WINNING SPACE RACE WITH DATA SCIENCE

A Thesis Submitted
In Partial Fulfilment of the Requirements
for the Degree of

MASTER OF COMPUTER APPLICATION

in

Data Science

by

SACHIN RATHI

(Enrolment No: 190029014005172)

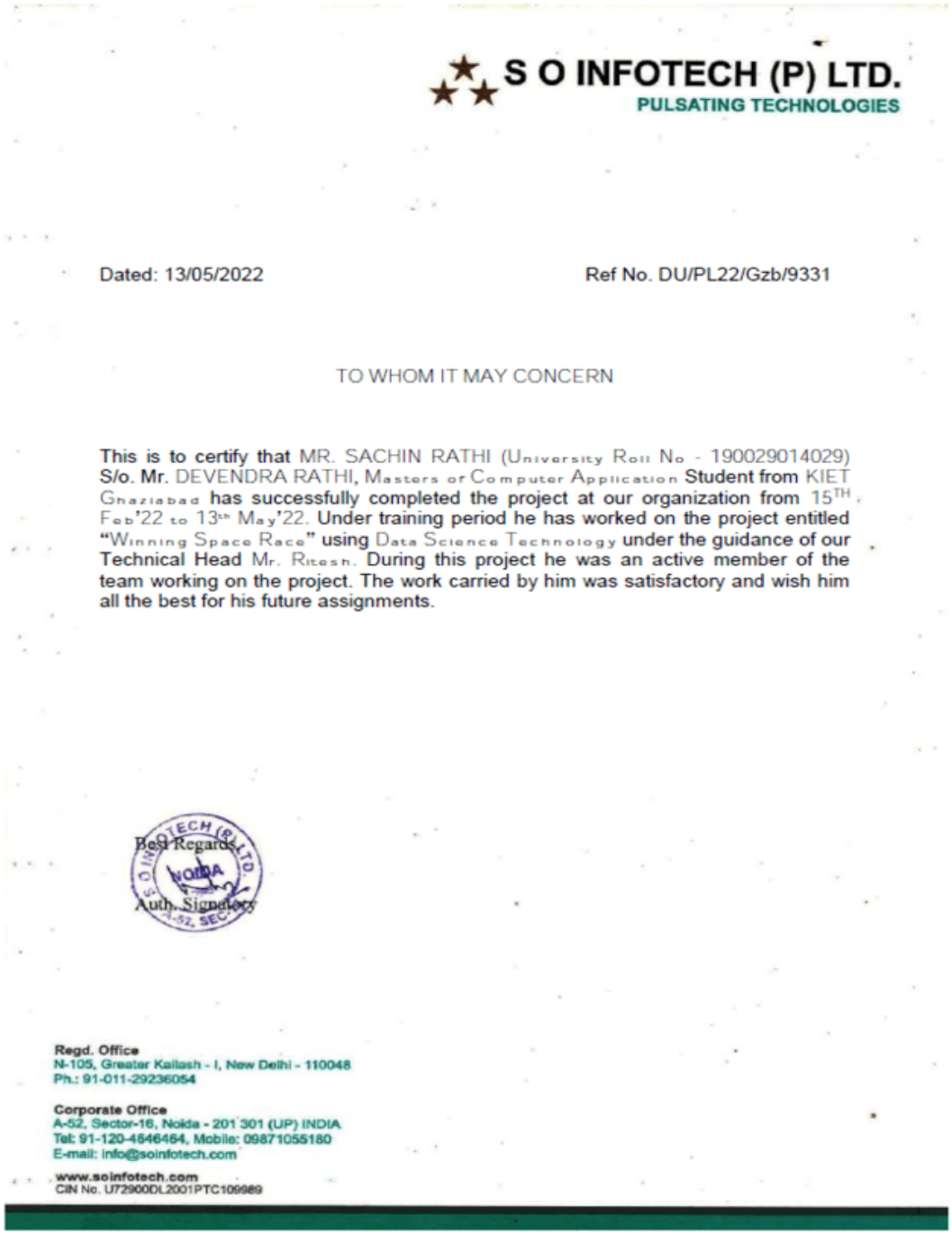
Under the Supervision of
Ms. Vidushi
(ASSISTANT PROFESSOR)
KIET Group of institutions, Delhi-NCR, Ghaziabad

to the

FACULTY OF COMPUTER APPLICATIONS

DR. APJ ABDUL KALAM TECHNICAL UNIVERSITY
LUCKNOW
(Formerly Uttar Pradesh Technical University, Lucknow)
June, 2022

Training Certificate



S O INFOTECH (P) LTD.
PULSATING TECHNOLOGIES

Dated: 13/05/2022 Ref No. DU/PL22/Gzb/9331

TO WHOM IT MAY CONCERN

This is to certify that MR. SACHIN RATHI (University Roll No - 190029014029) S/o. Mr. DEVENDRA RATHI, Masters of Computer Application Student from KIET Ghaziabad has successfully completed the project at our organization from 15TH Feb'22 to 13th May'22. Under training period he has worked on the project entitled "Winning Space Race" using Data Science Technology under the guidance of our Technical Head Mr. Ritesh. During this project he was an active member of the team working on the project. The work carried by him was satisfactory and wish him all the best for his future assignments.

Best Regards
Auth. Signatory

Regd. Office
N-105, Greater Kailash - I, New Delhi - 110048
Ph.: 91-011-29236054

Corporate Office
A-52, Sector-16, Noida - 201 301 (UP) INDIA
Tel: 91-120-4646464, Mobile: 09871055180
E-mail: info@soinfotech.com

www.soinfotech.com
CIN No. U72900DL2001PTC109989

DECLARATION

I hereby declare that the work presented in this report entitled “Winning Space Race with Data Science", was carried out by me. I have not submitted the matter embodied in this report for the award of any other degree or diploma of any other University or Institute. I have given due credit to the original authors/sources for all the words, ideas, diagrams, graphics, computer programs, experiments, results, that are not my original contribution. I have used quotation marks to identify verbatim sentences and given credit to the original authors/sources.

I affirm that no portion of my work is plagiarized, and the experiments and results reported in the report are not manipulated. In the event of a complaint of plagiarism and the manipulation of the experiments and results, I shall be fully responsible and answerable.

Name : Sachin Rathi

Enrol. No.: 190029014005172

Field : MCA 6th Sem.

(Candidate Signature)

Sachin Rathi

CERTIFICATE

Certified that **Sachin Rathi** **(**enrollment no 190029014005172) has carried out the project work presented in this thesis entitled **"Winning Space Race with Data Science"** for the award of **Master of Computer Application** from Dr. A P J Abdul Kalam Technical University, Lucknow under my supervision. The thesis embodies results of original work, and studies are carried out by the student himself and the contents of the thesis do not formthe basis for the award of any other degree to the candidate or to anybody else from this or any other University.

Ms. Vidushi External Examiner

(Internal Examiner)

Assistant Professor

Dept. of Computer Applications

KIET Group of Institutions, Delhi-NCR, Ghaziabad

Date: June, 2022

Winning Space Race with Data Science

Sachin Rathi

ABSTRACT

This project will give you a taste of what data scientists go through in real life when working with real datasets. You will assume the role of a Data Scientist working for a start-up intending to compete with SpaceX, and in the process follow the Data Science methodology involving data collection, data wrangling, exploratory data analysis, data visualization, model development, model evaluation, and reporting your results to stakeholders.

In this project, we will predict if the Falcon 9 first stage will land successfully. SpaceX advertises Falcon 9 rocket launches on its website, with a cost of 62 million dollars; other providers cost upward of 165 million dollars each, much of the savings is because SpaceX can reuse the first stage. Therefore, if we can determine if the first stage will land, we can determine the cost of a launch. This information can be used if an alternate company wants to bid against SpaceX for a rocket launch. In this module, you will be provided with an overview of the problem and the tools you need to complete the course.

Figure 1 Winning Space Race with Data Science

ACKNOWLEDGEMENTS

Success in life is never attained single headedly. My deepest gratitude goes to my thesis supervisor, **Ms. Vidushi for** his guidance, help and encouragement throughout my research work. Their enlightening ideas, comments, and suggestions.

Words are not enough to express my gratitude to **Dr. Ajay Kumar Shrivastava, Professor and Head, Department of Computer Applications**, for his insightful comments and administrative help at various occasions.

Fortunately, I have many understanding friends, who have helped me a lot on many critical conditions.

Finally, my sincere thanks go to my family members and all those who have directly and indirectly provided me moral support and other kind of help. Without their support, completion of this work would not have been possible in time. They keep my life filled with enjoyment and happiness.

Sachin Rathi

1900290140029

MCA 6th Sem.

TABLE OF CONTENT

Table of Contents Page No.

LIST OF TABLES

LIST OF FIGURES

Chapter 1
INTRODUCTION

1.1 DESCIPTION OF THE PROJECT

The commercial space age is here, companies are making space travel affordable for everyone. Virgin Galactic is providing suborbital spaceflights. Rocket Lab is a small satellite provider. Blue Origin manufactures sub-orbital and orbital reusable rockets. Perhaps the most successful is SpaceX. SpaceX's accomplishments include Sending spacecraft to the International Space Station. Starlink, a satellite internet constellation providing satellite Internet access. Sending manned missions to Space. One reason SpaceX can do this is the rocket launches are relatively inexpensive. SpaceX advertises Falcon 9 rocket launches on its website with a cost of 62 million dollars; other providers cost upwards of 165 million dollars each, much of the savings is because SpaceX can reuse the first stage.

Figure 2 Fail.gif

Figure 3 Success.gif

In this capstone, I take the role of a data scientist working for a new rocket company. Space Y that would like to compete with SpaceX founded by Billionaire industrialist Alon Musk. My job is to determine the price of each launch. I must do this by gathering information about Space X and creating dashboards for my team. I also determine if SpaceX will reuse the first stage. Instead of using rocket science to determine if the first stage will land successfully, I will train a machine learning model and use public information to predict if SpaceX will reuse the first stage.

CHAPTER 2
Literature Review

2.1. Abstract:

Since the evolution of internet, the amount of data produced each minute is increasing substantially. To manage this data, there has to be some mechanism. Thousands of terabytes produced each year need to be stored securely and should be accessible easily, this needs high amount of research and development. This new concern has given birth to field known as “data science” and people “data scientist”. Today data scientists are working hard to develop new solutions to process and store large amount of data using new techniques. Another major concern today is efficient use of available energy resources, data science helps immensely to predict the usage of resources and minimizing the wastage. This paper is about data science, current and future development in this field. The primary focus is on literature review of data science and various applications of data science.

2.2. Introduction:

The word “Science” is defined by Oxford as “it is a systematic enterprise that builds and organizes knowledge in the form of testable explanations and predictions about the universe” (Heilbron, 2003). Considering this definition data science can be defined as organizing data knowledge that can be used for experiments and prediction. The need for data science developed due to the immense increase in the amount of raw data such as images, text, video and others. As shown in the figure compared to 2012 the amount of data produced in 2015 is significantly high and in order to organize and use this data efficiently we need data science and data scientist. Source: (Dhar, 2013) Every field is contributing to this ever increasing data such as Engineering, mining, healthcare, hospitality, energy etc. Data scientists are developing various algorithms and techniques in order to process and analyze this data and make the best use out of it. Previously health industry used hard paper to store data regarding patients and other medical issues. But the new trend has helped doctors to store this data in electronic form (Raghupathi & Raghupathi, 2014). Space

exploration has produced a large amount of data considering the recent space missions, data scientists have also helped to store this data and use it for prediction to carry out further missions. Implementing business logics and strategies need data analysis and predictions which can be now easily done by the use of prediction algorithms. Energy companies are using data prediction algorithms to manage the energy production based on demands and supply. This prediction has aided to use the available energy efficiently. In coming years machines will have the ability to predict and generate the required resource as per the supply and demand. The current upcoming technology Artificial Intelligence is also being helped tremendously by the use of data mining and prediction algorithms.

2.3. Data Science:

History and Present As a method of prediction previously many years statistics was used. Until recent years these statistics have been transformed into data analysis. Ten to twenty years ago this data analysis was further urged to develop into a new stream known as data science. As maximum amount of data was stored electronically extracting and analysing it itself became a science. Around fifty years ago Jhon Tucky evolved statistics into the term data analysis (Donoho, 2015). Later around twenty years ago this data analysis was further evolved into new stream known as data science by John Chambers, Bill Cleveland and Leo Breiman (Donoho, 2015). Data Mining has been the oldest form of technique for data science. Various algorithms are being developed to aid this process. The evolution of these algorithms' dates back to 1957. In 2006 IEEE announced top 10 algorithms used for data mining (Wu et al., 2007):

- C4.5 and beyond
- The k-means algorithm
- Support vector machines
- The Apriori algorithm
- The EM algorithm
- PageRank

• AdaBoost

• kNN: k-nearest neighbor classification

• Naive Bayes

• CART: Classification and Regression Trees

All these algorithms have been widely used in all the fields. Although these algorithms do have some limitations and cannot be implemented in all the fields but still are helpful in many. Presently with the help of these algorithms various fields are using data mining and data science techniques to grow and aid their own field. One of the most efficient field using data science techniques is the medical field. The main concern for medical industry is analyzing supply and demand. In order to regulate this scenario researcher are using big data and data sciences. Prediction based cures for many illnesses are now available for many patients. Researchers at Tamil Nadu University have developed genetic algorithm to detect diabetes. This algorithm uses data mining algorithm for accuracy and prediction. Fuzzy logics and neural networks are used to learn more about the disease (S, Tamilarasi, & Pravin Kumar, 2012).

Climate control department also is using prediction methods to predict the climate changes and develop prevention measures. These algorithms have been successfully implemented in many countries and produced great results. One of the main algorithm used is the clustering and k-means algorithms. These algorithms have given most accurate results up till now. Also various new algorithms are developed recently. Researchers have developed specialized pattern-mining algorithms to overcome previous issues in climate anomaly detection (James H et al., 2014). In order to detect temperature changes for a specific area it is vital to represent data in gridded form in which each grid represents a location. “The NCEP/NCAR Re analysis from the National Centers for Environmental Protection and the National Center for Atmospheric Research, has a 10,000-gridpoint resolution. Each grid point has 12 monthly observations for each of the 30 years, and numerous variables such as wind and rainfall might impact temperatures in Brazil” (James H et al., 2014). As per James H et al these technology has helped immensely to predict the data for climate changes accurately.

2.4. Future scope:

Taking into account all these current research in each field data science technology has helped mankind in many ways. Still this field needs a lot of exploration and researchers are continuously doing it. One of the major point for successful data analysis is to obtain the P-values within the data set, today it is one of the major issue. Strategies such as manual extraction, web cleaning and data cleaning are used but, these methodologies don't produce accurate results. Hence we still have to wait until new methods are introduced to obtain appropriate values and develop good computational data sets. Donoho, D. and Matan Gavish have proposed various new methods to extract data which have given better results up till now. Donoho and Matan have proposed to use Verifiable Computational Result, cloud computing and cloud storage to record and store values and results, also retrieving those with help of URL's (Donoho, 2015).

Many researchers are also developing models for data science useful in business. One such example is BDS model developed by Russell Newman and others. This business model is developed and under experimentation for various business logics. Various big companies are now interested in data science to have the user databases and other business logics. The BDS model has following basic steps to get the dataset.

Algorithm: (Newman, Chang, Walters, & Wills, 2016).

• Collect data. Actual metrics to be derived from model.

• Run analysis to establish a quantified benchmark for the dot-com bubble period.

• Run the same analysis on a subsequent period.

• a Compare results for this period to the benchmark created in step 2. Any similarities/differences in the results may be indicative of the presence/absence of a bubble state.

• Return to step 3 for additional time periods (optional)

2.5. Conclusion:

These new developments in the field of data science will change the world and help humans to predict the data usage. The old methods of statistical analysis will not be

completely overpowered but still these new techniques will be more efficient than the old ways. Considering the progress speed there will be accurate and efficient techniques in next few years.

CHAPTER 3

Feasibility Study

The Feasibility Study determines the likelihood of a project's success. It delivers a single measure of this probability but comprises two independent evaluations. The first is the probability of Technical Success — P(T). It focuses on the project prototype or a proof of concept. The second determines the probability of Commercial Success — P(C). It evaluates the likelihood that the prototype can be scaled to production and adopted by the company or its customers. The product of these two measures is the Probability of Success: P(S) = P(T)P(C). We treat each measure as an independent variable. When evaluating the probability of commercial success, we assume that the prototype has already been successfully developed.

Before unpacking the derivation for each score I should clarify that our approach is seldom static. Most data science and research projects do share characteristics and benefit from a consistent approach to risk assessment. However, it is often necessary to adapt the details of an assessment to the organizational peculiarities of a project, and to take into account the nuances of industry verticals. Please consider, as we must with every project, what fits your needs. Do so with the intent to remain objective and to avoid the temptation to adjust the parameters of an assessment so that your favorite projects return a better, but inflated, score.

3.1. Technical Success Evaluation:

Estimating the probability of technical success — designated as P(T) from now on — is an exercise in identifying the salient risk factors, then scoring each individually on a consistent scale before plugging the values into a mathematical function. Easy. But what exactly is "technical success"?

The definition of P(T) is as follows: a likelihood that, given existing data, available technologies, required research, and development, the prototype will be completed

and will perform the defined function in a curated laboratory or constrained production environment. P(T) is given as a percentage.

The definition is important as it sets expectations for the deliverables. We limit the scope of P(T) to the prototype designed to test the technologies and convince a moderate sceptic that the approach works. The tests should run in a limited but representative case and use a vetted data set. Why the limitations? They decrease the time needed to develop the prototype without compromising the evaluation. They reduce the chance of misalignment with final goals by facilitating minor, continual adjustments. They reduce the risk

of investment. At the same time, the process still validates the difficult components and algorithms required for the production solution.

We defined the specifics of the prototype in the prior engagement stage: Concept Development. This is the subject for a different article, and we will assume that our prototype has a clear purpose, associated user persona, definition of function, and unacceptable failure modes.

Now, we evaluate what it takes to build the prototype. I will compress the process into a series of questions which elicit investigation and result in a score from 1 to 10, where 1 is a categorical no and 10 is a categorical yes. We group the evaluation into four equally weighted categories: Data, Technology, Business, and Expertise.

Data:

1. We have all required data features

2. The data for each feature is complete

3. The data for each feature is clean and accurate

4. We have ability to synthetically simulate sample data

5. We can access data without bureaucratic roadblocks

Technology:

6. The Technology Readiness Level (TRL) for the majority of the required technology. (See NASA https://www.nasa.gov/directorates/heo/scan/engineering/technology/technology_readiness_level)

7. Lowest TRL of an individual required component

8. We have access to source-code or libraries for a related solution

9. We have access to comprehensive documentation for the low TRL components.

10. We have the computational capacity to achieve adequate results

11. We have a clear path to validate the correctness of the result

Business:

12. The project has a strong internal champion

13. The project has a strong internal ROI

14. The project team has a defined charter

15. The project has an adequate budget

Expertise:

16. Client has an internal subject matter expert

17. The team (Ordinal Science) has the exact alignment of expertise with the low TRL components

18. The team has successfully concluded research and prototype development in a conceptually similar space

19. The team has access to external resources available for consulting on low TRL components

20. The team is granted a degree of independence during the initial research phase

We assign a score to each of the 20 parameters, then plug the numbers into the formula we developed. We call the resulting number the “ORA Score” — Ordinal Risk Assessment Score.

$$P(T) = \prod_{i=0}^{N} \frac{\ln(\hat{s_i} + C)}{\ln(S_{max} + C)}$$, where

$$\hat{s_i} = S_{max} - \lfloor (w_i * (S_{\max} - s_i) \rfloor$$

ORA Score

S is an array of individual parameter scores, where ***s*** is an integer between 1 and 10

N is a number of parameters, in our case ***N*** = 20

C is a compression coefficient, ***C = aN***, where ***a*** = 3 (empirically chosen)

W is an array of parameter weights that determine the criticality of each parameter, where ***s*** is an integer between 1 and 6. ***w = 6*** for a single parameter when ***s = 1*** will have an effect of suppressing the *P(T)* below 0.5.

w for all ***W*** is the safest, unless the reasons for particular weights are well understood.

If you said “Huh?” to the above, no worries — give us a call and we will do the Feasibility assessment for you. That is our function. If you followed along then a word of caution on the weights. They matter very much and must make sense in the context of your business. They are not equal to 1 for all features all the time. How we derive our weight vectors is a bit of a trade secret, but with much experimentation you may arrive at a reasonable set for your context.

The primary purpose of defining the P(T) is to decide whether to continue to the next phase. What is the right threshold for moving forward? I don’t know. We generally gate our projects at 70% likelihood of technical success, but the context matters as well as the risk tolerance of the client. The proper threshold should be determined by a business discussion with stakeholders.

Chapter 4

Data Collection and Wrangling

4.1. Collecting the Data via API:

SpaceX Falcon 9 first stage Landing Prediction:

Lab 1: Collecting the data

In this capstone, we will predict if the Falcon 9 first stage will land successfully. SpaceX advertises Falcon 9 rocket launches on its website with a cost of 62 million dollars; other providers cost upward of 165 million dollars each, much of the savings is because SpaceX can reuse the first stage. Therefore, if we can determine if the first stage will land, we can determine the cost of a launch. This information can be used if an alternate company wants to bid against SpaceX for a rocket launch. In this lab, you will collect and make sure the data is in the correct format from an API. The following is an example of a successful and launch.

Several examples of an unsuccessful landing are shown here:

Most unsuccessful landings are planned. Space X performs a controlled landing in the oceans.

Objectives:

In this lab, you will make a get request to the SpaceX API. You will also do some basic data wrangling and formatting.

- Request to the SpaceX API
- Clean the requested data

Import Libraries and Define Auxiliary Functions:

We will import the following libraries into the lab

In [1]:

```
# Requests allows us to make HTTP requests which we will use to get
data from an API
import requests
# Pandas is a software library written for the Python programming
language for data manipulation and analysis.
import pandas as pd
# NumPy is a library for the Python programming language, adding
support for large, multi-dimensional arrays and matrices, along with
a large collection of high-level mathematical functions to operate
on these arrays
import numpy as np
# Datetime is a library that allows us to represent dates
import datetime

# Setting this option will print all collumns of a dataframe
pd.set_option('display.max_columns', None)
# Setting this option will print all of the data in a feature
pd.set_option('display.max_colwidth', None)
```

Below we will define a series of helper functions that will help us use the API to extract information using identification numbers in the launch data.

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

In [2]:

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

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

In [3]:

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

In [4]:

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

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

In [5]:

```
# 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']).js
on()
```

```
                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:

In [6]:

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

In [7]:

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

Check the content of the response

In [8]:

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

Task 1: Request and parse the SpaceX launch data using the GET request

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/datasets/API_call_spacex_api.json'
```

We should see that the request was successful 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 data frame using .Json normalize()

In [14]:

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

Using the data frame data print the first 5 rows

In [15]:

```
# Get the head of the dataframe
data.head()
```

You will notice that a lot of the data are IDs. For example, the rocket column has no information about the rocket just an identification number.

We will now use the API again to get information about the launches using the IDs given for each launch. Specifically, we will be using
columns rocket, payloads, launchpad, and cores.

In [16]:

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

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

# Since payloads and cores are lists of size 1 we will also
extract the single value in the list and replace the feature.
```

```
data['cores'] = data['cores'].map(lambda x : x[0])
data['payloads'] = data['payloads'].map(lambda x : x[0])

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

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

- From the rocket we would like to learn the booster name
- From the payload we would like to learn the mass of the payload and the orbit that it is going to
- From the launchpad we would like to know the name of the launch site being used, the longitude, and the latitude.
- From cores we would like to learn the outcome of the landing, the type of the landing, number of flights with that core, whether grid fins were used, whether the core is reused, whether legs were used, the landing pad used, the block of the core which is a number used to separate version of cores, the number of times this specific core has been reused, and the serial of the core.

The data from these requests will be stored in lists and will be used to create a new data frame.

In [17]:

```
#Global variables
BoosterVersion = []
PayloadMass = []
Orbit = []
LaunchSite = []
Outcome = []
Flights = []
GridFins = []
Reused = []
Legs = []
LandingPad = []
Block = []
```

```
ReusedCount = []
Serial = []
Longitude = []
Latitude = []
```

These functions will apply the outputs globally to the above variables. Let's take a looks at BoosterVersion variable. Before we apply getBoosterVersion the list is empty:

In [18]:

```
BoosterVersion
```

Out [18]:

```
[]
```

Now, let's apply getBoosterVersion function method to get the booster version

In [19]:

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

the list has now been updated

In [20]:

```
BoosterVersion[0:5]
```

Out [20]:

```
['Falcon 1', 'Falcon 1', 'Falcon 1', 'Falcon 1', 'Falcon 9']
```

we can apply the rest of the functions here:

In [21]:

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

In [22]:

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

In [23]:

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

Finally let’s construct our dataset using the data we have obtained. We combine the columns into a dictionary.

In [24]:

```
launch_dict = {'FlightNumber': list(data['flight_number']),
```

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

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

In [25]:

```
# Create a data from launch_dict
launch_df = pd.DataFrame([launch_dict])
```

Show the summary of the data frame

In [26]:

```
# Show the head of the dataframe
launch_df.head()
```

Task 2: Filter the data frame to only include Falcon 9 launches

Finally, we will remove the Falcon 1 launches keeping only the Falcon 9 launches. Filter the data frame using the Booster Version column to only keep the Falcon 9 launches. Save the filtered data to a new data frame called data_falcon9.

In [27]:

```
# Hint data['BoosterVersion']!='Falcon 1'
data_falcon9 = launch_df[launch_df['BoosterVersion'] != 'Falcon 9']
data_falcon9
```

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

In [28]:

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

Data Wrangling:

We can see below that some of the rows are missing values in our dataset.

In [29]:

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

Out[29]:

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

Before we can continue, we must deal with these missing values. The Landing Pad column will retain No values to represent when landing pads were not used.

Task 3: Dealing with Missing Values

Calculate below the mean for the Payload Mass using the. mean (). Then use the mean and the. replace () function to replace np.nan values in the data with the mean you calculated.

In [30]:

```
# Calculate the mean value of PayloadMass column
```

```
PayloadMass =
pd.DataFrame(data_falcon9['PayloadMass'].values.tolist()).mean(1)

# Replace the np.nan values with its mean value
rows = data_falcon9['PayloadMass'].values.tolist()[0]

df_rows = pd.DataFrame(rows)
df_rows = df_rows.replace(np.nan, PayloadMass)

data_falcon9['PayloadMass'][0] = df_rows.values
data_falcon9.head()
# Replace the np.nan values with its mean value
```

You should see the number of missing values of the PayLoadMass change to zero.

In [31]:

```
data_falcon9.to_csv('dataset_part\_1.csv', index=False)
```

Now we should have no missing values in our dataset except for in Landing Pad.

We can now export it to a **CSV** for the next section, but to make the answers consistent, in the next lab we will provide data in a pre-selected date range.

data_falcon9.to_csv('dataset_part_1.csv', index=False)

Authors

Joseph Sant Arcangelo has a PhD in Electrical Engineering, his research focused on using machine learning, signal processing, and computer vision to determine how videos impact human cognition. Joseph has been working for IBM since he completed his PhD.

Change Log

Table 1: Week 1.1 Change log

Date (YYYY-MM-DD)	Version	Changed By	Change Description
2020-09-20	1.1	Joseph	get result each time you run
2020-09-20	1.1	Azim	Created Part 1 Lab using SpaceX API

4.2. Collecting the Data via Web Scraping:

Space X Falcon 9 First Stage Landing Prediction:

Web scraping Falcon 9 and Falcon Heavy Launches Records from Wikipedia

In this lab, you will be performing web scraping to collect Falcon 9 historical launch records from a Wikipedia page titled List of Falcon 9 and Falcon Heavy launches

https://en.wikipedia.org/wiki/List_of_Falcon_9_and_Falcon_Heavy_launches

Falcon 9 first stage will land successfully

Several examples of an unsuccessful landing are shown here:

More specifically, the launch records are stored in a HTML table shown below:

Objectives:

Web scrap Falcon 9 launch records with Beautiful Soup:

- Extract a Falcon 9 launch records HTML table from Wikipedia
- Parse the table and convert it into a Pandas data frame

First let's import required packages for this lab

In [1]:

```
!pip3 install beautifulsoup4
!pip3 install requests
Requirement already satisfied: beautifulsoup4 in /home/jupyterlab/co
nda/envs/python/lib/python3.7/site-packages (4.11.1)
Requirement already satisfied: soupsieve>1.2 in /home/jupyterlab/con
da/envs/python/lib/python3.7/site-packages (from beautifulsoup4) (2.
3.2.post1)
```

```
Requirement already satisfied: requests in /home/jupyterlab/conda/en
vs/python/lib/python3.7/site-packages (2.27.1)
Requirement already satisfied: certifi>=2017.4.17 in /home/jupyterla
b/conda/envs/python/lib/python3.7/site-packages (from requests) (202
1.10.8)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /home/jupyte
rlab/conda/envs/python/lib/python3.7/site-packages (from requests) (
1.26.9)
Requirement already satisfied: idna<4,>=2.5 in /home/jupyterlab/cond
a/envs/python/lib/python3.7/site-packages (from requests) (3.3)
Requirement already satisfied: charset-normalizer~=2.0.0 in /home/ju
pyterlab/conda/envs/python/lib/python3.7/site-packages (from request
s) (2.0.12)
```

In [7]:

```
import sys

import requests
from bs4 import BeautifulSoup
import re
import unicodedata
import pandas as pd
```

and we will provide some helper functions for you to process web scraped HTML table

In [8]:

```
def date_time(table_cells):
    """
    This function returns the data and time from the HTML  table
cell
    Input: the  element of a table data cell extracts extra row
    """
    return [data_time.strip() for data_time in
list(table_cells.strings)][0:2]

def booster_version(table_cells):
    """
    This function returns the booster version from the HTML  table
cell
    Input: the  element of a table data cell extracts extra row
    """
```

```
    out=''.join([booster_version for i,booster_version in enumerate(
table_cells.strings) if i%2==0][0:-1])
    return out

def landing_status(table_cells):
    """
    This function returns the landing status from the HTML table
cell
    Input: the  element of a table data cell extracts extra row
    """
    out=[i for i in table_cells.strings][0]
    return out


def get_mass(table_cells):
    mass=unicodedata.normalize("NFKD", table_cells.text).strip()
    if mass:
        mass.find("kg")
        new_mass=mass[0:mass.find("kg")+2]
    else:
        new_mass=0
    return new_mass


def extract_column_from_header(row):
    """
    This function returns the landing status from the HTML table
cell
    Input: the  element of a table data cell extracts extra row
    """
    if (row.br):
        row.br.extract()
    if row.a:
        row.a.extract()
    if row.sup:
        row.sup.extract()

    colunm_name = ' '.join(row.contents)

    # Filter the digit and empty names
```

```
    if not(colunm_name.strip().isdigit()):
        colunm_name = colunm_name.strip()
        return colunm_name
```

To keep the lab tasks consistent, you will be asked to scrape the data from a snapshot of the List of Falcon 9 and Falcon Heavy launches Wiki page updated on 9th June 2021

In [9]:

```
static_url =
"https://en.wikipedia.org/w/index.php?title=List_of_Falcon_9_and_Fal
con_Heavy_launches&oldid=1027686922"
```

Next, request the HTML page from the above URL and get a response object

TASK 1: Request the Falcon9 Launch Wiki page from its URL

First, let's perform an HTTP GET method to request the Falcon9 Launch HTML page, as an HTTP response.

In [14]:

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

Out[14]:

```
200
```

Create a BeautifulSoup object from the HTML response

In [15]:

```
# Use BeautifulSoup() to create a BeautifulSoup object from a
response text content
soup = BeautifulSoup(html_data.text, 'html.parser')
```

Print the page title to verify if the BeautifulSoup object was created properly

In [16]:

```
# Use soup.title attribute
soup.title
```

Out[16]:

```
<title>List of Falcon 9 and Falcon Heavy launches - Wikipedia</title
>
```

TASK 2: Extract all column/variable names from the HTML table header

Next, we want to collect all relevant column names from the HTML table header

Let's try to find all tables on the wiki page first. If you need to refresh your memory about BeautifulSoup, please check the external reference link towards the end of this lab

In [17]:

```
# 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 [18]:

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

Next, we just need to iterate through the <th> elements and apply the provided extract_column_from_header() to extract column name one by on

```
column_names = []

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

Check the extracted column names

In [22]:

```
print(column_names)
['Flight No.', 'Date and time ( )', '', 'Launch site', 'Payload', 'P
ayload mass', 'Orbit', 'Customer', 'Launch outcome', '', None, None,
None, None, None, None, None]
```

TASK 3: Create a data frame by parsing the launch HTML tables

We will create an empty dictionary with keys from the extracted column names in the previous task. Later, this dictionary will be converted into a Pandas data frame

In [32]:

```
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']=[]
```

Next, we just need to fill up the launch_dict with launch records extracted from table rows.

Usually, HTML tables in Wiki pages are likely to contain unexpected annotations and other types of noises, such as reference links B0004.1[8], missing values N/A [e], inconsistent formatting, etc.

To simplify the parsing process, we have provided an incomplete code snippet below to help you to fill up the launch_dict. Please complete the following code snippet with TODOs or you can choose to write your own logic to parse all launch tables:

In [33]:

```
extracted_row = 0
#Extract each table
for table_number,table in enumerate(soup.find_all('table',"wikitable
plainrowheaders collapsible")):
```

```
   # get table row
    for rows in table.find_all("tr"):
        #check to see if first table heading is as number
corresponding to launch a number
        if rows.th:
            if rows.th.string:
                flight_number=rows.th.string.strip()
                flag=flight_number.isdigit()
        else:
            flag=False
        #get table element
        row=rows.find_all('td')
        #if it is number save cells in a dictonary
        if flag:
            extracted_row += 1
            ## Flight Number value
            ## TODO: Append the flight_number into launch_dict with
key `Flight No.`
            ## print(flight_number)
            datatimelist=date_time(row[0])
            launch_dict['Flight No.'].append(extracted_row)
            print("flight number:",extracted_row)

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

            ## Time value
            ## TODO: Append the time into launch_dict with key
`Time`
            time = datatimelist[1]
            launch_dict['Time'].append(time)
            ## print(time)
            print("time:",time)

            ## Booster version
```

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

            ## Launch Site
            ## TODO: Append the bv into launch_dict with key `Launch
Site`
            launch_site = row[2].a.string
            launch_dict['Launch site'].append(launch_site)
            ##print(launch_site)
            print("launch site:",launch_site)

            ## Payload
            ## TODO: Append the payload into launch_dict with key
`Payload`
            payload = row[3].a.string
            launch_dict['Payload'].append(payload)
            ##print(payload)
            print("Payload:",payload)

            ## Payload Mass
            ## TODO: Append the payload_mass into launch_dict with
key `Payload mass`
            payload_mass = get_mass(row[4])
            launch_dict['Payload mass'].append(payload_mass)
            ##print(payload)
            print("Payload mass:",payload_mass)

            ## Orbit
            ## TODO: Append the orbit into launch_dict with key
`Orbit`
            orbit = row[5].a.string
            launch_dict['Orbit'].append(orbit)
            ##print(orbit)
            print("Orbit:",orbit)
```

```
            ## Customer
            ## TODO: Append the customer into launch_dict with key
`Customer`
            try: customer = row[6].a.string
            except AttributeError: customer = row[6].string
            launch_dict['Customer'].append(customer)
            ##print(customer)
            print("Customer:",customer)

            ## Launch outcome
            ## TODO: Append the launch_outcome into launch_dict with
key `Launch outcome`
            launch_outcome = list(row[7].strings)[0]
            launch_dict['Launch outcome'].append(launch_outcome)
            ## print(launch_outcome)
            print("Launch outcome:",launch_outcome)

            ## Booster landing
            ## TODO: Append the launch_outcome into launch_dict with
key `Booster landing`
            booster_landing = landing_status(row[8])
            launch_dict['Booster landing'].append(booster_landing)
            ## print(booster_landing)
            print("booster landing:",booster_landing)
```

After you have fill in the parsed launch record values into launch_dict, you can create a dataframe from it.

In [34]:

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

We can now export it to a **CSV** for the next section, but to make the answers consistent and in case you have difficulties finishing this lab.

Following labs will be using a provided dataset to make each lab independent.

In [35]:

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

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

Authors

Yan Luo

Nayef Abou Tayoun

Change Log

Table 2: Week 1.2 Change log

Date (YYYY-MM-DD)	Version	Changed By	Change Description
2021-06-09	1.0	Yan Luo	Tasks updates
2020-11-10	1.0	Nayef	Created the initial version

4.3. Data Wrangling:

Space X Falcon 9 First Stage Landing Prediction:

Lab 2: Data wrangling

Estimated time needed: **60** minutes

In this lab, we will perform some Exploratory Data Analysis (EDA) to find some patterns in the data and determine what would be the label for training supervised models.

In the data set, there are several different cases where the booster did not land successfully. Sometimes a landing was attempted but failed due to an accident; for example, True Ocean means the mission outcome was successfully landed to a specific region of the ocean while False Ocean means the mission outcome was unsuccessfully landed to a specific region of the ocean. True RTLS means the mission outcome was successfully landed to a ground pad False RTLS means the mission outcome was unsuccessfully landed to a ground pad.True ASDS means the mission outcome was successfully landed on a drone ship False ASDS means the mission outcome was unsuccessfully landed on a drone ship.

In this lab we will mainly convert those outcomes into Training Labels with 1 means the booster successfully landed 0 means it was unsuccessful.

Falcon 9 first stage will land successfully

Several examples of an unsuccessful landing are shown here:

Objectives:

Perform exploratory Data Analysis and determine Training Labels

- Exploratory Data Analysis
- Determine Training Labels

Import Libraries and Define Auxiliary Functions:

We will import the following libraries.

In [1]:

```
# Pandas is a software library written for the Python programming
language for data manipulation and analysis.
import pandas as pd
#NumPy is a library for the Python programming language, adding
support for large, multi-dimensional arrays and matrices, along with
a large collection of high-level mathematical functions to operate
on these arrays
import numpy as np
```

Data Analysis:

Load Space X dataset, from last section.

In [2]:

```
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)
```

Identify and calculate the percentage of the missing values in each attribute

In [3]:

```
 df.isnull().sum()/df.count()*100
FlightNumber       0.000
Date               0.000
BoosterVersion     0.000
PayloadMass        0.000
Orbit              0.000
LaunchSite         0.000
Outcome            0.000
Flights            0.000
GridFins           0.000
Reused             0.000
Legs               0.000
LandingPad        40.625
Block              0.000
```

```
ReusedCount        0.000
Serial             0.000
Longitude          0.000
Latitude           0.000
dtype: float64
```

Identify which columns are numerical and categorical:

In [4]:

```
df.dtypes
```

Out[4]:

```
FlightNumber        int64
Date               object
BoosterVersion     object
PayloadMass       float64
Orbit              object
LaunchSite         object
Outcome            object
Flights             int64
GridFins             bool
Reused               bool
Legs                 bool
LandingPad         object
Block             float64
ReusedCount         int64
Serial             object
Longitude         float64
Latitude          float64
dtype: object
```

TASK 1: Calculate the number of launches on each site

The data contains several Space X launch facilities: Cape Canaveral Space Launch Complex 40 **VAFB SLC 4E** , Vandenberg Air Force Base Space Launch Complex 4E **(SLC-4E)**, Kennedy Space Centre Launch Complex 39A **KSC LC 39A** .The location of each Launch Is placed in the column Launch Site

Next, let's see the number of launches for each site.

Use the method value_counts() on the column LaunchSite to determine the number of launches on each site:

In [5]:

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

Out[5]:

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

Each launch aims to a dedicated orbit, and here are some common orbit types:

- **LEO**: Low Earth orbit (LEO)is an Earth-centred orbit with an altitude of 2,000 km (1,200 mi) or less (approximately one-third of the radius of Earth),[1] or with at least 11.25 periods per day (an orbital period of 128 minutes or less) and an eccentricity less than 0.25.[2] Most of the manmade objects in outer space are in LEO \[1].

- **VLEO**: Very Low Earth Orbits (VLEO) can be defined as the orbits with a mean altitude below 450 km. Operating in these orbits can provide a number of benefits to Earth observation spacecraft as the spacecraft operates closer to the observation\[2].

- **GTO** A geosynchronous orbit is a high Earth orbit that allows satellites to match Earth's rotation. Located at 22,236 miles (35,786 kilometres) above Earth's equator, this position is a valuable spot for monitoring weather, communications and surveillance. Because the satellite orbits at the same speed that the Earth is turning, the satellite seems to stay in place over a single longitude, though it may drift north to south," NASA wrote on its Earth Observatory website \[3] .

- **SSO (or SO)**: It is a Sun-synchronous orbit also called a Heli synchronous orbit is a nearly polar orbit around a planet, in which the satellite passes over any given point of the planet's surface at the same local mean solar time [4] .

- **ES-L1** :At the Lagrange points the gravitational forces of the two large bodies cancel out in such a way that a small object placed in orbit there is in equilibrium relative to the centre of mass of the large bodies. L1 is one such point between the sun and the earth \[5] .

- **HEO** A highly elliptical orbit, is an elliptic orbit with high eccentricity, usually referring to one around Earth \[6].

- **ISS** A modular space station (habitable artificial satellite) in low Earth orbit. It is a multinational collaborative project between five participating space agencies: NASA (United States), Ros cosmos (Russia), JAXA (Japan), ESA (Europe), and CSA (Canada) \[7]

- **MEO** Geocentric orbits ranging in altitude from 2,000 km (1,200 mi) to just below geosynchronous orbit at 35,786 kilometres (22,236 mi). Also known as an intermediate circular orbit. These are "most commonly at 20,200 kilometres (12,600 mi), or 20,650 kilometres (12,830 mi), with an orbital period of 12 hours \[8]

- **HEO** Geocentric orbits above the altitude of geosynchronous orbit (35,786 km or 22,236 mi) \[9]

- **GEO** It is a circular geosynchronous orbit 35,786 kilometres (22,236 miles) above Earth's equator and following the direction of Earth's rotation \[10]

- **PO** It is one type of satellites in which a satellite passes above or nearly above both poles of the body being orbited (usually a planet such as the Earth \[11]

some are shown in the following plot:

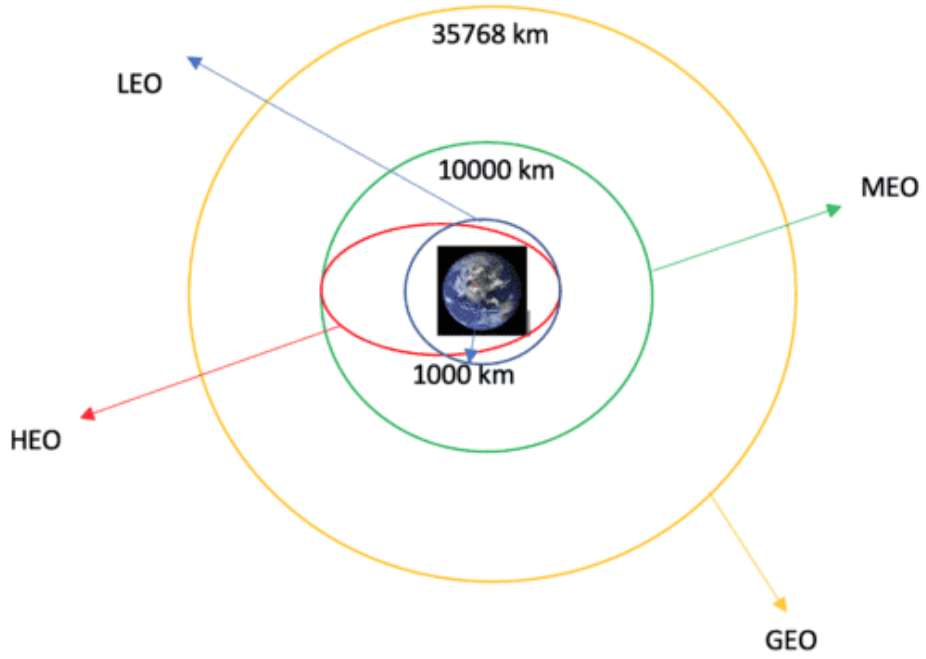


Figure 4: Orbit Information

TASK 2: Calculate the number and occurrence of each orbit

Use the method .**value_counts()** to determine the number and occurrence of each orbit in the column **Orbit**

In [6]:

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

Out[6]:

```
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
```

TASK 3: Calculate the number and occurrence of mission outcome per orbit type

Use the method .**value_counts()** on the column **Outcome** to determine the number of **landing_outcomes**.Then assign it to a variable landing_outcomes.

In [8]:

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

Out[8]:

```
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
```

True Ocean means the mission outcome was successfully landed to a specific region of the ocean while False Ocean means the mission outcome was unsuccessfully landed to a specific region of the ocean. True RTLS means the mission outcome was successfully landed to a ground pad False RTLS means the mission outcome was unsuccessfully landed to a ground pad. True ASDS means the mission outcome was successfully landed to a drone ship False ASDS means the mission outcome was unsuccessfully landed to a drone ship. None ASDS and None these represent a failure to land.

In [9]:

```
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
```

We create a set of outcomes where the second stage did not land successfully:

In [10]:

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

Out[10]:

```
{'False ASDS', 'False Ocean', 'False RTLS', 'None ASDS', 'None None'
}
```

TASK 4: Create a landing outcome label from Outcome column

Using the Outcome, create a list where the element is zero if the corresponding row in Outcome is in the set bad_outcome; otherwise, it's one. Then assign it to the variable landing_class:

In [11]:

```
# landing_class = 0 if bad_outcome
# landing_class = 1 otherwise
landing_class = []

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

This variable will represent the classification variable that represents the outcome of each launch. If the value is zero, the first stage did not land successfully; one means the first stage landed Successfully

In [12]:

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

Out[12]:

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

In [13]:

```
df.head(5)
df["Class"].mean()
```

Out[14]:

```
0.6666666666666666
```

We can now export it to a CSV for the next section, but to make the answers consistent, in the next lab we will provide data in a pre-selected date range.

In [15]:

```
df.to_csv("dataset_part\_2.csv", index=False)
```

df.to_csv("dataset_part_2.csv", index=False)

Authors

Joseph Santarcangelo has a PhD in Electrical Engineering, his research focused on using machine learning, signal processing, and computer vision to determine how videos impact human cognition. Joseph has been working for IBM since he completed his PhD.

Nayef Abou Tayoun is a Data Scientist at IBM and pursuing a Master of Management in Artificial intelligence degree at Queen's University.

Change Log

Table 3: Week 1.3 Change log

Date (YYYY-MM-DD)	Version	Changed By	Change Description
2021-08-31	1.1	Lakshmi Holla	Changed Markdown
2020-09-20	1.0	Joseph	Modified Multiple Areas
2020-11-04	1.1.	Nayef	updating the input data
2021-05-026	1.1.	Joseph	updating the input data

CHAPTER 5

Exploratory Data Analysis (EDA)

5.1. Exploratory Analysis Using SQL:

Assignment: SQL Notebook for Peer Assignment:

Estimated time needed: **60** minutes.

Introduction:

Using this Python notebook, you will:

1. Understand the Spacex DataSet
2. Load the dataset into the corresponding table in a Db2 database
3. Execute SQL queries to answer assignment questions

Overview of the Dataset:

SpaceX has gained worldwide attention for a series of historic milestones.

It is the only private company ever to return a spacecraft from low-earth orbit, which it first accomplished in December 2010. SpaceX advertises Falcon 9 rocket launches on its website with a cost of 62 million dollars whereas other providers cost upward of 165 million dollars each, much of the savings is because Space X can reuse the first stage.

Therefore, if we can determine if the first stage will land, we can determine the cost of a launch.

This information can be used if an alternate company wants to bid against SpaceX for a rocket launch.

This dataset includes a record for each payload carried during a SpaceX mission into outer space.

Download the datasets

This assignment requires you to load the spacex dataset.

In many cases the dataset to be analyzed is available as a .CSV (comma separated values) file, perhaps on the internet. Click on the link below to download and save the dataset (.CSV file):

[SpaceX Dataset]

Store the dataset in database table

it is highly recommended to manually load the table using the database console LOAD tool in DB2.

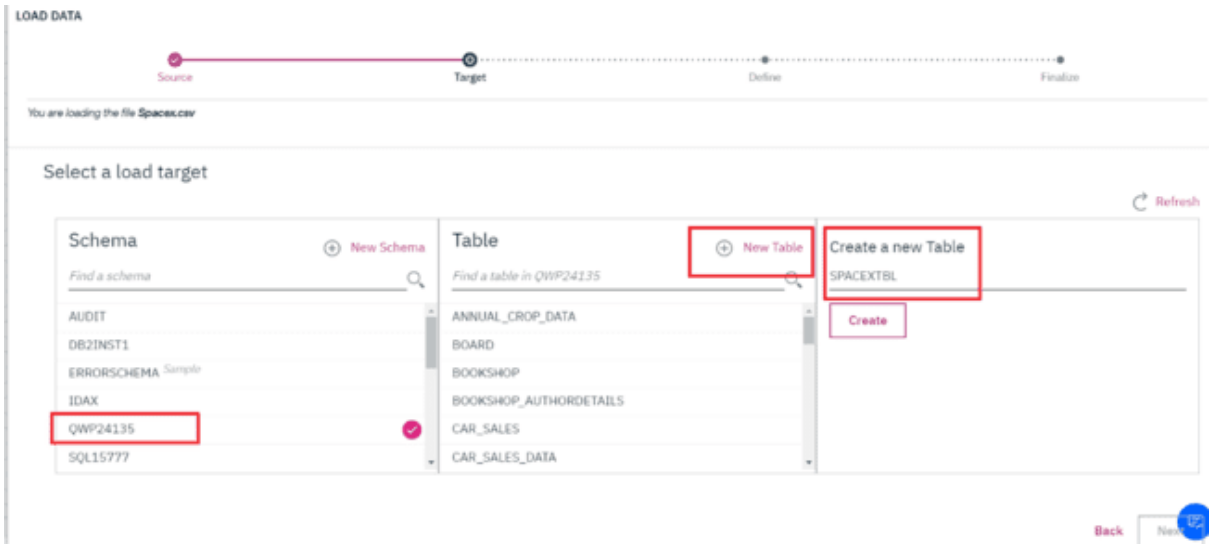


Figure 5: Load Data in New Table

Now open the Db2 console, open the LOAD tool, Select / Drag the .CSV file for the dataset, Next create a New Table, and then follow the steps on-screen instructions to load the data. Name the new table as follows:

SPACEX DATASET:

Follow these steps while using old DB2 UI which is having Open Console Screen

Note:While loading Spacex dataset, ensure that detect datatypes is disabled. Later click on the pencil icon(edit option).

1. Change the Date Format by manually typing DD-MM-YYYY and timestamp format as DD-MM-YYYY HH\:MM:SS.

   Here you should place the cursor at Date field and manually type as DD-MM-YYYY.

2. Change the PAYLOAD*MASS*_KG_ datatype to INTEGER.

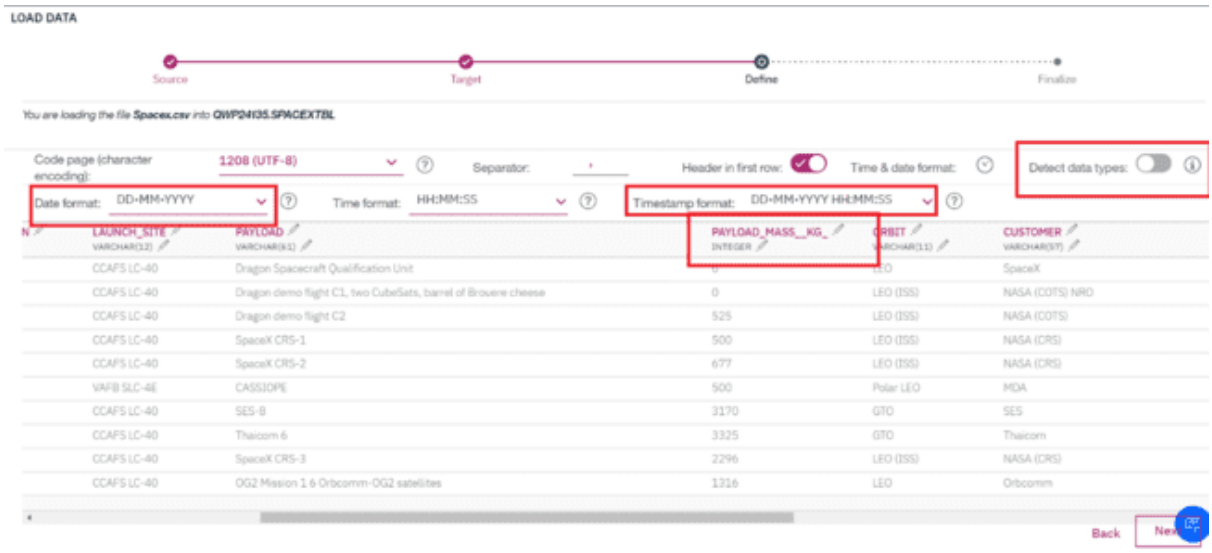


Figure 6: SpaceX.csv

Changes to be considered when having DB2 instance with the new UI having Go to UI screen

- Refer to this insruction in this link for viewing the new Go to UI screen.
- Later click on **Data link(below SQL)** in the Go to UI screen and click on **Load Data** tab.
- Later browse for the downloaded spacex file.

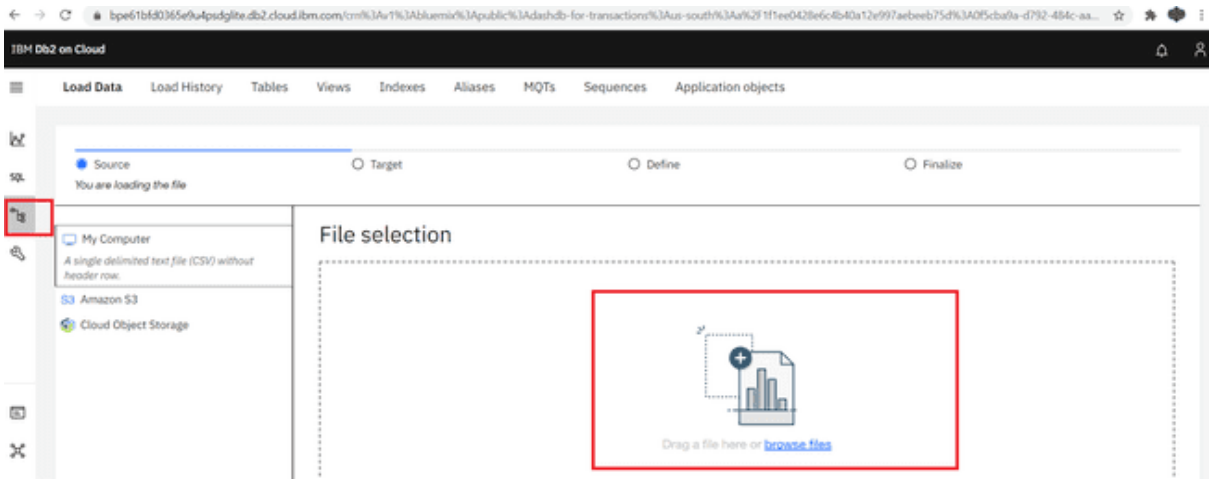


Figure 7: File Selection and Load the file

- Once done select the schema and load the file.

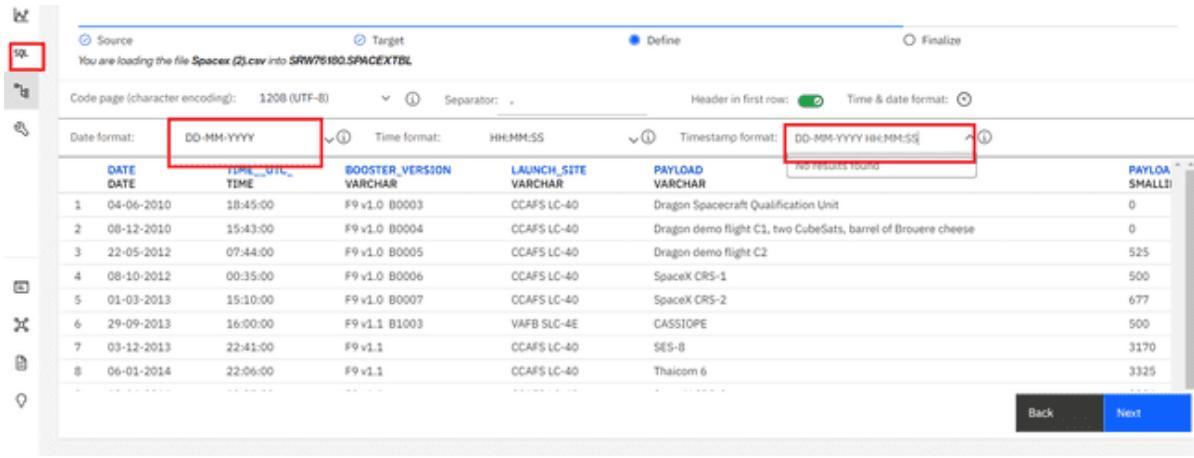


Figure 8: SpaceX(2).csv

In [1]:

```
!pip install sqlalchemy==1.3.9
!pip install ibm_db_sa
!pip install ipython-sql
Collecting sqlalchemy==1.3.9
  Downloading SQLAlchemy-1.3.9.tar.gz (6.0 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━
     ━━━━ 6.0/6.0 MB 53.7 MB/s eta 0:00:00:00:0100:01
  Preparing metadata (setup.py) ... done
Building wheels for collected packages: sqlalchemy
  Building wheel for sqlalchemy (setup.py) ... done
  Created wheel for sqlalchemy: filename=SQLAlchemy-1.3.9-cp37-cp37m
-linux_x86_64.whl size=1159140 sha256=42289e41625766e73004e9f6a3e00e
bddc3f73b4c8ff7ccfd7ae7eb8f922c5fc
  Stored in directory: /home/jupyterlab/.cache/pip/wheels/03/71/13/0
10faf12246f72dc76b4150e6e599d13a85b4435e06fb9e51f
Successfully built sqlalchemy
Installing collected packages: sqlalchemy
  Attempting uninstall: sqlalchemy
    Found existing installation: SQLAlchemy 1.3.24
    Uninstalling SQLAlchemy-1.3.24:
      Successfully uninstalled SQLAlchemy-1.3.24
Successfully installed sqlalchemy-1.3.9
Requirement already satisfied: ibm_db_sa in /home/jupyterlab/conda/e
nvs/python/lib/python3.7/site-packages (0.3.3)
```

```
Requirement already satisfied: sqlalchemy>=0.7.3 in /home/jupyterlab
/conda/envs/python/lib/python3.7/site-packages (from ibm_db_sa) (1.3
.9)
Requirement already satisfied: ipython-sql in /home/jupyterlab/conda
/envs/python/lib/python3.7/site-packages (0.3.9)
Requirement already satisfied: ipython>=1.0 in /home/jupyterlab/cond
a/envs/python/lib/python3.7/site-packages (from ipython-sql) (7.32.0
)
Requirement already satisfied: ipython-genutils>=0.1.0 in /home/jupy
terlab/conda/envs/python/lib/python3.7/site-packages (from ipython-s
ql) (0.2.0)
Requirement already satisfied: prettytable in /home/jupyterlab/conda
/envs/python/lib/python3.7/site-packages (from ipython-sql) (3.2.0)
Requirement already satisfied: six in /home/jupyterlab/conda/envs/py
thon/lib/python3.7/site-packages (from ipython-sql) (1.16.0)
Requirement already satisfied: sqlalchemy>=0.6.7 in /home/jupyterlab
/conda/envs/python/lib/python3.7/site-packages (from ipython-sql) (1
.3.9)
Requirement already satisfied: sqlparse in /home/jupyterlab/conda/en
vs/python/lib/python3.7/site-packages (from ipython-sql) (0.4.2)
Requirement already satisfied: jedi>=0.16 in /home/jupyterlab/conda/
envs/python/lib/python3.7/site-packages (from ipython>=1.0->ipython-
sql) (0.18.1)
Requirement already satisfied: prompt-toolkit!=3.0.0,!=3.0.1,<3.1.0,
>=2.0.0 in /home/jupyterlab/conda/envs/python/lib/python3.7/site-pac
kages (from ipython>=1.0->ipython-sql) (3.0.29)
Requirement already satisfied: pexpect>4.3 in /home/jupyterlab/conda
/envs/python/lib/python3.7/site-packages (from ipython>=1.0->ipython
-sql) (4.8.0)
Requirement already satisfied: pickleshare in /home/jupyterlab/conda
/envs/python/lib/python3.7/site-packages (from ipython>=1.0->ipython
-sql) (0.7.5)
Requirement already satisfied: traitlets>=4.2 in /home/jupyterlab/co
nda/envs/python/lib/python3.7/site-packages (from ipython>=1.0->ipyt
hon-sql) (5.1.1)
Requirement already satisfied: backcall in /home/jupyterlab/conda/en
vs/python/lib/python3.7/site-packages (from ipython>=1.0->ipython-sq
l) (0.2.0)
```

```
Requirement already satisfied: decorator in /home/jupyterlab/conda/e
nvs/python/lib/python3.7/site-packages (from ipython>=1.0->ipython-s
ql) (5.1.1)
Requirement already satisfied: pygments in /home/jupyterlab/conda/en
vs/python/lib/python3.7/site-packages (from ipython>=1.0->ipython-sq
l) (2.11.2)
Requirement already satisfied: setuptools>=18.5 in /home/jupyterlab/
conda/envs/python/lib/python3.7/site-packages (from ipython>=1.0->ip
ython-sql) (62.1.0)
Requirement already satisfied: matplotlib-inline in /home/jupyterlab
/conda/envs/python/lib/python3.7/site-packages (from ipython>=1.0->i
python-sql) (0.1.3)
Requirement already satisfied: wcwidth in /home/jupyterlab/conda/env
s/python/lib/python3.7/site-packages (from prettytable->ipython-sql)
(0.2.5)
Requirement already satisfied: importlib-metadata in /home/jupyterla
b/conda/envs/python/lib/python3.7/site-packages (from prettytable->i
python-sql) (4.11.3)
Requirement already satisfied: parso<0.9.0,>=0.8.0 in /home/jupyterl
ab/conda/envs/python/lib/python3.7/site-packages (from jedi>=0.16->i
python>=1.0->ipython-sql) (0.8.3)
Requirement already satisfied: ptyprocess>=0.5 in /home/jupyterlab/c
onda/envs/python/lib/python3.7/site-packages (from pexpect>4.3->ipyt
hon>=1.0->ipython-sql) (0.7.0)
Requirement already satisfied: typing-extensions>=3.6.4 in /home/jup
yterlab/conda/envs/python/lib/python3.7/site-packages (from importli
b-metadata->prettytable->ipython-sql) (4.2.0)
Requirement already satisfied: zipp>=0.5 in /home/jupyterlab/conda/e
nvs/python/lib/python3.7/site-packages (from importlib-metadata->pre
ttytable->ipython-sql) (3.8.0)
```

Connect to the database

Let us first load the SQL extension and establish a connection with the database

In [2]:

```
%load_ext sql
```

DB2 magic in case of old UI service credentials.

In the next cell enter your db2 connection string. Recall you created Service Credentials for your Db2 instance before. From the **uri** field of your Db2 service credentials copy

everything after db2:// (except the double quote at the end) and paste it in the cell below after ibm_db_sa://

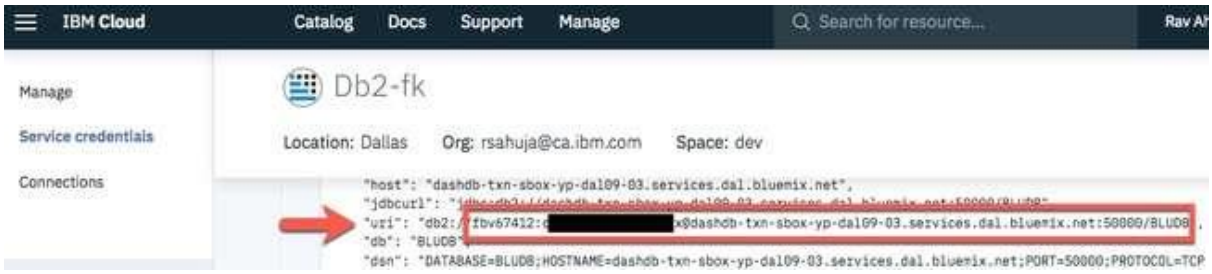


Figure 9: Db2-fk

in the following format

%sql ibm_db_sa://my-username:my-password\@my-hostname:my-port/my-db-name

DB2 magic in case of new UI service credentials.

Figure 10: %sql ibm_db_sa://my-username:my-password\@my-hostname:my-port/my-db-name

- Use the following format.
- Add security=SSL at the end

%sql ibm_db_sa://my-username:my-password\@my-hostname:my-port/my-db-name?security=SSL

In [6]:

```
%sql ibm_db_sa://glq09342:qwptGfJNs4DgeVNk@1bbf73c5-d84a-4bb0-85b9-ab1a4348f4a4.c3n41cmd0nqnrk39u98g.databases.appdomain.cloud:32286/BLUDB?security=SSL
```

Out[6]:

```
'Connected: glq09342@BLUDB'
```

In [8]:

```
%sql select * from spacex
 * ibm_db_sa://glq09342:***@1bbf73c5-d84a-4bb0-85b9-ab1a4348f4a4.c3n41cmd0nqnrk39u98g.databases.appdomain.cloud:32286/BLUDB
Done.
```

Out[8]:

Table 4: Dataset

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 Broue re	0	LEO (ISS)	NASA (COTS) NRO	Success	Failure (parachute)

DATE	time__utc_	booster_version	launch_site	payload	payload_mass__kg_	orbit	customer	mission_outcome	landing__outcome
				cheese					
2012-05-22	07:44:00	F9 v1.0 B0005	CCAFS LC-40	Dragon demo flight C2	525	LEO (ISS)	NASA (COTS)	Success	No attempt
2012-10-08	00:35:00	F9 v1.0 B0006	CCAFS LC-40	SpaceX CRS-1	500	LEO (ISS)	NASA (CRS)	Success	No attempt
2013-03-01	15:10:00	F9 v1.0 B0007	CCAFS LC-40	SpaceX CRS-2	677	LEO (ISS)	NASA (CRS)	Success	No attempt
2013-09-29	16:00:00	F9 v1.1 B1003	VAFB SLC-4E	CASSIOPE	500	Polar LEO	MDA	Success	Uncontrolled (ocean)

DATE	time__utc_	booster_version	launch_site	payload	payload_mass__kg_	orbit	customer	mission_outcome	landing__outcome
2013-12-03	22:41:00	F9 v1.1	CCAFS LC-40	SES-8	3170	GTO	SES	Success	No attempt
2014-01-06	22:06:00	F9 v1.1	CCAFS LC-40	Thaicom 6	3325	GTO	Thaicom	Success	No attempt
2014-04-18	19:25:00	F9 v1.1	CCAFS LC-40	SpaceX CRS-3	2296	LEO (ISS)	NASA (CRS)	Success	Controlled (ocean)
2014-07-14	15:15:00	F9 v1.1	CCAFS LC-40	OG2 Mission 16 Orbcomm-OG2 satellites	1316	LEO	Orbcomm	Success	Controlled (ocean)

DATE	time__utc_	booster_version	launch_site	payload	payload_mass__kg_	orbit	customer	mission_outcome	landing__outcome
2014-08-05	08:00:00	F9 v1.1	CCAFS LC-40	AsiaSat 8	4535	GTO	AsiaSat	Success	No attempt
2014-09-07	05:00:00	F9 v1.1 B1011	CCAFS LC-40	AsiaSat 6	4428	GTO	AsiaSat	Success	No attempt
2014-09-21	05:52:00	F9 v1.1 B1010	CCAFS LC-40	SpaceX CRS-4	2216	LEO (ISS)	NASA (CRS)	Success	Uncontrolled (ocean)
2015-01-10	09:47:00	F9 v1.1 B1012	CCAFS LC-40	SpaceX CRS-5	2395	LEO (ISS)	NASA (CRS)	Success	Failure (drone ship)
2015-02-	23:03:00	F9 v1.1 B1013	CCAFS LC-40	DSCOVR	570	HEO	U.S. Air Force NASA NOAA	Success	Controlled (ocean)

DATE	time__utc_	booster_version	launch_site	payload	payload_mass__kg_	orbit	customer	mission_outcome	landing__outcome
11									
2015-03-02	03:50:00	F9 v1.1 B1014	CCAFS LC-40	ABS-3A Eutelsat 115 West B	4159	GTO	ABS Eutelsat	Success	No attempt
2015-04-14	20:10:00	F9 v1.1 B1015	CCAFS LC-40	SpaceX CRS-6	1898	LEO (ISS)	NASA (CRS)	Success	Failure (drone ship)
2015-04-27	23:03:00	F9 v1.1 B1016	CCAFS LC-40	Turkmen 52 / MonacoSAT	4707	GTO	Turkmenistan National Space Agency	Success	No attempt
2015-06-28	14:21:00	F9 v1.1 B1018	CCAFS LC-40	SpaceX CRS-7	1952	LEO (ISS)	NASA (CRS)	Failure (in flight)	Precluded (drone ship)

DATE	time__utc_	booster_version	launch_site	payload	payload_mass__kg_	orbit	customer	mission_outcome	landing__outcome
2015-12-22	01:29:00	F9 FT B1019	CCAFS LC-40	OG2 Mission 2 11 Orbcomm-OG2 satellites	2034	LEO	Orbcomm	Success	Success (ground pad)
2016-01-17	18:42:00	F9 v1.1 B1017	VAFB SLC-4E	Jason-3	553	LEO	NASA (LSP) NOAA CNES	Success	Failure (drone ship)
2016-03-04	23:35:00	F9 FT B1020	CCAFS LC-40	SES-9	5271	GTO	SES	Success	Failure (drone ship)
2016-04-08	20:43:00	F9 FT B1021.1	CCAFS LC-40	SpaceX CRS-8	3136	LEO (ISS)	NASA (CRS)	Success	Success (drone ship)

DATE	time__utc_	booster_version	launch_site	payload	payload_mass__kg_	orbit	customer	mission_outcome	landing__outcome
2016-05-06	05:21:00	F9 FT B1022	CCAFS LC-40	JCSAT-14	4696	GTO	SKY Perfect JSAT Group	Success	Success (drone ship)
2016-05-27	21:39:00	F9 FT B1023.1	CCAFS LC-40	Thaicom 8	3100	GTO	Thaicom	Success	Success (drone ship)
2016-06-15	14:29:00	F9 FT B1024	CCAFS LC-40	ABS-2A Eutelsat 117 West B	3600	GTO	ABS Eutelsat	Success	Failure (drone ship)
2016-07-18	04:45:00	F9 FT B1025.1	CCAFS LC-40	SpaceX CRS-9	2257	LEO (ISS)	NASA (CRS)	Success	Success (ground pad)
2016-08-	05:26:00	F9 FT B1026	CCAFS LC-40	JCSAT-16	4600	GTO	SKY Perfect JSAT Group	Success	Success (drone ship)

DATE	time__utc_	booster_version	launch_site	payload	payload_mass__kg_	orbit	customer	mission_outcome	landing__outcome
14									
2017-01-14	17:54:00	F9 FT B1029.1	VAFB SLC-4E	Iridium NEXT 1	9600	Polar LEO	Iridium Communications	Success	Success (drone ship)
2017-02-19	14:39:00	F9 FT B1031.1	KSC LC-39A	SpaceX CRS-10	2490	LEO (ISS)	NASA (CRS)	Success	Success (ground pad)
2017-03-16	06:00:00	F9 FT B1030	KSC LC-39A	EchoStar 23	5600	GTO	EchoStar	Success	No attempt
2017-03-30	22:27:00	F9 FT B1021.2	KSC LC-39A	SES-10	5300	GTO	SES	Success	Success (drone ship)

DATE	time__utc_	booster_version	launch_site	payload	payload_mass__kg_	orbit	customer	mission_outcome	landing__outcome
2017-05-01	11:15:00	F9 FT B1032.1	KSC LC-39A	NROL-76	5300	LEO	NRO	Success	Success (ground pad)
2017-05-15	23:21:00	F9 FT B1034	KSC LC-39A	Inmarsat-5 F4	6070	GTO	Inmarsat	Success	No attempt
2017-06-03	21:07:00	F9 FT B1035.1	KSC LC-39A	SpaceX CRS-11	2708	LEO (ISS)	NASA (CRS)	Success	Success (ground pad)
2017-06-23	19:10:00	F9 FT B1029.2	KSC LC-39A	BulgariaSat-1	3669	GTO	Bulsatcom	Success	Success (drone ship)

Tasks

Now write and execute SQL queries to solve the assignment tasks.

Task 1

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

In [9]:

```
%sql select distinct launch_site from spacex
 * ibm_db_sa://glq09342:***@1bbf73c5-d84a-4bb0-85b9-ab1a4348f4a4.c3n
41cmd0nqnrk39u98g.databases.appdomain.cloud:32286/BLUDB
Done.
```

Out[9]:

Table 5: Launch_site

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

Task 2

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

In [18]:

```
%sql select * from Spacex where launch_site like 'CCA%' limit 5
 * ibm_db_sa://glq09342:***@1bbf73c5-d84a-4bb0-85b9-ab1a4348f4a4.c3n
41cmd0nqnrk39u98g.databases.appdomain.cloud:32286/BLUDB
Done.
```

Out[18]:

Table 6: Display 5 records where launch sites begin with the string 'CCA'

DATE	time__utc_	booster_version	launch_site	payload	payload_mass__kg_	orbit	customer	mission_outcome	landing__outcome
2010-06-04	18:45:00	F9 v1.0 B0003	CCAFS LC-40	Dragon Spacecraft Qualification Unit	0	LEO	SpaceX	Success	Failure (parachute)
2010-12-08	15:43:00	F9 v1.0 B0004	CCAFS LC-40	Dragon demo flight C1, two CubeSats, barrel of Brouere cheese	0	LEO (ISS)	NASA (COTS) NRO	Success	Failure (parachute)
2012-05-22	07:44:00	F9 v1.0 B0005	CCAFS LC-40	Dragon demo flight C2	525	LEO (ISS)	NASA (COTS)	Success	No attempt
2012-10	00:35:00	F9 v1.0 B0006	CCAFS	SpaceX	500	LEO (I	NASA	Success	No attempt

DATE	time__utc_	booster_version	launch_site	payload	payload_mass__kg_	orbit	customer	mission_outcome	landing__outcome
-08			LC-40	CRS-1		SS)	(CRS)		
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)

In [19]:

```
%sql select sum(payload_mass__kg_) from spacex where customer like
'NASA%'
 * ibm_db_sa://glq09342:***@1bbf73c5-d84a-4bb0-85b9-ab1a4348f4a4.c3n
41cmd0nqnrk39u98g.databases.appdomain.cloud:32286/BLUDB
Done.
```

Out[19]:

Table 7: Display the total payload mass carried by boosters launched by NASA (CRS)

1
99980

Task 4

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

In [20]:

```
%sql select AVG(payload_mass__kg_) from spacex where booster_version
like 'F9 v1.1%'
 * ibm_db_sa://glq09342:***@1bbf73c5-d84a-4bb0-85b9-ab1a4348f4a4.c3n
41cmd0nqnrk39u98g.databases.appdomain.cloud:32286/BLUDB
Done.
```

Out[20]:

Table 8: Display average payload mass carried by booster version F9 v1.1

1
2534

Task 5

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

Hint: Use min function

In [21]:

```
%sql select MIN(DATE) from spacex where landing__outcome like
'%ground pad%'
 * ibm_db_sa://glq09342:***@1bbf73c5-d84a-4bb0-85b9-ab1a4348f4a4.c3n
41cmd0nqnrk39u98g.databases.appdomain.cloud:32286/BLUDB
Done.
```

Out[21]:

Table 9: List the date when the first successful landing outcome in ground pad was achieved.

1
2015-12-22

Task 6

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

In [22]:

```
%sql select distinct booster_version from spacex where
(payload_mass__kg_ < 6000 AND payload_mass__kg_ > 4000) AND
(landing__outcome like '%drone%ship%')
 * ibm_db_sa://glq09342:***@1bbf73c5-d84a-4bb0-85b9-ab1a4348f4a4.c3n
41cmd0nqnrk39u98g.databases.appdomain.cloud:32286/BLUDB
Done.
```

Out[22]:

Table 10: List the names of the boosters which have success in drone ship and have payload mass greater than 4000 but less than 6000

booster_version
F9 FT B1021.2
F9 FT B1031.2
F9 FT B1020
F9 FT B1022
F9 FT B1026

Task 7

List the total number of successful and failure mission outcomes

In [23]:

```
%sql SELECT mission_outcome, count(*) AS total FROM spacex GROUP BY
mission_outcome
```

```
 * ibm_db_sa://glq09342:***@1bbf73c5-d84a-4bb0-85b9-ab1a4348f4a4.c3n
41cmd0nqnrk39u98g.databases.appdomain.cloud:32286/BLUDB
Done.
```

Out[23]:

Table 11: List the total number of successful and failure mission outcomes

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

Task 8

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

In [24]:

```
%sql SELECT DISTINCT (booster_version) from spacex where
payload_mass__kg_ = (SELECT MAX(payload_mass__kg_) from spacex)
 * ibm_db_sa://glq09342:***@1bbf73c5-d84a-4bb0-85b9-ab1a4348f4a4.c3n
41cmd0
qnrk39u98g.databases.appdomain.cloud:32286/BLUDB
Done.
```

Out[24]:

Table 12: List the names of the booster versions which have carried the maximum payload mass. Use a subquery

booster_version
F9 B5 B1048.4
F9 B5 B1048.5
F9 B5 B1049.4

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

Task 9

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

In [25]:

```
%sql select booster_version, launch_site from spacex where
landing__outcome = 'Failure (drone ship)' AND DATE like '2015%'
 * ibm_db_sa://glq09342:***@1bbf73c5-d84a-4bb0-85b9-ab1a4348f4a4.c3n
41cmd0nqnrk39u98g.databases.appdomain.cloud:32286/BLUDB
Done.
```

Out[25]:

Table 13: List the failed landing_outcomes in drone ship, their booster versions, and launch site names for in year 2015

booster_version	launch_site
F9 v1.1 B1012	CCAFS LC-40
F9 v1.1 B1015	CCAFS LC-40

Task 10

Rank the count of landing outcomes (such as Failure (drone ship) or Success (ground pad)) between the date 2010-06-04 and 2017-03-20, in descending order

In [26]:

```
%sql SELECT * FROM spacex where DAYNAME(DATE)='Friday' LIMIT 5
 * ibm_db_sa://glq09342:***@1bbf73c5-d84a-4bb0-85b9-ab1a4348f4a4.c3n41cmd0nqnrk39u98g.databases.appdomain.cloud:32286/BLUDB
Done.
```

Out[26]:

Table 14: 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

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)
2013-03	15:10:00	F9 v1.0 B0007	CCAFS LC-40	SpaceX CRS-2	677	LEO (I	NASA (CRS)	Success	No attempt

DATE	time__utc_	booster_version	launch_site	payload	payload_mass__kg_	orbit	customer	mission_outcome	landing__outcome
-01						SS)			
2014-04-18	19:25:00	F9 v1.1	CCAFS LC-40	SpaceX CRS-3	2296	LEO (ISS)	NASA (CRS)	Success	Controlled (ocean)
2016-03-04	23:35:00	F9 FT B1020	CCAFS LC-40	SES-9	5271	GTO	SES	Success	Failure (drone ship)
2016-04-08	20:43:00	F9 FT B1021.1	CCAFS LC-40	SpaceX CRS-8	3136	LEO (ISS)	NASA (CRS)	Success	Success (drone ship)

Author(s)

Lakshmi Holla

Other Contributors

Rav Ahuja

Change log

Table 15: Week 2.1 Change Log

Date	Version	Changed by	Change Description
2021-10-12	0.4	Lakshmi Holla	Changed markdown
2021-08-24	0.3	Lakshmi Holla	Added library update
2021-07-09	0.2	Lakshmi Holla	Changes made in magic sql

5.2. Exploratory Analysis Using Pandas and Matplotlib:

SpaceX Falcon 9 First Stage Landing Prediction:

Assignment: Exploring and Preparing Data

Estimated time needed: **70** minutes

In this assignment, we will predict if the Falcon 9 first stage will land successfully. SpaceX advertises Falcon 9 rocket launches on its website with a cost of 62 million dollars; other providers cost upward of 165 million dollars each, much of the savings is because SpaceX can reuse the first stage.

In this lab, you will perform Exploratory Data Analysis and Feature Engineering.

Falcon 9 first stage will land successfully

Several examples of an unsuccessful landing are shown here:

Most unsuccessful landings are planned. Space X performs a controlled landing in the oceans.

Objectives:

Perform exploratory Data Analysis and Feature Engineering
using **Pandas** and **Matplotlib**

- Exploratory Data Analysis
- Preparing Data Feature Engineering

Import Libraries and Define Auxiliary Functions:

We will import the following libraries the lab

In [6]:

```
# andas is a software library written for the Python programming
language for data manipulation and analysis.
import pandas as pd
#NumPy is a library for the Python programming language, adding
support for large, multi-dimensional arrays and matrices, along with
```

```
a large collection of high-level mathematical functions to operate
on these arrays
import numpy as np
# Matplotlib is a plotting library for python and pyplot gives us a
MatLab like plotting framework. We will use this in our plotter
function to plot data.
import matplotlib.pyplot as plt
#Seaborn is a Python data visualization library based on matplotlib.
It provides a high-level interface for drawing attractive and
informative statistical graphics
import seaborn as sns
```

Exploratory Data Analysis:

First, let's read the SpaceX dataset into a Pandas dataframe and print its summary

In [7]:

```
df=pd.read_csv("https://cf-courses-data.s3.us.cloud-object-
storage.appdomain.cloud/IBM-DS0321EN-
SkillsNetwork/datasets/dataset_part_2.csv")

# If you were unable to complete the previous lab correctly you can
uncomment and load this csv

# df = pd.read_csv('https://cf-courses-data.s3.us.cloud-object-
storage.appdomain.cloud/IBMDeveloperSkillsNetwork-DS0701EN-
SkillsNetwork/api/dataset_part_2.csv')

df.head(5)
```

Out[7]:

Table 16: SpaceX dataset into a Pandas dataframe and its summary

	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	CCAFSSLC40	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	CCAFSSLC40	None None	1	False	False	False	NaN	1.0	0	B0005	-80.577366	28.561857	0
2	3	2013-	Falco	677.0	ISS	CCAFS	None N	1	Fal	Fal	Fal	NaN	1.0	0	B000	-80.5	28.56	0

	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins	Reused	Legs	LandingPad	Block	ReusedCount	Serial	Longitude	Latitude	Class
		03-01	n9	0000		SLC40	one		se	se	se				07	77366	18557	
3	4	2013-09-29	Falcon9	5000.0000000	PO	VAFBSLC4E	FalseOcean	1	False	False	False	NaN	1.0	0	B1003	-120.6108293	34.632093	0
4	5	2013-12-03	Falcon9	3170.0000000	GTO	CCAFSSLC40	NoneNone	1	False	False	False	NaN	1.0	0	B1004	-80.577366	28.561857	0

First, let's try to see how the **FlightNumber** (indicating the continuous launch attempts.) and **Payload** variables would affect the launch outcome.

We can plot out the **FlightNumber** vs. **PayloadMassand** overlay the outcome of the launch. We see that as the flight number increases, the first stage is more likely to land successfully. The payload mass is also important; it seems the more massive the payload, the less likely the first stage will return.

In [8]:

```
sns.catplot(y="PayloadMass", x="FlightNumber", hue="Class", data=df,
aspect = 5)
plt.xlabel("Flight Number",fontsize=20)
plt.ylabel("Pay load Mass (kg)",fontsize=20)
plt.show()
```

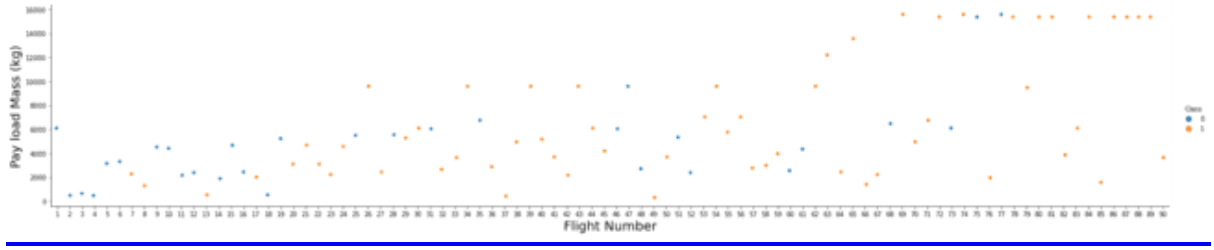


Figure 11: FlightNumber vs. PayloadMassand

We see that 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%.

Next, let's drill down to each site visualize its detailed launch records.

TASK 1: Visualize the relationship between Flight Number and Launch Site

Use the function **catplot** to plot **FlightNumber** vs **LaunchSite**, set the parameter **x** parameter to **FlightNumber**, set the **y** to **Launch** Site and set the parameter **hue** to '**class**'

Now try to explain the patterns you found in the Flight Number vs. Launch Site scatter point plots.

In [10]:

```
# 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 class value
sns.catplot(x="FlightNumber", y="LaunchSite", data=df, hue="Class",
aspect = 5)
plt.xlabel("Flight Number")
```

```
plt.ylabel("LaunchSite")
plt.show()
```

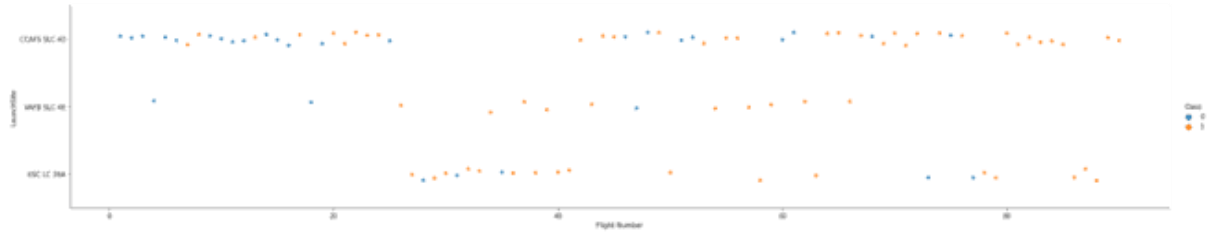


Figure 12: Visualize the relationship between Flight Number and Launch Site

TASK 2: Visualize the relationship between Payload and Launch Site

We also want to observe if there is any relationship between launch sites and their payload mass.

In [11]:

```
# 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 the class value
sns.catplot(x="PayloadMass", y="LaunchSite", hue="Class", data=df,
aspect = 5)
plt.xlabel("Flight Number")
plt.ylabel("LaunchSite")
plt.show()
```

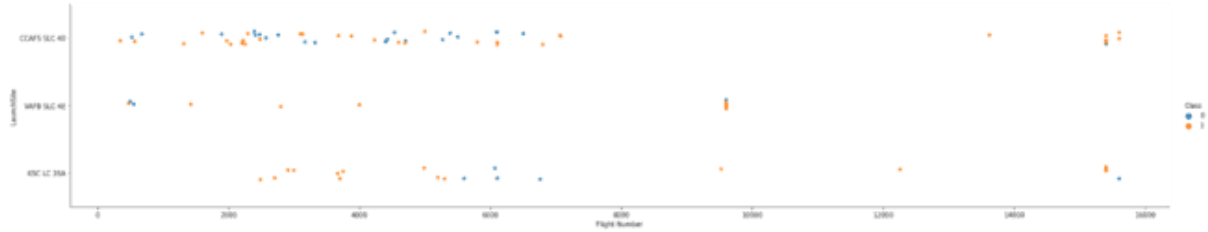


Figure 13: Visualize the relationship between Payload and Launch Site

Now if you observe Payload Vs. Launch Site scatter point chart you will find for the VAFB-SLC launchsite there are no rockets launched for heavypayload mass(greater than 10000).

TASK 3: Visualize the relationship between success rate of each orbit type

Next, we want to visually check if there are any relationship between success rate and orbit type.

Let's create a bar chart for the sucess rate of each orbit

In [13]:

```
# HINT use groupby method on Orbit column and get the mean of Class column
Orbittype=df.groupby(by="Orbit", axis=0).mean()["Class"]

Orbittype=pd.DataFrame(Orbittype)
Orbittype['orbit'] = Orbittype.index

sns.catplot(x="orbit", y="Class",  data=Orbittype, kind="bar", aspect=5)
```

Out[13]:

```
<seaborn.axisgrid.FacetGrid at 0x7f2b5972d490>
```

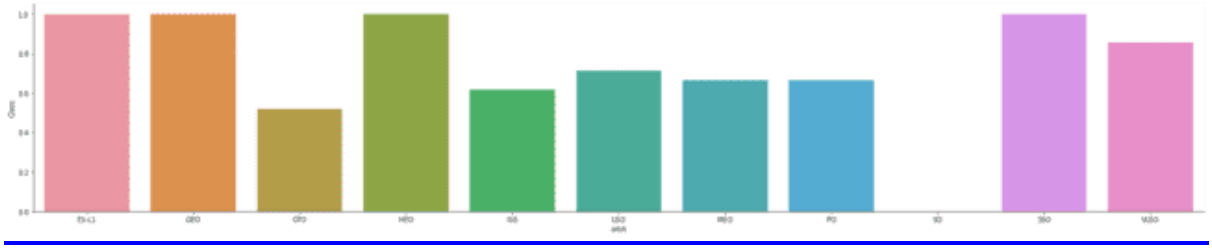


Figure 14: Visualize the relationship between success rate of each orbit type

Analyze the ploted bar chart try to find which orbits have high sucess rate.

Explain the patterns - which orbits have high sucess rate.

ES-L1, GEO, HEO, SSO has highest Sucess rates. *SO has poorest.*

TASK 4: Visualize the relationship between FlightNumber and Orbit type

For each orbit, we want to see if there is any relationship between FlightNumber and Orbit type.

In [14]:

```
# Plot a scatter point chart with x axis to be FlightNumber and y
axis to be the Orbit, and hue to be the class value
sns.catplot(x="FlightNumber", y="Orbit", hue="Class", data=df,
aspect = 5)
plt.xlabel("Flight Number")
plt.ylabel("Orbit")
plt.show()
```

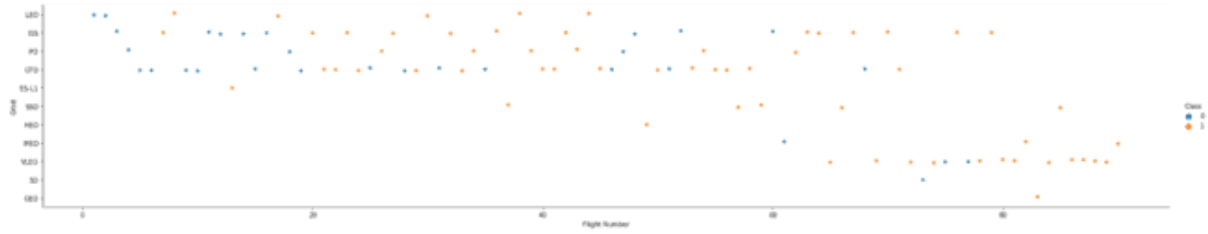


Figure 15: Visualize the relationship between success rate of each orbit type

You should see that in the LEO orbit the Success appears related to the number of flights; on the other hand, there seems to be no relationship between flight number when in GTO orbit.

TASK 5: Visualize the relationship between Payload and Orbit type

Similarly, we can plot the Payload vs. Orbit scatter point charts to reveal the relationship between Payload and Orbit type

In [15]:

```
# 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.catplot(x="PayloadMass", y="Orbit", data=df, hue="Class", aspect
= 5)
plt.xlabel("PayloadMass")
plt.ylabel("Orbit")
plt.show()
```

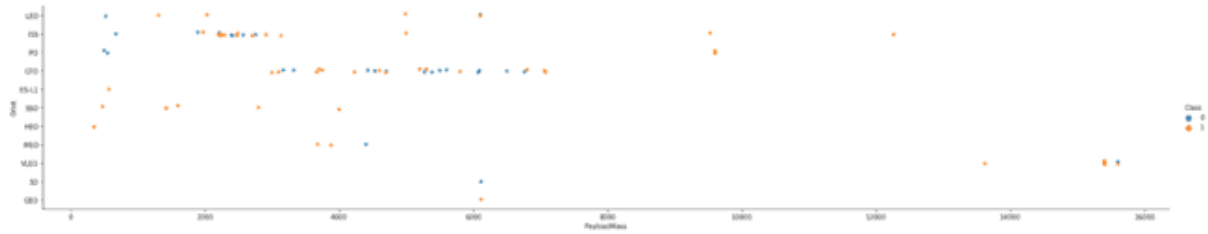


Figure 16: Visualize the relationship between Payload and Orbit type

With heavy payloads the successful landing or positive landing rate are more for Polar,LEO and ISS.

However for GTO we cannot distinguish this well as both positive landing rate and negative landing(unsuccessful mission) are both there here.

TASK 6: Visualize the launch success yearly trend

You can plot a line chart with x axis to be Year and y axis to be average success rate, to get the average launch success trend.

The function will help you get the year from the date:

In [19]:

```
# 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

df['year']=Extract_year(df)
by_year=df.groupby(by="year", axis=0).mean()["Class"]
by_year = pd.DataFrame(by_year)
by_year.reset_index(level=0, inplace=True)
by_year
```

Out[19]:

Table 17: Data Years from the date

	year	Class
0	2010	0.000000
1	2012	0.000000
2	2013	0.000000
3	2014	0.333333

	year	Class
4	2015	0.333333
5	2016	0.625000
6	2017	0.833333
7	2018	0.611111
8	2019	0.900000
9	2020	0.842105

In [20]:

```
# Plot a line chart with x axis to be the extracted year and y axis
to be the success rate
sns.lineplot( x="year", y="Class", data=by_year)
plt.xlabel("year")
plt.ylabel("class")
plt.show()
```

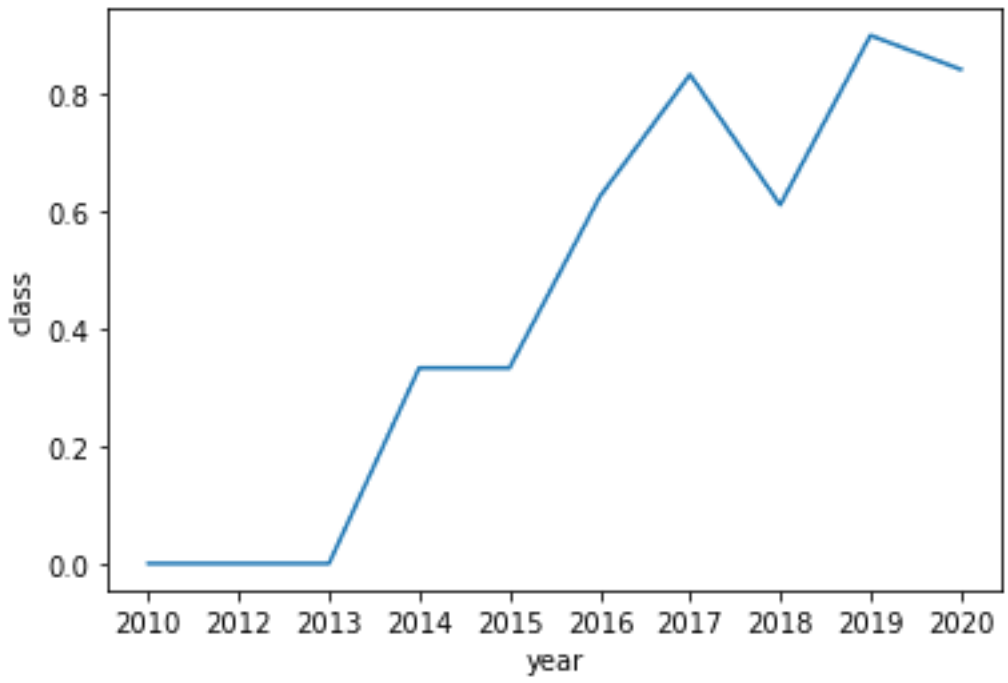


Figure 17: Visualize the launch success yearly trend

you can observe that the success rate since 2013 kept increasing till 2020

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.

In [27]:

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

Out[27]:

Table 18:

	FlightNumber	PayloadMass	Orbit	LaunchSite	Flights	GridFins	Reused	Legs	LandingPad	Block	ReusedCount	Serial
0	1	6104.959412	LEO	CCAFSSLC40	1	False	False	False	NaN	1.0	0	B0003
1	2	525.000000	LEO	CCAFSSLC40	1	False	False	False	NaN	1.0	0	B0005
2	3	677.000000	ISS	CCAFSSLC40	1	False	False	False	NaN	1.0	0	B0007

	FlightNumber	PayloadMass	Orbit	LaunchSite	Flights	GridFins	Reused	Legs	LandingPad	Block	ReusedCount	Serial
3	4	500.000000	PO	VAFBSLC4E	1	False	False	False	NaN	1.0	0	B1003
4	5	3170.000000	GTO	CCAFSSLC40	1	False	False	False	NaN	1.0	0	B1004

TASK 7: Create dummy variables to categorical columns

Use the function get_dummies and features dataframe to apply OneHotEncoder to the column Orbits, LaunchSite, LandingPad, and Serial. Assign the value to the variable features_one_hot, display the results using the method head. Your result dataframe must include all features including the encoded ones.

In [29]:

```
# HINT: Use get_dummies() function on the categorical columns
features_one_hot =
pd.get_dummies(features[['Orbit','LaunchSite','LandingPad','Serial']
])
features_one_hot.head()
```

Out[29]:

Table 19:

	Orbit_ES-L1	Orbit_GEO	Orbit_GTO	Orbit_HEO	Orbit_ISS	Orbit_LEO	Orbit_MEO	Orbit_PO	Orbit_SO	Orbit_SSO	...	Serial_B1048	Serial_B1049	Serial_B1050	Serial_B1051	Serial_B1054	Serial_B1056	Serial_B1058	Serial_B1059	Serial_B1060	Serial_B1062
0	0	0	0	0	0	1	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	1	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	1	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	1	0	0	...	0	0	0	0	0	0	0	0	0	0
4	0	0	1	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

5 rows × 72 columns

TASK 8: Cast all numeric columns to float64

Now that our **features_one_hot** dataframe only contains numbers cast the entire dataframe to variable type float64

In [30]:

```
# HINT: use astype function
features_one_hot =  features_one_hot.astype(float)
features_one_hot
```

	FlightNumber	PayloadMass	Flights	GridFins	Reused	Legs	Block	ReusedCount	ES-L1	GEO	.	B1048	B1049	B1050	B1051	B1054	B1056	B1058	B1059	B1060	B1062
0	1.0	6104.9594 12	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	.	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	2.0	525.00000 0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	.	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	3.0	677.00000 0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	.	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	4.0	500.00000 0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	.	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

	FlightNumber	PayloadMass	Flights	GridFins	Reused	Legs	Block	ReusedCount	ES-L1	GEO	.	B1048	B1049	B1050	B1051	B1054	B1056	B1058	B1059	B1060	B1062
4	5.0	3170.0000000	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	.	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
85	86.0	15400.0000000	2.0	1.0	1.0	1.0	5.0	2.0	0.0	0.0	.	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0
86	87.0	15400.0000000	3.0	1.0	1.0	1.0	5.0	2.0	0.0	0.0	.	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0
87	88.0	15400.000	6.0	1.0	1.0	1.0	5.0	5.0	0.0	0.0	.	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0

	FlightNumber	PayloadMass	Flights	GridFins	Reused	Legs	Block	ReusedCount	ESL1	GEO	.	B1048	B1049	B1050	B1051	B1054	B1056	B1058	B1059	B1060	B1062
88	89.0	15400.000000	3.0	1.0	1.0	1.0	5.0	2.0	0.0	0.0	.	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0
89	90.0	3681.000000	1.0	1.0	0.0	1.0	5.0	0.0	0.0	0.0	.	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0

90 rows × 80 columns

Out[30]:

We can now export it to a **CSV** for the next section,but to make the answers consistent, in the next lab we will provide data in a pre-selected date range.

In [31]:

```
features_one_hot.to_csv('dataset_part\_3.csv', index=False)
```

features_one_hot.to_csv('dataset_part_3.csv', index=False)

Authors

Joseph Santarcangelo has a PhD in Electrical Engineering, his research focused on using machine learning, signal processing, and computer vision to determine how videos impact human cognition. Joseph has been working for IBM since he completed his PhD.

Nayef Abou Tayoun is a Data Scientist at IBM and pursuing a Master of Management in Artificial intelligence degree at Queen's University.

Change Log

Table 20: Week 2.2 Change log

Date (YYYY-MM-DD)	Version	Changed By	Change Description
2021-10-12	1.1	Lakshmi Holla	Modified markdown
2020-09-20	1.0	Joseph	Modified Multiple Areas
2020-11-10	1.1	Nayef	updating the input data

CHAPTER 6

INTERACTIVE VISUAL ANALYTICS AND DASHBOARD

6.1:Interactive Visual Analytics with Folium

Launch Sites Locations Analysis with Folium:

Estimated time needed: **40** minutes

The 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. Finding an optimal location for building a launch site certainly involves many factors and hopefully we could discover some of the factors by analyzing the existing launch site locations.

In the previous exploratory data analysis labs, you have visualized the SpaceX launch dataset using matplotlib and seaborn and discovered some preliminary correlations between the launch site and success rates. In this lab, you will be performing more interactive visual analytics using Folium.

Objectives:

This lab contains the following tasks:

- **TASK 1:** Mark all launch sites on a map
- **TASK 2:** Mark the success/failed launches for each site on the map
- **TASK 3:** Calculate the distances between a launch site to its proximities

After completed the above tasks, you should be able to find some geographical patterns about launch sites.

Let's first import required Python packages for this lab:

In [1]:

```
!pip3 install folium
!pip3 install wget
```

```
Requirement already satisfied: folium in /Users/debdattasarkar/opt/a
naconda3/lib/python3.8/site-packages (0.12.1)
Requirement already satisfied: branca>=0.3.0 in /Users/debdattasarka
r/opt/anaconda3/lib/python3.8/site-packages (from folium) (0.4.2)
Requirement already satisfied: requests in /Users/debdattasarkar/opt
/anaconda3/lib/python3.8/site-packages (from folium) (2.25.1)
Requirement already satisfied: jinja2>=2.9 in /Users/debdattasarkar/
opt/anaconda3/lib/python3.8/site-packages (from folium) (2.11.3)
Requirement already satisfied: numpy in /Users/debdattasarkar/opt/an
aconda3/lib/python3.8/site-packages (from folium) (1.20.1)
Requirement already satisfied: MarkupSafe>=0.23 in /Users/debdattasa
rkar/opt/anaconda3/lib/python3.8/site-packages (from jinja2>=2.9->fo
lium) (1.1.1)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /Users/debda
ttasarkar/opt/anaconda3/lib/python3.8/site-packages (from requests->
folium) (1.26.4)
Requirement already satisfied: idna<3,>=2.5 in /Users/debdattasarkar
/opt/anaconda3/lib/python3.8/site-packages (from requests->folium) (
2.10)
Requirement already satisfied: certifi>=2017.4.17 in /Users/debdatta
sarkar/opt/anaconda3/lib/python3.8/site-packages (from requests->fol
ium) (2020.12.5)
Requirement already satisfied: chardet<5,>=3.0.2 in /Users/debdattas
arkar/opt/anaconda3/lib/python3.8/site-packages (from requests->foli
um) (4.0.0)
Requirement already satisfied: wget in /Users/debdattasarkar/opt/ana
conda3/lib/python3.8/site-packages (3.2)
```

In [2]:

```
import folium
import wget
import pandas as pd
```

In [3]:

```
# Import folium MarkerCluster plugin
from folium.plugins import MarkerCluster
# Import folium MousePosition plugin
from folium.plugins import MousePosition
# Import folium DivIcon plugin
from folium.features import DivIcon
```

Task 1: Mark all launch sites on a map

First, let's try to add each site's location on a map using site's latitude and longitude coordinates

The following dataset with the name spacex_launch_geo.csv is an augmented dataset with latitude and longitude added for each site.

In [4]:

```
# Download and read the `spacex_launch_geo.csv`
spacex_csv_file = wget.download('https://cf-courses-
data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-
SkillsNetwork/datasets/spacex_launch_geo.csv')
spacex_df=pd.read_csv(spacex_csv_file)
```

Now, you can take a look at what are the coordinates for each site.

In [5]:

```
# Select relevant sub-columns: `Launch Site`, `Lat(Latitude)`,
`Long(Longitude)`, `class`
spacex_df = spacex_df[['Launch Site', 'Lat', 'Long', 'class']]
launch_sites_df = spacex_df.groupby(['Launch Site'],
as_index=False).first()
launch_sites_df = launch_sites_df[['Launch Site', 'Lat', 'Long']]
launch_sites_df
```

Out[5]:

Table 21: Mark all launch sites on a 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

Above coordinates are just plain numbers that cannot give you any intuitive insights about where those are launch sites. If you are very good at geography, you can interpret those numbers directly in your mind. If not, that's fine too. Let's visualize those locations by pinning them on a map.

We first need to create a folium Map object, with an initial centre location to be NASA Johnson Space Centre at Houston, Texas.

In [6]:

```
# Start location is NASA Johnson Space Center
nasa_coordinate = [29.559684888503615, -95.0830971930759]
site_map = folium.Map(location=nasa_coordinate, zoom_start=10)
```

We could use folium.Circle to add a highlighted circle area with a text label on a specific coordinate. For example,

In [7]:

```
# Create a blue circle at NASA Johnson Space Center's coordinate
with a popup label showing its name

# Circle should be blue so changing color code from #d35400 to
#2572e6
circle = folium.Circle(nasa_coordinate, radius=1000,
color='#2572e6', fill=True).add_child(folium.Popup('NASA Johnson
Space Center'))
# Create a blue circle at NASA Johnson Space Center's coordinate
with a icon showing its name
marker = folium.map.Marker(
    nasa_coordinate,
    # Create an icon as a text label
    icon=DivIcon(
        icon_size=(20,20),
        icon_anchor=(0,0),
        html='<div style="font-size: 12;
color:#d35400;"><b>%s</b></div>' % 'NASA JSC',
        )
    )
site_map.add_child(circle)
site_map.add_child(marker)
```

Out[7]:

Make this Notebook Trusted to load map: File -> Trust Notebook

and you should find a small yellow circle near the city of Houston, and you can zoom-in to see a larger circle.

Now, let's add a circle for each launch site in data frame **launch_sites**

TODO: Create and add **folium.Circle** and **folium.Marker** for each launch site on the site map

Converting the dataframe to dictionary:

In [8]:

launch_sites_dict = launch_sites_df.set_index('Launch Site').T.to_dict('list')

```
launch_sites_dict
```

Out[8]:

```
{'CCAFS LC-40': [28.56230196799018, -80.57735647504778],
 'CCAFS SLC-40': [28.563197177407144, -80.57682003124195],
 'KSC LC-39A': [28.57325457037815, -80.64689528960382],
 'VAFB SLC-4E': [34.632834161782775, -120.61074553068327]}
```

My Own Custom Functions for Markers:

In [9]:

```
def add_marker_on_map(name, coordinate, circle_color='#d35400'):
    circle = folium.Circle(coordinate, radius=1000,
color=circle_color, fill=True).add_child(folium.Popup(name))
    marker = folium.map.Marker(
        coordinate,
        icon=DivIcon(
        icon_size=(20,20),
        icon_anchor=(0,0),
        html="<div style='font-size: 12;
color:{};'><b>{}</b></div>". format(circle_color,name)
        )
    )
    site_map.add_child(circle)
    site_map.add_child(marker)

def add_text_distance_marker_on_map(name, coordinate, distance,
ccolor='#4d0f1d'):
    txt="{}, {:0.2f} km".format(name, distance)
    folium.Marker(
```

```
    location=coordinate,
    icon=DivIcon(
            icon_size=(150,30),
            icon_anchor=(0,0),
            html='<div style="font-size: 9pt; font-weight:bold;
color: {}">{}</div>'.format(ccolor, txt))
    ).add_to(site_map)

def add_type_marker_on_map(name, coordinate, ccolor='red',
icon_type='train'):
    marker = folium.map.Marker(
        location=coordinate,
        popup=name,
        tooltip='click',
        icon=folium.Icon(color=ccolor, icon=icon_type, prefix='fa')
    )
    site_map.add_child(marker)

def add_type_distance_marker_on_map(name, coordinate, distance,
ccolor='red', icon_type='bolt'):
    text = "{} {:0.2f}km".format(name, distance)
    marker = folium.map.Marker(
        location=coordinate,
        popup=text,
        icon=folium.Icon(color=ccolor, icon=icon_type, prefix='fa')
    )
    site_map.add_child(marker)

def add_distance_marker_on_map(name, coordinate, distance,
ccolor='#8D208B'):
    text = "{} {:0.2f}km".format(name, distance)
    marker = folium.map.Marker(
        location=coordinate,
        tooltip=text,
        icon=plugins.BeautifyIcon(
            number=round(distance,0),
            border_color=ccolor,
            border_width=2,
            text_color=ccolor,
            inner_icon_style='margin-top:0px;')
```

```
    )
    site_map.add_child(marker)

def draw_distance_lines(place1, place2, lcolor='blue', icolor='red',
tcolor='#4d0f1d'):
    add_type_marker_on_map(place2['name'], place2['coordinates'],
ccolor=icolor, icon_type=place2['placetype'])
    points=[]
    points.append(place1['coordinates'])
    points.append(place2['coordinates'])
    distance = calculate_distance(place1['coordinates'][0],
place1['coordinates'][1], place2['coordinates'][0],
place2['coordinates'][1])
    add_text_distance_marker_on_map(place2['name'],
place2['coordinates'], distance, tcolor)
    linetext = "{}{:0.2f} km".format(place2['name'], distance)
    # folium.plugins.AntPath(points, color=lcolor, weight=1.5, delay
= .5, popup=linetext).add_to(site_map)
    folium.PolyLine(points, color=lcolor, weight=1.5,
popup=linetext).add_to(site_map)

def draw_equater_line():
    points=[]
    points.append([0,195])
    points.append([0,-450])
    folium.plugins.AntPath(points, color='green', weight=1.5, delay
= .5).add_to(site_map)

def add_marker_cluster_on_map(name, coordinate, color):
    folium.Marker(
        location=coordinate,
        popup=name,
        icon=folium.Icon(color='white', icon_color=color),
    ).add_to(marker_cluster)
```

In [10]:

```
# Initial the map
# site_map = folium.Map(location=nasa_coordinate, zoom_start=5)
# Zooming on a launch site 'VAFB SLC-4E' rather than NASA for better
visibiliy of work mentioned in below tasks.
VAFB_SLC_4E = [34.632834, -120.610746]
```

```
site_map = folium.Map(location=VAFB_SLC_4E, zoom_start=3)
# CCAFS_SLC_40 = [28.563197177407144, -80.57682003124195]
# site_map = folium.Map(location=CCAFS_SLC_40, zoom_start=5)
# For each launch site, add a Circle object based on its coordinate
(Lat, Long) values. In addition, add Launch site name as a popup
label
for name,coordinate in launch_sites_dict.items():
    add_marker_on_map(name, coordinate)
site_map
```

Out[10]:

Make this Notebook Trusted to load map: File -> Trust Notebook

The generated map with marked launch sites should look like the following:

Figure 18: Mark all launch sites on a map

Now, you can explore the map by zoom-in/out the marked areas, and try to answer the following questions:

- Are all launch sites in proximity to the Equator line?
- Are all launch sites in very close proximity to the coast?

Also please try to explain your findings.

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

Next, let's try to enhance the map by adding the launch outcomes for each site, and see which sites have high success rates. Recall that data frame spacex_df has detailed launch records, and the class column indicates if this launch was successful or not

In [11]:

```
spacex_df.tail(10)
```

Out[11]:

Table 22: Mark the success/failed launches for each site on the map

	Launch Site	Lat	Long	class
46	KSC LC-39A	28.573255	-80.646895	1
47	KSC LC-39A	28.573255	-80.646895	1
48	KSC LC-39A	28.573255	-80.646895	1
49	CCAFS SLC-40	28.563197	-80.576820	1
50	CCAFS SLC-40	28.563197	-80.576820	1
51	CCAFS SLC-40	28.563197	-80.576820	0
52	CCAFS SLC-40	28.563197	-80.576820	0
53	CCAFS SLC-40	28.563197	-80.576820	0
54	CCAFS SLC-40	28.563197	-80.576820	1
55	CCAFS SLC-40	28.563197	-80.576820	0

Next, let's create markers for all launch records. If a launch was successful (class=1), then we use a green marker and if a launch was failed, we use a red marker (class=0)

Note that a launch only happens in one of the four launch sites, which means many launch records will have the exact same coordinate. Marker clusters can be a good way to simplify a map containing many markers having the same coordinate.

Let's first create a MarkerCluster object

In [12]:

```
marker_cluster = MarkerCluster()
```

TODO: Create a new column in launch_sites dataframe called marker_color to store the marker colors based on the class value

In [13]:

```
# 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
```

In [14]:

```
# Function to assign color to launch outcome
def assign_marker_color(launch_outcome):
    if launch_outcome == 1:
        return 'green'
    else:
        return 'red'

spacex_df['marker_color'] =
spacex_df['class'].apply(assign_marker_color)
spacex_df.tail(10)
```

Out[14]:

Table 23: Mark the success/failed launches for each site on the map

	Launch Site	Lat	Long	class	marker_color
46	KSC LC-39A	28.573255	-80.646895	1	green
47	KSC LC-39A	28.573255	-80.646895	1	green
48	KSC LC-39A	28.573255	-80.646895	1	green

	Launch Site	Lat	Long	class	marker_color
49	CCAFS SLC-40	28.563197	-80.576820	1	green
50	CCAFS SLC-40	28.563197	-80.576820	1	green
51	CCAFS SLC-40	28.563197	-80.576820	0	red
52	CCAFS SLC-40	28.563197	-80.576820	0	red
53	CCAFS SLC-40	28.563197	-80.576820	0	red
54	CCAFS SLC-40	28.563197	-80.576820	1	green
55	CCAFS SLC-40	28.563197	-80.576820	0	red

TODO: For each launch result in **spacex_df** data frame, add a **folium.Marker** to **marker_cluster**

In [15]:

```
launch_coordinates=spacex_df[['Lat','Long']]
color_list=spacex_df['marker_color'].values.tolist()
name_list=spacex_df['Launch Site'].values.tolist()
coordinate_list=launch_coordinates.values.tolist()

# for each row in spacex_df data frame
# create a Marker object with its coordinate
# and customize the Marker's icon property to indicate if this
launch was successed or failed, e.g.,
icon=folium.Icon(color='white', icon_color=row['marker_color']
for i in range(len(coordinate_list)):
    name = name_list[i]
    coordinate = coordinate_list[i]
    color = color_list[i]
    add_marker_cluster_on_map(name, coordinate, color)

# Add the Marker cluster to the site map
site_map.add_child(marker_cluster)
```

Out[15]:

Make this Notebook Trusted to load map: File -> Trust Notebook

Your updated map may look like the following screenshots:

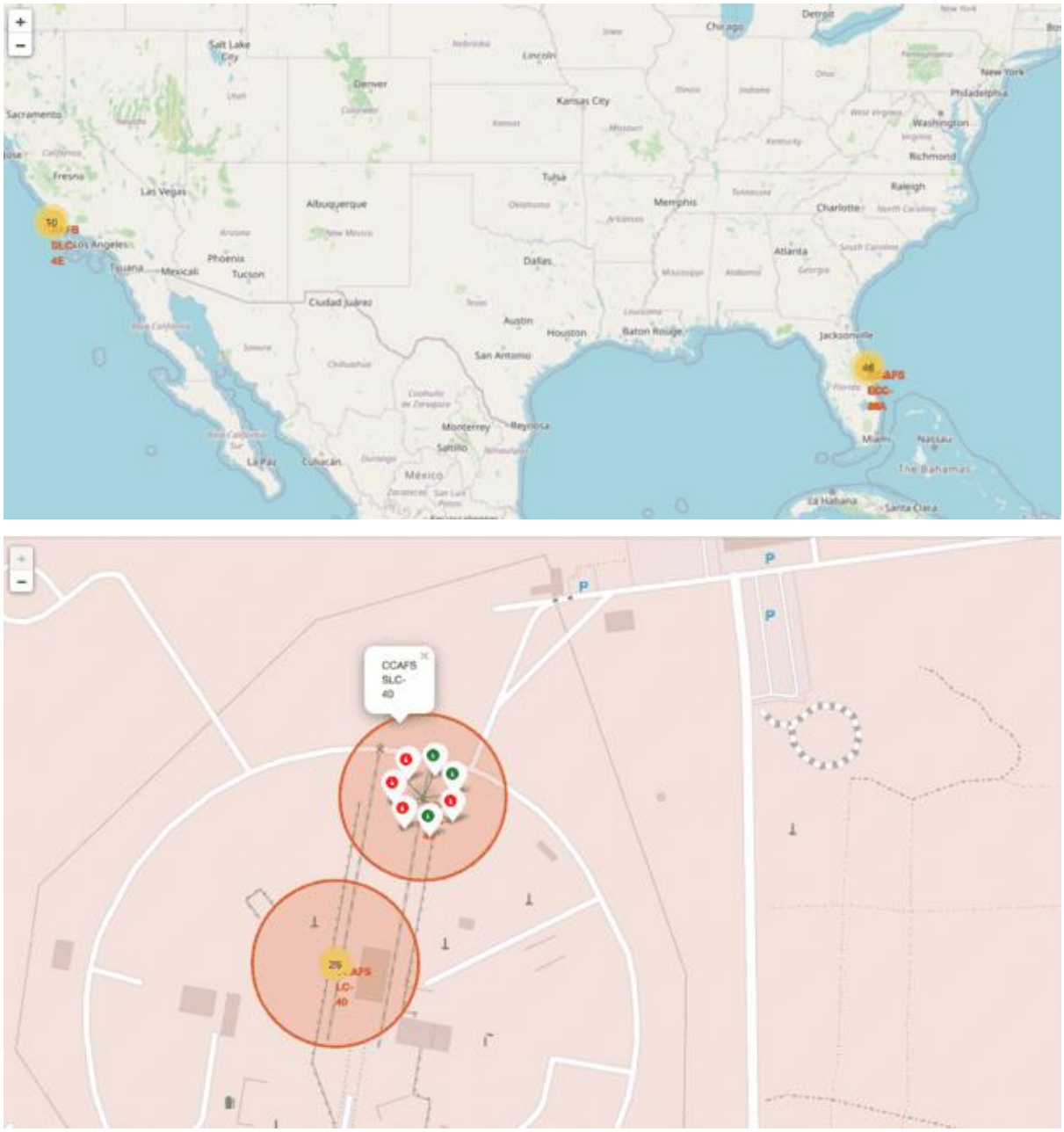


Figure 19: Mark the success/failed launches for each site on the map

From the color-labeled markers in marker clusters, you should be able to easily identify which launch sites have relatively high success rates.

TASK 3: Calculate the distances between a launch site to its proximities

Next, we need to explore and analyze the proximities of launch sites.

Let's first add a **MousePosition** on the map to get coordinate for a mouse over a point on the map. As such, while you are exploring the map, you can easily find the coordinates of any points of interests (such as railway)

In [16]:

```
# Add Mouse Position to get the coordinate (Lat, Long) for a mouse
over on the map
formatter = "function(num) {return L.Util.formatNum(num, 5);};"
mouse_position = MousePosition(
    position='topright',
    separator=' Long: ',
    empty_string='NaN',
    lng_first=False,
    num_digits=20,
    prefix='Lat:',
    lat_formatter=formatter,
    lng_formatter=formatter,
)

site_map.add_child(mouse_position)
site_map
```

Out[16]:

Make this Notebook Trusted to load map: File -> Trust Notebook

Now zoom in to a launch site and explore its proximity to see if you can easily find any railway, highway, coastline, etc. Move your mouse to these points and mark down their coordinates (shown on the top-left) in order to the distance to the launch site.

You can calculate the distance between two points on the map based on their Lat and Long values using the following method:

In [17]:

```
from math import sin, cos, sqrt, atan2, radians

def calculate_distance(lat1, lon1, lat2, lon2):
    # approximate radius of earth in km
    R = 6373.0

    lat1 = radians(lat1)
    lon1 = radians(lon1)
    lat2 = radians(lat2)
    lon2 = radians(lon2)
```

```
    dlon = lon2 - lon1
    dlat = lat2 - lat1

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

    distance = R * c
    return distance
```

TODO: Mark down a point on the closest railway using MousePosition and calculate the distance between the railway point to the launch site.

In [18]:

```
spacex_df[spacex_df['Launch Site'] == 'VAFB SLC-4E']
```

Out[18]:

Table 24: Calculate the distances between a launch site to its proximities

	Launch Site	Lat	Long	class	marker_color
26	VAFB SLC-4E	34.632834	-120.610746	0	red
27	VAFB SLC-4E	34.632834	-120.610746	0	red
28	VAFB SLC-4E	34.632834	-120.610746	1	green
29	VAFB SLC-4E	34.632834	-120.610746	1	green
30	VAFB SLC-4E	34.632834	-120.610746	1	green
31	VAFB SLC-4E	34.632834	-120.610746	1	green
32	VAFB SLC-4E	34.632834	-120.610746	0	red
33	VAFB SLC-4E	34.632834	-120.610746	0	red
34	VAFB SLC-4E	34.632834	-120.610746	0	red

	Launch Site	Lat	Long	class	marker_color
35	VAFB SLC-4E	34.632834	-120.610746	0	red

In [19]:

```
# distance_railway = calculate_distance(lat1, lon1, lat2, lon2)
# Launch Site -
# VAFB SLC-4E 34.632834 -120.610746

# Railway Station -
# Guadalupe Amtrak Station/Coordinates
# 34.9629 -120.5733

# guadalupe_amtrak = [34.9629, -120.5733]
# VAFB_SLC_4E = [34.632834, -120.610746]

# distance_railway = calculate_distance(guadalupe_amtrak[0],
guadalupe_amtrak[1], VAFB_SLC_4E[0], VAFB_SLC_4E[1])
# distance_railway
```

TODO: After obtained its coordinate, create a folium.Marker to show the distance

In [20]:

```
# create and add a folium.Marker on your selected closest raiwaly
point on the map
# show the distance to the launch site using the icon property
# railway_name = "Guadalupe Amtrak Station, {:0.2f}
km".format(distance_railway)
# add_type_marker_on_map(railway_name, guadalupe_amtrak,
ccolor='green', icon_type='train')
# railway_name = "Guadalupe Amtrak Station"
# add_distance_marker_on_map(railway_name, guadalupe_amtrak,
distance_railway)
# add_type_distance_marker_on_map(railway_name, guadalupe_amtrak,
distance_railway, ccolor='green', icon_type='train')
```

TODO: Draw a PolyLine between a launch site to the selected

In [21]:

```
# Create a `folium.PolyLine` object using the railway point
coordinate and launch site coordinate
# points=[]
```

```
# points.append(VAFB_SLC_4E)
# points.append(guadalupe_amtrak)
# linetext = "{}, {:0.2f} km".format(railway_name, distance_railway)
# folium.PolyLine(points, color='blue', weight=2.5,
popup=linetext).add_to(site_map)
# add_text_distance_marker_on_map(railway_name, guadalupe_amtrak,
distance_railway, ccolor='#4d0f1d')
# site_map
draw_equater_line()

launchsite_VAFB = {'name':'VAFB SLC-4E', 'coordinates':[34.632834, -
120.610746], 'placetype':'rocket'}
equator1 = {'name':'Equator Distance from VAFB SLC-4E',
'coordinates':[0, -120.61], 'placetype':'map'}
draw_distance_lines(launchsite_VAFB, equator1,'green', 'green',
'#325931')

launchsite_KSC = {'name':'KSC LC-39A', 'coordinates':[28.573255, -
80.646895], 'placetype':'rocket'}
equator2 = {'name':'Equator Distance from KSC LC-39A',
'coordinates':[0, -80.64651], 'placetype':'map'}
draw_distance_lines(launchsite_KSC, equator2, 'green', 'green',
'#325931')

site_map
```

Out[21]:

Make this Notebook Trusted to load map: File -> Trust Notebook

Your updated map with distance line should look like the following screenshot:

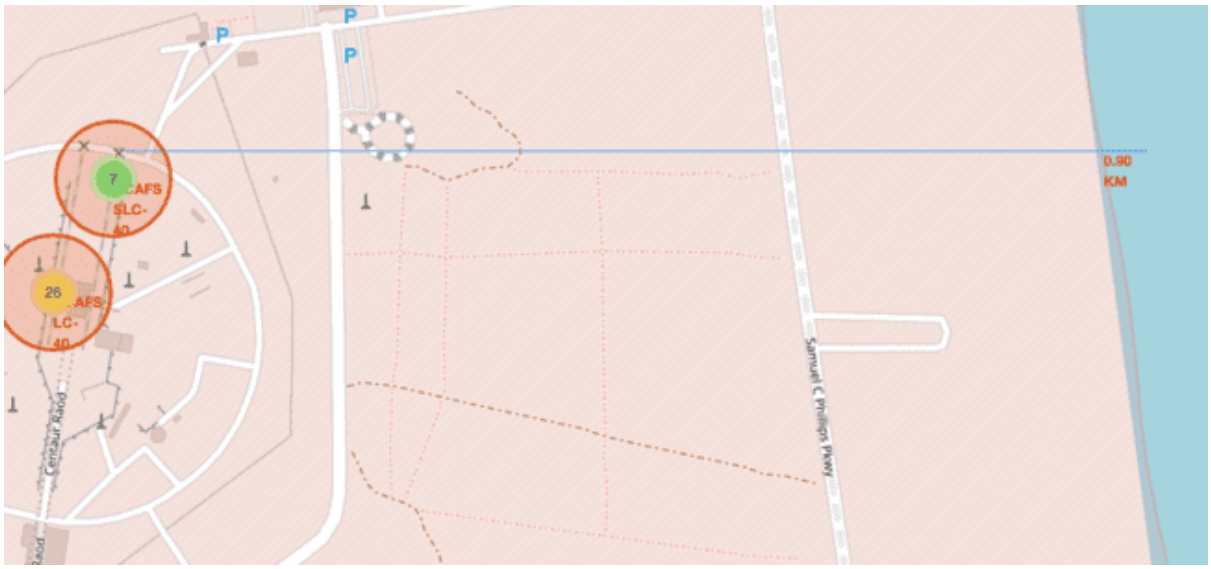


Figure 20: Calculate the distances between a launch site to its proximities

TODO: Similarly, you can draw a line betwee a launch site to its closest city, coastline, highway, etc.

In [22]:

```
# Create a marker with distance to a closest city, coastline,
highway, etc.
# Draw a line between the marker to the launch site
launchsite_VAFB_railway = {'name':'Railway -',
'coordinates':[34.63141, -120.62568], 'placetype':'train'}
draw_distance_lines(launchsite_VAFB, launchsite_VAFB_railway,
'darkgreen', 'green', '#325931')

launchsite_VAFB_highway = {'name':'Highway -',
'coordinates':[34.66992, -120.45753], 'placetype':'road'}
draw_distance_lines(launchsite_VAFB,
launchsite_VAFB_highway,'brown', 'darkred', '#823604')

launchsite_VAFB_coastline = {'name':'Coastline -',
'coordinates':[34.6336, -120.62606], 'placetype':'life-ring'}
draw_distance_lines(launchsite_VAFB,
launchsite_VAFB_coastline,'darkblue', 'darkblue', '#00237d')

launchsite_VAFB_city = {'name':'City -', 'coordinates':[34.63658, -
120.4542], 'placetype':'building'}
draw_distance_lines(launchsite_VAFB, launchsite_VAFB_city,'orange',
'orange', '#995700')
```

```
site_map
```

Out[22]:

Make this Notebook Trusted to load map: File -> Trust Notebook

In [23]:

```
launchsite_KSC_railway = {'name':'Railway -',
'coordinates':[28.573255, -80.65411], 'placetype':'train'}
draw_distance_lines(launchsite_KSC, launchsite_KSC_railway,
'darkgreen', 'green', '#325931')

launchsite_KSC_highway = {'name':'Highway -',
'coordinates':[28.573255, -80.646895], 'placetype':'road'}
draw_distance_lines(launchsite_KSC, launchsite_KSC_highway, 'brown',
'darkred', '#823604')

launchsite_KSC_coastline = {'name':'Coastline -',
'coordinates':[28.573255, -80.60669], 'placetype':'life-ring'}
draw_distance_lines(launchsite_KSC, launchsite_KSC_coastline,
'darkblue', 'darkblue', '#00237d')

launchsite_KSC_city = {'name':'City -', 'coordinates':[28.53748, -
81.38672], 'placetype':'building'}
draw_distance_lines(launchsite_KSC, launchsite_KSC_city, 'orange',
'orange', '#995700')

site_map
```

Out[23]:

Make this Notebook Trusted to load map: File -> Trust Notebook

In [24]:

```
launchsite_CCAFS_SLC = {'name':'CCAFS SLC-40',
'coordinates':[28.563197, -80.576820], 'placetype':'rocket'}

launchsite_CCAFS_SLC_railway = {'name':'Railway -',
'coordinates':[28.57367, -80.58472], 'placetype':'train'}
launchsite_CCAFS_SLC_highway = {'name':'Highway -',
'coordinates':[28.563197, -80.64651], 'placetype':'road'}
launchsite_CCAFS_SLC_coastline = {'name':'Coastline -',
'coordinates':[28.563197, -80.56772], 'placetype':'life-ring'}
launchsite_CCAFS_SLC_city = {'name':'City -',
'coordinates':[28.66288, -81.35925], 'placetype':'building'}
```

```
draw_distance_lines(launchsite_CCAFS_SLC,
launchsite_CCAFS_SLC_railway, 'darkgreen', 'green', '#325931')
draw_distance_lines(launchsite_CCAFS_SLC,
launchsite_CCAFS_SLC_highway, 'brown', 'darkred', '#823604')
draw_distance_lines(launchsite_CCAFS_SLC,
launchsite_CCAFS_SLC_coastline, 'darkblue', 'darkblue', '#00237d')
draw_distance_lines(launchsite_CCAFS_SLC, launchsite_CCAFS_SLC_city,
'orange', 'orange', '#995700')

site_map
```

Out[24]:

Make this Notebook Trusted to load map: File -> Trust Notebook

After you plot distance lines to the proximities, you can answer the following questions easily:

- Are launch sites in close proximity to railways?
- Are launch sites in close proximity to highways?
- Are launch sites in close proximity to coastline?
- Do launch sites keep certain distance away from cities?

Also please try to explain your findings.

Next Steps:

Now you have discovered many interesting insights related to the launch sites' location using folium, in a very interactive way. Next, you will need to build a dashboard using Ploty Dash on detailed launch records.

Authors

[Yan Luo]

Other Contributors

Joseph Santarcangelo

6.2. Plotly:

```
'''
        Build a Dashboard Application with Plotly Dash
        ----------------------------------------------
        In this lab, you will be building a Plotly Dash
        application for users to perform interactive visual analytics
        on SpaceX launch data in real-time.
        '''
        import dash
        import dash_html_components as html
        import dash_core_components as dcc
        from dash.dependencies import Input, Output
        import pandas as pd
        import plotly.express as px

        external_stylesheets = ['https://codepen.io/chriddyp/pen/bWLwgP.css']
        app = dash.Dash(__name__, external_stylesheets=external_stylesheets)

        spacex_df = pd.read_csv("spacex_launch_dash.csv")
        max_payload = spacex_df['Payload Mass (kg)'].max()
        min_payload = spacex_df['Payload Mass (kg)'].min()

        unique_launch_sites = spacex_df['Launch Site'].unique().tolist()
        launch_sites = []
        launch_sites.append({'label': 'All Sites', 'value': 'All Sites'})
        for launch_site in unique_launch_sites:
            launch_sites.append({'label': launch_site, 'value': launch_site})

        app.layout = html.Div(children=[
            html.Div([
                html.H1('SpaceX Launch Records Dashboard',
                    style={'textAlign': 'center', 'color': '#503D36','font-
        size': 40}),
            ]),

            html.Div([
                # TASK 1: Add a Launch Site Drop-down Input Component
                dcc.Dropdown(
                        id = 'site-dropdown',
                        options = launch_sites,
                        placeholder = 'Select a Launch Site here',
                        searchable = True ,
```

```
                clearable = False,
                value = 'All Sites'
            ),
        # TASK 2: Add a callback function to render success-pie-chart
based on selected site dropdown
        html.Div(dcc.Graph(id='success-pie-chart')),
    ], style={'padding': '0 30px'}),

    html.Div([
        # TASK 3: Add a Range Slider to Select Payload
        html.Div("Payload range (Kg):",
            style={'color': '#503D36','font-size': 20, 'padding': '0
30px', 'margin-left': '11px'}
            ),
        html.Div([
            dcc.RangeSlider(
                id = 'payload_slider',
                min = 0,
                max = 10000,
                step = 1000,
                marks = {
                        0: {'label': '0 Kg', 'style': {'color':
'#77b0b1'}},
                        1000: {'label': '1000 Kg'},
                        2000: {'label': '2000 Kg'},
                        3000: {'label': '3000 Kg'},
                        4000: {'label': '4000 Kg'},
                        5000: {'label': '5000 Kg'},
                        6000: {'label': '6000 Kg'},
                        7000: {'label': '7000 Kg'},
                        8000: {'label': '8000 Kg'},
                        9000: {'label': '9000 Kg'},
                        10000: {'label': '10000 Kg', 'style': {'color':
'#f50'}},
                },
                value = [min_payload,max_payload]
            ),
        ], style={'padding': '40px 30px'}),

        # TASK 4: Add a callback function to render the success-payload-
scatter-chart scatter plot
        html.Div(dcc.Graph(id = 'success-payload-scatter-chart')),
    ]),
```

```
],style={'padding': '0 20px'})

# TASK 2: success-pie-chart callback 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 update_piegraph(site_dropdown):
    if (site_dropdown == 'All Sites' or site_dropdown == 'None'):
        all_sites  = spacex_df[spacex_df['class'] == 1] # All Success
only for all sites.
        fig = px.pie(
                all_sites,
                names = 'Launch Site',
                title = 'Total Success Launches by All Sites',
                hole = .2,
                color_discrete_sequence = px.colors.sequential.RdBu
            )
    else:
        site_specific  = spacex_df.loc[spacex_df['Launch Site'] ==
site_dropdown]
        fig = px.pie(
                site_specific,
                names = 'class',
                title = 'Total Success Launches for Site &#8608;
'+site_dropdown,
                hole = .2
            )
    return fig

# TASK 3, 4: Range Slider or Scatter Chart Callback
@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_scattergraph(site_dropdown,payload_slider):
    if (site_dropdown == 'All Sites' or site_dropdown == 'None'):
```

```
        low, high = payload_slider
        all_sites  = spacex_df
        inrange = (all_sites['Payload Mass (kg)'] > low) &
(all_sites['Payload Mass (kg)'] < high)
        fig = px.scatter(
                all_sites[inrange],
                x = "Payload Mass (kg)",
                y = "class",
                title = 'Correlation Between Payload and Success for All
Sites',
                color="Booster Version Category",
                size='Payload Mass (kg)',
                hover_data=['Payload Mass (kg)']
            )
    else:
        low, high = payload_slider
        site_specific  = spacex_df.loc[spacex_df['Launch Site'] ==
site_dropdown]
        inrange = (site_specific['Payload Mass (kg)'] > low) &
(site_specific['Payload Mass (kg)'] < high)
        fig = px.scatter(
                site_specific[inrange],
                x = "Payload Mass (kg)",
                y = "class",
                title = 'Correlation Between Payload and Success for
Site &#8608; '+site_dropdown,
                color="Booster Version Category",
                size='Payload Mass (kg)',
                hover_data=['Payload Mass (kg)']
            )
    return fig

if __name__ == '__main__':
    app.run_server(debug=True)
```

CHAPTER 7

PREDICTIVE ANALYSIS (CLASSIFICATION)

7.1: Machine Learning Prediction

Space X Falcon 9 First Stage Landing Prediction:

Assignment: Machine Learning Prediction

Estimated time needed: **60** minutes

Space X advertises Falcon 9 rocket launches on its website with a cost of 62 million dollars; other providers cost upward of 165 million dollars each, much of the savings is because Space X can reuse the first stage. Therefore, if we can determine if the first stage will land, we can determine the cost of a launch. This information can be used if an alternate company wants to bid against space X for a rocket launch. In this lab, you will create a machine learning pipeline to predict if the first stage will land given the data from the preceding labs.

Several examples of an unsuccessful landing are shown here:

Most unsuccessful landings are planned. Space X; performs a controlled landing in the oceans.

Objectives:

Perform exploratory Data Analysis and determine Training Labels

- create a column for the class
- Standardize the data
- Split into training data and test data

-Find best Hyperparameter for SVM, Classification Trees and Logistic Regression

- Find the method performs best using test data

Import Libraries and Define Auxiliary Functions:

We will import the following libraries for the lab

In [1]:

```
# Pandas is a software library written for the Python programming
language for data manipulation and analysis.
import pandas as pd
# NumPy is a library for the Python programming language, adding
support for large, multi-dimensional arrays and matrices, along with
a large collection of high-level mathematical functions to operate
on these arrays
import numpy as np
# Matplotlib is a plotting library for python and pyplot gives us a
MatLab like plotting framework. We will use this in our plotter
function to plot data.
import matplotlib.pyplot as plt
#Seaborn is a Python data visualization library based on matplotlib.
It provides a high-level interface for drawing attractive and
informative statistical graphics
import seaborn as sns
# Preprocessing allows us to standarsize our data
from sklearn import preprocessing
# Allows us to split our data into training and testing data
from sklearn.model_selection import train_test_split
# Allows us to test parameters of classification algorithms and find
the best one
from sklearn.model_selection import GridSearchCV
# Logistic Regression classification algorithm
from sklearn.linear_model import LogisticRegression
# Support Vector Machine classification algorithm
from sklearn.svm import SVC
# Decision Tree classification algorithm
from sklearn.tree import DecisionTreeClassifier
# K Nearest Neighbors classification algorithm
from sklearn.neighbors import KNeighborsClassifier
```

This function is to plot the confusion matrix.

In [2]:

```
def plot_confusion_matrix(y,y_predict):
    "this function plots the confusion matrix"
    from sklearn.metrics import confusion_matrix

    cm = confusion_matrix(y, y_predict)
    ax= plt.subplot()
    sns.heatmap(cm, annot=True, ax = ax); #annot=True to annotate
cells
    ax.set_xlabel('Predicted labels')
    ax.set_ylabel('True labels')
    ax.set_title('Confusion Matrix');
    ax.xaxis.set_ticklabels(['did not land', 'land']);
ax.yaxis.set_ticklabels(['did not land', 'landed'])
```

Load the dataframe:

Load the data

In [3]:

```
data = pd.read_csv("https://cf-courses-data.s3.us.cloud-object-
storage.appdomain.cloud/IBM-DS0321EN-
SkillsNetwork/datasets/dataset_part_2.csv")

# If you were unable to complete the previous lab correctly you can
uncomment and load this csv

# data = pd.read_csv('https://cf-courses-data.s3.us.cloud-object-
storage.appdomain.cloud/IBMDeveloperSkillsNetwork-DS0701EN-
SkillsNetwork/api/dataset_part_2.csv')

data.head()
```

Out[3]:

	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	CCAFSSLC40	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	CCAFSSLC40	None None	1	False	False	False	NaN	1.0	0	B0005	-80.577366	28.561857	0
2	3	2013-09-29	Falcon 9	677.000000	ISS	CCAFSSLC40	None None	1	False	False	False	NaN	1.0	0	B0007	-80.577366	28.561857	0

	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins	Reused	Legs	LandingPad	Block	ReusedCount	Serial	Longitude	Latitude	Class
		3-01		0000		LC40	none									73666	857	
3	4	2013-09-29	Falcon9	5000.00000	PO	VAFBSLC4E	FalseOcean	1	False	False	False	NaN	1.0	0	B1003	-120.610829	34.632093	0
4	5	2013-12-03	Falcon9	3170.00000	GTO	CCAFSSLC40	NoneNone	1	False	False	False	NaN	1.0	0	B1004	-80.577366	28.561857	0

In [4]:

```
X = pd.read_csv('https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/datasets/dataset_part_3.csv')

# If you were unable to complete the previous lab correctly you can uncomment and load this csv

# X = pd.read_csv('https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-DS0701EN-SkillsNetwork/api/dataset_part_3.csv')

X.head(100)
```

Out[4]:

	FlightNumber	PayloadMass	Flights	Block	ReusedCount	Orbit_ES-L1	Orbit_GEO	Orbit_GTO	Orbit_HEO	Orbit_ISS	...	Serial_B1058	Serial_B1059	Serial_B1060	Serial_B1062	GridFins_False	GridFins_True	Reused_False	Reused_True	Legs_False	Legs_True
0	1.0	6104.959412	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0

	FlightNumber	PayloadMass	Flights	Block	ReusedCount	Orbit_ES-L1	Orbit_GEO	Orbit_GTO	Orbit_HEO	Orbit_ISS	...	Serial_B1058	Serial_B1059	Serial_B1060	Serial_B1062	GridFins_False	GridFins_True	Reused_False	Reused_True	Legs_False	Legs_True
1	2.0	525.000000	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0
2	3.0	677.000000	1.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	...	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0
3	4.0	500.000000	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0

FlightNumber	PayloadMass	Flights	Block	ReusedCount	Orbit_ES-L1	Orbit_GEO	Orbit_GTO	Orbit_HEO	Orbit_ISS	...	Serial_B1058	Serial_B1059	Serial_B1060	Serial_B1062	GridFins_False	GridFins_True	Reused_False	Reused_True	Legs_False	Legs_True
1	6104.959412	1.0	0	0	0	0	0	0	0	...	0	0	0	0	1	0	0	0	1	0
2	525.000000	1.0	0	0	0	0	0	0	0	...	0	0	0	0	1	0	0	0	1	0
3	677.000000	1.0	0	0	0	0	0	0	0	...	0	0	0	0	1	0	0	0	1	0
4	500.000000	1.0	0	0	0	0	0	0	0	...	0	0	0	0	1	0	0	0	1	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...

FlightNumber	PayloadMass	Flights	Block	ReusedCount	Orbit_ES-L1	Orbit_GEO	Orbit_GTO	Orbit_HEO	Orbit_ISS	...	Serial_B1058	Serial_B1059	Serial_B1060	Serial_B1062	GridFins_False	GridFins_True	Reused_False	Reused_True	Legs_False	Legs_True
85	86.0	154000.0000000	2.0	5.0	2.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	1.0	0.0	0.0	1.0	0.0	1.0	0.0	1.0
86	87.0	154000.0000000	3.0	5.0	2.0	0.0	0.0	0.0	0.0	0.0	...	1.0	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	1.0

	FlightNumber	PayloadMass	Flights	Block	ReusedCount	Orbit_ES-L1	Orbit_GEO	Orbit_GTO	Orbit_HEO	Orbit_ISS	...	Serial_B1058	Serial_B1059	Serial_B1060	Serial_B1062	GridFins_False	GridFins_True	Reused_False	Reused_True	Legs_False	Legs_True
87	88.0	15400.000000	6.0	5.0	5.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	1.0
88	89.0	15400.000000	3.0	5.0	2.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	1.0	0.0	0.0	1.0	0.0	1.0	0.0	1.0

	FlightNumber	PayloadMass	Flights	Block	ReusedCount	Orbit_ES-L1	Orbit_GEO	Orbit_GTO	Orbit_HEO	Orbit_ISS	...	Serial_B1058	Serial_B1059	Serial_B1060	Serial_B1062	GridFins_False	GridFins_True	Reused_False	Reused_True	Legs_False	Legs_True
0	1.0	6104.959412	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0
1	2.0	525.000000	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0
2	3.0	677.000000	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0
3	4.0	500.000000	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0
4	5.0	3170.000000	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0
5	6.0	3325.000000	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0
6	7.0	2296.000000	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0
7	8.0	1316.000000	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0
8	9.0	4535.000000	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0
9	10.0	4428.000000	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0

90 rows × 83 columns

TASK 1

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

In [5]:

```
y = data['Class'].to_numpy()
```

TASK 2

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

In [6]:

```
# students get this
transform = preprocessing.StandardScaler()
```

In [7]:

```
X = transform.fit(X).transform(X)
```

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

TASK 3

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

X_train, X_test, Y_train, Y_test

In [8]:

```
X_train, X_test, Y_train, Y_test = train_test_split(X, y,
test_size=0.2, random_state=2)
```

we can see we only have 18 test samples.

In [9]:

```
Y_test.shape
```

Out[9]:

```
(18,)
```

TASK 4

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

In [10]:

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

```
              'solver':['lbfgs']}
```

In [11]:

```
parameters ={"C":[0.01,0.1,1],'penalty':['l2'], 'solver':['lbfgs']}#
l1 lasso l2 ridge
lr=LogisticRegression()
gs_cv = GridSearchCV(lr, parameters, scoring='accuracy', cv=10)
logreg_cv = gs_cv.fit(X_train, Y_train)
```

We output the GridSearchCV object for logistic regression. We display the best parameters using the data attribute best_params_ and the accuracy on the validation data using the data attribute best_score_.

In [12]:

```
print("Tuned hpyerparameters :(best parameters) ",
logreg_cv.best_params_)
print("Accuracy for logistic regression:", logreg_cv.best_score_)
Tuned hpyerparameters :(best parameters)  {'C': 0.01, 'penalty': 'l2
', 'solver': 'lbfgs'}
Accuracy for logistic regression: 0.8464285714285713
```

TASK 5

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

In [13]:

```
print('Accuracy for logistic regression using the method score:',
logreg_cv.score(X_test, Y_test))
Accuracy for logistic regression using the method score: 0.833333333
3333334
```

Lets look at the confusion matrix:

In [14]:

```
yhat=logreg_cv.predict(X_test)
plot_confusion_matrix(Y_test, yhat)
```

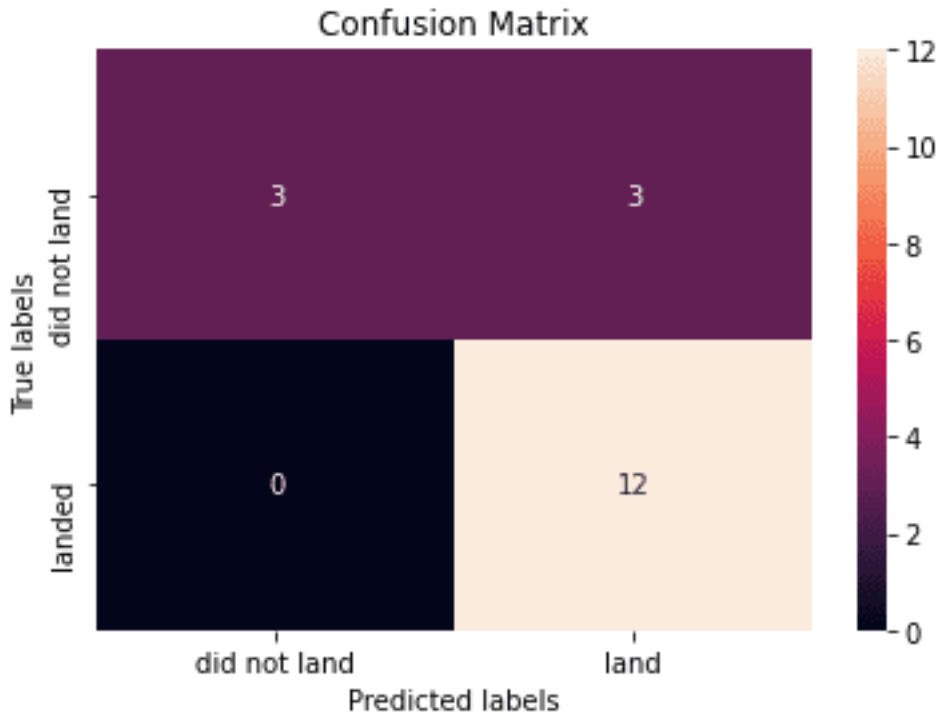


Figure 21: Calculate the accuracy on the test data using the method score:

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

TASK 6

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

In [15]:

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

In [16]:

```
gs_cv = GridSearchCV(svm,parameters, scoring='accuracy', cv=10)
svm_cv = gs_cv.fit(X_train, Y_train)
```

In [17]:

```
print("Tuned hpyerparameters :(best parameters) ",
svm_cv.best_params_)
print("Accuracy for support vector machine:", svm_cv.best_score_)
Tuned hpyerparameters :(best parameters)  {'C': 1.0, 'gamma': 0.0316
2277660168379, 'kernel': 'sigmoid'}
```

```
Accuracy for support vector machine: 0.8482142857142856
```

TASK 7

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

In [18]:

```
print("Accuracy for support vector machine on the test data using
the method score:", svm_cv.score(X_test, Y_test))
Accuracy for support vector machine on the test data using the metho
d score: 0.8333333333333334
```

We can plot the confusion matrix

In [19]:

```
yhat=svm_cv.predict(X_test)
plot_confusion_matrix(Y_test,yhat)
```

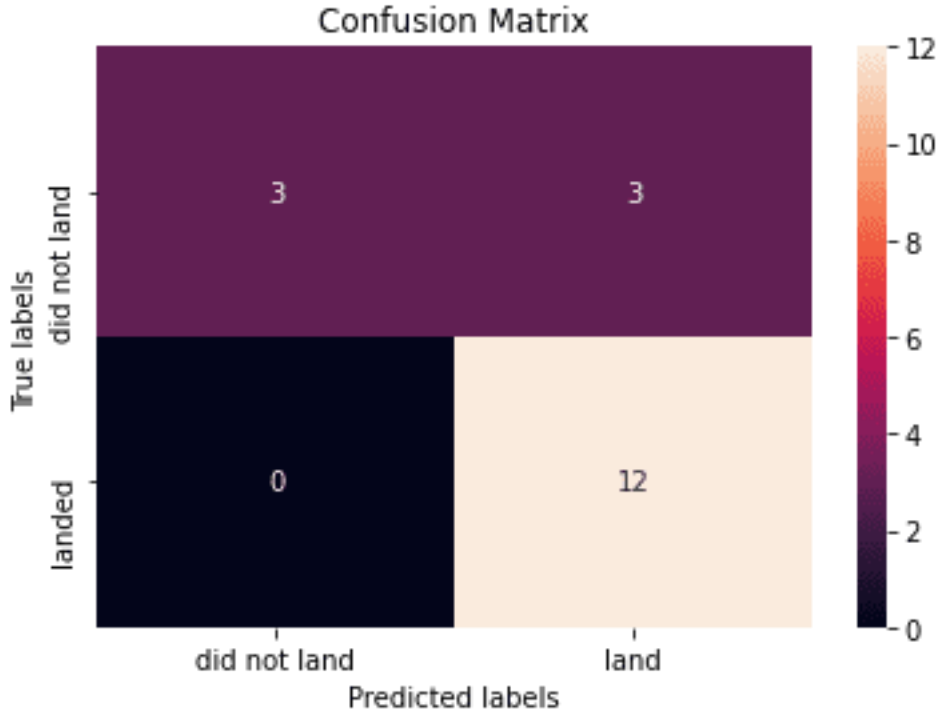


Figure 22: Calculate the accuracy on the test data using the method score:

TASK 8

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

In [20]:

```
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 [21]:

```
gs_cv = GridSearchCV(tree,parameters, scoring='accuracy', cv=10)
tree_cv = gs_cv.fit(X_train, Y_train)
```

In [22]:

```
print("Tuned hpyerparameters :(best parameters) ",
tree_cv.best_params_)
print("Accuracy for decision tree classifier:", tree_cv.best_score_)
Tuned hpyerparameters :(best parameters)  {'criterion': 'gini', 'max
_depth': 10, 'max_features': 'sqrt', 'min_samples_leaf': 1, 'min_sam
ples_split': 2, 'splitter': 'best'}
Accuracy for decision tree classifier: 0.9017857142857144
```

TASK 9

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

In [23]:

```
print("Accuracy for decision tree classifier on the test data using
the method score:", tree_cv.score(X_test, Y_test))
Accuracy for decision tree classifier on the test data using the met
hod score: 0.8333333333333334
```

We can plot the confusion matrix

In [24]:

```
yhat = svm_cv.predict(X_test)
plot_confusion_matrix(Y_test, yhat)
```

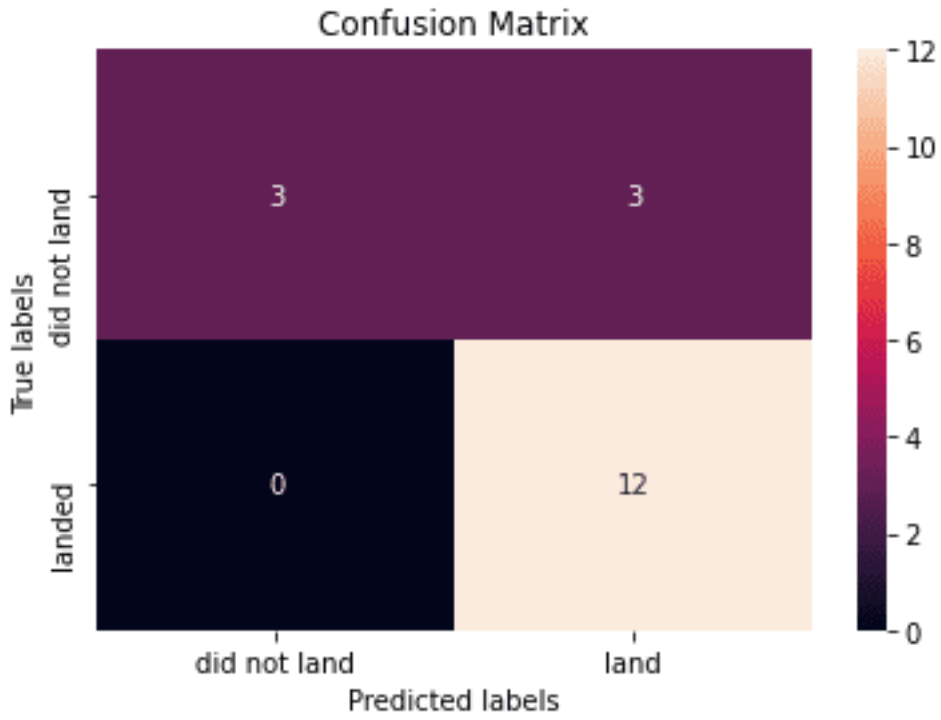


Figure 23: Calculate the accuracy of tree_cv on the test data using the method score:

TASK 10

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

In [25]:

```
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 [26]:

```
gs_cv = GridSearchCV(KNN, parameters, scoring='accuracy', cv=10)
knn_cv = gs_cv.fit(X_train, Y_train)
```

In [27]:

```
print("Tuned hpyerparameters :(best parameters)
",knn_cv.best_params_)
print("Accuracy for k nearest neighbors:",knn_cv.best_score_)
Tuned hpyerparameters :(best parameters)  {'algorithm': 'auto', 'n_n
eighbors': 10, 'p': 1}
Accuracy for k nearest neighbors: 0.8482142857142858
```

TASK 11

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

In [28]:

```
print("Accuracy for k nearest neighbors on the test data using the
method score: ",knn_cv.score(X_test, Y_test))
Accuracy for k nearest neighbors on the test data using the method s
core:  0.8333333333333334
```

We can plot the confusion matrix

In [29]:

```
yhat = knn_cv.predict(X_test)
plot_confusion_matrix(Y_test,yhat)
```

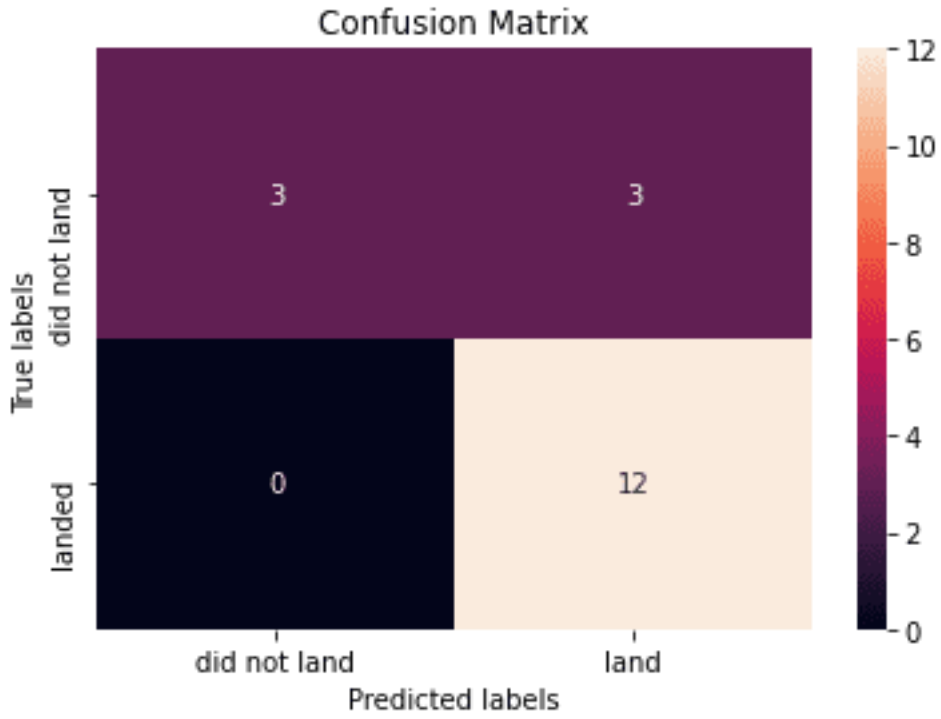


Figure 24: Calculate the accuracy of tree_cv on the test data using the method score:

TASK 12

Find the method performs best:

In [30]:

```
algorithms = {'KNN':knn_cv.best_score_,'Decision
Tree':tree_cv.best_score_,'Logistic
Regression':logreg_cv.best_score_,'SVM':svm_cv.best_score_}
best_algorithm = max(algorithms, key= lambda x: algorithms[x])
```

```
print('The method which performs best is \"',best_algorithm,'\" with
a score of',algorithms[best_algorithm])
The method which performs best is " Decision Tree " with a score of
0.9017857142857144
```

In [31]:

```
algo_df = pd.DataFrame.from_dict(algorithms, orient='index',
columns=['Accuracy'])
```

In [32]:

```
algo_df.head()
```

Out[32]:

Table 25: Find the method performs best:

	Accuracy
KNN	0.848214
Decision Tree	0.901786
Logistic Regression	0.846429
SVM	0.848214

In [33]:

```
algo_df.sort_values(['Accuracy'], inplace=True)
```

In [34]:

```
algo_df.head()
```

Out[34]:

	Accuracy
Logistic Regression	0.846429
SVM	0.848214
KNN	0.848214

	Accuracy
Decision Tree	0.901786

In [35]:

```
algo_df = algo_df.reset_index()
algo_df.head()
```

Out[35]:

	index	Accuracy
0	Logistic Regression	0.846429
1	SVM	0.848214
2	KNN	0.848214
3	Decision Tree	0.901786

In [36]:

```
algo_df.rename(columns = {'index': 'Algorithm'}, inplace = True)
algo_df.head()
```

Out[36]:

	Algorithm	Accuracy
0	Logistic Regression	0.846429
1	SVM	0.848214
2	KNN	0.848214
3	Decision Tree	0.901786

In [37]:

```
import plotly.express as px
import plotly.graph_objects as go
```

In [38]:

```
fig = px.bar(algo_df, x='Algorithm', y='Accuracy',
hover_data=['Algorithm', 'Accuracy'], color='Accuracy',
color_continuous_scale='rdylbu')
fig.update_layout(title='Algorithm vs. Accuracy',
xaxis_title='Algorithm', yaxis_title='Accuracy' )
fig.show()
```

Authors

Joseph Santarcangelo has a PhD in Electrical Engineering, his research focused on using machine learning, signal processing, and computer vision to determine how videos impact human cognition. Joseph has been working for IBM since he completed his PhD.

Change Log

Table 26: Week 4.1 Change Log

Date (YYYY-MM-DD)	Version	Changed By	Change Description
2020-09-20	1.0	Joseph	Modified Multiple Areas

CHAPTER 8

PRESENT YOUR DATA-DRIVEN INSIGHTS

8.1. Present Your Data-Driven Insights with Presentation

SpaceX Industrial Project.docx

Insights with Presentation 1

SpaceX_Industrial_Project_PPT_Slides.pdf

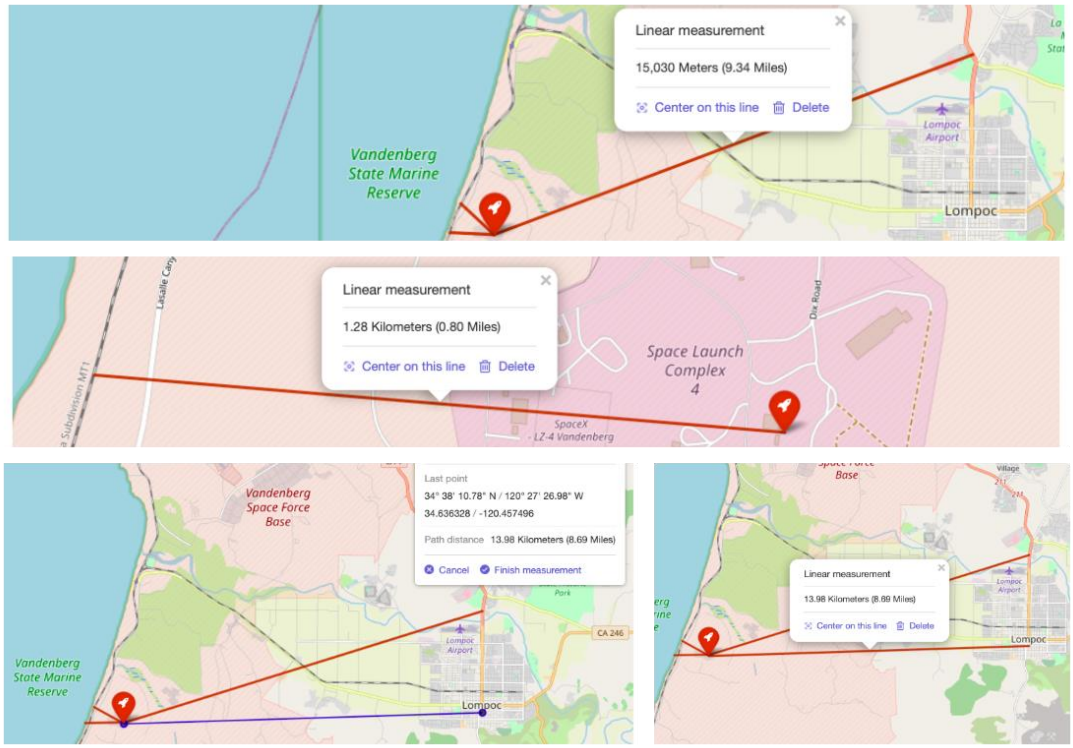
Insights with Presentation 2

SpaceX Industrial Project PPT.pptx

Insights with Presentation 3

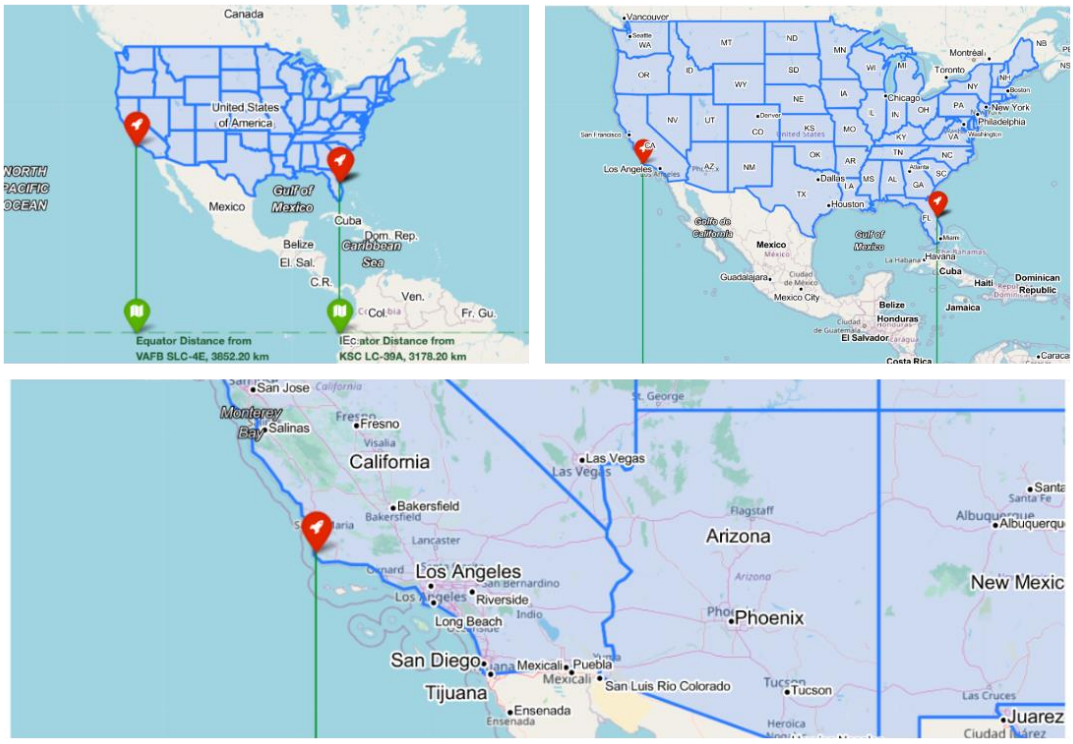
8.2. 🌬️ Out of the Box Thinking:

- Advanced Folium Function [Measurement Control Plugin with Folium]



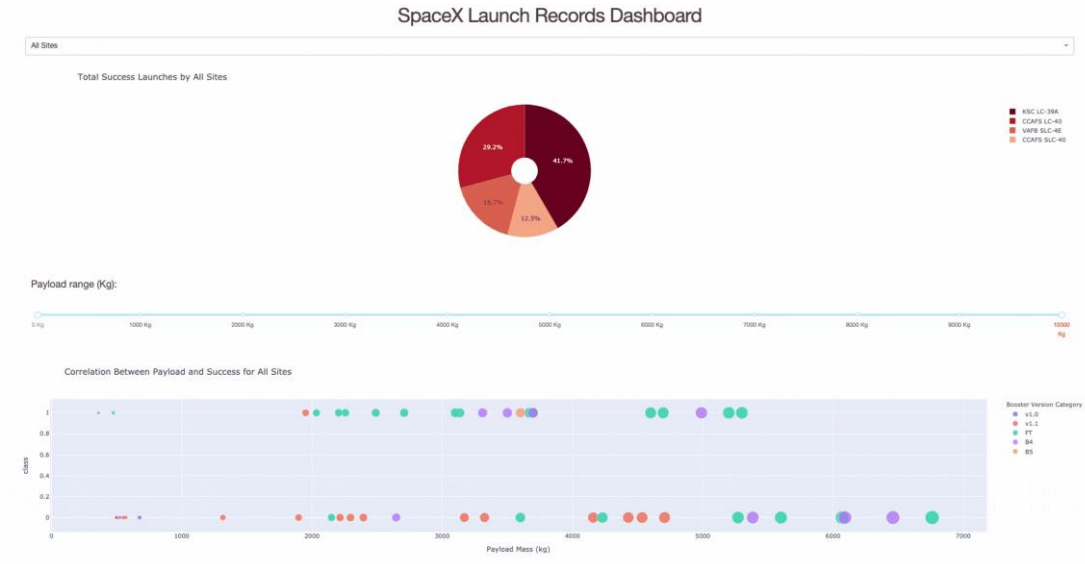
Insights with Presentation 4: measurementcontrol.png

- **Advanced Folium Function [Folium Custom Pane with Labels]**



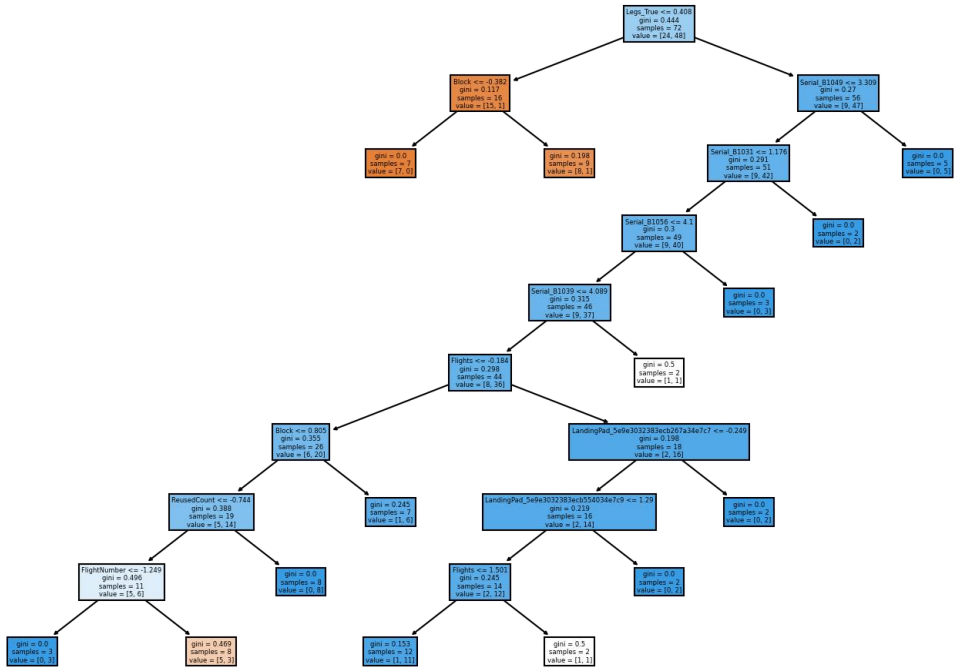
Insights with Presentation 5: custompanes.png

- **[Hosting the Plotly on "Python Anywhere" Server with Flask]**



Insights with Presentation 6: interactive-plotly.gif

- **Machine Learning - Decision Tree Construction**

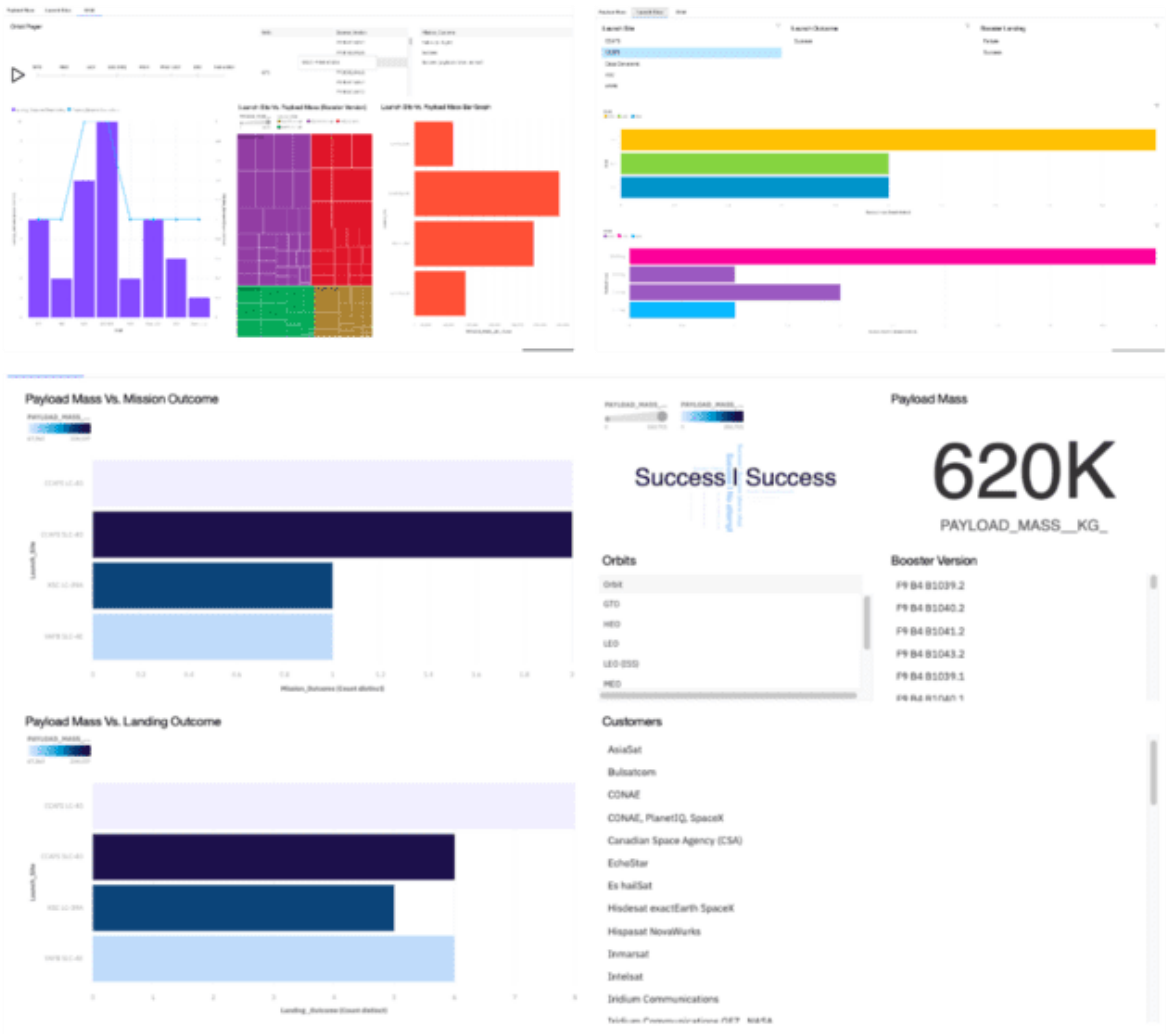


Insights with Presentation 7: decision-tree.png

- **EDA Visualization with Interactive Plotly (Instead of Seaborn)**

Insights with Presentation 8: plotly.png

- **IBM Cognos Visualization with Player**



Insights with Presentation 9: IBM-Cognos.png

8.3 🔨 Built With:

- Python - Programming Language
- IBM Watson Studio - IBM’s software platform for data science
- IBM Db2 - Db2 is a family of data management products, including database servers, developed by IBM
- Jupiter Notebooks - Open-source web application that allows data scientists to create and share documents that integrate live code, equations, computational output, visualizations, and other multimedia resources.
- Anaconda - Local environment for practice
- pythonanywhere.com - Host your Python App via Flask
- plotly.com - Plotly stewards Python's leading data viz and UI libraries. With Dash Open Source, Dash apps run on your local laptop or server
- IBM Cognos Dashboard - IBM Cognos Analytics provides dashboards and stories to communicate your insights and analysis. You can assemble a view that contains visualizations such as a graph, chart, plot, table, map, or any other visual representation of data.
- GitHub - Repository for storing all files

8.4. 📚 Extra Study Materials:

Don’t jump into labs without study. Take time to study previous labs step by step and practice on your own laptop locally by installing Anaconda. This will also limit the usage of IBM Cloud Trial version.

- Plotly
- Folium
- Folium Examples
- Confusion Matrix

REFERENCES

1. Joseph Santarcangelo has a PhD in Electrical Engineering, his research focused on using machine learning, signal processing, and computer vision to determine how videos impact human cognition. Joseph has been working for IBM since he completed his PhD.
2. Yan Luo
3. Nayef Abou Tayoun
4. Joseph Santarcangelo has a PhD in Electrical Engineering, his research focused on using machine learning, signal processing, and computer vision to determine how videos impact human cognition. Joseph has been working for IBM since he completed his PhD.
5. Nayef Abou Tayoun is a Data Scientist at IBM and pursuing a Master of Management in Artificial intelligence degree at Queen's University.
6. Heilbron, J. (2003). The Oxford companion to the history of modern science. Oxford: Oxford University PRESS.
7. Dhar, V. (2013). Data science and prediction. Communications Of The ACM, 56(12), 64-73. http://dx.doi.org/10.1145/2500499
8. Raghupathi, W. & Raghupathi, V. (2014). Big data analytics in healthcare: promise and potential. Health Inf Sci Syst, 2(1), 3. http://dx.doi.org/10.1186/2047-2501-2-3
9. Donoho, D. (2015). 50 years of Data Science. Presentation, Tukey Centennial workshop, Princeton NJ.
10. Wu, X., Kumar, V., Ross Quinlan, J., Ghosh, J., Yang, Q., & Motoda, H. et al. (2007). Top 10 algorithms in data mining. Knowledge And Information Systems, 14(1), 1-37. http://dx.doi.org/10.1007/s10115-007-0114-2
11. S, S., Tamilarasi, A., & Pravin Kumar, M. (2012). Implementation of Genetic Algorithm in Predicting Diabetes. International Journal Of Computer Science, 9(1), 234-240.

12. James H. Faghmous, Arindam Banerjee, Shashi Shekhar, Michael Steinbach, Vipin Kumar, Auroop R. Ganguly, Nagiza Samatova, "Theory-Guided Data Science for Climate Change", Computer, vol.47, no. 11, pp. 74-78, Nov. 2014, doi:10.1109/MC.2014.3
13. Newman, R., Chang, V., Walters, R., & Wills, G. (2016). Model and experimental development for Business Data Science. International Journal Of Information Management, 36(4), 607-617. http://dx.doi.org/10.1016/j.ijinfomgt.2016.04.004
14. Hands-on Lab: String Patterns, Sorting and Grouping
15. Hands-on Lab: Built-in functions
16. Hands-on Lab: Sub-queries and Nested SELECT Statements
17. Hands-on Tutorial: Accessing Databases with SQL magic
18. Hands-on Lab: Analysing a real-World Data Set

SACHIN RATHI

MCA - Computer Application

Ph: +91-8755600635
Email: sachin.1922mca1001@kiet.edu
Meerut, Uttar Pradesh, India - 250001

BRIEF OVERVIEW / CAREER OBJECTIVE / SUMMARY

I want to become a data Scientist.

KEY EXPERTISE / SKILLS

Problem Solving | Algorithms | HTML | CSS | SQL | Python | NumPy | Pandas | C

EDUCATION

KIET Group of Institutions 2019 - 2022
MCA - Computer Application | Percentage: **67.00 / 100.00**

IIMT Engineering College, Meerut 2014 - 2017
BCA - Computer Applications | Percentage : **56.00 / 100.00**

Gyan Deep Public School, Muzaffarnagar 2014
12th | CBSE | Percentage: **49.00 / 100.00**

Gyan Deep Public School, Muzaffarnagar 2012
10th | CBSE | CGPA: **7.20 / 10.00**

PROJECTS

Privlok April 15, 2021 - June 15, 2021
Mentor: Dr. Vipin Kumar | **Team Size:** 4
Key Skills: SQLite | Android Studio | Java
It is a password keeper app which stores data in internally without the use of Cloud.

Indoor Game Nov. 1, 2019 - Dec. 15, 2019
Team Size: 2
Key Skills: HTML | CSS | Javascript
Indoor game website(static).

ACHIEVEMENTS

- 3rd level on 1st sem. (Department level competition)

PERSONAL INTERESTS / HOBBIES

- Badminton
- Singing
- Travelling
- Photography

IMs

- WhatsApp - 8755600635

PERSONAL DETAILS

Gender: Male
Marital Status: Unmarried
Current Address: 192/108z, street no 11, harinagar, Sharadhapuri phase1, Kankerkhera, Meerut, Meerut, Uttar Pradesh, India - 250001
Email: sachin.1922mca1001@kiet.edu

Date of Birth: July 27, 1997
Permanent Address: mainmana, Tikri Rural ,Tikri, Baghpat, Baraut, Uttar Pradesh, India - 250625
Phone Number: +91-8755600635