Winning Space Race with Data Science

Janet Zhang
October 2022

Outline

- Executive Summary
- Introduction
- Methodology
- Results
- Conclusion

Introduction

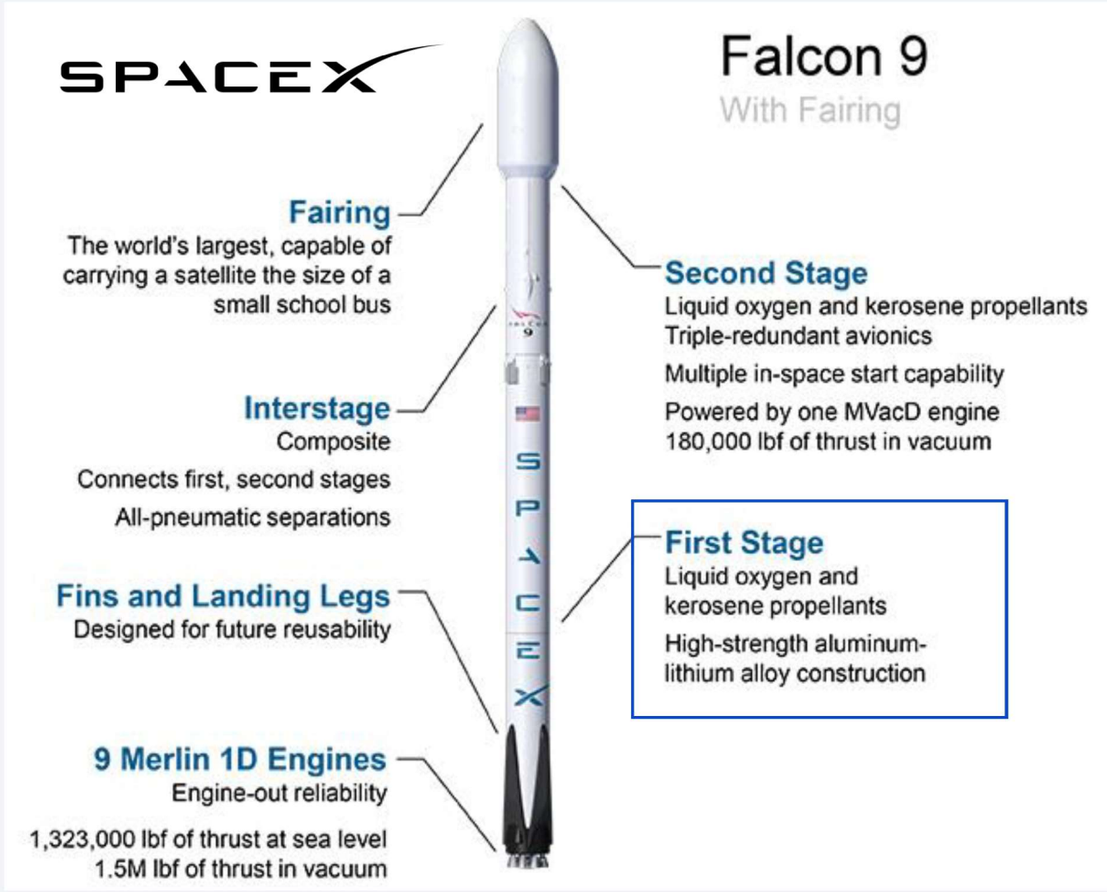
- ## Project background

Falcon 9 rocket is a reusable, two-stage rocket designed and manufactured by SpaceX for the reliable and safe transport of people and payloads into Earth orbit and beyond.

The first stage does most of the work and is much larger and more expensive than second stage. Sometimes the first stage does not land. Sometimes it will crash. Other times, Space X will sacrifice the first stage due to the mission parameters like payload, orbit, and customer.

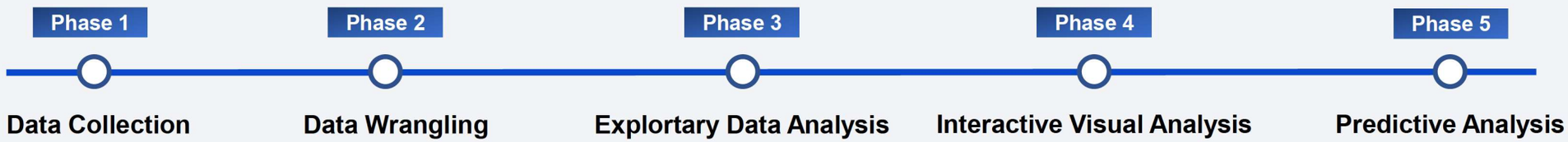
- ## Research Questions

1. What factors will influence the launch/landing outcomes of the first stage?
2. What is the tendency of the first stage's success rate over time?
3. Which Machine Learning Model performs best to predict if the first stage will land successfully?



Executive Summary

- Summary of methodologies



- Summary of all results

By gathering information about SpaceX, the study found that launch success rate may depend on many factors such as payload mass, orbit type, and so on. It may also depend on the location and proximities of a launch site, i.e., the initial position of rocket trajectories.

Section 1

Methodology

Methodology

Executive Summary

- Data collection methodology:
  - Data collection with an API and Web scraping
- Perform data wrangling
  - Data wrangling using API/Deal with Nulls/Convert categorical data to numerical data
- Perform exploratory data analysis (EDA) using visualization and SQL
- Perform interactive visual analytics using Folium and Plotly Dash
- Perform predictive analysis using classification models
- build a machine learning pipeline to predict and find the model with the best accuracy

https://github.com/JanetZhang128/SpaceX_Project

Data Collection

Using the SpaceX REST API

https://api.spacexdata.com/v4/

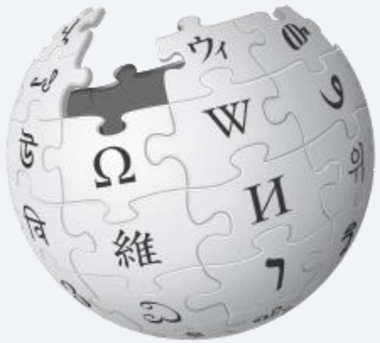
api.spacexdata.com/v4/capsules

api.spacexdata.com/v4/cores

api.spacexdata.com/v4/launches/past

Web scraping related Wiki pages

Falcon 9 launch records
from Wikipedia pages



Data Collection – SpaceX API

Request and parse the SpaceX launch data using the GET request

1. Use the URL to target a specific endpoint of the API to get past launch data
2. Perform a get request using the requests library to obtain the launch data
3. Call the .json() method to review the result
   Use the json_normalize function to convert this JSON to a dataframe

```
In [6]: spacex_url="https://api.spacexdata.com/v4/launches/past"

In [7]: response = requests.get(spacex_url)
```

To make the requested JSON results more consistent, we will use the following static response object for this project:

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

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

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

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

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

GitHub URL https://github.com/JanetZhang128/SpaceX_Project/blob/main/Spacex-data-collection-api.ipynb

Data Collection - Scraping

Web scrap Falcon 9 launch records with BeautifulSoup

1. Request the Falcon9 Launch Wiki page from its URL
2. Extract all column/variable names from the HTML table header
3. Create a data frame by parsing the launch HTML tables

GitHub URL
https://github.com/JanetZhang128/SpaceX_Project/blob/main/Spacex-data-collection-webscraping.ipynb

```
In [6]: # use requests.get() method with the provided static_url
        response = requests.get(static_url)
        # assign the response to a object
        content = response.text
```

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

```
In [7]: # Use BeautifulSoup() to create a BeautifulSoup object from a response text content
        soup = BeautifulSoup(content, 'html.parser')
```

```
In [11]: # 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.

```
In [12]: # Let's print the third table and check its content
         first_launch_table = html_tables[2]
         print(first launch table)
```

```
In [22]: column_names = []

         # Apply find_all() function with `th` element on first_launch_table
         th_elements = first_launch_table.find_all('th')

         # 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 th_elements:
             name = extract_column_from_header(row)
             if name !=None and len(name)>0:
                 column_names.append(name)
```

```
In [24]: 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']=[]
```

```
In [26]: df=pd.DataFrame(launch_dict)
```

1
- Perform an HTTP GET method to request the Falcon9 Launch HTML page, as an HTTP response
- Create a BeautifulSoup object from the HTML response

2
- Use the find_all function in the BeautifulSoup object, with element type ‘table’
- Starting from the third table is our target table contains the actual launch records.
- iterate through the <th> elements and apply extract_column_from_header() to extract column name one by one

3
- Create an empty dictionary with keys from the extracted column names
- fill up the launch_dict with launch records extracted from table rows (see GitHub URL)
- Convert the dictionary into a Pandas dataframe

Data Wrangling

Wrangling Data using an API

use the API again targeting another endpoint to gather specific data for each ID number

Sampling Data

Filter/Sample the data to remove Falcon 1 launches and keep Falcon 9 launches

Dealing with NULLs

Replace the NULL values inside the PayloadMass with its mean value

Convert categorical values

Convert landing_outcomes to Classes y (1 meansthe booster successfully landed, 0 means it was unsuccessful)

GitHub URL https://github.com/JanetZhang128/SpaceX_Project/blob/main/Spacex-Data%20wrangling.ipynb

Data Wrangling

Wrangling Data using an API

use the API again targeting another endpoint to gather specific data for each ID number

1. Take a subset of the dataframe only keeping the features i.e. rocket, payloads, launchpad, and cores
2. Apply getBoosterVersion()/getLaunchSite()/ getPayLoadData()/getCoreData() function
3. Combine the columns into a dictionary Create a Pandas data frame from the dictionary launch_dict.

```
In [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']]
```

```
In [14]: #Global variables
BoosterVersion = []
PayloadMass = []
Orbit = []
LaunchSite = []
Outcome = []
Flights = []
GridFins = []
Reused = []
Legs = []
LandingPad = []
Block = []
ReusedCount = []
Serial = []
Longitude = []
Latitude = []
```

1

- data[‘rocket’] learn variable ‘BoosterVersion’
- data[‘payload’] learn variables ‘PayloadMass’ ,‘Orbit’
- data[‘launchpad’] learn variables ‘LaunchSite’ ‘Longitude’, ‘Latitude’
- data[‘ cores’] learn variables ‘Outcome’, ‘Flights’, ‘GridFins’, ‘Reused’, ‘ReusedCount’ ‘Legs’, ‘LandingPad’, ‘Block’, ‘Serial’

```
In [16]: # Call getBoosterVersion
getBoosterVersion(data)
```

```
In [18]: # Call getLaunchSite
getLaunchSite(data)
```

```
In [19]: # Call getPayloadData
getPayloadData(data)
```

```
In [20]: # Call getCoreData
getCoreData(data)
```

2

```
In [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}
```

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

3

GitHub URL https://github.com/JanetZhang128/SpaceX_Project/blob/main/Spacex-data-collection-api.ipynb

Data Wrangling

Sampling Data

1. Filter the data dataframe using the BoosterVersion column to only keep the Falcon 9 launches

Dealing with NULLs

2. Check the missing values in the dataset
3. Calculate below the mean for the PayloadMass using the .mean()
4. Use the mean and the .fillna() function to replace np.nan values in the data

1

```
In [25]: # Hint data['BoosterVersion']!='Falcon 1'
         data_falcon9 = df[df['BoosterVersion']=='Falcon 9']
```

2

```
In [27]: data_falcon9.isnull().sum()
Out[27]: FlightNumber       0
         Date               0
         BoosterVersion     0
         PayloadMass        5
         Orbit              0
         LaunchSite         0
         Outcome            0
         Flights            0
         GridFins           0
         Reused             0
         Legs               0
         LandingPad        26
         Block              0
         ReusedCount        0
         Serial             0
         Longitude          0
         Latitude           0
         dtype: int64
```

3 4

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

         # Replace the np.nan values with its mean value
         data_falcon9 = data_falcon9.fillna(PayloadMass_mean)

         #check the number of missing values of the PayLoadMass again
         data_falcon9['PayloadMass'].isnull().sum()

         data_falcon9
```

GitHub URL https://github.com/JanetZhang128/SpaceX_Project/blob/main/Spacex-Data%20wrangling.ipynb

Data Wrangling

Convert categorical values

1. Calculate the number and occurrence of mission outcome

2. Create a landing outcome label from Outcome column, 1 for landing successfully, 0 for landing unsuccessfully

```
In [9]: # landing_outcomes = values on Outcome column
        landing_outcomes = df['Outcome'].value_counts()
        landing_outcomes

In [10]: 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
```

1

- Use the method .value_counts() on the column Outcome to determine the number of landing_outcomes

```
In [11]: bad_outcomes=set(landing_outcomes.keys()[[1,3,5,6,7]])
         bad_outcomes

In [12]: # landing_class = 0 if bad_outcome
         # landing_class = 1 otherwise
         landing_class = []

         for outcome in df['Outcome']:
             if outcome in bad_outcomes:
                 landing_class.append("0")
             else:
                 landing_class.append("1")

         landing_class
```

2

- Create a set bad_outcomes where the mission outcome was not successfully landed
- Create a list where the element is zero if the corresponding row in Outcome is in the set bad_outcome; otherwise, it's one
- assign it to the variable landing_class

GitHub URL https://github.com/JanetZhang128/SpaceX_Project/blob/main/Spacex-Data%20wrangling.ipynb

Data Wrangling

Overview of Dataset


```
df.head(5)
```

	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins	Reused	Legs	LandingPad	Block	ReusedCount	Serial	Longitude	Latitude	Class
0	1	2010-06-04	Falcon 9	6104.959412	LEO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B0003	-80.577366	28.561857	0
1	2	2012-05-22	Falcon 9	525.000000	LEO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B0005	-80.577366	28.561857	0
2	3	2013-03-01	Falcon 9	677.000000	ISS	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B0007	-80.577366	28.561857	0
3	4	2013-09-29	Falcon 9	500.000000	PO	VAFB SLC 4E	False Ocean	1	False	False	False	NaN	1.0	0	B1003	-120.610829	34.632093	0
4	5	2013-12-03	Falcon 9	3170.000000	GTO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B1004	-80.577366	28.561857	0

1 Launch Sites

```
In [7]: # Apply value_counts() on column LaunchSite
        df['LaunchSite'].value_counts()
Out[7]: CCAFS SLC 40    55
        KSC LC 39A      22
        VAFB SLC 4E     13
        Name: LaunchSite, dtype: int64
```

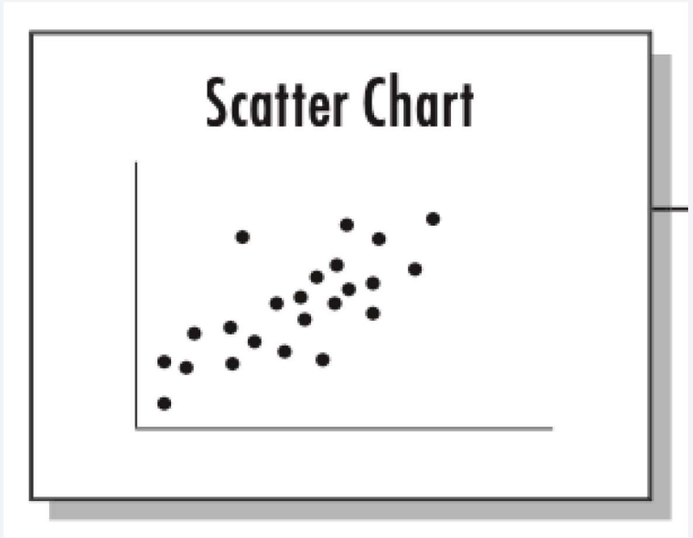
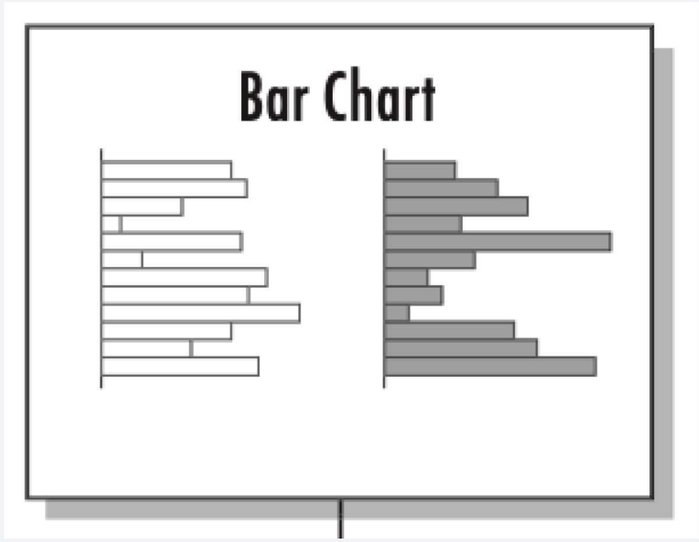
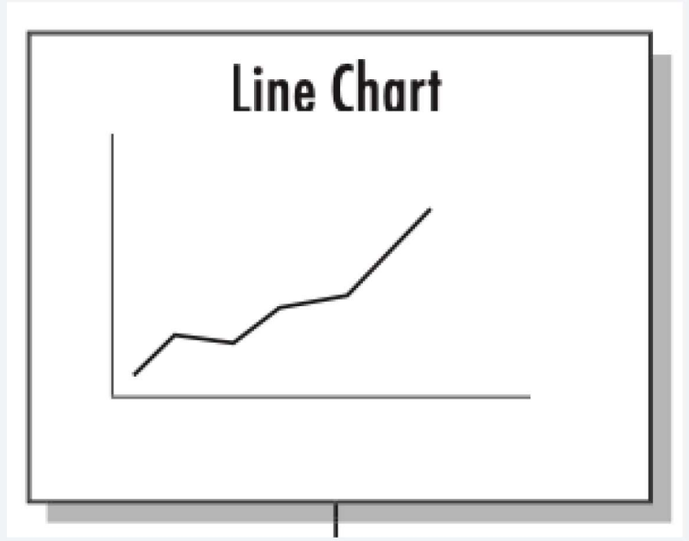
2 Orbits

```
In [8]: # Apply value_counts on Orbit column
        df['Orbit'].value_counts()
Out[8]: GTO      27
        ISS      21
        VLEO     14
        PO        9
        LEO       7
        SSO       5
        MEO       3
        ES-L1     1
        HEO       1
        SO        1
        GEO       1
        Name: Orbit, dtype: int64
```

3 Landing Outcomes

```
In [9]: # landing_outcomes = values on Outcome column
        landing_outcomes = df['Outcome'].value_counts()
        landing_outcomes
Out[9]: 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
```

EDA with Data Visualization

Relationship	Comparison Between Many Items	Comparison Over Time
		
• Flight Number vs. Launch Site • Payload vs. Launch Site • Flight Number vs. Orbit Type • Payload vs. Orbit Type	• Success Rate vs. Orbit Type	• Launch Success Yearly Trend

GitHub URL https://github.com/JanetZhang128/SpaceX_Project/blob/main/SpaceX_eda-dataviz.ipynb

EDA with SQL

1 SQL Queries about Launch Sites

- Display the names of the unique launch sites in the space mission
- Display 5 records where launch sites begin with the string 'CCA'

2 SQL Queries about Payload Mass

- Display the total payload mass carried by boosters launched by NASA (CRS)
- Display average payload mass carried by booster version F9 v1.1

3 SQL Queries about Booster Versions

- List the names of the boosters which have success in drone ship and have payload mass greater than 4000 but less than 6000
- List the names of booster_versions which have carried the maximum payload mass

4 SQL Queries about Landing Outcomes

- List the total number of successful and failure mission outcomes
- List the date when the first successful landing outcome in ground pad was achieved
- List the records which will display the month names, failure landing_outcomes in drone ship/booster versions/launch_site for the months in year 2015
- Rank the count of successful landing_outcomes between the date 04-06-2010 and 20-03-2017 in descending order

GitHub URL https://github.com/JanetZhang128/SpaceX_Project/blob/main/SpaceX_eda-sql-coursera_sqllite.ipynb

Build an Interactive Map with Folium

1. Mark all launch sites on a map

- create a folium Map object
- for each launch site, add a Circle object and Marker object based on its coordinate (Lat, Long) values. In addition, add Launch site name as a popup label
- folium.Circle(coordinate, radius=1000, color='#000000', fill=True).add_child(folium.Popup(...))
- folium.map.Marker(coordinate, icon=DivIcon(icon_size=(20,20),icon_anchor=(0,0), html=' %s ' % 'label',))

2. Mark the success/failed launches for each site on the map

- create a MarkerCluster object
- apply a function to check the value of `class` column. If class=1, marker_color value will be green; If class=0, marker_color value will be red
- for each launch result in spacex_df data frame, add a folium.Marker to marker_cluster

3. Calculate the distances between a launch site to its proximities

- add a MousePosition on the map to get coordinate for a mouse over a point on the map
- mark down a point on the closest proximities using MousePosition and calculate the distance between the closest proximities and the launch site
- after obtained its coordinate, create a folium.Marker to show the distance

GitHub URL https://github.com/JanetZhang128/SpaceX_Project/blob/main/SpaceX_launch_site_location.ipynb

Build a Dashboard with Plotly Dash

1. Add a launch site Drop-down input component

```
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
             ),
```

2. Add a callback function to render success-pie-chart based on selected site dropdown

```
@app.callback(
    Output(component_id='success-pie-chart', component_property='figure'),
    Input(component_id='site-dropdown', component_property='value'))

def build_graph(site_dropdown):
    if site_dropdown == 'ALL':
        piechart = px.pie(data_frame = spacex_df, names='Launch Site', values='class' ,title='Total Success Launches
        return piechart
    else:
        #specific_df = spacex_df['Launch Site']
        specific_df=spacex_df.loc[spacex_df['Launch Site'] == site_dropdown]
        piechart = px.pie(data_frame = specific_df, names='class',title='Total Success Launch for a Specific Site')
        return piechart
```

GitHub URL
https://github.com/JanetZhang128/SpaceX_Project/blob/main/SpaceX_Interactive%20Visual%20Analytics%20and%20Dashboard.ipynb

3. Add a range slider to select payload

```
dcc.RangeSlider(id='payload-slider',
                min=0,max=10000,step=1000,
                value=[min_payload,max_payload],
                marks={0: '0', 2500:'2500',5000:'5000',
                7500:'7500', 10000: '10000'}),
```

4. Add a callback function to render the success-payload-scatter-chart

```
@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 update_graph(site_dropdown, payload_slider):
    if site_dropdown == 'ALL':
        filtered_data = spacex_df[(spacex_df['Payload Mass (kg)']>=payload_slider[0])
        &(spacex_df['Payload Mass (kg)']<=payload_slider[1])]
        scatterplot = px.scatter(data_frame=filtered_data, x="Payload Mass (kg)", y="class",
        color="Booster Version Category")
        return scatterplot
    else:
        specific_df=spacex_df.loc[spacex_df['Launch Site'] == site_dropdown]
        filtered_data = specific_df[(specific_df['Payload Mass (kg)']>=payload_slider[0])
        &(spacex_df['Payload Mass (kg)']<=payload_slider[1])]
        scatterplot = px.scatter(data_frame=filtered_data, x="Payload Mass (kg)", y="class",
        color="Booster Version Category")
        return scatterplot
```

Predictive Analysis (Classification)

1. Preprocessing and split data into trainning and testing data

```
In [6]: # students get this
        transform = preprocessing.StandardScaler()

In [7]: X = transform.fit(X).transform(X)

In [8]: X_train, X_test, Y_train, Y_test = train_test_split (X, Y, test_size = 0.2, random_state = 2)
```

2. Train the models and perform Grid Search for each model to find the best parameters

```
In [96]: parameters ={'C':[0.01,0.1,1],
                      'penalty':['l2'],
                      'solver':['lbfgs']}

In [97]: parameters ={"C":[0.01,0.1,1],'penalty':['l2'], 'solver':['lbfgs']}# l1 lasso l2 ridge
         lr=LogisticRegression()

         logreg_cv = GridSearchCV(estimator = lr,
                                  param_grid = parameters,
                                  scoring = 'accuracy',
                                  cv = 10,
                                  verbose=0)
         logreg_cv.fit(X_train, Y_train)
```

Logistic Regression

```
In [101]: parameters = {'kernel':('linear', 'rbf','poly','rbf', 'sigmoid'),
                       'C': np.logspace(-3, 3, 5),
                       'gamma':np.logspace(-3, 3, 5)}
          svm = SVC()

In [102]: svm_cv = GridSearchCV(estimator = svm,
                                 param_grid = parameters,
                                 scoring = 'accuracy',
                                 cv = 10,
                                 verbose=0)
          svm_cv.fit(X_train, Y_train)
```

SVM

```
In [106]: parameters = {'criterion': ['gini', 'entropy'],
                       'splitter': ['best', 'random'],
                       'max_depth': [2*n for n in range(1,10)],
                       'max_features': ['auto', 'sqrt'],
                       'min_samples_leaf': [1, 2, 4],
                       'min_samples_split': [2, 5, 10]}

          tree = DecisionTreeClassifier()

In [107]: tree_cv = GridSearchCV(estimator = tree,
                                 param_grid = parameters,
                                 scoring = 'accuracy',
                                 cv = 10)
          tree_cv.fit(X_train, Y_train)
```

Decision Tree

```
In [111]: parameters = {'n_neighbors': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
                       'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute'],
                       'p': [1,2]}

          KNN = KNeighborsClassifier()

In [112]: KNN_cv = GridSearchCV(estimator = KNN,
                                param_grid = parameters,
                                scoring = 'accuracy',
                                cv = 10,
                                verbose=0)
          KNN_cv.fit(X_train, Y_train)
```

K Nearest Neighbours

3. Calculate the accuracy on the test data using the method score() and plot the confusion matrix

4. Determine the model with the best accuracy

GitHub URL
https://github.com/JanetZhang128/SpaceX_Project/blob/main/SpaceX_Machine%20Learning%20Prediction.ipynb

Section 2

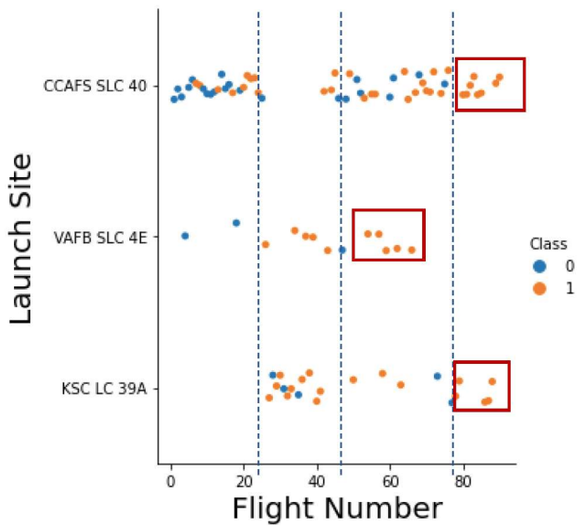
Insights drawn from EDA

Flight Number vs. Launch Site

Code

```
In [6]: # Plot a scatter point chart with x axis to be Flight Number and y axis to be the launch site, and hue to be the cla
sns.catplot(y="LaunchSite", x="FlightNumber", hue="Class", data=df)
plt.xlabel("Flight Number",fontsize=20)
plt.ylabel("Launch Site",fontsize=20)
plt.show()
```

Outcome



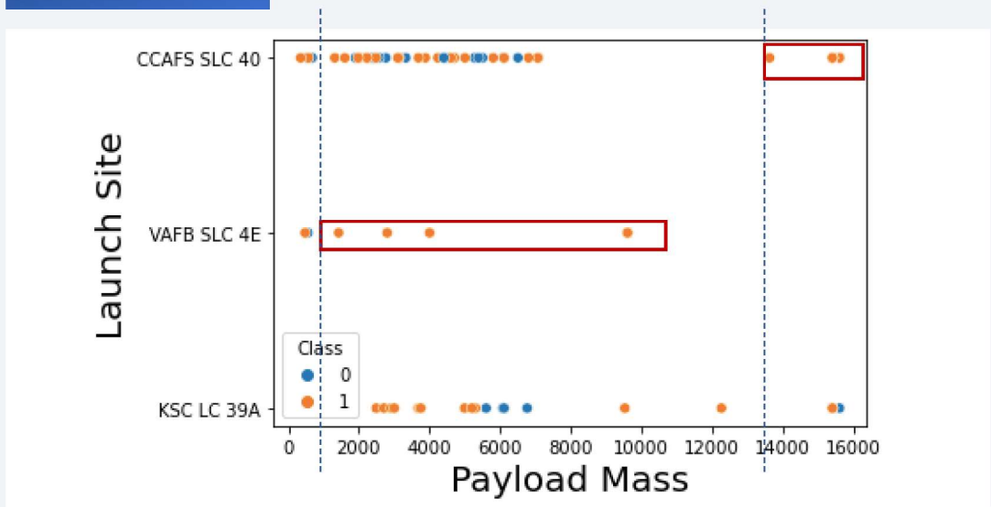
- In general, as the flight number increases, the first stage is more likely to land successfully.
- When the flight number reached to 25, more landing outcome were successful than before.
- Different launch sites have different success rates. CCAFS LC-40, has a success rate of 60 %, while KSC LC-39A and VAFB SLC 4E has a success rate of 77%.
- At Launch site VAFB SLC4E, when the flight number was above 50, the success rate reached 100%. While at Launch site CCAFS SLC 40 and KSC LC 39A, when the flight number was around 78, the success rate reached 100%.

Payload vs. Launch Site

Code

```
In [7]: # Plot a scatter point chart with x axis to be Pay Load Mass (kg) and y axis to be the launch site, and hue to be th
        sns.scatterplot(data=df, x="PayloadMass", y="LaunchSite",hue="Class")
        plt.xlabel("Payload Mass",fontsize=20)
        plt.ylabel("Launch Site",fontsize=20)
        plt.show()
```

Outcome



- At Launch site VAFB SLC4E and CCAFS SLC 40, it seems that the more massive the payload was, the more likely they achieved a successful landing outcome. While there is no such trend showed at KSC LC 39A.
- At Launch site VAFB SLC4E, the success rate reached 100%, when the payload mass was above 2000. While at Launch site CCAFS SLC 40, the success rate reached 100%, when the payload mass was above 13000.

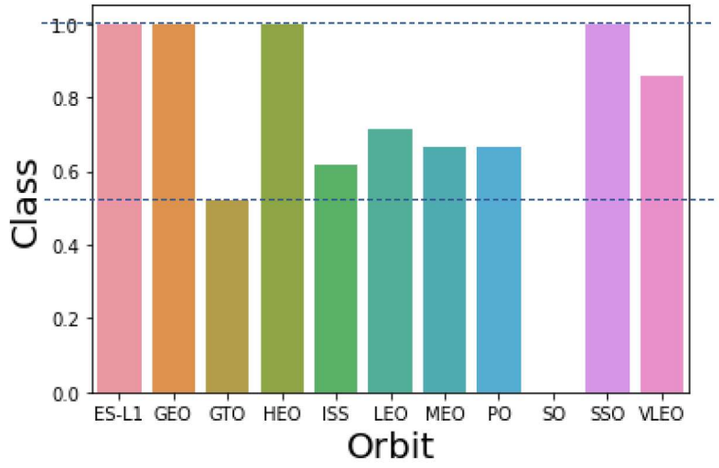
Success Rate vs. Orbit Type

Code

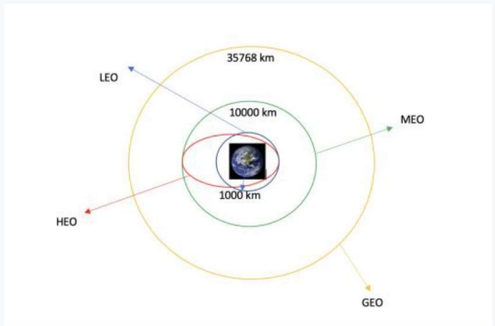
```
In [38]: # HINT use groupby method on Orbit column and get the mean of Class column
         orbit_successrate = df.groupby('Orbit').mean().reset_index()

         sns.barplot(data=df_orbit_successrate, x="Orbit", y="Class")
         plt.xlabel("Orbit",fontsize=20)
         plt.ylabel("Class",fontsize=20)
         plt.show()
```

Outcome



- Orbits ES-L1, GEO, HEO and SSO have the highest success rate, reaching 100%, while GTO has the lowest success rate.

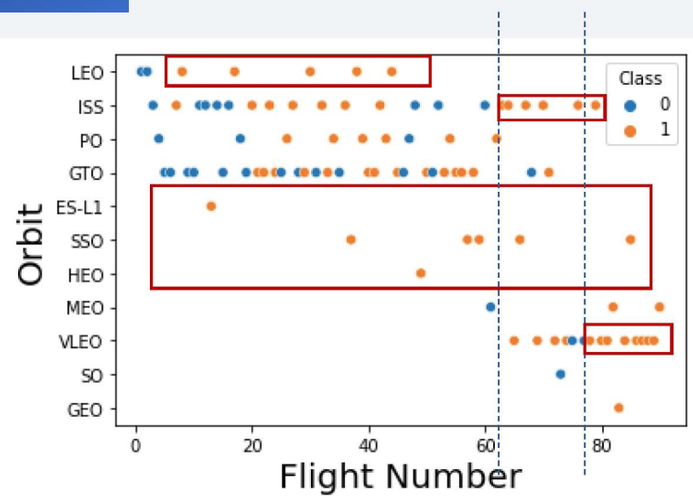


Flight Number vs. Orbit Type

Code

```
In [31]: # Plot a scatter point chart with x axis to be FlightNumber and y axis to be the Orbit, and hue to be the class valu
sns.scatterplot(data=df, x="FlightNumber", y="Orbit", hue="Class")
plt.xlabel("Flight Number",fontsize=20)
plt.ylabel("Orbit",fontsize=20)
plt.show()
```

Outcome



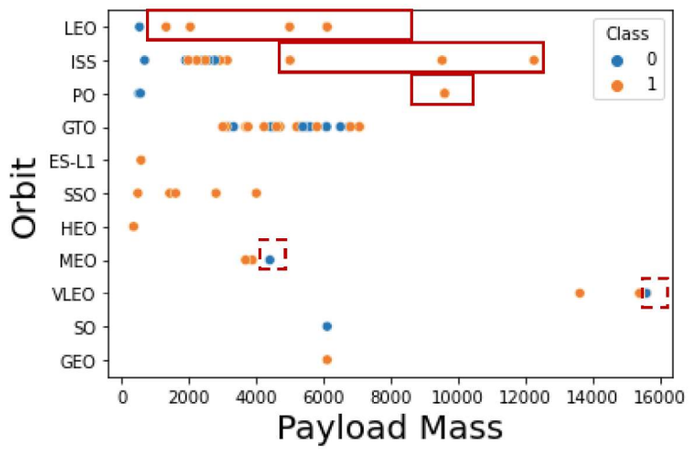
- In LEO Orbit, as the flight number increases, it appears more likely to achieve a good landing outcome.
- In SSO, ES-L1 and HEO Orbits, all of the landings were successful. But it is worth noting that the flight number was relatively limited, especially in ES-L1 and HEO Orbits, only 1 landing.
- In ISS orbit, when the flight number reached around 65, it was more likely to land successfully. While in VLEO orbit, it showed similar trend when the flight number reached around 78.

Payload vs. Orbit Type

Code

```
In [32]: # Plot a scatter point chart with x axis to be Payload and y axis to be the Orbit, and hue to be the class value
sns.scatterplot(data=df, x="PayloadMass", y="Orbit", hue="Class")
plt.xlabel("Payload Mass",fontsize=20)
plt.ylabel("Orbit",fontsize=20)
plt.show()
```

Outcome



- In LEO, ISS and PO Orbits, it appears more likely to achieve a good landing outcome with heavy payload. While in MEO and VLEO Orbits, it was less likely to land successfully with heavy payload.

Launch Success Yearly Trend

Code

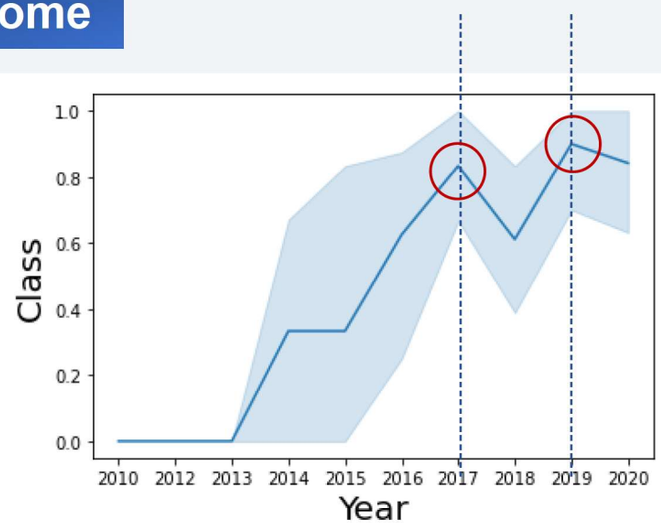
```
In [35]: # A function to Extract years from the date
         year=[]
         def Extract_year(date):
             for i in df["Date"]:
                 year.append(i.split("-")[0])
             return year

         # Add a year column in the dataframe
         df['Year']= Extract_year('Date')

         df.head()
```

```
In [36]: # Plot a line chart with x axis to be the extracted year and y axis to be the success rate
         sns.lineplot(data=df, x="Year", y="Class")
         plt.xlabel("Year",fontsize=20)
         plt.ylabel("Class",fontsize=20)
         plt.show()
```

Outcome



- In the initial stage, most of the landing were unsuccessful. Until 2013, the success rates were improved significantly, reaching 80% in 2017.
- But there appears a turning point in 2018, with success rate falling to 60%.
- Since 2018, the overall landing outcomes were good, especially reaching its peak at approximately 90% in 2019.

Launch Sites

Find the names of the unique launch sites

```
In [41]: %sql SELECT DISTINCT LAUNCH_SITE FROM SPACEXTBL
 * sqlite:///my_data1.db
Done.
```

Out[41]:

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

Find 5 records where launch sites begin with `CCA`

```
In [42]: %sql SELECT LAUNCH_SITE FROM SPACEXTBL WHERE LAUNCH_SITE LIKE 'CCA%' LIMIT 5
 * sqlite:///my_data1.db
Done.
```

Out[42]:

Launch_Site
CCAFS LC-40
CCAFS LC-40
CCAFS LC-40
CCAFS LC-40
CCAFS LC-40

- SpaceX has four launch sites, including CCAF LC-40, VAFB SLC-4E, KSC LC-39A and CCAFS SLC-40.

Boosters Carried Maximum Payload

List the names of the booster which have carried the maximum payload mass

```
In [66]: %sql SELECT DISTINCT(BOOSTER_VERSION) \
             FROM SPACEXTBL \
             WHERE PAYLOAD_MASS__KG_ = (SELECT MAX(PAYLOAD_MASS__KG_)FROM SPACEXTBL )
          * sqlite:///my_data1.db
         Done.
```

Out[66]:

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

- As the result showed, Falcon 9 block 5 first-stage boosters can carry the maximum payload mass. Block 5 is the final iteration of the Falcon 9.
- It is also worth mentioning that B1049 and B1060 are in this list. B1049 is the oldest and earliest launched of the active Falcon 9 boosters, whereas B1060 is the booster with most satellites launched by it.

Falcon 9 Block 5

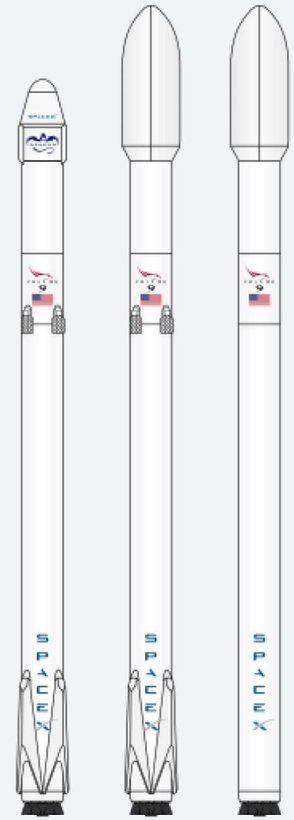
Payload Mass Carried by Different Boosters

Calculate the average payload mass carried by booster version F9 v1.1

```
In [77]: %sql SELECT AVG(PAYLOAD_MASS__KG_)AS AVERAGE_PAYLOAD_MASS FROM SPACEXTBL WHERE BOOSTER_VERSION ='F9 v1.1'
 * sqlite:///my_data1.db
Done.
```

```
Out[77]: AVERAGE_PAYLOAD_MASS
         2928.4
```

- Version 1.0 of the Falcon 9 was the first version. It looked very different from what it does today and it was much smaller and had much less power. As showed, the first version of Falcon 9 boosters can carry 2928.4 kg payload in average.
- This data could be seen as a baseline to evaluate the boosters 'Payload Mass' feature. If a new booster version carries less weight of payload than F9 v1.1, then it should not be launched.



Falcon 9 v1.1

Payload Mass Carried by Different Boosters

Calculate the total payload mass carried by boosters from NASA (CRS)

```
In [43]: %sql SELECT SUM(PAYLOAD_MASS__KG_)AS TOTAL_PAYLOAD_MASS FROM SPACEXTBL WHERE CUSTOMER ='NASA (CRS)'
          * sqlite:///my_data1.db
         Done.
```

```
Out[43]: TOTAL_PAYLOAD_MASS
         45596
```

- SpaceX launched its **commercial resupply services** (CRS) mission for the International Space Station (ISS) on Falcon 9. As the result suggested, the boosters from NASA(CRS) carry 45596 KG of payload in total.

Payload Mass Carried by Different Boosters

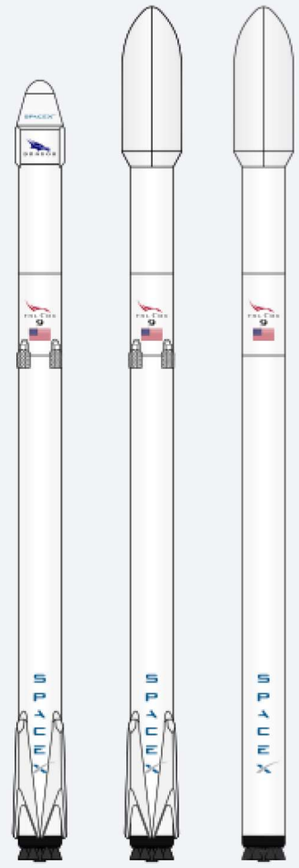
List the names of boosters which have successfully landed on drone ship and had payload mass greater than 4000 but less than 6000

```
In [90]: %sql SELECT BOOSTER_VERSION FROM SPACEXTBL \
             WHERE ([Landing _Outcome]='Success (drone ship)') AND(PAYLOAD_MASS__KG_ BETWEEN 4000 AND 6000)
          * sqlite:///my_data1.db
         Done.
```

Out[90]:

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

- As showed, most of the boosters which fulfil the filter conditions are the Falcon 9 Full Thrust.
- Falcon 9 Full Thrust was the first version of the Falcon 9 to successfully land.



Falcon 9 v1.2 (FT)

Successful and Failure Mission Outcomes

Calculate the total number of successful and failure mission outcomes

```
In [63]: %sql SELECT MISSION_OUTCOME, COUNT(MISSION_OUTCOME) AS TOTAL_NUMBER \
             FROM SPACEXTBL \
             GROUP BY MISSION_OUTCOME

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

Out[63]:

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

- As showed, the majority of the missions have been completed successfully, the success rate of mission is nearly 99%.
- Only 1 failure over the entire time period.
- Only 1 mission outcome is success but with payload status unclear.

2015 Launch Records

List the failed landing_outcomes in drone ship/their booster versions/launch site names for in 2015

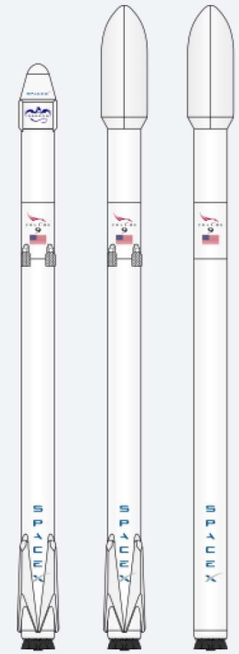
```
In [91]: %sql SELECT SUBSTR(Date, 4, 2) AS MONTH,BOOSTER_VERSION,LAUNCH_SITE\
             FROM SPACEXTBL\
             WHERE [Landing _Outcome] = 'Failure (drone ship)' AND (SUBSTR(Date,7,4)='2015')
          * sqlite:///my_data1.db
         Done.
```

Out[91]:

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

- The launch records which fulfil the filtering condition are as follows:

B1012	v1.1	10 January 2015	F9-014	Dragon C107 (CRS-5)	Success (40)	Failure	Destroyed
B1015	v1.1	14 April 2015	F9-017	Dragon C108 (CRS-6)	Success (40)	Failure	Destroyed



Falcon 9 v1.1

First Successful Ground Landing Date

Find the dates of the first successful landing outcome on ground pad

```
In [89]: SELECT MIN(DATE) AS FIRST_SUCCESSFUL_GROUND_LANDING FROM SPACEXTBL WHERE [Landing _Outcome] = 'Success (ground pad)'
 * sqlite:///my_data1.db
Done.
```

Out[89]:

FIRST_SUCCESSFUL_GROUND_LANDING
01-05-2017

- The launch record which fulfils the filtering condition is:

B1032	FT	1 May 2017	F9-033	—	USA-276 (NROL-76)[50]	Success (39A)	Success (LZ-1)	Expended[51]
		31 January 2018	F9-048 ♺	275 days	GovSat-1 / SES-16[52]	Success (40)	Controlled (ocean) [d]	

Falcon 9 v1.2 (FT)

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

Rank the count of landing outcomes between the date 2010-06-04 and 2017-03-20

```
In [104]: %sql SELECT [Landing _Outcome], COUNT([Landing _Outcome]) AS TOTAL_NUMBER \
              FROM SPACEXTBL \
              WHERE Date BETWEEN ' 04-06-2010' AND '20-03-2017' \
              GROUP BY [Landing _Outcome] \
              ORDER BY TOTAL_NUMBER DESC
           * sqlite:///my_data1.db
          Done.
```

Out[104]:

Landing _Outcome	TOTAL_NUMBER
Success	24
No attempt	14
Success (ground pad)	8
Success (drone ship)	8
Failure (drone ship)	5
Failure	3
Controlled (ocean)	3
Failure (parachute)	2
No attempt	1

- As showed, the total number of successful landing between 2010-06-04 and 2017-03-20 were 40, accounting for 58.8% of the entire landing outcomes. Among them, there were 8 ground-pad success and 8 drone-ship success.
- The total number of landing failure between 2010-06-04 and 2017-03-20 were 10, accounting for 14.7% of the entire landing outcomes. Among them, 5 drone-ship failure and 2 parachute-failure.
- During this time period, 15 landing outcomes were “no attempt” while 3 ocean-controlled landing.

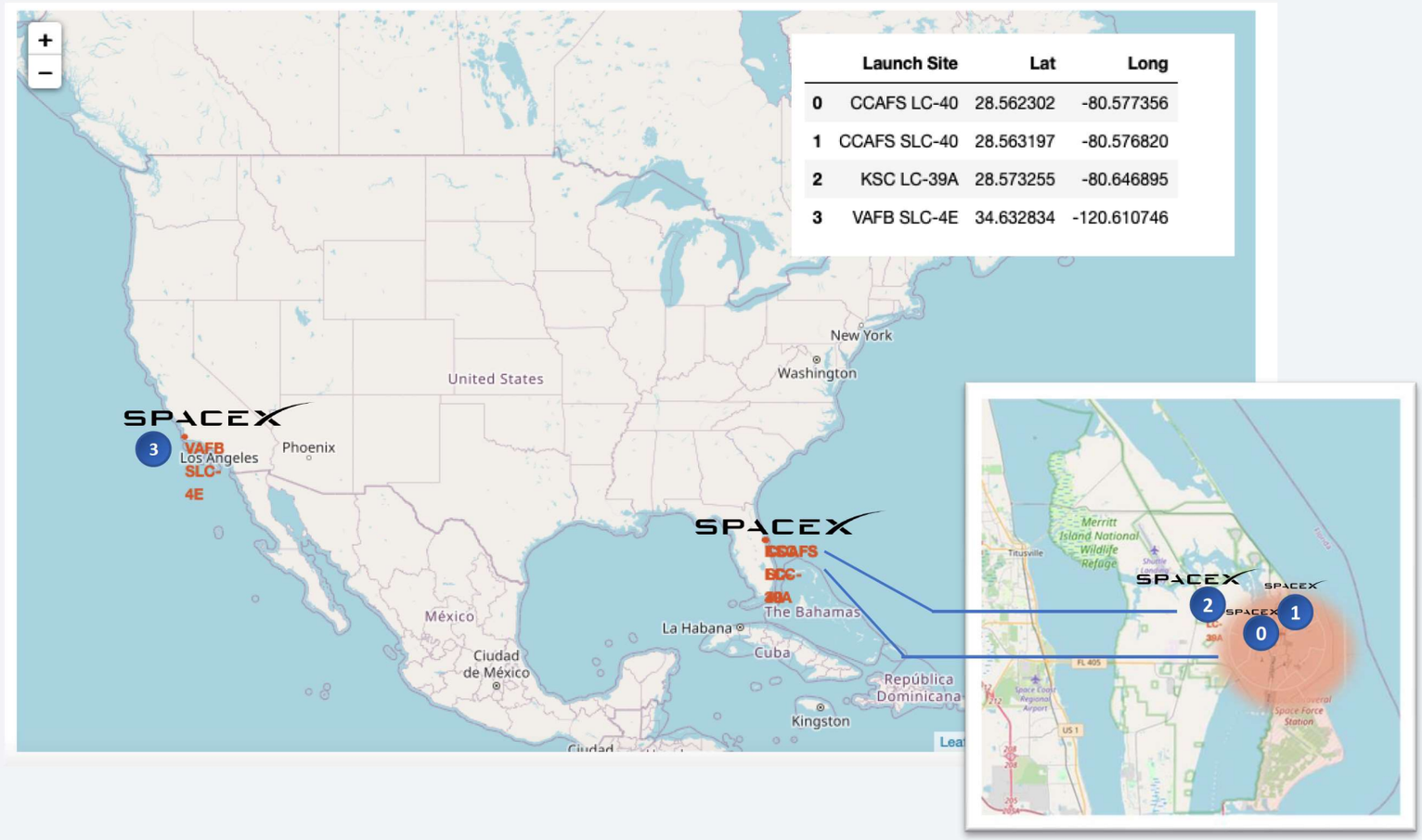
Section 3

Launch Sites Proximities Analysis

All Launch Sites

- All launch sites are located near coastlines.
- 3 launch sites are located in Florida, and 1 site in California.

	Launch Site	Lat	Long
0	CCAFS LC-40	28.562302	-80.577356
1	CCAFS SLC-40	28.563197	-80.576820
2	KSC LC-39A	28.573255	-80.646895
3	VAFB SLC-4E	34.632834	-120.610746

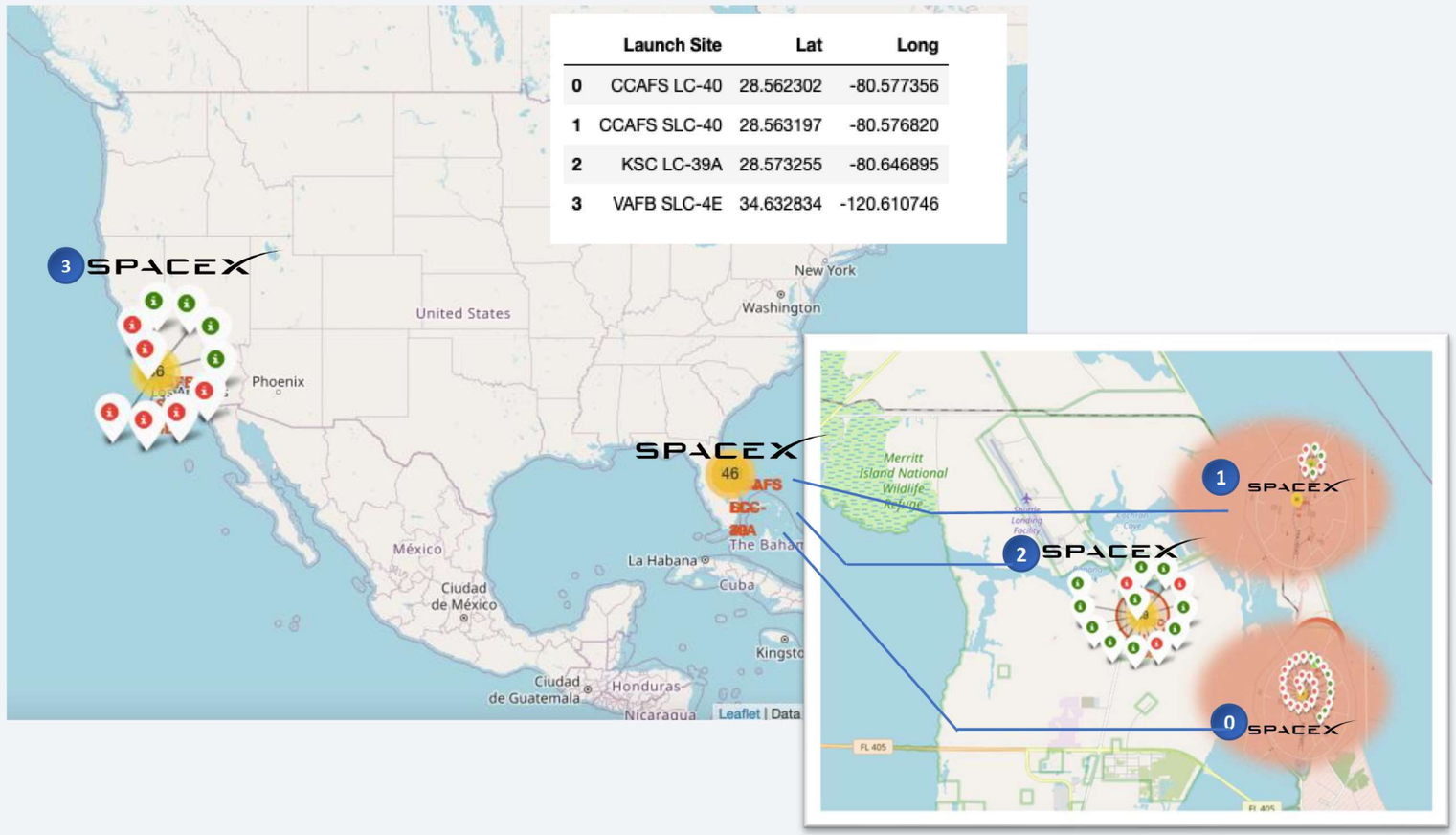


Success/Failure of Each Launch Site

- Launch site KSC LC-39A has more success launches than the other sites.
- Launch site CCAFS LC-40 has more failed launches compared to the other sites.

Note: the successful launch are marked as green, whereas the failed launch are marked as red on the map.

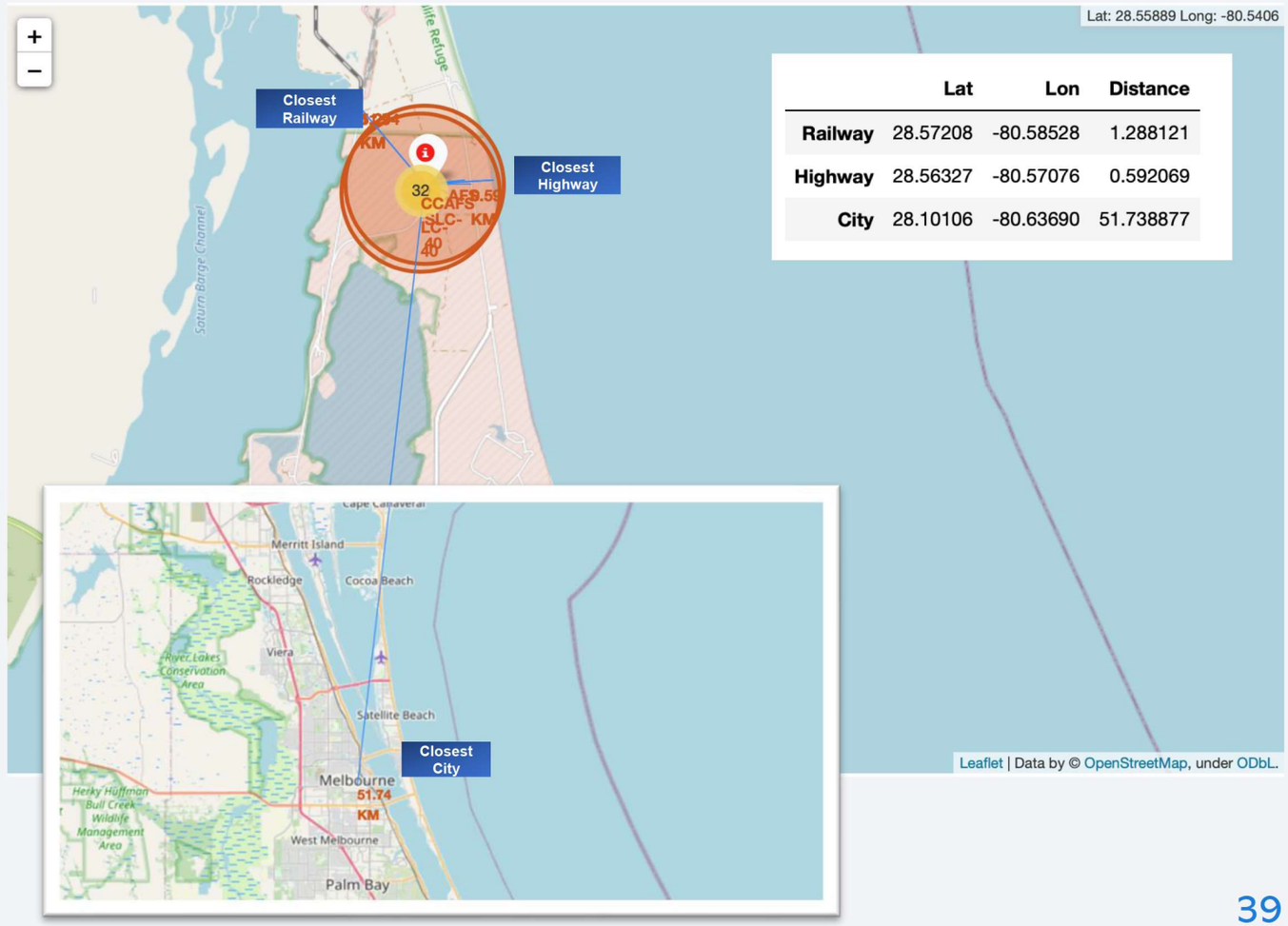
	Launch Site	Lat	Long
0	CCAFS LC-40	28.562302	-80.577356
1	CCAFS SLC-40	28.563197	-80.576820
2	KSC LC-39A	28.573255	-80.646895
3	VAFB SLC-4E	34.632834	-120.610746



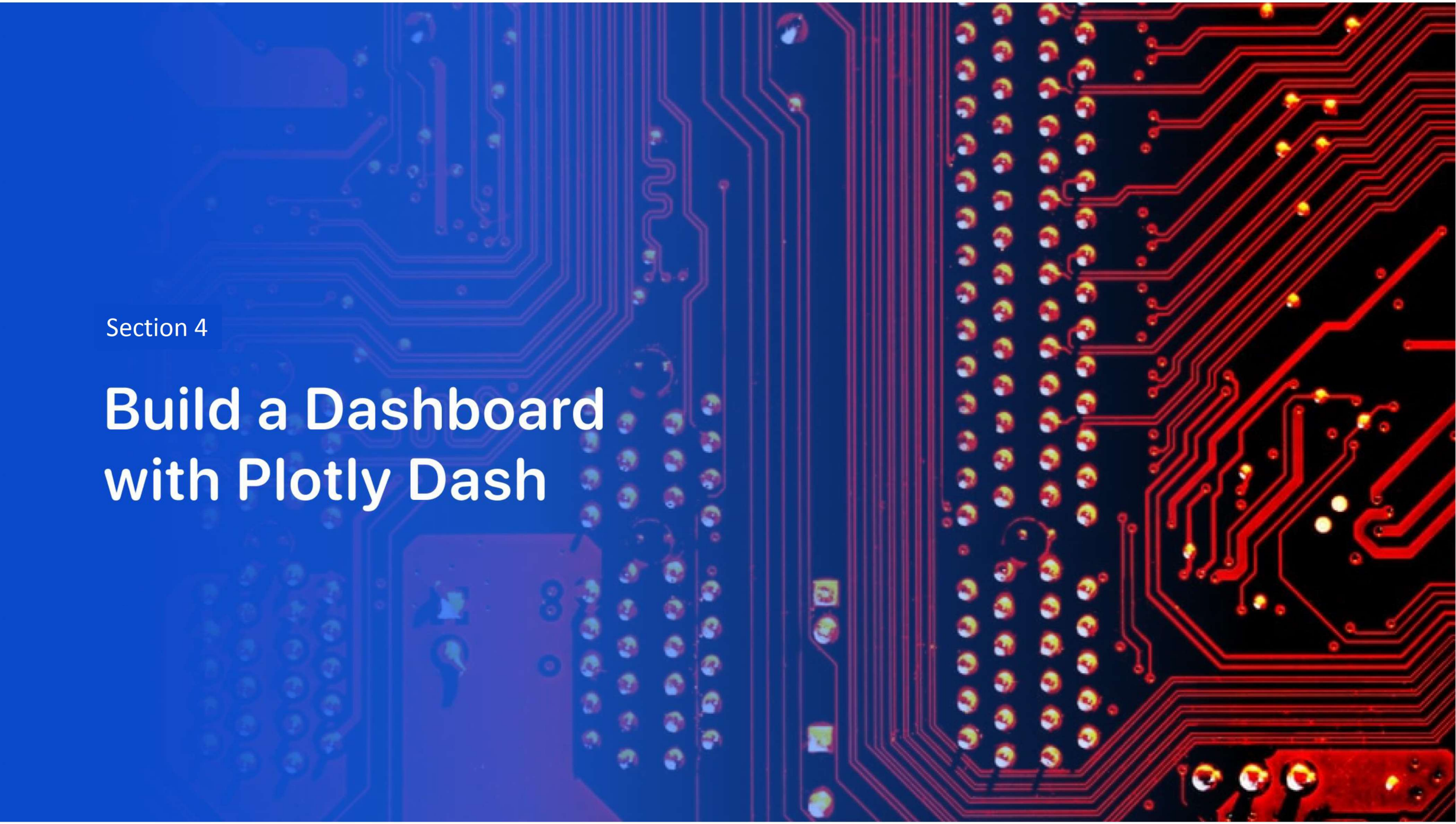
Distance Lines to the Proximities

Choose Launch site CCAFS SLC-40
and its proximities as an example
to explore the pattern

- The launch site in close proximity to railway is 1.29km.
- The launch site in close proximity to highway is 0.59km.
- The launch site in close proximity to coastline is 0.9km.
- The launch site keeps certain distance, i.e. 51.71km away from the city Melbourne.



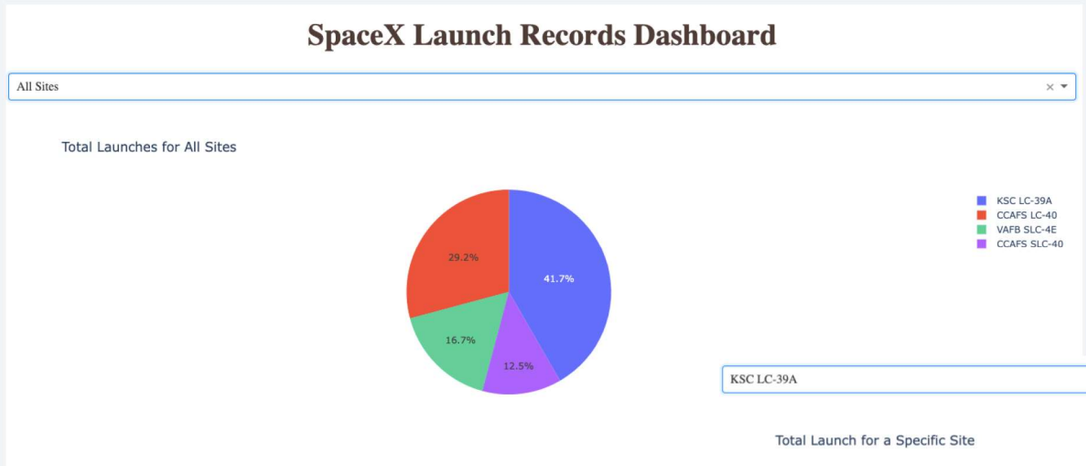
	Lat	Lon	Distance
Railway	28.57208	-80.58528	1.288121
Highway	28.56327	-80.57076	0.592069
City	28.10106	-80.63690	51.738877



Section 4

Build a Dashboard with Plotly Dash

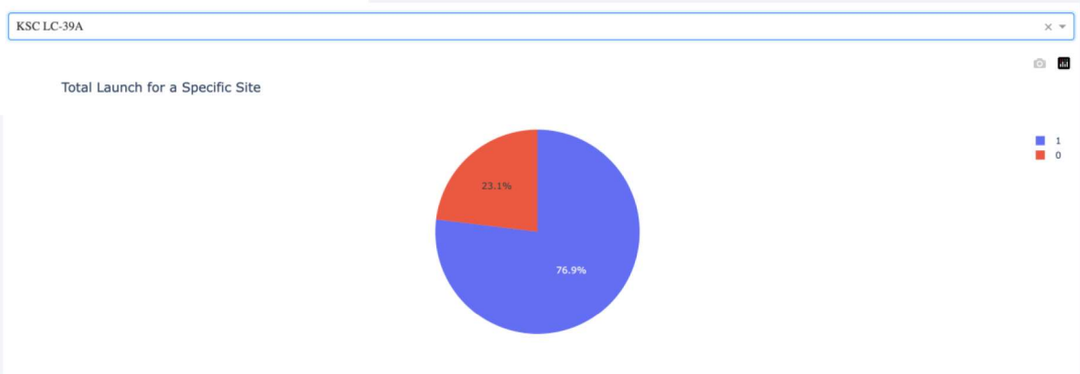
Launch Site with the Highest Launch Success Rate



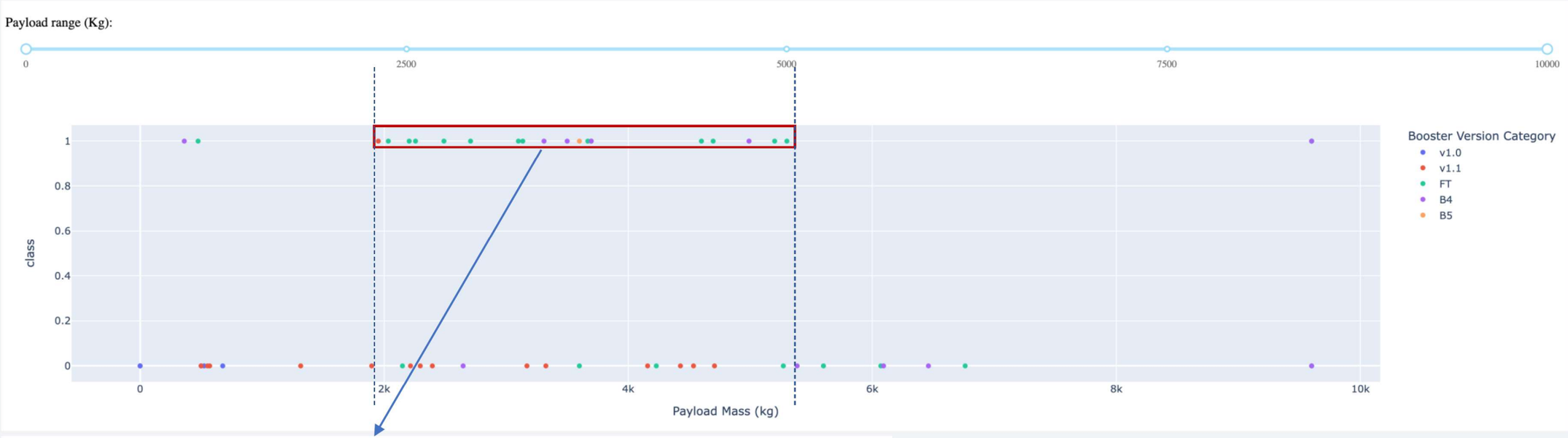
KSC LC-39A

has the largest successful launches, accounting for 41.7% of the total success launches for all sites.

Its launch success rate reaches 76.9%, ranking top 1 in all launch sites.



Payload Range with the Highest Success Rate

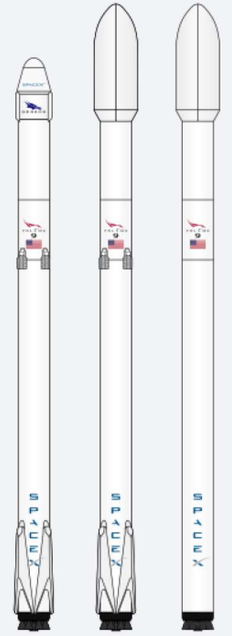


Payload range from 2000-5500Kg

has the highest success rate. As the figure below showed, there were 17 successful launches. Afterwards, with the payload increases, the success rate was falling.

In particular, when the payloads' weight was between 5500 and 10000Kg, it was more likely to launch unsuccessfully.

Booster Version with the Highest Success Rate



Falcon 9 v1.2 (FT)

Falcon 9 Booster Version FT

has the highest launch success rate, reaching 57.1%, when the payload range is between 0-10000Kg.
Whereas Booster Version B5 and v1.1 has a relatively low success rate.

Section 5

Predictive Analysis (Classification)

Classification Accuracy

Best Machine Learning Model

```
In [121]: data = [['logistic regression',logreg_accuracy_score,logreg_cv.best_score_],
                  ['support vector machine',svm_accuracy_score,svm_cv.best_score_],
                  ['decision tree',tree_accuracy_score,tree_cv.best_score_],
                  ['k nearest neighbors',KNN_accuracy_score,KNN_cv.best_score_]]

          df = pd.DataFrame(data, columns= ['Names','Accuracy Scores','Best Scores'] )
          df
```

Out[121]:

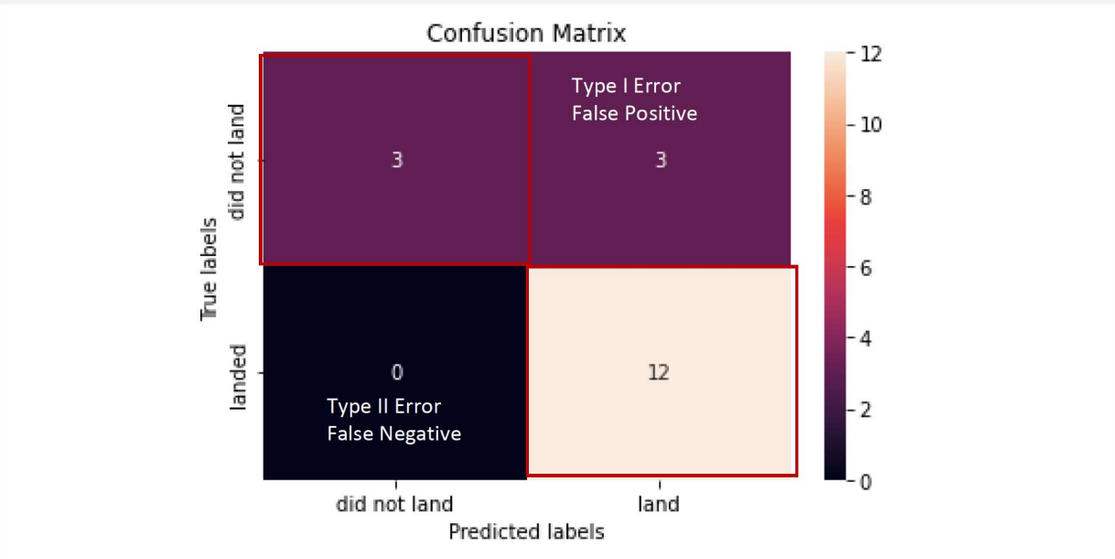
	Names	Accuracy Scores	Best Scores
0	logistic regression	0.833333	0.846429
1	support vector machine	0.833333	0.848214
2	decision tree	0.833333	0.876786
3	k nearest neighbors	0.833333	0.848214

Most of the Machine Learning Models (i.e Logistic Regression, Support Vector Machine, Decision Tree Classifier and K Nearest Neighbors) achieved a high accuracy score 0.83.

The Decision Tree Classifier Model performs the best accuracy using the training data.
Its Best Score is 0.876786.

Confusion Matrix

Best Machine Learning Model



Here is the Confusion Matrix of Decision Tree Model.

As presented before, Decision Tree Classifier Model performs best, with its best score of 0.876786.

As shown, 15 cases were predicted correctly, including 12 cases landing as predicted and 3 cases not landing as predicted.

3 cases fall into the Type I Error. It indicates that the model predicted False Positive for 3 times.

Conclusions

Exploratory Data Analysis Results

- In the initial stage, most of the landing were unsuccessful. Until 2013, the success rates were improved significantly, reaching **80%** in **2017**. But afterwards, in **2018** there appeared a turning point where the success rate falling to **60%**. After that, the overall landing outcomes were good, especially reaching its peak at approximately **90%** in **2019**.
- Generally, as the flight number increases, the first stage is more likely to land successfully. When the **flight number** reached to **25**, more landing outcome were successful than before.
- The launch success rate may depend on many factors such as payload mass, orbit type, and so on. **Orbits ES-L1, GEO, HEO** and **SSO** have the highest success rate, reaching 100%, while GTO has the lowest success rate. **Falcon 9 block 5 first-stage boosters** can carry the **maximum payload mass**. The first version of Falcon 9 boosters can carry 2928.4 kg payload in average. This data could be seen as a baseline to evaluate the boosters 'Payload Mass' feature. If a new booster version carries less weight of payload than F9 v1.1, then it should not be launched.

Conclusions

Interactive analytics Results

The launch success rate may depend on many factors such as launch site location, payload mass, and booster version, etc.,.

- SpaceX launch sites are in California and Florida, USA. Among them, KSC LC-39A has the highest launch success rate reaching 76.9%, while CCAFS SLC-40 has the lowest launch success rate. It indicates that the launch success rate may also depend on **the location and proximities of a launch site**. Taking CCAFS SLC-40 as an example, it is only 0.9km away from the closest coastline. Based on this finding, we can further explore how to optimize a launch site's location.
- Payload range from **2000-5500Kg** has the highest success rate.
- **Falcon booster version FT** has the highest launch success rate, reaching 57.1%, when the payload range is between 0-10000Kg. Whereas Booster Version B5 and v1.1 has a relatively low success rate.

Predictive analysis Results

To predict the landing outcome of Falcon 9's first stage, we can build a machine learning pipeline. The result shows that Decision Tree Classifier Model performs the best accuracy. In 83% cases, it predicts correctly(True Positive/ True Negative).

Thank you!