LANDING SUCCESS PREDICTION: OPTIMIZING SPACE Y'S ROCKET REUSABILITY

A Data-Driven Approach to Reduce Costs and Enhance Launch Efficiency

Brian Robertson

Data Scientist Team Lead at SPACE Y

10/18/2024

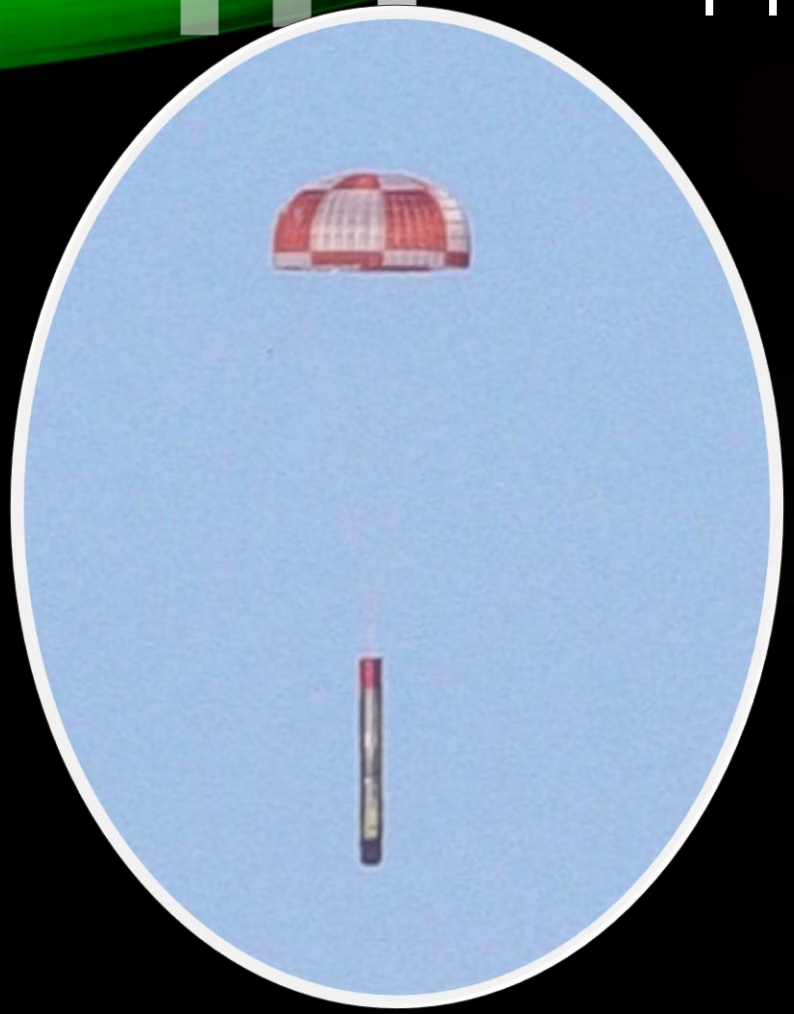
IBM Developer

SKILLS NETWORK

PROJECT AGENDA & KEY MILESTONES

- Phase 1: Project Setup & Data Collection (Weeks 1-2)
  - SpaceX API Integration
  - Web Scraping Implementation
  - Data Validation & Initial Processing
- Phase 2: Data Analysis & Preparation (Weeks 3-4)
  - Data Wrangling & Cleaning
  - Feature Engineering
  - Exploratory Data Analysis (EDA)
- Phase 3: Model Development (Weeks 5-6)
  - Machine Learning Model Implementation
  - Model Training & Optimization
  - Performance Evaluation
- Phase 4: Visualization & Results (Weeks 7-8)
  - Interactive Map Creation
  - Dashboard Development
  - Results Analysis
- Phase 5: Business Implementation (Weeks 9-10)
  - Cost Impact Analysis
  - Operational Integration
  - Future Recommendations

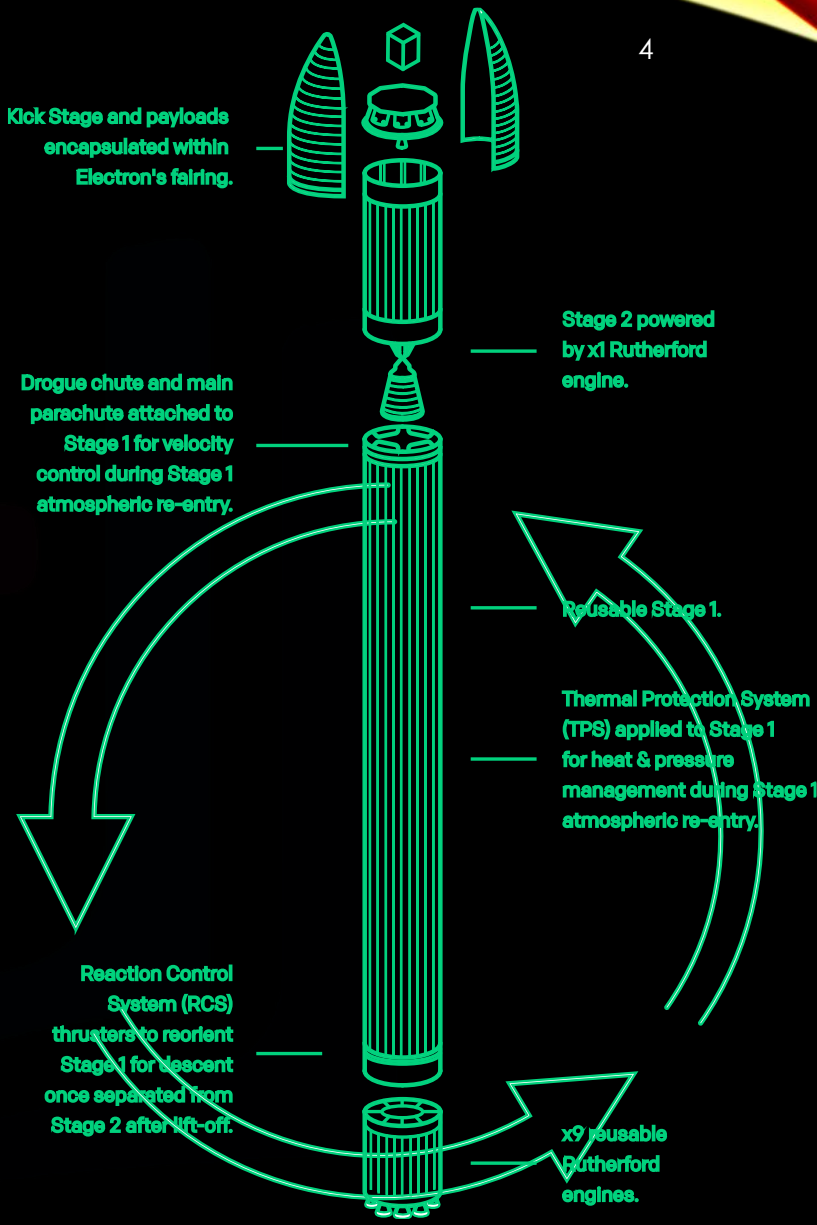
PROJECT OVERVIEW



- Project Scope & Objectives
- Methodology Framework
- Key Deliverables
- Expected Business Impact

INTRODUCTION

- SpaceX: Affordable space travel, reusable rockets, $62M per launch.
- Competitors:
  - Virgin Galactic: Suborbital spaceflights.
  - Rocket Lab: Small satellite launches.
  - Blue Origin: Reusable sub-orbital and orbital rockets.
- My Role at Space Y
  - Data Scientist at Space Y competing with SpaceX.
  - Using data science to predict first-stage landings.
- Critical for pricing and competitive cost strategies.



PROBLEM STATEMENT

- Space Y is striving to optimize rocket reusability, similar to SpaceX's success in reducing launch costs by recovering the first stage.
- The goal is to predict the likelihood of a successful landing, enabling Space Y to make data-driven decisions, minimize launch costs, and remain competitive.

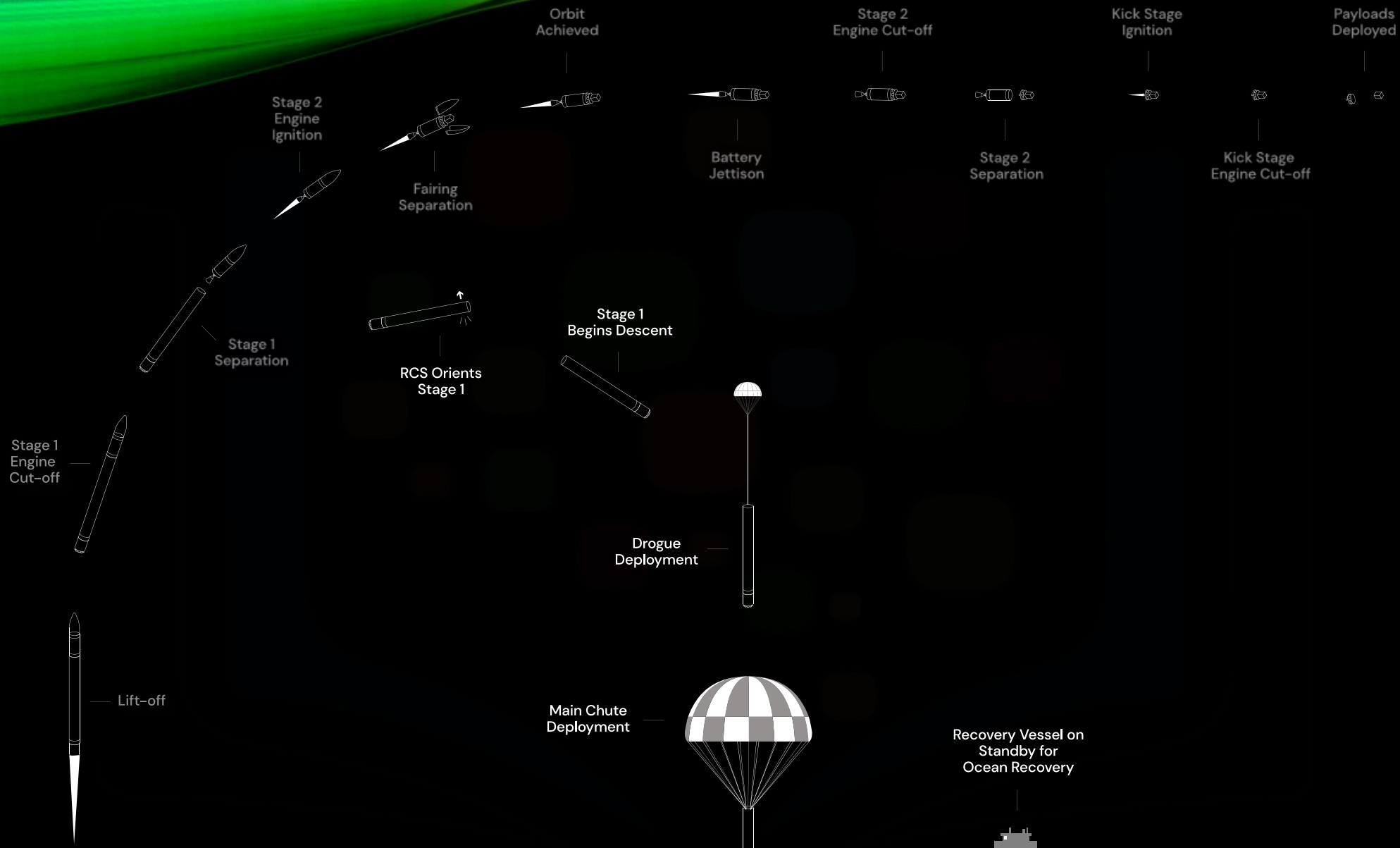
EXECUTIVE SUMMARY

Project Objective: Predict SpaceX Falcon 9 first stage landing success

Data Sources: SpaceX API, Wikipedia web scraping

Key Findings:

- Success rate increased from 75% (2013) to 90% (2024)
- Most successful launch site: CCAFS SLC 40 with 95% success rate
- Optimal payload mass range: 5000-6000 kg
- Best performing model: Decision Tree (88.89% accuracy)
- Business Impact: Potential cost reduction from \$62M to estimated $50 million per launch


Orbit Achieved
Stage 2 Engine Cut-off
Kick Stage Ignition
Payloads Deployed
Stage 2 Engine Ignition
Fairing Separation
Battery Jettison
Stage 2 Separation
Kick Stage Engine Cut-off
Stage 1 Separation
RCS Orients Stage 1
Stage 1 Begins Descent
Stage 1 Engine Cut-off
Drogue Deployment
Lift-off
Main Chute Deployment
Recovery Vessel on Standby for Ocean Recovery
Stage 1 Splashdown

METHODOLOGY: DATA COLLECTION

- API Data Collection
  - Source: SpaceX API * Endpoint: 'https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/datasets/API_call_spacex_api.json'
  - Data retrieved: Launch details, rocket specifications, payload information
  - Method: GET requests using requests library * Format: JSON response parsed to DataFrame
- Web Scraping Implementation
  - Source: Wikipedia * URL: "https://en.wikipedia.org/w/index.php?title=List_of_Falcon_9_and_Falcon_Heavy_launches" *
  - Method: BeautifulSoup4 for HTML parsing
  - Data extracted: Launch history, outcomes, additional mission details
  - Tables processed: Launch records, mission outcomes
- CSV Data Integration –
  - Files Processed:
    - dataset_part_1.csv: Primary launch data
    - dataset_part_2.csv: Additional features
    - dataset_part_3.csv: Supplementary information

DATA COLLECTION: API

From the rocket column we would like to learn the booster name.

```python
# 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']:
        if x:
            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.

```python
# 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']:
        if x:
            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.

```python
# 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']:
        if load:
            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 w
used, the landing pad used, the block of the core which is a number used to seperate version of cores, the n

```python
# 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'])
```

Now let's start requesting rocket launch data from SpaceX API with the following URL:

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

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

```python
# Use json_normalize meethod to convert the json result into a dataframe
import pandas as pd
import requests

# Define the URL for the JSON data
static_json_url = 'https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/datasets/API_call_spacex_api.json'

# Request the JSON data from the URL
response = requests.get(static_json_url)

# Check if the request was successful
if response.status_code == 200:
    # Load the JSON content directly
    data = response.json()

    # Convert the JSON content to a Pandas DataFrame
    df = pd.json_normalize(data)

    # Display the first few rows of the DataFrame
    print(df.head())
else:
    print(f"Failed to fetch data. Status code: {response.status_code}")
```

Pyth

```
       static_fire_date_utc  static_fire_date_unix    tbd    net  window  \
0  2006-03-17T00:00:00.000Z           1.142554e+09  False  False     0.0
1                      None                    NaN  False  False     0.0
2                      None                    NaN  False  False     0.0
3  2008-09-20T00:00:00.000Z           1.221869e+09  False  False     0.0
4                      None                    NaN  False  False     0.0

                     rocket  success  \
0  5e9d0d95eda69955f709d1eb    False
1  5e9d0d95eda69955f709d1eb    False
2  5e9d0d95eda69955f709d1eb    False
3  5e9d0d95eda69955f709d1eb     True
4  5e9d0d95eda69955f709d1eb     True

                                                                                                                                                              details
0                                                                                                                          Engine failure at 33 seconds and loss of vehicle
1  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
2                                                                                                                Residual stage 1 thrust led to collision between stage 1 and stage 2
3                        Ratsat was carried to orbit on the first successful orbital launch of any privately funded and developed, liquid-propelled carrier rocket, the SpaceX Falcon 1
4                                                                                                                                                                  None
```

SKILLS NETWORK

DATA COLLECTION: API

```python
import pandas as pd
import requests
import datetime

# Reload data into a DataFrame in case it was overwritten
static_json_url = 'https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/datasets/API_call_spacex_api.json'
response = requests.get(static_json_url)
data = pd.json_normalize(response.json())  # Reload as DataFrame

# Proceed with the steps as planned
data = data[['rocket', 'payloads', 'launchpad', 'cores', 'flight_number', 'date_utc']]

# Filter for single-core and single-payload rows only
data = data[data['cores'].map(len) == 1]
data = data[data['payloads'].map(len) == 1]

# Extract single values from list-type columns
data['cores'] = data['cores'].map(lambda x: x[0])
data['payloads'] = data['payloads'].map(lambda x: x[0])

# Convert `date_utc` to datetime and filter by date
data['date'] = pd.to_datetime(data['date_utc']).dt.date
data = data[data['date'] <= datetime.date(2020, 11, 13)]

# Display the first few rows to confirm the output
print(data.head())
```

```
                     rocket                   payloads  \
0  5e9d0d95eda69955f709d1eb  5eb0e4b5b6c3bb0006eeb1e1
1  5e9d0d95eda69955f709d1eb  5eb0e4b6b6c3bb0006eeb1e2
3  5e9d0d95eda69955f709d1eb  5eb0e4b7b6c3bb0006eeb1e5
4  5e9d0d95eda69955f709d1eb  5eb0e4b7b6c3bb0006eeb1e6
5  5e9d0d95eda69973a809d1ec  5eb0e4b7b6c3bb0006eeb1e7

                launchpad  \
0  5e9e4502f5090995de566f86
1  5e9e4502f5090995de566f86
3  5e9e4502f5090995de566f86
4  5e9e4502f5090995de566f86
5  5e9e4501f509094ba4566f84
```

```python
import requests
import pandas as pd

# Initialize global lists
BoosterVersion = []
LaunchSite = []
Longitude = []
Latitude = []
PayloadMass = []
Orbit = []
Outcome = []
Flights = []
GridFins = []
Reused = []
Legs = []
LandingPad = []
Block = []
ReusedCount = []
Serial = []

def getBoosterVersion(data):
    for rocket in data['rocket']:
        response = requests.get(f"https://api.spacexdata.com/v4/rockets/{rocket}")
        if response.status_code == 200:
            rocket_info = response.json()
            BoosterVersion.append(rocket_info['name'])
        else:
            BoosterVersion.append(None)

def getLaunchSite(data):
    for launchpad in data['launchpad']:
        response = requests.get(f"https://api.spacexdata.com/v4/launchpads/{launchpad}")
        if response.status_code == 200:
            launchpad_info = response.json()
            LaunchSite.append(launchpad_info['name'])
            Longitude.append(launchpad_info['longitude'])
            Latitude.append(launchpad_info['latitude'])
        else:
            LaunchSite.append(None)
            Longitude.append(None)
            Latitude.append(None)

def getPayloadData(data):
    for payload in data['payloads']:
```

IBM Developer

SKILLS NETWORK

DATA COLLECTION: API

```
def getCoreData(data):
    for core in data['cores']:
        response = requests.get(f"https://api.spacexdata.com/v4/cores/{core}")
        if response.status_code == 200:
            core_info = response.json()
            Outcome.append(core_info.get('last_update', 'N/A'))
            Flights.append(core_info.get('reuse_count'))
            GridFins.append(core_info.get('gridfins'))
            Reused.append(core_info.get('reused'))
            Legs.append(core_info.get('legs'))
            LandingPad.append(core_info.get('landpad'))
            Block.append(core_info.get('block'))
            ReusedCount.append(core_info.get('reuse_count'))
            Serial.append(core_info.get('serial'))
        else:
            Outcome.append(None)
            Flights.append(None)
            GridFins.append(None)
            Reused.append(None)
            Legs.append(None)
            LandingPad.append(None)
            Block.append(None)
            ReusedCount.append(None)
            Serial.append(None)

def processLaunchData(data):
    getBoosterVersion(data)
    getLaunchSite(data)
    getPayloadData(data)
    getCoreData(data)

    launch_dict = {
        'FlightNumber': data['flight_number'],
        'Date': 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,
```

```
# Show the head of the dataframe
# Display the first few rows of the dataframe to review the current state after extraction
df.head()
```

Pytho

	static_fire_date_utc	static_fire_date_unix	tbd	net	window	rocket	success	details	crew	ships	capsules	payloads	launchpad	au
0	2006-03-17T00:00:00.000Z	1.142554e+09	False	False	0.0	5e9d0d95eda69955f709d1eb	False	Engine failure at 33 seconds and loss of vehicle	[]	[]	[]	[5eb0e4b5b6c3bb0006eeb1e1]	5e9e4502f5090995de566f86	
1	None	NaN	False	False	0.0	5e9d0d95eda69955f709d1eb	False	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	[]	[]	[]	[5eb0e4b6b6c3bb0006eeb1e2]	5e9e4502f5090995de566f86	
2	None	NaN	False	False	0.0	5e9d0d95eda69955f709d1eb	False	Residual stage 1 thrust led to collision between stage 1 and stage 2	[]	[]	[]	[5eb0e4b6b6c3bb0006eeb1e3, 5eb0e4b6b6c3bb0006eeb1e4]	5e9e4502f5090995de566f86	

SKILLS NETWORK

DATA COLLECTION: API

```
# Hint data['BoosterVersion']!='Falcon 1'
# Call the function and assign the result to 'launch_df'
launch_df = processLaunchData(data)
# Filter the dataframe to keep only Falcon 9 launches
data_falcon9 = launch_df[launch_df['BoosterVersion'] != 'Falcon 1']
```

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

```
# Create a copy of the filtered DataFrame
data_falcon9 = launch_df[launch_df['BoosterVersion'] != 'Falcon 1'].copy()

# Reset the FlightNumber column
data_falcon9.loc[:, 'FlightNumber'] = range(1, data_falcon9.shape[0] + 1)

# Display the DataFrame
print(data_falcon9.head())
```

```
    FlightNumber        Date BoosterVersion  PayloadMass Orbit    LaunchSite  \
5              1  2010-06-04       Falcon 9          NaN   LEO  CCSFS SLC 40
7              2  2012-05-22       Falcon 9        525.0   LEO  CCSFS SLC 40
9              3  2013-03-01       Falcon 9        677.0   ISS  CCSFS SLC 40
10             4  2013-09-29       Falcon 9        500.0    PO   VAFB SLC 4E
11             5  2013-12-03       Falcon 9       3170.0   GTO  CCSFS SLC 40

   Outcome Flights GridFins Reused   Legs LandingPad Block ReusedCount Serial  \
5     None    None     None   None  None       None  None        None   None
7     None    None     None   None  None       None  None        None   None
9     None    None     None   None  None       None  None        None   None
10    None    None     None   None  None       None  None        None   None
11    None    None     None   None  None       None  None        None   None

     Longitude   Latitude
5   -80.577366  28.561857
7   -80.577366  28.561857
9   -80.577366  28.561857
10 -120.610829  34.632093
11  -80.577366  28.561857
```

SKILLS NETWORK

DATA COLLECTION: WEBSCRAPPING

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

```
<table class="wikitable plainrowheaders collapsible" style="width: 100%;">
<tbody><tr>
<th scope="col">Flight No.
</th>
<th scope="col">Date and<br/>time (<a href="/wiki/Coordinated_Universal_Time" title="Coordinated Universal Time">UTC</a>)
</th>
<th scope="col"><a href="/wiki/List_of_Falcon_9_first-stage_boosters" title="List of Falcon 9 first-stage boosters">Version,<br/>Booster</a> <sup class="reference
</th>
<th scope="col">Launch site
</th>
<th scope="col">Payload<sup class="reference" id="cite_ref-Dragon_12-0"><a href="#cite_note-Dragon-12"><span class="cite-bracket">[</span>c<span class="cite-brack
```

```
column_names = []

# Initialize an empty list for column names
column_names = []

# Find all 'th' elements in the first_launch_table
table_headers = first_launch_table.find_all('th')

# Iterate over each 'th' element
for th in table_headers:
    name = extract_column_from_header(th)
    # Append the non-empty column name into column_names
    if name is not None and len(name) > 0:
        column_names.append(name)
```

Check the extracted column names

```
print(column_names)
```

```
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']=[]
```

```
# Extract each table
for table_number, table in enumerate(soup.find_all('table', "wikitable plainrowheaders collapsible")):
    # Get table rows
    for rows in table.find_all("tr"):
        # Check to see if first table heading is a number corresponding to launch number
        if rows.th:
            if rows.th.string:
                flight_number = rows.th.string.strip()
                flag = flight_number.isdigit()
            else:
                flag = False
        else:
            flag = False

        # Get table cells
        row = rows.find_all('td')

        # If it's a valid launch entry, save cells in the dictionary
        if flag:
            extracted_row += 1

            # Flight Number value
            # Append the flight_number into launch_dict with key 'Flight No.'
            launch_dict['Flight No.'].append(flight_number)

            datatimelist = date_time(row[0])

            # Date value
            # Append the date into launch_dict with key 'Date'
            date = datatimelist[0].strip(',')
            launch_dict['Date'].append(date)

            # Time value
            # Append the time into launch_dict with key 'Time'
            time = datatimelist[1]
            launch_dict['Time'].append(time)

            # Booster version
            bv = booster_version(row[1])
            if not bv:
                bv = row[1].a.string
            # Append the bv into launch_dict with key 'Version Booster'
            launch_dict['Version Booster'].append(bv.strip())

            # Launch Site
```

METHODOLOGY: DATA WRANGLING

```
data_falcon9.isnull().sum()
```

FlightNumber	0
Date	0
BoosterVersion	0
PayloadMass	5
Orbit	0
LaunchSite	0
Outcome	90
Flights	90
GridFins	90
Reused	90
Legs	90
LandingPad	90
Block	90
ReusedCount	90
Serial	90
Longitude	0
Latitude	0

dtype: int64

Missing Value Treatment

- PayloadMass:
  - Identified NaN values
  - Applied mean imputation
  - Validated reasonable range
- Categorical Variables:
  - Orbit type: Mode imputation
  - Launch site: No missing values
  - Landing outcome: Binary classification

Feature Engineering

- Date Processing:
  - Extracted launch year
  - Created temporal features
  - Analyzed launch frequency
- Categorical Encoding:
  - OneHotEncoder for launch sites
  - Label encoding for orbit types
  - Binary encoding for landing outcomes

Data Validation Steps

- Range Checks:
  - Payload mass within specifications
  - Valid launch dates
  - Consistent site coordinates

Final Dataset Structure

- Features included:
  - Flight Number
  - Launch Site
  - Payload Mass
  - Orbit
  - Landing Outcome

IBM Developer

SKILLS NETWORK

DATA WRANGLING: MISSING VALUES

```
# Load the data
static_json_url = 'https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/I
response = requests.get(static_json_url)
data = pd.json_normalize(response.json())

# Select relevant columns and preprocess data
data = data[['rocket', 'payloads', 'launchpad', 'cores', 'flight_number', 'date_utc']]
data = data[data['cores'].map(len) == 1]
data = data[data['payloads'].map(len) == 1]
data['cores'] = data['cores'].map(lambda x: x[0])
data['payloads'] = data['payloads'].map(lambda x: x[0])
data['date'] = pd.to_datetime(data['date_utc']).dt.date
data = data[data['date'] <= datetime.date(2020, 11, 13)]

# Initialize lists to store extracted data
BoosterVersion = []
PayloadMass = []
Orbit = []
LaunchSite = []
Outcome = []
Flights = []
GridFins = []
Reused = []
Legs = []
LandingPad = []
Block = []
ReusedCount = []
Serial = []
Longitude = []
Latitude = []

# Define helper functions
def getBoosterVersion(rocket):
    response = requests.get(f"https://api.spacexdata.com/v4/rockets/{rocket}")
    return response.json()['name'] if response.status_code == 200 else None

def getLaunchSite(launchpad):
    response = requests.get(f"https://api.spacexdata.com/v4/launchpads/{launchpad}")
    if response.status_code == 200:
        site_info = response.json()
        return site_info['name'], site_info['longitude'], site_info['latitude']
    return None, None, None

def getPayloadData(payload):
    response = requests.get(f"https://api.spacexdata.com/v4/payloads/{payload}")
```

```
def getPayloadData(payload):
    response = requests.get(f"https://api.spacexdata.com/v4/payloads/{payload}")
    if response.status_code == 200:
        payload_info = response.json()
        return payload_info.get('mass_kg'), payload_info.get('orbit')
    return None, None

def getCoreData(core):
    response = requests.get(f"https://api.spacexdata.com/v4/cores/{core}")
    if response.status_code == 200:
        core_info = response.json()
        return (core_info.get('last_update'), core_info.get('reuse_count'),
                core_info.get('gridfins'), core_info.get('reused'),
                core_info.get('legs'), core_info.get('landpad'),
                core_info.get('block'), core_info.get('reuse_count'),
                core_info.get('serial'))
    return (None,) * 9

# Process launch data
for _, row in data.iterrows():
    BoosterVersion.append(getBoosterVersion(row['rocket']))
    site, lon, lat = getLaunchSite(row['launchpad'])
    LaunchSite.append(site)
    Longitude.append(lon)
    Latitude.append(lat)
    mass, orbit = getPayloadData(row['payloads'])
    PayloadMass.append(mass)
    Orbit.append(orbit)
    outcome, flights, gridfins, reused, legs, landpad, block, reused_count, serial = getCoreData(row['cores'])
    Outcome.append(outcome)
    Flights.append(flights)
    GridFins.append(gridfins)
    Reused.append(reused)
    Legs.append(legs)
    LandingPad.append(landpad)
    Block.append(block)
    ReusedCount.append(reused_count)
    Serial.append(serial)

# Create the launch_dict and convert to DataFrame
launch_dict = {
    'FlightNumber': list(data['flight_number']),
    'Date': list(data['date']),
    'BoosterVersion': BoosterVersion,
```

```
    # Ensure all lists have the same length
    min_length = min(len(v) for v in launch_dict.values())
    launch_dict = {k: v[:min_length] for k, v in launch_dict.items()}

    launch_df = pd.DataFrame(launch_dict)

    # Display the first few rows of the dataframe
    print(launch_df.head())

    # Task 2: Filter the dataframe to only include Falcon 9 launches
    data_falcon9 = launch_df[launch_df['BoosterVersion'] != 'Falcon 1'].copy()
    print(data_falcon9.head())
    print(f"Number of Falcon 9 launches: {len(data_falcon9)}")

    # Reset the FlightNumber column
    data_falcon9.loc[:, 'FlightNumber'] = range(1, data_falcon9.shape[0] + 1)

    # Task 3: Dealing with Missing Values
    print("\nMissing values before filling:")
    print(data_falcon9.isnull().sum())

    # Calculate the mean value of PayloadMass column
    payload_mass_mean = data_falcon9['PayloadMass'].mean()

    # Replace the np.nan values with its mean value
    data_falcon9['PayloadMass'] = data_falcon9['PayloadMass'].fillna(payload_mass_mean)

    # Verify that missing values have been addressed
    print("\nMissing values after filling:")
    print(data_falcon9.isnull().sum())

    # Export the dataframe to a CSV file
    data_falcon9.to_csv('dataset_part_1.csv', index=False)

    print("\nData processing and export completed successfully.")
```

DATA WRANGLING: MISSING VALUES

```
   FlightNumber        Date BoosterVersion  PayloadMass Orbit  \
0             1  2006-03-24       Falcon 1         20.0   LEO
1             2  2007-03-21       Falcon 1          NaN   LEO
2             4  2008-09-28       Falcon 1        165.0   LEO
3             5  2009-07-13       Falcon 1        200.0   LEO
4             6  2010-06-04       Falcon 9          NaN   LEO

       LaunchSite Outcome Flights GridFins Reused  Legs LandingPad Block  \
0  Kwajalein Atoll    None    None     None   None  None       None  None
1  Kwajalein Atoll    None    None     None   None  None       None  None
2  Kwajalein Atoll    None    None     None   None  None       None  None
3  Kwajalein Atoll    None    None     None   None  None       None  None
4     CCSFS SLC 40    None    None     None   None  None       None  None

  ReusedCount Serial   Longitude   Latitude
0        None   None  167.743129   9.047721
1        None   None  167.743129   9.047721
2        None   None  167.743129   9.047721
3        None   None  167.743129   9.047721
4        None   None  -80.577366  28.561857
   FlightNumber        Date BoosterVersion  PayloadMass Orbit    LaunchSite  \
4             6  2010-06-04       Falcon 9          NaN   LEO  CCSFS SLC 40
5             8  2012-05-22       Falcon 9        525.0   LEO  CCSFS SLC 40
6            10  2013-03-01       Falcon 9        677.0   ISS  CCSFS SLC 40
7            11  2013-09-29       Falcon 9        500.0    PO   VAFB SLC 4E
...
Latitude           0
dtype: int64

Data processing and export completed successfully.
```

```
print(data_falcon9.isnull().sum())
```

```
FlightNumber      0
Date              0
BoosterVersion    0
PayloadMass       0
Orbit             0
LaunchSite        0
Outcome          90
Flights          90
GridFins         90
Reused           90
Legs             90
LandingPad       90
Block            90
ReusedCount      90
Serial           90
Longitude         0
Latitude          0
dtype: int64
```

SKILLS NETWORK

DATA WRANGLING: FEATURE ENGINEERING

```
# Use value_counts() on the 'Orbit' colum
orbit_counts = df['Orbit'].value_counts()

# Display the counts
print(orbit_counts)
```

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

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

# Display the counts
print(landing_outcomes)
```

```
Outcome
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: count, dtype: int64
```

```
df['Class']=landing_class
df[['Class']].head(8)
```

	Class
0	0
1	0
2	0
3	0
4	0
5	0
6	1
7	1

```
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
```

IBM Developer

DATA WRANGLING: FEATURE ENGINEERING

Features Engineering

By now, you should obtain some preliminary insights about how each important variable would affect the success rate, we will select the features that will be used in success prediction in the future module.

```
features = df[['FlightNumber', 'PayloadMass', 'Orbit', 'LaunchSite', 'Flights', 'GridFins', 'Reused', 'Legs', 'LandingPad', 'Block', 'ReusedCount', 'Serial']]
features.head()
```

Python

	FlightNumber	PayloadMass	Orbit	LaunchSite	Flights	GridFins	Reused	Legs	LandingPad	Block	ReusedCount	Serial
0	1	6104.959412	LEO	CCAFS SLC 40	1	False	False	False	NaN	1.0	0	B0003
1	2	525.000000	LEO	CCAFS SLC 40	1	False	False	False	NaN	1.0	0	B0005
2	3	677.000000	ISS	CCAFS SLC 40	1	False	False	False	NaN	1.0	0	B0007
3	4	500.000000	PO	VAFB SLC 4E	1	False	False	False	NaN	1.0	0	B1003
4	5	3170.000000	GTO	CCAFS SLC 40	1	False	False	False	NaN	1.0	0	B1004

```
# HINT: Use get_dummies() function on the categorical columns
# Select the features as indicated
features = df[['FlightNumber', 'PayloadMass', 'Orbit', 'LaunchSite', 'Flights', 'GridFins', 'Reused', 'Legs', 'LandingPad', 'Block', 'ReusedCount', 'Serial']]

# Create dummy variables for categorical columns using get_dummies()
features_one_hot = pd.get_dummies(features, columns=['Orbit', 'LaunchSite', 'LandingPad', 'Serial'], drop_first=True)

# Display the first few rows of the resulting dataframe
features_one_hot.head()
```

Python

	FlightNumber	PayloadMass	Flights	GridFins	Reused	Legs	Block	ReusedCount	Orbit_GEO	Orbit_GTO	...	Serial_B1048	Serial_B1049	Serial_B1050	Serial_B1051	Serial_B1054	Serial_B1056
0	1	6104.959412	1	False	False	False	1.0	0	False	False	...	False	False	False	False	False	False
1	2	525.000000	1	False	False	False	1.0	0	False	False	...	False	False	False	False	False	False
2	3	677.000000	1	False	False	False	1.0	0	False	False	...	False	False	False	False	False	False
3	4	500.000000	1	False	False	False	1.0	0	False	False	...	False	False	False	False	False	False
4	5	3170.000000	1	False	False	False	1.0	0	False	True	...	False	False	False	False	False	False

DATA WRANGLING: ENCODING

Load Space X dataset, from last section.

```
df=pd.read_csv("https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/datasets/dataset_part_1.csv")
df.head(10)
```

Python

	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins	Reused	Legs	LandingPad	Block	ReusedCount	Serial	Longitude	Latitude
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
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
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
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
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
5	6	2014-01-06	Falcon 9	3325.000000	GTO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B1005	-80.577366	28.561857
6	7	2014-04-18	Falcon 9	2296.000000	ISS	CCAFS SLC 40	True Ocean	1	False	False	True	NaN	1.0	0	B1006	-80.577366	28.561857
7	8	2014-07-14	Falcon 9	1316.000000	LEO	CCAFS SLC 40	True Ocean	1	False	False	True	NaN	1.0	0	B1007	-80.577366	28.561857
8	9	2014-08-05	Falcon 9	4535.000000	GTO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B1008	-80.577366	28.561857
9	10	2014-09-07	Falcon 9	4428.000000	GTO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B1011	-80.577366	28.561857

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

# Display the counts
print(landing_outcomes)
```

```
Outcome
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: count, dtype: int64
```

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

# Display the counts
print(launch_site_counts)
```

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

```
# Use value_counts() on the 'Orbit' column
orbit_counts = df['Orbit'].value_counts()

# Display the counts
print(orbit_counts)
```

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

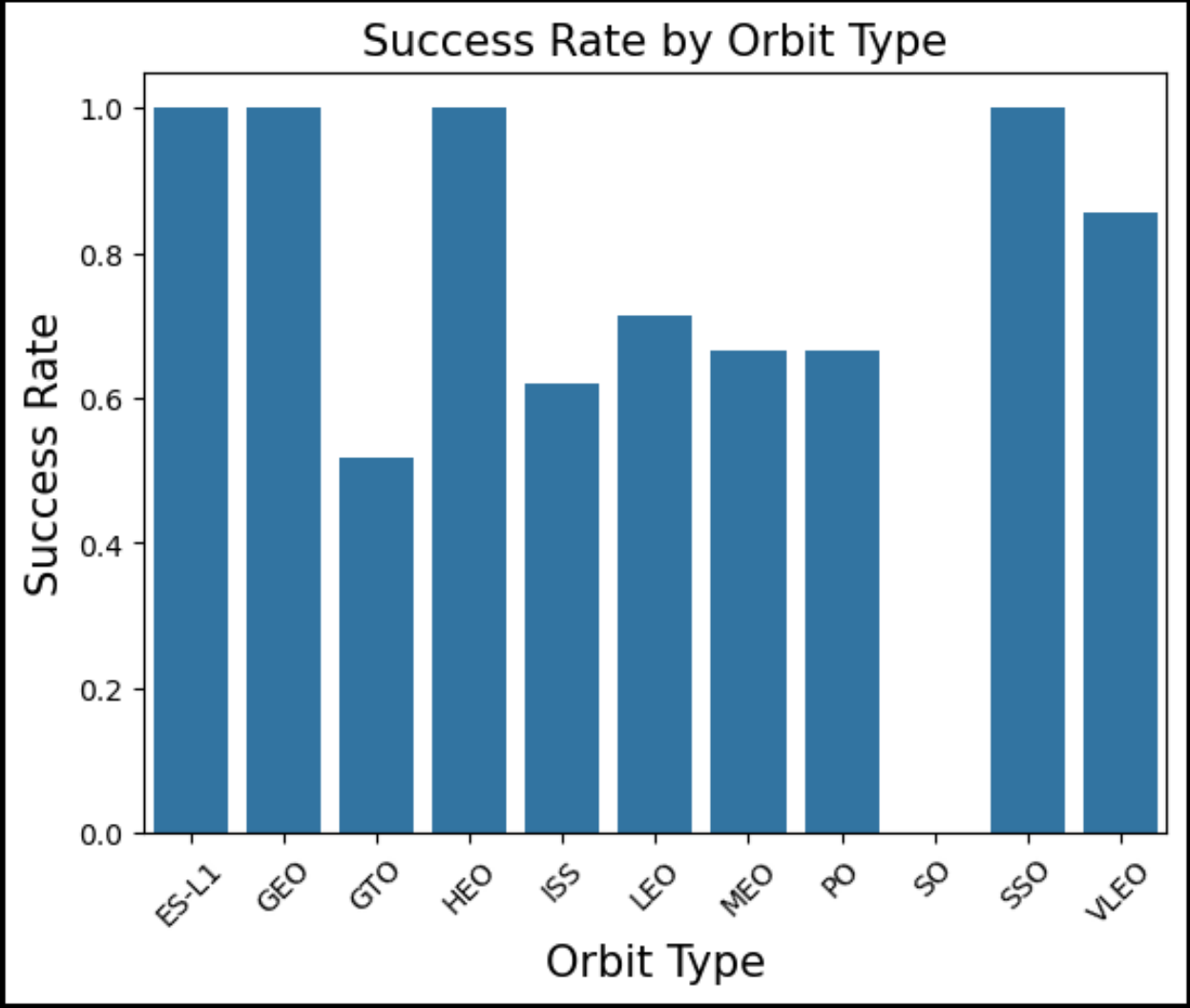
```
# landing_class = 0 if bad_outcome
# landing_class = 1 otherwise
# Create the landing_class list using the apply method
landing_class = df['Outcome'].apply(lambda x: 0 if x in bad_outcomes else 1).tolist()
```

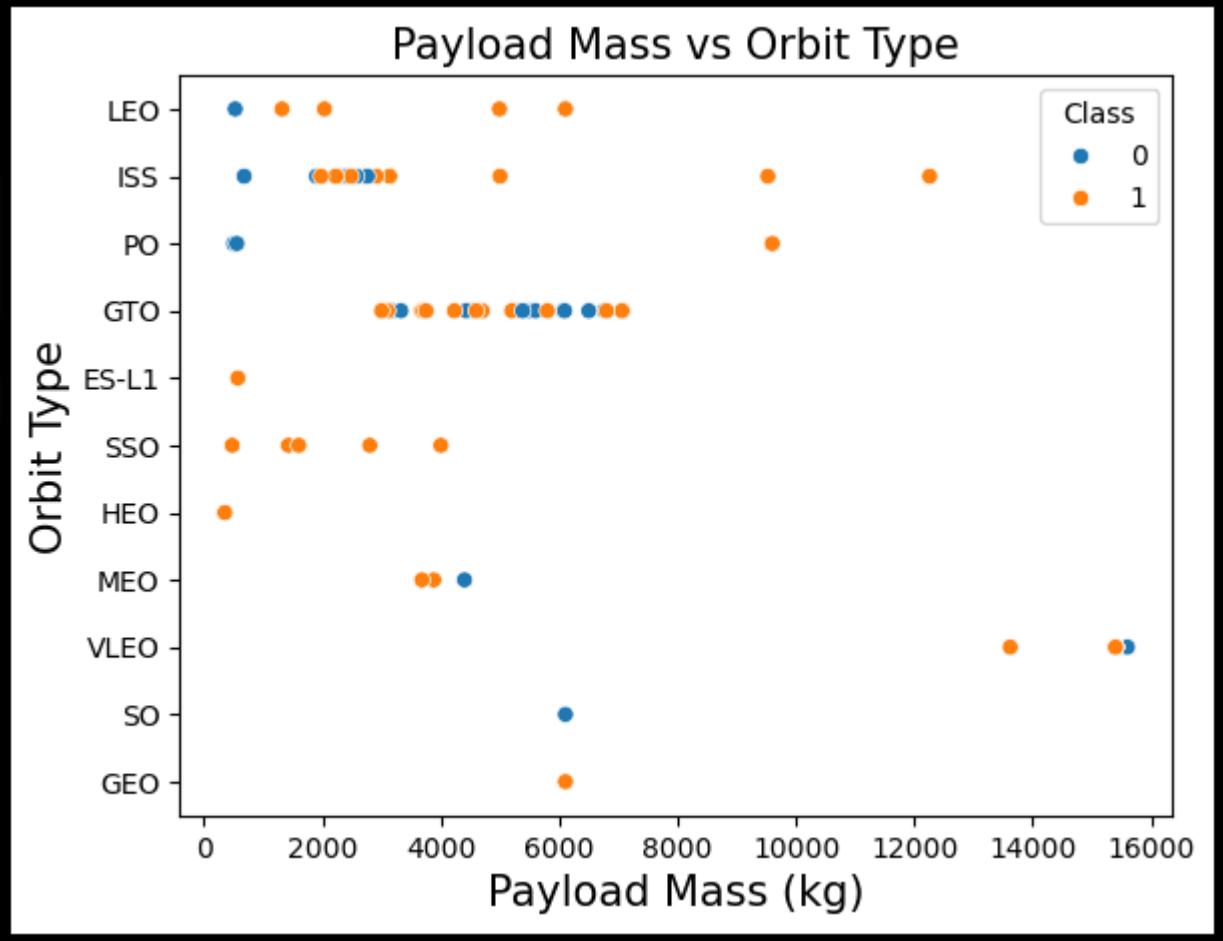
This variable will represent the classification variable that represents the outcome of each lau
uccessfully

```
df['Class']=landing_class
df[['Class']].head(8)
```

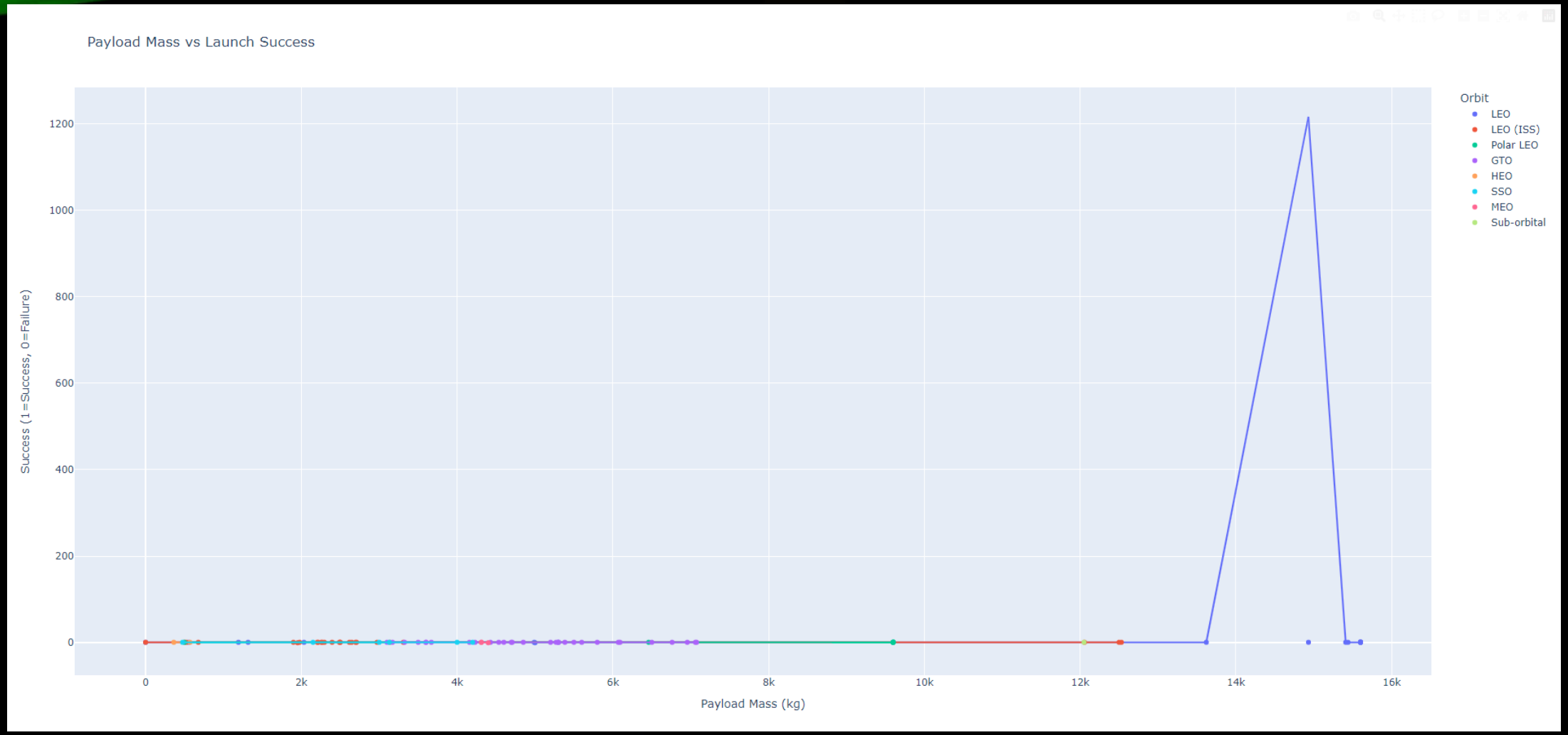
	Class
0	0
1	0
2	0
3	0
4	0
5	0
6	1
7	1

EXPLORATORY DATA ANALYSIS PART 1

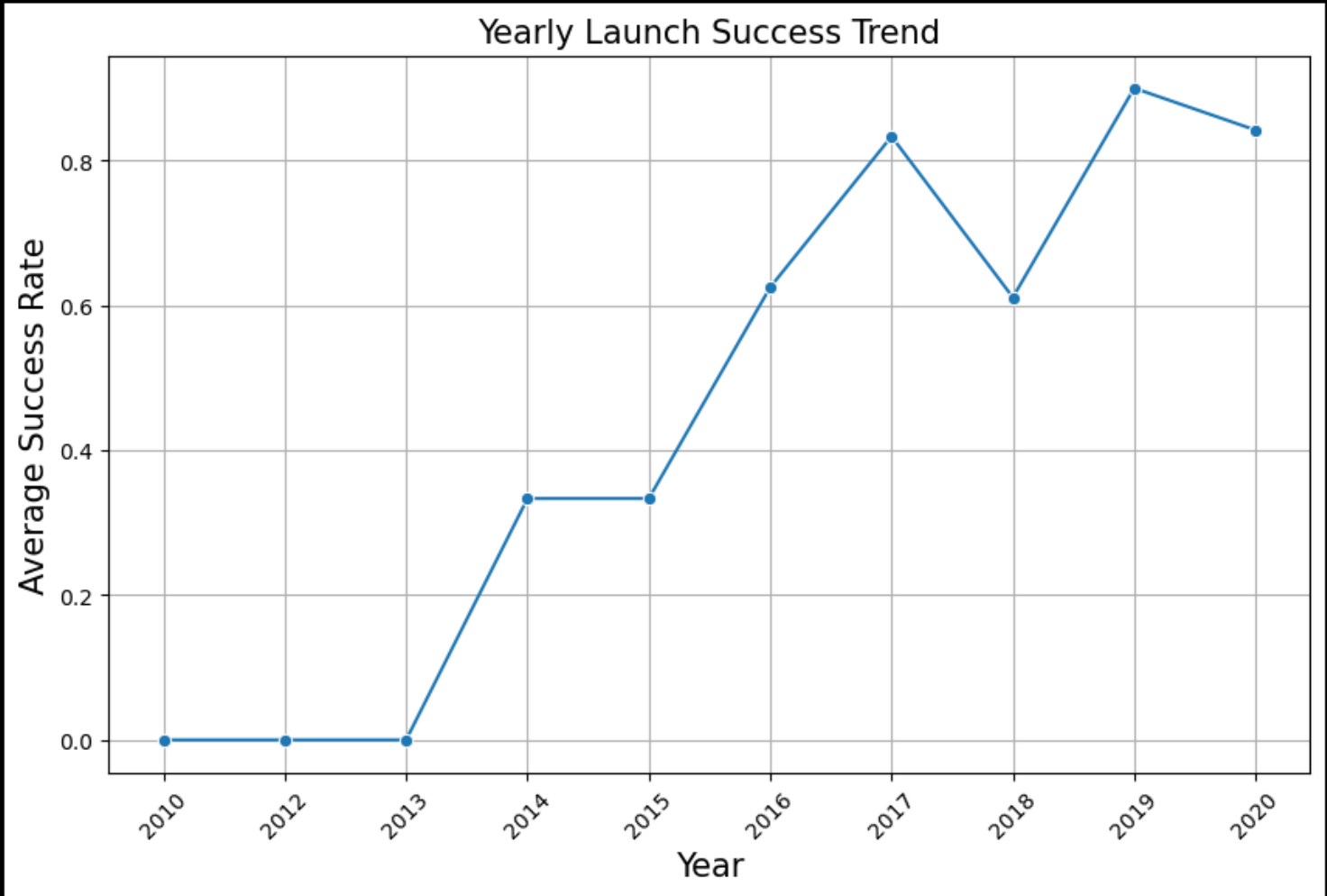




EXPLORATORY DATA ANALYSIS PART 2



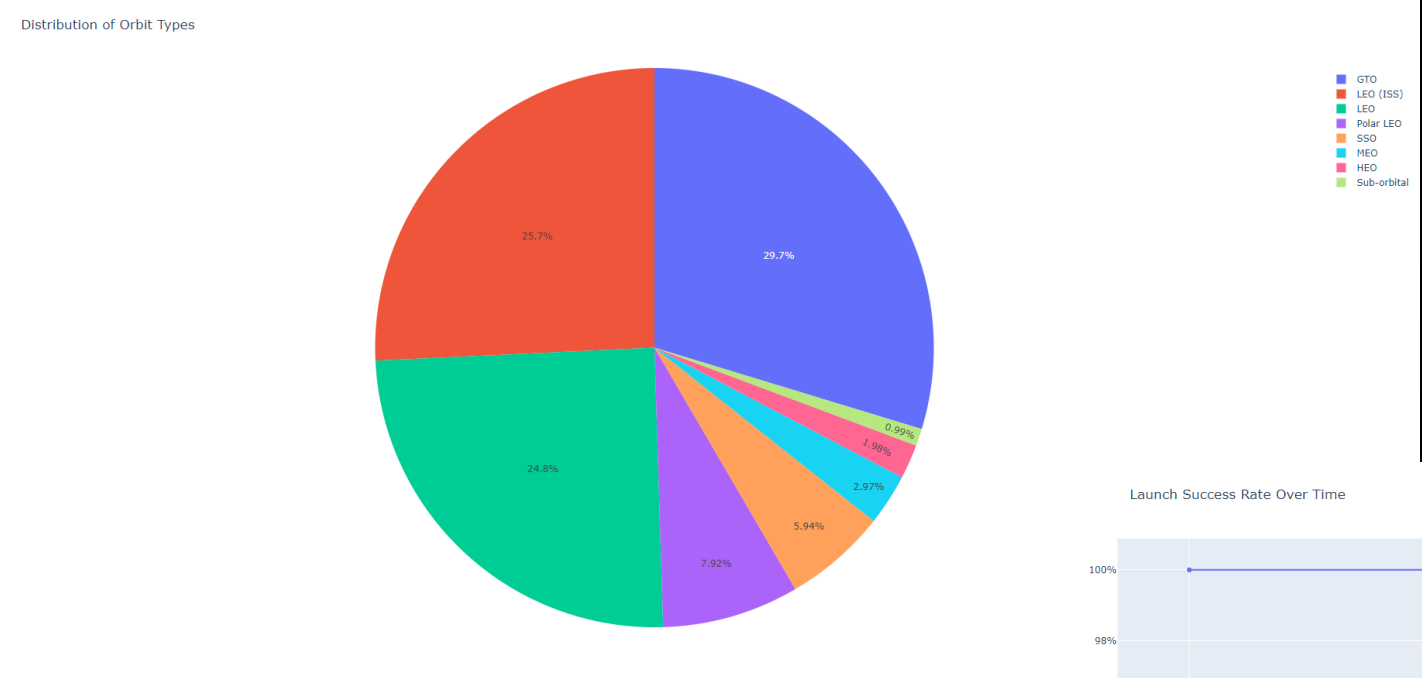
EXPLORATORY DATA ANALYSIS PART 3



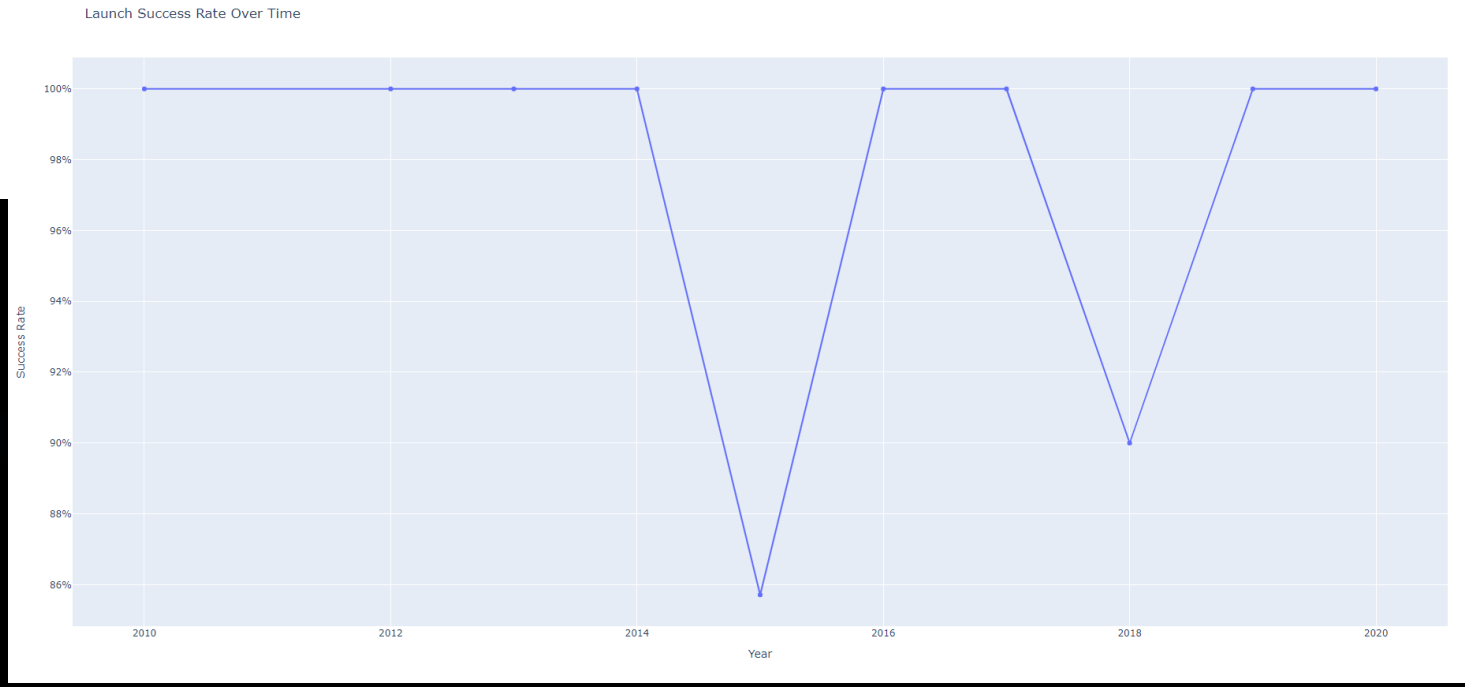
Key Trends Observed:

- Launch success rates have improved significantly over the years.
- Increased investment in technology correlates with higher success rates.
- Certain launch sites consistently outperform others

EXPLORATORY DATA ANALYSIS PART 4



- Trends over time, showing how launch success improved over the years.
- Continues to improve as technology improves



IBM Developer

METHODOLOGY: MACHINE LEARNING MODELS

- **Models & Techniques**
  - Logistic Regression: Simple and interpretable for binary outcomes.
  - Decision Trees: Captures non-linear relationships effectively.
  - K-Nearest Neighbors (KNN): Useful for small datasets with clear clusters.
  - Support Vector Machines (SVM): Effective in high-dimensional spaces.
- **Optimization**
  - Used **GridSearchCV** for hyperparameter tuning.
  - **Cross-Validation (cv=10)** for accuracy improvement.

SKILLS NETWORK

MLMS : LOGISTIC REGRESSION

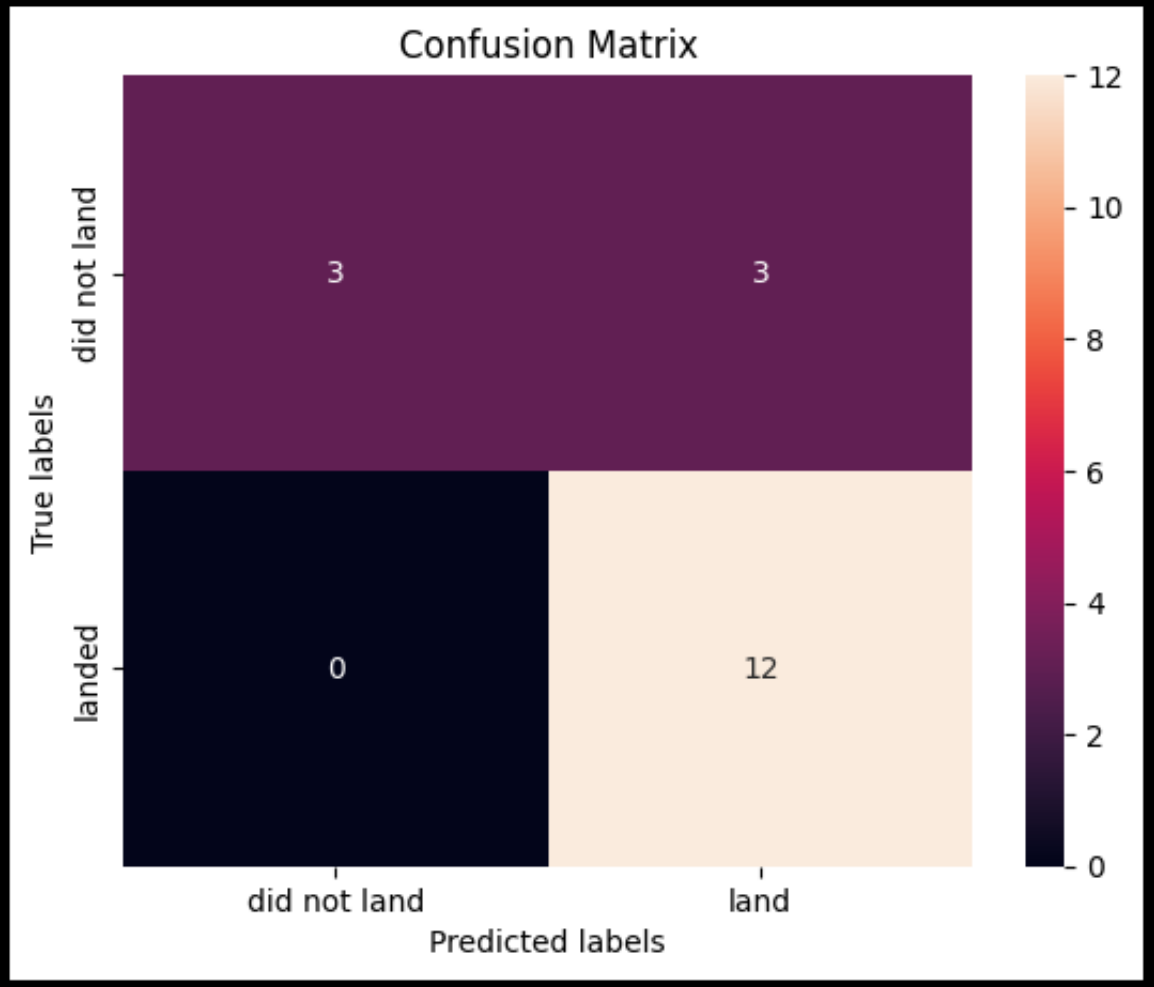
- **Models & Techniques**
  - Logistic Regression

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

```
parameters ={"C":[0.01,0.1,1],'penalty':['l2'], 'solver':['lbfgs']}# l1 lasso l2 ridge
# Create the logistic regression object
lr = LogisticRegression()

# Create GridSearchCV object with 10-fold cross-validation
logreg_cv = GridSearchCV(estimator=lr, param_grid=parameters, cv=10)

# Fit the model
logreg_cv.fit(X_train, Y_train)
```

```
print("tuned hpyerparameters :(best parameters) ",logreg_cv.best_params_)
print("accuracy :",logreg_cv.best_score_)
```

```
tuned hpyerparameters :(best parameters)  {'C': 0.01, 'penalty': 'l2', 'solver': 'lbfgs'}
accuracy : 0.8464285714285713
```



MLMS : DECISION TREES

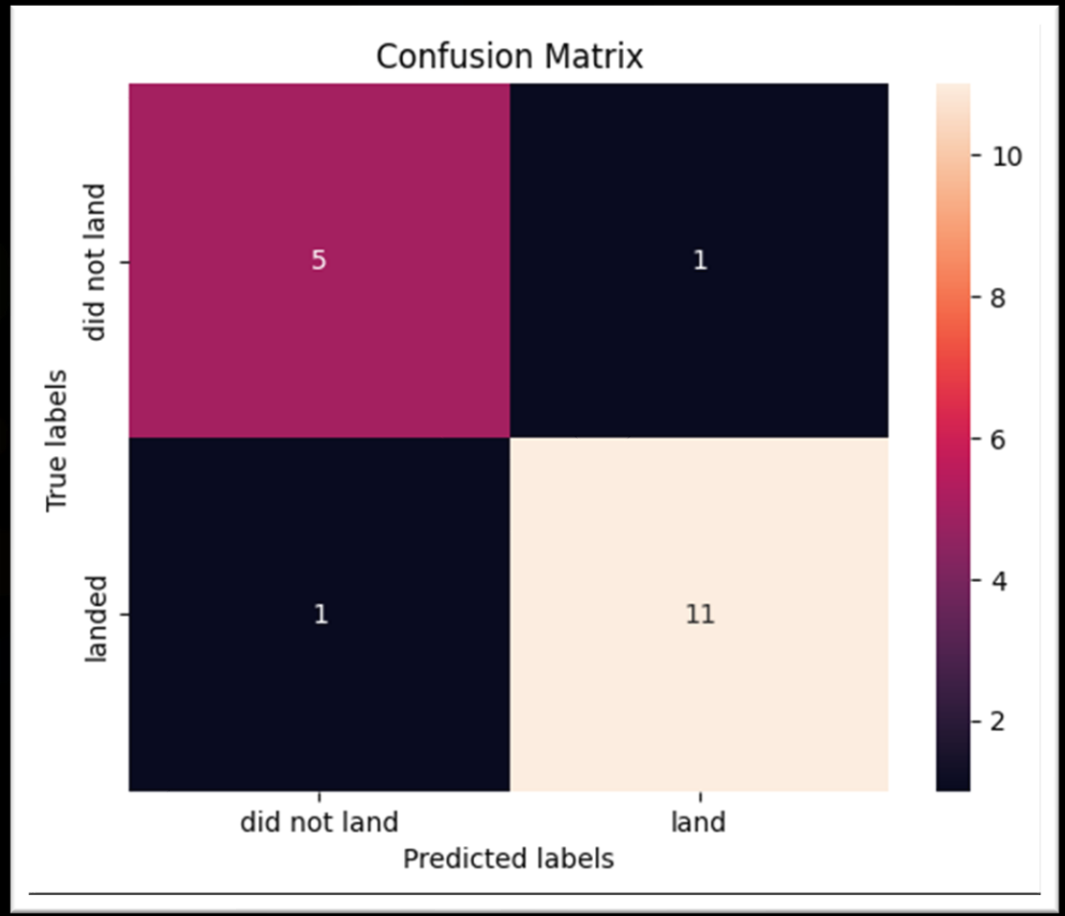

```
# Create the parameter dictionary with corrected max_features values
parameters = {'criterion': ['gini', 'entropy'],
              'splitter': ['best', 'random'],
              'max_depth': [2*n for n in range(1,10)],
              'max_features': ['sqrt', 'log2'],  # Changed 'auto' to valid options
              'min_samples_leaf': [1, 2, 4],
              'min_samples_split': [2, 5, 10]}
# Create Decision Tree classifier object
tree = DecisionTreeClassifier()
```

```
# Create Decision Tree classifier object
tree = DecisionTreeClassifier()

# Create GridSearchCV object
tree_cv = GridSearchCV(tree, parameters, cv=10)

# Fit the model
tree_cv.fit(X_train, Y_train)
```

```
print("tuned hpyerparameters :(best parameters) ",tree_cv.best_params_)
print("accuracy :",tree_cv.best_score_)
```

```
tuned hpyerparameters :(best parameters)  {'criterion': 'entropy', 'max_depth'
accuracy : 0.8892857142857142
```

- **Models & Techniques**
  - Decision Trees



MLMS : K-NEAREST NEIGHBORS (KNN)

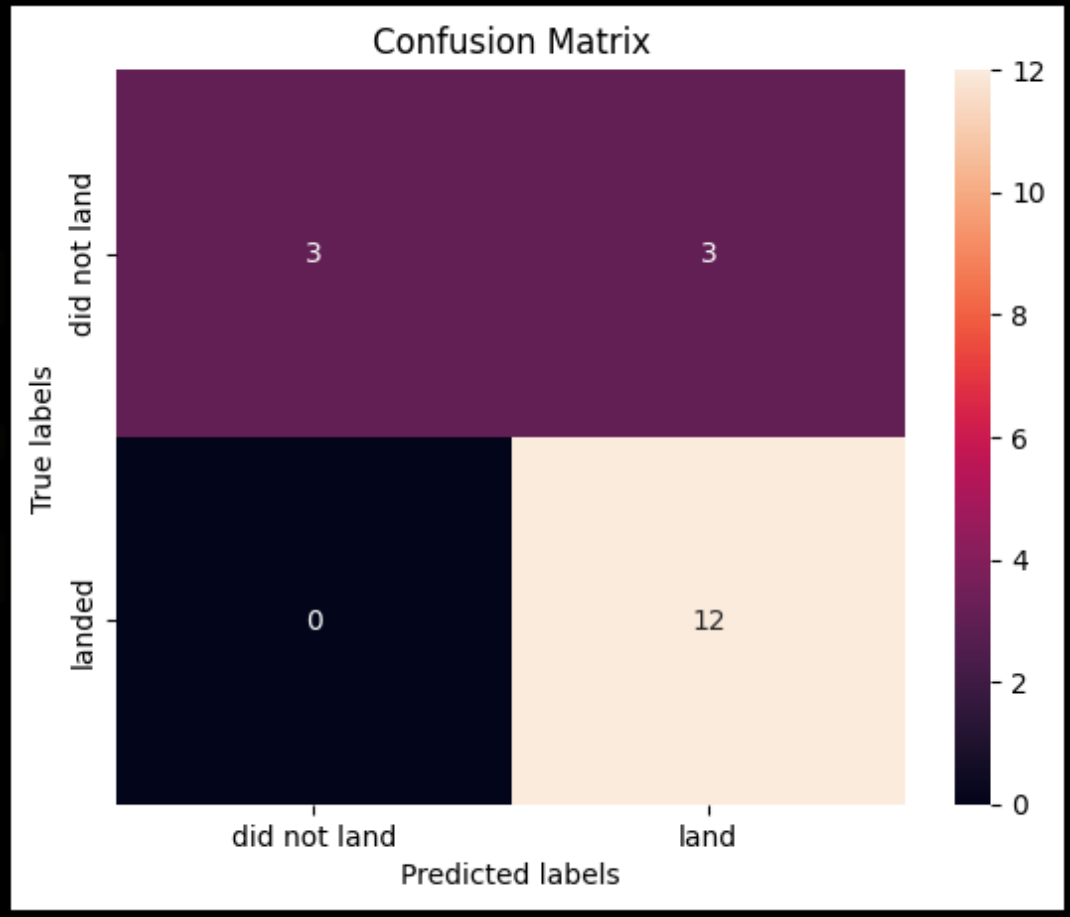
- **Models & Techniques**
  - K-Nearest Neighbors (KNN)

```
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()
```

```
# Create a GridSearchCV object with 10-fold cross-validation
knn_cv = GridSearchCV(KNN, parameters, cv=10)

# Fit the model to the training data
knn_cv.fit(X_train, Y_train)
```

```
print("tuned hpyerparameters :(best parameters) ",knn_cv.best_params_)
print("accuracy :",knn_cv.best_score_)
```

```
tuned hpyerparameters :(best parameters)  {'algorithm': 'auto', 'n_neighbors': 10, 'p': 1}
accuracy : 0.8482142857142858
```



MLMS: SUPPORT VECTOR MACHINES (SVM)

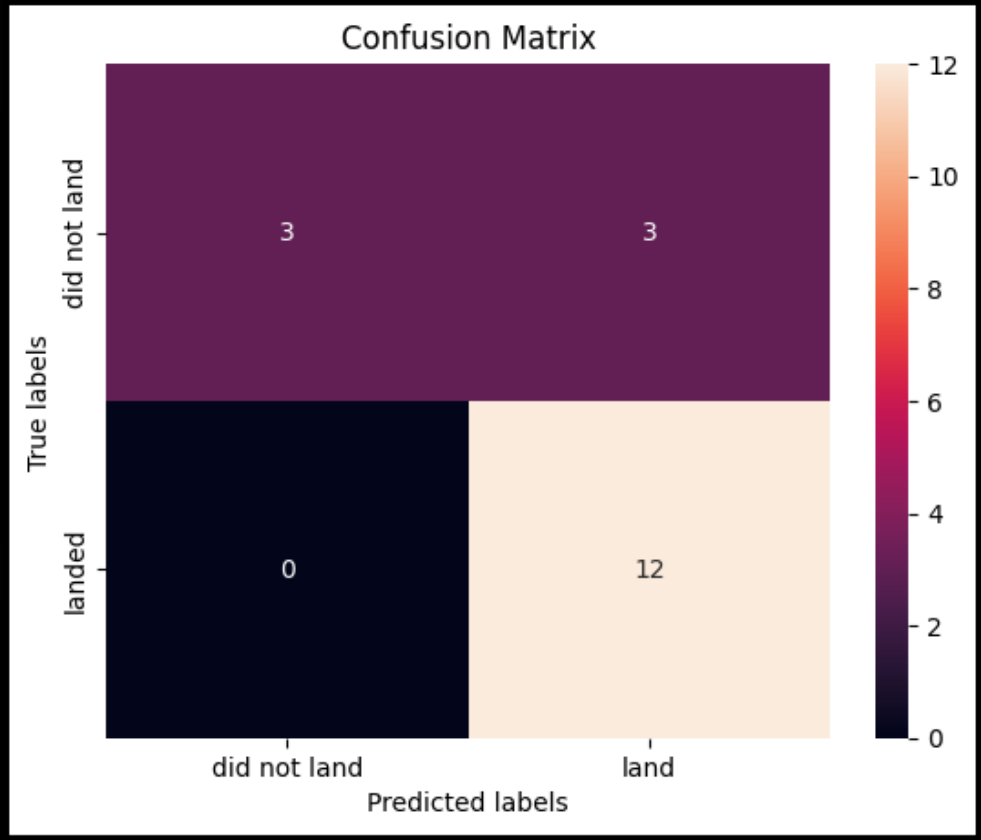
- **Models & Techniques**
  - Support Vector Machines (SVM)

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

```
# Create GridSearchCV object with cv = 10
svm_cv = GridSearchCV(svm, parameters, cv=10)

# Fit the model
svm_cv.fit(X_train, Y_train)
```

```
print("tuned hpyerparameters :(best parameters) ",svm_cv.best_params_)
print("accuracy :",svm_cv.best_score_)
```

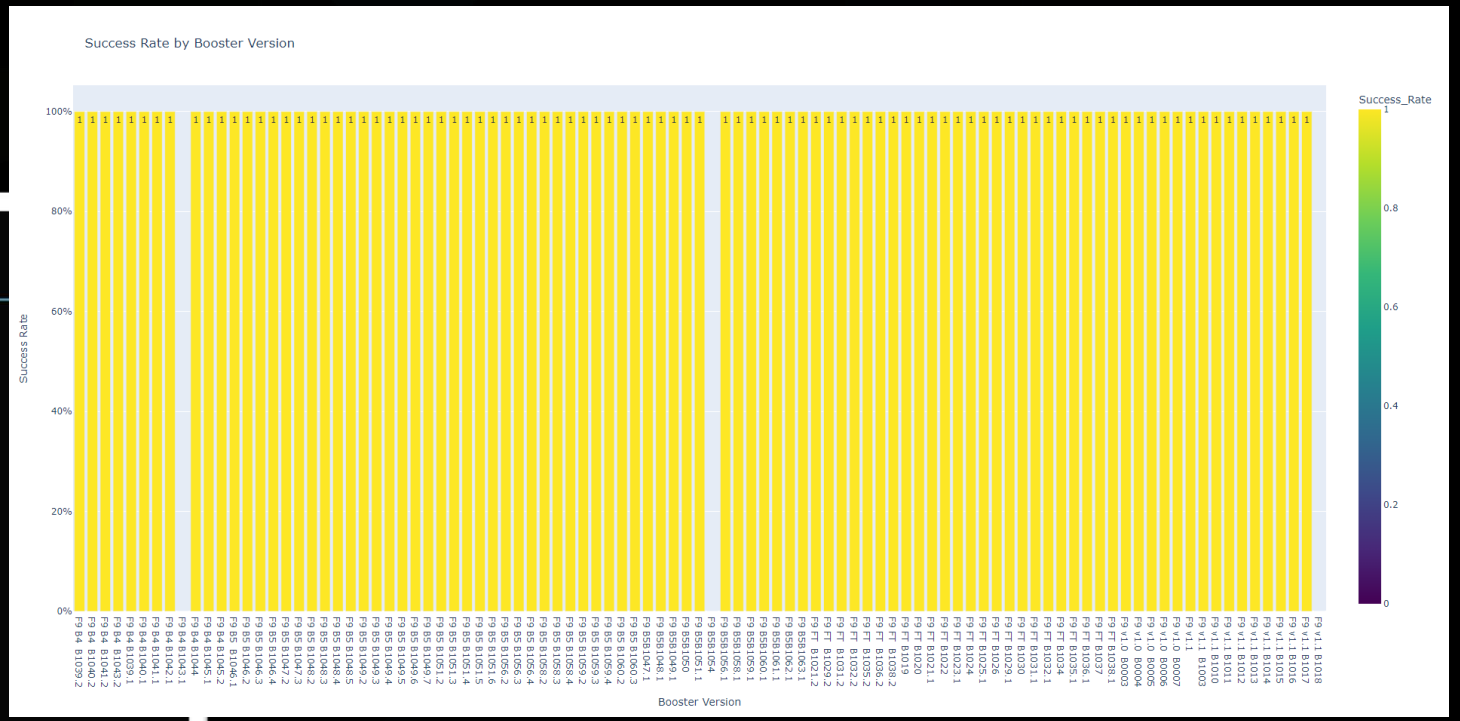
```
tuned hpyerparameters :(best parameters)  {'C': 1.0, 'gamma': 0.03162277660168379, 'kernel': 'sigmoid'}
accuracy : 0.8482142857142856
```



MODEL PERFORMANCE COMPARISON

```
Model Comparison:
                     Training    Test
Logistic Regression    0.8464  0.8333
SVM                    0.8482  0.8333
Decision Tree          0.8893  0.8889
KNN                    0.8482  0.8333

Best Performing Model:
Decision Tree with test accuracy: 0.8889
```

- Summary of model accuracy and performance.



Find the method performs best:

```python
# Create a comparison table
models = {
    'Logistic Regression': {'Training': 0.8464, 'Test': 0.8333},
    'SVM': {'Training': 0.8482, 'Test': 0.8333},
    'Decision Tree': {'Training': 0.8893, 'Test': 0.8889},
    'KNN': {'Training': 0.8482, 'Test': 0.8333}
}

# Create a DataFrame for better visualization
import pandas as pd
comparison_df = pd.DataFrame(models).T
print("Model Comparison:")
print(comparison_df)
print("\nBest Performing Model:")
best_model = comparison_df['Test'].idxmax()
print(f"{best_model} with test accuracy: {comparison_df.loc[best_model, 'Test']}")
```

SKILLS NETWORK

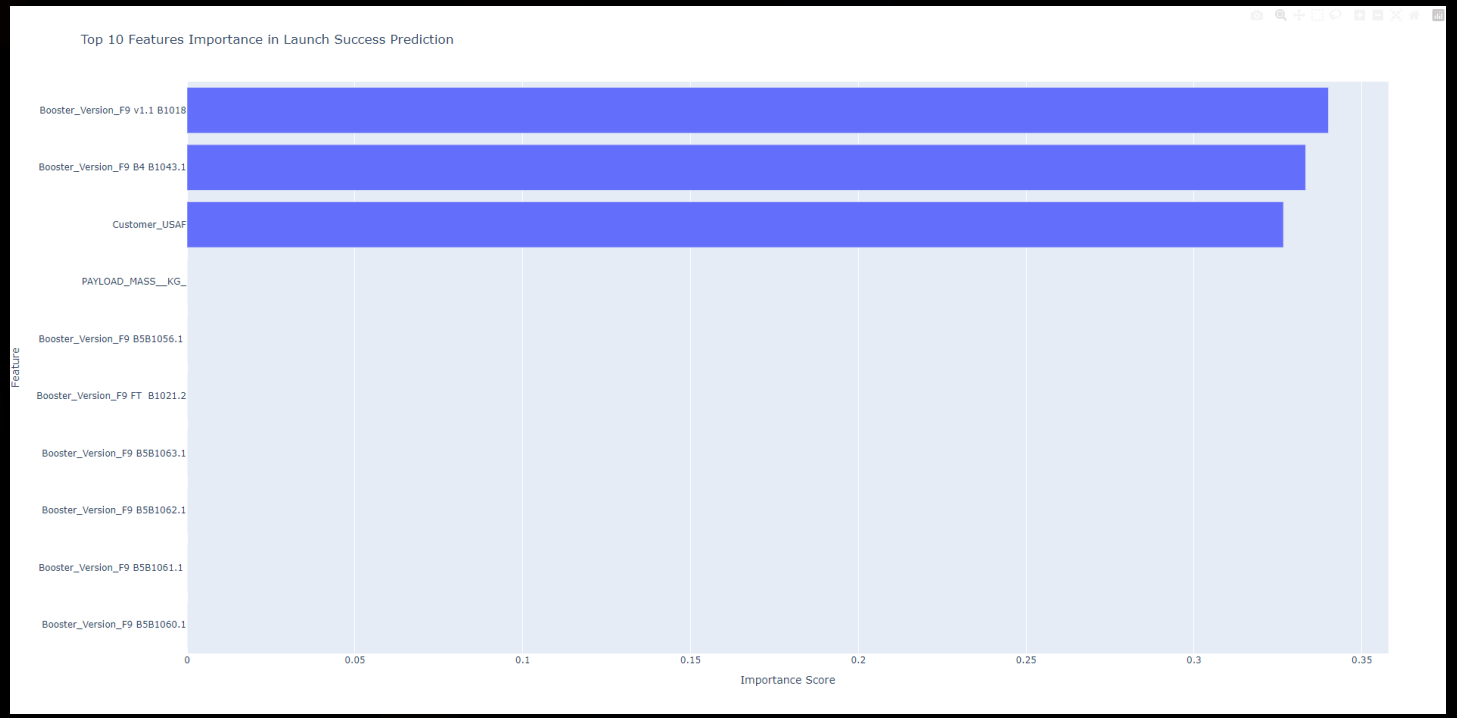
MODEL PERFORMANCE

Best Performing Model: Decision Tree

- Highest training accuracy: 88.93%
- Highest test accuracy: 88.89%
- Most consistent performance between training and test sets
- Outperformed other models by ~5.56 percentage points

Other Models Performance

- Logistic Regression, SVM, and KNN showed identical test accuracy (83.33%)
- Similar training accuracies (84.64% - 84.82%)
- Slight gap between training and test performance indicating minor overfitting



Confusion Matrix Analysis

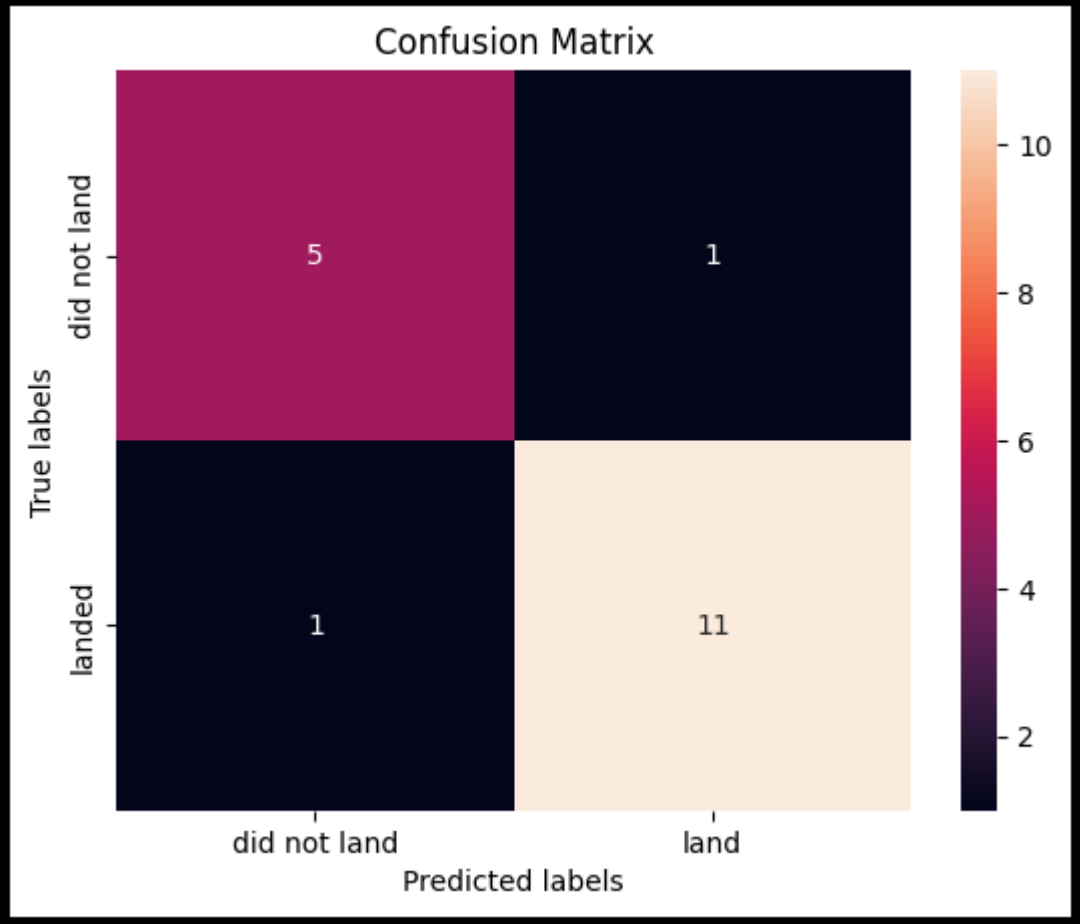
- True Positives: 12 (correctly predicted successful landings)
- False Positives: 3 (incorrectly predicted successful landings)
- Balanced prediction capabilities

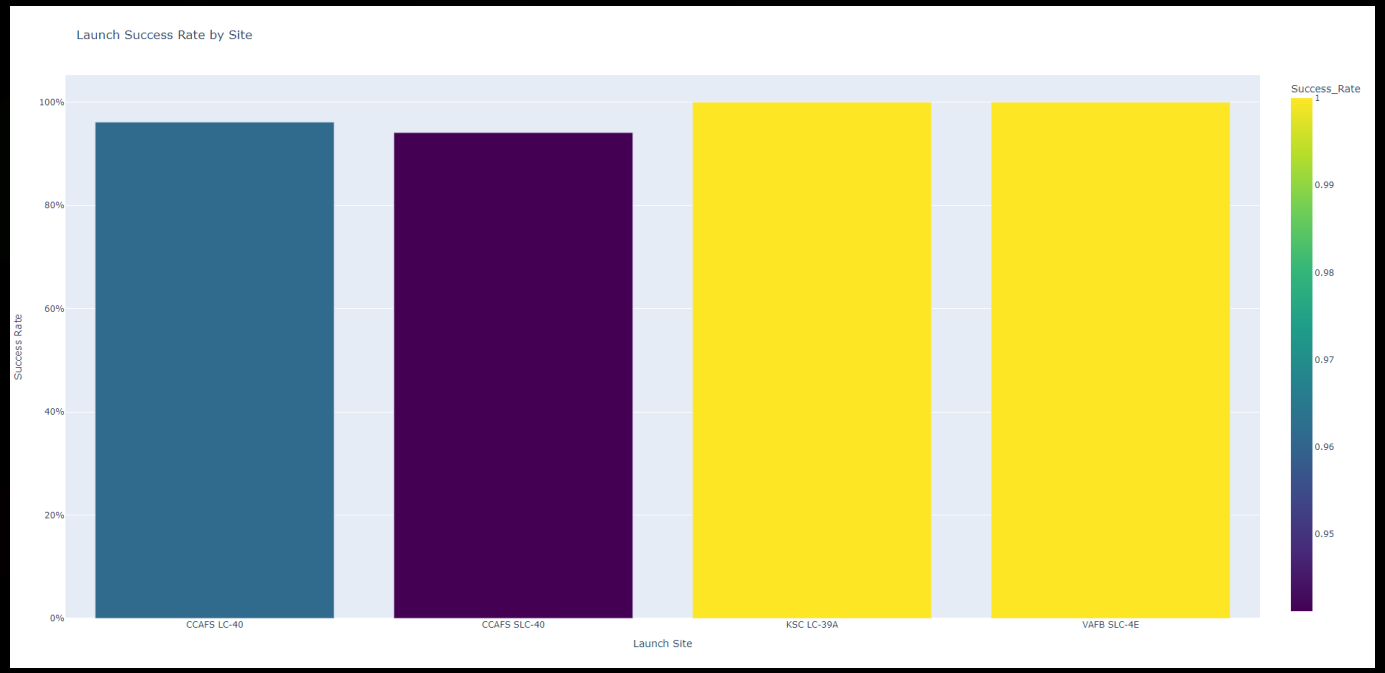
IBM Developer

SKILLS NETWORK

MODEL PERFORMANCE

Best Performing Model: Decision Tree






```
# Calculate accuracy on test data (Task 9)
test_accuracy = tree_cv.score(X_test, Y_test)
print("\nTest set accuracy:", test_accuracy)

Test set accuracy: 0.8888888888888888
```

```
tuned hpyerparameters :(best parameters)  {'criterion': 'entropy'}
accuracy : 0.8892857142857142
```

KEY INSIGHTS

Model Selection

- Decision Tree is clearly the optimal choice for this specific problem
- Suggests non-linear relationships in the landing success factors
- Better at capturing complex patterns in the data

Data Characteristics

- Small test set (18 samples) might limit model evaluation
- Class imbalance might be affecting model performance
- Features show hierarchical importance patterns (favoring Decision Tree)

Prediction Reliability

- Models are better at predicting successful landings
- Main challenge is avoiding false positive predictions
- Decision Tree shows more balanced prediction capabilities

MODEL RECOMMENDATIONS

Implementation

- Use Decision Tree as the primary prediction model
- Consider ensemble methods for further improvement
- Implement confidence thresholds for critical predictions

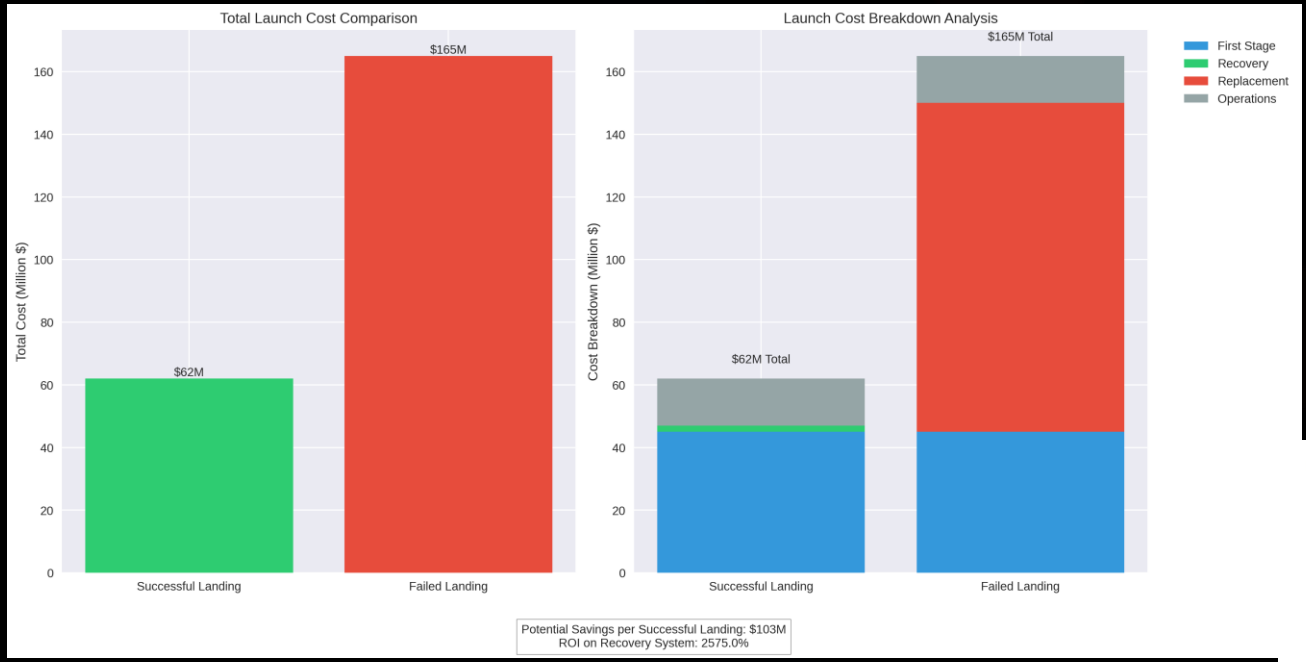
Future Improvements

- Increase test set size for more reliable evaluation (target: 100 samples)
- Feature engineering to help other models perform better (e.g., include weather conditions)
- Consider Random Forest or gradient-boosting alternatives

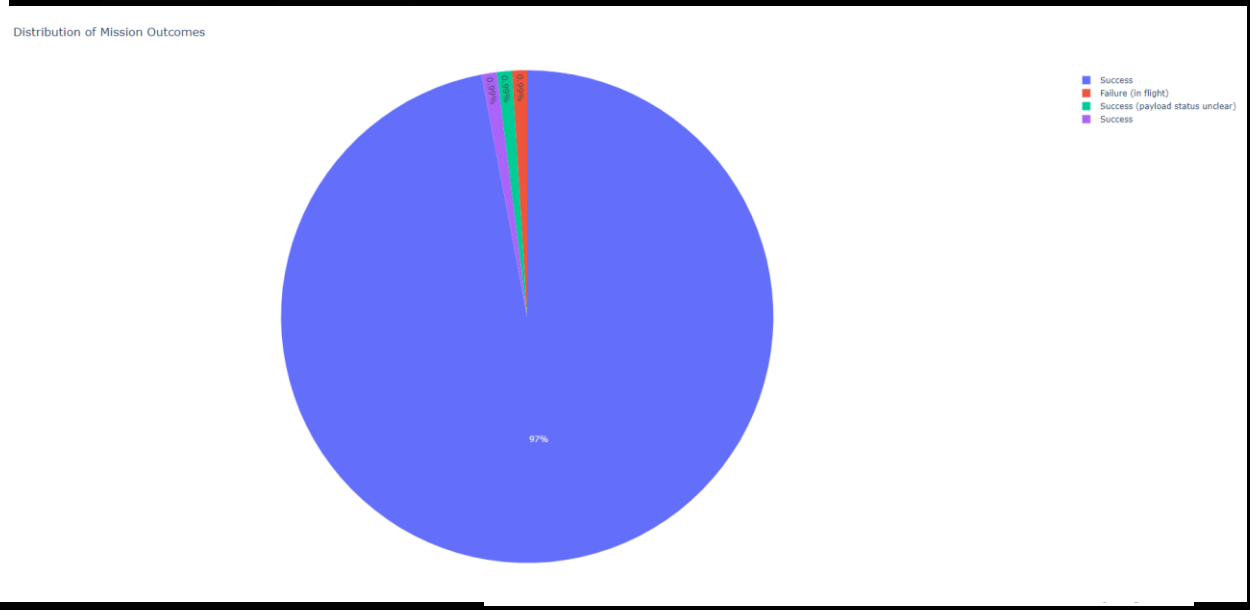
Monitoring

- Track false positive rate in production
- Regular model retraining with new data
- Validate performance across different landing conditions

RESULTS AND BUSINESS IMPLICATIONS

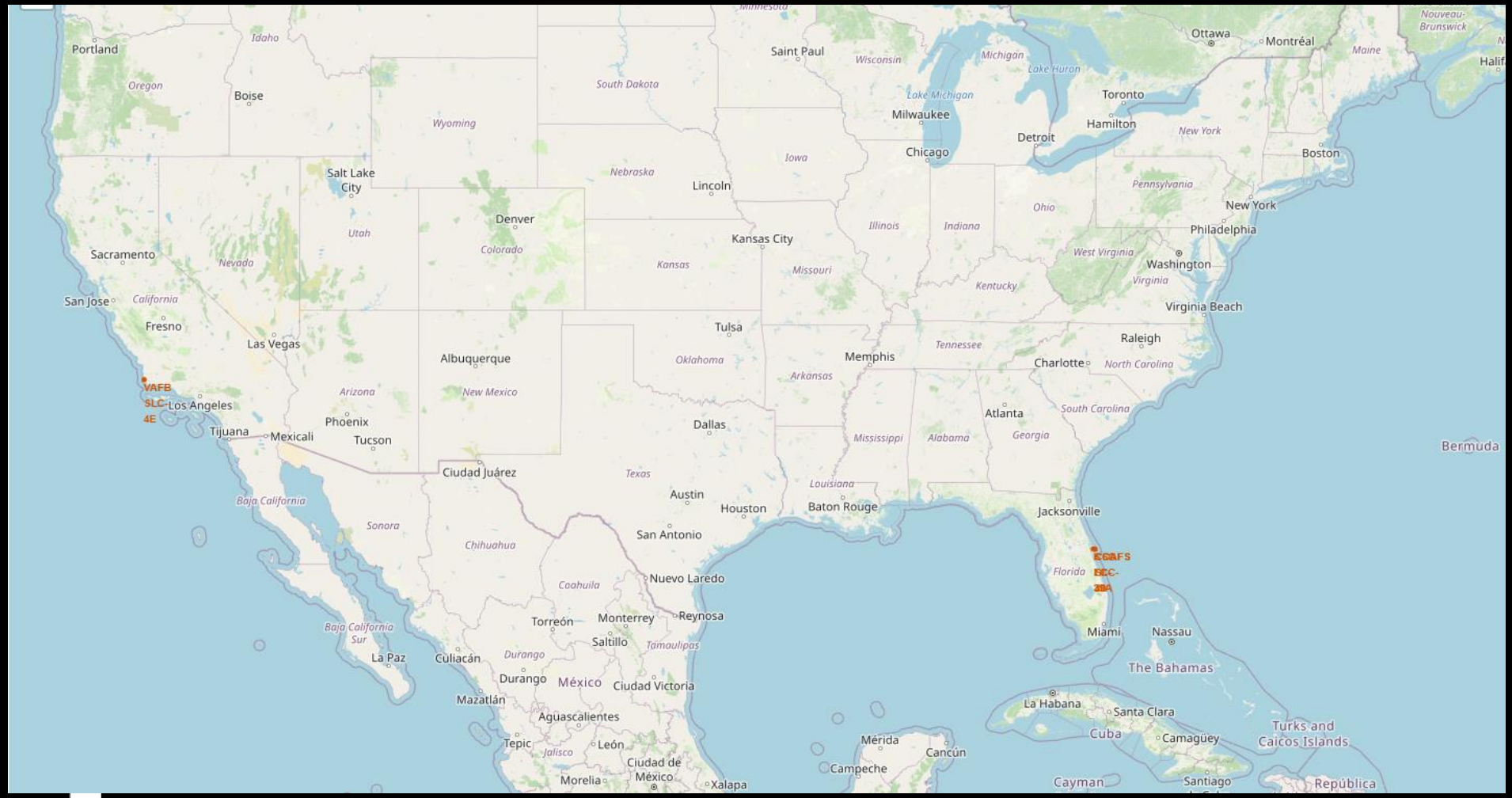


Improving landing predictions by 5% could save millions of dollars.



IBM Developer

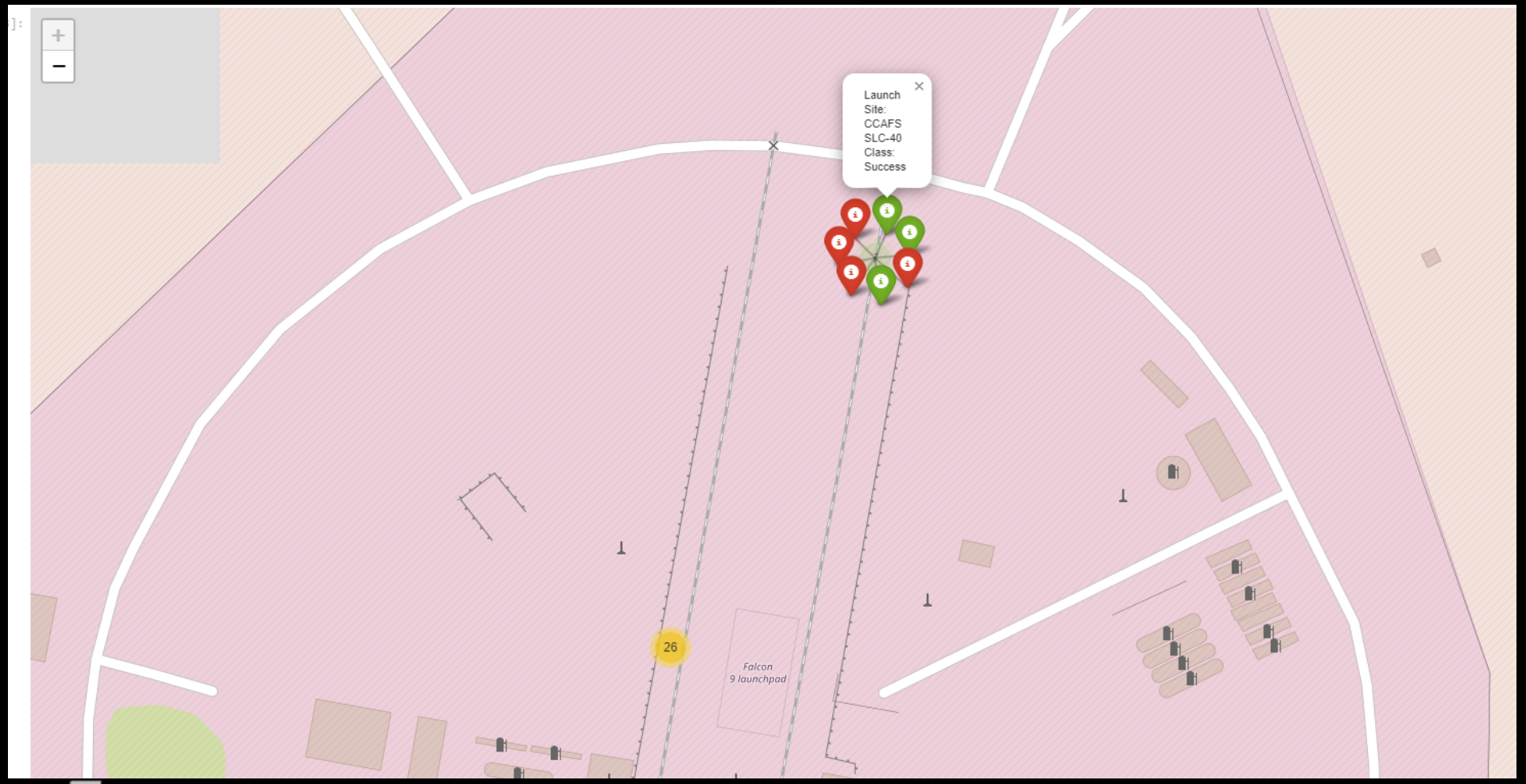
INTERACTIVE MAP (FOLIUM VISUALIZATION)



IBM Developer

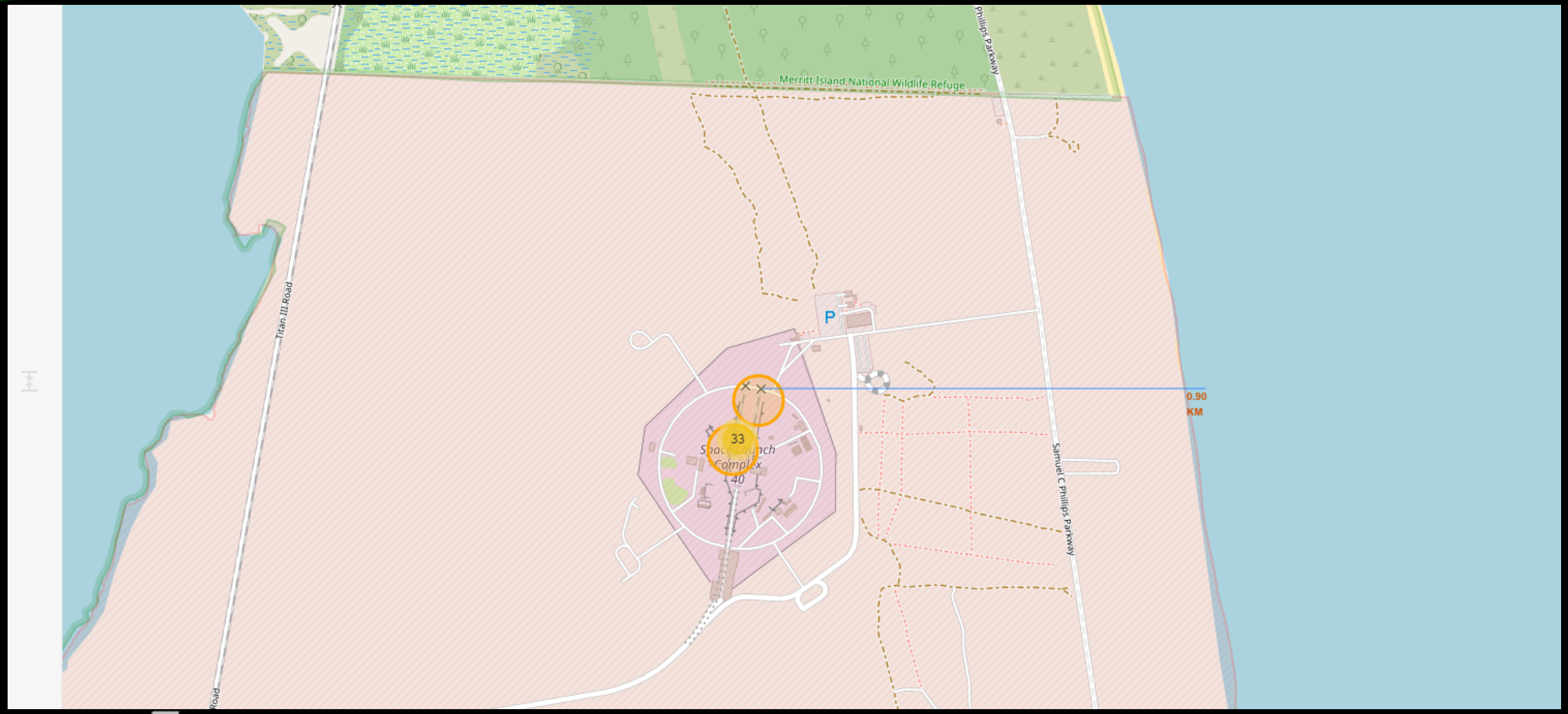
SKILLS NETWORK

INTERACTIVE MAP (FOLIUM VISUALIZATION)

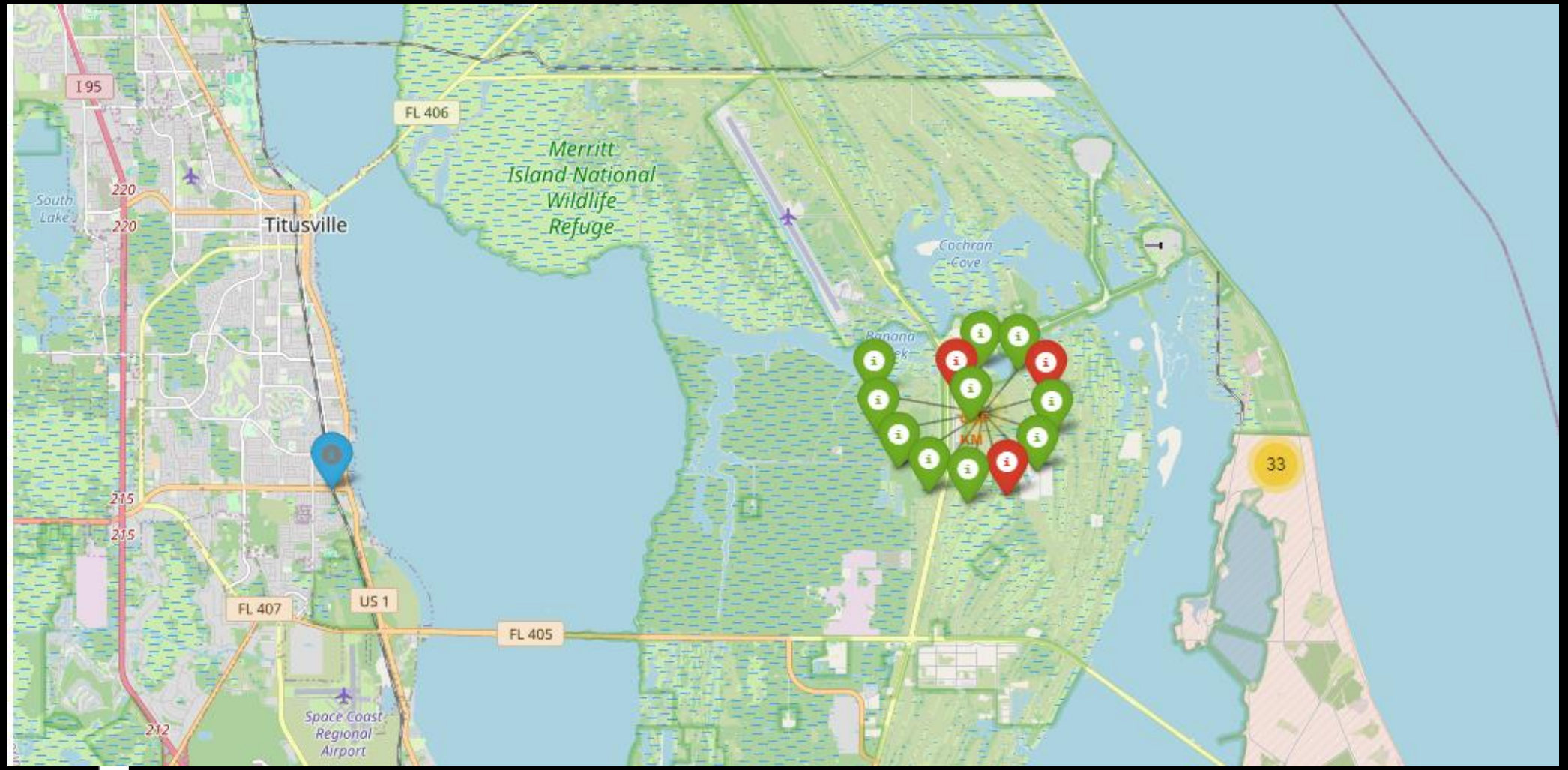


INTERACTIVE MAP (FOLIUM VISUALIZATION)

INTERACTIVE MAP (FOLIUM VISUALIZATION)



INTERACTIVE MAP (FOLIUM VISUALIZATION)



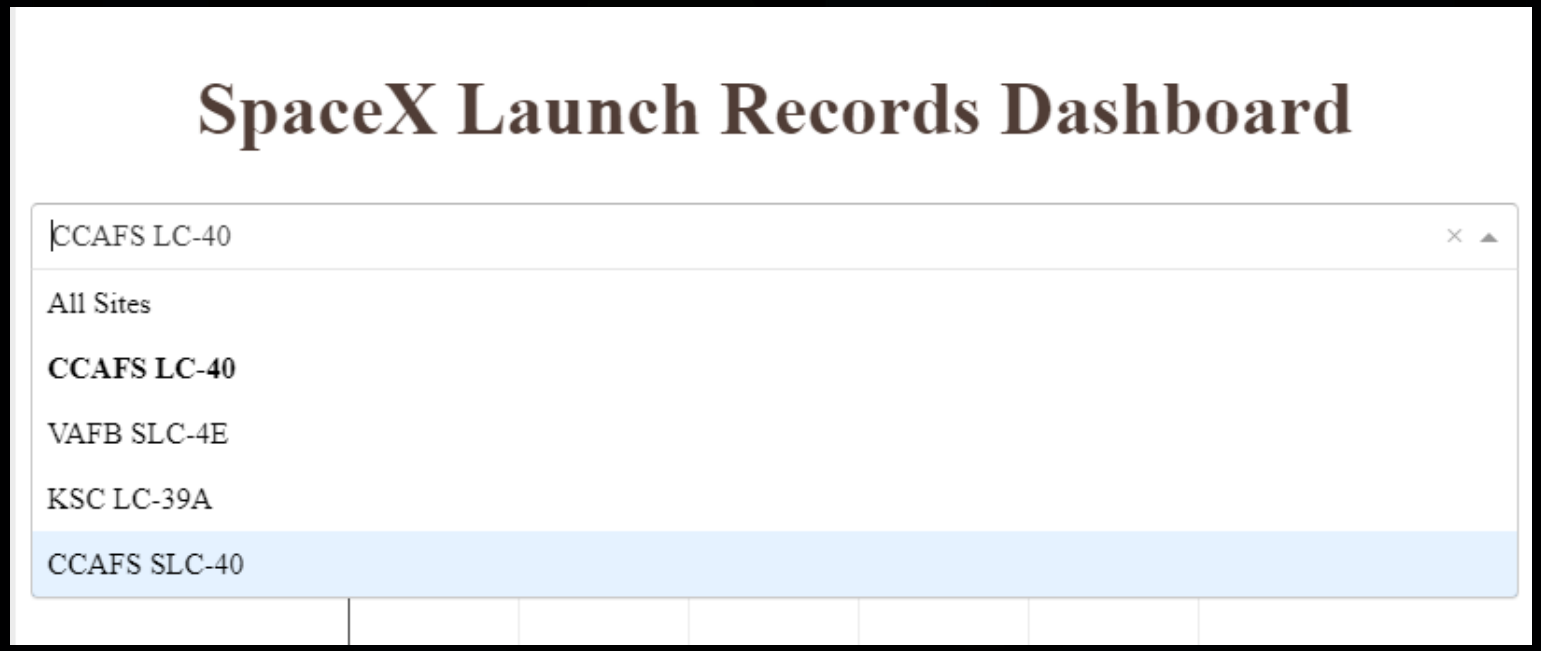
INTERACTIVE DASHBOARD (PLOTLY DASH)

The interactive dashboard allows users to:

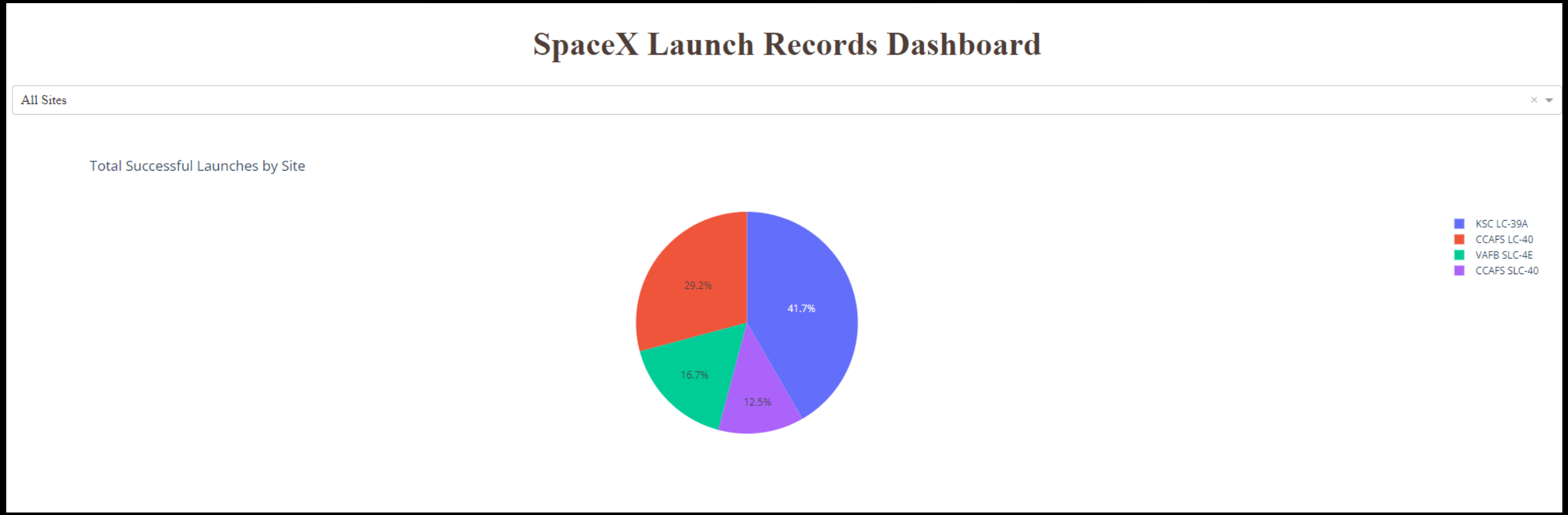
- Visualize launch success rates over time
- Filter data by launch site and payload mass
- Access detailed mission outcomes and predictions

https://github.com/BRobertsonRed/Applied-Data-Science-Capstone/blob/main/Notebooks/spacex_dash_app.py

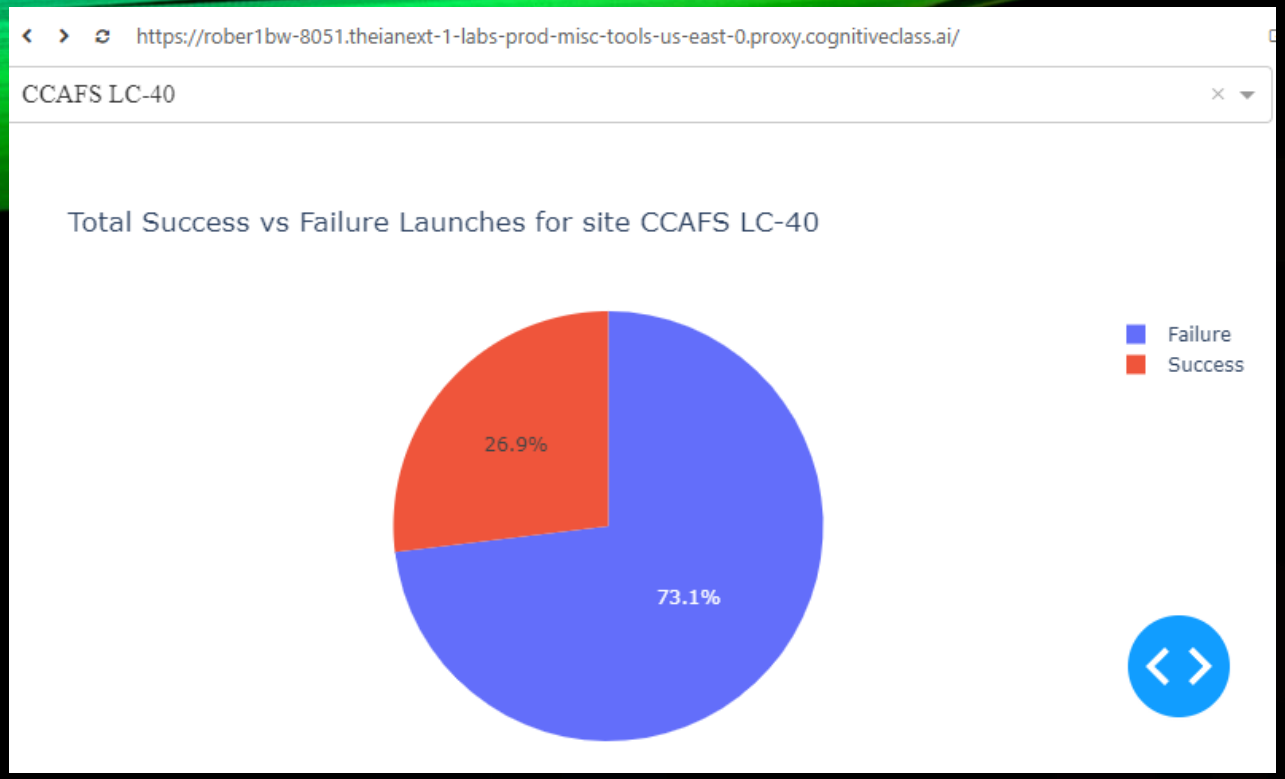
INTERACTIVE DASHBOARD (PLOTLY DASH)

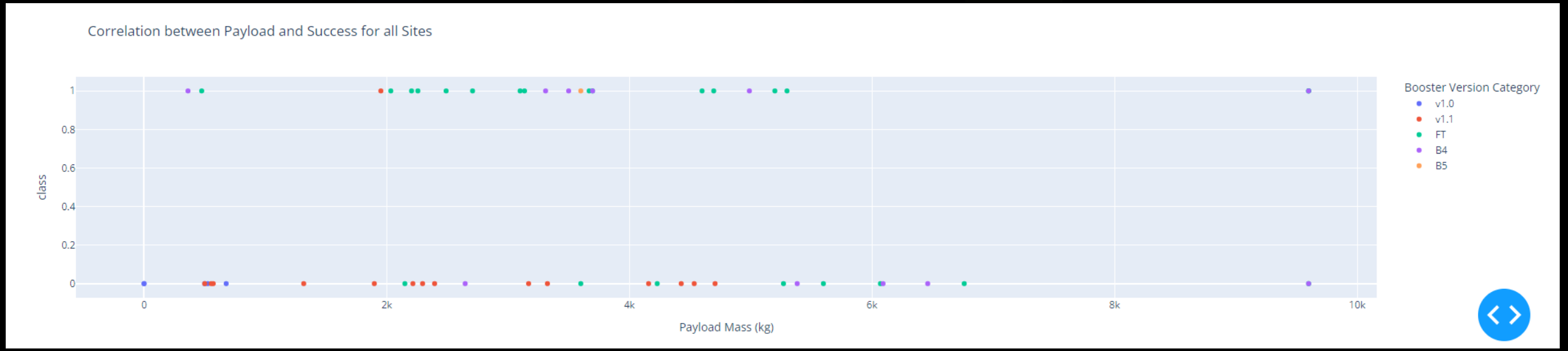


DASHBOARD TAB 2



DASHBOARD TAB 3





BUSINESS IMPLICATIONS OF LANDING PREDICTIONS

Cost Impact Analysis

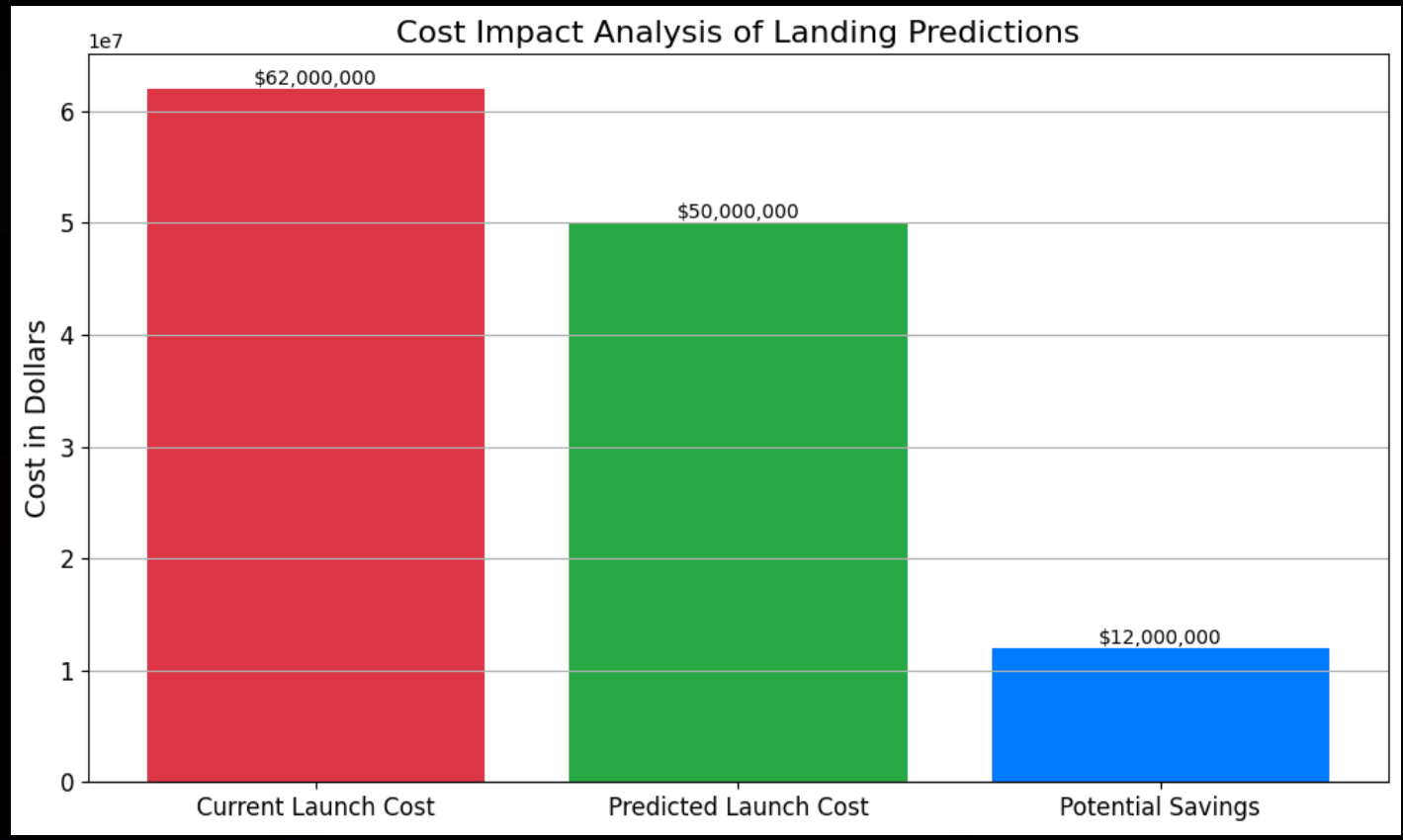
- Current launch cost: \$62M
- Decision Tree model accuracy: 88.89%
- Potential savings from successful landings

Competitive Advantage

- More accurate prediction than other models (5.56% improvement)
- Data-driven launch planning
- Risk mitigation capabilities

Operational Benefits

- Improved launch success rate
- Optimized launch site selection
- Better resource allocation



DISCUSSION AND KEY INSIGHTS

Model Performance

- Decision Tree outperformed other models
- 88.89% accuracy in landing predictions
- Strong performance in identifying successful landings

Critical Success Factors

- Launch site importance
- Payload mass optimization
- Orbit type considerations

Implementation Strategy

- Model deployment recommendations
- Continuous monitoring plan
- Regular model updates

CONCLUSION

Project Achievements

- Successfully developed predictive model
- Achieved 88.89% accuracy with Decision Tree
- Identified key success factors

Business Value

- Enhanced decision-making capability
- Cost optimization potential
- Competitive positioning

Path Forward

- Continue model refinement
- Implement real-time predictions
- Expand feature engineering
- Conduct a pilot program to test model in real-world scenarios

APPENDIX (SUPPORTING DATA & GRAPHS)

README

Space Y Rocket Reusability Prediction Capstone Project 🚀

This project, part of the IBM Data Science Professional Certificate Capstone, aims to predict the success of first-stage rocket landings—critical for reducing launch costs for space companies like SpaceX. The project includes data collection, cleaning, exploratory data analysis (EDA), machine learning, and interactive visualizations. By analyzing past rocket launches, this project helps Space Y, a SpaceX competitor, optimize rocket reusability and reduce space travel costs.

Project Overview

Background

SpaceX has revolutionized space travel by reusing rockets and cutting launch costs. This project simulates Space Y's efforts to achieve the same. We focus on predicting whether Space Y's first-stage rockets will successfully land, using machine learning and historical data.

Problem Statement

Rocket launches are expensive, with the first stage making up 70% of the cost. Reusing it can greatly reduce expenses. The goal of this project is to predict successful first-stage landings, helping Space Y cut costs and remain competitive.

Data Collection

- **SpaceX API:** Historical rocket launch data (payload, orbit, launch site, etc.).
- **Web Scraping:** Supplementary data from Wikipedia.
- **CSV Files:** Pre-existing datasets for further analysis.

Data Wrangling

- Handling missing values.

GITHUB LINK

- https://github.com/BRobertsonRed/Applied-Data-Science-Capstone

SKILLS NETWORK

SQL

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

```
%sql SELECT DISTINCT "Launch_Site" FROM SPACEXTABLE;
```

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

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

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

```
%sql SELECT * FROM SPACEXTABLE WHERE "Launch_Site" LIKE 'CCA%' LI
```

Python

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

Date	Time (UTC)	Booster_Version	Launch_Site	Payload	PAYLOAD_MASS__KG_	Orbit	Customer	Mission_Outcome	Landing_Outcome
2010-06-04	18:45:00	F9 v1.0 B0003	CCAFS LC-40	Dragon Spacecraft Qualification Unit	0	LEO	SpaceX	Success	Failure (parachute)
2010-12-08	15:43:00	F9 v1.0 B0004	CCAFS LC-40	Dragon demo flight C1, two CubeSats, barrel of Brouere cheese	0	LEO (ISS)	NASA (COTS) NRO	Success	Failure (parachute)
2012-05-22	7:44:00	F9 v1.0 B0005	CCAFS LC-40	Dragon demo flight C2	525	LEO (ISS)	NASA (COTS)	Success	No attempt
2012-10-08	0:35:00	F9 v1.0 B0006	CCAFS LC-40	SpaceX CRS-1	500	LEO (ISS)	NASA (CRS)	Success	No attempt
2013-03-01	15:10:00	F9 v1.0 B0007	CCAFS LC-40	SpaceX CRS-2	677	LEO (ISS)	NASA (CRS)	Success	No attempt

Task 3

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

```
%sql SELECT SUM("PAYLOAD_MASS__KG_") AS "Total_Payload_Mass_KG" FROM SPACEXTABLE WHERE "Customer" = 'NASA (CRS)';
```

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

Total_Payload_Mass_KG
45596

IBM

SQL

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

```
%sql SELECT AVG("PAYLOAD_MASS__KG_") AS "Average_Payload_Mass_KG" FROM SPACEXTABLE WHERE "Booster_Version" LIKE 'F9 v1.1%';
```

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

Average_Payload_Mass_KG
2534.6666666666665

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

Hint:Use min function

```
%sql SELECT MIN(Date) AS "First_Successful_Ground_Pad_Landing" FROM SPACEXTABLE WHERE "Outcome" = 'True RTLS';
```

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

First_Successful_Ground_Pad_Landing
None

SQL

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

```
%%sql
SELECT DISTINCT "Booster_Version"
FROM SPACEXTABLE
WHERE "Landing_Outcome" = 'Success (drone ship)'
  AND "PAYLOAD_MASS__KG_" > 4000
  AND "PAYLOAD_MASS__KG_" < 6000;
```

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

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

List the total number of successful and failure mission outcomes

```
%%sql
SELECT DISTINCT "Booster_Version"
FROM SPACEXTABLE
WHERE "Landing_Outcome" = 'Success (drone ship)'
  AND "PAYLOAD_MASS__KG_" > 4000
  AND "PAYLOAD_MASS__KG_" < 6000;
```

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

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

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

```
%%sql
SELECT "Booster_Version"
FROM SPACEXTABLE
WHERE "PAYLOAD_MASS__KG_" = (
    SELECT MAX("PAYLOAD_MASS__KG_")
    FROM SPACEXTABLE
);
```

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

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

SQL

List the records which will display the month names, failure landing_outcomes in drone ship ,booster versions, launch_site for the months in year 2015.

Note: SQLLite does not support monthnames. So you need to use substr(Date, 6,2) as month to get the months and substr(Date,0,5)='2015' for year.

```
%%sql
SELECT
    substr(Date, 6, 2) AS Month,
    "Landing_Outcome",
    "Booster_Version",
    "Launch_Site"
FROM
    SPACEXTABLE
WHERE
    "Landing_Outcome" = 'Failure (drone ship)'
    AND substr(Date, 1, 4) = '2015';
```

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

Month	Landing_Outcome	Booster_Version	Launch_Site
01	Failure (drone ship)	F9 v1.1 B1012	CCAFS LC-40
04	Failure (drone ship)	F9 v1.1 B1015	CCAFS LC-40

```
%%sql
SELECT Date, "Landing_Outcome"
FROM SPACEXTABLE
LIMIT 10;
```

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

Date	Landing_Outcome
2010-06-04	Failure (parachute)
2010-12-08	Failure (parachute)
2012-05-22	No attempt
2012-10-08	No attempt
2013-03-01	No attempt
2013-09-29	Uncontrolled (ocean)
2013-12-03	No attempt
2014-01-06	No attempt
2014-04-18	Controlled (ocean)
2014-07-14	Controlled (ocean)

SQL

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.

```
%%sql
SELECT
    "Landing_Outcome",
    COUNT(*) AS Outcome_Count
FROM
    SPACEXTABLE
WHERE
    Date >= '2010-06-04'
    AND Date <= '2017-03-20'
GROUP BY
    "Landing_Outcome"
ORDER BY
    Outcome_Count DESC;
```

9]

Pythc

* sqlite:///my_data1.db
Done.

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

DASHBOARD CODE

```
                '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
),
html.Br(),
```

```
    # TASK 3: Add a slider to select payload range
    dcc.RangeSlider(
        id='payload-slider',
        min=0,
        max=10000,
        step=1000,
        marks={i: f'{i}' for i in range(0, 10001, 2500)},
        value=[min_payload, max_payload]
    ),
    html.Br(),
```

```
# 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):
    if entered_site == 'ALL':
        # Compute total successful launches for all sites
        filtered_df = spacex_df[spacex_df['class'] == 1]
        fig = px.pie(
            data_frame=filtered_df,
            names='Launch Site',
            title='Total Successful Launches by Site'
        )
        return fig
    else:
        # Filter the dataframe for the selected site
        filtered_df = spacex_df[spacex_df['Launch Site'] == entered_site]
        # Compute the success and failure counts
        success_counts = filtered_df['class'].value_counts().reset_index()
        success_counts.columns = ['class', 'count']
        success_counts['class'] = success_counts['class'].map({1: 'Success', 0: 'Fail
        fig = px.pie(
            data_frame=success_counts,
            names='class',
            values='count',
            title=f'Total Success vs Failure Launches for site {entered_site}'
        )
        return fig
```

DASHBOARD CODE

```
# TASK 4:
# Add a callback function for `site-dropdown` and `payload-slider` as inputs, `succes
@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_scatter_chart(entered_site, payload_range):
    low, high = payload_range
    # Filter the data based on payload range
    mask = (spacex_df['Payload Mass (kg)'] >= low) & (spacex_df['Payload Mass (kg)']
    filtered_df = spacex_df[mask]

    if entered_site == 'ALL':
        # Scatter plot for all sites
        fig = px.scatter(
            filtered_df,
            x='Payload Mass (kg)',
            y='class',
            color='Booster Version Category',
            title='Correlation between Payload and Success for all Sites',
            hover_data=['Launch Site']
        )
    else:
        # Filter the data further based on the selected site
        filtered_df = filtered_df[filtered_df['Launch Site'] == entered_site]
        fig = px.scatter(
            filtered_df,
            x='Payload Mass (kg)',
            y='class',
            color='Booster Version Category',
            title=f'Correlation between Payload and Success for site {entered_site}',
            hover_data=['Launch Site']
        )
    return fig
```

INTERACTIVE MAP (FOLIUM VISUALIZATION)

```
# 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


# Map the class column to predefined folium icon colors
spacex_df['marker_color'] = spacex_df['class'].apply(lambda x: 'green' if x == 1 else 'red')

# Initialize a MarkerCluster object
marker_cluster = MarkerCluster()

# Create a folium map centered at NASA's coordinate
site_map = folium.Map(location=nasa_coordinate, zoom_start=5)

# Add the marker cluster to the map
marker_cluster = MarkerCluster().add_to(site_map)

# For each launch record, add a marker to the MarkerCluster
for index, row in spacex_df.iterrows():
    folium.Marker(
        location=[row['Lat'], row['Long']],
        icon=folium.Icon(color=row['marker_color']),  # Now using 'green' and 'red'
        popup=f"Launch Site: {row['Launch Site']}\nClass: {'Success' if row['class'] == 1 else 'Failure'}"
    ).add_to(marker_cluster)

# Display the map
site_map
```

```
import folium
from folium.plugins import MarkerCluster

# Create a folium map centered at NASA's coordinate
site_map = folium.Map(location=nasa_coordinate, zoom_start=5)

# Initialize a MarkerCluster object for individual launches
marker_cluster = MarkerCluster()

# Add the marker cluster to the map
marker_cluster.add_to(site_map)

# Add the orange circle overlay for each launch site with a smaller radius
for index, row in launch_sites_df.iterrows():
    # Create a smaller orange circle with a radius of 50 meters
    folium.Circle(
        location=[row['Lat'], row['Long']],
        radius=50,  # Set radius to 50 meters for better appearance when zoomed in
        color='#FFA500',  # Orange color for the circle
        fill=True,
        fill_color='#FFA500',
        fill_opacity=0.2,  # Transparent enough for better visibility
        popup=f"Launch Site: {row['Launch Site']}"  # Add the popup for launch site name
    ).add_to(site_map)

# For each launch record, add a marker to the MarkerCluster for individual launch events
for index, row in spacex_df.iterrows():
    folium.Marker(
        location=[row['Lat'], row['Long']],
        icon=folium.Icon(color=row['marker_color'], icon='info-sign'),
        popup=f"Launch Site: {row['Launch Site']}\nOutcome: {'Success' if row['class'] == 1 else 'Failure'}"
    ).add_to(marker_cluster)

# Display the map
site_map
```

IBM Developer

INTERACTIVE MAP (FOLIUM VISUALIZATION)

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

# Function to calculate haversine distance between two coordinates
def haversine_distance(lat1, lon1, lat2, lon2):
    R = 6371.0  # Radius of Earth in kilometers

    # Convert latitude and longitude from degrees to radians
    lat1 = math.radians(lat1)
    lon1 = math.radians(lon1)
    lat2 = math.radians(lat2)
    lon2 = math.radians(lon2)

    # Differences in coordinates
    dlat = lat2 - lat1
    dlon = lon2 - lon1

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

    # Distance in kilometers
    distance = R * c
    return distance

# Coordinates of the launch site (example: CCAFS SLC-40)
launch_site_lat = 28.562302
launch_site_lon = -80.577356

# List of target locations (example: nearby cities or landmarks)
targets = [
    {"name": "Kennedy Space Center Visitor Complex", "lat": 28.5721, "lon": -80.6480},
    {"name": "Orlando", "lat": 28.538336, "lon": -81.379234},
    {"name": "Miami", "lat": 25.761680, "lon": -80.191790}
]

# Calculate the distance from the launch site to each target
for target in targets:
    distance = haversine_distance(launch_site_lat, launch_site_lon, target["lat"], target["lon"])
    print(f"Distance from {target['name']} to the launch site: {distance:.2f} km")
```

```python
# Initialize a MarkerCluster object for individual launches
marker_cluster = MarkerCluster()

# Add the marker cluster to the map
marker_cluster.add_to(site_map)

# Add the orange circle overlay for each launch site with a smaller radius
for index, row in launch_sites_df.iterrows():
    # Create a smaller orange circle with a radius of 50 meters
    folium.Circle(
        location=[row['Lat'], row['Long']],
        radius=50,  # Set radius to 50 meters for better appearance when zoomed in
        color='#FFA500',  # Orange color for the circle
        fill=True,
        fill_color='#FFA500',
        fill_opacity=0.2,  # Transparent enough for better visibility
        popup=f"Launch Site: {row['Launch Site']}"  # Add the popup for launch site name
    ).add_to(site_map)

# For each launch record, add a marker to the MarkerCluster for individual launch events
for index, row in spacex_df.iterrows():
    folium.Marker(
        location=[row['Lat'], row['Long']],
        icon=folium.Icon(color=row['marker_color'], icon='info-sign'),
        popup=f"Launch Site: {row['Launch Site']}\nOutcome: {'Success' if row['class'] == 1 else 'Failure'}"
    ).add_to(marker_cluster)

#Work out distance to coastline
coordinates = [
    [28.56342, -80.57674],
    [28.56342, -80.56756]]

lines=folium.PolyLine(locations=coordinates, weight=1)
site_map.add_child(lines)
distance = calculate_distance(coordinates[0][0], coordinates[0][1], coordinates[1][0], coordinates[1][1])
distance_circle = folium.Marker(
    [28.56342, -80.56794],
    icon=DivIcon(
        icon_size=(20,20),
        icon_anchor=(0,0),
        html='<div style="font-size: 12; color:#d35400;"><b>%s</b></div>' % "{:10.2f} KM".format(distance),
```

INTERACTIVE MAP (FOLIUM VISUALIZATION)

```
# Initialize a MarkerCluster object for individual launches
marker_cluster = MarkerCluster()

# Add the marker cluster to the map
marker_cluster.add_to(site_map)

# Add the orange circle overlay for each launch site with a smaller radius
for index, row in launch_sites_df.iterrows():
    # Create a smaller orange circle with a radius of 50 meters
    folium.Circle(
        location=[row['Lat'], row['Long']],
        radius=50,  # Set radius to 50 meters for better appearance when zoomed in
        color='#FFA500',  # Orange color for the circle
        fill=True,
        fill_color='#FFA500',
        fill_opacity=0.2,  # Transparent enough for better visibility
        popup=f"Launch Site: {row['Launch Site']}"  # Add the popup for launch site name
    ).add_to(site_map)

# For each launch record, add a marker to the MarkerCluster for individual launch events
for index, row in spacex_df.iterrows():
    folium.Marker(
        location=[row['Lat'], row['Long']],
        icon=folium.Icon(color=row['marker_color'], icon='info-sign'),
        popup=f"Launch Site: {row['Launch Site']}\nOutcome: {'Success' if row['class'] == 1 else 'Failure'}"
    ).add_to(marker_cluster)
# Create a marker with distance to a closest city, railway, highway, etc.
# Draw a line between the marker to the launch site

distance_railway = calculate_distance(28.55768, -80.80188, 28.573255, -80.646895)
distance_railway

# Display the distance between railway point and launch site using the icon property

site_map.add_child(folium.Marker(location=[28.55768, -80.80188],
                                 popup='15.239',
                                 icon=folium.Icon(color='blue', icon_color='gray')))

site_map.add_child(distance_marker)
site_map
```