Winning Space Race with Data Science

Müller G. Urias

Nov 10th, 2021

Outline / Index

Executive Summary

Abstract

Introduction: **Space X reduced costs in rocket launches** and allowed a number of technology advantages facilitating reaching space. One of main factors of this reduction cost **is related to the landing and the reuse of the second stage rocket**. **This project aims to analyse features involved in SpaceX Falcon 9 launches and their landing outcomes**, **to map their launch site and perform geographical analysis**, and **to evaluate algorithms to predict a successful landing** based on a set of features.

Methodology: Data analysis was performed using **SpaceX data** available from their **API**. Collection and wrangling were performed to include features from Falcon 9 launches up to June 9th, 2021. Data scrapping was also perfomed on Wikipedia usind BeautifulSoup library. Maps were created using Folium library and dashboards using Dash library. Logistic regression (LR), Support Vector Machine (SVM), Decision Tree Classifier (DT) and K-Nearest Neighbours (KNN) were tested using GridSearchCV from SKLearn.

Results: **KSC launch site is more favorable to more success on reduced payloads**. For **the heaviest payloads, F9 B5 booster and CCAFS SLC-40 launch site were related to have more successful outcomes**, and **Polar, LEO and ISS orbits are more prone to successful landings**. Using our data, **we were able to predict 83% of successful landings**. There was no difference between algorithms in our analysis, using the proposed sample.

Conclusion: It is possible to use Data Science in order to **understand current results** on successful Falcon 9 landings, to **improve current methods** through analysis and predictions, and finally to **enhance results, optimize costs** and add one step closer to win this race.

Appendix: Python files, jupyter notebooks and support data are available on github.com/uriasmg.

contato@drmuller.com.br

1 In · tro duc · tion

Space Exploration Technologies Corp. (Space X) was founded by 2002 and provided in that year a suborbital trip to space [1]. Initially wanting to provide resupply missions to the International Space Station (ISS), SpaceX develop technology at a point to reduced costs in rocket launches and to allow a number of other advantages when facilitating reaching space [2]–[5] . One of main factors of this reduction cost is related to the landing and reusability of the Rocket's first stage. An example of that is the Falcon 9, the cheapest in the industry [6], [7] which had, in late 2015, its first successfully first stage landed [8].

A successful landing outcome is a strong factor in keep reduced costs and keep venue for companies - as customers - wanting to invest in reaching space. Therefore, this is a key parameter for SpaceX grow and reducing monopoly establishments from earlier days. The way to achieve first stage landing is, less to say, a complex engineer task and involves years of hard development and the eager of courage investors.

Apart from technology development and human handling, a multitude of variables could interfere in a landing, such as geographically related conditions from a given launch site, the landing site, the rocket related conditions – such as payload mass, rocket engine, the targeted orbit, number of reuses, even the attempt number could be related given the learning and experience rate.

However, given the data available on third party sources and from SpaceX itself - through its Application Programming Interface (API) – its interesting to know if a company could use Data Science to predict landing success and which features are related to more accomplishment of the landing goal.

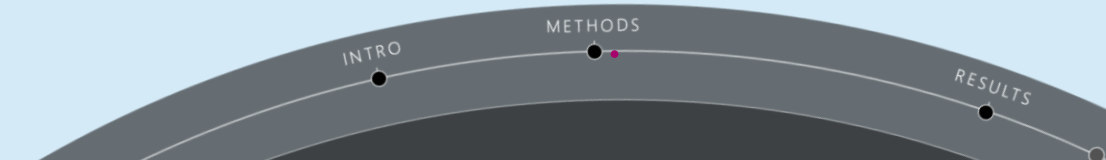
This project aims to analyse features involved in SpaceX Falcon 9 launches and their landing outcomes, to map their launch site and perform geographical analysis, and to evaluate algorithms to predict a successful landing based on a set of features.

2 Meth·od·ol·o·gy

2 Methodology

Data collection methodology:

- Data collection
- Data wrangling
- Exploratory data analysis (EDA) using visualization and SQL
- Visual analytics using Folium and Plotly Dash
- Predictive analysis using classification models
- Build, tune, evaluate classification models



contato@drmuller.com.br

2 Data Collection

```
spacex_url="https://api.spacexdata.com/v4/launches/past"
```

```
response = requests.get(spacex_url)
```

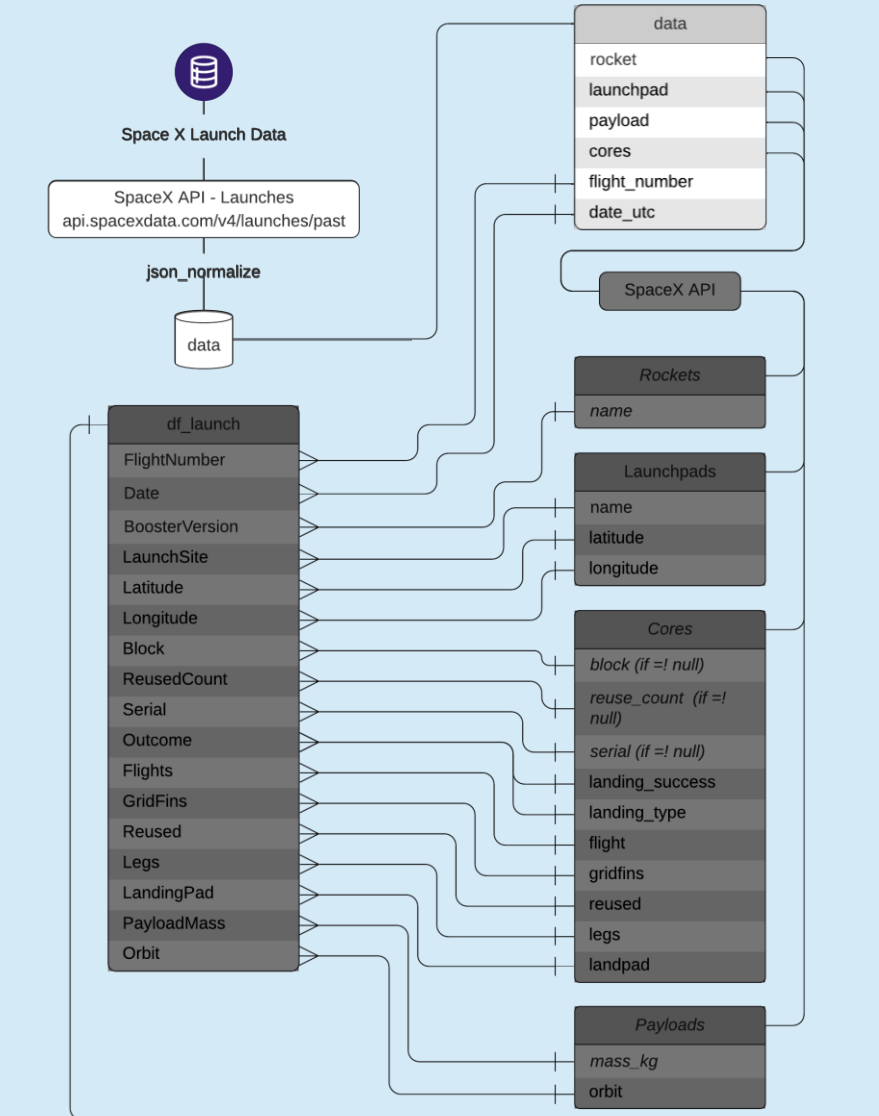
Check the content of the response

```
print(response.content)
```

```
b'[{"fairings":{"reused":false,"recovery_attempt":false,"recovered":false,"ships":[]},"links":{"patch":{"small":"http
s://images2.imgbox.com/3c/0e/T8iJcSN3_o.png","large":"https://images2.imgbox.com/40/e3/GypSkayF_o.png"},"reddit":{"cam
paign":null,"launch":null,"media":null,"recovery":null},"flickr":{"small":[],"original":[]},"presskit":null,"webcas
t":"https://www.youtube.com/watch?v=0a_00nJ_Y88","youtube_id":"0a_00nJ_Y88","article":"https://www.space.com/2196-spac
ex-inaugural-falcon-1-rocket-lost-launch.html","wikipedia":"https://en.wikipedia.org/wiki/DemoSat"},"static_fire_date_
utc":"2006-03-17T00:00:00.000Z","static_fire_date_unix":1142553600,"net":false,"window":0,"rocket":"5e9d0d95eda69955f7
09d1eb","success":false,"failures":[{"time":33,"altitude":null,"reason":"merlin engine failure"}],"details":"Engine fa
ilure at 33 seconds and loss of vehicle","crew":[],"ships":[],"capsules":[],"payloads":["5eb0e4b5b6c3bb0006eeb1e1"],"l
aunchpad":"5e9e4502f5090995de566f86","flight_number":1,"name":"FalconSat","date_utc":"2006-03-24T22:30:00.000Z","date_
unix":1143239400,"date_local":"2006-03-25T10:30:00+12:00","date_precision":"hour","upcoming":false,"cores":[{"core":"5
e9e289df35918033d3b2623","flight":1,"gridfins":false,"legs":false,"reused":false,"landing_attempt":false,"landing_succ
ess":null,"landing_type":null,"landpad":null}],"auto_update":true,"tbd":false,"launch_library_id":null,"id":"5eb87cd9f
fd86e000604b32a"},{"fairings":{"reused":false,"recovery_attempt":false,"recovered":false,"ships":[]},"links":{"patch":
{"small":"https://images2.imgbox.com/4f/e3/I0lkuJ2e_o.png","large":"https://images2.imgbox.com/be/e7/iNqsqVYM_o.pn
g"},"reddit":{"campaign":null,"launch":null,"media":null,"recovery":null},"flickr":{"small":[],"original":[]},"presski
t":null,"webcast":"https://www.youtube.com/watch?v=Lk4zQ2wP-Nc","youtube_id":"Lk4zQ2wP-Nc","article":"https://www.spac
e.com/3590-spacex-falcon-1-rocket-fails-reach-orbit.html","wikipedia":"https://en.wikipedia.org/wiki/DemoSat"},"static
_fire_date_utc":null,"static_fire_date_unix":null,"net":false,"window":0,"rocket":"5e9d0d95eda69955f709d1eb","succes
s":false,"failures":[{"time":301,"altitude":289,"reason":"harmonic oscillation leading to premature engine shutdow
n"}],"details":"Successful first stage burn and transition to second stage, maximum altitude 289 km, Premature engine
shutdown at T+7 min 30 s, Failed to reach orbit, Failed to recover first stage","crew":[],"ships":[],"capsules":[],"pa
```

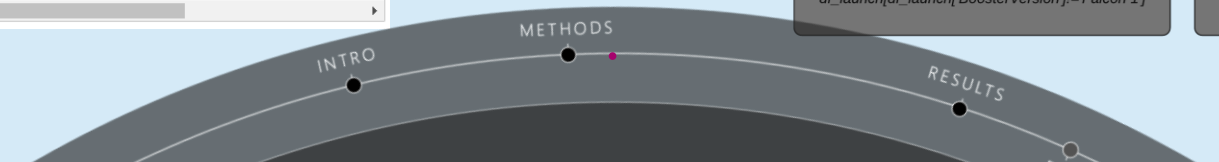
```
# Use json_normalize meethod to convert the json result into a dataframe
# requested_json = requests.get(static_json_url).json()
data = pd.json_normalize(response.json())
```

```
# 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 m
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)]
```



Complete code on:

www.github.com/uriasmg



contato@drmuller.com.br

2
Data Collection SpaceX API

```
# Takes the dataset and uses the rocket column to call the API and append the data to the list
def getBoosterVersion(data):
    for x in data['rocket']:
        response = requests.get("https://api.spacexdata.com/v4/rockets/"+str(x)).json()
        BoosterVersion.append(response['name'])
```

From the `launchpad` we would like to know the name of the launch site being used, the logitude, and the latitude.

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

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

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

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

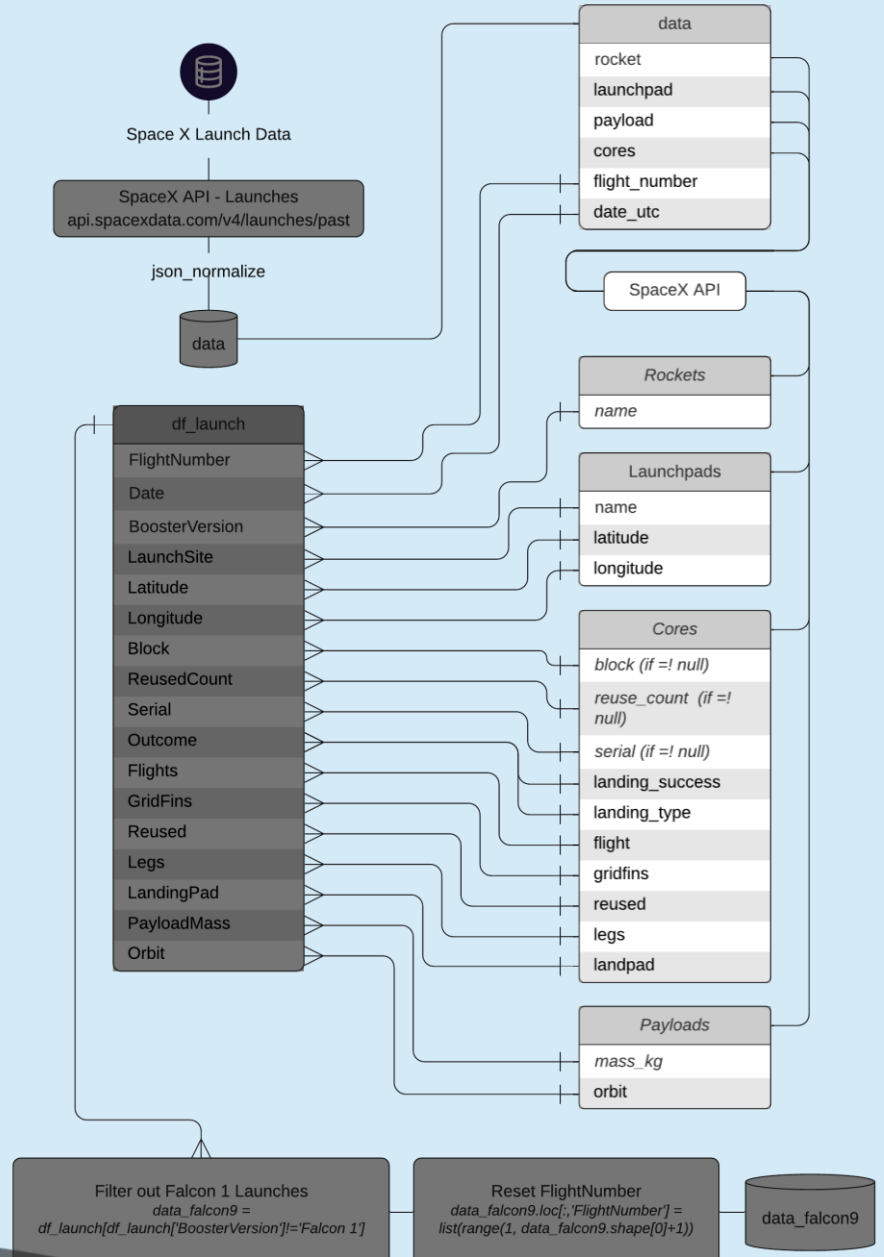
```
# Takes the dataset and uses the cores column to call the API and append the data to the lists
def getCoreData(data):
    for core in data['cores']:
            if core['core'] != None:
                response = requests.get("https://api.spacexdata.com/v4/cores/"+core['core']).json()
          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

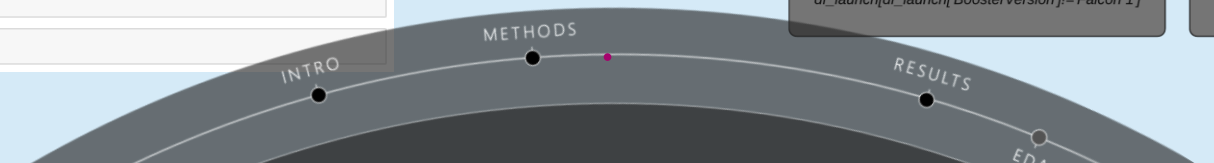
```
# Call getBoosterVersion
getBoosterVersion(data)
```

```
# Call getLaunchSite
getLaunchSite(data)
```

```
# Call getPayloadData
getPayloadData(data)
```

```
# Call getCoreData
getCoreData(data)
```





Complete code on:

www.github.com/uriasmg

contato@drmuller.com.br

2

Data Collection building the database

```
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}
```

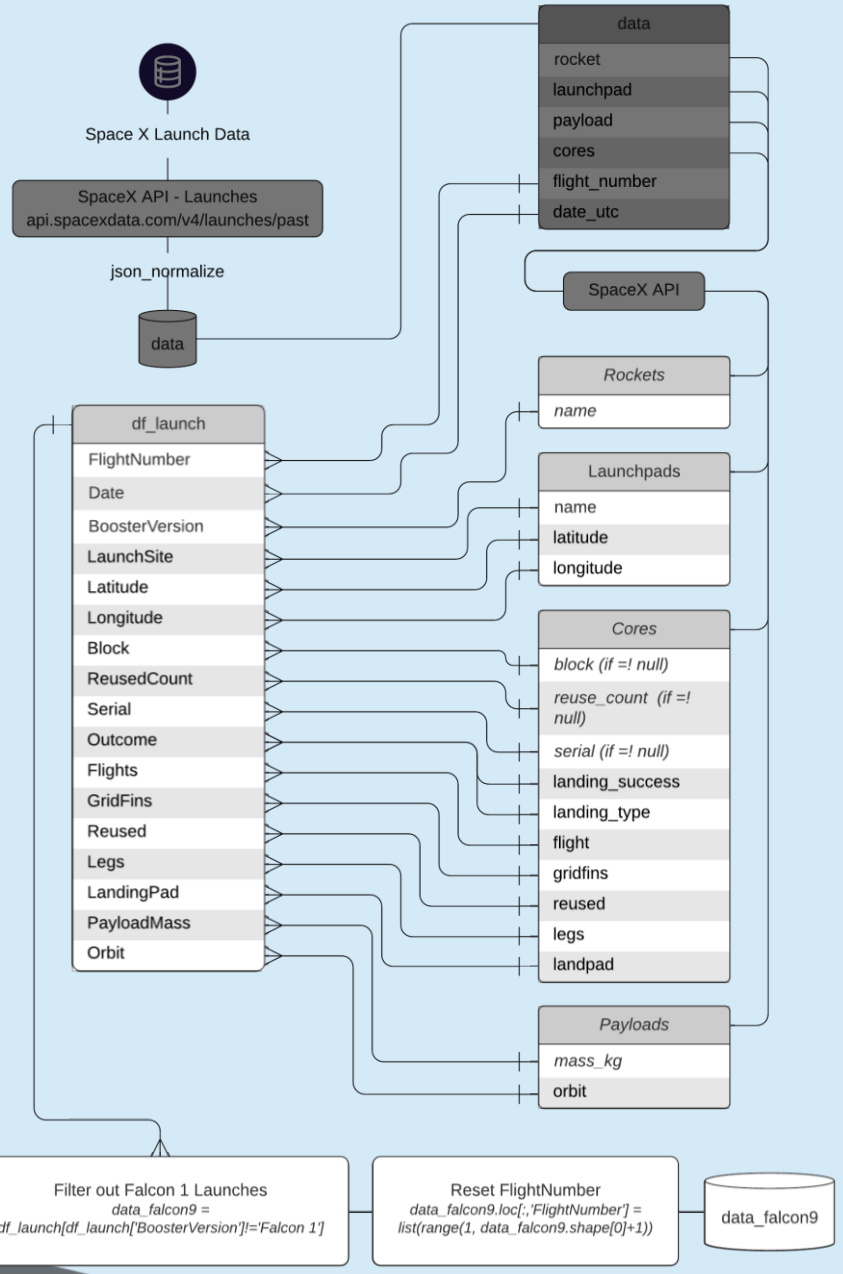
Then, we need to create a Pandas data frame from the dictionary launch_dict.

```
# Create a data from Launch_dict
df_launch = pd.DataFrame.from_dict(launch_dict)
```

```
# Hint data['BoosterVersion']!='Falcon 1'
data_falcon9 = df_launch[df_launch['BoosterVersion']!='Falcon 1']
```

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

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



Complete code on:

www.github.com/uriasmg

contato@drmuller.com.br

2
Data Collection
Web Scraping

```
static_url = "https://en.wikipedia.org/w/index.php?title=List_of_Falcon_9_and_Falcon_Heavy_la"
```

```
# use requests.get() method with the provided static_url
# assign the response to a object
response = requests.get(static_url).text
```

```
# Use BeautifulSoup() to create a BeautifulSoup object from a response text content
soup = BeautifulSoup(response, 'html5lib')
```

```
# 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')
```

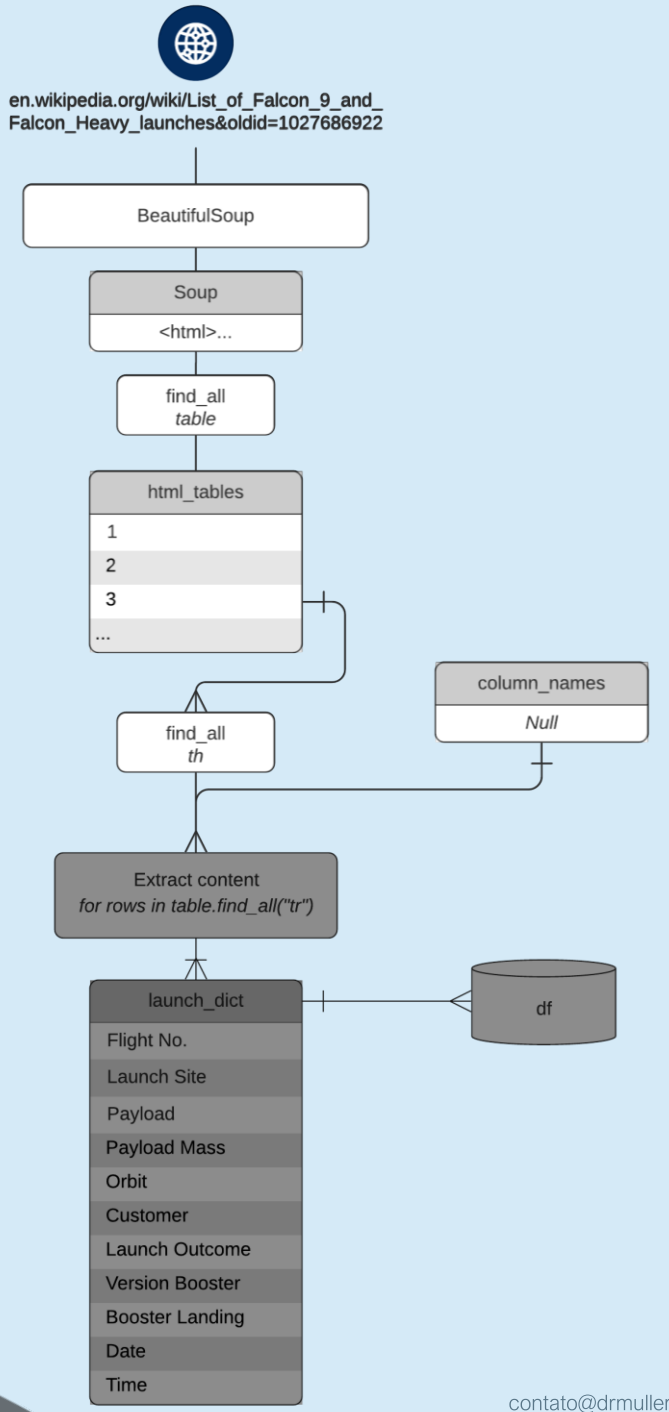
```
# Let's print the third table and check its content
first_launch_table = html_tables[2]
print(first_launch_table)
```

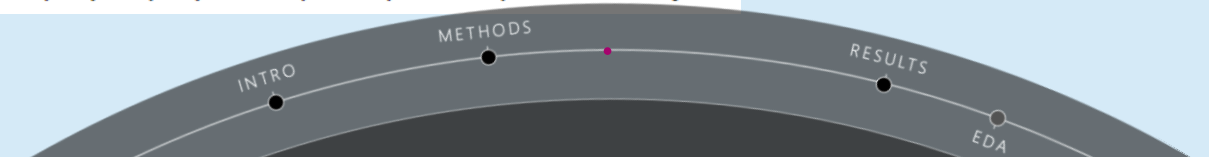
```
column_names = []

# 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

for row in first_launch_table.find_all('th'):
    name = extract_column_from_header(row)
    if (name != None and len(name) > 0):
        column_names.append(name)
```

Check the extracted column names

```
print(column_names)
```

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





Complete code on:

www.github.com/uriasmg

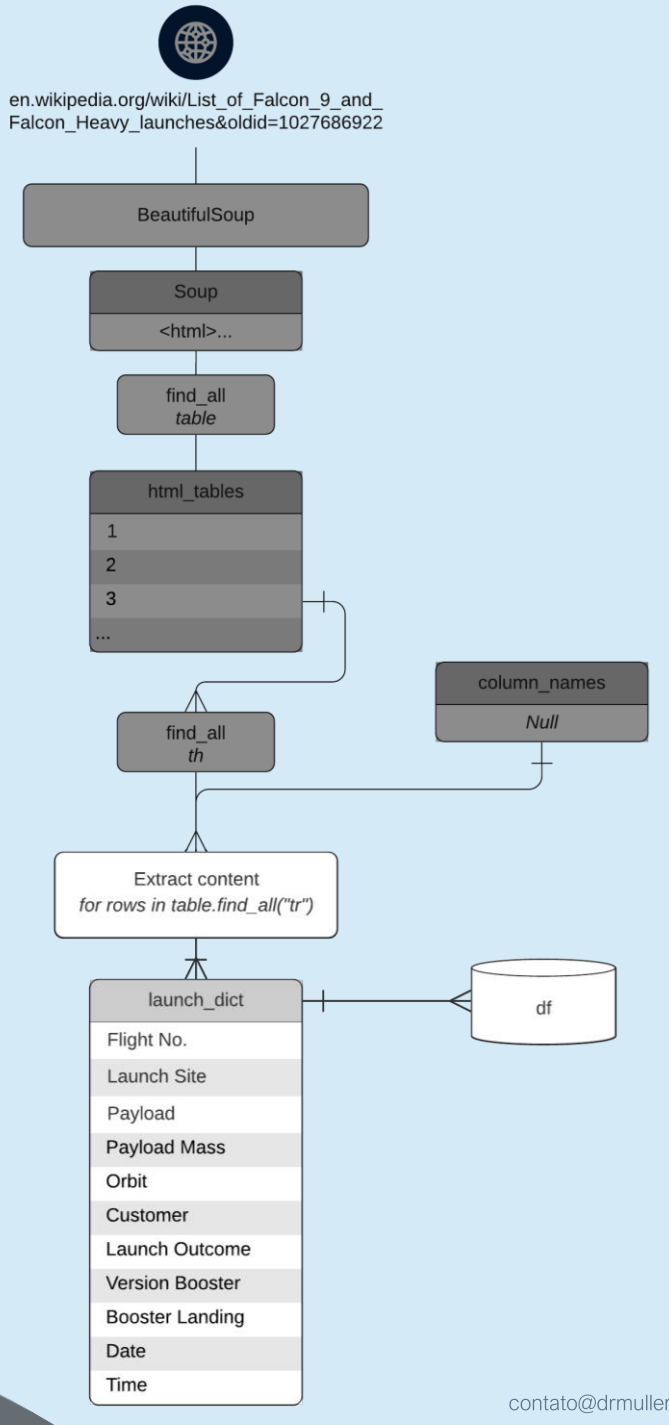
contato@drmuller.com.br

2
Data Collection Web Scraping

```
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'] = []
```

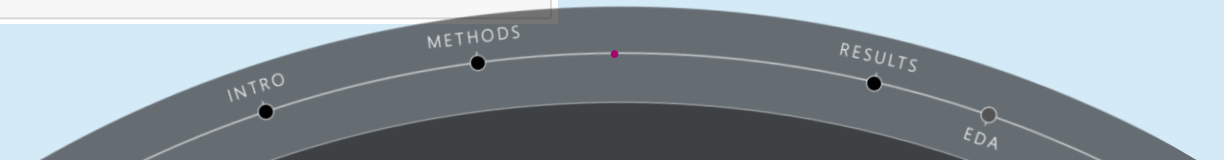
```
extracted_row = 0
#Extract each table
for table_number,table in enumerate(soup.find_all('table',"wikitable plainrowheaders collapsible")):
   # get table row
    for rows in table.find_all("tr"):
        #check to see if first table heading is as number corresponding to launch a number
        if 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)

            # Date value
            # TODO: Append the date into launch_dict with key `Date`
            datatimelist=date_time(row[0])
```

```
df=pd.DataFrame.from_dict(launch_dict)
df
```



Complete code on:

www.github.com/uriasmg



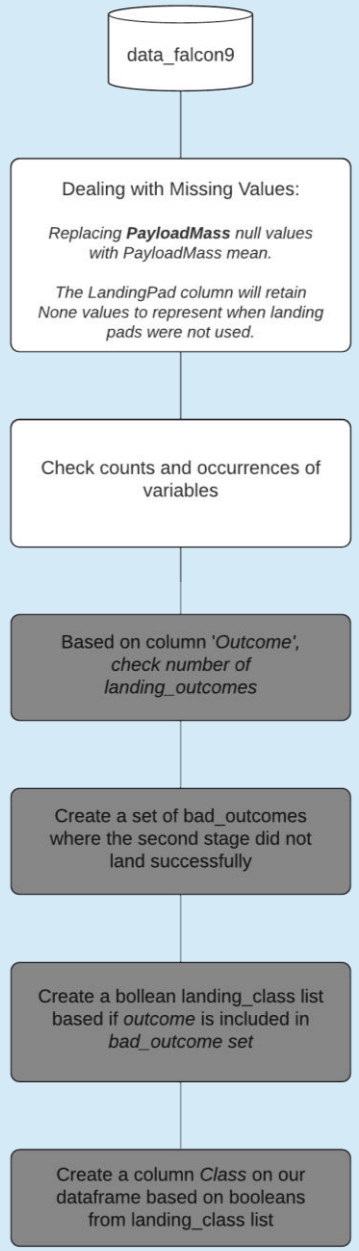
contato@drmuller.com.br

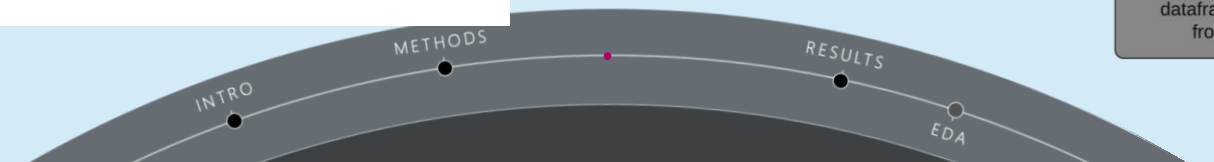
2 Data Wrangling

```
In [33]: # Calculate the mean value of PayloadMass column
         payload_mean = data_falcon9['PayloadMass'].mean

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

         # Checking replacement
         data_falcon9.isnull().sum()

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

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

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

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

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





Complete code on:

www.github.com/uriasmg

contato@drmuller.com.br

2 Data Wrangling

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

```
0]: True ASDS      41
    None None      19
    True RTLS      14
    False ASDS      6
    True Ocean      5
    False Ocean     2
    None ASDS       2
    False RTLS      1
    Name: Outcome, dtype: int64
```

```
for i,outcome in enumerate(landing_outcomes.keys()):
    print(i,outcome)
```

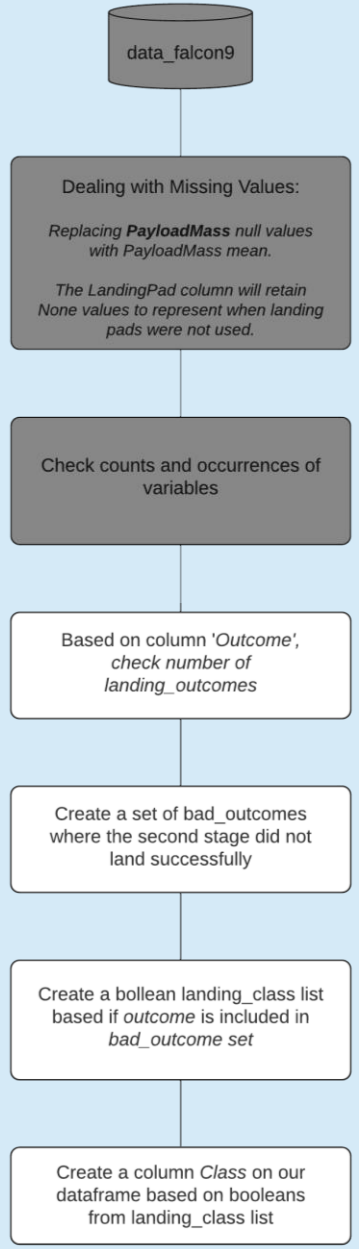
```
0 True ASDS
1 None None
2 True RTLS
3 False ASDS
4 True Ocean
5 False Ocean
6 None ASDS
7 False RTLS
```

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

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

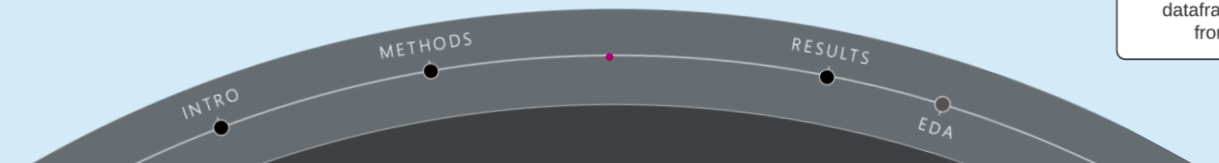
```
landing_class=[]
# landing_class = 0 if bad_outcome
for outcome in df['Outcome']:
    if outcome in bad_outcomes:
        landing_class.append(0)
# landing_class = 1 otherwise
    else:
        landing_class.append(1)
```

```
df['Class']=landing_class
```



Complete code on:

www.github.com/uriasmg



contato@drmuller.com.br

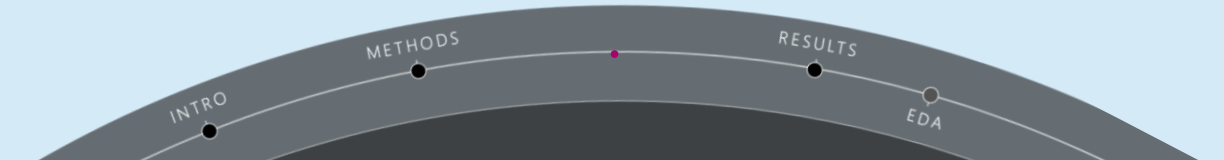
2 Exploratory Data Analysis (EDA) with Data Visualization

Libraries and uses in our case:

- Pandas for dataframes
- Numpy for arrays and math functions
- Matplotlib for bar plot
- Seaborn for scatter plots

Plots:

- Flight Number vs Payload Mass (scatter plot)
- Flight Number vs Launch Site (scatter plot)
- Payload Mass vs Launch Site (scatter plot)
- Orbit Type vs Success Rate (bar chart)
- Flight Number vs Orbit Type (scatter plot)
- Payload Mass vs Orbit Type (scatter plot)
- Launch Success yearly trend (line plot)



contato@drmuller.com.br

2 Exploratory Data Analysis (EDA) with SQL

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

```
SELECT DISTINCT LAUNCH_SITE FROM SPACEX;
```

2. Display 5 records where launch sites begin with the string 'CCA'

```
SELECT * FROM SPACEX WHERE LAUNCH_SITE LIKE 'CCA%' LIMIT 5;
```

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

```
SELECT SUM(payload_mass__kg_) as Total_Payload_Mass FROM SPACEX;
```

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

- Average payload mass carried by booster version F9 v1.1 (any)

```
SELECT AVG(payload_mass__kg_) average_payload FROM SPACEX WHERE BOOSTER_VERSION LIKE 'F9 v1.1%';
```

- Average payload mass carried by booster version F9 v1.1 (exactly)

```
SELECT AVG(payload_mass__kg_) average_payload FROM SPACEX WHERE BOOSTER_VERSION = 'F9 v1.1';
```

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

```
SELECT MIN(date) first_successful_landing_date FROM SPACEX WHERE LANDING__OUTCOME LIKE 'Success%';
```

6. List the names of the boosters which have success in drone ship and have payload mass greater than 4000 but less than 6000

```
SELECT DISTINCT BOOSTER_VERSION FROM SPACEX
    WHERE LANDING__OUTCOME = 'Success (drone ship)' AND 4000 < payload_mass__kg_ < 6000;
```

7. List the total number of successful and failure mission outcomes

```
SELECT mission_outcome, count(mission_outcome) total_number FROM SPACEX group by mission_outcome;
```

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

```
SELECT booster_version, payload_mass__kg_ FROM SPACEX
    WHERE payload_mass__kg_ =(
        SELECT MAX(payload_mass__kg_) FROM SPACEX
        );
```

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

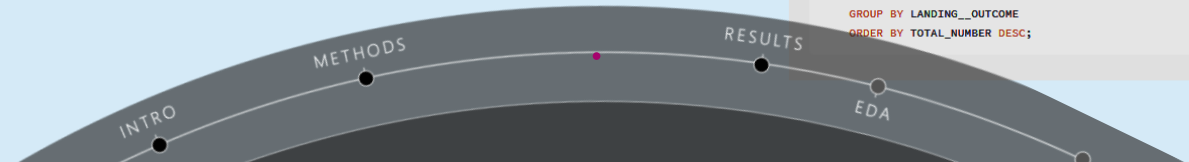
```
SELECT LANDING__OUTCOME, BOOSTER_VERSION, LAUNCH_SITE
FROM SPACEX
WHERE Landing__Outcome = 'Failure (drone ship)'
    AND YEAR(DATE) = 2015;
```

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

```
SELECT LANDING__OUTCOME, COUNT(LANDING__OUTCOME) TOTAL_NUMBER FROM SPACEX
    WHERE DATE BETWEEN '2010-06-04' AND '2017-03-20'
    GROUP BY LANDING__OUTCOME
    ORDER BY TOTAL_NUMBER DESC;
```

Complete code on:

www.github.com/uriasmg



2 Interactive Map with Folium

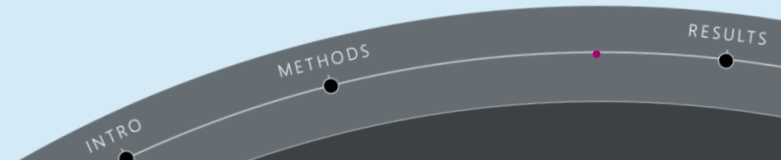
- Libraries:
- Folium
- Folium.plugins:
    - MarkerCluster -
    - MousePosition
    - DivIcon

Tasks:

- Mark all sites on the map
- Verify their geographical relationship (coastlines, tropics / equatorial line proximities)
- Create a MarkerCluster with each LaunchSite outcomes
- Verify distances to highways, railway and nearest city

Complete code on:

www.github.com/uriasmg



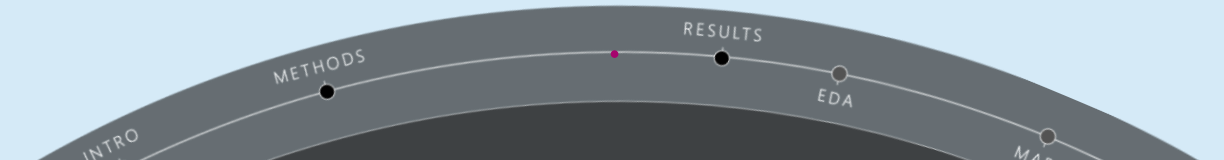
2 Dashboard with Plotly Dash

```
# SpaceX Dashboard

# Import required libraries
import pandas as pd
import dash
import dash_html_components as html
import dash_core_components as dcc
from dash.dependencies import Input, Output
import plotly.express as px

# Read the airline data into pandas dataframe
spacex_df = pd.read_csv("https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-
max_payload = spacex_df['Payload Mass (kg)'].max()
min_payload = spacex_df['Payload Mass (kg)'].min()

# Create a dash application
app = dash.Dash(__name__)
```

```
# Create an app layout
app.layout = html.Div(children=[html.H1('SpaceX Launch Records Dashboard',
                                        style={'textAlign': 'center', 'color': '#503D36',
                                               'font-size': 40}),
                                # TASK 1: Add a dropdown list to enable Launch Site selection
                                # The default select value is for ALL sites
                                # dcc.Dropdown(id='site-dropdown',...)
                                dcc.Dropdown(id='site-dropdown',
                                             options=[
                                                     {'label': 'All Sites', 'value': 'ALL'},
                                                     {'label': 'CCAFS LC-40', 'value': 'CCAFS LC-40'},
                                                     {'label': 'VAFB SLC-4E', 'value': 'VAFB SLC-4E'},
                                                     {'label': 'KSC LC-39A', 'value': 'KSC LC-39A'},
                                                     {'label': 'CCAFS SLC-40', 'value': 'CCAFS SLC-40'}
                                                     ],
                                             value='ALL',
                                             placeholder='Select a Launch Site here',
                                             searchable=True
                                             # style={'width':'80%','padding':'3px','font-size':'20px','text-align-last':'center'}
                                             ),
                                html.Br(),

                                # TASK 2: Add a pie chart to show the total successful launches count for all sites
                                # If a specific launch site was selected, show the Success vs. Failed counts for the site
                                html.Div(dcc.Graph(id='success-pie-chart')),
                                html.Br(),

                                html.P("Select Payload range (Kg):"),
                                # TASK 3: Add a slider to select payload range
                                #dcc.RangeSlider(id='payload-slider',...)
                                dcc.RangeSlider(id='payload-slider',
                                                min=0,
                                                max=10000,
                                                step=100,
                                                value=[min_payload, max_payload],
                                                marks={0:  '0 ton',2500: '2.5 ton',5000: '5 ton',7500: '7.5 ton',10000: '10 ton'},
                                                tooltip={"placement": "bottom", "always_visible": False}
                                               ),

                                # TASK 4: Add a scatter chart to show the correlation between payload and launch success
                                html.Div(dcc.Graph(id='success-payload-scatter-chart')),
                                ])
```

Complete code on:

www.github.com/uriasmg



contato@drmuller.com.br

2 Dashboard with Plotly Dash

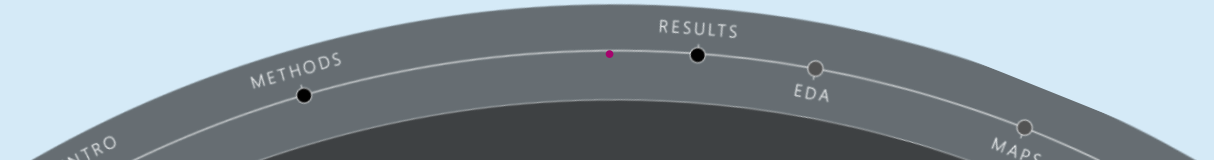
```python
# TASK 2:
# Add a callback function for `site-dropdown` as input, `success-pie-chart` as output
@app.callback(Output(component_id='success-pie-chart', component_property='figure'),
              Input(component_id='site-dropdown', component_property='value'))
def get_pie_chart(entered_site):
    filtered_df = spacex_df
    if entered_site == 'ALL':
        fig = px.pie(filtered_df, values='class',
        names='Launch Site',
        title=f"Success Count for all launch sites: {spacex_df['class'].sum()}")
        return fig
    else:
        # return the outcomes piechart for a selected site
        filtered_df=spacex_df[spacex_df['Launch Site']== entered_site]
        filtered_df=filtered_df.groupby(['Launch Site','class']).size().reset_index(name='class_count')
        fig=px.pie(filtered_df,values='class_count',names='class',
            title=f"Total Success Launches for {entered_site} : {spacex_df[spacex_df['Launch Site']== entered_site]['class'].sum()}")
        return fig
```

```python
# TASK 4:
# Add a callback function for `site-dropdown` and `payload-slider` as inputs, `success-payload-scatter-chart` as output
@app.callback(Output(component_id='success-payload-scatter-chart',component_property='figure'),
                [Input(component_id='site-dropdown',component_property='value'),
                Input(component_id='payload-slider',component_property='value')])
def scatter(entered_site,payload):
    filtered_df = spacex_df[spacex_df['Payload Mass (kg)'].between(payload[0],payload[1])]
    # thought reusing filtered_df may cause issues, but tried it out of curiosity and it seems to be working fine

    if entered_site=='ALL':
        fig=px.scatter(filtered_df,x='Payload Mass (kg)',y='class',color='Booster Version Category',
            title=f"Success count on Payload mass between {payload[0]} and {payload[1]} kg for all sites: {spacex_df['class'].sum()}")
        return fig
    else:
        fig=px.scatter(filtered_df[filtered_df['Launch Site']==entered_site],
            x='Payload Mass (kg)',y='class',color='Booster Version Category',
                title=f"Success count on Payload mass between {payload[0]} and
                    {payload[1]} kg for {entered_site} : {filtered_df[filtered_df['Launch Site']== entered_site]['class'].sum()}")
        return fig

# Run the app
if __name__ == '__main__':
    app.run_server(port = 8090 ,dev_tools_ui=True, dev_tools_hot_reload =True, threaded=True)
```

- A dashboard was created to show a pie chart with successful landings from each launch site and a Payload vs Success scatter plot (with Booster Version in color detail).
- A dropdown list was created to filter the results
- A slider was created to filter Payload Mass.

Complete code on:

www.github.com/uriasmg



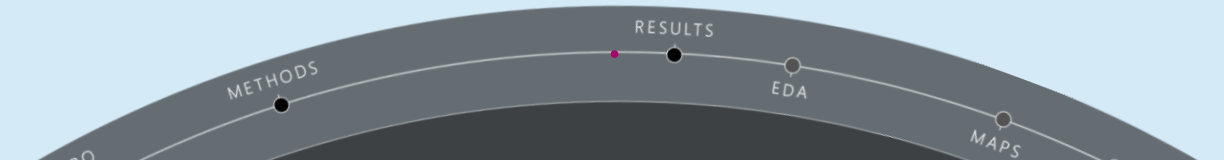
contato@drmuller.com.br

2 Predictive Analysis (Classification)

- Features: 'FlightNumber', 'PayloadMass', 'Orbit', 'LaunchSite', 'Flights', 'GridFins', 'Reused', 'Legs', 'LandingPad', 'Block', 'ReusedCount', 'Serial'
- One hot encoding for 'Orbit', 'LaunchSite', 'LandingPad', 'Serial'
- Train / Test Split → 80% : Training
- Standardization

- Logistic Regression (LR), Support Vector Machine (SVM), Decision Tree Classifier (DT), K-Nearest Neighbours (KNN)
- GridSearchCV to find best parameters
- All parameters will be described in the code
- Accuracy were verified using .score from GridSearch
- Results were included in a data frame and into a bar chart
- Inconsistencies were test at the end.
- Changes in suggested split size were tested to verify changes in accuracy.

Complete code on:
www.github.com/uriasmg



2 Predictive Analysis (Classification)

TASK 1

Create a NumPy array from the column `Class` in `data` , by applying the method `to_numpy()` then assign it to the variable `Y` ,make sure the output is a Pandas series (only one bracket df['name of column']).

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

TASK 2

Standardize the data in `X` then reassign it to the variable `X` using the transform provided below.

```
In [ ]: transform = preprocessing.StandardScaler()
```

```
In [ ]: X = transform.fit_transform(X)
```

We split the data into training and testing data using the function `train_test_split` . The training data is divided into validation data, a second set used for training data; then the models are trained and hyperparameters are selected using the function `GridSearchCV` .

TASK 3

Use the function train_test_split to split the data X and Y into training and test data. Set the parameter test_size to 0.2 and random_state to 2. The training data and test data should be assigned to the following labels.

`X_train, X_test, Y_train, Y_test`

```
In [ ]: X_train, X_test, Y_train, Y_test = train_test_split( X, Y, test_size=0.2, random_state=2) #shuffle=True
        print ('Train set:', X_train.shape,  Y_train.shape)
        print ('Test set:', X_test.shape,  Y_test.shape)
```

we can see we only have 18 test samples.

```
In [ ]: Y_test.shape
```

TASK 4

Create a logistic regression object then create a GridSearchCV object `logreg_cv` with cv = 10. Fit the object to find the best parameters from the dictionary `parameters` .

```
In [ ]: parameters_lr ={"C":[0.001,0.01,0.1,1],'penalty':['l2'], 'solver':['lbfgs']}# l1 lasso l2 ridge
        lr=LogisticRegression()
```

```
In [ ]: logreg = GridSearchCV(lr,parameters_lr,cv=10, refit=True)
        logreg.fit(X_train, Y_train)
```

TASK 5

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

```
In [ ]: logreg.fit(X_train, Y_train)
        print(f"LR Test set accuracy :{logreg.score(X_test, Y_test)}")
```

Lets look at the confusion matrix:

```
In [ ]: yhat_lr=logreg.predict(X_test)
        print('LR Confusion Matrix')
        plot_confusion_matrix(Y_test,yhat_lr)
```

Examining the confusion matrix, we see that logistic regression can distinguish between the different classes. We see that the major problem is false positives.

TASK 6

Create a support vector machine object then create a `GridSearchCV` object `svm_cv` with cv - 10. Fit the object to find the best parameters from the dictionary `parameters` .

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

```
In [ ]: parameters_svm = {'kernel':('linear', 'rbf','poly','rbf', 'sigmoid'),
                          'C': [0.001,0.01,0.1,1, 10, 100], 'gamma': [1,0.1,0.01,0.001]}
        svm = SVC()
```

```
In [ ]: svm_cv = GridSearchCV(svm,parameters_svm,cv=10, refit=True)
        svm_cv.fit(X_train, Y_train)
```

```
In [ ]: print(f"SVM GridSearch best hyperparameters: {svm_cv.best_params_}")
        print(f"SVM Accuracy: {svm_cv.best_score_}")
```

TASK 7

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

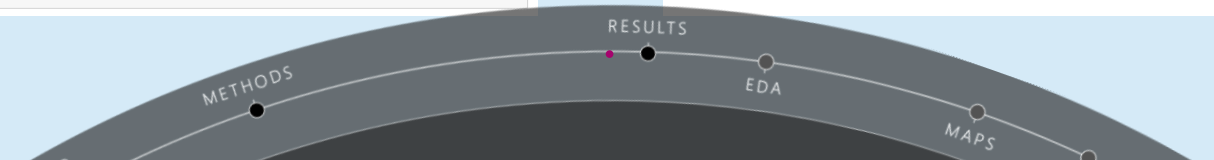
```
In [ ]: svm_cv.fit(X_train, Y_train)
        print(f"SVM Test set accuracy: {svm_cv.score(X_test, Y_test)}")
```

We can plot the confusion matrix

```
In [ ]: yhat_svm=svm_cv.predict(X_test)
        print('SVM Confusion Matrix')
        plot_confusion_matrix(Y_test,yhat_svm)
```

Complete code on:

www.github.com/uriasmg



contato@drmuller.com.br

2 Predictive Analysis (Classification)

TASK 8

Create a decision tree classifier object then create a `GridSearchCV` object `tree_cv` with cv = 10. Fit the object to find the best parameters from the dictionary `parameters`.

```
In [ ]: parameters_tree = {'criterion': ['gini', 'entropy'],
             'splitter': ['best', 'random'],
             'max_depth': [2*n for n in range(1,10)],
             'max_features': ['auto', 'sqrt'],
             'min_samples_leaf': [1, 2, 4],
             'min_samples_split': [2, 5, 10]}

        tree = DecisionTreeClassifier()
```

```
In [ ]: tree_cv = GridSearchCV(tree,parameters_tree,cv=10, refit=True)
        tree_cv.fit(X_train, Y_train)
```

```
In [ ]: print(f"Decision Tree GridSearch best hyperparameters: {tree_cv.best_params_}")
        print(f"Decision Tree Accuracy: {tree_cv.best_score_}")
```

TASK 9

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

```
In [ ]: tree_cv.fit(X_train, Y_train)
        print(f"Decision Tree Test set accuracy: {tree_cv.score(X_test, Y_test)}")
```

We can plot the confusion matrix

```
In [ ]: yhat_tree = tree_cv.predict(X_test)
        print('Decision Tree Confusion Matrix')
        plot_confusion_matrix(Y_test,yhat_tree)
```

TASK 10

Create a k nearest neighbors object then create a `GridSearchCV` object `knn_cv` with cv = 10. Fit the object to find the best parameters from the dictionary `parameters`.

```
In [ ]: parameters_knn = {'n_neighbors': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
                        'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute'],
                        'p': [1,2]}

        KNN = KNeighborsClassifier()
```

```
In [ ]: knn_cv = GridSearchCV(KNN,parameters_knn,cv=10, refit=True)
        knn_cv.fit(X_train, Y_train)
```

```
In [ ]: print(f"KNN GridSearch best hyperparameters: {knn_cv.best_params_}")
        print(f"KNN Accuracy: {knn_cv.best_score_}")
```

TASK 11

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

```
In [ ]: knn_cv.fit(X_train, Y_train)
        print(f"KNN Test set accuracy: {knn_cv.score(X_test, Y_test)}")
```

We can plot the confusion matrix

```
In [ ]: yhat_knn = knn_cv.predict(X_test)
        print('KNN Confusion Matrix')
        plot_confusion_matrix(Y_test,yhat_knn)
```

TASK 12

Find the method performs best:

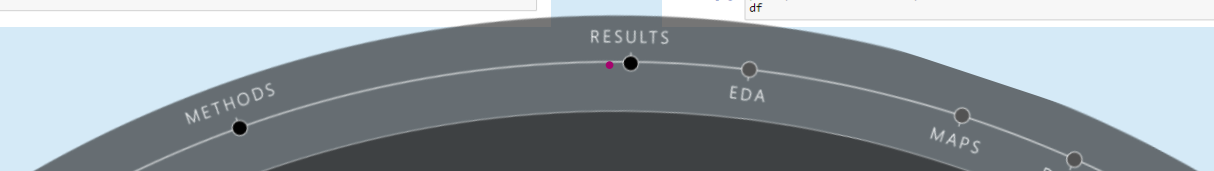
```
In [ ]: print('Training Scores')
        print("LR Accuracy :",logreg.best_score_)
        print(f"SVM Accuracy: {svm_cv.best_score_}")
        print(f"Decision Tree Accuracy: {tree_cv.best_score_}")
        print(f"KNN Accuracy: {knn_cv.best_score_}")
```

```
In [ ]: # Report table
        print('\nSplit train/test using train set of 80%. Run #1')
        test_scores = []
        for classifier_name, cv, yhat in [
            ('K-Nearest Neighbours', knn_cv, yhat_knn),
            ('Decision Tree', tree_cv, yhat_tree),
            ('SVM', svm_cv, yhat_svm),
            ('Logistic Regression', logreg, yhat_lr)
        ]:
            scores = {
                'Algorithm': classifier_name,
                'Accuracy': cv.score(X_test, Y_test),
                #'Accuracy(f1)': f1_score(Y_test, yhat, average='weighted')
            }
            test_scores.append(scores)
        df=pd.DataFrame(test_scores)

        #df.set_index('Algorithm').plot(kind='bar')
        #plt.xticks(rotation=315, horizontalalignment="left")
        #plt.title('Prediction Accuracy')
        #plt.ylabel('Accuracy(score)')
        #plt.xlabel('Algorithm')

        #plt.show()
```

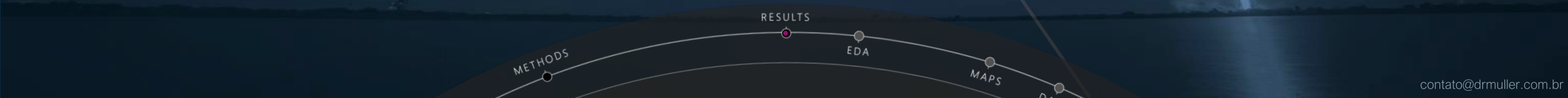
```
In [ ]: print('Perform on tests:')
        df
```

Complete code on:

www.github.com/uriasmg



3 Re·sults

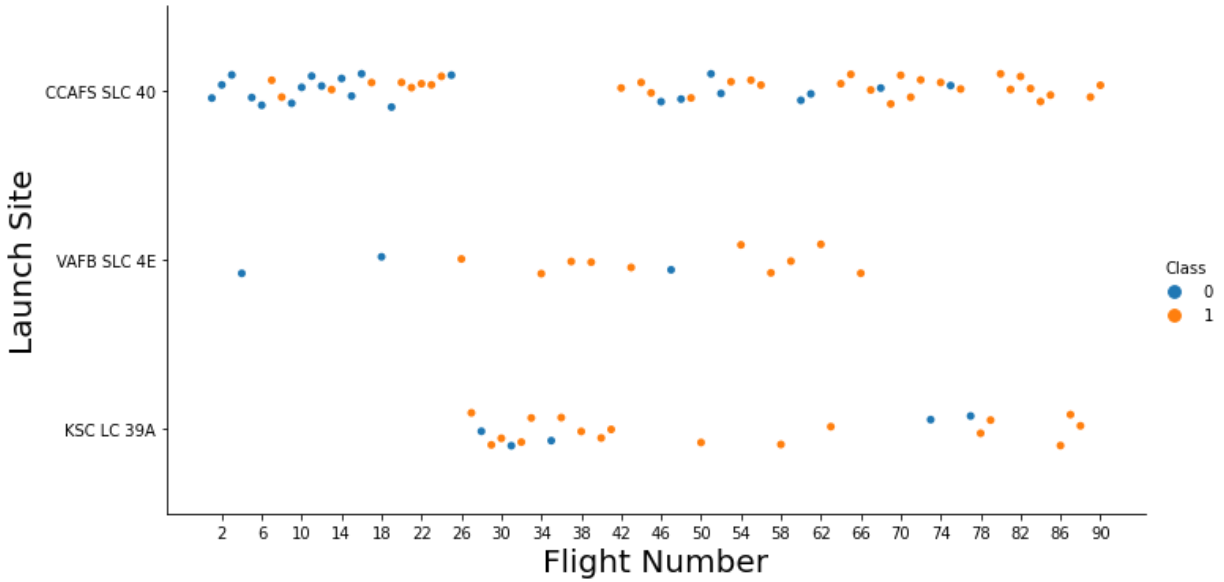


contato@drmuller.com.br

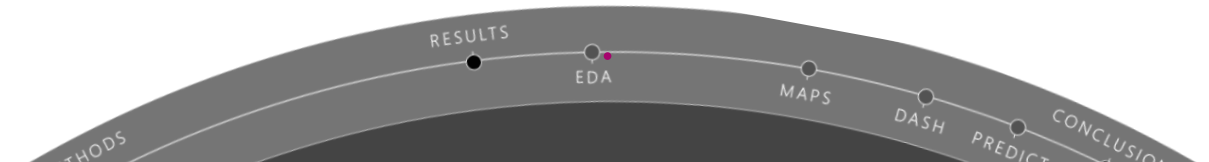
3.1 Insights from Exploratory Data Analysis (EDA)

contato@drmuller.com.br

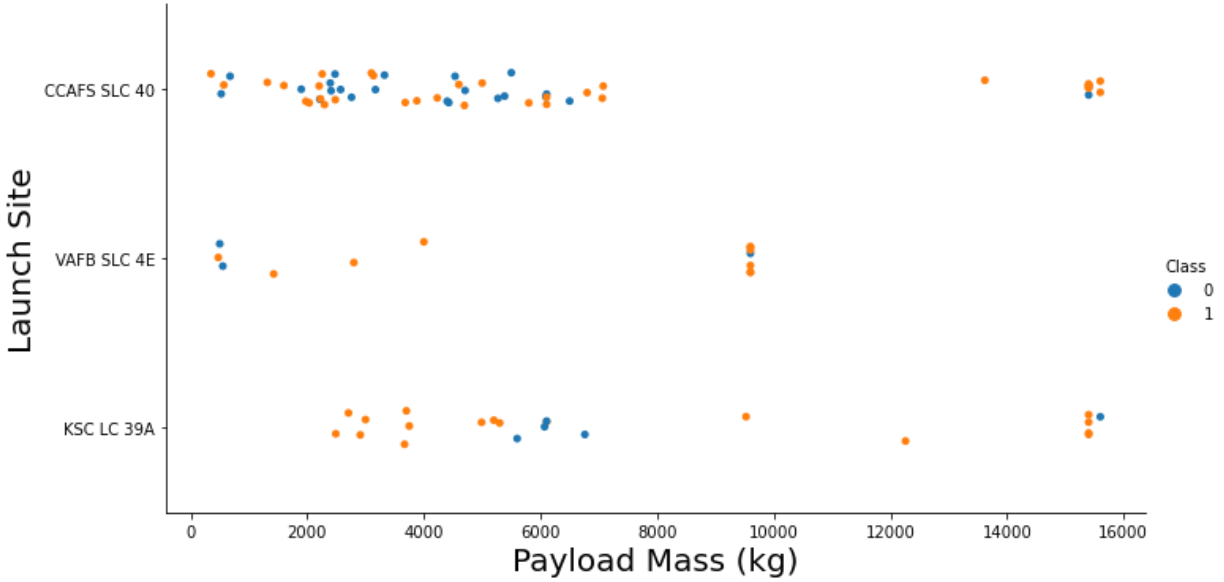
3.1 Insights from EDA
Flight Number vs. Launch Site



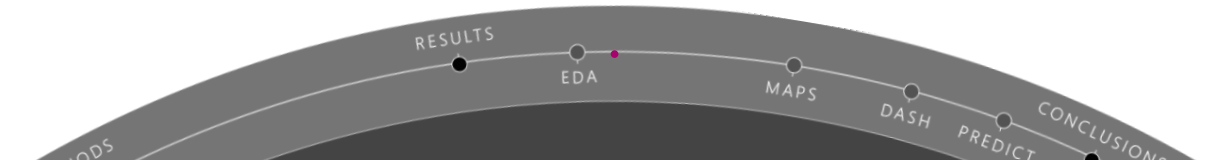
- **‘CCAFS SLC 40’** had **more overall flights**
- Site **‘KSC LC 39A’** started to be **used after 25th flight**
- **All** sites **show progression in class 1 flights** (successful outcomes)



3.1 Insights from EDA
Payload vs. Launch Site

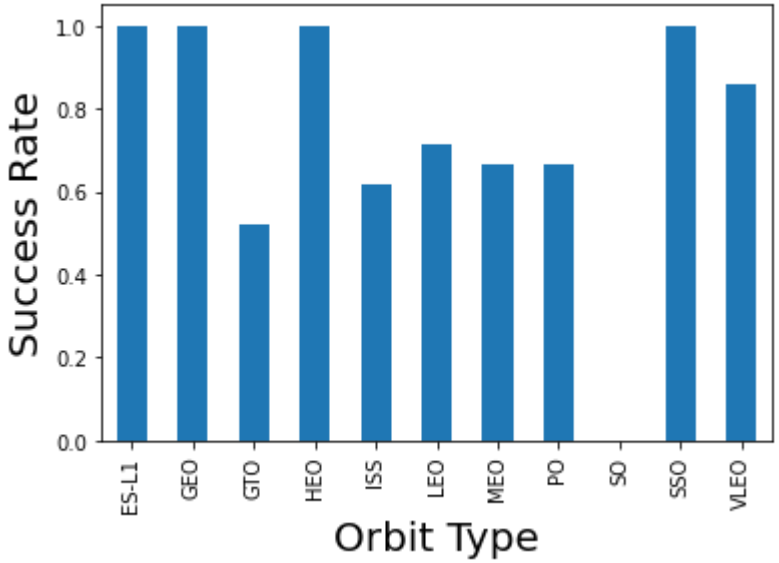


- **‘VAFB-SLC’** site : **no rockets launched** for heavy payload mass (**greater than 10000kg**)
- **Most launches were up to 7000kg**
- **Both KSC LC 39A and CCAFS SLC 40** were able to **handle launches for payloads around 15000kg**

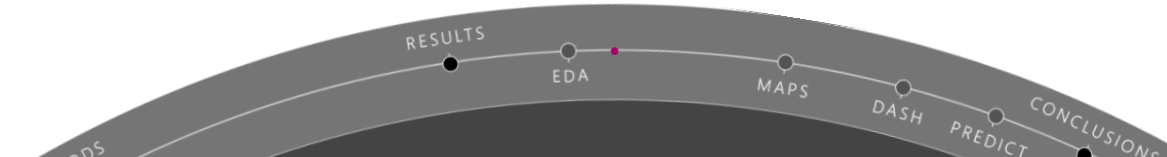


contato@drmuller.com.br

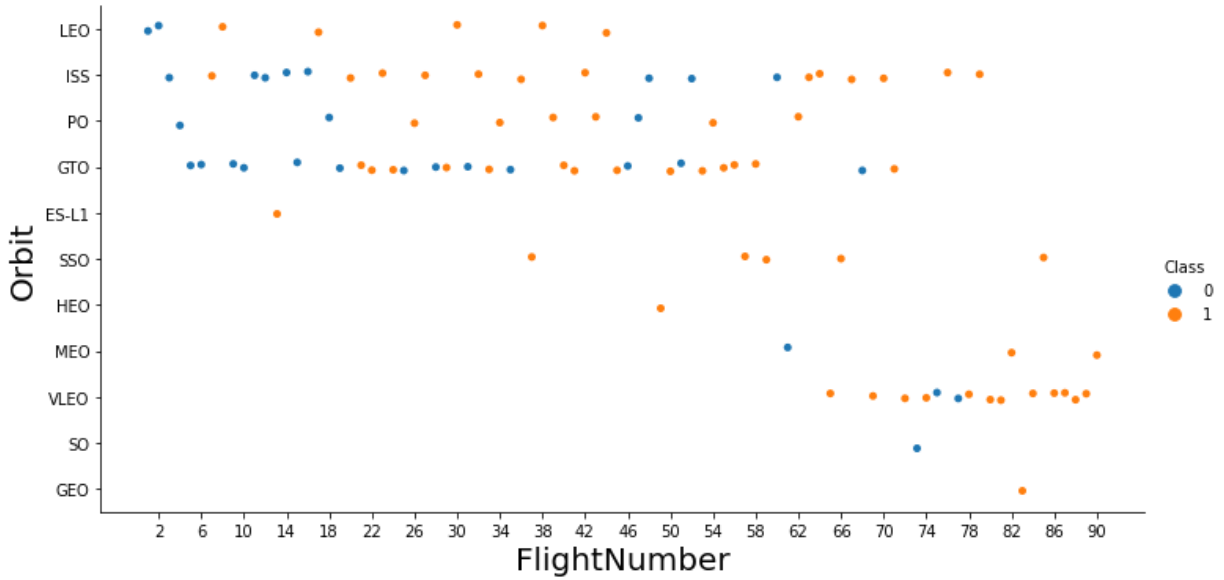
3.1 Insights from EDA
Success Rate vs. Orbit Type



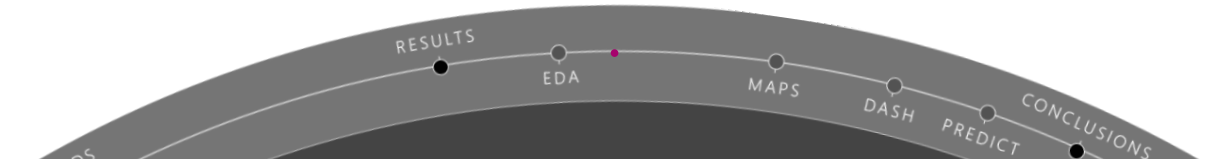
- **‘ES-L1’, ‘GEO’, ‘HEO’, ‘SSO’** orbit types had 100% success rate
- **‘GTO’** orbit type **had a success rate around 50%**
- **‘SO’** orbit type **had no success** in its only launch



3.1 Insights from EDA
Flight Number vs. Orbit Type

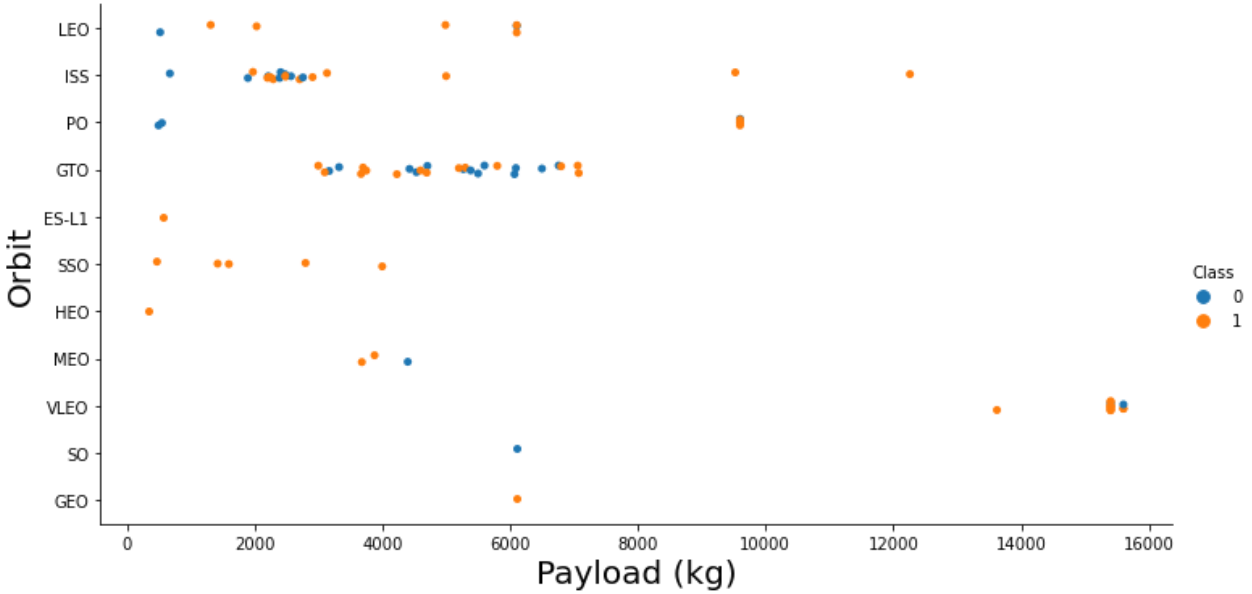


- For **LEO orbit the success** appears **to be related to the number of flights**;
- There seems to be **no relationship** between **flight number when in GTO orbit**
- **SO and GEO orbits** were **tested** in **later flights,** probably related to challenges related

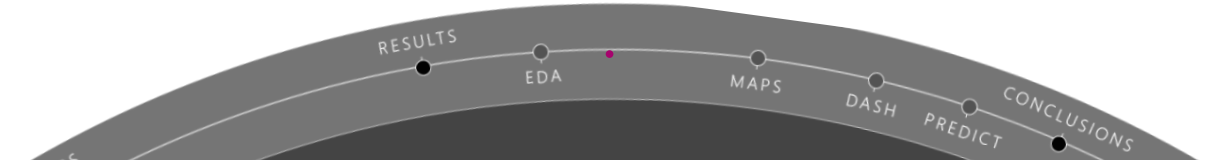


contato@drmuller.com.br

3.1 Insights from EDA
Payload vs. Orbit Type

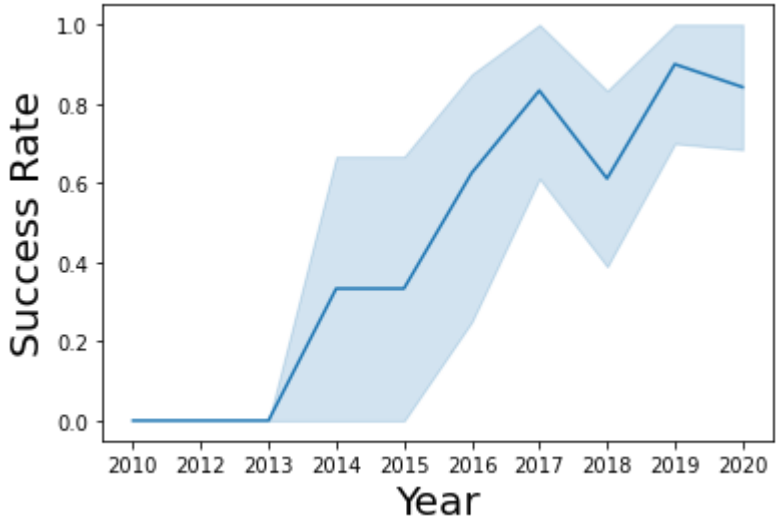


- With **heavy payloads** the **successful landing are more for Polar, LEO and ISS orbits.**
- For GTO we cannot associate the payload mass with a successful or unsuccessful landing

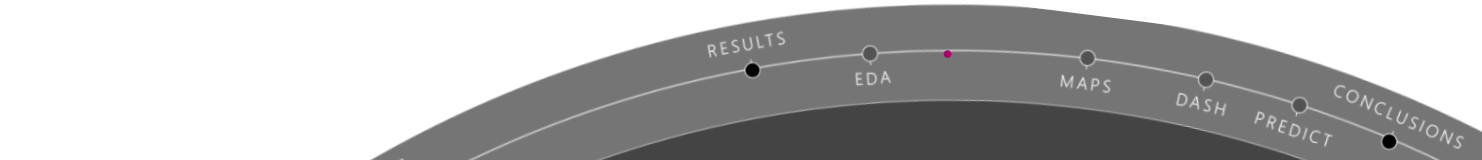


3.1 Insights from EDA

Launch Success Yearly Trend



- The **success rate has increased from 2013 until 2020**
- In 2018, success rate returned to be similar to 2016 rates, just before restarted to grow again in 2019.



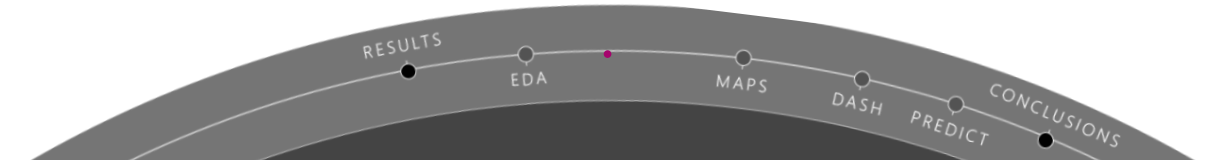
3.1 Insights from EDA
All Launch Site Names

```
In [12]: %sql SELECT DISTINCT LAUNCH_SITE FROM SPACEX;
 * ibm_db_sa://ncg11739:***@fbd88901-ebdb-4a
b
Done.
```

Out[12]:

launch_site
CCAFS LC-40
CCAFS SLC-40
KSC LC-39A
VAFB SLC-4E

- Launch Site names:
  - CCAFS LC-40
  - CCAFS SLC-40
  - KSC LC-39A
  - VAFB SLC-4E



3.1 Insights from EDA

Launch Site Names Begin with 'CCA'

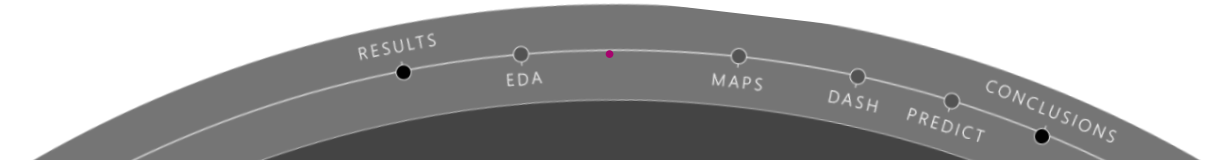
```
In [13]: %sql SELECT * FROM SPACEX WHERE LAUNCH_SITE LIKE 'CCA%' LIMIT 5;
```

```
 * ibm_db_sa://ncg11739:***@fbd88901-ebdb-4a4f-a32e-9822b9fb237b.c1ogj3sd0tgtu0lqde00.databases.appdomain.cloud:32731/bludb
Done.
```

Out[13]:

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	07:44:00	F9 v1.0 B0005	CCAFS LC-40	Dragon demo flight C2	525	LEO (ISS)	NASA (COTS)	Success	No attempt
2012-10-08	00:35:00	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

Here, we specify site names starting with CCA and limit them to 5 results



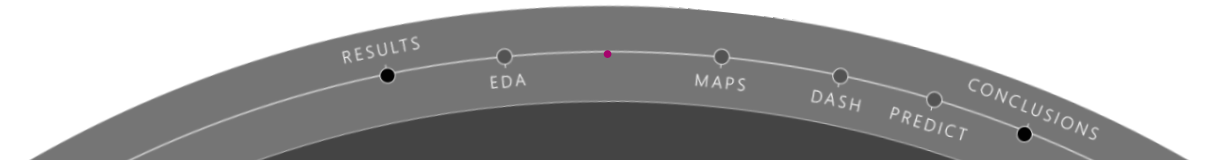
3.1 Insights from EDA
Total Payload Mass

```
In [16]: %sql SELECT SUM(payload_mass__kg_) as Total_Payload_Mass FROM SPACEX WHERE CUSTOMER = 'NASA (CRS)';
          * ibm_db_sa://ncg11739:***@fbd88901-ebdb-4a4f-a32e-9822b9fb237b.c1ogj3sd0tgtu0lqde00.databases.a
         b
         Done.
```

Out[16]:

total_payload_mass
45596

- Here, we sum the payload mass, give it a name and specify the customer, in this case, NASA (CRS), returning the value 45596 kg.
- If we do not specify, it will return the total payload mass from the database, in this case, the sum would be 61967 kg.



3.1 Insights from EDA
Average Payload Mass by F9 v1.1

```
In [22]: # Average payload mass carried by booster version F9 v1.1 (any)
         %sql SELECT AVG(payload_mass__kg_) average_payload FROM SPACEX WHERE BOOSTER_VERSION LIKE 'F9 v1.1%';
          * ibm_db_sa://ncg11739:***@fbd88901-ebdb-4a4f-a32e-9822b9fb237b.c1ogj3sd0tgtu0lqde00.databases.app
         b
         Done.
```

Out[22]:

average_payload
2534

```
In [23]: # Average payload mass carried by booster version F9 v1.1 (exactly)
         %sql SELECT AVG(payload_mass__kg_) average_payload FROM SPACEX WHERE BOOSTER_VERSION = 'F9 v1.1';
          * ibm_db_sa://ncg11739:***@fbd88901-ebdb-4a4f-a32e-9822b9fb237b.c1ogj3sd0tgtu0lqde00.databases.app
         b
         Done.
```

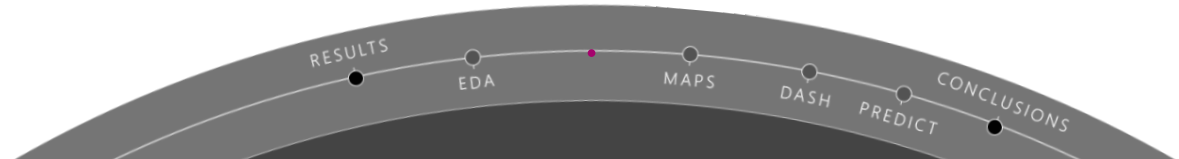
Out[23]:

average_payload
2928

- The booster F9 v1.1 had other versions related in the database, such as ‘F9 v1.1’, ‘F9 v1.1 B1011’, ‘F9 v1.1 B1010’, etc and could be displayed with the following command:

*%sql SELECT * FROM SPACEX WHERE BOOSTER_VERSION LIKE 'F9 v1.1%';*

- Therefore, we could verify the average payload for only ‘F9 v1.1’ (exactly), or for all versions that include ‘F9 v1.1’ in their name (any), showed in this Figure (left).



3.1 Insights from EDA
First Successful Ground Landing Date

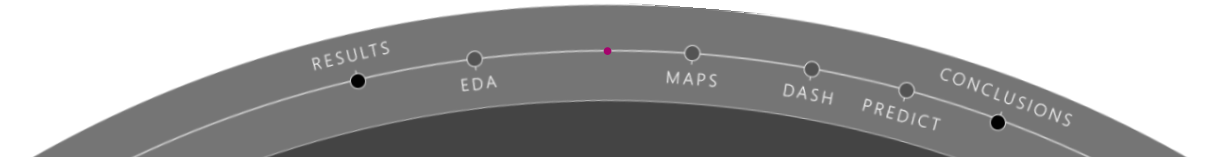
In [25]:
```
%%sql
SELECT MIN(date) first_successful_landing_date FROM SPACEX WHERE LANDING__OUTCOME LIKE 'Success%';
```

```
 * ibm_db_sa://ncg11739:***@fbd88901-ebdb-4a4f-a32e-9822b9fb237b.c1ogj3sd0tgtu0lqde00.databases.appd
b
Done.
```

Out[25]:

first_successful_landing_date
2015-12-22

- Here, we perform a query with the minimal data where the landing outcome include success.



3.1 Insights from EDA

Successful Drone Ship Landing with Payload between 4000 and 6000

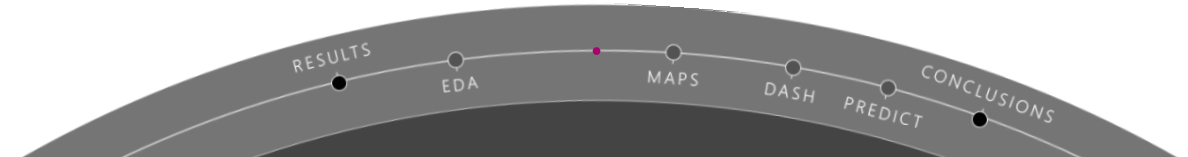
In [27]:
```
%%sql
SELECT DISTINCT BOOSTER_VERSION FROM SPACEX
    WHERE LANDING__OUTCOME = 'Success (drone ship)' AND 4000 < payload_mass__kg_ < 6000;
```

```
 * ibm_db_sa://ncg11739:***@fbd88901-ebdb-4a4f-a32e-9822b9fb237b.c1ogj3sd0tgtu0lqde00.da
b
Done.
```

Out[27]:

booster_version
F9 B4 B1042.1
F9 B4 B1045.1
F9 B5 B1046.1
F9 FT B1029.2
F9 FT B1021.1
F9 FT B1023.1
F9 FT B1038.1

- Here, we perform a query selecting distinct booster names with success as a landing outcome and payload mass between 4000 and 6000 kg.



3.1 Insights from EDA

Total Number of Successful and Failure Mission Outcomes

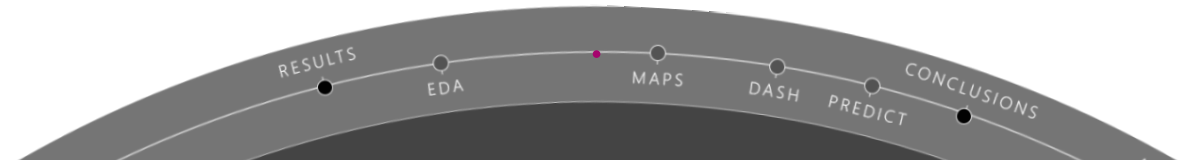
In [12]:
```
%%sql
SELECT mission_outcome, count(mission_outcome) total_number FROM SPACEX group by mission_outcome;
```

```
 * ibm_db_sa://ncg11739:***@fbd88901-ebdb-4a4f-a32e-9822b9fb237b.c1ogj3sd0tgtu0lqde00.databases.a
b
Done.
```

Out[12]:

mission_outcome	total_number
Failure (in flight)	1
Success	99
Success (payload status unclear)	1

- In this query, we select mission outcome and the count of them and grouped them by the outcomes.



contato@drmuller.com.br

3.1 Insights from EDA

Boosters Carried Maximum Payload

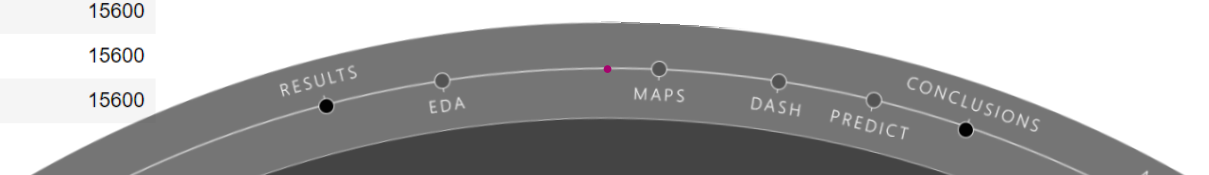
In [28]:
```
%%sql
SELECT booster_version, payload_mass__kg_ FROM SPACEX
    WHERE payload_mass__kg_ =(
        SELECT MAX(payload_mass__kg_) FROM SPACEX
        );
```

```
 * ibm_db_sa://ncg11739:***@fbd88901-ebdb-4a4f-a32e-9822b9fb2
b
Done.
```

Out[28]:

booster_version	payload_mass__kg_
F9 B5 B1048.4	15600
F9 B5 B1049.4	15600
F9 B5 B1051.3	15600
F9 B5 B1056.4	15600
F9 B5 B1048.5	15600
F9 B5 B1051.4	15600
F9 B5 B1049.5	15600
F9 B5 B1060.2	15600
F9 B5 B1058.3	15600
F9 B5 B1051.6	15600
F9 B5 B1060.3	15600
F9 B5 B1049.7	15600

- In this query, we perform a sub-query to verify the maximum payload.
- Then, we select booster versions and the payload mass (just to illustrate), where their payload mass is the value from the sub-query.



3.1 Insights from EDA
2015 Launch Records

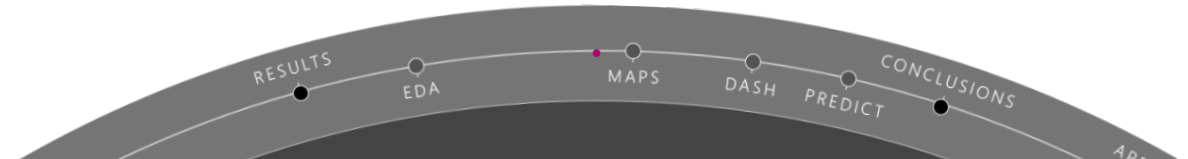
In [14]:
```
%%sql
SELECT LANDING__OUTCOME, BOOSTER_VERSION, LAUNCH_SITE
FROM SPACEX
WHERE Landing__Outcome = 'Failure (drone ship)'
    AND YEAR(DATE) = 2015;
```

```
 * ibm_db_sa://ncg11739:***@fbd88901-ebdb-4a4f-a32e-9822b9
b
Done.
```

Out[14]:

landing__outcome	booster_version	launch_site
Failure (drone ship)	F9 v1.1 B1012	CCAFS LC-40
Failure (drone ship)	F9 v1.1 B1015	CCAFS LC-40

- For this query, landing outcome, booster version and launch site were selected, filtering for failed landings in 2015.



contato@drmuller.com.br

3.1 Insights from EDA

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

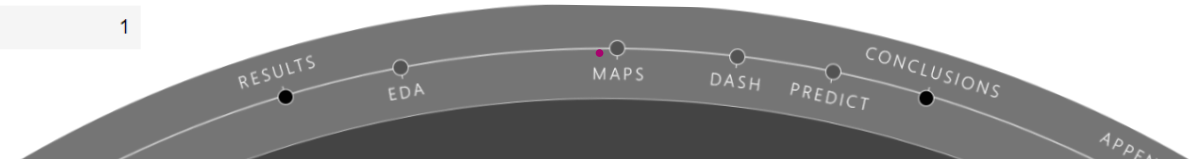
```
In [29]: %%sql
SELECT LANDING__OUTCOME, COUNT(LANDING__OUTCOME) TOTAL_NUMBER FROM SPACEX
    WHERE DATE BETWEEN '2010-06-04' AND '2017-03-20'
    GROUP BY LANDING__OUTCOME
    ORDER BY TOTAL_NUMBER DESC;
```

```
 * ibm_db_sa://ncg11739:***@fbd88901-ebdb-4a4f-a32e-9822b9fb237b.c1ogj3sd0tgtu0
b
Done.
```

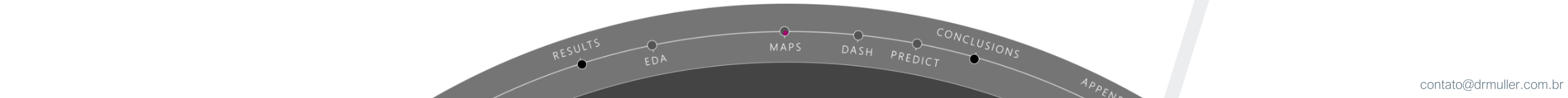
Out[29]:

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

- Here, we perform 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.



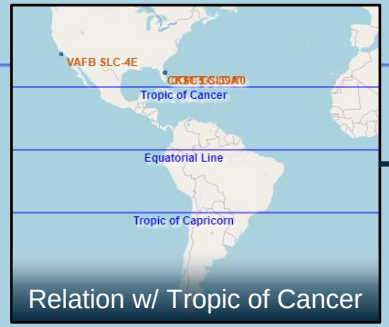
3.2
Launch Sites Proximities Analysis



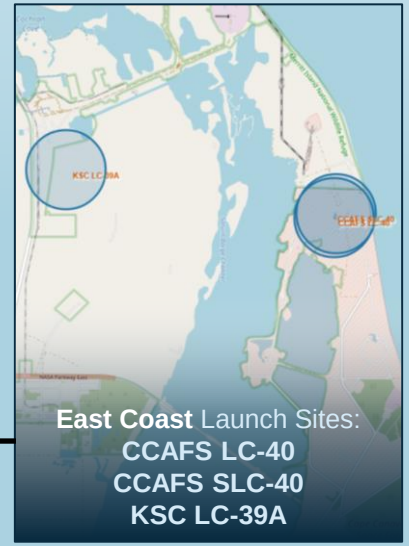
contato@drmuller.com.br

3.2 Launch Site Proximities Analysis

West Coast Launch Site:
VAFB SLC-4E



Relation w/ Tropic of Cancer



East Coast Launch Sites:
CCAFS LC-40
CCAFS SLC-40
KSC LC-39A

RESULTS EDA MAPS DASH PREDICT CONCLUSIONS APPENDIX

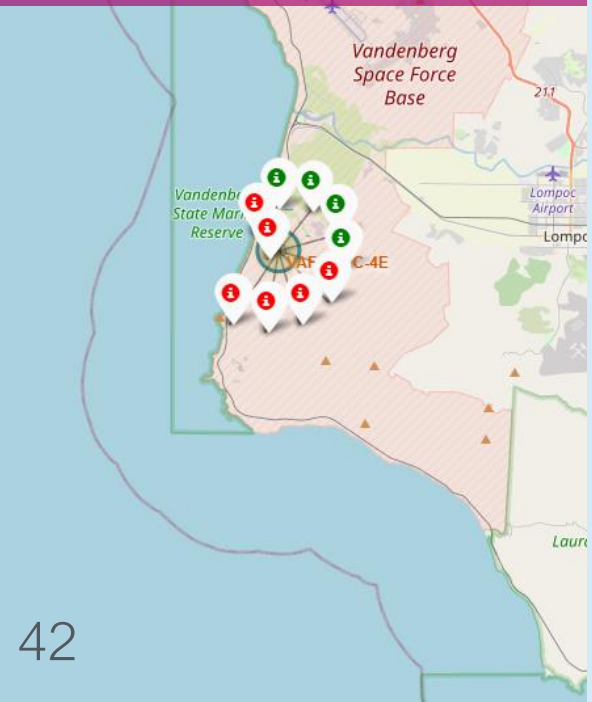
contato@drmuller.com.br

3.2 Launch Site Proximities Analysis

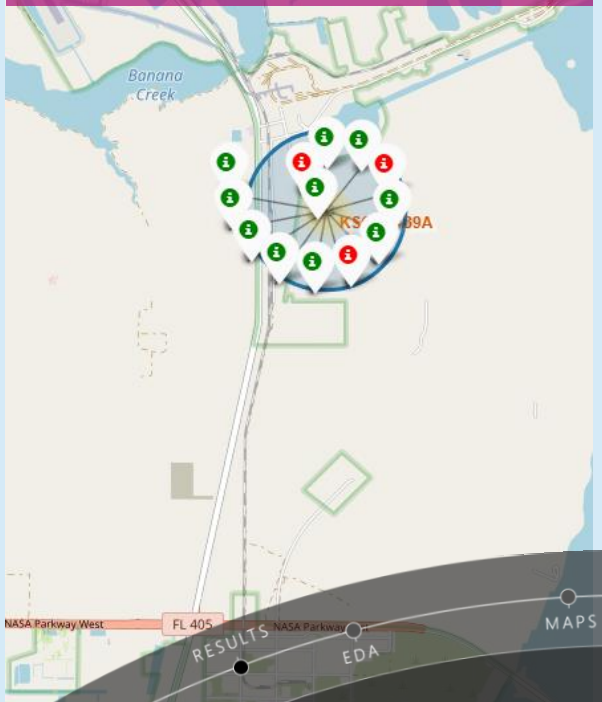
Launch Outcomes / Site

- **KSC LC-39A was the most successful site with regards to** outcomes (numerical and percentages)
- **CCAFS LC-40 had the most numerical launches** among the four launch sites
- **CCAFS SLC-40 had the least numerical launches** among the four launch sites

VAFB SLC-4E



KSC LC-39A



CCAFS LC-40

CCAFS SLC-40

Successful outcome
Failed outcome

contato@drmuller.com.br

3.2 Launch Site Proximities Analysis

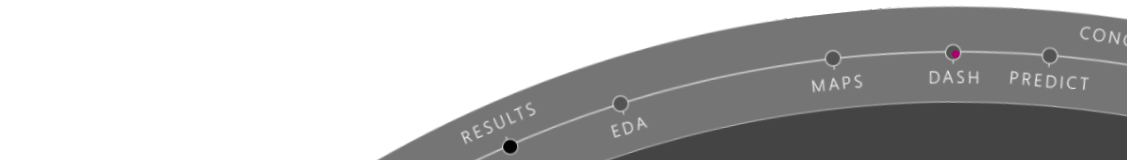
KSC LC-39A Analysis

Distance to...	
Coastline :	6.57 Km
Highway :	0.81 Km
Railroad :	0.66 Km
Closest city :	14.73 Km

Outcomes

contato@drmuller.com.br

3.3
Dashboard with Plotly Dash



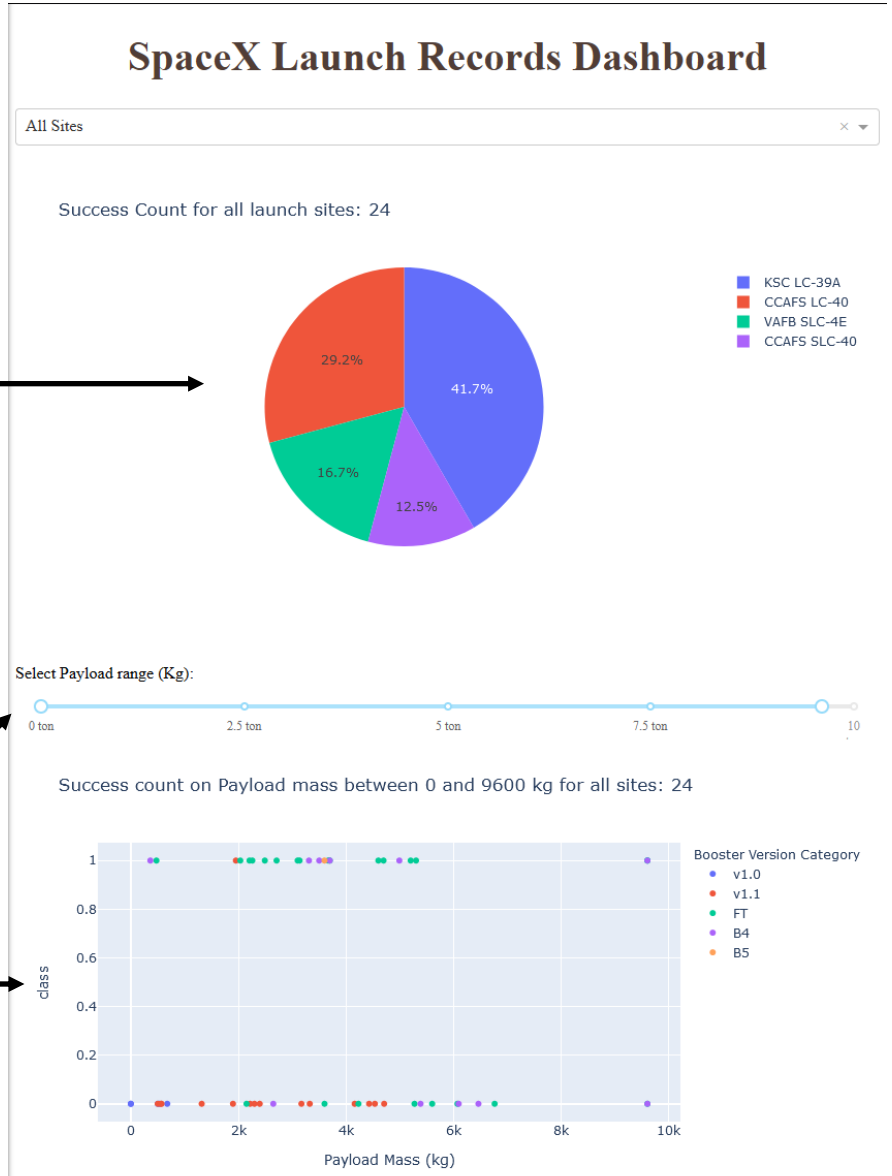
3.3 Dashboard Components

Drop menu

Pie chart

Slide input

Scatter plot

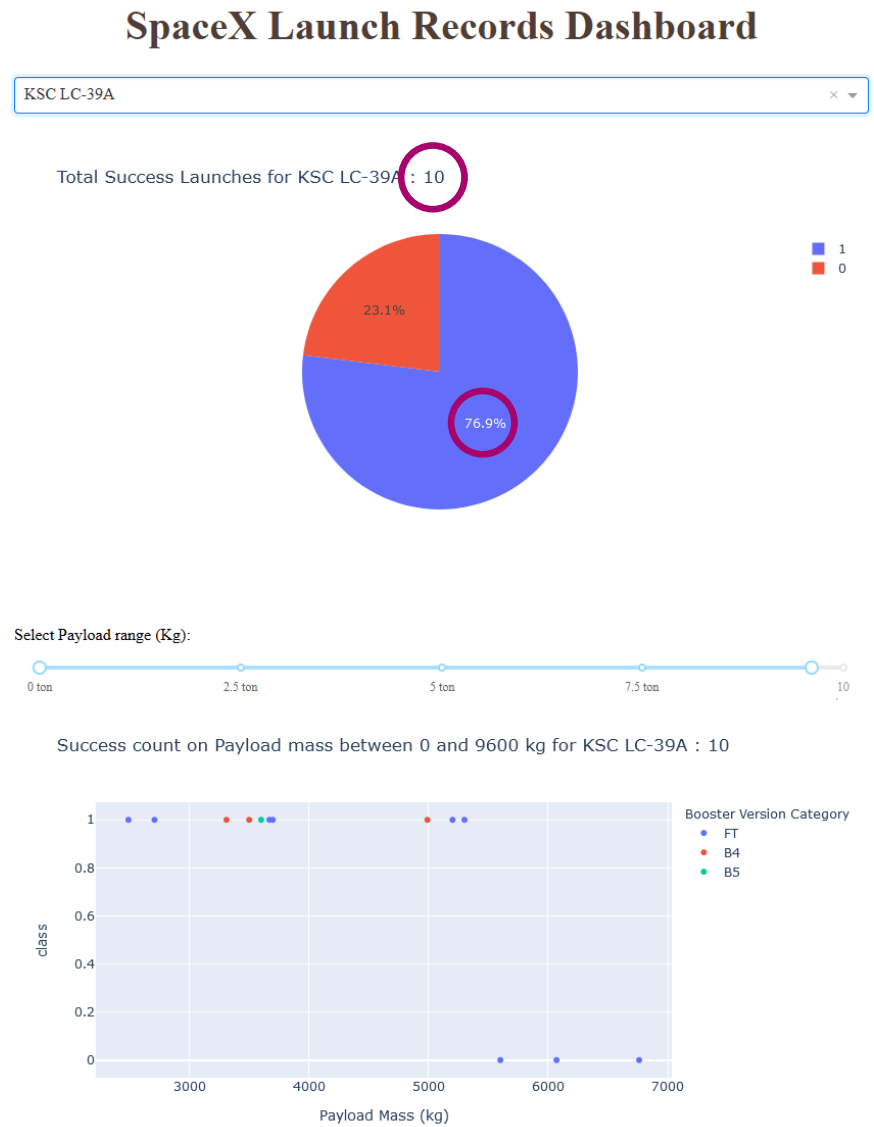


Code available on : github.com/uriasmg

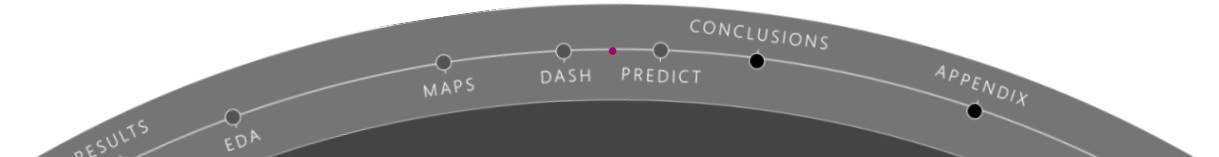
3.3 Dashboard

KSC LC-39A: highest launch success rate

- **10 successful outcomes**
- **76.9% overall**
- No success on heavy payloads (over 5500 kg)

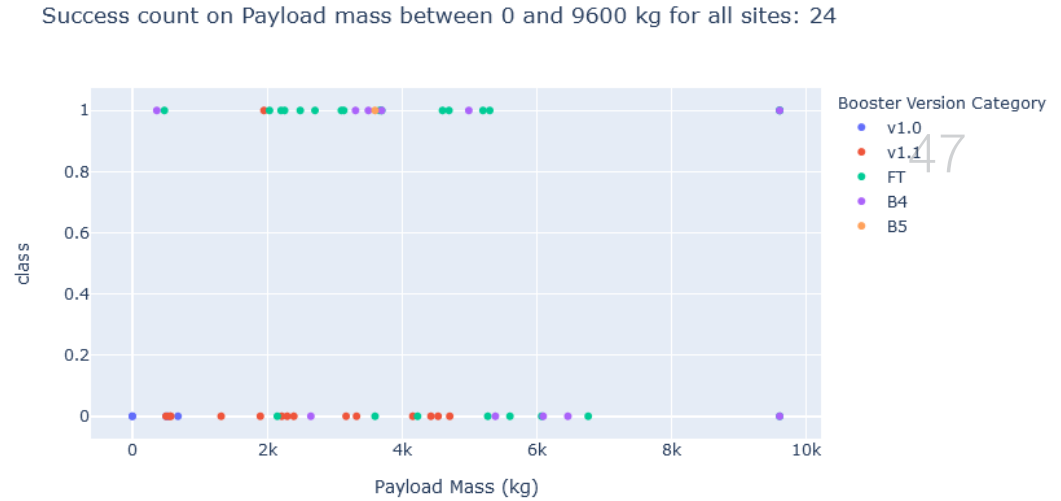


Code available on : github.com/uriasmg

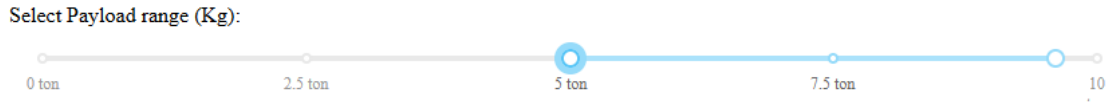


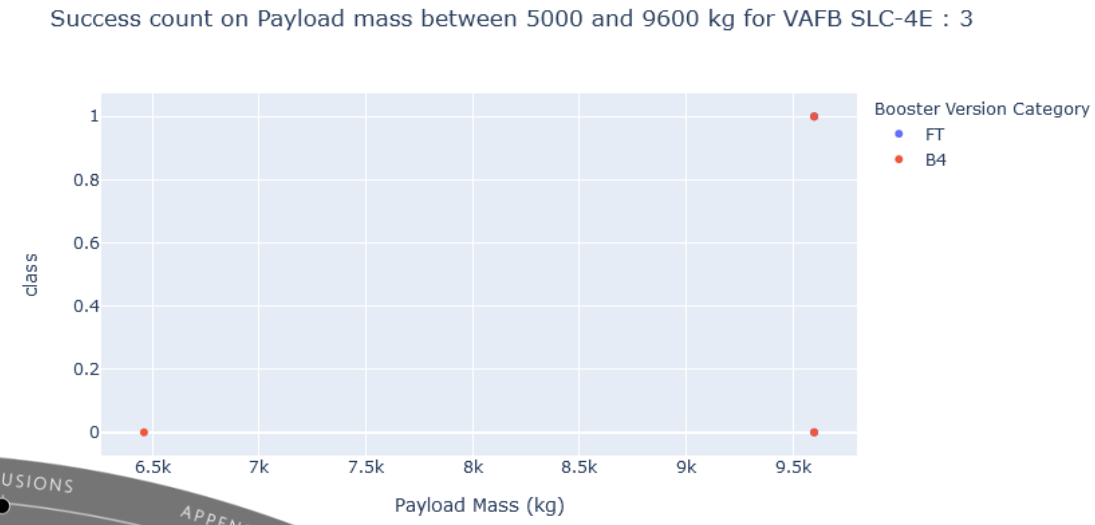
contato@drmuller.com.br

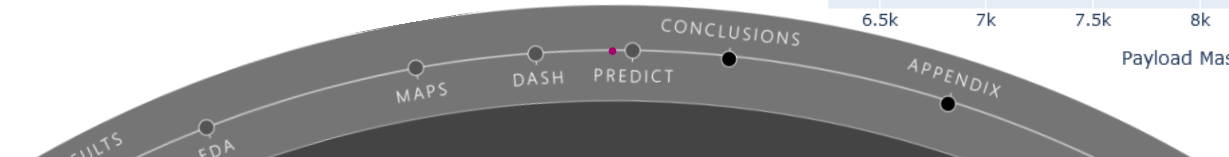
3.3 Dashboard
Payload vs. Launch Outcome



- **FT and B4 boosters were capable of success carrying payloads over 5000 kg AND UP TO 10000.**
- **Only B4 had success carrying payloads over 6000 kg**
- Only **VAFB SLC-4E were able to success with such heavy payload (over 6000) in this sample(did not include payloads over 10000kg)**
- **VAFB SLC-4E had 33% success rate in payloads between 5000 kg and 10000 kg**







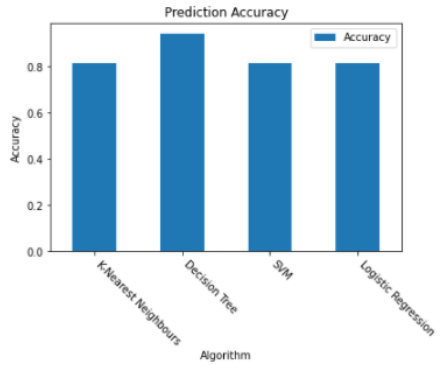
contato@drmuller.com.br

3.4
Predictive Analysis
(Classification)

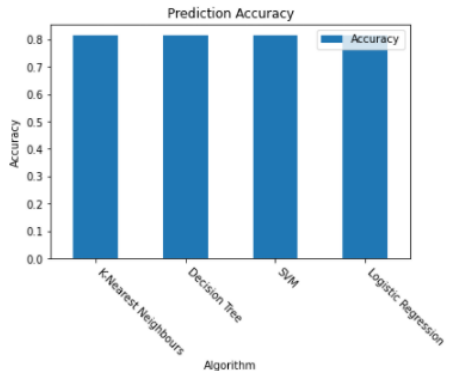
3.4 Predictive Analysis Classification Accuracy

- Using a *train / test split of 80:20 ,* ***d*uring training and validation, the decision tree model had the highest accuracy** (88.93%), followed by Support Vector Machine (SVM) (86.25%).
- **During testing**, however, the **accuracy was similar between models** (83,33%)
- **After a number of runs, Decision Tree showed to have more inconsistencies than other models** (bar charts on left)
- **A plot was performed to check decision tree inconsistency along runs and metrics were obtained** (bottom right)

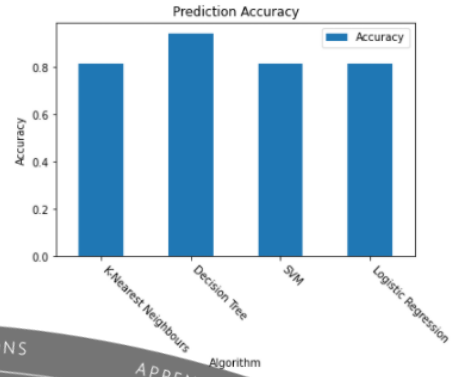
Split train/test using train set of 80%. Run #1



Split train/test using train set of 80%. Run #2



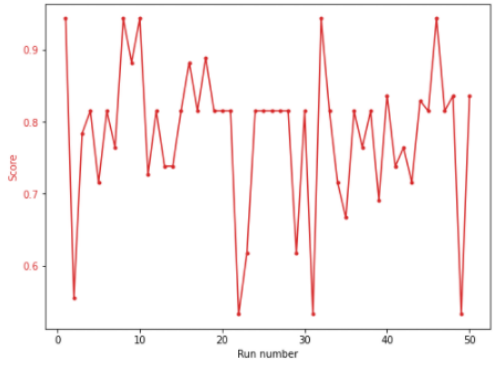
Split train/test using train set of 80%. Run #3

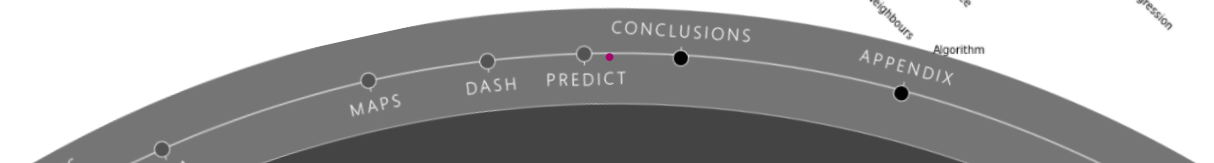


	Algorithm	Accuracy
0	K-Nearest Neighbours	0.833333
1	Decision Tree	0.833333
2	SVM	0.833333
3	Logistic Regression	0.833333

Accuracy of Decision Tree using GridSearchCV along runs

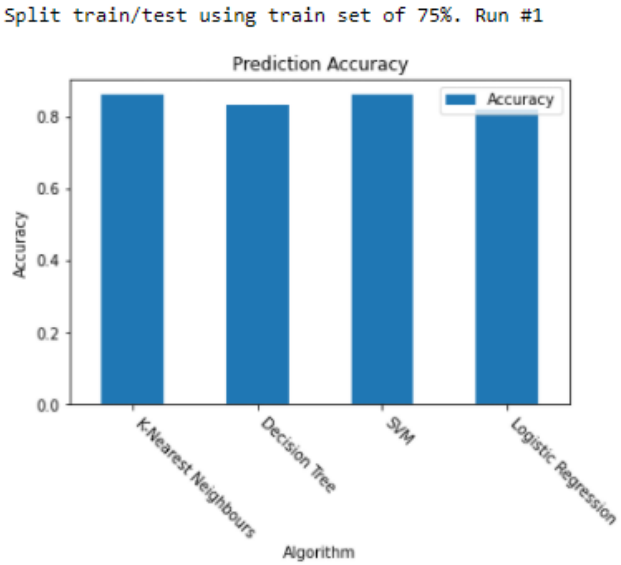
Mean Accuracy after 50 runs (Train test split of 80:20): 0.7820
Median Accuracy after 50 runs (Train test split of 80:20): 0.8148

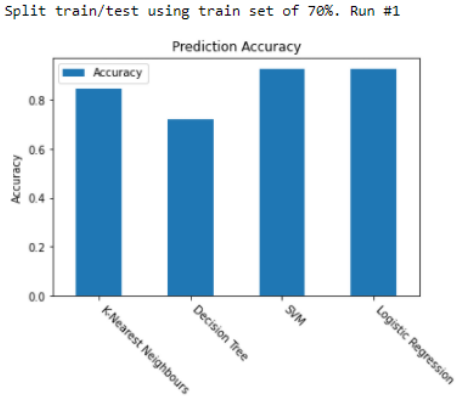




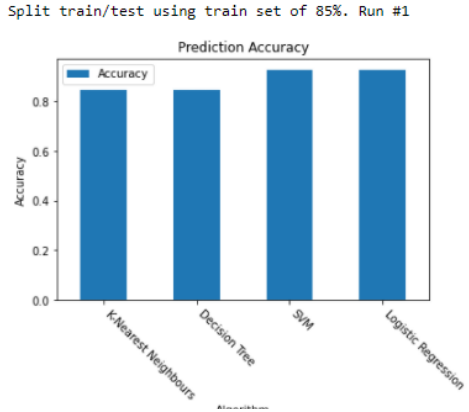
3.4 Predictive Analysis Classification Accuracy

- Using a *train / test split of 75:25 ,* **during training and validation, the decision tree model had again the highest accuracy** (88.09%), followed by K-Nearest Neighbours (KNN) (85.00%).
- **During testing**, however, the **accuracy was similar between KNN and SVM** (86,04%) (Bar chart on left)
- Again, **Decision Tree showed inconsistencies** and a **plot was performed** (top right)
- A train / test split of 70:30 and of 85:15 were also tested, with results on bottom.
- Reduction in test set, increased the risk of overfitting data. Reducing training, results could be more inconsistent.

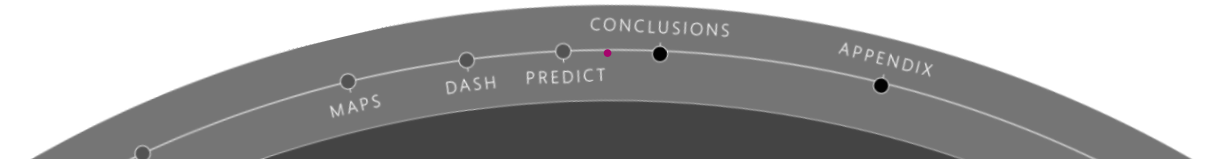




	Algorithm	Accuracy
0	K-Nearest Neighbours	0.84643
1	Decision Tree	0.72024
2	SVM	0.92648
3	Logistic Regression	0.92648



	Algorithm	Accuracy
0	K-Nearest Neighbours	0.84643
1	Decision Tree	0.84643
2	SVM	0.92648
3	Logistic Regression	0.92648

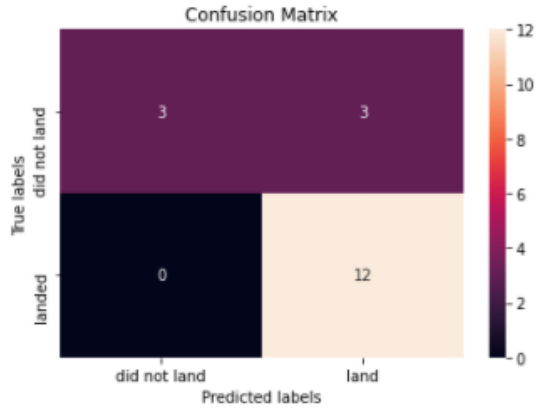


contato@drmuller.com.br

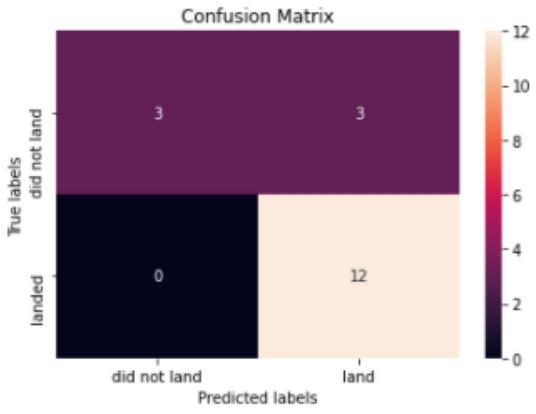
3.4 Predictive Analysis
Confusion Matrix

- For our tests, for the given parameters, such as train set size, confusion matrix were similar among models (except when DT Classifier had inconsistencies).

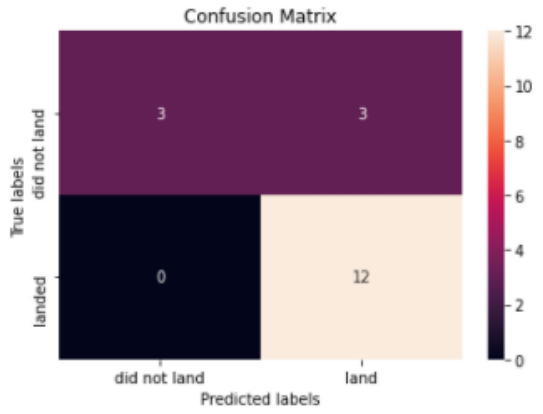
LR Confusion Matrix



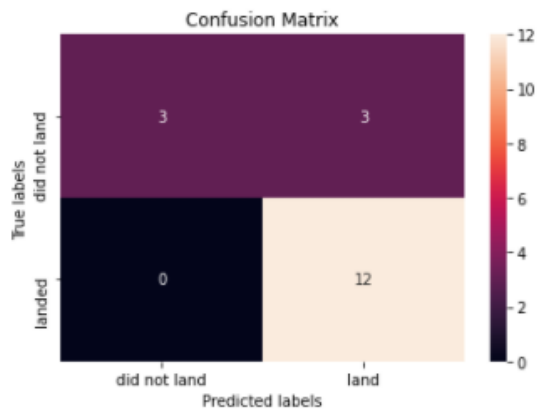
SVM Confusion Matrix

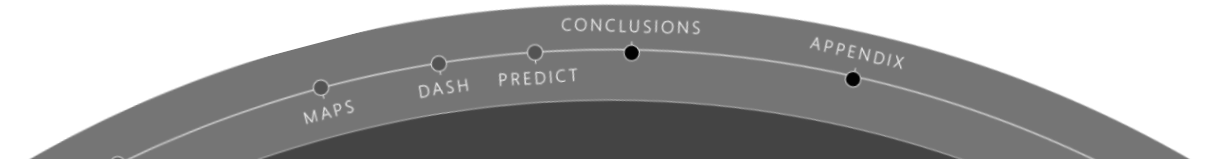


Decision Tree Confusion Matrix

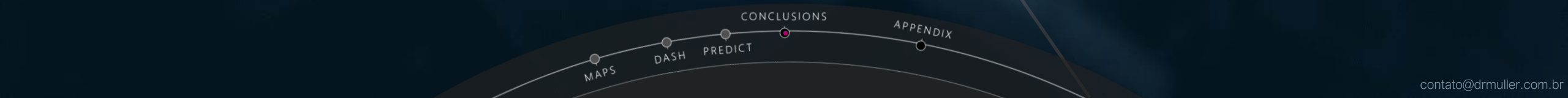


KNN Confusion Matrix





4
Con·clu·sions



4
Con·clu·sions

- KSC launch site is more favorable to more success on reduced payloads.
- For heaviest payloads, F9 B5 boosters and the CCAFS SLC-40 launch site were related to more successful outcomes, and Polar, LEO and ISS orbits are more prone to successful landings.
- Using our data, we were able to predict 83% of successful landings.
- Therefore, it is possible to use Data Science in order to understand current results and to improve current methods based on analysis and predictions.

Ap·pen·dix

This material is a Capstone Project for the IBM Data Science Course and the main objective doing this was to learn.

All code and files used in this project are available on github.com/uriasmg.

The author declare that he do not have any financial interest or competing interest in this analysis.

References:

[1] E. Seedhouse, SpaceX: making commercial spaceflight a reality. Springer Science & Business Media, 2013.

[2] H. W. Jones, “The Recent Large Reduction in Space Launch Cost,” 48th Int. Conf. Environ. Syst., 2018.

[3] A. Dinardi, B. Bjelde, and J. Insprueker, “New opportunities for small satellite programs provided by the Falcon family of launch vehicles,” in European Space Agency, (Special Publication) ESA SP, 2008.

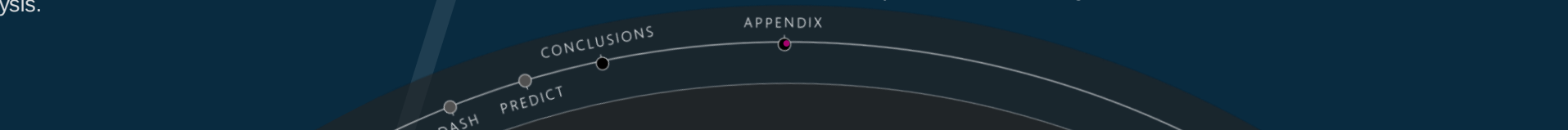
[4] H. W. Jones, “The impact of lower launch cost on space life support,” in 2018 AIAA SPACE and Astronautics Forum and Exposition, 2018.

[5] Space X, “SpaceX - Press Release,” 2013. [Online]. Available: https://web.archive.org/web/20130623215759/http://www.spacex.com/press.php?page=20100616.

[6] ForeignPolicy.com, “Falcon 9 - The Rocketeer,” 2017. [Online]. Available: https://web.archive.org/web/20170317221323/http://foreignpolicy.com/2013/12/09/the-rocketeer/.

[7] Spacex-info.com, “Falcon 9 and Falcon 9 Heavy,” 2020. [Online]. Available: https://web.archive.org/web/20200721035853/https://spacex-info.com/falcon-9-falcon-heavy/.

[8] Space.com, “SpaceX rocket landing sucess,” 2015. [Online]. Available: https://web.archive.org/web/20181128061324/https://www.space.com/31420-spacex-rocket-landing-success.html.



contato@drmuller.com.br

Thank you

Müller Gonçalves Urias, M.D.
contato@drmuller.com.br