IBM Developer
SKILLS NETWORK

Winning Space Race with Data Science

Saad Joiya
30th July, 2022

Outline

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

Executive Summary

- Summary of methodologies
  - Data Collection
  - Data Wrangling
  - EDA with Visualization
  - Building interactive dashboards with Plotly Dash
  - Building interactive maps with Folium
  - Predictive analysis using Classification algorithms

- Summary of all results
  - EDA results
  - Interactive Dashboarding results
  - Predictive Analytics results

Introduction

- Project background and context
  - SpaceX advertises significantly lower costs of rocket launches compared to its competitors(62 Million $ vs 165 Million $) owing to high success rate of safe landings. It can reuse the first stage to compensate for costs.
- Problems you want to find answers
  - Predicting the probability of safe landing of SpaceX rocket launches and analyzing probable factors behind failures.

Section 1

Methodology

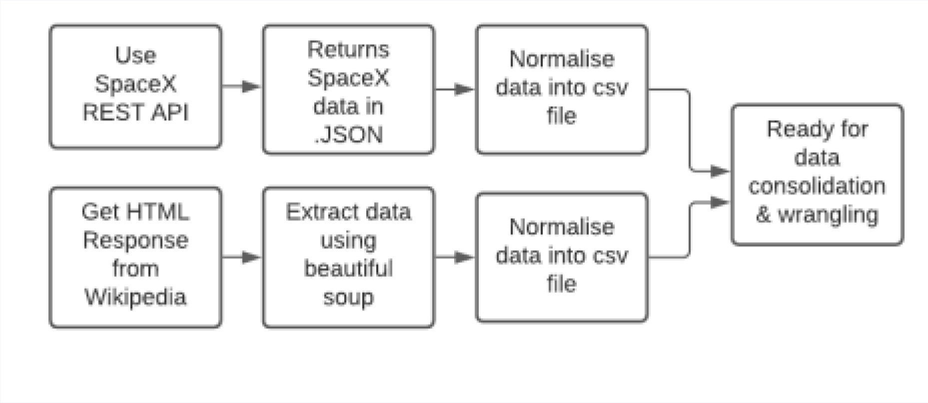
Methodology

Executive Summary

- Data collection methodology:
  - The data was collected majorly from SpaceX Rest API and Web Scrapping from Wikipedia.
- Perform data wrangling
  - Data was encoded using one hot encoding and null values were replaced. Irrelevant columns were also excluded.
- Perform exploratory data analysis (EDA) using visualization and SQL
- Perform interactive visual analytics using Folium and Plotly Dash
- Perform predictive analysis using classification models
  - Different classification models using K Nearest Neighbours, SVM, Logistic Regression and Decision Trees were built and evaluated for best classifier.

Data Collection

- Datasets were collected using:
  - SpaceX launch data that is gathered from the SpaceX REST API, giving information around the launches, rockets used, landing specifications, payloads and outcomes.
  - Another popular data source for obtaining Falcon 9 Launch data through web scraping its Wikipedia page using BeautifulSoup.



Data Collection – SpaceX API

```
[9]: static_json_url='https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/datasets/API_call_spacex_api.json'
```

We should see that the request was successfull with the 200 status response code

```
[10]: response.status_code
[10]: 200
```

Now we decode the response content as a Json using `.json()` and turn it into a Pandas dataframe using `.json_normalize()`

```
[11]: # Use json_normalize meethod to convert the json result into a dataframe
response = requests.get(static_json_url).json()
data = pd.json_normalize(response)
```

Using the dataframe `data` print the first 5 rows

```
[12]: # Get the head of the dataframe
data.head(5)
```

Requesting data from Static URL and converting response into data using json.normalize()

```
[13]: # Lets take a subset of our dataframe keeping only the features we want and the flight number, and date_utc.
data = data[['rocket', 'payloads', 'launchpad', 'cores', 'flight_number', 'date_utc']]

# We will remove rows with multiple cores because those are falcon rockets with 2 extra rocket boosters and rows that have multiple payloads in a single rocket.
data = data[data['cores'].map(len)==1]
data = data[data['payloads'].map(len)==1]

# Since payloads and cores are lists of size 1 we will also extract the single value in the list and replace the feature.
data['cores'] = data['cores'].map(lambda x : x[0])
data['payloads'] = data['payloads'].map(lambda x : x[0])

# We also want to convert the date_utc to a datetime datatype and then extracting the date leaving the time
data['date'] = pd.to_datetime(data['date_utc']).dt.date

# Using the date we will restrict the dates of the launches
data = data[data['date'] <= datetime.date(2020, 11, 13)]
```

Data Cleaning to get rid of extra features, duplicating rows and for changing data types

```
[21]: launch_dict = {'FlightNumber': list(data['flight_number']),
'Date': list(data['date']),
'BoosterVersion':BoosterVersion,
'PayloadMass':PayloadMass,
'Orbit':Orbit,
'LaunchSite':LaunchSite,
'Outcome':Outcome,
'Flights':Flights,
'GridFins':GridFins,
'Reused':Reused,
'Legs':Legs,
'LandingPad':LandingPad,
'Block':Block,
'ReusedCount':ReusedCount,
'Serial':Serial,
'Longitude': Longitude,
'Latitude': Latitude}
```

Then, we need to create a Pandas data frame from the dictionary launch_dict.

```
[22]: # Create a data from launch_dict
df = pd.DataFrame.from_dict(launch_dict)
```

Creating a dictionary with keys as relevant feature and values as their data obtained from functions. Converting this dict into df .

```
[26]: # Hint data['BoosterVersion']!='Falcon 1'
data_falcon9 = df.loc[df['BoosterVersion']!= "Falcon 1"]
```

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

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

Filtering Data to get just Falcon 9 launch data and reindexing df

```
29]: # Calculate the mean value of PayloadMass column
mean_value=data_falcon9['PayloadMass'].mean()
data_falcon9['PayloadMass'].fillna(value=mean_value, inplace=True)

# Replace the np.nan values with its mean value
```

Final Data Wrangling to fill NaNs and missing values with mean

- https://github.com/saadwali/testrepo/blob/main/jupyter-labs-spacex-data-collection-api%20(1).ipynb

Data Collection - Scraping

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

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

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

Getting response from HTML and creating BeautifulSoup object

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

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

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

Finding tables in HTML

```
[11]: column_names = []

# Apply find_all() function with `th` element on first_launch_table
# Iterate each th element and apply the provided extract_column_from_header() to get a column name
# Append the Non-empty column name (`if name is not None and len(name) > 0`) into a list called column_names

temp = soup.find_all('th')

for x in range(len(temp)):
    try:
        name = extract_column_from_header(temp[x])
        if (name is not None and len(name) > 0):
            column_names.append(name)
    except:
        pass
```

Extracting all column names that are present in the HTML table

```
[15]: 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']=[]
```

```
#If you run this more than once you will continualy append data to the dictinoary.  Run the prior cel

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
```

Appending data using column names into dictionary

```
[17]: df=pd.DataFrame(launch_dict)
```

We can now export it to a CSV for the next section, but to make

Following labs will be using a provided dataset to make each lat

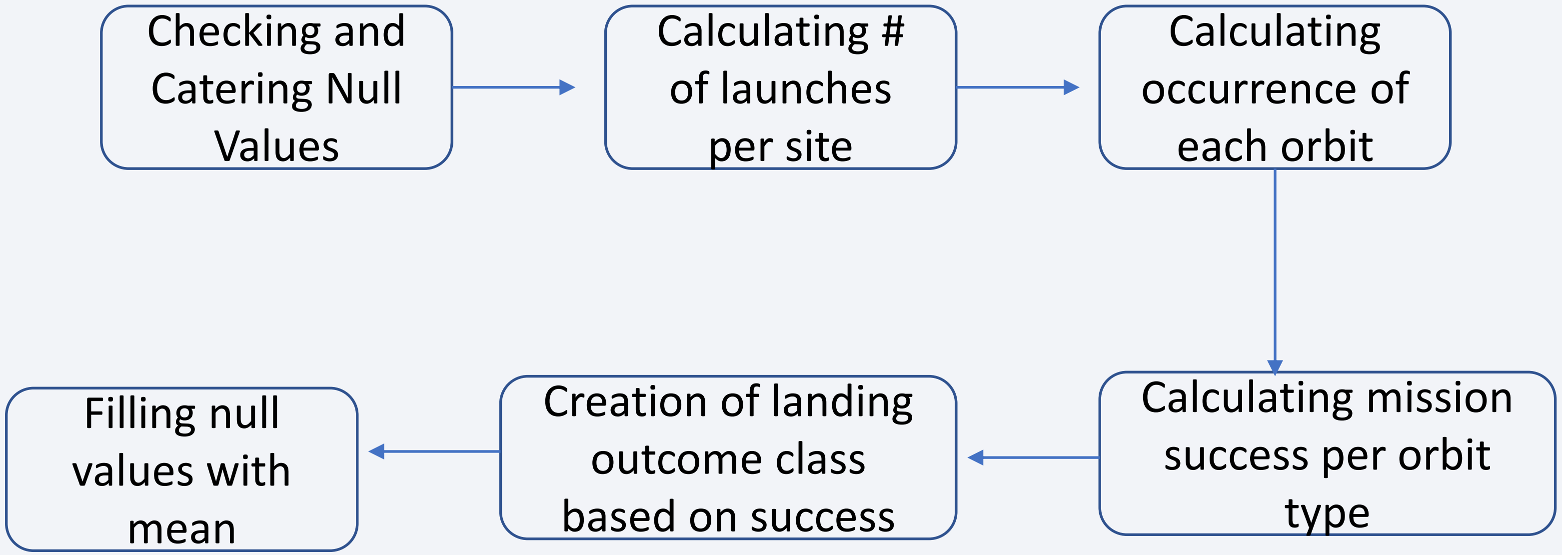
`df.to_csv('spacex_web_scraped.csv', index=False)`

```
[18]: df.to_csv('spacex_web_scraped.csv', index=False)
```

Converting Dictionary to Dataframe and saving CSV.

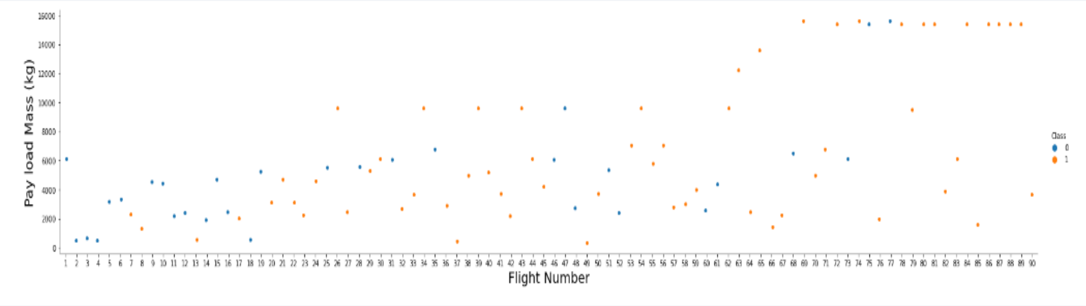
- https://github.com/saadwali/testrepo/blob/main/jupyter-labs-webscraping.ipynb

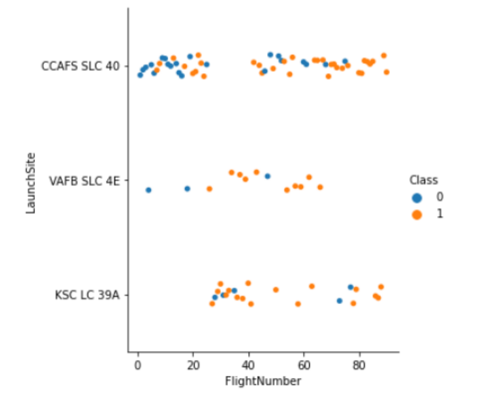
Data Wrangling

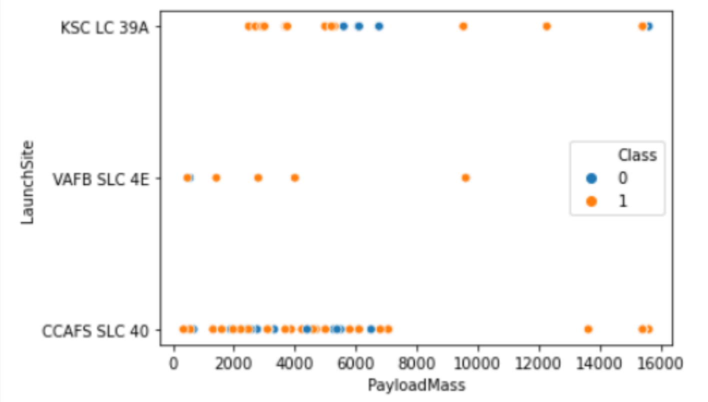


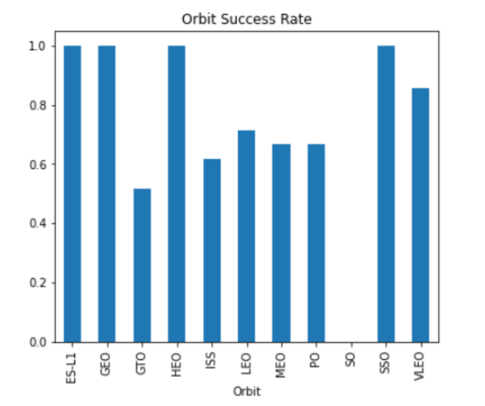
- https://github.com/saadwali/testrepo/blob/main/labs-jupyter-spacex-Data%20wrangling.ipynb

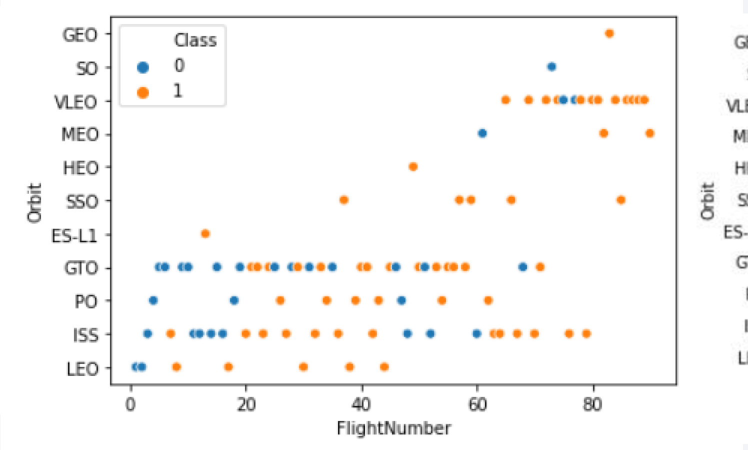
EDA with Data Visualization

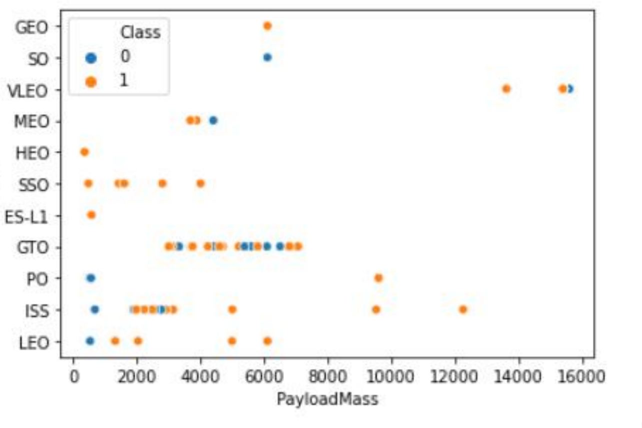


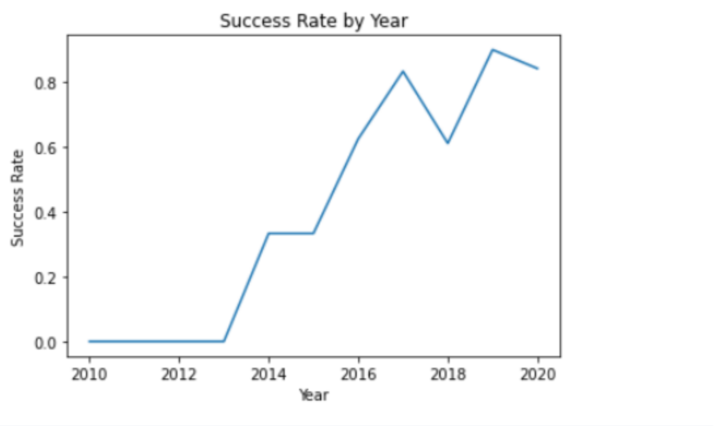












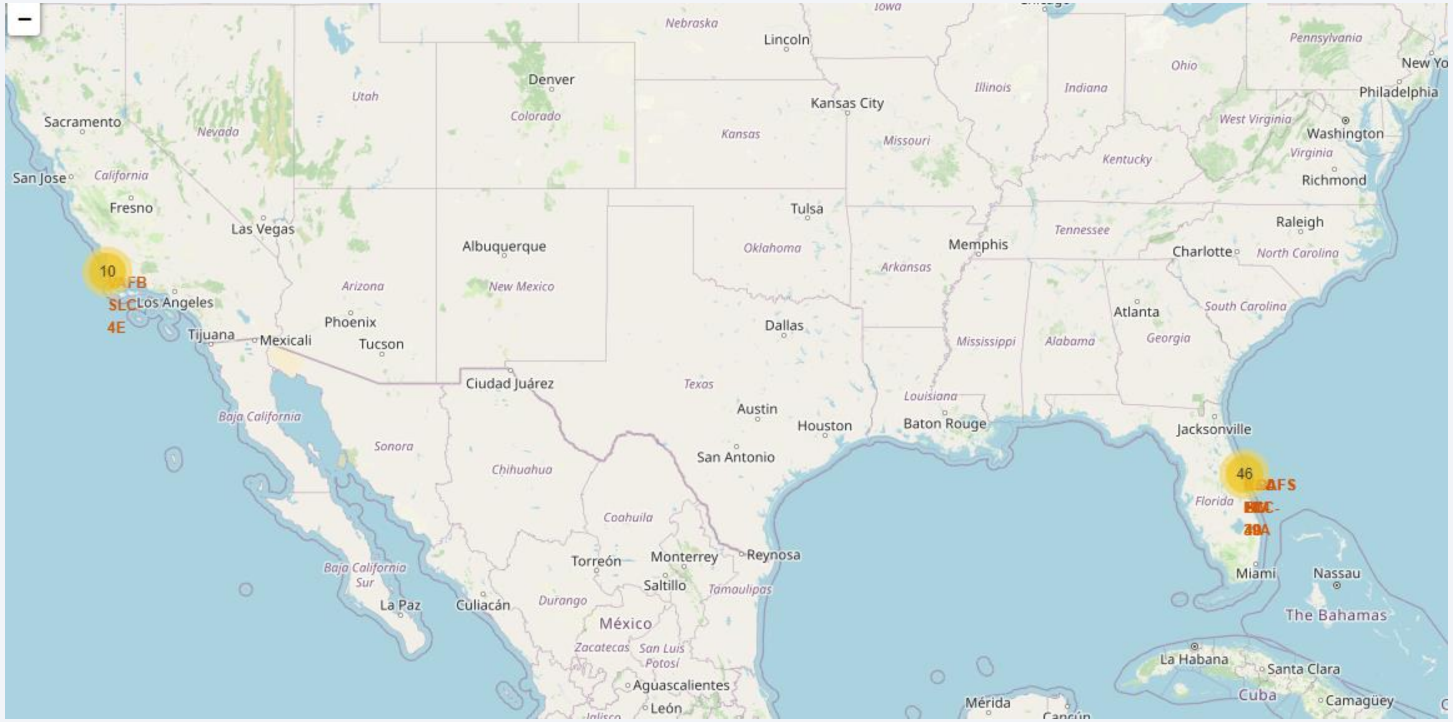
https://github.com/saadwali/testrepo/blob/main/jupyter-labs-eda-dataviz.ipynb

EDA with SQL

- SQL Queries Performed:
  - Display name of unique launch sites in SpaceX Mission
  - Display 5 records where launch site name begins with 'CCA'
  - Display total payload mass carried by NASA (CRS) boosters
  - Display Average payload mass carried by F9 v1.1
  - List the first successful landing date
  - List names of boosters having success in drone ship landing and carrying payload mass greater than 4000 and less than 6000kg
  - List total number of successful and failure mission outcomes
  - List Boosters with maximum payload mass carried successfully
  - List failure landing outcomes on drone ship by month in 2015
  - Rank count of successful landing outcomes between 2016-06-04 and 2017-03-20
- https://github.com/saadwali/testrepo/blob/main/jupyter-labs-eda-sql-coursera_sqllite.ipynb

Build an Interactive Map with Folium

• Different elements were added to the map for optimal launch site analysis such:



- folium.Marker() was used to create marks on the maps.
- folium.Circle() was used to create a circles above markers on the map.
- folium.Icon() was used to create an icon on the map.
- folium.PolyLine() was used to create polynomial line between the points.
- folium.plugins.AntPath() was used to create animated line between the points.
- markerCluster() was used to simplify the maps which contain several markers with identical coordination.

https://github.com/saadwali/testrepo/blob/main/lab_jupyter_launch_site_location.ipynb

Build a Dashboard with Plotly Dash

- Different functions were performed using different elements such as:
  - ❖ Pandas was used to simply work frame and creating a dataframe object.
  - ❖ Plotly was used to plot graphs and different graphs and charts such as pie chart and scatter chart were used
  - ❖ Interactive graphs and tables were made possible using dropdowns which leveraged different dash and html components.
  - ❖ Dropdowns were then used to filter launch sites.
  - ❖ A range slider was also added to select different payload mass ranges.

https://github.com/saadwali/testrepo/blob/main/dash_interactivity.py

Predictive Analysis (Classification)

Preparing Data for Modelling:

- Creating Dependent Variable (Class)
- Standardizing the columns
- Splitting Data into Train and Test

Model Building:

- Creating 4 different models including KNN, Decision Trees, SVM and Logistic Regression.
- Building GridCV function for optimal parameters.

Model Evaluation:

- Fitting the 4 models on to the training data
- Calculating accuracies on test data and Confusion Matrices

Finding the Optimal Model :

- Test all 4 models on the test data with best hyperparameters
- Model with highest accuracy is the best model.

https://github.com/saadwali/testrepo/blob/main/SpaceX_Machine%20Learning%20Prediction_Part_5.ipynb

Results

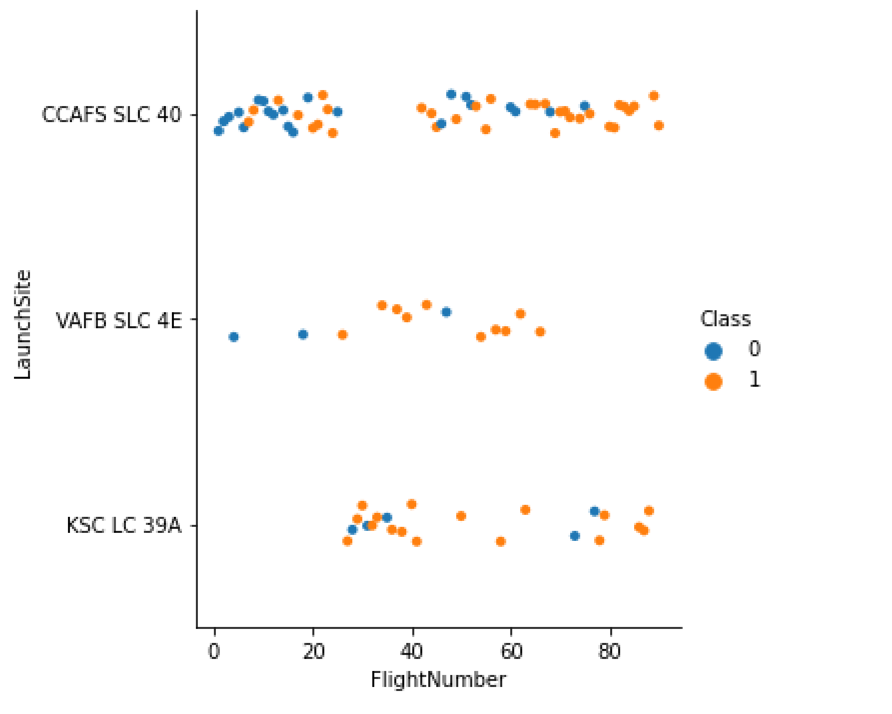
- Orbit GEO,HEO,SSO,ES L1 has the best Launch Success Rate.
- Low weighted payloads have higher chances of success
- As time progresses the success rate of launches increases probably as past mistakes are avoided
- The decision tree model is the best in terms of accuracy however all model fair equally when looking at the confusion matrix classifications.

Section 2

Insights drawn from EDA

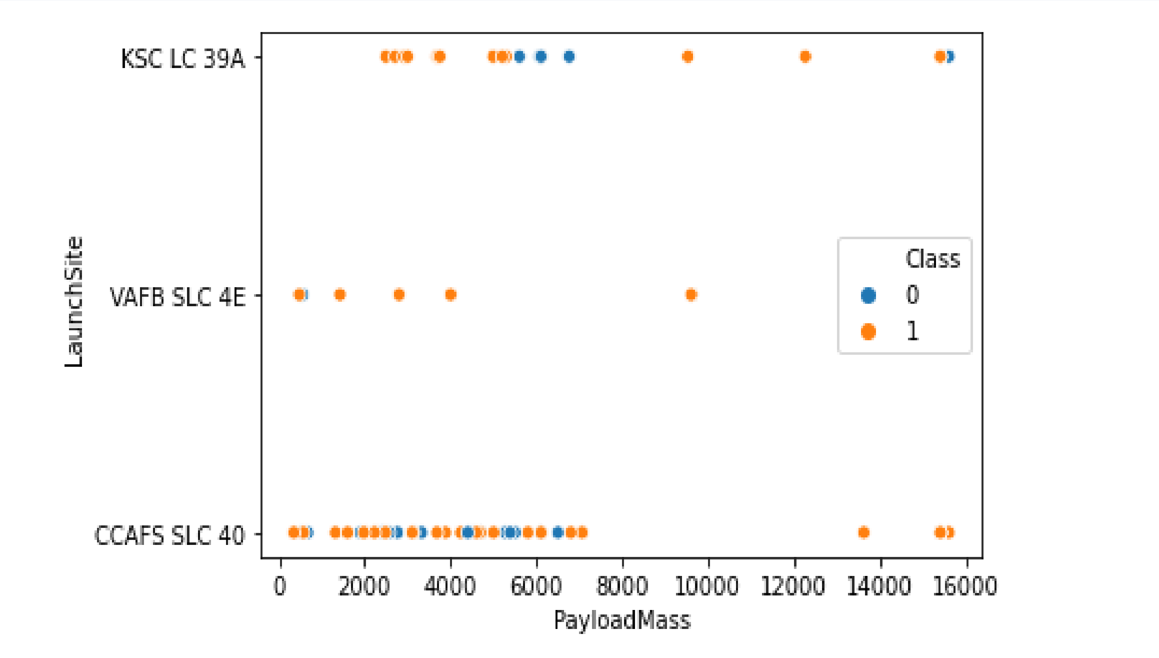
Flight Number vs. Launch Site

- Launches from site CCAFS SLC 40 are more frequent than other sites.
- Higher flight numbers lead to higher chances of successful launches.



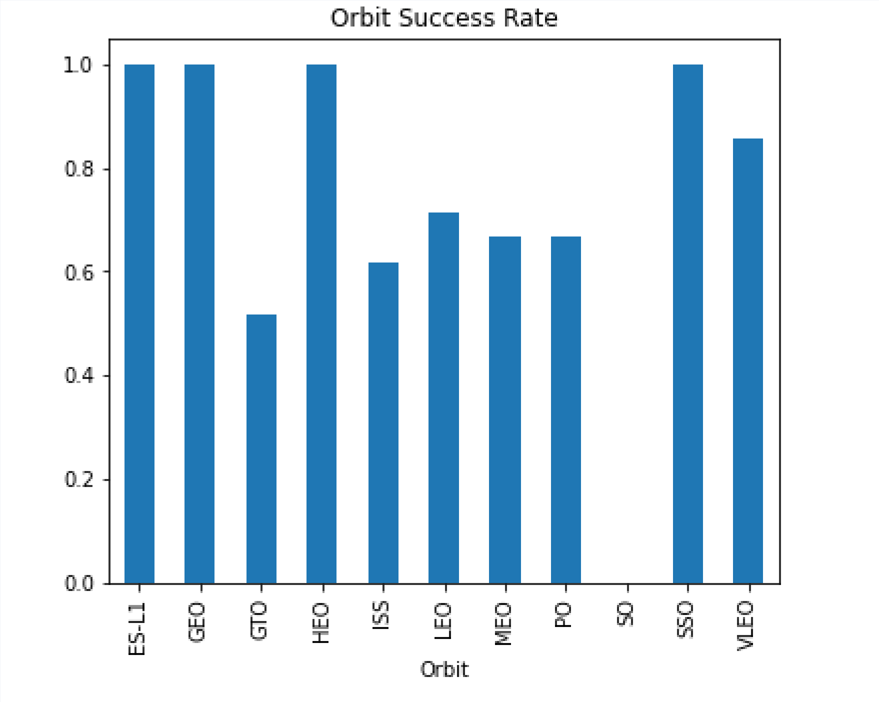
Payload vs. Launch Site

- For the VAFB-SLC launchsite there are no rockets launched for heavypayload mass(greater than 10000)
- Extreme payloads (either near max or min) lead to higher chances of successful launches.



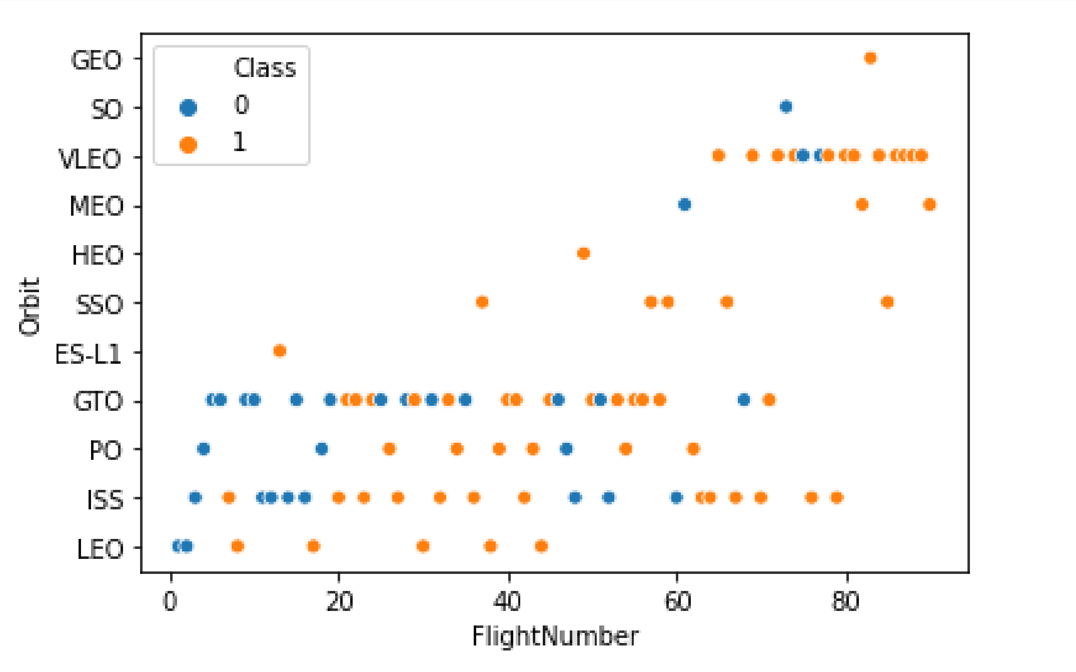
Success Rate vs. Orbit Type

- ES-L1, GEO, HEO and SSO have a 100% success rate.
- SO has no successful launches for the data.
- Other orbits have around 60 to 70% success rate except GTO which stands at 50%.



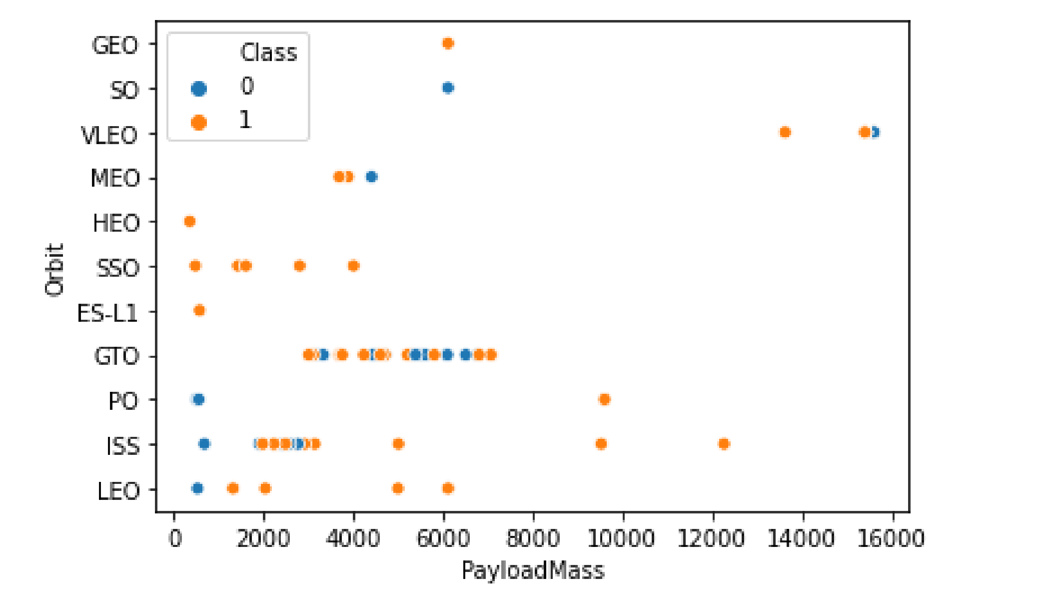
Flight Number vs. Orbit Type

- In the LEO orbit the Success appears related to the number of flights.
- In GTO orbit there is no such relationship.
- Generally success rates tend to increase with increasing flight numbers.



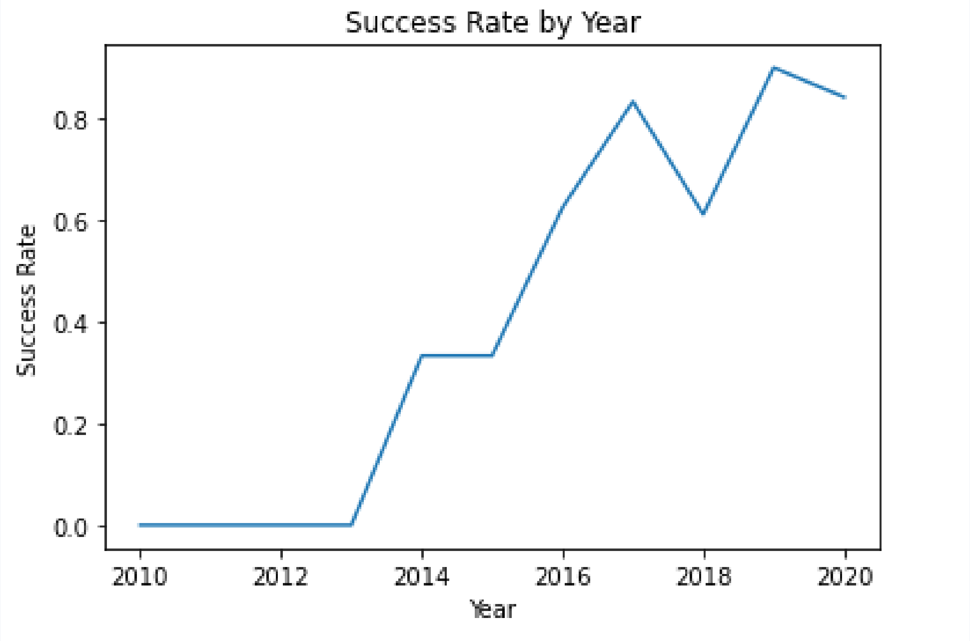
Payload vs. Orbit Type

- With heavy payloads the successful landing or positive landing rate are more for Polar, LEO and ISS.
- No such relationship is found for GTO.
- VLEO, ISS and PO are used for high payload mass launches.



Launch Success Yearly Trend

- After 2013 the Success Rate seems to be increasing till 2017 probably due to increasing flight numbers.
- 2018 seems anomalous with a dip in success rate, which recovers in 2019.



All Launch Site Names

- Cur.execute command can be used to fetch the sql query and give results.
- 4 distinct launch sites are present as shown below the cell.

```
[8]: query = "select distinct launch_site from SPACEXTBL"

     results = cur.execute(query).fetchall()
     print(results)

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

Launch Site Names Begin with 'CCA'

- Sql Magic (%sql) can be used to execute sql queries in jupyter notebooks. Using this 5 records with launch site like CCA are shown below.
- Select * is used to display this with appropriate where condition.

```
[7]: %sql select * from SPACEXTBL where launch_site like 'CCA%' limit 5
```

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

[7]:

Date	Time (UTC)	Booster_Version	Launch_Site	Payload	PAYLOAD_MASS__KG_	Orbit	Customer	Mission_Outcome	Landing _Outcome
04-06-2010	18:45:00	F9 v1.0 B0003	CCAFS LC-40	Dragon Spacecraft Qualification Unit	0	LEO	SpaceX	Success	Failure (parachute)
08-12-2010	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)
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 B0007	CCAFS LC-40	SpaceX CRS-2	677	LEO (ISS)	NASA (CRS)	Success	No attempt

Total Payload Mass

- Total Payload Mass carried by NASA (CRS) is 45596 kg in the data.
- Sum function is used to find this with appropriate where condition.

Task 3

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

```
[8]: %sql select sum(PAYLOAD_MASS__KG_) as TotalMass from SPACEXTBL where Customer = 'NASA (CRS)' limit 5
```

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

[8]:

TotalMass
45596

Average Payload Mass by F9 v1.1

- Average payload mass carried by booster version F9 v1 is around 2928 kg.
- Avg function is used to find this with appropriate where condition.

```
[10]: %sql select AVG(PAYLOAD_MASS__KG_) as AvgMass from SPACEXTBL where Booster_Version = 'F9 v1.1' limit 5

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

[10]:

AvgMass
2928.4

First Successful Ground Landing Date

- First successful landing outcome on ground pad occurred on 01-05-2017.
- Min function is used to find this with appropriate where condition.

```
[11]: %sql select min(Date) as MinDate from SPACEXTBL where [Landing _Outcome] = 'Success (ground pad)' limit 5

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

[11]:

MinDate
01-05-2017

Successful Drone Ship Landing with Payload between 4000 and 6000

- Names of boosters which have successfully landed on drone ship and had payload mass greater than 4000 but less than 6000 are mentioned below.
- Distinct Booster_Version with appropriate where conditions is used to find this.

```
[12]: %sql select distinct Booster_Version from SPACEXTBL where PAYLOAD_MASS__KG_ > 4000 AND PAYLOAD_MASS__KG_< 6000 AND [Landing _Outcome] = 'Success (drone ship)'

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

[12]:

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

Total Number of Successful and Failure Mission Outcomes

- Group by condition on Mission_Outcome gives the count of each probable outcome.
- A Total of 101 outcomes are present with majority being successful.

```
[13]: %sql select Mission_Outcome,count(*) as TotalOutcomes from SPACEXTBL group by Mission_Outcome

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

[13]:

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

Boosters Carried Maximum Payload

- A subquery to find max payload is used and then distinct Booster Versions which have carried that max payload are displayed.

```
[15]: %sql select distinct Booster_Version from SPACEXTBL where PAYLOAD_MASS__KG_ = (Select max(PAYLOAD_MASS__KG_) from SPACEXTBL)

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

[15]:

Booster_Version
F9 B5 B1048.4
F9 B5 B1049.4
F9 B5 B1051.3
F9 B5 B1056.4
F9 B5 B1048.5
F9 B5 B1051.4
F9 B5 B1049.5
F9 B5 B1060.2
F9 B5 B1058.3
F9 B5 B1051.6
F9 B5 B1060.3
F9 B5 B1049.7

2015 Launch Records

- Substring commands are used to pick month and year from date. Failure in drone ship is used in where condition and the month, booster version and launch site are displayed with total number of failures, using group by.

```
[22]: %sql select substr(Date, 4, 2) as Month,Booster_Version,launch_site,Count(*) as Failures from SPACEXTBL where substr(Date,7,4)='2015' and [Landing _Outcome] = 'Failure (drone ship)' group by 1,2,3

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

[22]:

Month	Booster_Version	Launch_Site	Failures
01	F9 v1.1 B1012	CCAFS LC-40	1
04	F9 v1.1 B1015	CCAFS LC-40	1

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

- Count of landing outcomes between the date 2010-06-04 and 2017-03-20 is shown below. Order by command is used for putting the output in descending order.

```
[26]: %sql select [Landing _Outcome],Count(*) as OutcomeCount from SPACEXTBL where Date > '04-06-2010' and Date < '20-03-2017' group by 1 order by 2 Desc
```

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

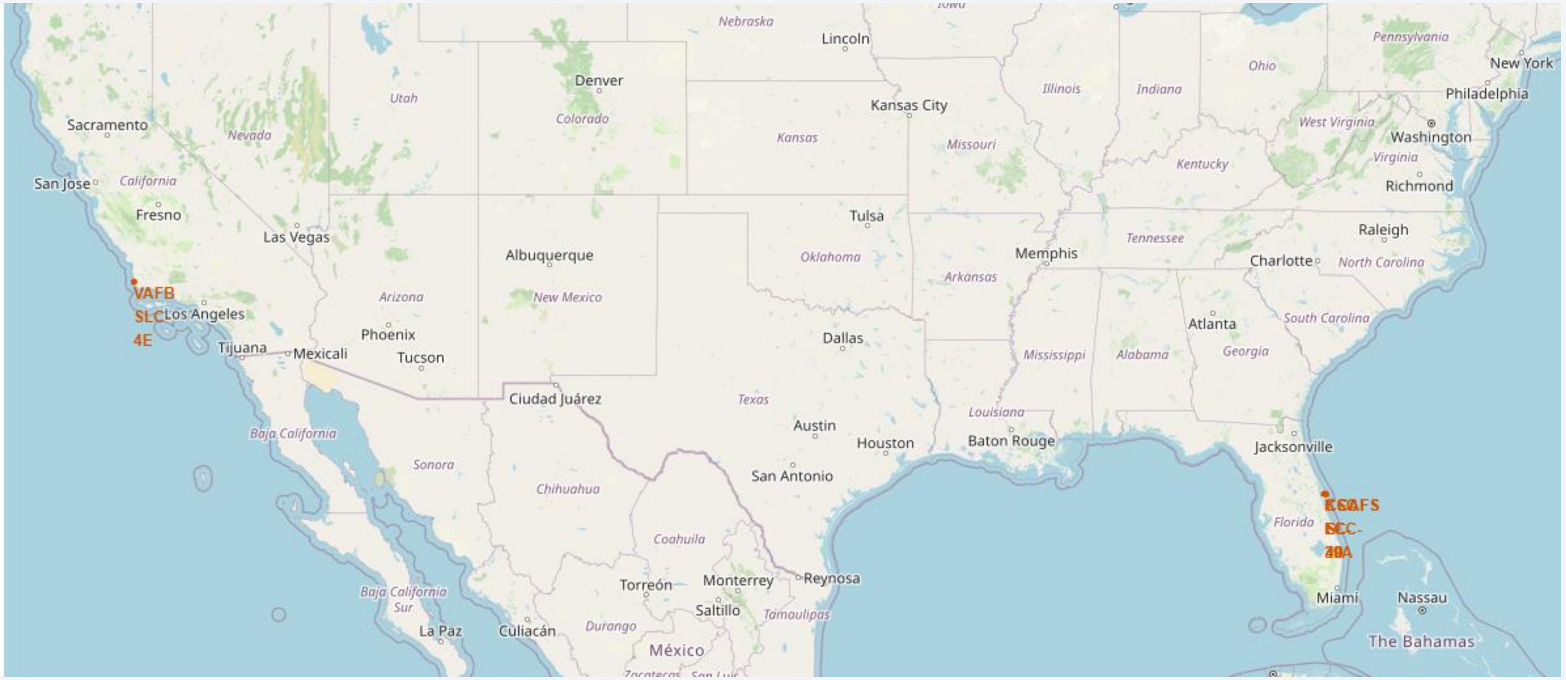
[26]:

Landing _Outcome	OutcomeCount
Success	20
No attempt	10
Success (drone ship)	8
Success (ground pad)	6
Failure (drone ship)	4
Failure	3
Controlled (ocean)	3
No attempt	1
Failure (parachute)	1

Section 3

Launch Sites Proximities Analysis

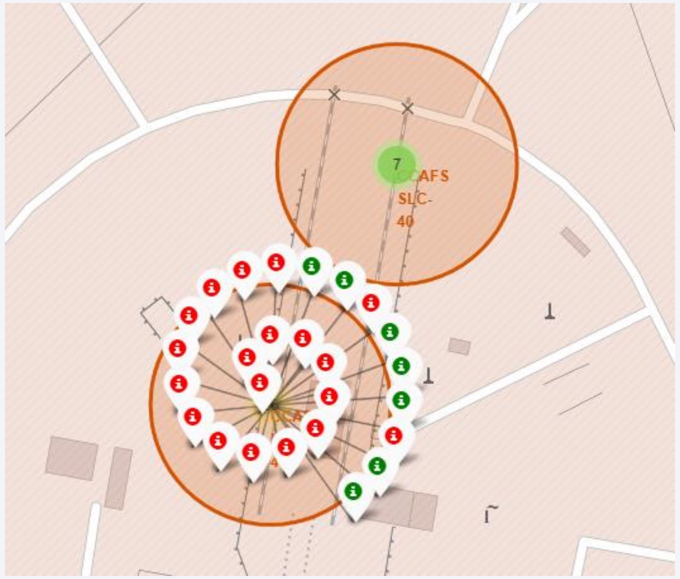
Launch Site Locations on Map

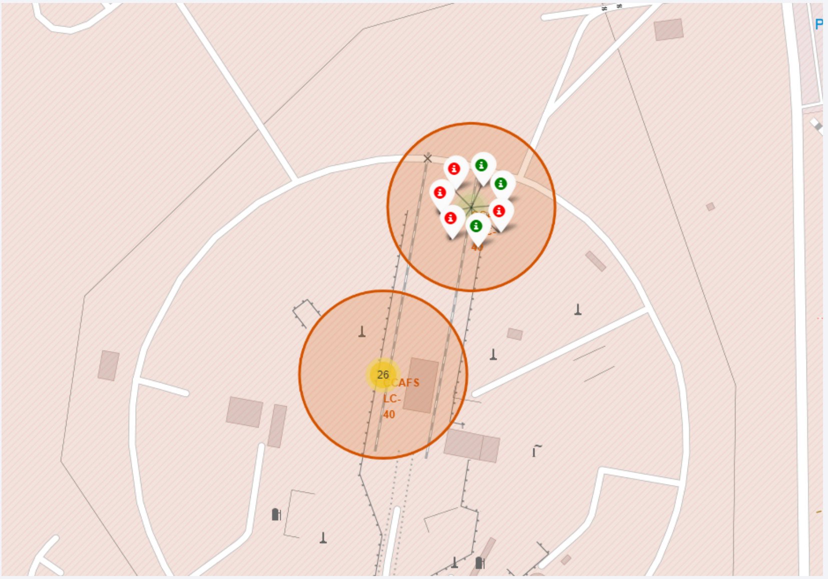


All the launches are in USA near coasts of Florida, and California

Labeling Launch Outcomes with Color

- Green market denotes successful launches.
- Red marker denotes failed launches.
- Two such results for sites in proximity are shown below.





Launch Site Proximity

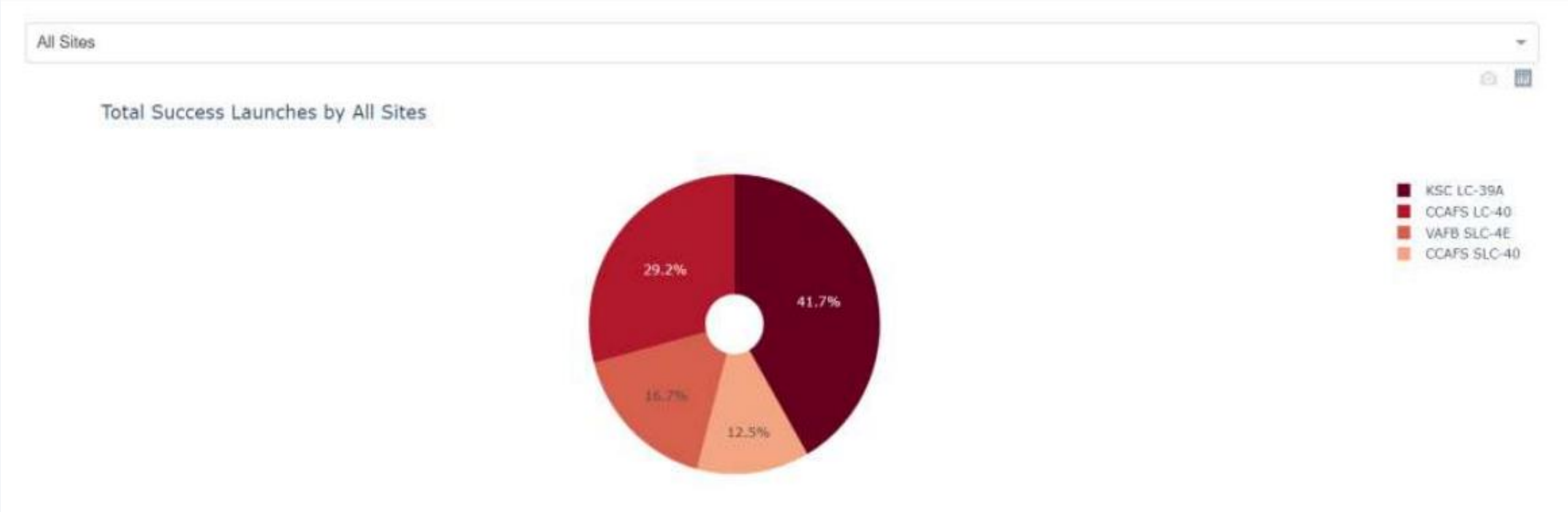
- Launch site and its proximities such as railway, highway, coastline, with distance calculated and displayed is shown.
- Most sites are close to railway tracks.

Section 4

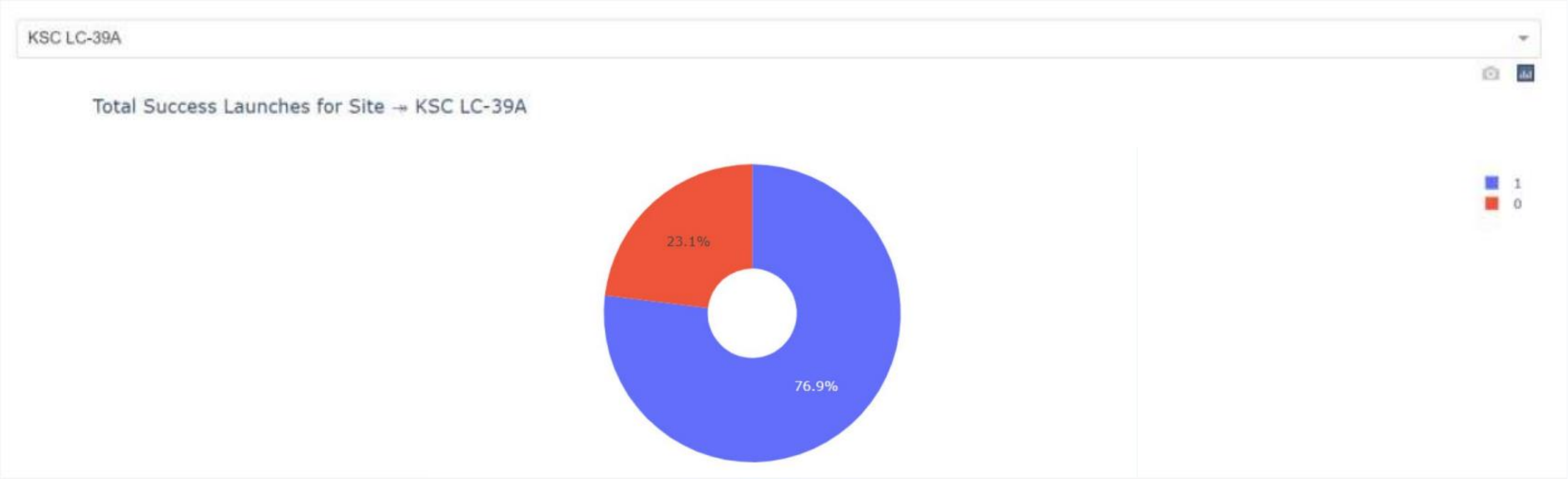
Build a Dashboard with Plotly Dash

Successful Launches by Site

- KSC-LC-39A has the highest launch success rate followed by CCAPS-LC-40 and VAFB SLC-4E.
- CCAFS SLC-40 has the least success % among sites.

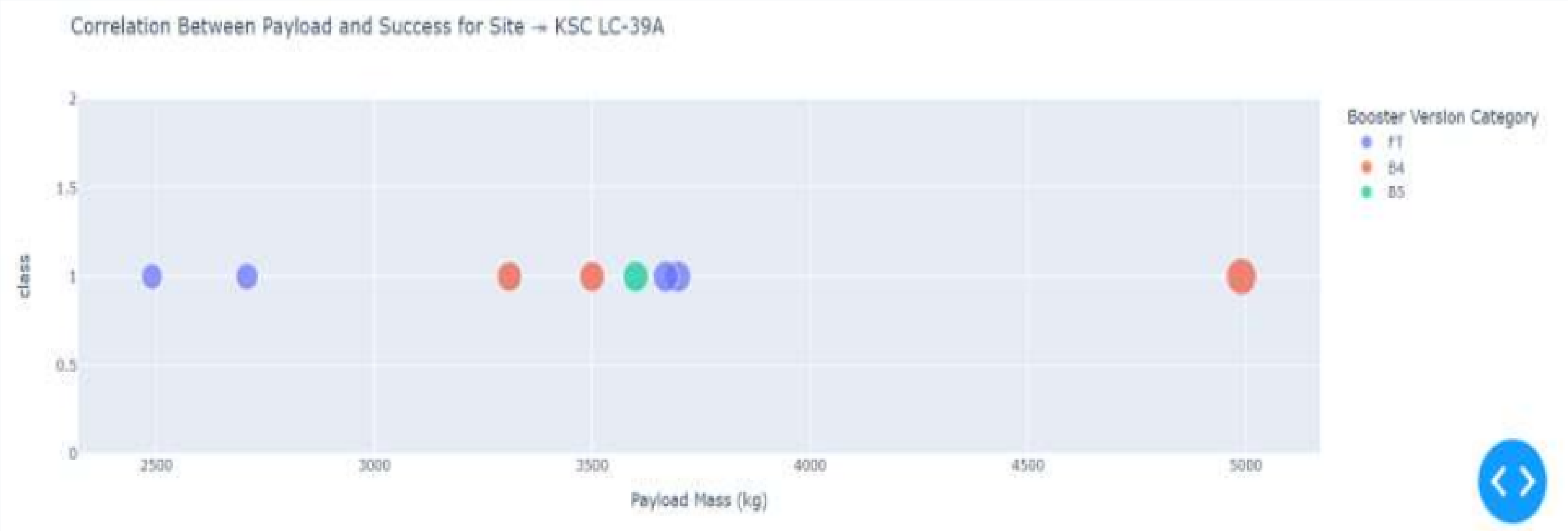


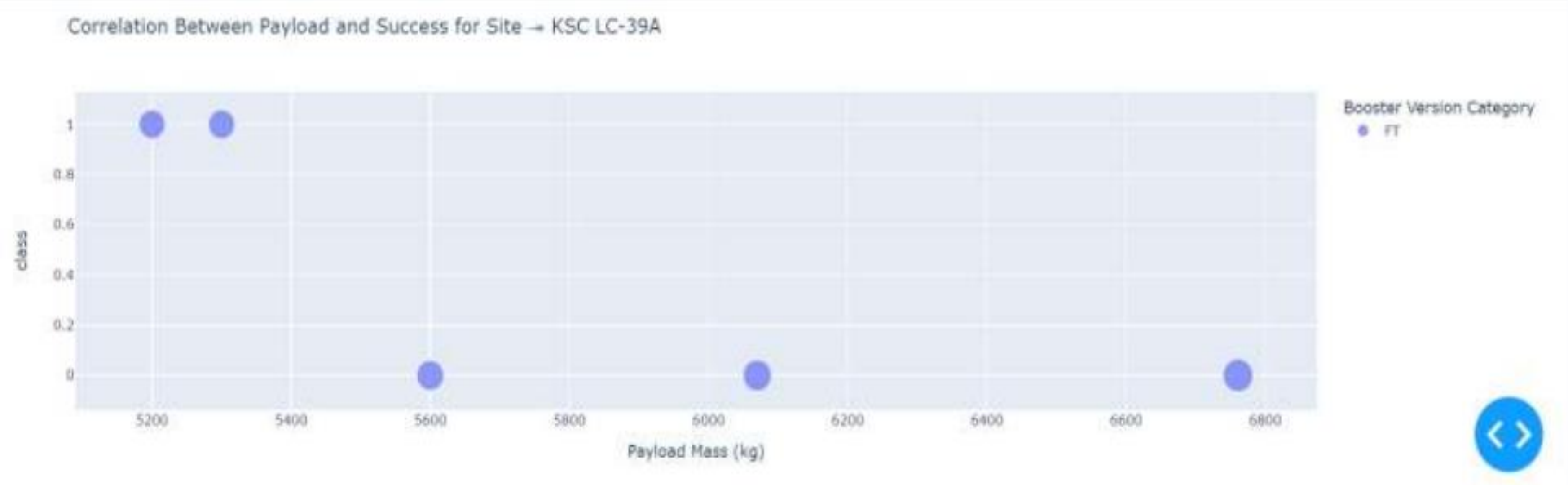
Success Rate by Site



KSC LC-39A has the highest score with 76.9% success rate.

Payload vs Launch Outcome



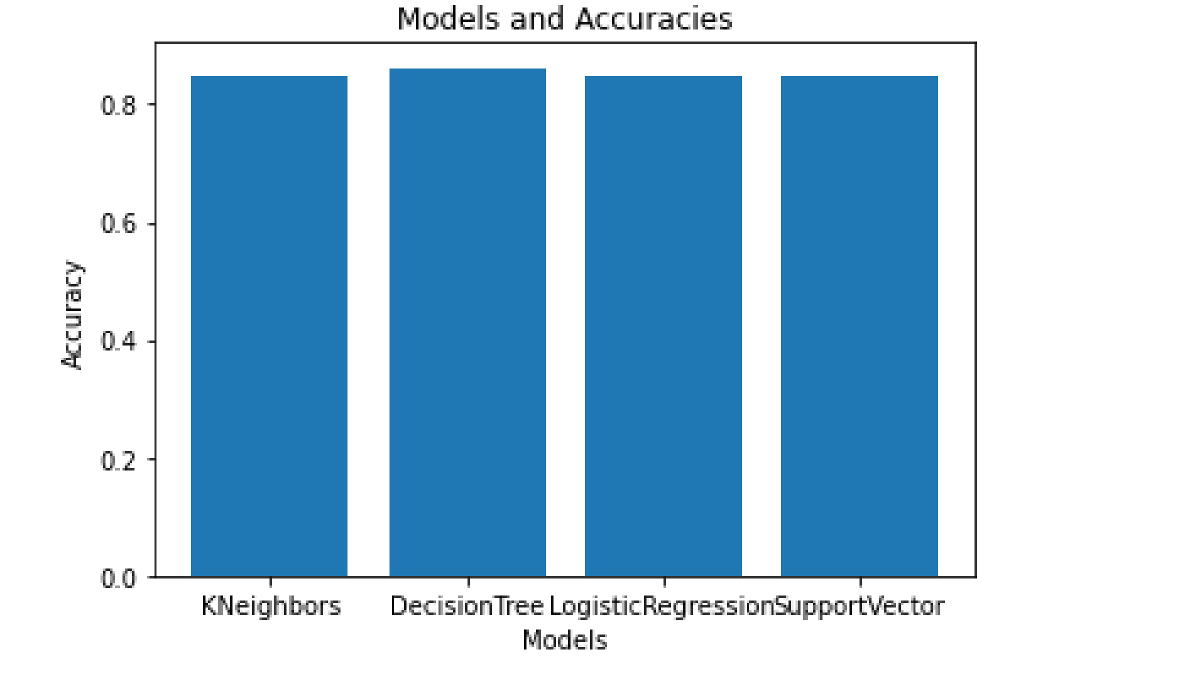


Low weighted payloads have a higher success rate as compared to high weight payloads

Section 5

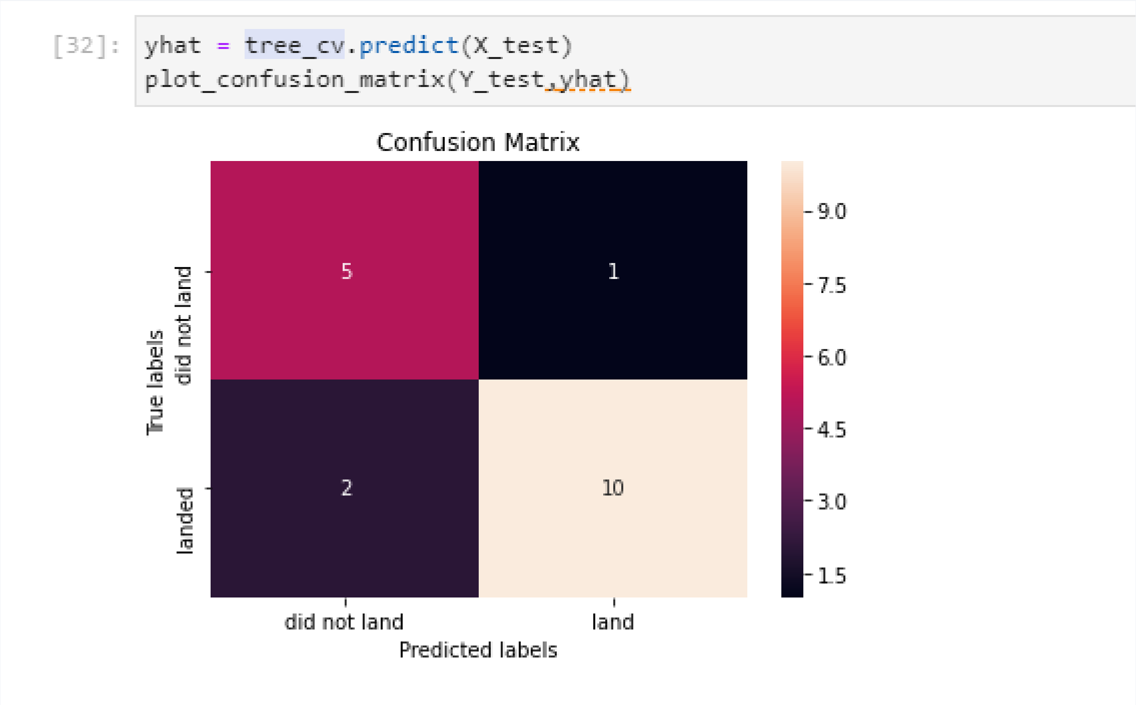
Predictive Analysis (Classification)

Classification Accuracy



Decision Tree has the highest accuracy with 86.1%, followed by KNeighbors, Logistic Regression and then Support Vector Machine.

Confusion Matrix



- The confusion matrix contains True Labels (actual outcome) on the y-axis and Predicted Labels (Predicted Outcome) on the x axis.
- The values at the top-down diagonal (5,10) are the ones that show predictions being similar to actuals so they represent correct predictions.
- The values at bottom-up diagonal (2,1) are the ones that show incorrect predictions.
- In this case 15/18 predictions were made correctly.
- Precision = 0.91, formula: TP / (TP + FP)
- Recall = 0.83, formula: TP / (TP + FN)

Conclusions

- Orbit GEO,HEO,SSO,ES L1 has the best Launch Success Rate.
- Low weighted payloads have higher chances of success
- As time progresses the success rate of launches increases probably as past mistakes are avoided
- The decision tree model is the best in terms of accuracy however all model fair almost equally when looking at the confusion matrix classifications.

Appendix

All the notebooks used in this project are placed at :

https://github.com/saadwali/testrepo

Thank you!