Applied Data Science Capstone

Alexios Korpas

22/03/2024

OUTLINE

- Executive Summary
- Introduction
- Methodology
- Exploratory Data Analysis (EDA)
- Visualization – Charts
- Model prediction
- Results
- Dashboard
- Conclusion
- Appendix

EXECUTIVE SUMMARY

- Determining Falcon9 first stage landing status on SpaceX missiles
- Collection of data from SpaceX REST API to obtain information about (amidst others):
  - Launches
  - Payload delivered
  - Launch Specifications
  - Landing Specifications
  - Landing outcome
- Selection of relevant parameters
- Prediction of landing outcomes
- Results

INTRODUCTION

- SpaceY aims to compete in rocket launching market.
- SpaceX is the main competitor in this market segment.
- Based on available data by SpaceX, we aim to predict whether using the same parameters (payload mass, orbit type, launch site etc.) we can predict a successful or unsuccessful first stage landing.

METHODOLOGY

- Data Wrangling
- Exploratory Data Analysis
- Data Extraction
- Data Visualization
- Implementation of machine learning algorithms to determine landing status and selection of the most accurate model
  - Logistic Regression
  - Support Vector Mechanism (SVM)
  - Decision Trees
  - K-nearest neighbor

Exploratory Data Analysis (EDA)

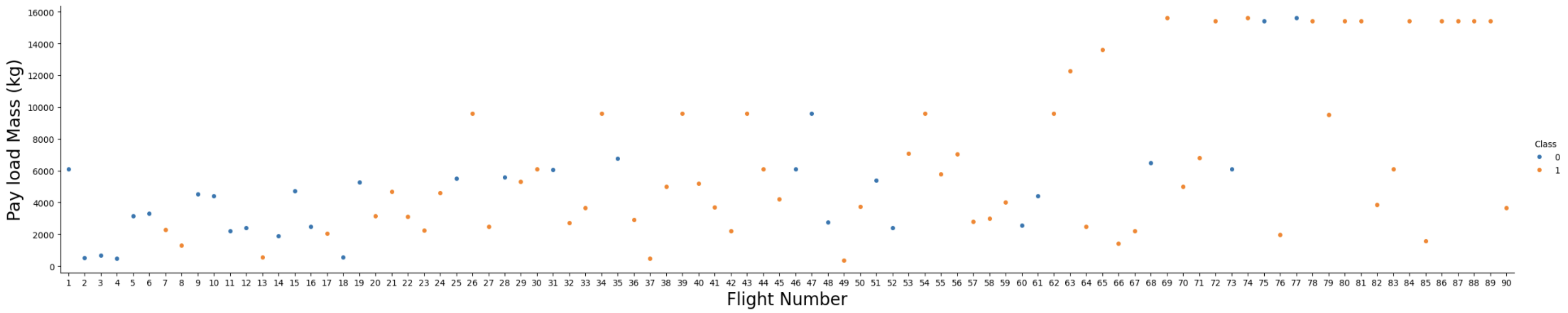
- Exploratory Data Analysis (EDA) to find patterns in the data and determine what would be the label for training supervised models.
- Three (3) SpaceX launch facilities:
  - Cape Canaveral Space Launch Complex **(CCAFS SLC 40)**;
  - Vandenberg Air Force Base Space Launch Complex **(VAFB SLC 4E)**;
  - Kennedy Space Center Launch Complex (**KSC LC 39A**)
- Number of launches per site:
  - CCAFS SLC 40: 55
  - VAFB SLC 4E: 13
  - KSC LC 39A: 22

Exploratory Data Analysis (EDA)

- Successful landings (60 Total):
  - **5 "True Ocean"**: *the mission outcome was successfully landed to a specific region of the ocean;*
  - **14 "True RTLS":** *the mission outcome was successfully landed to a ground pad;*
  - **41"True ASDS":** *the mission outcome was successfully landed to a drone ship.*
- Unsuccessful landings (30 Total):
  - **2 "False Ocean"**: *the mission outcome was unsuccessfully landed to a specific region of the ocean;*
  - **1 "False RTLS"**: *the mission outcome was unsuccessfully landed to a ground pad;*
  - **6 "False ASDS"**: *the mission outcome was unsuccessfully landed to a drone ship.*
  - **19 "None None"**: *failure to land;*
  - **2 "None ASDS"**: *failure to land;*

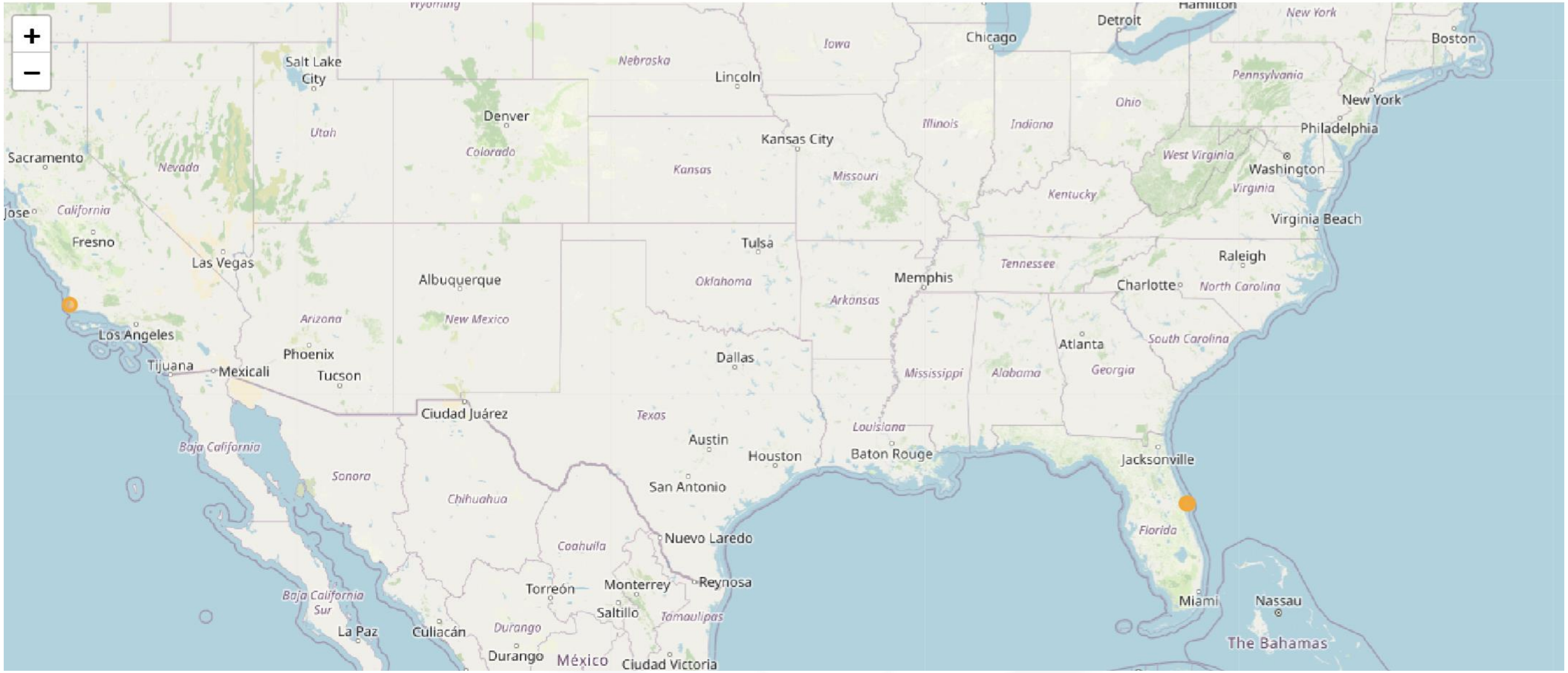
Data Visualization (1/10)

Figure 1: Flight number vs Pay load Mass (kg) per result (Class 0 = Failure to land, Class 1 = Successful landing).



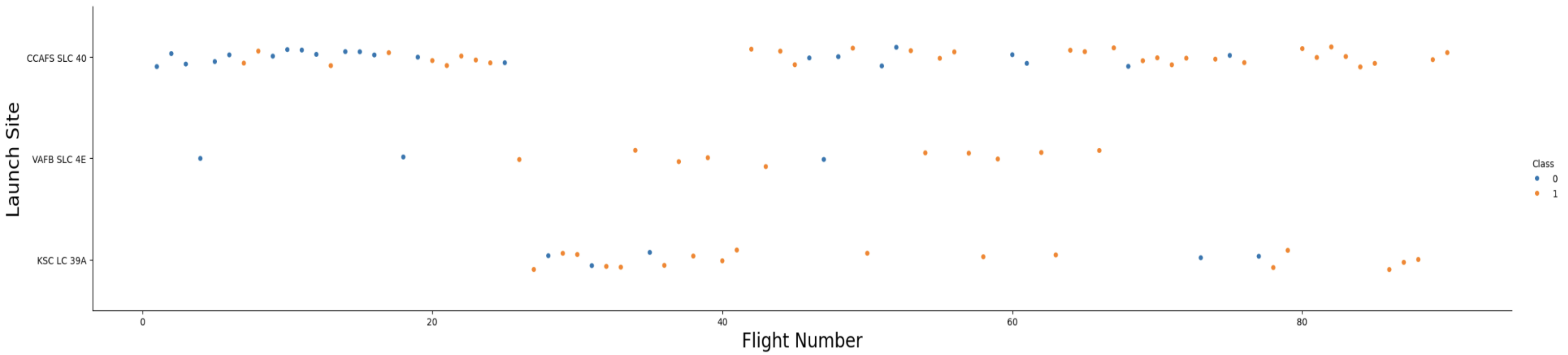
Note: We see that as the flight number increases, the first stage is more likely to land successfully. The payload mass is also important; it seems the more massive the payload, the less likely the first stage will return

Data Visualization (2/10)



Data Visualization (3/10)

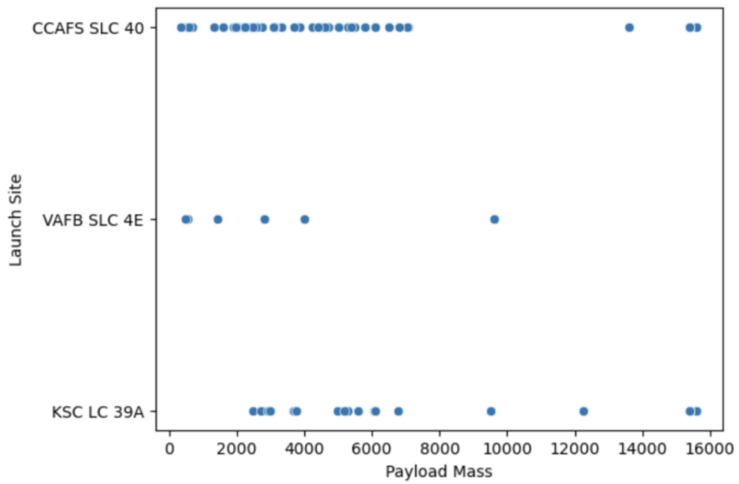
Figure 2: Flight number vs Launch Site (Class 0 = Failure to land, Class 1 = Successful landing)



Note: We see that different launch sites have different success rates. CCAFS LC-40, has a success rate of 60 %, while KSC LC-39A and VAFB SLC 4E has a success rate of 77%.

Data Visualization (4/10)

Figure 3: Payload mass vs Launch Site



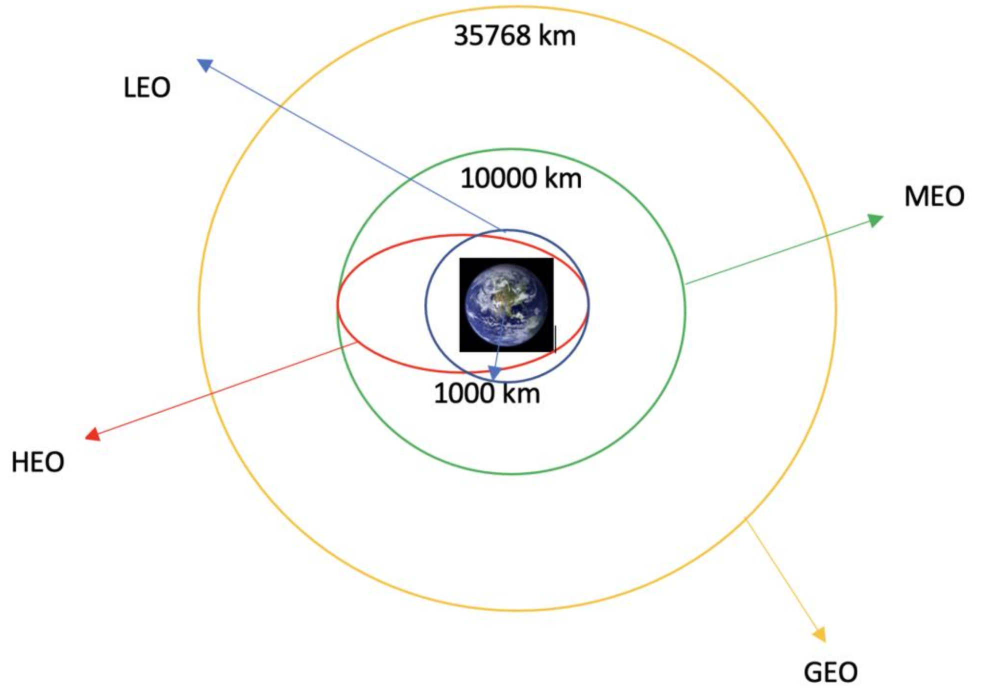
Note: Most launches with payload mass < 8000 took place at Cape Caraveral Space Launch Complex. For the Vandenberg Air Force Base Space Launch Complex, there are no rockets launched for heavy payload mass (greater than 10000).

Data Visualization (5/10)

- Orbit: Each launch aims to an dedicated orbit, and here are some common orbit types:

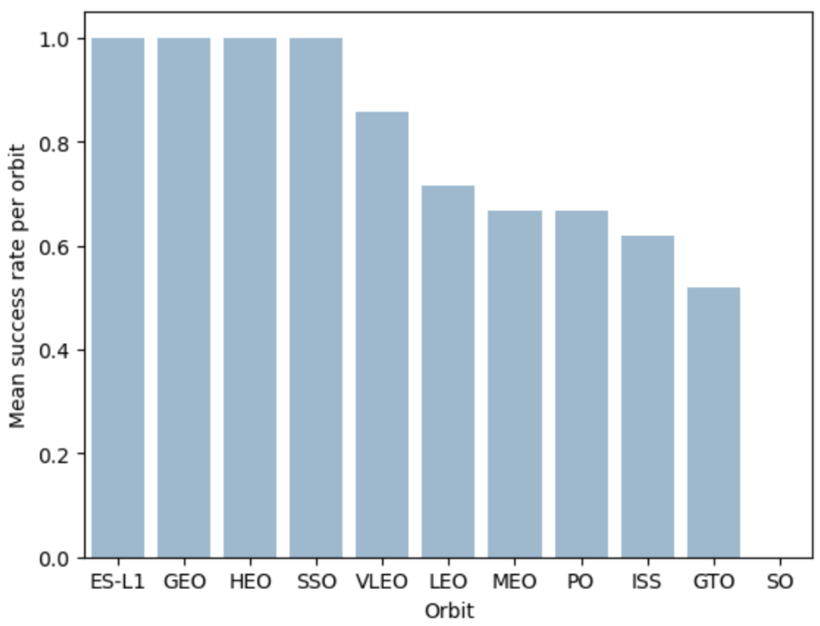
- **LEO**: Low Earth orbit (LEO)is an Earth-centred orbit with an altitude of 2,000 km (1,200 mi) or less (approximately one-third of the radius of Earth),[1] or with at least 11.25 periods per day (an orbital period of 128 minutes or less) and an eccentricity less than 0.25.[2] Most of the manmade objects in outer space are in LEO [1].
- **VLEO**: Very Low Earth Orbits (VLEO) can be defined as the orbits with a mean altitude below 450 km. Operating in these orbits can provide a number of benefits to Earth observation spacecraft as the spacecraft operates closer to the observation[2].
- **GTO** A geosynchronous orbit is a high Earth orbit that allows satellites to match Earth's rotation. Located at 22,236 miles (35,786 kilometers) above Earth's equator, this position is a valuable spot for monitoring weather, communications and surveillance. Because the satellite orbits at the same speed that the Earth is turning, the satellite seems to stay in place over a single longitude, though it may drift north to south," NASA wrote on its Earth Observatory website [3] .
- **SSO (or SO)**: It is a Sun-synchronous orbit also called a heliosynchronous orbit is a nearly polar orbit around a planet, in which the satellite passes over any given point of the planet's surface at the same local mean solar time [4] .
- **ES-L1** :At the Lagrange points the gravitational forces of the two large bodies cancel out in such a way that a small object placed in orbit there is in equilibrium relative to the center of mass of the large bodies. L1 is one such point between the sun and the earth [5] .
- **HEO** A highly elliptical orbit, is an elliptic orbit with high eccentricity, usually referring to one around Earth [6].
- **ISS** A modular space station (habitable artificial satellite) in low Earth orbit. It is a multinational collaborative project between five participating space agencies: NASA (United States), Roscosmos (Russia), JAXA (Japan), ESA (Europe), and CSA (Canada) [7]
- **MEO** Geocentric orbits ranging in altitude from 2,000 km (1,200 mi) to just below geosynchronous orbit at 35,786 kilometers (22,236 mi). Also known as an intermediate circular orbit. These are "most commonly at 20,200 kilometers (12,600 mi), or 20,650 kilometers (12,830 mi), with an orbital period of 12 hours [8]
- **HEO** Geocentric orbits above the altitude of geosynchronous orbit (35,786 km or 22,236 mi) [9]
- **GEO** It is a circular geosynchronous orbit 35,786 kilometres (22,236 miles) above Earth's equator and following the direction of Earth's rotation [10]
- **PO** It is one type of satellites in which a satellite passes above or nearly above both poles of the body being orbited (usually a planet such as the Earth [11]

Data Visualization (6/10)



Data Visualization (7/10)

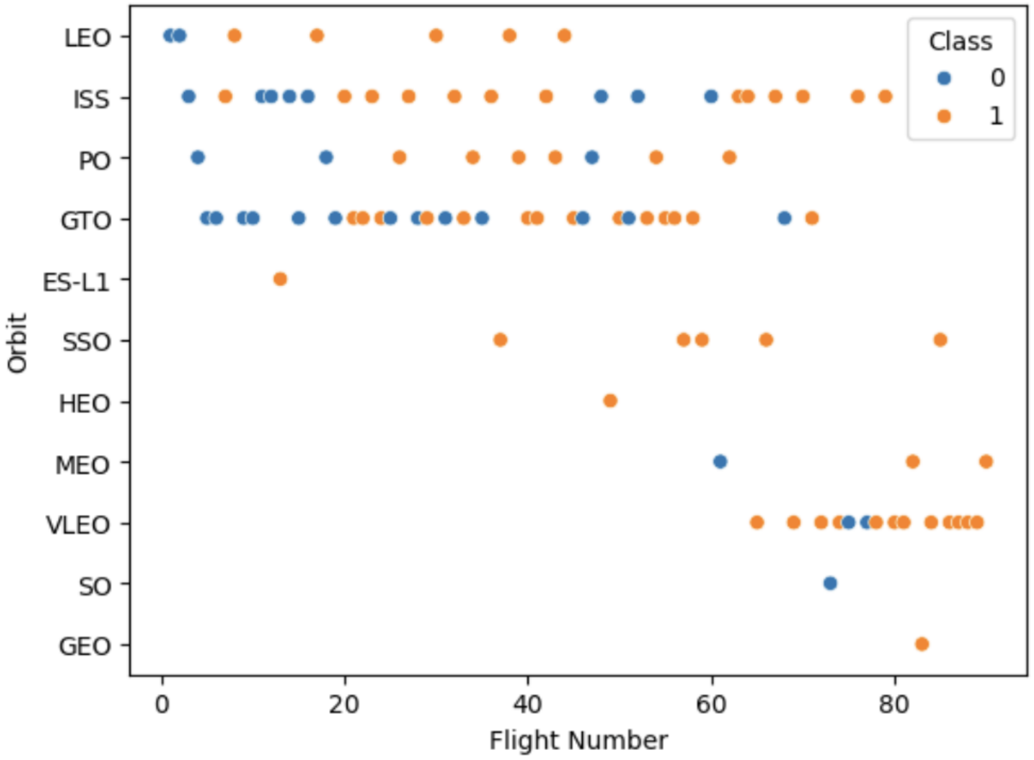
Figure 4: Mean success rate per Orbit type.



Note: The orbit types with absolute success are ES-L1, GEO, HEO, SSO.

Data Visualization (8/10)

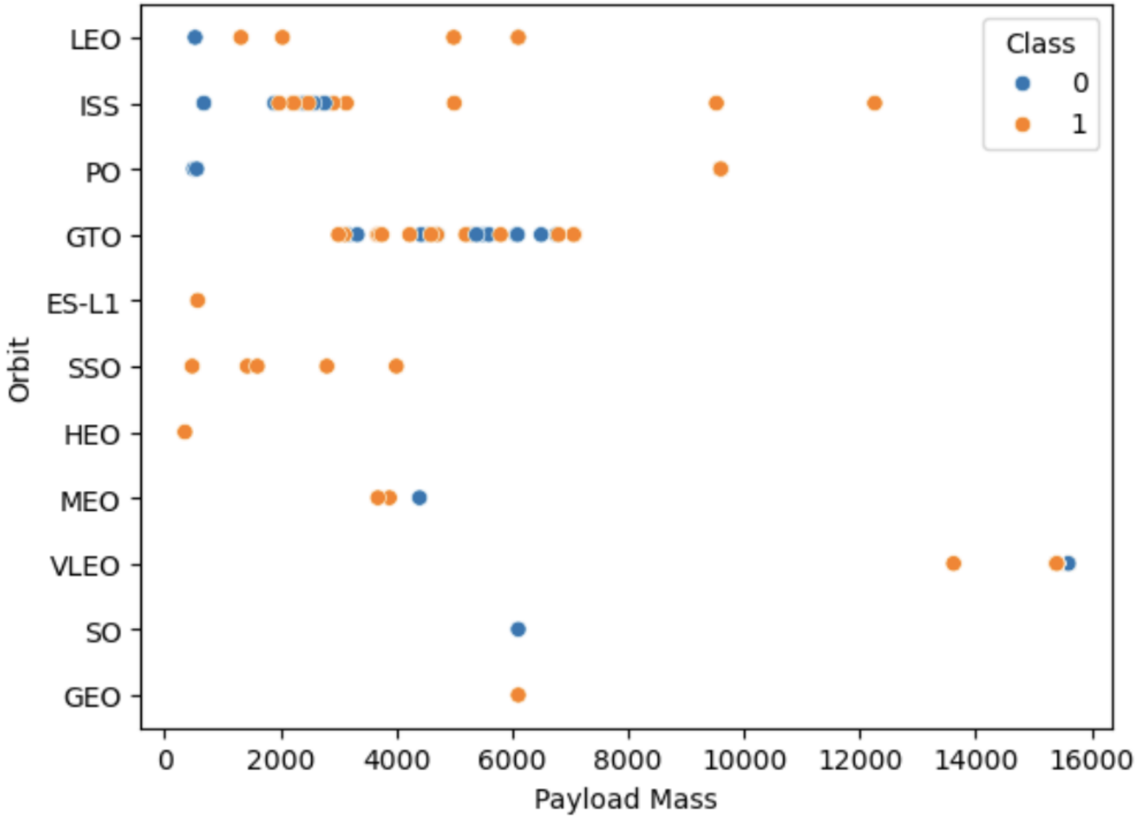
Figure 5: Relationship between Flight Number vs Orbit type (Class 0 = Failure to land, Class 1 = Successful landing).



Note: In the LEO orbit the Success appears related to the number of flights; on the other hand, there seems to be no relationship between flight number when in GTO orbit.

Data Visualization (9/10)

Figure 6: Relationship between Payload Mass vs Orbit type (Class 0 = Failure to land, Class 1 = Successful landing).

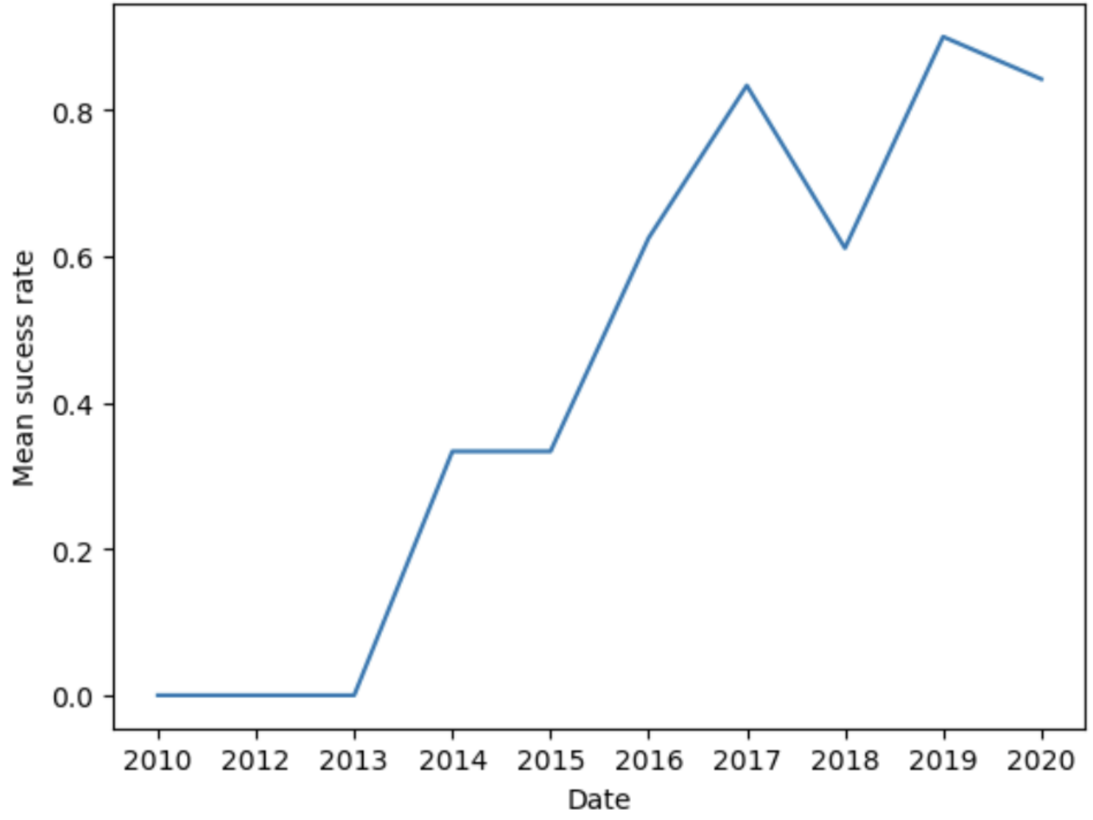


Note: With heavy payloads the successful landing or positive landing rate are more for Polar, LEO and ISS.

However for GTO we cannot distinguish this well as both positive landing rate and negative landing(unsuccessful mission) are both there here.

Data Visualization (10/10)

Figure 7: Mean annual success rate.



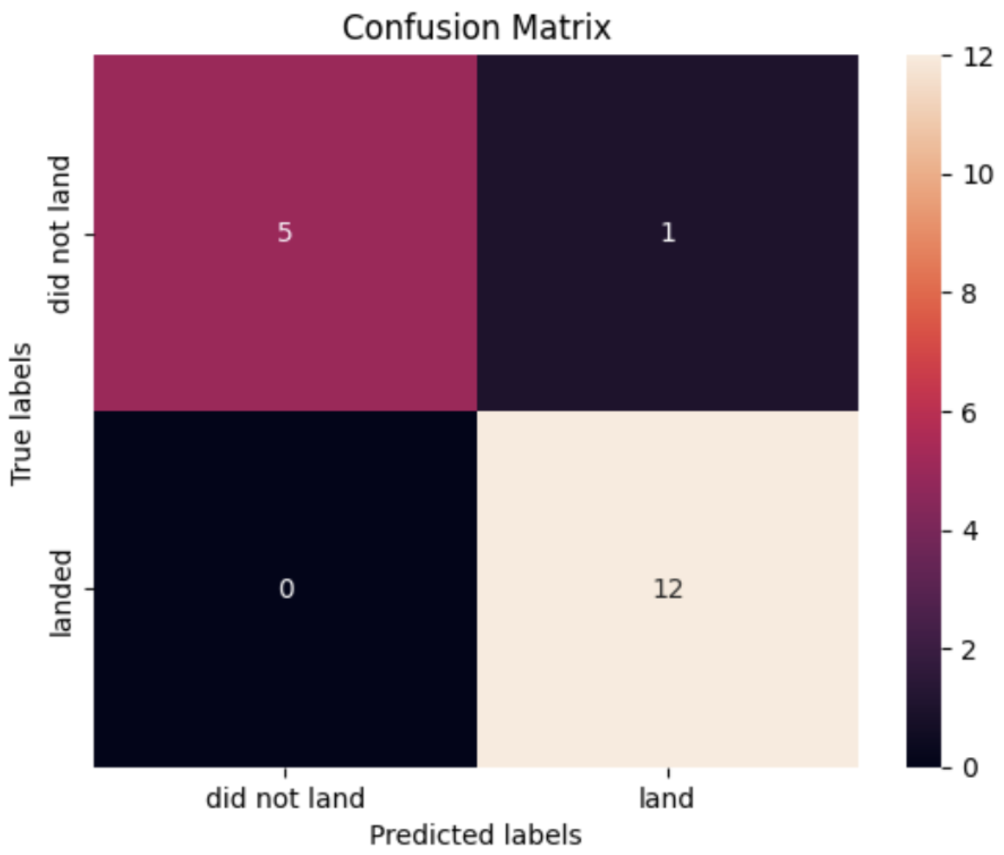
Note: Gradual success rate increasing.

Machine Learning Prediction

- Different Models used to check prediction consistency and compare accuracy to determine best model for implementation based on many parameters
  - Logistic Regression
  - Support Vector Machine
  - Decision Trees
  - K-nearest neighbour

Data was split between training set and test set with 80% of it being in the training set.

Results



Note: Out of the total test set (18 rows), our models accurately predicted the landing outcome of 17 rows.

5 true unsuccessful landings were predicted and 12 out of 13 sucessful landings were predicted.

The model with the highest accuracy was K-nearest neighbors.

DASHBOARD

https://github.com/Ale3isk/ads_capstone

CONCLUSION

- Our model can predict landing outcomes with high accuracy.

Appendix Data Wrangling

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** , Vandenberg Air Force Base Space Launch Complex 4E **(SLC-4E)**, Kennedy Space Center Launch Complex 39A **KSC LC 39A** .The location of each Launch Is placed in the column `LaunchSite`

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:

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

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

Each launch aims to an dedicated orbit, and here are some common orbit types:

TASK 2: Calculate the number and occurrence of each orbit

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

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

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

Appendix Data Wrangling

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

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

```
[9]: {'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`:

```
[15]: # landing_class = 0 if bad_outcome
      # landing_class = 1 otherwise

      my_l = []

      for outcome in df['Outcome']:
          if outcome in bad_outcomes:
              my_l.append(0)
          else:
              my_l.append(1)
      print(my_l)
```

```
[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]
```

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

Appendix SQL

Tasks

Now write and execute SQL queries to solve the assignment tasks.

Note: If the column names are in mixed case enclose it in double quotes For Example "Landing_Outcome"

Task 1

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

```
[9]: %sql select DISTINCT(Launch_Site) from SPACEXTABLE
```

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

[9]:

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'

```
[12]: %sql SELECT * from SPACEXTABLE WHERE Launch_Site LIKE "CCA%" LIMIT 5
```

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

[12]:

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)

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

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

[15]:

SUM(PAYLOAD_MASS__KG_)
45596

Task 4

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

```
[17]: %sql SELECT * FROM SPACEXTABLE LIMIT 5
```

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

[17]:

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

```
[18]: %sql SELECT AVG(PAYLOAD_MASS__KG_) FROM SPACEXTABLE WHERE Booster_Version = "F9 v1.1"
```

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

[18]:

AVG(PAYLOAD_MASS__KG_)
2928.4

Appendix SQL

Task 5

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

Hint:Use min function

```
[20]: %sql SELECT DISTINCT LANDING_OUTCOME FROM SPACEXTABLE
 * sqlite:///my_data1.db
Done.
```

[20]:

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

```
[23]: %sql SELECT MIN(DATE) FROM SPACEXTABLE WHERE LANDING_OUTCOME = "Success"
 * sqlite:///my_data1.db
Done.
```

[23]:

MIN(DATE)
2018-07-22

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

```
[28]: %sql SELECT DISTINCT(Booster_Version) FROM SPACEXTABLE WHERE Landing_Outcome = "Success" AND PAYLOAD_MASS__KG_ BETWEEN 4000 AND 6000;
 * sqlite:///my_data1.db
Done.
```

[28]:

Booster_Version
F9 B5 B1046.2
F9 B5 B1047.2
F9 B5 B1046.3
F9 B5 B1048.3
F9 B5 B1051.2
F9 B5B1060.1
F9 B5 B1058.2
F9 B5B1062.1

Task 7

List the total number of successful and failure mission outcomes

```
[33]: %sql SELECT COUNT(Mission_Outcome) as SUM, Mission_Outcome FROM SPACEXTABLE GROUP BY Mission_Outcome
 * sqlite:///my_data1.db
Done.
```

[33]:

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

Task 7

List the total number of successful and failure mission outcomes

```
[33]: %sql SELECT COUNT(Mission_Outcome) as SUM, Mission_Outcome FROM SPACEXTABLE GROUP BY Mission_Outcome
 * sqlite:///my_data1.db
Done.
```

[33]:

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

Task 8

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

```
[37]: %sql SELECT DISTINCT Booster_Version FROM SPACEXTABLE WHERE PAYLOAD_MASS__KG_ = (SELECT MAX(PAYLOAD_MASS__KG_) FROM SPACEXTABLE)
 * sqlite:///my_data1.db
Done.
```

[37]:

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

Appendix 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.

[52]:

```
%%sql SELECT CASE substr(Date, 6, 2)
WHEN '01' THEN 'January'
WHEN '02' THEN 'February'
WHEN '03' THEN 'March'
WHEN '04' THEN 'April'
WHEN '05' THEN 'May'
WHEN '06' THEN 'June'
WHEN '07' THEN 'July'
WHEN '08' THEN 'August'
WHEN '09' THEN 'September'
WHEN '10' THEN 'October'
WHEN '11' THEN 'November'
WHEN '12' THEN 'December'
END AS Month,
Booster_Version, launch_site
FROM SPACEXTABLE
WHERE Landing_Outcome = "Failure (drone ship)" AND substr(Date,0,5) = "2015"
```

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

[52]:

Month	Booster_Version	Launch_Site
January	F9 v1.1 B1012	CCAFS LC-40
April	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.

[53]:

```
%sql SELECT * FROM SPACEXTABLE LIMIT 1
```

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

[53]:

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)

[61]:

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

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

[61]:

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

Appendix Folium

TODO: Create a new column in `launch_sites` dataframe called `marker_color` to store the marker colors based on the `class` value

[30]:
```python
# Apply a function to check the value of `class` column
# If class=1, marker_color value will be green
# If class=0, marker_color value will be red

launch_sites = []
for result in spacex_df['class']:
    if result == 0:
        launch_sites.append('red')
    else:
        launch_sites.append('green')

spacex_df['marker_color'] = launch_sites
```

TODO: For each launch result in `spacex_df` data frame, add a `folium.Marker` to `marker_cluster`

[31]:
```python
# Add marker_cluster to current site_map
site_map.add_child(marker_cluster)

# for each row in spacex_df data frame
# create a Marker object with its coordinate
# and customize the Marker's icon property to indicate if this launch was successed or failed,
# e.g., icon=folium.Icon(color='white', icon_color=row['marker_color']]
for index, record in spacex_df.iterrows():
    # TODO: Create and add a Marker cluster to the site map
    marker = folium.Marker(
        location=[record['Lat'],record['Long']],
        icon=folium.Icon(color='white',icon_color=record['marker_color']),
        popup=record['class']
    )
    marker_cluster.add_child(marker)

site_map
```

[ ]:
```python
# TASK 3: Calculate the distances between a launch site to its proximities
```

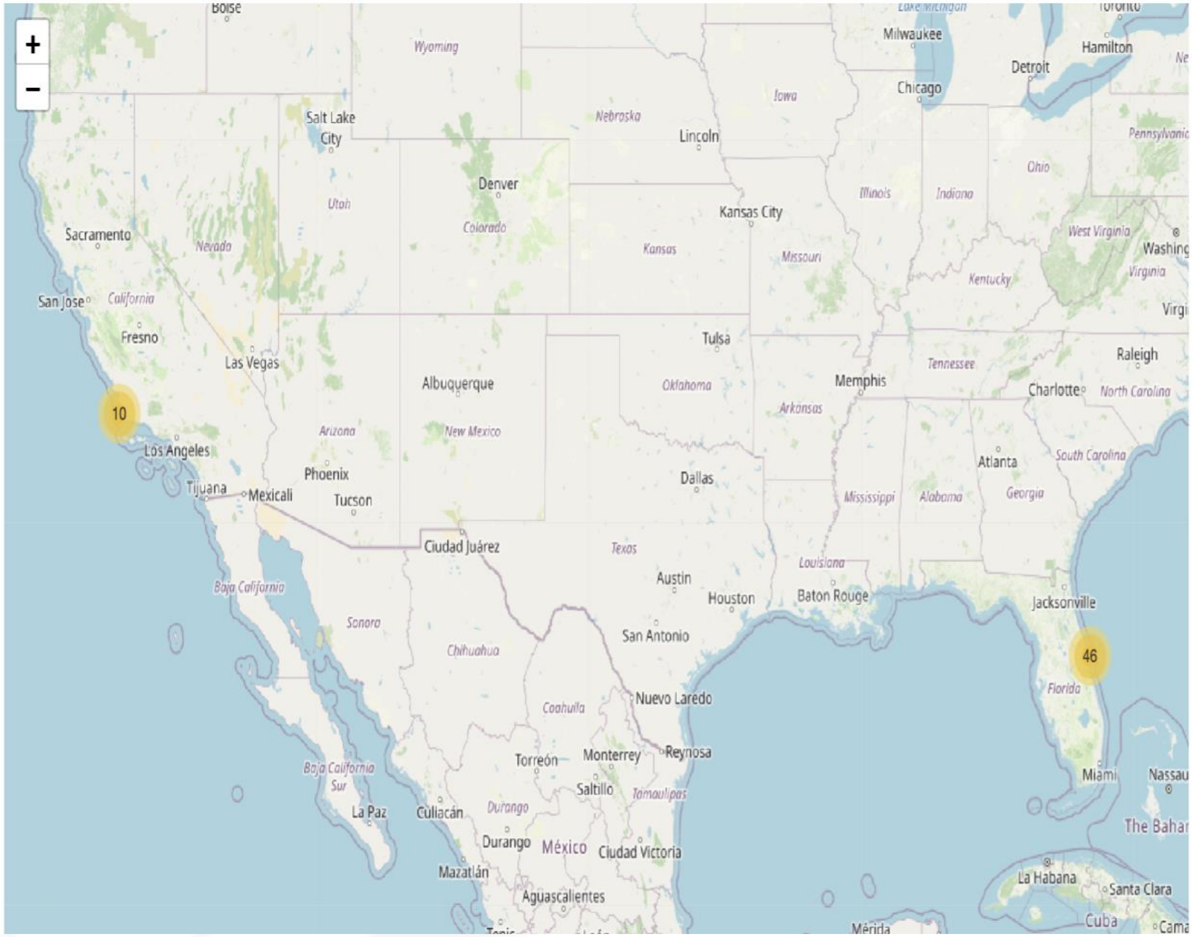
Next, we need to explore and analyze the proximities of launch sites.

Let's first add a `MousePosition` on the map to get coordinate for a mouse over a point on the map. As such, while you are exploring the map, you can easily find the coordinates of any points of interests (such as railway)

[32]:
```python
# Add Mouse Position to get the coordinate (Lat, Long) for a mouse over on the map
formatter = "function(num) {return L.Util.formatNum(num, 5);};"
mouse_position = MousePosition(
    position='topright',
    separator=' Long: ',
    empty_string='NaN',
    lng_first=False,
    num_digits=20,
    prefix='Lat:',
    lat_formatter=formatter,
    lng_formatter=formatter,
)

site_map.add_child(mouse_position)
site_map
```

[31]:



Appendix Folium

```
[33]: from math import sin, cos, sqrt, atan2, radians

def calculate_distance(lat1, lon1, lat2, lon2):
    # approximate radius of earth in km
    R = 6373.0

    lat1 = radians(lat1)
    lon1 = radians(lon1)
    lat2 = radians(lat2)
    lon2 = radians(lon2)

    dlon = lon2 - lon1
    dlat = lat2 - lat1

    a = sin(dlat / 2)**2 + cos(lat1) * cos(lat2) * sin(dlon / 2)**2
    c = 2 * atan2(sqrt(a), sqrt(1 - a))

    distance = R * c
    return distance
```

TODO: Mark down a point on the closest coastline using MousePosition and calculate the distance between the coastline point and the launch site.

```
[34]: # find coordinate of the closet coastline
# e.g.,: Lat: 28.56367  Lon: -80.57163
launch_site_latitute = 28.56321
launch_site_longitude = -80.57679
coastline_latitute = 28.56347
coastline_longitude = -80.56785
# distance_coastline = calculate_distance(launch_site_lat, launch_site_lon, coastline_lat, coastline_lon)
distance = calculate_distance(launch_site_latitute,launch_site_longitude,coastline_latitute,coastline_longitude)
```

```
[35]: # Create and add a folium.Marker on your selected closest coastline point on the map
# Display the distance between coastline point and launch site using the icon property
# for example
coordinates = [coastline_latitute,coastline_longitude]
distance_marker = folium.Marker(
    coordinates,
    icon=DivIcon(
        icon_size=(20,20),
        icon_anchor=(0,0),
        html='<div style="font-size: 12; color:#d35400;"><b>%s</b></div>' % "{:10.2f} KM".format(distance),
        )
    )

site_map.add_child(distance_marker)
site_map
```

[35]:

TODO: Draw a `PolyLine` between a launch site to the selected coastline point

```
[36]: # Create a `folium.PolyLine` object using the coastline coordinates and launch site coordinate
lines=folium.PolyLine(locations=[[28.56347, -80.56785],[launch_site_latitute,launch_site_longitude]], weight=1)
site_map.add_child(lines)
```

[36]:

Appendix Dash