collection and Wrangling

- Data gathered via SpaceX Data API
  - Only Falcon 9 Booster Version was assessed in this analysis
  - Payload Mass was the only variable with null values
    - Null values replaced with average Payload Mass
- Additional launch data for Falcon 9 boosters scraped from Wikipedia using BeautifulSoup
- Classification variable for launch outcome added (binary variable for first stage successful land or not)

Exploratory Data Analysis

- Determined number of successful and unsuccessful landing attempts for only Falcon 9 Booster rockets and their associated variables
- Explored launch outcome as it correlates to Flight Number, Launch Site, Payload Mass, and Orbit Type (including creating dummy variables)
- Launch success over time trend
- Created plots to explore the variable correlations

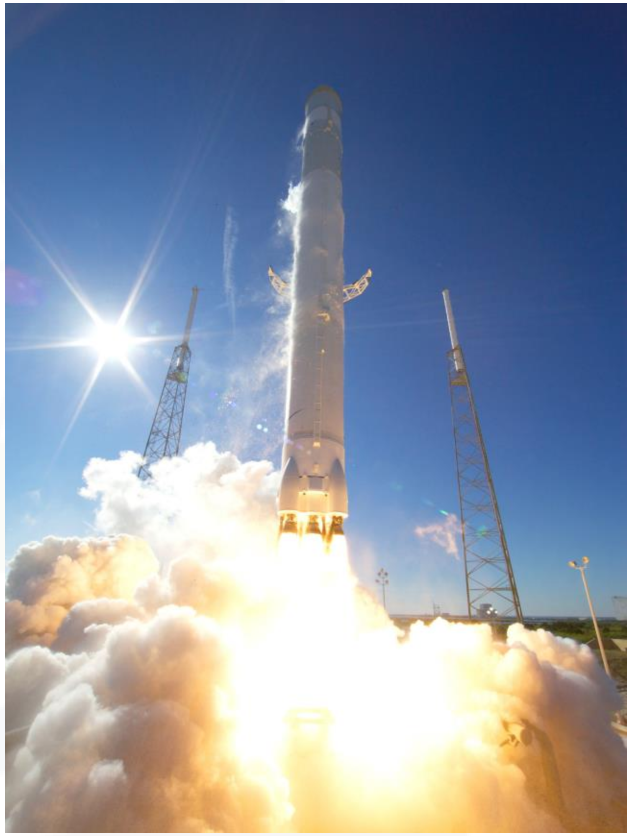
EDA – Interactive Data Visualization

- Created Folium maps to mark launch sites, denote successful/failed launches, marked key geographic proximities to launch site

- Created interactive Ploty dashboard to further explore data visualizations for Launch Site, Payload Mass, and Successful launches

Predictive Data Analysis

- Set Y variable as binary classification of successful (or not) Falcon 9 first stage landing
- Set X variables as all other variables in data set (83 total variables, 90 total instances)
- Created Test and Train split for X and Y, with test size = 0.2
- Performed Logistic Regression, Support Vector Machine, Decision Tree, and K Nearest Neighbors
  - Used Score value to evaluate performance of all four algorithms



RESULTS

Data Collection and Wrangling

- Determined number of flights from each launch site
- Determined number of occurrences of each orbit type for launch
- Determined number of each type of launch type
  - Successful vs failure and landing location
    - Ocean (Ocean)
    - Ground Pad (RTLS)
    - Drone Ship (ASDS)
- Determined total number of successful (1) and failed (0) launches

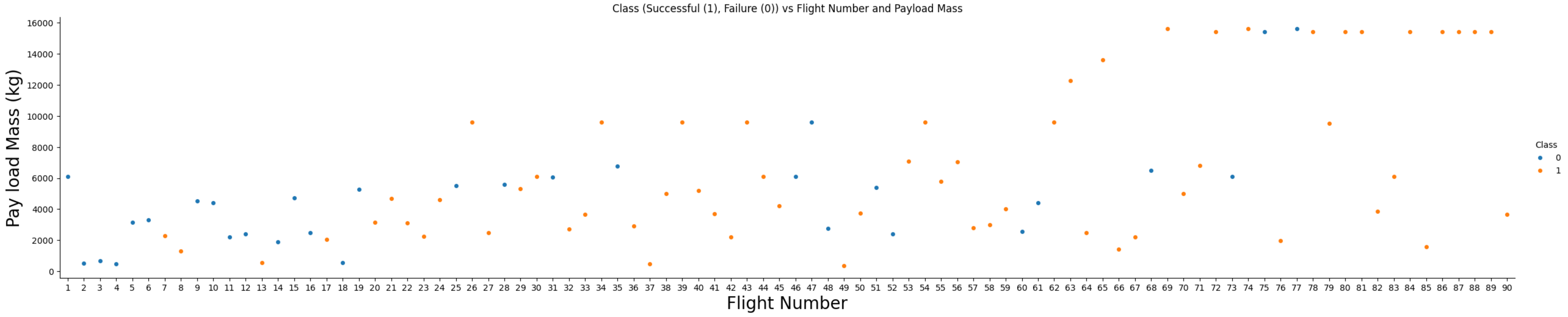
LaunchSite	
CCAFS SLC 40	55
KSC LC 39A	22
VAFB SLC 4E	13

Orbit	
GTO	27
ISS	21
VLEO	14
PO	9
LEO	7
SSO	5
MEO	3
HEO	1
ES-L1	1
SO	1
GEO	1

Outcome	
True ASDS	41
None None	19
True RTLS	14
False ASDS	6
True Ocean	5
False Ocean	2
None ASDS	2
False RTLS	1

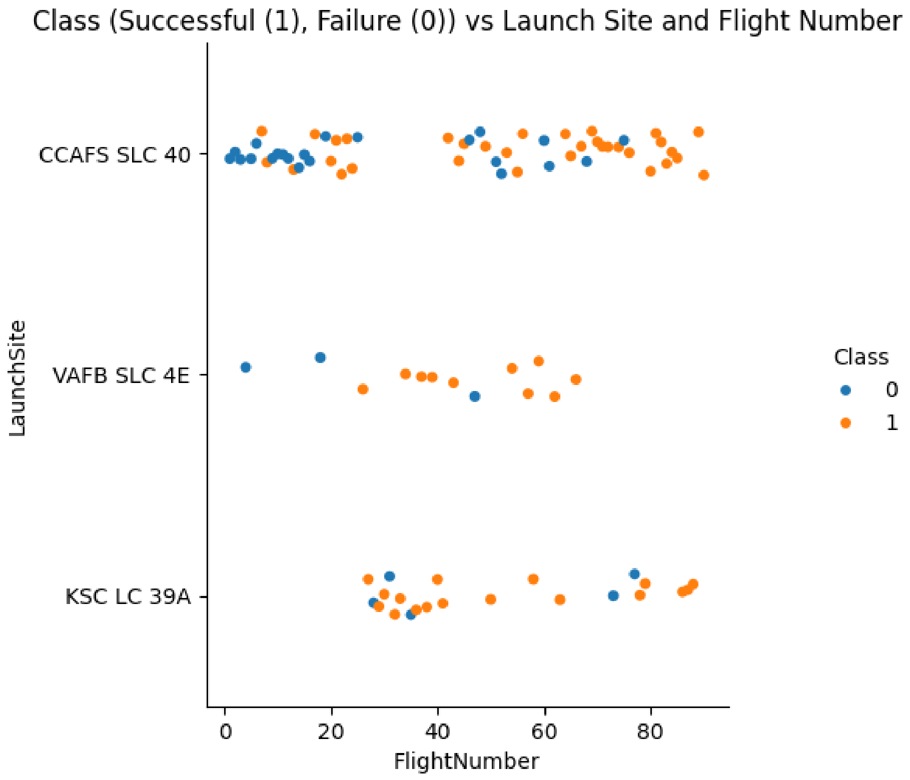
Class	
1	60
0	30

EDA - Visualizations

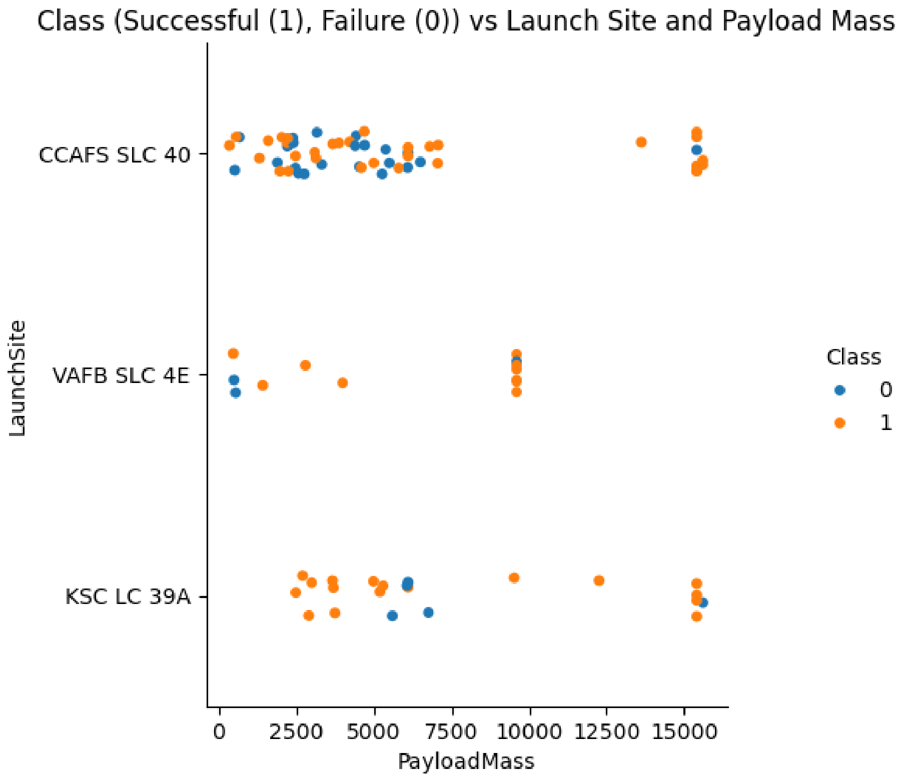


Plot showing whether first stage landing was successful (Class) based on Payload Mass (kg) and Flight Number. No clear relationship established with this visualization

EDA - Visualizations

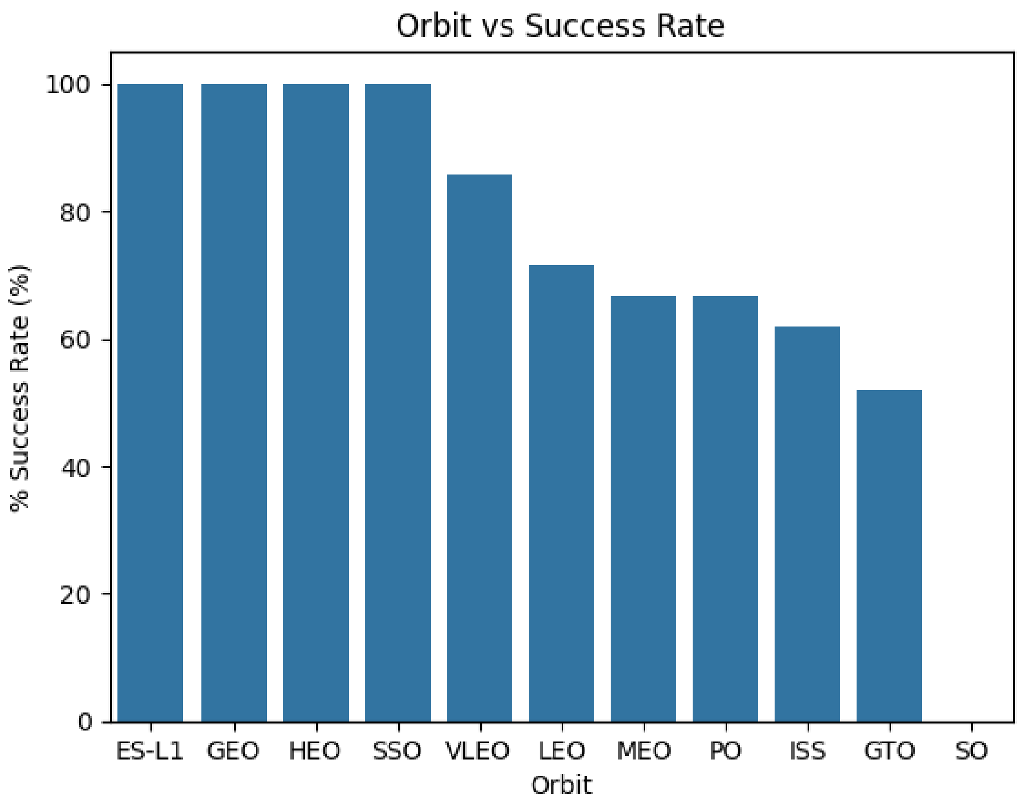


Plot showing whether first stage landing was successful (Class) based on Flight Number and Launch Site. No strong relationships established, but appears to be higher success rate for higher Flight Numbers across all three locations



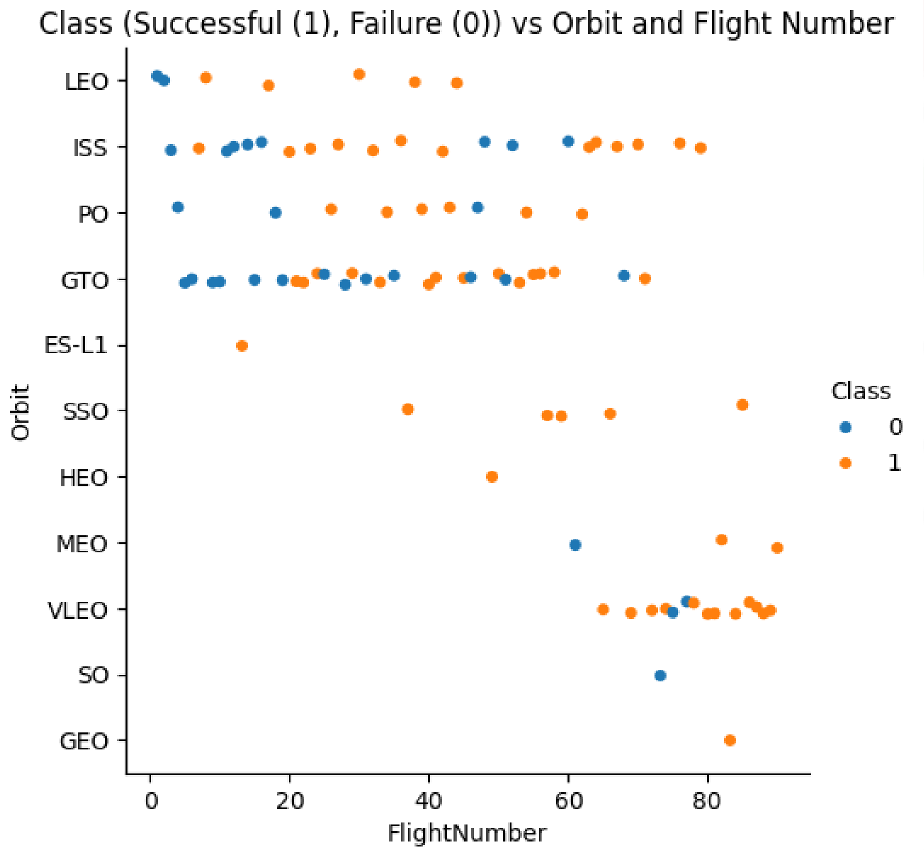
Plot showing whether first stage landing was successful (Class) based on Payload Mass (kg) and Launch Site. No strong relationships established, but appears to be higher success rate for larger Payload Mass across all three locations

EDA - Visualizations

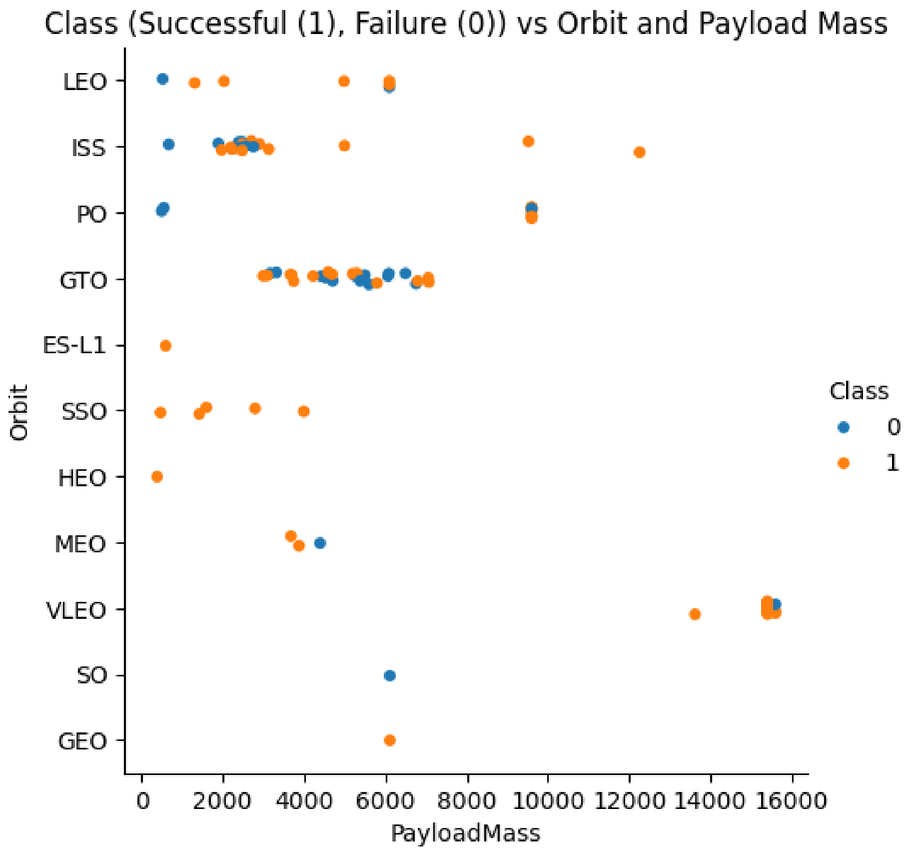


Plot showing Success Rate (%) vs target Orbit. ES-L1, GEO, HEO, and SSO have 100% success rate.

EDA - Visualizations

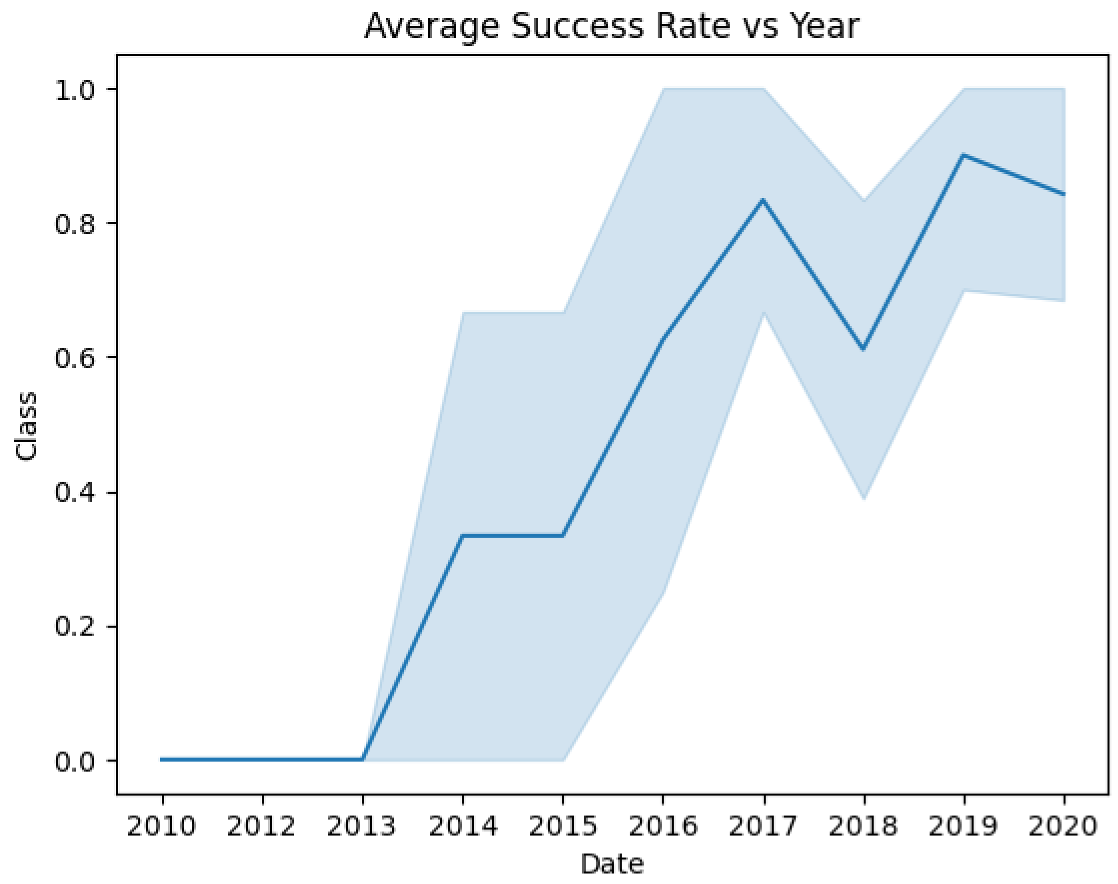


Plot showing whether first stage landing was successful (Class) based on Flight Number and Orbit. ES-L1, GEO, HEO, and SSO have 100% success rate and earlier trend of higher Flight Number relating to improved success rate holds



Plot showing whether first stage landing was successful (Class) based on Payload Mass (kg) and Orbit. ES-L1, GEO, HEO, and SSO have 100% success rate. Earlier trend of higher Payload Mass relating to improved success rate is not as apparent in this figure.

EDA - Visualizations



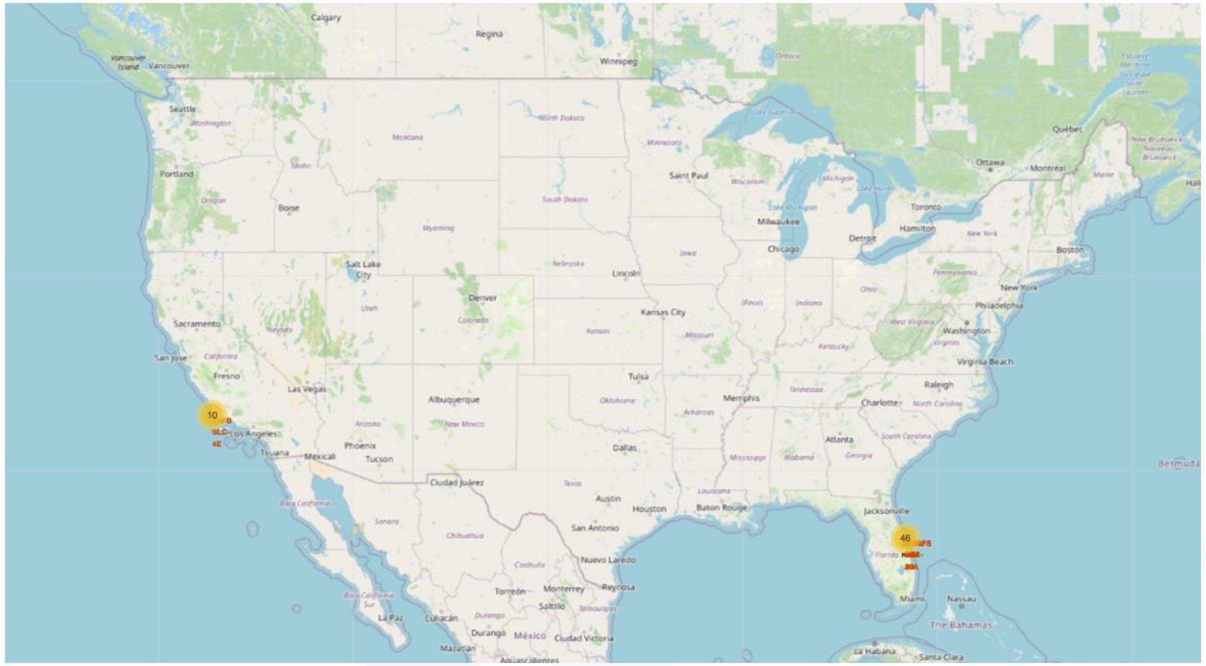
Plot showing average success rate (Class) vs Year. Shows general trend towards higher success rate as years progress

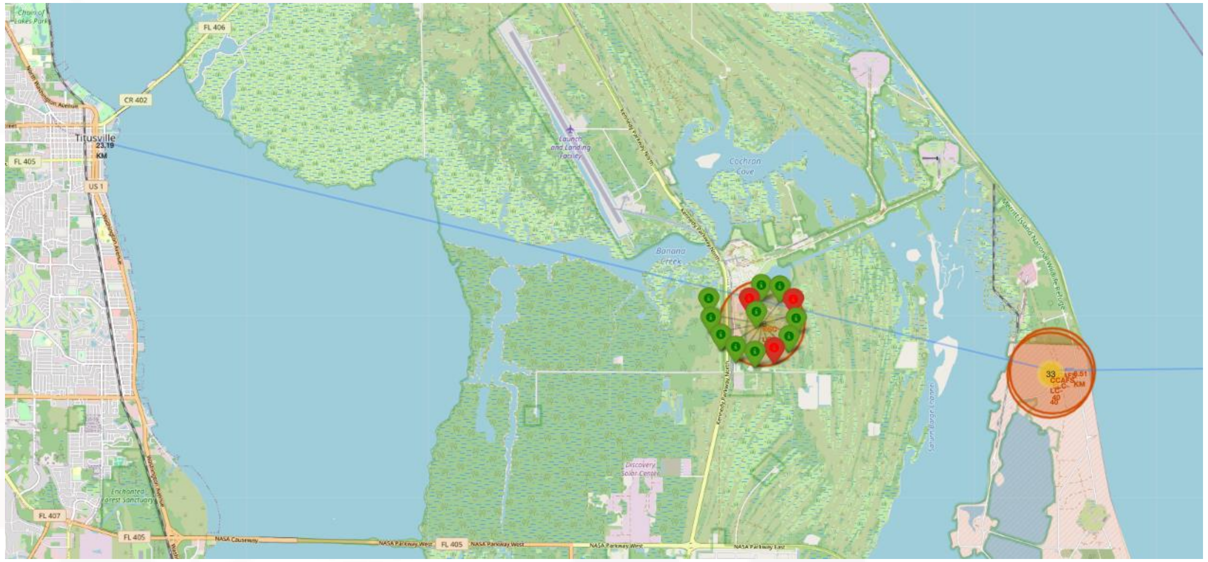
EDA – SQL

- Key Findings
  - Average Payload Mass = 2534.667 kg
  - First Successful Landing = 2015-12-22
  - Counts of 8 different landing outcome scenarios

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

EDA – Folium Interactive Data Visualization



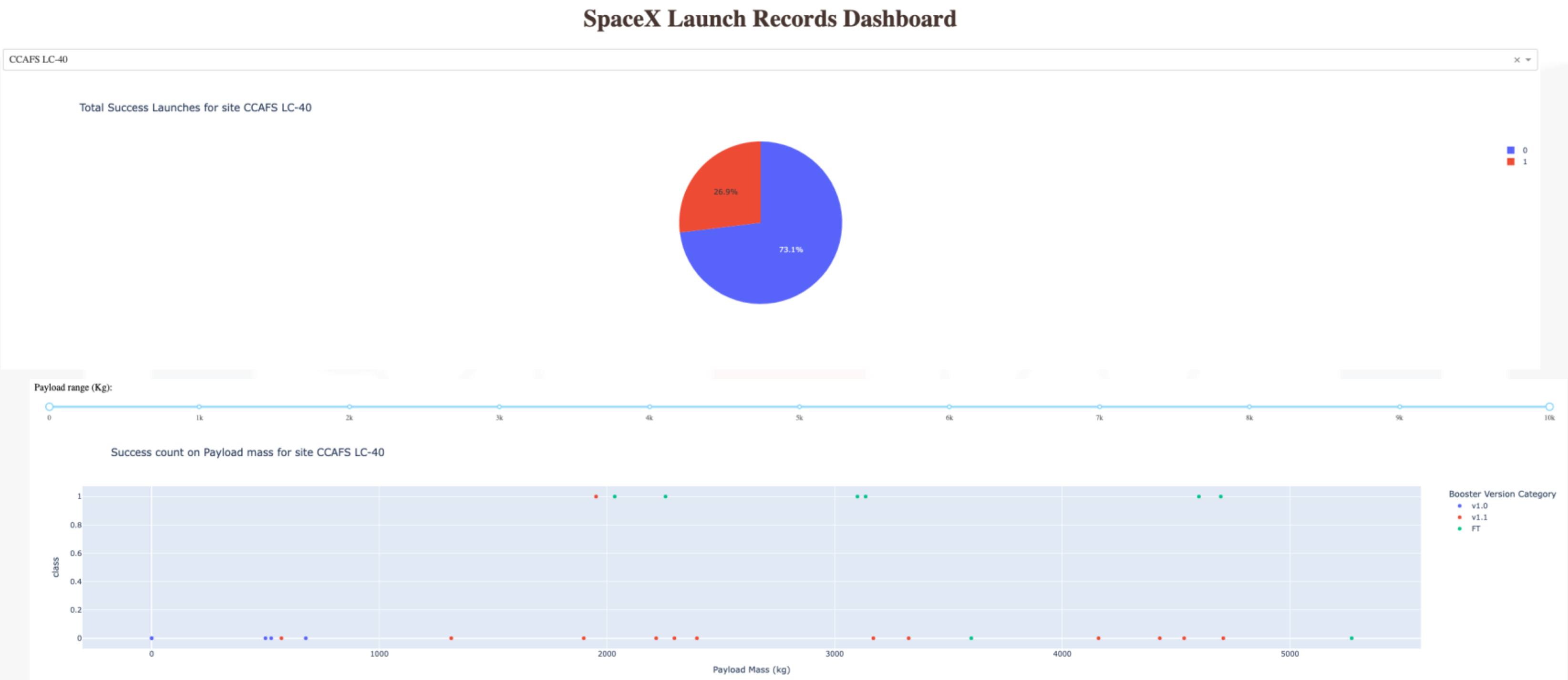


Screenshots of Folium interactive map showing examples of user functionality. Launch Site locations are marked, successful and failed launches indicated for each Site, and distance markers drawn to key geographical features such as distance to coast or distance to nearest city

IBM Developer

SKILLS NETWORK

EDA – Ploty Dash Dashboard



Screenshots of Ploty Dash interactive dashboard. Dashboard can be used to explore relationship between Launch Site, Payload Mass and Class (successful or failed).

Note, CCAFS SLC – 40, CCAFS LC – 40, and KSC LC – 39A had similar proximities to coastline and to the nearest city, but CCCAFS SLC – 40 had the highest success rate.

Link to Dashboard

Predictive Data Analysis

- Logistic Regression, Support Vector Machine, Decision Tree, and K Nearest Neighbors algorithms assessed
  - All algorithms were trained on the same train/test data split
- Key Findings:
  - All four predictive models received the same score, suggesting any of the four models are acceptable to use as predictors

Log Reg Score:
0.8333333333333334
SVM:
0.8333333333333334
Tree:
0.8333333333333334
KNN:
0.8333333333333334

CONCLUSION

- Launch Site, Orbit, Payload Mass, Flight Number, and Year all contribute to the success or failure of a launch
- CCAFS LC – 40 had a success rate of 60%, KSC LC – 39A and VAFB SLC 4E had success rate of 77%
- Orbits ES – L1, GEO, HEO, and SSO had success rate of 100%. SO had a success rate of 0%
- Generally, higher Launch Numbers correlated to higher success rates
  - GTO and ISS orbits did not tend to have a relationship between Flight Number or Payload Mass and success rates
- As the years progressed, the average annual success rate had a general upward trend
- Based on the data, Logistic Regression, Support Vector Machine, Decision Tree, and K Nearest can all be used for predictive classification of future flights

APPENDIX

Collecting Data

SpaceX API to Collect Data

[GitHub Link]

Task 2: Filter the dataframe to only include `Falcon 9` launches

Finally we will remove the Falcon 1 launches keeping only the Falcon 9 launches. Filter the data dataframe using the `BoosterVersion` column to only keep the Falcon 9 launches. Save the filtered data to a new dataframe called `data_falcon9`.

```
[44]: # Hint data['BoosterVersion']!='Falcon 1'
data_falcon9 = launch_df[launch_df['BoosterVersion'] != 'Falcon 1']
data_falcon9.head()
```

```
[44]: 1530
```

Now that we have removed some values we should reset the FlgihtNumber column

```
[32]: data_falcon9.loc[:,'FlightNumber'] = list(range(1, data_falcon9.shape[0]+1))
data_falcon9
```

[32]:

	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins	Reused	Legs	LandingPad	Block	ReusedCount	Serial	Longitude	Latitude
4	1	2010-06-04	Falcon 9	NaN	LEO	CCSFS SLC 40	None None	1	False	False	False	None	1.0	0	B0003	-80.577366	28.561857
5	2	2012-05-22	Falcon 9	525.0	LEO	CCSFS SLC 40	None None	1	False	False	False	None	1.0	0	B0005	-80.577366	28.561857
6	3	2013-03-01	Falcon 9	677.0	ISS	CCSFS SLC 40	None None	1	False	False	False	None	1.0	0	B0007	-80.577366	28.561857
7	4	2013-09-29	Falcon 9	500.0	PO	VAFB SLC 4E	False Ocean	1	False	False	False	None	1.0	0	B1003	-120.610829	34.632093
8	5	2013-12-03	Falcon 9	3170.0	GTO	CCSFS SLC 40	None None	1	False	False	False	None	1.0	0	B1004	-80.577366	28.561857
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
89	86	2020-09-03	Falcon 9	15600.0	VLEO	KSC LC 39A	True ASDS	2	True	True	True	5e9e3032383ecb6bb234e7ca	5.0	12	B1060	-80.603956	28.608058
90	87	2020-10-06	Falcon 9	15600.0	VLEO	KSC LC 39A	True ASDS	3	True	True	True	5e9e3032383ecb6bb234e7ca	5.0	13	B1058	-80.603956	28.608058
91	88	2020-10-18	Falcon 9	15600.0	VLEO	KSC LC 39A	True ASDS	6	True	True	True	5e9e3032383ecb6bb234e7ca	5.0	12	B1051	-80.603956	28.608058
92	89	2020-10-24	Falcon 9	15600.0	VLEO	CCSFS SLC 40	True ASDS	3	True	True	True	5e9e3033383ecbb9e534e7cc	5.0	12	B1060	-80.577366	28.561857
93	90	2020-11-05	Falcon 9	3681.0	MEO	CCSFS SLC 40	True ASDS	1	True	False	True	5e9e3032383ecb6bb234e7ca	5.0	8	B1062	-80.577366	28.561857

90 rows × 17 columns

SpaceX API to Collect Data

GitHub Link

Data Wrangling

We can see below that some of the rows are missing values in our dataset.

```
[33]: data_falcon9.isnull().sum()
```

```
[33]: FlightNumber       0
      Date               0
      BoosterVersion     0
      PayloadMass        5
      Orbit              0
      LaunchSite         0
      Outcome            0
      Flights            0
      GridFins           0
      Reused             0
      Legs               0
      LandingPad        26
      Block              0
      ReusedCount        0
      Serial             0
      Longitude          0
      Latitude           0
      dtype: int64
```

```
[40]: # Calculate the mean value of PayloadMass column
      mean_payloadmass = data_falcon9['PayloadMass'].mean(axis=0)
      mean_payloadmass

      # Replace the np.nan values with its mean value
      data_falcon9['PayloadMass'].replace(np.nan, mean_payloadmass, inplace = True)
      data_falcon9.isnull().sum()
```

```
[40]: FlightNumber       0
      Date               0
      BoosterVersion     0
      PayloadMass        0
      Orbit              0
      LaunchSite         0
      Outcome            0
      Flights            0
      GridFins           0
      Reused             0
      Legs               0
      LandingPad        26
      Block              0
      ReusedCount        0
      Serial             0
      Longitude          0
      Latitude           0
      dtype: int64
```

Webscraping to Collect Data

[GitHub Link]

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.

```
[12]: # use requests.get() method with the provided static_url
requests.get(static_url)
# assign the response to a object
response = requests.get(static_url)
```

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

```
[15]: # Use BeautifulSoup() to create a BeautifulSoup object from a response text content
html_content = response.text
soup = BeautifulSoup(html_content, 'html.parser')
soup.title
```

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

```
[16]: # Use soup.title attribute
soup.title
```

```
[16]: <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

```
[22]: # 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')
html_tables
```

...

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

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

...

You should able to see the columns names embedded in the table header elements `<th>` as follows:

...

Next, we just need to iterate through the `<th>` elements and apply the provided `extract_column_from_header()` to extract column name one by one

```
[26]: column_names = []

# Apply find_all() function with `th` element on first_launch_table
th = first_launch_table.find_all('th')
# Iterate each th element and apply the provided extract_column_from_header() to get a column name
for table in th:
    name = extract_column_from_header(table)

# Append the Non-empty column name (`if name is not None and len(name) > 0`) into a list called column_names
    if name is not None and len(name) > 0:
        column_names.append(name)
```

Check the extracted column names

```
[27]: print(column_names)
```

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

Webscraping to Collect Data

GitHub Link

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

We will create an empty dictionary with keys from the extracted column names in the previous task. Later, this dictionary will be converted into a Pandas dataframe

```
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']=[]
```

Webscraping to Collect Data

[GitHub Link]

To simplify the parsing process, we have provided an incomplete code snippet below to help you to fill up the `launch_dict`. Please complete the following code snippet with TODOs or you can choose to write your own logic to parse all launch tables:

```
[32]:
extracted_row = 0
#Extract each table
for table_number,table in enumerate(soup.find_all('table',"wikitable plainrowheaders collapsible")):
   # get table row
    for rows in table.find_all("tr"):
        #check to see if first table heading is as number corresponding to launch a number
        if rows.th:
            if rows.th.string:
                flight_number=rows.th.string.strip()
                flag=flight_number.isdigit()
        else:
            flag=False
        #get table element
        row=rows.find_all('td')
        #if it is number save cells in a dictonary
        if flag:
            extracted_row += 1
            # Flight Number value
            # TODO: Append the flight_number into launch_dict with key `Flight No.`
            launch_dict['Flight No.'].append(flight_number)
            #print(flight_number)
            datatimelist=date_time(row[0])

            # Date value
            # TODO: Append the date into launch_dict with key `Date`
            date = datatimelist[0].strip(',')
            launch_dict['Date'].append(date)
            #print(date)

            # Time value
            # TODO: Append the time into launch_dict with key `Time`
            time = datatimelist[1]
            launch_dict['Time'].append(time)
            #print(time)

            # Booster version
            # TODO: Append the bv into launch_dict with key `Version Booster`
            bv=booster_version(row[1])
            if not(bv):
                bv=row[1].a.string
            launch_dict['Version Booster'].append(bv)
            print(bv)

            # Launch Site
            # TODO: Append the bv into launch_dict with key `Launch Site`
            launch_site = row[2].a.string
            launch_dict['Launch site'].append(launch_site)
            #print(launch_site)

            # Payload
            # TODO: Append the payload into launch_dict with key `Payload`
            payload = row[3].a.string
            launch_dict['Payload'].append(payload)
            #print(payload)

            # Payload Mass
            # TODO: Append the payload_mass into launch_dict with key `Payload mass`
            payload_mass = get_mass(row[4])
            launch_dict['Payload mass'].append(payload_mass)
            #print(payload)

            # Orbit
            # TODO: Append the orbit into launch_dict with key `Orbit`
            orbit = row[5].a.string
            launch_dict['Orbit'].append(orbit)
            #print(orbit)

            # Customer
            # TODO: Append the customer into launch_dict with key `Customer`
            customer = row[6]
            launch_dict['Customer'].append(customer)
            #print(customer)

            # Launch outcome
            # TODO: Append the launch_outcome into launch_dict with key `Launch_outcome`
            launch_outcome = list(row[7].strings)[0]
            launch_dict['Launch outcome'].append(launch_outcome)
            #print(launch_outcome)

            # Booster landing
            # TODO: Append the launch_outcome into launch_dict with key `Booster_landing`
            booster_landing = landing_status(row[8])
            launch_dict['Booster landing'].append(booster_landing)
            #print(booster_landing)
```

Data Wrangling

GitHub Link

TASK 1: Calculate the number of launches on each site

The data contains several Space X launch facilities: Cape Canaveral Space Launch Complex 40 **VAFB SLC 4E** , Vandenb

Next, let's see the number of launches for each site.

Use the method `value_counts()` on the column `LaunchSite` to determine the number of launches on each site:

```
[6]: # Apply value_counts() on column LaunchSite
df['LaunchSite'].value_counts()
```

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

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`

```
[7]: # Apply value_counts on Orbit column
df['Orbit'].value_counts()
```

```
[7]: Orbit
GTO      27
ISS      21
VLEO     14
PO        9
LEO       7
SSO       5
MEO       3
HEO       1
ES-L1     1
SO        1
GEO       1
Name: count, dtype: int64
```

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

Use the method `.value_counts()` on the column `Outcome` to determine the number of `landing_outcomes` .Then assign it to a variable landing_outcomes.

```
[8]: # landing_outcomes = values on Outcome column
landing_outcomes = df['Outcome'].value_counts()
landing_outcomes
```

```
[8]: Outcome
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: count, dtype: int64
```

`True Ocean` means the mission outcome was successfully landed to a specific region of the ocean while `False Ocean` means the mission outcome was unsuc
was successfully landed to a drone ship `False ASDS` means the mission outcome was unsuccessfully landed to a drone ship. `None ASDS` and `None None` the

```
[9]: 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
```

We create a set of outcomes where the second stage did not land successfully:

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

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

Data Wrangling

[GitHub Link]

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`:

E303 too many blank lines (8)
flake8

[13]:
```
# landing_class = 0 if bad_outcome
# landing_class = 1 otherwise
landing_class = [0 if x in bad_outcomes else 1 for x in df['Outcome']]
landing_class
```

```
[0, •••
```

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

[26]:
```
df['Class']=landing_class
df[['Class']].head(8)
df['Class'].value_counts()
```

[26]:
```
Class
1    60
0    30
Name: count, dtype: int64
```

[15]:
```
df.head(5)
```

[15]:

	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins	Reused	Legs	LandingPad	Block	ReusedCount	Serial	Longitude	Latitude	Class
0	1	2010-06-04	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	2012-05-22	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	2013-03-01	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	2013-09-29	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	2013-12-03	Falcon 9	3170.000000	GTO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B1004	-80.577366	28.561857	0

We can use the following line of code to determine the success rate:

[16]:
```
df["Class"].mean()
```

[16]:
```
np.float64(0.6666666666666666)
```

EDA - SQL

[GitHub Link]

Task 1

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

[45]:
```
%sql select distinct(LAUNCH_SITE) from SPACEXTABLE
```

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

[45]:

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'

[46]:
```
%sql select * from SPACEXTABLE where LAUNCH_SITE like 'CCA%' limit 5
```

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

[46]:

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

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

[47]:
```
%sql select sum(PAYLOAD_MASS__KG_) from SPACEXTABLE where CUSTOMER = 'NASA (CRS)'
```

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

[47]:

sum(PAYLOAD_MASS__KG_)
45596

Task 4

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

[49]:
```
%sql select avg(PAYLOAD_MASS__KG_) from SPACEXTABLE where BOOSTER_VERSION LIKE 'F9 v1.1%'
```

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

[49]:

avg(PAYLOAD_MASS__KG_)
2534.6666666666665

Task 5

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

Hint:Use min function

[51]:
```
%sql select min(DATE) from SPACEXTABLE where Landing_Outcome = 'Success (ground pad)'
```

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

[51]:

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

[52]:
```
%sql select Booster_Version from SPACEXTABLE where Landing_Outcome = 'Success (drone ship)' and PAYLOAD_MASS__KG_ > 4000 AND PAYLOAD_MASS__KG_ < 6000
```

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

[52]:

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

Task 7

List the total number of successful and failure mission outcomes

[61]:
```
%sql select "Mission_Outcome", count("Mission_Outcome") as Total from SPACEXTABLE GROUP BY "Mission_Outcome"
```

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

[61]:

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

EDA - SQL

[GitHub Link]

Task 8

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

[63]:
```
%sql select Booster_Version from SPACEXTABLE where PAYLOAD_MASS__KG_ = (select max(PAYLOAD_MASS__KG_) from SPACEXTABLE)
```

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

[63]:

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, 6,2) as month to get the months and substr(Date,0,5)='2015' for year.

[65]:
```
%sql SELECT SUBSTR(Date, 6, 2) AS Month, Landing_Outcome, Booster_Version, Launch_site from SPACEXTABLE WHERE Landing_Outcome like 'Failure%drone%' AND SUBSTR(Date, 0, 5) = '2015'
```

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

[65]:

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

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.

[73]:
```
%sql SELECT Landing_Outcome, COUNT(*) FROM SPACEXTABLE WHERE (DATE BETWEEN '2010-06-04' AND '2017-03-20') GROUP BY Landing_Outcome ORDER BY COUNT(*) DESC
```

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

[73]:

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

EDA - Visualizations

[GitHub Link]

Exploratory Data Analysis

First, let's read the SpaceX dataset into a Pandas dataframe and print its summary

[4]:
```python
from js import fetch
import io

URL = "https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/datasets/dataset_part_2.csv"
resp = await fetch(URL)
dataset_part_2_csv = io.BytesIO((await resp.arrayBuffer()).to_py())
df=pd.read_csv(dataset_part_2_csv)
df.head(5)
```

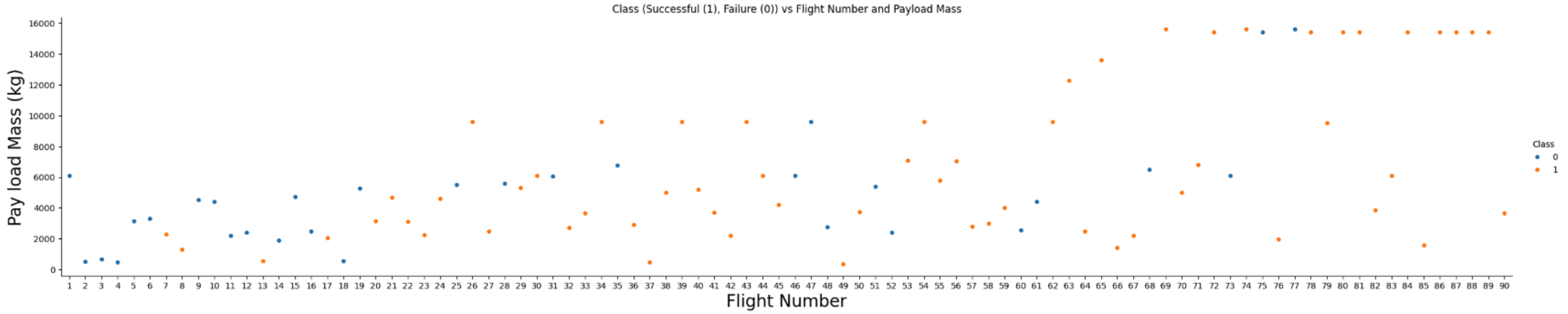
[4]:

	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins	Reused	Legs	LandingPad	Block	ReusedCount	Serial	Longitude	Latitude	Class
0	1	2010-06-04	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	2012-05-22	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	2013-03-01	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	2013-09-29	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	2013-12-03	Falcon 9	3170.000000	GTO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B1004	-80.577366	28.561857	0

First, let's try to see how the `FlightNumber` (indicating the continuous launch attempts.) and `Payload` variables would affect the launch outcome.

We can plot out the `FlightNumber` vs. `PayloadMass` and overlay the outcome of the launch. We see that as the flight number increases, the first stage is more likely to land successfully. The payload mass also appears to be a factor; even with more massive payloads, the first stage often returns successfully.

[5]:
```python
sns.catplot(y="PayloadMass", x="FlightNumber", hue="Class", data=df, aspect = 5)
plt.xlabel("Flight Number",fontsize=20)
plt.ylabel("Pay load Mass (kg)",fontsize=20)
plt.title("Class (Successful (1), Failure (0)) vs Flight Number and Payload Mass")
plt.show()
```



EDA - Visualizations

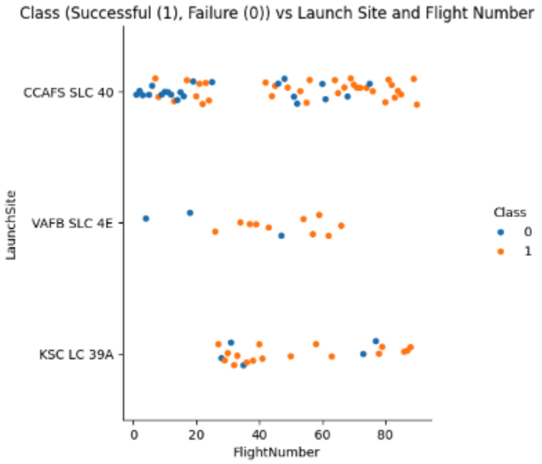
[GitHub Link]

TASK 1: Visualize the relationship between Flight Number and Launch Site

Use the function `catplot` to plot `FlightNumber` vs `LaunchSite`, set the parameter `x` parameter to `FlightNumber`,set the `y` to `Launch Site` and set the parameter `hue` to `'class'`

[6]:
```
# Plot a scatter point chart with x axis to be Flight Number and y axis to be the launch site, and hue to be the class
sns.catplot(y = "LaunchSite", x = "FlightNumber", data = df, hue = 'Class')
plt.title("Class (Successful (1), Failure (0)) vs Launch Site and Flight Number")
```

[6]: `Text(0.5, 1.0, 'Class (Successful (1), Failure (0)) vs Launch Site and Flight Number')`

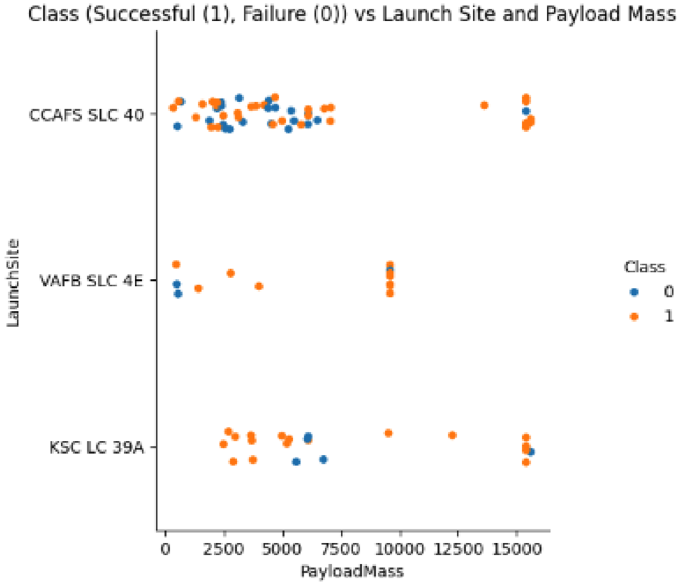


TASK 2: Visualize the relationship between Payload Mass and Launch Site

We also want to observe if there is any relationship between launch sites and their payload mass.

[7]:
```
# Plot a scatter point chart with x axis to be Pay Load Mass (kg) and y axis to be the launch site, and hue to be the class value
sns.catplot(x = 'PayloadMass', y = 'LaunchSite', hue = 'Class', data = df)
plt.title("Class (Successful (1), Failure (0)) vs Launch Site and Payload Mass")
```

[7]: `Text(0.5, 1.0, 'Class (Successful (1), Failure (0)) vs Launch Site and Payload Mass')`



EDA - Visualizations

[GitHub Link]

TASK 3: Visualize the relationship between success rate of each orbit type

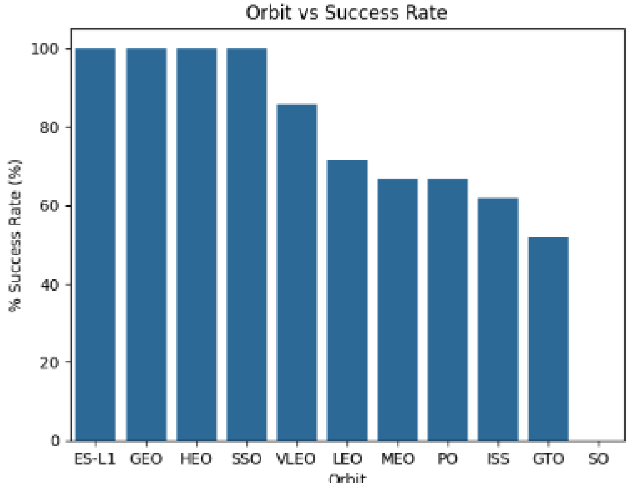
Next, we want to visually check if there are any relationship between success rate and orbit type.

Let's create a `bar chart` for the sucess rate of each orbit

[8]:
```
# HINT use groupby method on Orbit column and get the mean of Class column
data_2 = df.groupby('Orbit')['Class'].mean().reset_index().sort_values(by = 'Class', ascending = False)
data_2['Class'] = data_2['Class']*100

sns.barplot(x = 'Orbit', y = 'Class', data = data_2)
plt.ylabel('% Success Rate (%)')
plt.xlabel('Orbit')
plt.title('Orbit vs Success Rate')
```

[8]: `Text(0.5, 1.0, 'Orbit vs Success Rate')`

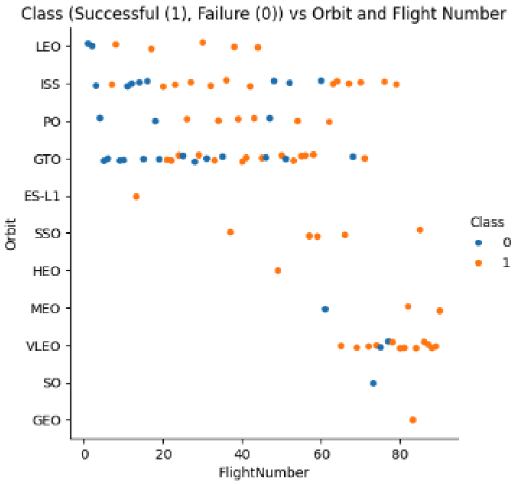


TASK 4: Visualize the relationship between FlightNumber and Orbit type

For each orbit, we want to see if there is any relationship between FlightNumber and Orbit type.

[9]:
```
# Plot a scatter point chart with x axis to be FlightNumber and y axis to be the Orbit, and hue to be the class value
sns.catplot(x = 'FlightNumber', y = 'Orbit', hue = 'Class', data = df)
plt.title("Class (Successful (1), Failure (0)) vs Orbit and Flight Number")
```

[9]: `Text(0.5, 1.0, 'Class (Successful (1), Failure (0)) vs Orbit and Flight Number')`



You can observe that in the LEO orbit, success seems to be related to the number of flights. Conversely, in the GTO orbit, there appears to be no relationship between flight number and success.

EDA - Visualizations

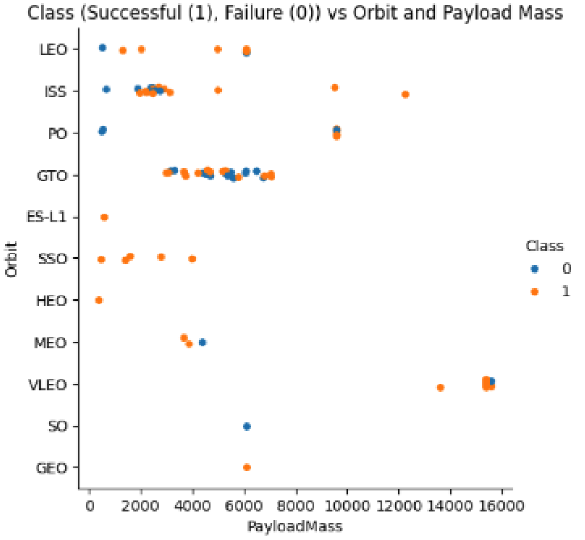
[GitHub Link]

TASK 5: Visualize the relationship between Payload Mass and Orbit type

Similarly, we can plot the Payload Mass vs. Orbit scatter point charts to reveal the relationship between Payload Mass and Orbit type

```
[10]: # Plot a scatter point chart with x axis to be Payload Mass and y axis to be the Orbit, and hue to be the class value
sns.catplot(x = 'PayloadMass', y = 'Orbit', hue = 'Class', data = df)
plt.title("Class (Successful (1), Failure (0)) vs Orbit and Payload Mass")
```

```
[10]: Text(0.5, 1.0, 'Class (Successful (1), Failure (0)) vs Orbit and Payload Mass')
```



With heavy payloads the successful landing or positive landing rate are more for Polar,LEO and ISS.

However, for GTO, it's difficult to distinguish between successful and unsuccessful landings as both outcomes are present.

TASK 6: Visualize the launch success yearly trend

You can plot a line chart with x axis to be `Year` and y axis to be average success rate, to get the average launch success trend.

The function will help you get the year from the date:

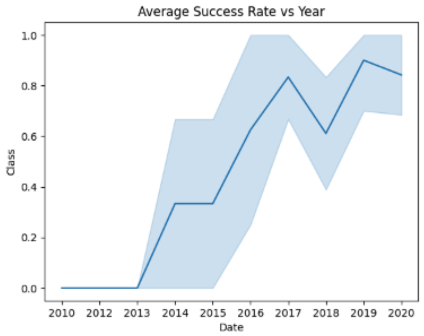
```
[12]: # A function to Extract years from the date
year=[]
def Extract_year():
    for i in df["Date"]:
        year.append(i.split("-")[0])
    return year
Extract_year()
df['Date'] = year
df.head()
```

[12]:

	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

```
[13]: # Plot a line chart with x axis to be the extracted year and y axis to be the success rate
sns.lineplot(x = 'Date', y = 'Class', data = df)
plt.title('Average Success Rate vs Year')
```

```
[13]: Text(0.5, 1.0, 'Average Success Rate vs Year')
```



you can observe that the sucess rate since 2013 kept increasing till 2020

EDA - Visualizations

[GitHub Link]

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.

[16]:
```
features = df[['FlightNumber', 'PayloadMass', 'Orbit', 'LaunchSite', 'Flights', 'GridFins', 'Reused', 'Legs', 'LandingPad', 'Block', 'ReusedCount', 'Serial']]
features.head()
```

[16]:

	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

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.

[21]:
```
# 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()
```

[21]:

	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	False	False	...	False	False	False	False	False	False	False	False	False	False
1	2	525.000000	1	False	False	False	1.0	0	False	False	...	False	False	False	False	False	False	False	False	False	False
2	3	677.000000	1	False	False	False	1.0	0	False	False	...	False	False	False	False	False	False	False	False	False	False
3	4	500.000000	1	False	False	False	1.0	0	False	False	...	False	False	False	False	False	False	False	False	False	False
4	5	3170.000000	1	False	False	False	1.0	0	False	False	...	False	False	False	False	False	False	False	False	False	False

5 rows × 80 columns

TASK 8: Cast all numeric columns to `float64`

Now that our `features_one_hot` dataframe only contains numbers, cast the entire dataframe to variable type `float64`

[22]:
```
# HINT: use astype function
features_one_hot.astype('float64')
```

[22]:

	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

We can now export it to a **CSV** for the next section but to make the answers consistent, in the next lab we will provide data in a pre-selected date range

Interactive Folium Map

[GitHub Link]

Interactive Ploty Dash Dashboard

[GitHub Link]

PDA

GitHub Link

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']).

[44]:
```
Y = data['Class'].to_numpy()
Y
```

[44]:
```
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` using the transform provided below.

[45]:
```
# students get this
transform = preprocessing.StandardScaler()

X = transform.fit_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`

[46]:
```
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.

[47]:
```
Y_test.shape
```

[47]:
```
(18,)
```

PDA – LogReg

[GitHub Link]

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`.

[48]:
```
parameters ={'C':[0.01,0.1,1],
             'penalty':['l2'],
             'solver':['lbfgs']}
```

[53]:
```
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)
```

[53]:
```
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_`.

[54]:
```
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

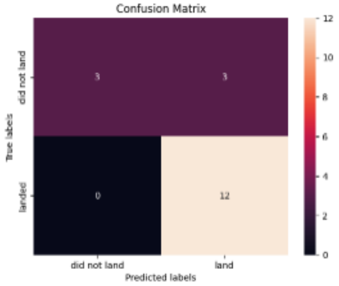
Calculate the accuracy on the test data using the method `score`:

[56]:
```
logreg_cv.score(X_test, Y_test)
```

[56]:
```
0.8333333333333334
```

Lets look at the confusion matrix:

[57]:
```
yhat=logreg_cv.predict(X_test)
plot_confusion_matrix(Y_test,yhat)
```



PDA - SVM

[GitHub Link]

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`.

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

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

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

```
[64]: 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

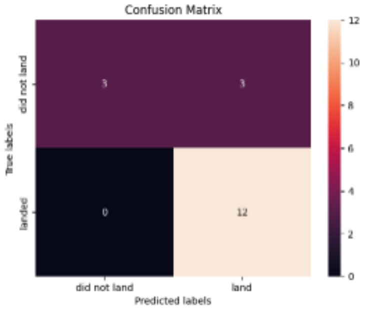
Calculate the accuracy on the test data using the method `score`:

```
[65]: svm_cv.score(X_test, Y_test)
```

```
[65]: 0.8333333333333334
```

We can plot the confusion matrix

```
[66]: yhat=svm_cv.predict(X_test)
      plot_confusion_matrix(Y_test,yhat)
```



PDA – Decision Tree

[GitHub Link]

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`.

[67]:
```
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()
```

[68]:
```
tree_cv = GridSearchCV(tree, parameters, cv = 10)
tree_cv.fit(X_train, Y_train)
```

```
/lib/python3.12/site-packages/sklearn/model_selection/_validation.py:547: FitFailedWarning: ...
```

[69]:
```
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': 'sqrt', 'min_samples_leaf': 2, 'min_samples_split': 5, 'splitter': 'random'}
accuracy : 0.8714285714285713
```

TASK 9

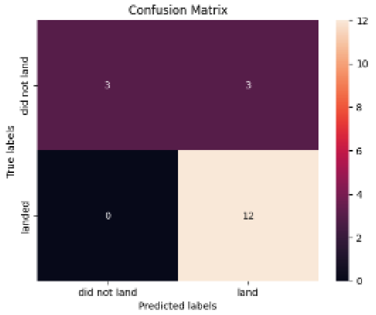
Calculate the accuracy of tree_cv on the test data using the method `score` :

[70]:
```
tree_cv.score(X_test, Y_test)
```

[70]: 0.8333333333333334

We can plot the confusion matrix

[71]:
```
yhat = tree_cv.predict(X_test)
plot_confusion_matrix(Y_test,yhat)
```



PDA - KNN

[GitHub Link]

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`.

```
[72]: 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()
```

```
[73]: knn_cv = GridSearchCV(KNN, parameters, cv = 10)
      knn_cv.fit(X_train, Y_train)
```

```
[73]: GridSearchCV
      estimator: KNeighborsClassifier
        KNeighborsClassifier
```

```
[74]: 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`:

```
[76]: knn_cv.score(X_test, Y_test)

[76]: 0.8333333333333334
```

We can plot the confusion matrix

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