IBM Developer
SKILLS NETWORK

Winning Space Race with Data Science

Ian Gibbs
Nov. 9, 2022

Outline

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

Executive Summary

- Summary of methodologies
  - Data was collected from SpaceX using a JSON-centric API and web-scraping techniques
  - Exploratory Data Analysis using Python/Pandas, SQL, and graphing/charting to gain preliminary understanding of the data collected and to undertake minor “cleanup”
- Summary of all results
  - SpaceX Launch sites are close to railways, roads, and/or waterways (within 1.5km) and are at least a modest distance from townships/populations (at least 14km)
  - Launch site KSC LC 39A has the highest percentage of successful launches (76.9%) and site CCAFS LC-40 has the lowest (26.9%)
  - Payloads in the range of 3000 - 4000 kg are the most successful (70%) and payloads in the range of 6000 – 7000 kg were the least successful (0%). Of the booster categories that had been used in multiple launches, the “FT” boosters were the most successful (66.7%). The “B5” boosters have a 100% success rating, but it was only used 1 time so it doesn’t seem like a good basis for judgment.
  - The Decision Tree model should be used, given this data, to predict the successful outcome of future Falcon 9 centric missions as it has a slightly higher accuracy score than the other models.

Introduction

- Launching rockets into space is notoriously expensive. SpaceX has found a way to reduce the costs of these launches by re-using the first-stage rocket booster. (Competitors are advertising launches for $165M, whereas SpaceX is advertising launches for $62M.) However, boosters don't always land/return-to-Earth successfully (intact) and thus can't be reused.
- The aim of this study is to examine several factors that facilitate successful landings. If we can predict which types of launches will result in a reusable booster and which won't, we can set pricing accordingly.
- How do the Launch Site, Payload Weight, Booster Type, and Sequence (being among the earliest vs. most recent missions) affect the ability to recover booster rockets? If all other factors remained the same, which variable values would be the most likely to result in a successfully-recovered booster rocket?

Section 1

Methodology

Methodology - Summary

- Data collection methodology – details start here
  - SpaceX Falcon 9 launch records were collected from Space X using their REST API
  - SpaceX Falcon 9 launch records were also collected from Wikipedia using web scraping techniques
- Perform data wrangling – details start here
  - Explored the data using Pandas functions to gain familiarity
  - Created a new column/label to summarize if a mission is successful or not; to be used in future queries/analysis
- Perform exploratory data analysis (EDA) using visualization and SQL – details start here
  - Explored the data using SQL queries to gain familiarity
  - Explored the data using charts/graphs to gain familiarity with the relationship between different data points
- Perform interactive visual analytics using Plotly Dash and Folium – details start here
  - Created a Plotly Dashboard to facilitate interactive exploration of key data elements and their relationship to each other
  - Created a Folium map to gain an understanding of the environmental factors related to launches/landings
- Perform predictive analysis using classification models – details start here
  - Built, tuned, evaluated 4 classification models
  - Determined which model worked best for this dataset

Data Collection

- Data was collected using both REST API calls and web-scraping HTML tables.
- As we progressed from one analysis to the next, the gathered data was made to be exported to CSV files that could be reused in successive work.
- The following 3 slides describe the data collection and data wrangling processes

Data Collection – SpaceX API

- The general flowchart for importing SpaceX data using their RESTful API is documented to the right
- For further detail, see the exact code at: https://github.com/icgibbs/DataScienceCapstone/blob/322f36e9ddd4d9a18ce362dc355effc82ad05d9d/Wk%201,%20Lab%201:%20Data%20Collection%20API%20Notebook.ipynb

1. Import appropriate libraries (including "requests"), and create new functions to pull specific data elements
2. Use "get" and "read_json" functions to pull in JSON file into a dataframe
3. Clean data (remove irrelevant columns & rows, convert text string to date format, etc.)
4. Use functions from first step to convert "IDs" to meaningful data
5. Combine the data from said functions into a dictionary, then into a new dataframe
6. Final data cleaning: filter out Falcon 1 rockets (leaving only Falcon 9), reset the Flight # (so it's sequencing only the Falcon 9 launches), and replace the unspecified PayloadMasses with the average Payload Mass for all

Data Collection – Web Scraping

- The general flowchart for importing SpaceX data scraping data from HTML tables (on a Wikipedia page) is documented to the right
- For further detail, see the exact code at: https://github.com/icgibbs/DataScienceCapstone/blob/322f36e9ddd4d9a18ce362dc355effc82ad05d9d/Wk%201,%20Lab%202:%20Data%20Collection%20with%20Web%20Scraping%20Notebook.ipynb

1. Import appropriate libraries (including "requests" and "BeautifulSoup"), and create new functions to pull specific data elements from the HTML tables
2. Use the "get" function to pull in the HTML table as a block of text, then convert it into a BeautifulSoup object
3. Iterate through the HTML tables, adding individual data elements/cells into data dictionary
4. Convert the data dictionary into a dataframe
5. Note: additional "data cleaning" was not needed in this case as the "get_mass" function defined in step 1 already converted null Payload mass values to 0, the mechanism to iterate through the HTML table ignores rows that don't start with a number (and thus filters out anything except past Falcon 9 launches), and the flight numbers are already properly sequenced for Falcon 9 flights.

Data Wrangling

- The general flowchart for data wrangling is presented to the right
- For further detail, see the exact code at: https://github.com/icgibbs/DataScienceCapstone/blob/322f36e9ddd4d9a18ce362dc355effc82ad05d9d/Wk%201,%20Lab%203:%20Data%20Wrangling%20EDA%20notebook.ipynb

1. Data that had been imported using the REST API technique was read into a dataframe
2. Explore the data, including
    1. determining which columns are missing data (LandingPad)
    2. which are numeric vs. categorical
    3. # of launches from each site
    4. # of launches into each orbit type
    5. # of each Mission Outcomes
3. Create a new column in the dataframe to indicate if the "landing outcome" was successful (value = 1) or not (value = 0)

EDA with Data Visualization

- Created scatterplots to answer the questions:
  - "Does more experience bring with it better odds of successful missions at each of the different launch sites?" Answer: Yes, generally – see Flight Number vs. Launch Site
  - "Are some sites better able to handle heavier payloads?" Answer: Inconclusive:– see Payload vs. Launch Site
  - "Does more experience bring with it better odds of successful missions into the different orbit ranges?" Answer: True for some orbit ranges but not all – see Flight Number vs. Orbit Type
  - "Do we have more success launching heavier payloads into different orbit ranges?" Answer: Partially true – see Payload vs. Orbit Type
- Created a bar chart to answer the question:
  - "Are we more likely to successfully launch into certain orbit ranges?" Answer: Yes – see Success Rate vs. Orbit Type
- Created a line chart to answer the question:
  - "Is our overall success rate improving over time?" Answer: Yes, generally – see Launch Success Yearly Trend
- Link to notebook: https://github.com/icgibbs/DataScienceCapstone/blob/322f36e9ddd4d9a18ce362dc355effc82ad05d9d/Wk%202,%20Lab%202:%20jupyter-labs-eda-dataviz.ipynb

EDA with SQL

- Summary of SQL queries performed during Exploratory Data Analysis:
  - Used “update” query to change a “Mission_Outcome” from “Success ” to “Success” (removed trailing space)
  - Used “where” clause with wildcard (“%”) and a “limit 5” clause to provide list of 5 records in which the launch site had a name starting with “CCA”
  - Used “groupby” with “sum”, “avg”, and “count” to find the aggregate sum, average, and quantity of values, respectively, in a given column
  - Used “substr” to parse date information from a string, and “order by” to sort the results in descending order
  - Used complex where clause (with “and” statements) to restrict results to only those records meeting a combination of criteria
  - Used a subquery to determine that highest payload weight so that the main query could select all of the booster versions that had carried that weight.
- Additional information: The results of the SQL queries are returned as a list of tuples, which aren’t always easy to read. I converted these lists into dataframes, which were easier to follow.
- The SQL and resulting data can be seen in this notebook: https://github.com/icgibbs/DataScienceCapstone/blob/322f36e9ddd4d9a18ce362dc355effc82ad05d9d/Wk%202,%20Lab%201:%20jupyter-labs-eda-sql-coursera_sqllite.ipynb

Build an Interactive Map with Folium

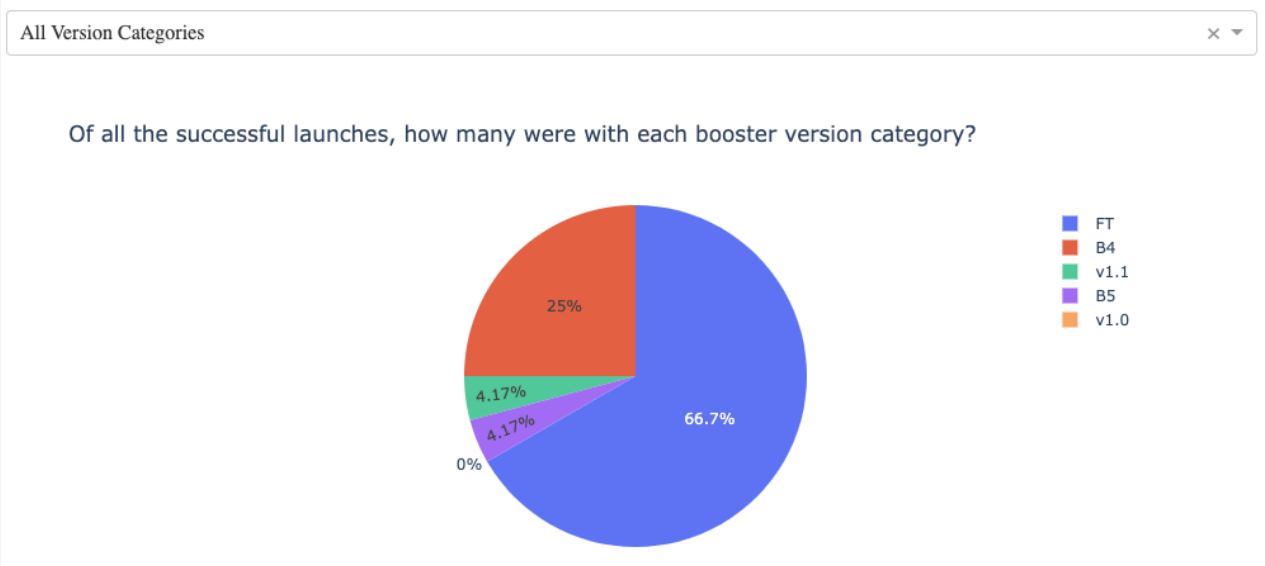
- The following objects were added to a Folium-based map:
  - Circles were added to represent launch sites. (See Map of Launch Sites, Magnified)
  - Markers were added to label/name the launch sites and nearby "points of interest" (shore, train, roadway, etc.). (See Items Close to CCAFS SLC-40)
  - Marker Clusters were added to group the launches that occur at the same sites (See Successes/Failures Pictured at Vandenberg)
  - Lines (PolyLine) were added to reflect the direct (and measured) path between a launch site and nearby "points of interest". (See Items Close to CCAFS SLC-40)
  - A MousePosition object was added to show the Lat/Long coordinates of where the mouse cursor was placed within the map
- Notebook stored in GitHub: https://github.com/icgibbs/DataScienceCapstone/blob/b5ed252f3ec6932ad2d18b2e8633d36899fd13e7/Wk%203,%20Lab%201:%20lab_jupyter_launch_site_location.ipynb

Build a Dashboard with Plotly Dash – Part 1

- An interactive dashboard was originally built with:
  - A pie chart showing the % of successful launches at each site, where "site" is selected from a menu. (See Dashboard: Successful Launches, By Site and Dashboard: Study of Site with Most Successful Launch Ratio)
  - A scatterplot chart showing the success of launches of different weights using different booster versions, where the payload weight is selected from a slider. (See Which Booster Version Worked Best with Which Payload Weights and Which Booster Version Worked Best in Specific Payload Weight Range)
- Notebook stored in GitHub: https://github.com/icgibbs/DataScienceCapstone/blob/b5ed252f3ec6932ad2d18b2e8633d36899fd13e7/Wk%203,%20Lab%202:%20spacex_dash_app.py

Build a Dashboard with Plotly Dash – Part 2

- An additional pie chart was added to the Interactive Dashboard to show
  - The % of successful launches using each of the booster categories, where “booster category” is selected from a menu.
  - In addition to showing which booster versions have been most successful, its selection menu was also tied to the scatterplot chart mentioned above as an additional input/filter to help focus the reader’s attention to the weights used by specific booster versions.



Predictive Analysis (Classification)

- The general flowchart for building, evaluating, improving, and determining the best performing classification model is documented to the right
- For further detail, see the exact code at: https://github.com/icgibbs/DataScienceCapstone/blob/b5ed252f3ec6932ad2d18b2e8633d36899fd13e7/Wk%204,%20Lab%201:%20SpaceX_Machine%20Learning%20Prediction_Part_5.ipynb

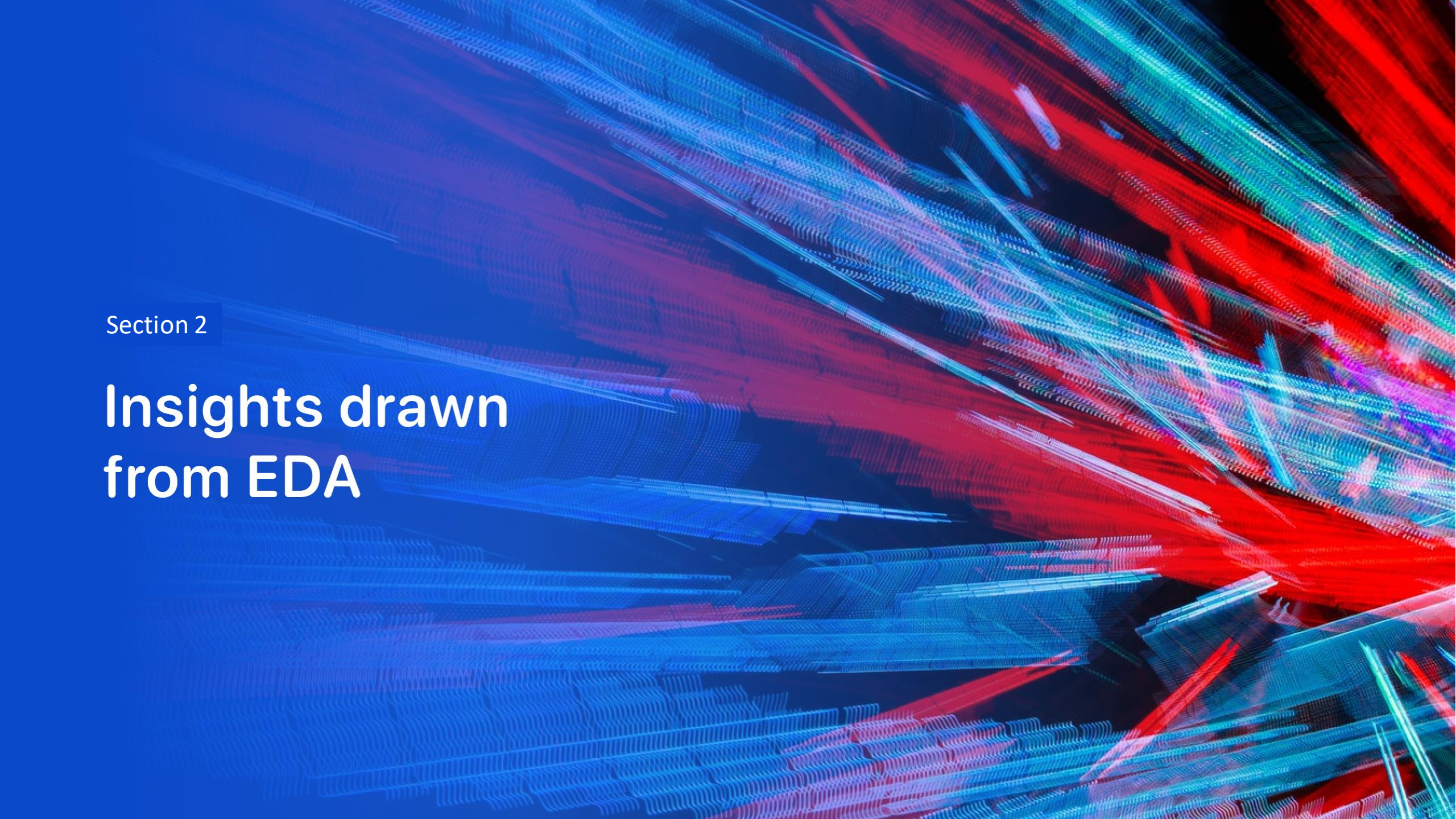
1. Preliminary tasks: Import appropriate libraries (including "sklearn"), create a new function to plot a confusion matrix, and load data
2. Standardize the data
3. Split data into a subset for training and another for testing
4. Create a predictive analysis object (either Logistic Regression, Support Vector Machine, Decision Tree, or K Nearest Neighbor) and use the GridSearchCV object methods to determine the best parameters and accuracy for that predictive analysis method.
5. Create a Confusion Matrix to show the situations in which the model predicts incorrectly (false positives, false negatives)
6. Repeat steps 4 – 5 for remaining analysis object types

Results

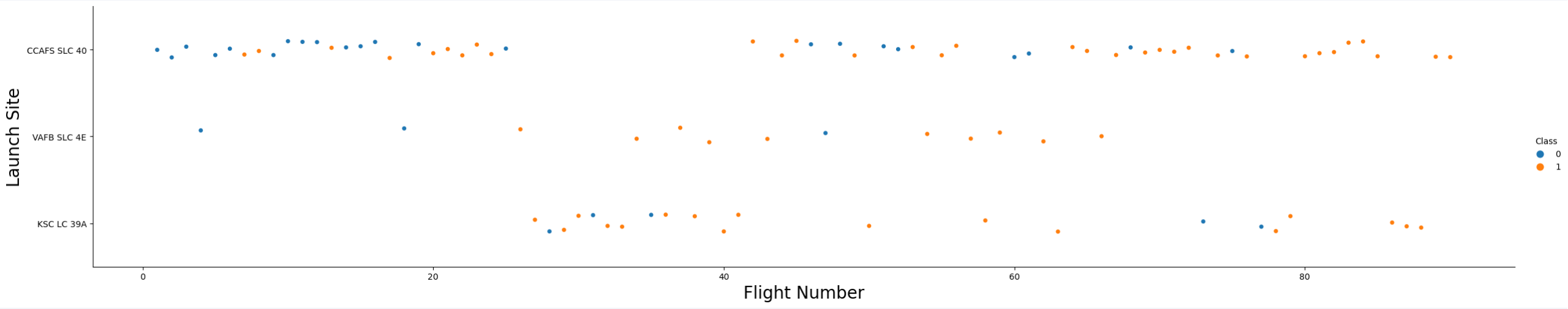
- Exploratory data analysis results
  - Used SQL to explore the data & make a minor update (removed trailing space from a data value)
  - Used various graphing/plotting techniques to visualize the relationships between data elements
- Interactive analytics demo in screenshots
  - Created an interactive dashboard to explore the relationship between launch site, payload weight, booster version, and mission success.
  - Screenshots of dashboard provided in upcoming slides: Successful Launches, By Site, Study of Site with Most Successful Launch Ratio, Which Booster Version Worked Best with Which Payload Weights, and Which Booster Version Worked Best in Specific Payload Weight Range
  - Based on the interactive tools, if I were managing the next launch and being judged on a successful landing, I would hope the next launch would have a 3000 – 4000 kg payload, to be launched from KSC LC-39A, using FT-class boosters. 2 such missions have occured and both have been successful.
- Predictive analysis results
  - Used 4 different predictive models; determined that the Decision Tree model was most effective, though all 4 produced nearly identical results. See Classification Accuracy.
  - The Decision Tree method had an 83% accuracy with the Test Data, and a Confusion Matrix with 15 out of 18 correct predictions (the remaining 3 were false positives). See Confusion Matrix: Decision Tree

Section 2

Insights drawn from EDA

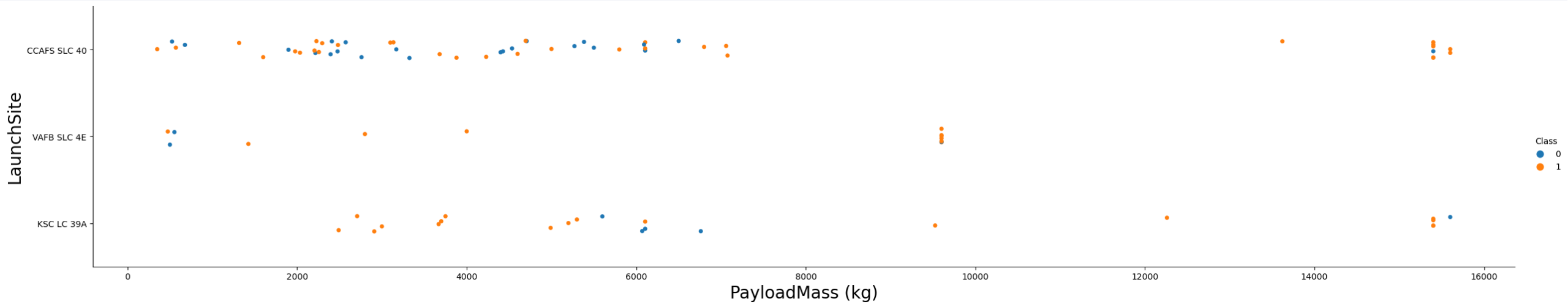


Flight Number vs. Launch Site



- Given that successful landings (”Class 1”) are represented by orange dots and unsuccessful landings (“Class 0”) are represented by blue dots, and given that flights are sequenced chronologically by Flight Number, we see that generally speaking each site has an increased success rate over time.

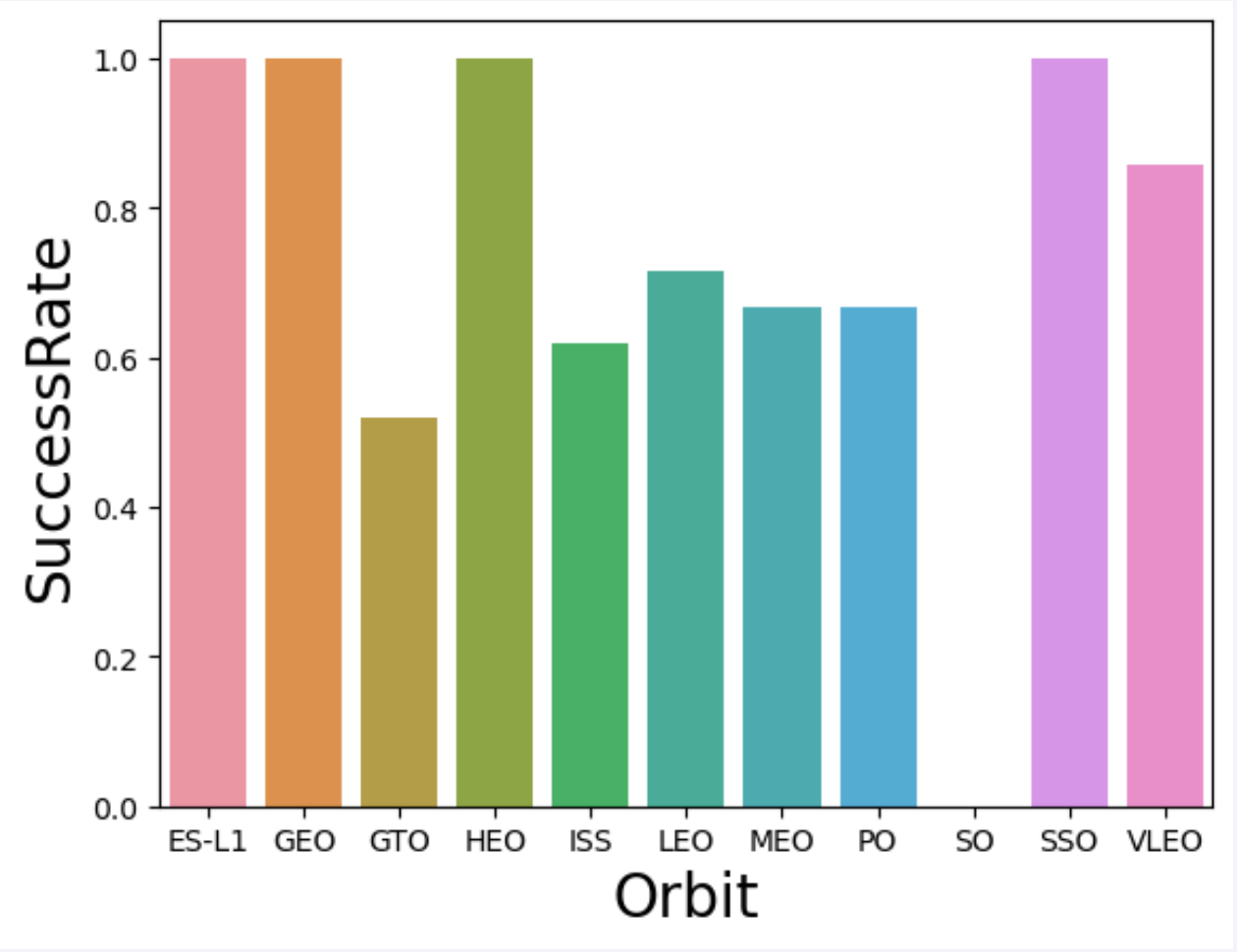
Payload vs. Launch Site



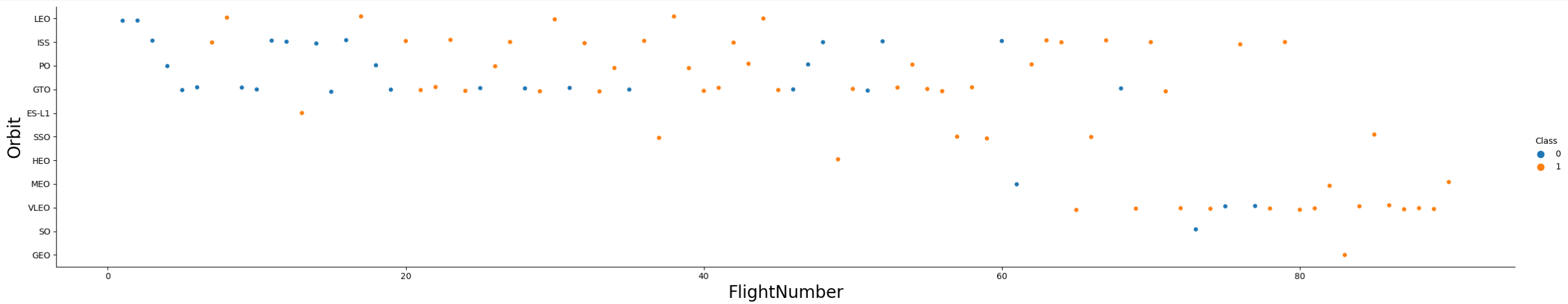
- Given that successful landings (”Class 1”) are represented by orange dots and unsuccessful landings (“Class 0”) are represented by blue dots, we see that the heavier payloads (> ~ 7500kg) have a higher success rate than lighter payloads, and this is true regardless of the launch site involved.
- We can also observe that no “very heavy” (> 10,000kg) launches were attempted from VAFB SLC 4E and no “moderately heavy” (between ~ 7500kg and ~13,000kg) launches were attempted from CCAFS SLC 40. The reason for this is not clear and would be worthy of investigation if it were desired to undertake launches in those weight categories.

Success Rate vs. Orbit Type

- Landings have been very successful (between 80% - 100%) after launching into ES-L1, GEO, HEO, SSO, and VLEO orbits.
- While it appears there have been no successful landings after launching into SO, it is understood that SO and SSO really are the same (Sun-synchronous Orbit). Grouping these two together (a step missed during data wrangling) would have produced a very effective "SO/SSO" group, though slightly less effective than the SSO group is reported to be now.
- Launching into the remaining orbit regions have yielded only moderately successful landings (~50% - ~75% successful)
- Note that launches into ES-L1, GEO, and HEO occurred only 1 time each. Their 100% success rate may be a matter of other factors more than the orbit range involved.

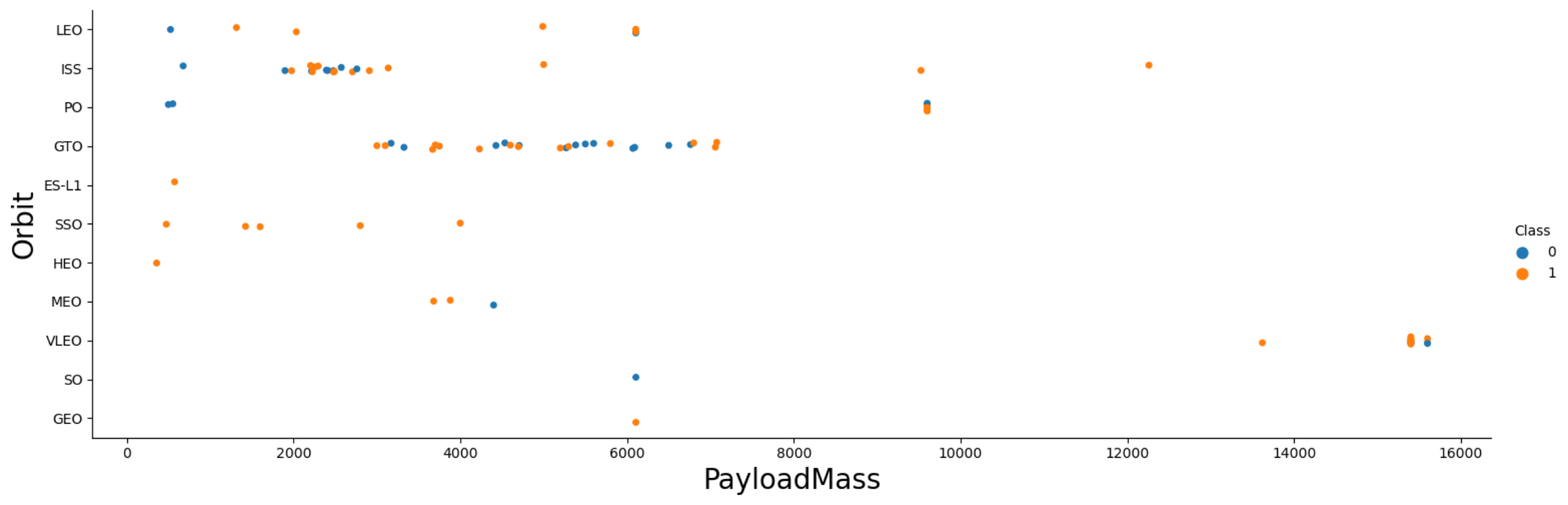


Flight Number vs. Orbit Type



- Given that successful landings ("Class 1") are represented by orange dots and unsuccessful landings ("Class 0") are represented by blue dots, and given that flights are sequenced chronologically by Flight Number, we see that generally speaking each orbit type has an increased success rate over time.
- Potential exceptions are GTO (it's second-most-recent flight resulted in a failed landing), and SO/SSO (when the SO launch is included with SSO, the third-most-recent fight resulted in a failed landing).
- It may be worth noting that certain orbit types weren't attempted until the more recent launches. Was that merely happenstance – was there not a demand for those launches until recently? Were these orbit types considered more challenging and therefore delayed until the team had more experience?

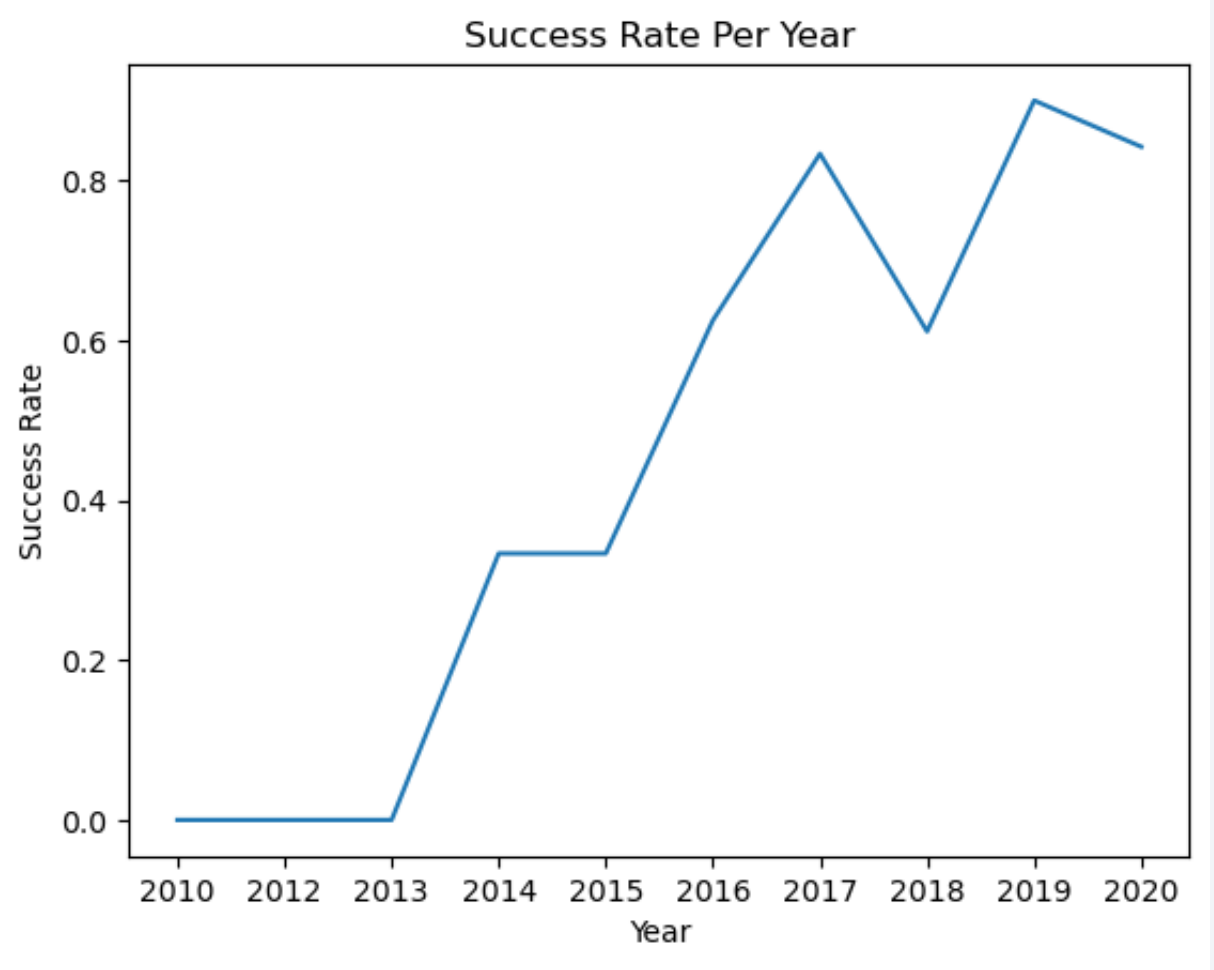
Payload vs. Orbit Type



- Within the GTO orbit type, payload mass doesn't appear to have any bearing on the success or failure of landings
- The heaviest payloads (> 10,000kg) seem to be sent to VLEO and ISS, with largely successful landings.
- Booster landings returning from missions to ISS have mixed results when the payload was < 4,000kg.

Launch Success Yearly Trend

- With the exception of 2018, the success rate has generally increased over time with very little success early in the program and notably higher success rates in more recent years.
- This corroborates the data seen in the "Flight Number vs. Launch Site" slide
- Further investigation would be required to determine why there was a notable dip in 2018 and a slight dip in 2020. (Perhaps those years were when launches were undertaken into the more "challenging" orbits or involved more challenging payloads.)



All Launch Site Names

- Passing a “select distinct” query to the SQLite DB provides a list of the unique launch sites.
- Converting that list to a dataframe makes it easier to read.

[9]:
```
LaunchSiteList_sql = """
SELECT distinct Launch_Site from SPACEXTBL;
"""

cur.execute(LaunchSiteList_sql)

LaunchSiteList = cur.fetchall()
print(LaunchSiteList)

print("\n\nResult shown as DF looks better:\n")
LaunchSiteListDF = pd.DataFrame(LaunchSiteList, columns = ['Launch_Site'])
LaunchSiteListDF.head()
```

```
[('CCAFS LC-40',), ('VAFB SLC-4E',), ('KSC LC-39A',), ('CCAFS SLC-40',)]


Result shown as DF looks better:
```

[9]:

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

Launch Site Names Begin with 'CCA'

- Using a “like ‘CCA%’” clause in a query means that it will search for those rows having a Launch_Site whose name starts with CCA. % is a wildcard meaning “any number of any characters”.
- Using a “LIMIT 5” clause at the end of a query passed to the SQLite DB means that only 5 rows will be returned.
- Converting that list to a dataframe makes it easier to read.

[10]:
```
FiveCCARecords_sql = """
SELECT  * from SPACEXTBL
where Launch_Site like 'CCA%'
LIMIT 5;
"""
result = cur.execute(FiveCCARecords_sql)
result_list = cur.fetchall()
print("results as a list =\n", result_list)

ccadf5 = pd.DataFrame(result_list, columns = ['Date', 'Time', 'Booster_Version', 'Launch_Site', 'Payload', 'PAYLOAD_MASS_KG', 'Orbit', 'Customer', 'Mission_Outcome', 'L
print("\n\nResult shown as DF looks better:\n")
ccadf5.head()
```

```
results as a list =
 [('04-06-2010', '18:45:00', 'F9 v1.0  B0003', 'CCAFS LC-40', 'Dragon Spacecraft Qualification Unit', 0, 'LEO', 'SpaceX', 'Success', 'Failure (parachute)'), ('08-12-20
10', '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', 'Failu
re (parachute)'), ('22-05-2012', '07:44:00', 'F9 v1.0  B0005', 'CCAFS LC-40', 'Dragon demo flight C2', 525, 'LEO (ISS)', 'NASA (COTS)', 'Success', 'No attempt'), ('08-
10-2012', '00:35:00', 'F9 v1.0  B0006', 'CCAFS LC-40', 'SpaceX CRS-1', 500, 'LEO (ISS)', 'NASA (CRS)', 'Success', 'No attempt'), ('01-03-2013', '15:10:00', 'F9 v1.0  B
0007', 'CCAFS LC-40', 'SpaceX CRS-2', 677, 'LEO (ISS)', 'NASA (CRS)', 'Success', 'No attempt')]


Result shown as DF looks better:
```

[10]:

	Date	Time	Booster_Version	Launch_Site	Payload	PAYLOAD_MASS_KG	Orbit	Customer	Mission_Outcome	Landing_Outcome
0	04-06-2010	18:45:00	F9 v1.0 B0003	CCAFS LC-40	Dragon Spacecraft Qualification Unit	0	LEO	SpaceX	Success	Failure (parachute)
1	08-12-2010	15:43:00	F9 v1.0 B0004	CCAFS LC-40	Dragon demo flight C1, two CubeSats, barrel of...	0	LEO (ISS)	NASA (COTS) NRO	Success	Failure (parachute)
2	22-05-2012	07:44:00	F9 v1.0 B0005	CCAFS LC-40	Dragon demo flight C2	525	LEO (ISS)	NASA (COTS)	Success	No attempt
3	08-10-2012	00:35:00	F9 v1.0 B0006	CCAFS LC-40	SpaceX CRS-1	500	LEO (ISS)	NASA (CRS)	Success	No attempt
4	01-03-2013	15:10:00	F9 v1.0 B0007	CCAFS LC-40	SpaceX CRS-2	677	LEO (ISS)	NASA (CRS)	Success	No attempt

Total Payload Mass

- Using the “Sum” and “Groupby” functions in SQL allows us to aggregate values.
- The results of the query are stored in an array, but we know it will be the first entry in that array, so extracting it is a matter of using the [0][0] indices.

```
[11]: CustomerNASA_sql = """
SELECT SUM(PAYLOAD_MASS__KG_) from SPACEXTBL
WHERE Customer = 'NASA (CRS)'
GROUP BY Customer;
"""
NASAResult = cur.execute(CustomerNASA_sql)
NASAMass = cur.fetchall()
print("NASA (CRS) Mass = ", NASAMass[0][0], " Kgs")
```

```
NASA (CRS) Mass =  45596  Kgs
```

Average Payload Mass by F9 v1.1

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

```
[12]: BosterVersionMass_sql = """
      SELECT AVG(PAYLOAD_MASS__KG_) from SPACEXTBL
      WHERE Booster_Version = 'F9 v1.1'
      GROUP BY Booster_Version;
      """
      BoosterVersionMassResult = cur.execute(BosterVersionMass_sql)
      BoosterVersionAvgMass = cur.fetchall()
      print("Booster version F9 v1.1 Avg Mass = ", BoosterVersionAvgMass[0][0], " Kgs")
```

```
Booster version F9 v1.1 Avg Mass =  2928.4  Kgs
```

- Using the “Avg” and “Groupby” functions in SQL allows us to find the average value of a column.
- Including a “Where” clause allows us to only work on a certain booster version.
- The results of the query are stored in an array, but we know it will be the first entry in that array, so extracting it is a matter of using the [0][0] indices.

First Successful Ground Landing Date

```
[15]: FirstGoodDate_sql2 = """
SELECT min((Substr(Date, 7, 4)||Substr(Date, 4, 2)||Substr(Date,1,2)))
from SPACEXTBL
where "Landing _Outcome" = 'Success (ground pad)'
"""

FirstGoodDateResult2 = cur.execute(FirstGoodDate_sql2)
FirstGoodDate2 = cur.fetchall()
print("First date of successfull landing on ground pad = ", FirstGoodDate2[0][0])
```

```
First date of successfull landing on ground pad =  20151222
```

- The primary trick here is using the “Substr” function to parse out Year, Month, and Date from the textual representation as stored in the SQLite database (which doesn’t support dates as a data type), then it’s a matter of using the “Min” function to select the earliest date and a “Where” clause to ensure that we’re only counting dates for successful landings on ground pads.

Successful Drone Ship Landing with Payload between 4000 and 6000

- Passing a “select distinct” query to the SQLite DB provides a list of the unique “Booster_Version”s and using a “where” clause limits the results to only the successful landings on drone ships with payload in the desired range
- Converting that list to a dataframe makes it easier to read.

List the names of the boosters which have success in drone ship and have payload mass greater than 4000 but less than 6

[16]:
```
WhichVersionSQL = """
SELECT DISTINCT Booster_Version from SPACEXTBL
WHERE "Landing _Outcome" = "Success (drone ship)"
AND (PAYLOAD_MASS__KG_ > 4000) AND (PAYLOAD_MASS__KG_ < 6000)
"""

VersionResultList = cur.execute(WhichVersionSQL)
VersionResult = cur.fetchall()
print("Boosters successful with drone ships and payloads between 4000 and 6000 Kgs:\n", VersionResult)

VersionResultDF = pd.DataFrame(VersionResult, columns = ['Booster_Version'])

print("\n\nResult shown as DF looks better:\n")
VersionResultDF.head()
```

```
Boosters successful with drone ships and payloads between 4000 and 6000 Kgs:
 [('F9 FT B1022',), ('F9 FT B1026',), ('F9 FT  B1021.2',), ('F9 FT  B1031.2',)]


Result shown as DF looks better:
```

[16]:

	Booster_Version
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

- Using the “Count” and “Groupby” functions in SQL allows us to count the quantity of occurrence of each value.
- Converting that list to a dataframe makes it easier to read.

```
[17]: SuccessfulMissionRateSQL = """
SELECT Mission_Outcome, COUNT(Mission_Outcome) from SPACEXTBL
GROUP BY Mission_Outcome;
"""

SuccessChartSQL = cur.execute(SuccessfulMissionRateSQL)
SuccessChart = cur.fetchall()
print ("Quantity of successful missions = \n", SuccessChart)

print("\n\nResult shown as DF looks better:\n")
SuccessChartDF = pd.DataFrame(SuccessChart, columns = ['Mission_Outcome', 'Qty'])
SuccessChartDF
```

```
Quantity of successful missions =
 [('Failure (in flight)', 1), ('Success', 99), ('Success (payload status unclear)', 1)]


Result shown as DF looks better:
```

[17]:

	Mission_Outcome	Qty
0	Failure (in flight)	1
1	Success	99
2	Success (payload status unclear)	1

Boosters Carried Maximum Payload

- A subquery is needed here to determine the maximum payload, and then that value is used by the main query to select the “Booster_Version”s that have carried that weight.
- Converting that list to a dataframe makes it easier to read.

```
OverallSQL = """
Select distinct Booster_Version from SPACEXTBL
where PAYLOAD_MASS__KG_ = (SELECT max(PAYLOAD_MASS__KG_) from SPACEXTBL)
"""

OverallSQLCursor = cur.execute(OverallSQL)
BoosterVersionWithHeavyLoads = cur.fetchall()
print("Heavy Booster Versionss = \n", BoosterVersionWithHeavyLoads)

print("\n\nResult shown as DF looks better:\n")
BosterVersionWithHeaverLoadsDF = pd.DataFrame(BoosterVersionWithHeavyLoads, columns = ['Booster_Version'])
BosterVersionWithHeaverLoadsDF
```

```
Heavy Booster Versionss =
 [('F9 B5 B1048.4',), ('F9 B5 B1049.4',), ('F9 B5 B1051.3',), ('F9 B5 B1056.4',), ('F9 B5 B1048.5',), ('F9 B5 B10
51.4',), ('F9 B5 B1049.5',), ('F9 B5 B1060.2 ',), ('F9 B5 B1058.3 ',), ('F9 B5 B1051.6',), ('F9 B5 B1060.3',),
('F9 B5 B1049.7 ',)]


Result shown as DF looks better:
```

	Booster_Version
0	F9 B5 B1048.4
1	F9 B5 B1049.4
2	F9 B5 B1051.3
3	F9 B5 B1056.4
4	F9 B5 B1048.5
5	F9 B5 B1051.4
6	F9 B5 B1049.5
7	F9 B5 B1060.2
8	F9 B5 B1058.3
9	F9 B5 B1051.6
10	F9 B5 B1060.3
11	F9 B5 B1049.7

2015 Launch Records

```
FailedDroneShipLandings2015sql = """
Select Date, substr(Date, 4, 2) as Month, substr(Date, 7,4) as Year, Booster_Version, Launch_Site, "Landing _Outcome" from SPACEXTBL
where "Landing _Outcome" == 'Failure (drone ship)'
AND substr(Date, 7, 4)== '2015';

"""

FailedDroneShipLandingCursor = cur.execute(FailedDroneShipLandings2015sql)
FailedDroneShipLanding2015 = cur.fetchall()

print("Troubled drone ship landings, 2015\n", FailedDroneShipLanding2015)

print("\n\nResult shown as DF looks better:\n")
FailedDroneShipLanding2015DF = pd.DataFrame(FailedDroneShipLanding2015, columns = [ 'Date', 'Month', 'Year', 'Booster_Version', 'Launch_Site', "Landing _Outcome"])
FailedDroneShipLanding2015DF
```

```
Troubled drone ship landings, 2015
 [('10-01-2015', '01', '2015', 'F9 v1.1 B1012', 'CCAFS LC-40', 'Failure (drone ship)'), ('14-04-2015', '04', '2015', 'F9 v1.1 B1015', 'CCAFS LC-40', 'Failure (drone
ship)')]


Result shown as DF looks better:
```

	Date	Month	Year	Booster_Version	Launch_Site	Landing _Outcome
0	10-01-2015	01	2015	F9 v1.1 B1012	CCAFS LC-40	Failure (drone ship)
1	14-04-2015	04	2015	F9 v1.1 B1015	CCAFS LC-40	Failure (drone ship)

- To find the 2015 launches that failed to land on a drone ship, we need to parse the date data (which had been stored as a string) so that we only select the 2015 launches, and also filter so that we're only looking at those that failed to land on a drone ship.
- Converting that list to a dataframe makes it easier to read.

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

- Selecting the records within the appropriate date ranges required complex parsing of the date string. Using “Group by” and “Count” allowed us to count the occurrences of each Landing Outcome. Using “Order by count(*) Desc” meant the list would be presented in largest-to-smallest fashion.
- NOTE: I interpreted the request to mean show ALL of the Landing Outcomes. However, if the instructions intended to mean that we were ONLY interested in “Failure (drone ship)” and “Success (ground pad)”, then an additional clause could be added to the “Where” clause (“… and (Landing_Outcome = “Failure (drone ship)” or Landing_Outcome = “Success (ground pad)”)

[21]:
```
#The other version of this assignment stated "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"
#This makes a little more sense to me as a question to address than the version posed in this version
#of the assignment, so here's the query for that

RankedOutcomesSQL = """
Select "Landing _Outcome", count(*)
From SPACEXTBL
Where ((Substr(Date, 7, 4) > '2010' and Substr(Date, 7,4)< '2017')
or ((Substr(Date,7,4) = '2017' and (Substr(Date, 4, 2) < '03' or (Substr(Date, 4, 2) = '03' and (Substr(Date,1,2) <= 20))))
or ((Substr(Date, 1, 2) = '2010') and (Substr(Date, 4,2) > '06' or (Substr(Date,4,2) = '06' and (Substr(Date,1,2) >='04'))))
)
GROUP BY "Landing _Outcome"
ORDER BY count(*) DESC
"""

RankedOutcomesCursor = cur.execute(RankedOutcomesSQL)
RankedOutcomesList = cur.fetchall()

print("Ranked Landing Types between June 4, 2010 and March 20, 2017 (inclusive)\n", RankedOutcomesList)

print("\n\nResult shown as DF looks better:\n")
RankedOutcomesListDF = pd.DataFrame(RankedOutcomesList, columns = ['Landing _Outcome', 'Qty'])
RankedOutcomesListDF
```

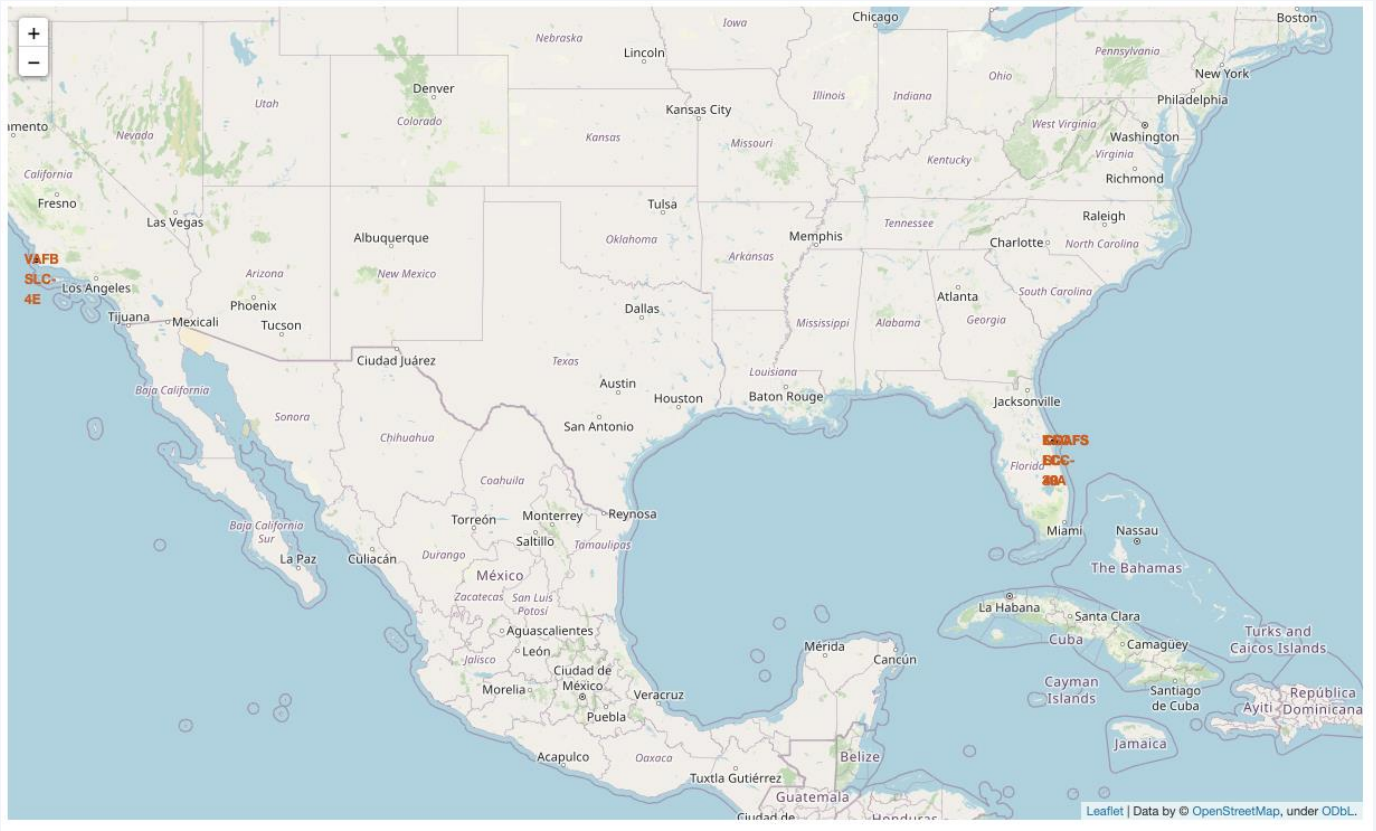
```
Ranked Landing Types between June 4, 2010 and March 20, 2017 (inclusive)
 [('No attempt', 9), ('Success (drone ship)', 5), ('Failure (drone ship)', 5), ('Success (ground pad)', 3), ('Controlled (ocea
n)', 3), ('Uncontrolled (ocean)', 2), ('Precluded (drone ship)', 1)]


Result shown as DF looks better:
```

[21]:

	Landing _Outcome	Qty
0	No attempt	9
1	Success (drone ship)	5
2	Failure (drone ship)	5
3	Success (ground pad)	3
4	Controlled (ocean)	3
5	Uncontrolled (ocean)	2
6	Precluded (drone ship)	1

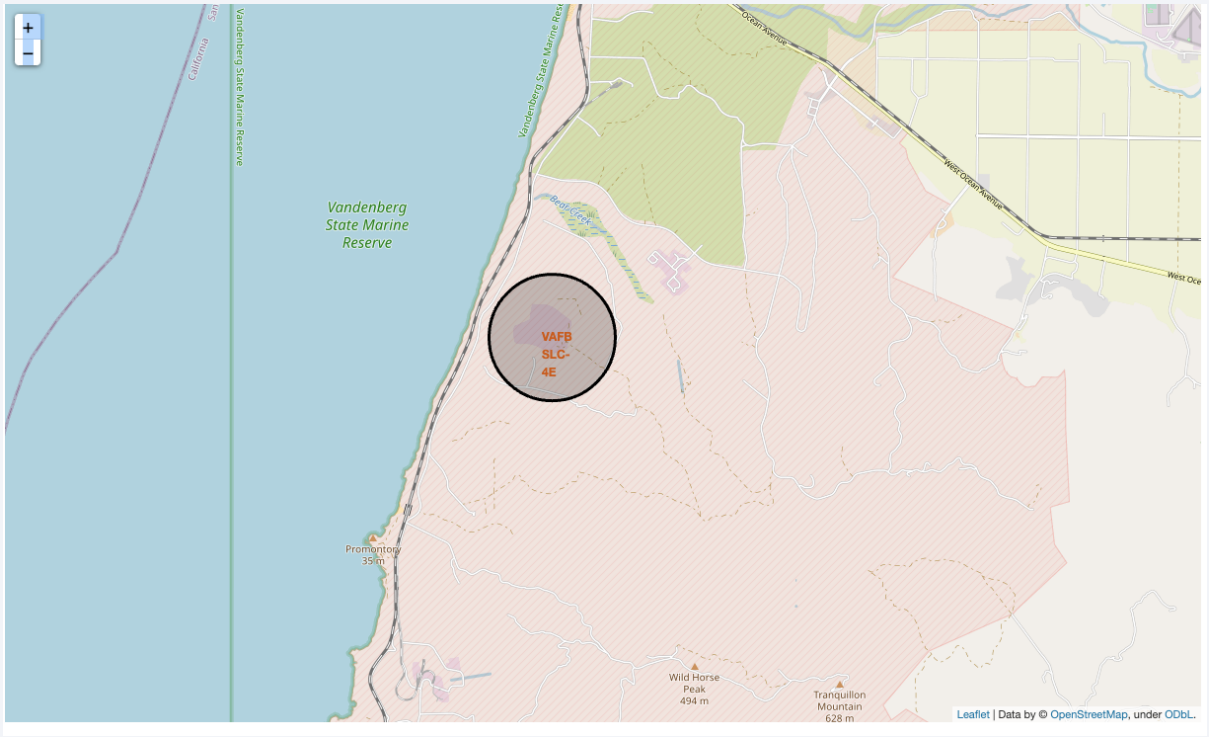

Section 3
Launch Sites
Proximities Analysis

Map of Launch Sites, High Level



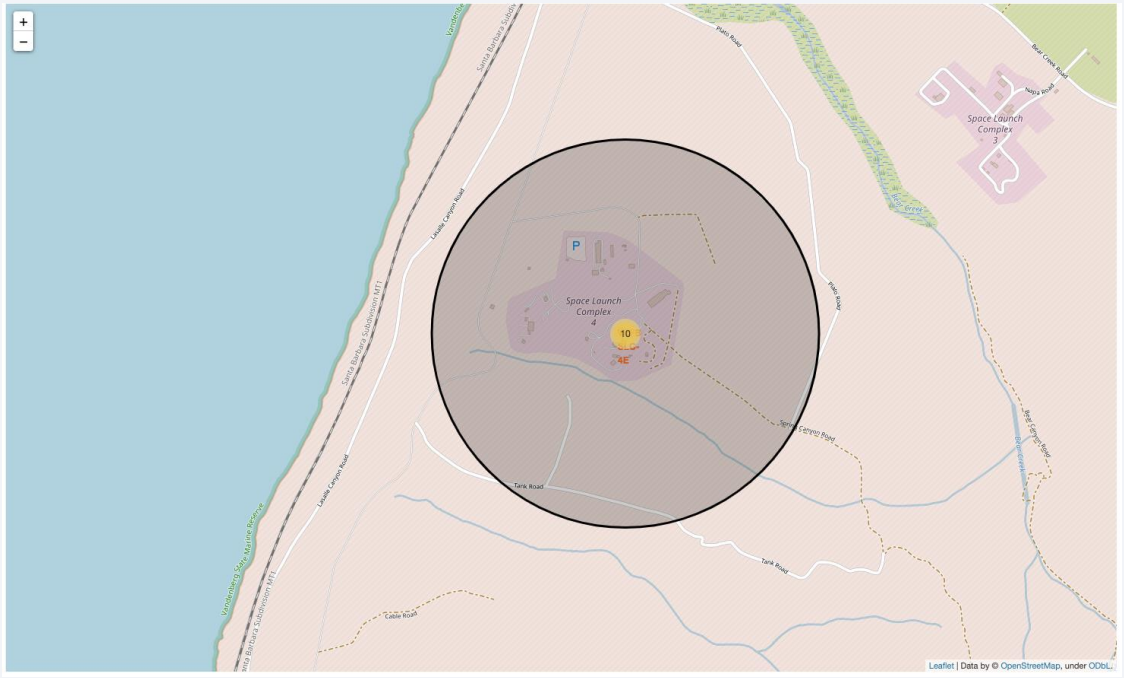
- View of the map showing all 4 launch sites, each labeled in red text
- Note that the 3 sites in Florida are so close to each other that their labels overlap when viewed at this zoom/magnification level

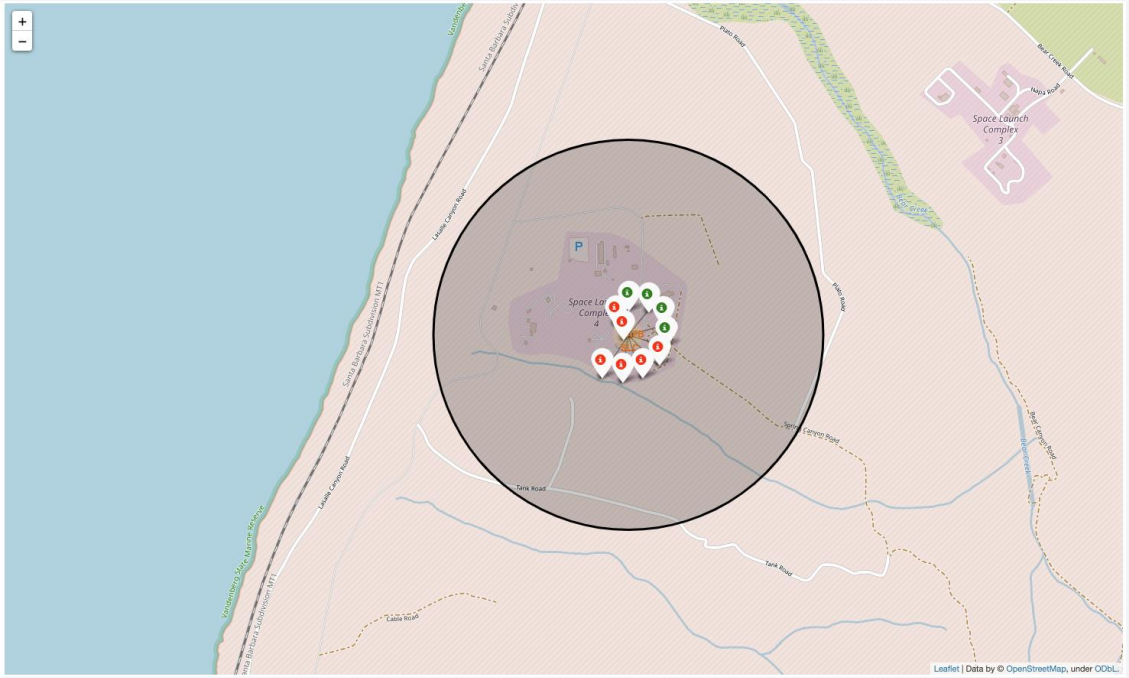
Map of Launch Sites, Magnified



- The California site (VAFB SLC-4E) shown on the left, 3 Florida sites (CCAFS LC-40, CCAFS SLC-40, KSC LC 39A) shown on the right
- Note that even at this magnification level, two of the three sites are so close that their labels overlap.
- It's clear that the launch sites pictured are close to waterways/ocean as well as land transportation (train tracks, roadways). Presumably this is to facilitate bringing large rockets and equipment to/from the launch sites.
- None of the three launch sites are terribly close to metropolitan areas, presumably for safety and possibly security.

Successes/Failures Pictured at Vandenberg

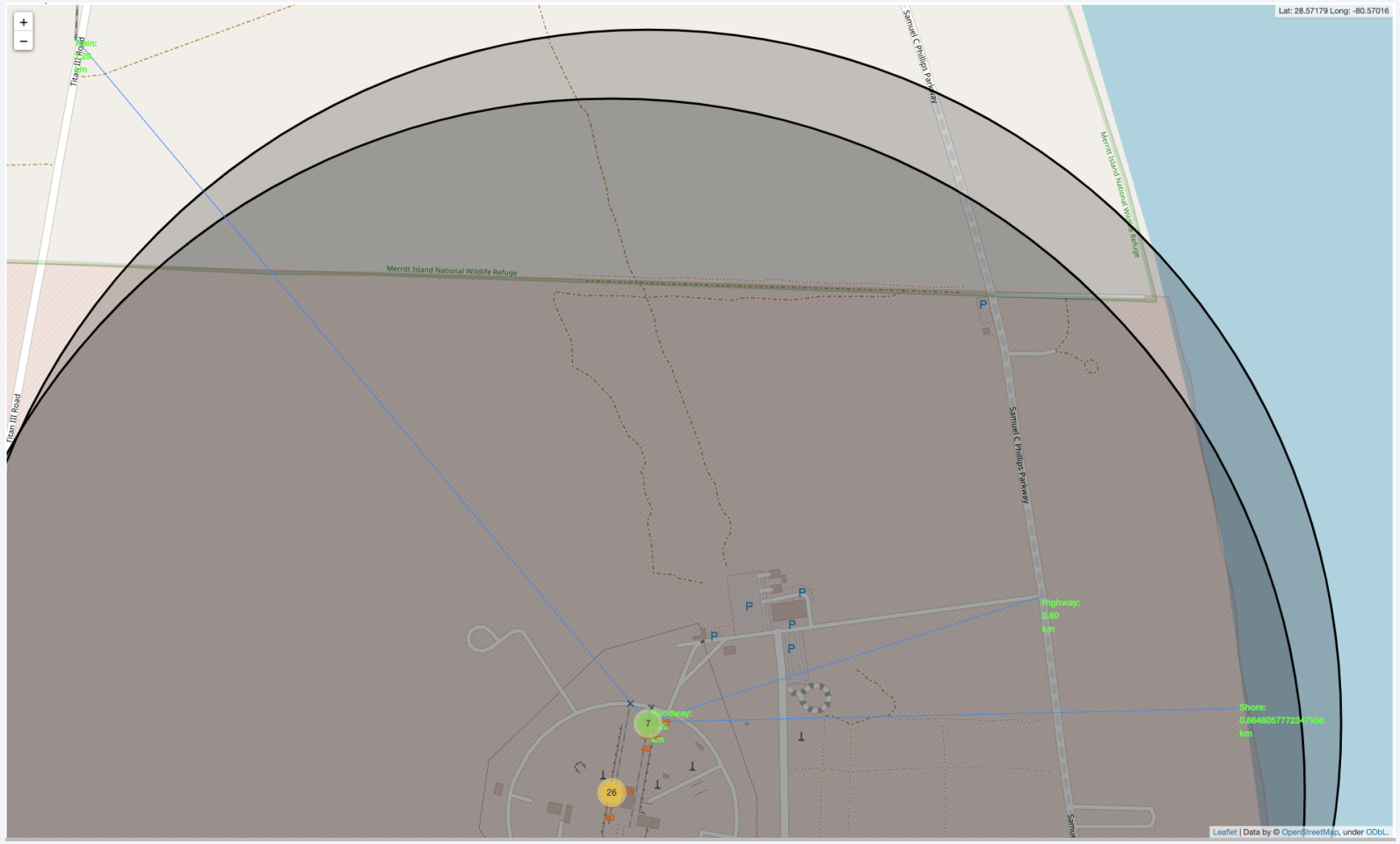


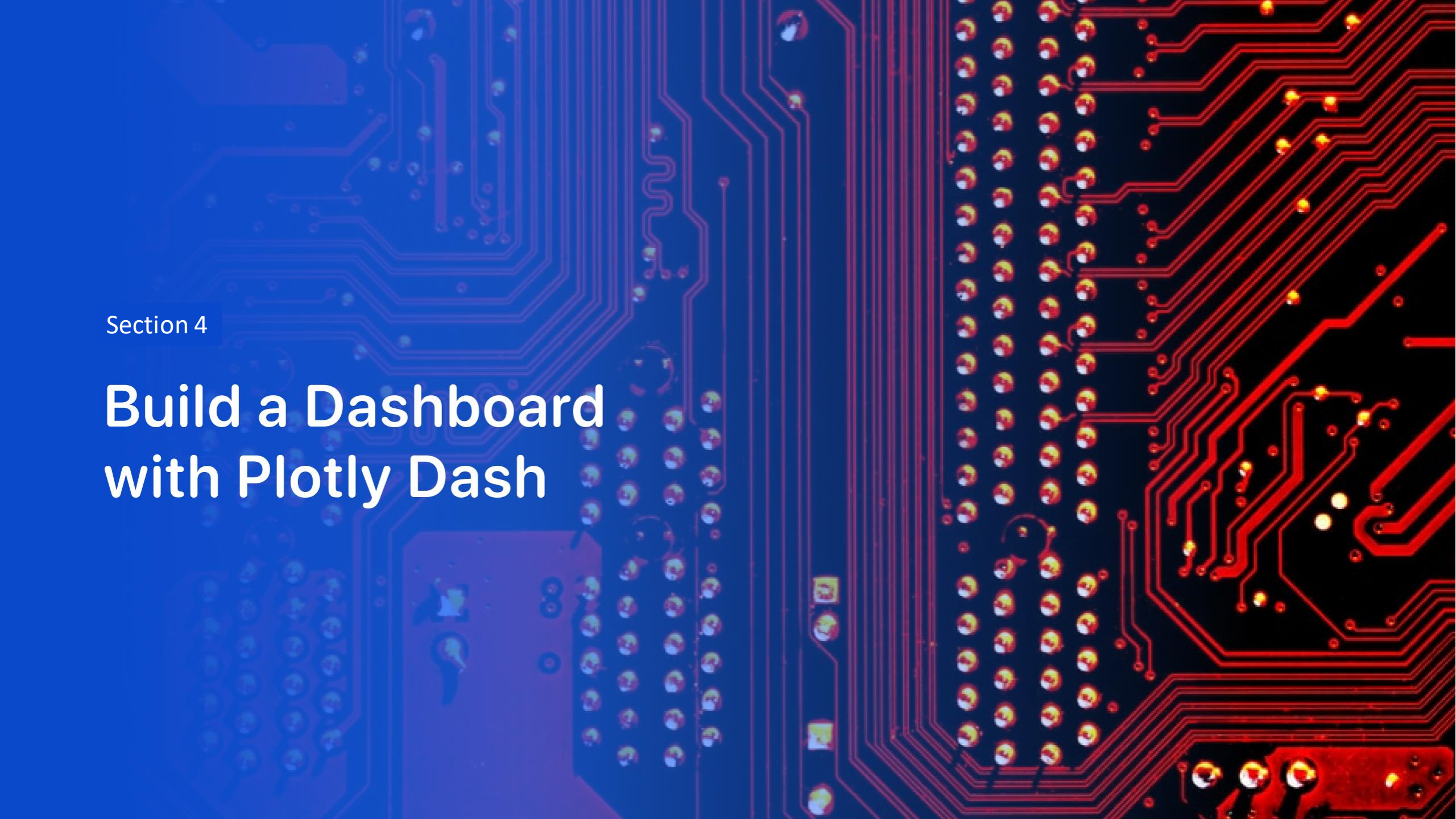


- When zoomed in to the Vandenberg site (VAFB SLC-4E), the total number of launches from that site (10) is displayed on the site marker
- Clicking the site marker reveals color-coded symbols indicating successful (green) launches and failed (red) launches; 4 green and 6 red marks indicate 4 successful and 6 failed launches.
- Observation: this method would get fairly crowded with many launches. To combat that, the lines are drawn at increasing lengths and in a "seashell" pattern (with increasing radius), but still seems likely to get confusing with enough data points.

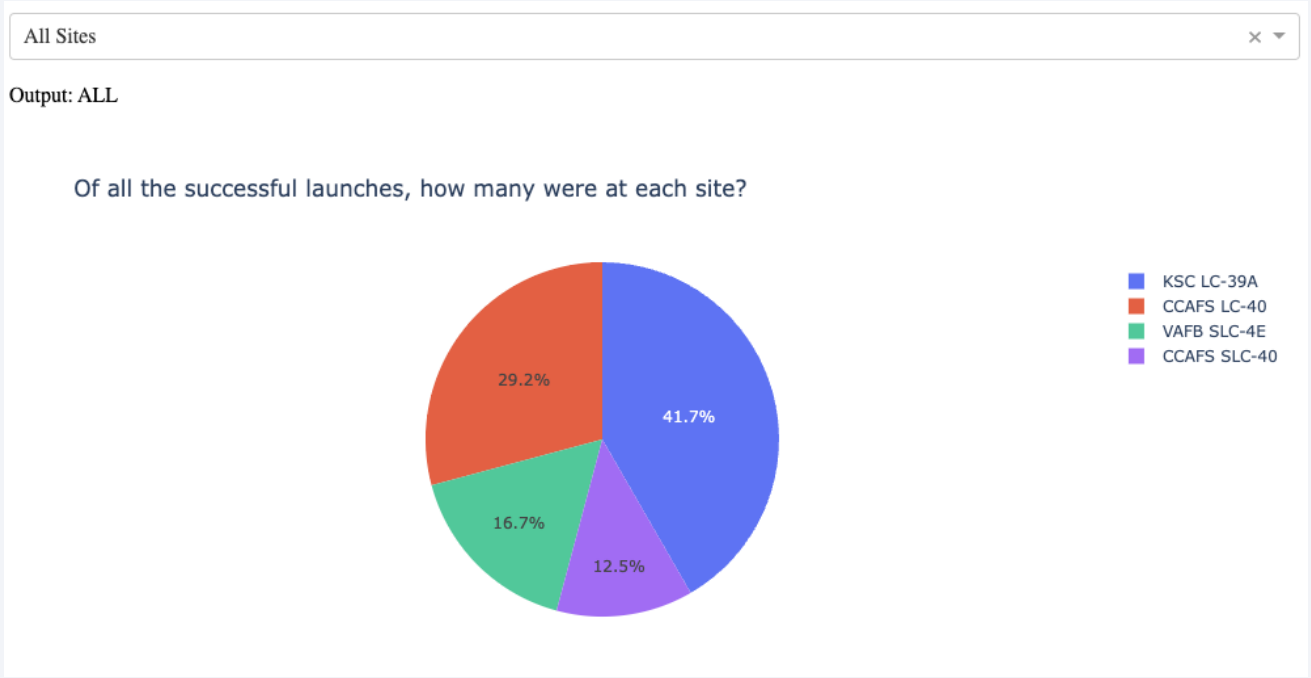
Items Close to CCAFS SLC-40

- Means to transport heavy items to/from the launch area are fairly close
  - Nearest railway is 1.29 km to the Northwest
  - Nearest “Highway” (really, a parkway) is 0.6 km East.
  - Nearest shore is 0.86 km to the East
  - Nearest roadway (circling the launch complex, but connected to other roadways) is 0.02 km to the North



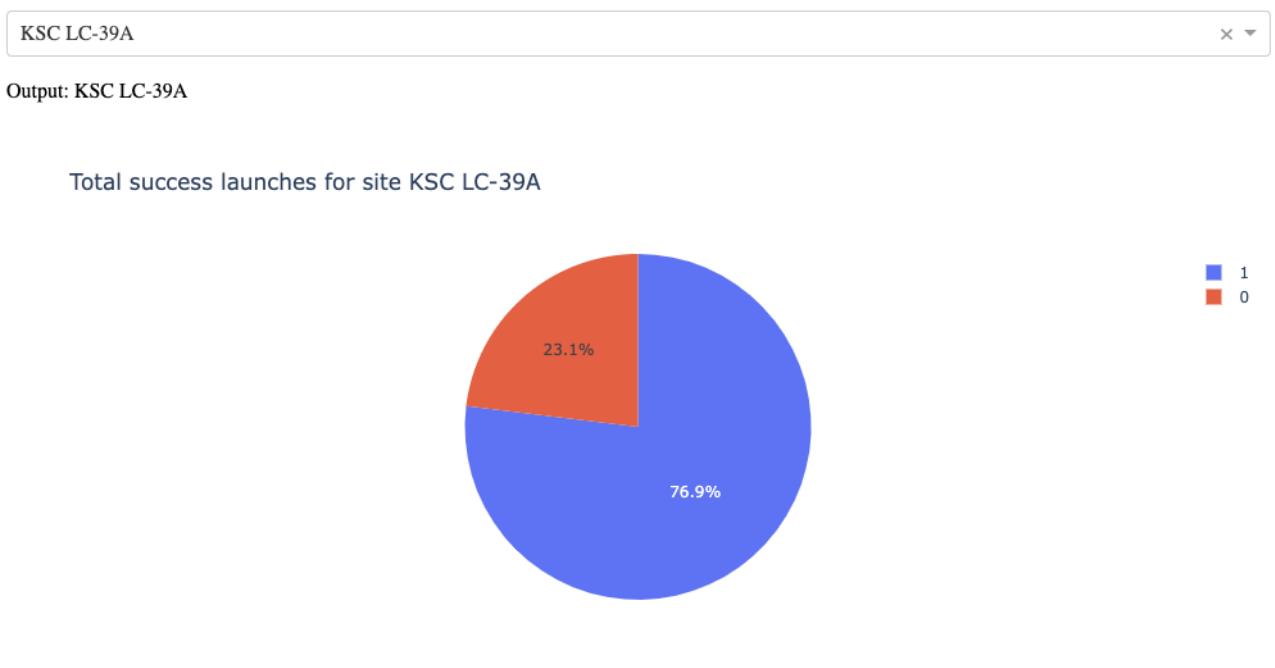

Section 4
Build a Dashboard with Plotly Dash

Dashboard: Successful Launches, By Site



- Based on the menu selection of “All Sites”, the pie chart shows what percent of the successful launches were held at each site.
- KSC LC-39A had the greatest share of successful launches (41.7% of them)
- Side note: the results above might suggest that KSC LC-39A also had the most launches in total. However, CCAFS LC-40 had twice as many launches yet contributed to a smaller percent of successful launches.

Dashboard: Study of Site with Most Successful Launch Ratio

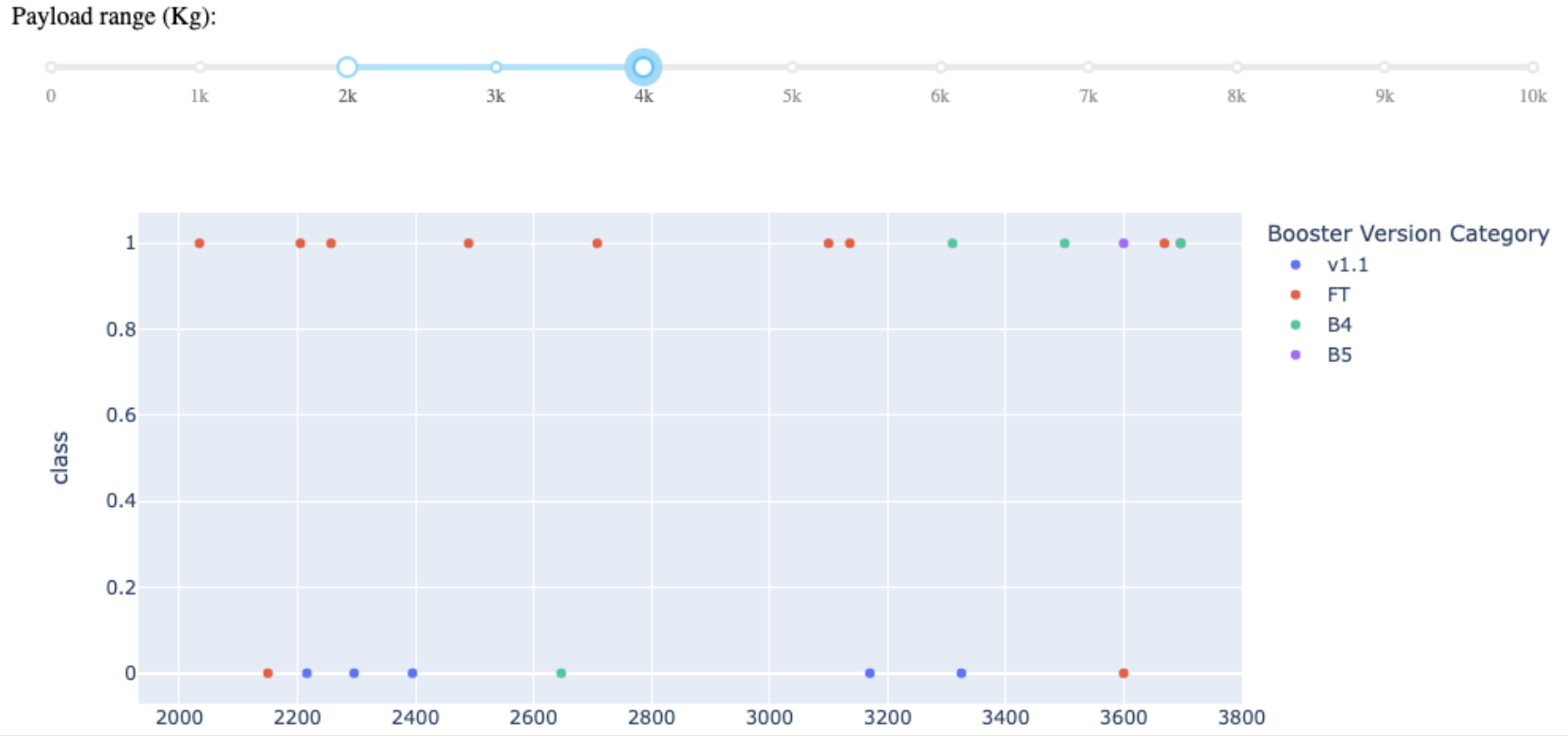


- Based on the menu selection of “KSC LC-39A”, the pie chart shows what percent of launches were successful (class 1) and unsuccessful (class 0) there.
- From the previous slide, we know that KSC LC-39A contributed to 41.7% of the successful launches overall; 76.9% successful launches here were successful

Which Booster Version Worked Best with Which Payload Weights

- This scatter plot shows the successful (class 1) and failed (class 0) launches, color-coded by Booster Version Category, for all payload ranges. (Slider at the top covers the full weight range.)
- A quick look indicates that FT (green dots) had the highest number of successful launches and that v1.1 (red dots) had the highest number of failures overall, and that most of the successful launches had payloads in the 2,000 – 4,000 kg range.

Which Which Booster Version Worked Best in Specific Payload Weight Range



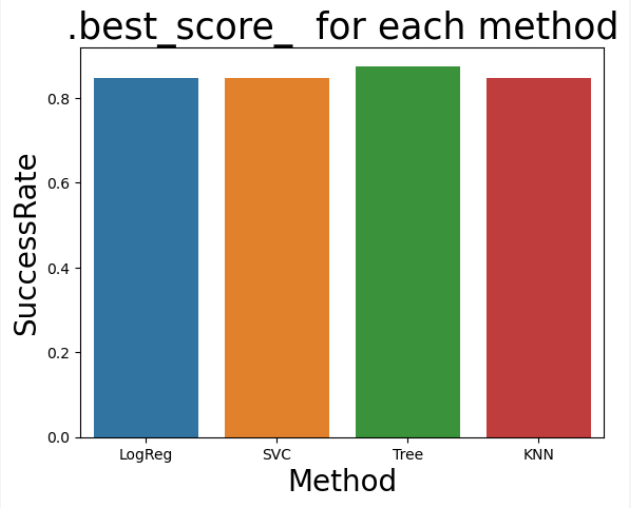
- From the previous chart, the Payload range of 2,000 – 4,000 kg appeared to have the largest grouping of successful launches, so this screenshot narrows our view to just that range for further study. (Slider at the top is constrained to that range.)
- As with the previous chart, a quick look indicates that FT (orange dots this time) had the highest number of successful launches (class 1) and that v1.1 (blue dots) had the highest number of failures (class 0) in this weight range.

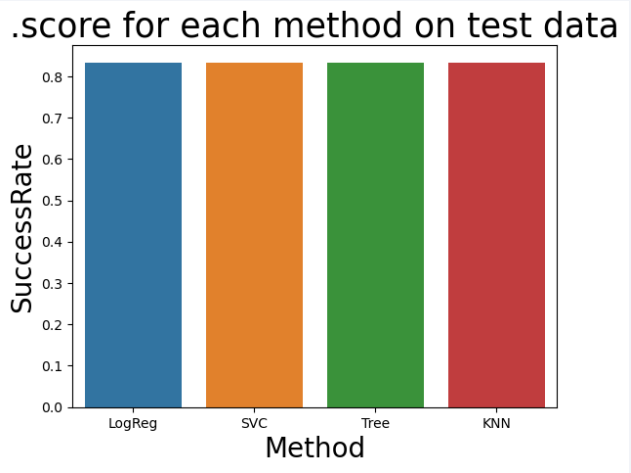
Section 5

Predictive Analysis (Classification)

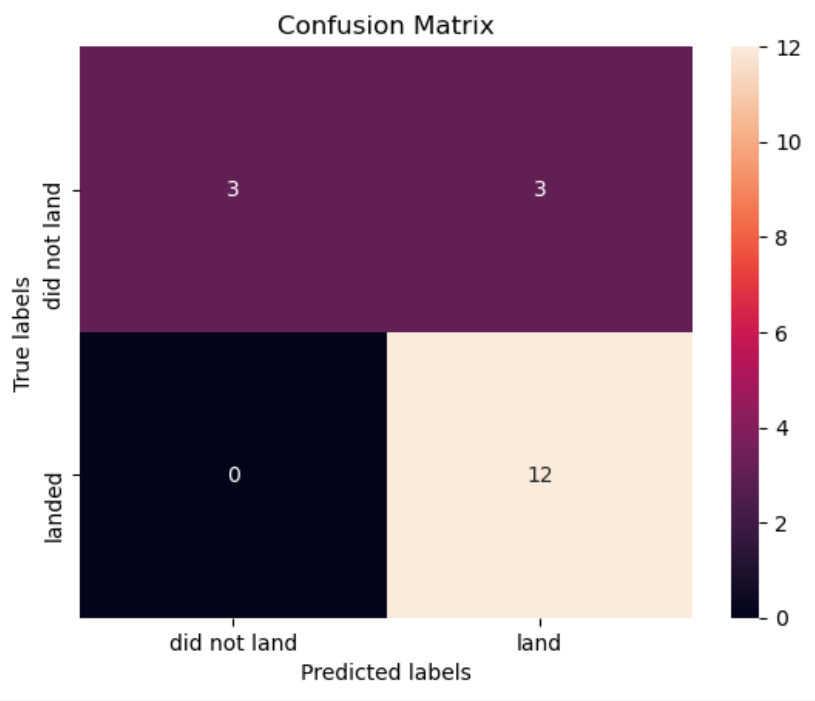
Classification Accuracy

- The accuracy of each of the predictive models, as determined by the ".best_score_" method, indicates that 3 of the 4 approaches (Logistical Regression, Support Vector Machine, and K Nearest Neighbor) all have exactly the same accuracy (0.84722), and the Decision Tree method is only slightly more accurate (0.88889).
- All 4 methods produced exactly the same score when used on the Test data (.83333).
- Given the data we have, the Decision Tree method has the highest classification accuracy. However, the sample size here is relatively small; it would be interesting to see if this comparison remains true over time as more launches are documented.
- Code measuring the accuracy of the models can be found: https://github.com/icgibbs/DataScienceCapstone/blob/74868ce65395a9b0a547ae18456d418fdec167f7/Wk%204,%20Lab%201:%20SpaceX_Machine%20Learning%20Prediction_Part_5.ipynb





Confusion Matrix: Decision Tree



- The confusion matrix for the Decision Tree model indicates that of the 18 missions in the test data set:
  - The model correctly predicted landings 12 times (true positives) and non-landings 3 times (true negatives).
  - However, it also incorrectly predicted 3 landings that were actually non-landings (false positive). Fortunately, it didn't predict any non-landings that were actually landings (false negatives).
- It may be of interest to note that all 4 of the prediction models had exactly the same confusion matrix.

Conclusions

- Generally, the likelihood of being able to recover a booster increases with time and experience.
- If other factors (costs of personnel, getting equipment to/from launch site, fuel, etc.) were constant, the least expensive launch (or the most profitable) would be one which:
  - Launched from KSC LC-39A
  - Used an FT-class booster
  - Had a payload weighing between 2,000 – 4,000 kg
- Similarly, the most expensive (least profitable) would likely be one which:
  - Launched from CCAFS LC-40
  - Used a v1.1-class booster
  - Had a payload weighing between 6,000 – 8,000 kg
- All the prediction models (including the selected Decision Tree model) yielded 3 “false positives” in a field of 18 launches worth of test data – or ~ 17%.

Appendix 1: Additional Thoughts for Future Research

- Is weather a factor in the success of a mission? (Can we get the temperature, humidity, and wind speed/direction at the time of each of these launches and landings?)
- Does the intended landing surface impact the success of a mission – is it easier to land on land vs. on a drone ship at sea? How does that impact the cost-effectiveness of the mission (are drone ships more expensive to build/maintain than landing surfaces on land)?
- Does the day-of-week impact the success of a mission?
- Assuming that we don't use the same crew for every single launch, is the experience level of the crew conducting the launch relevant to the success of the mission? (Can we get information about the number of launches – or # of similar launches – that the team lead had been involved in prior to each launch?)
- Expand the study – do rockets other than the Falcon 9 have a better/worse track record? Could it be that other rockets are more cost effective? (Are Falcon 9 rockets the only one with re-usable boosters, and thus the only ones relevant for discussion?)
- How many of the successful landings had been rockets that had been used before? Do "recycled" rockets have a better/worse track record than "new" rockets?
- Do the recovered booster rockets need to be "refurbished"? At what cost? How many times can a rocket be re-used before it's no longer cost-effective?
- Do the booster rockets always "detach" at the same altitude? If not, does the height at which the booster detached have an impact on the success of the landing? Similarly, does the distance the booster was attached have an impact? (Distance would be different than altitude if the rockets were launched at slightly different angles.)
- Right after collecting data via the REST API, "PayloadMass" was found to be empty for 5 entries. We "corrected" that by replacing the null values with the average value for all the data in our dataset. How would the accuracy of our predictions change if we have been more "specific" in selecting an average? Perhaps only using the average weight of the other launches from the same launch site? Or average weight of the other launches using the same rocket type? Or going into the same orbit area?
- Would our prediction models be more accurate with more data? Can we get data about more launches? (Would the prediction models' "confusion matrices" show fewer false positives/false negatives if we had more data?)

Appendix 2: Thoughts on Interactive Dashboard

- The original Interactive Dashboard included only 1 pie chart (with a drop-down menu to filter based on launch site) and 1 scatterplot (with a slider to filter based on payload weight, and also made use of the pie chart's menu to filter).
  - To explore the effectiveness of different booster categories, an additional pie chart (with a drop-down menu to filter based on booster category) was added.
  - The drop-down menu from this new pie chart was also used as a (third) input for the scatterplot. Thus, one can now view scatter points based on the combination of a specified launch site (or all sites), a specified booster category (or all of them), and a specified payload weight range.
  - Each of the two pie charts still use only their own drop-down menus as inputs. For future study, it might be interesting to include all three input mechanisms for the pie charts.
- Screenshot of this Expanded Dashboard: https://github.com/icgibbs/DataScienceCapstone/blob/d0d78580399d32a5e795df098c53988d1967e348/ExpandedDashboard.png

Appendix 3: Improved Readability of SQL Results

- The results of SQL “select” statements are returned as list of tuples. These are sometimes a little challenging to read - especially when there are multiple data points (columns) in the results.
- Converting/reformatting these results as Dataframes made them notably easier to read.

```
SuccessfulMissionRateSQL = """
SELECT Mission_Outcome, COUNT(Mission_Outcome) from SPACEXTBL
GROUP BY Mission_Outcome;
"""

SuccessChartSQL = cur.execute(SuccessfulMissionRateSQL)
SuccessChart = cur.fetchall()
print ("Quantity of successful missions = \n", SuccessChart)

print("\n\nResult shown as DF looks better:\n")
SuccessChartDF = pd.DataFrame(SuccessChart, columns = ['Mission_Outcome', 'Qty'])
SuccessChartDF
```

Code to create/execute SQL query

Convert result to dataframe

```
Quantity of successful missions =
 [('Failure (in flight)', 1), ('Success', 99), ('Success (payload status unclear)', 1)]


Result shown as DF looks better:
```

Hard to read list

	Mission_Outcome	Qty
0	Failure (in flight)	1
1	Success	99
2	Success (payload status unclear)	1

Easy to read dataframe

Appendix 4: Links to Python Notebooks and Code

- General GitHub repository: https://github.com/icgibbs/DataScienceCapstone/tree/master
- Collecting Data via an API: Notebook
- Collecting Data via Web Scraping: Notebook
- Data Wrangling: Notebook
- EDA using SQL: Notebook
- EDA via Graphing/Plotting: Notebook
- Geographic Analysis via Folium Maps: Notebook
- Interactive Dashboard: Python code/Dash app
  - Screenshot, showing extra Pie Chart: Image
- Machine Learning Prediction: Notebook

Thank you!