Space X Rocket Launch Analysis

Sarmilan Sreekaran

May 12, 2024

OUTLINE

- Executive Summary
- Introduction
- Methodology
- Results
- Conclusion

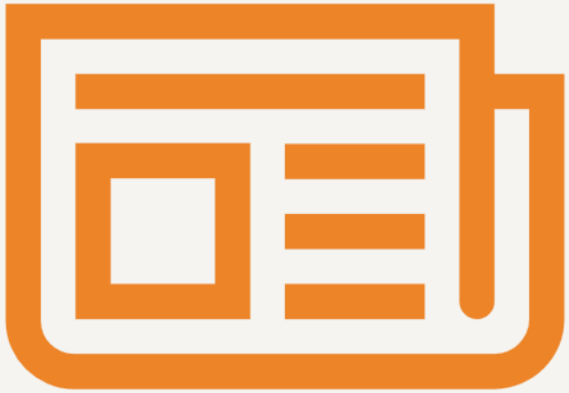
EXECUTIVE SUMMARY

- Methodology
  - Data Collection
  - Data Wrangling
  - Web Scraping
  - EDA with Data Visualization
  - EDA with SQL
  - Interactive Map with Folium
  - Plotly Dash dashboard
  - Predictive Analysis (Classification)
- Results
- Conclusion

INTRODUCTION

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 module, you will be provided with an overview of the problem and the tools you need to complete the course.

METHODOLOGY



- Data Collection
- Data Wrangling
- Web Scraping
- EDA with Data Visualization
- EDA with SQL
- Interactive Map with Folium
- Plotly Dash dashboard
- Predictive Analysis (Classification)

Data Collection

The Data was collected using a get request on SpaceX's API which is used to attain the rocket, launchpad and payload data. Then we make the requested JSON results more consistent by decoding and turn it into a Pandas Data frame

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:

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

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

```
]: response.status_code
```

```
]: 200
```

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

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

SpaceX API

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:

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

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

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

[10]:
```
200
```

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

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

First we use the get request on the SpaceX API, then we decode the response and turn it into a Panda's Dataframe using .json_normalize()

SpaceX API cont.

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

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

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

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

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

From `cores` we would like to learn the outcome of the landing, the type of the landing, number of flig
version of cores, the number of times this specific core has been reused, and the serial of the core.

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

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

[19]: # Call getPayloadData
      getPayloadData(data)

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

- Next we grabbed all the necessary data, the rocket, launchpad and payload data

SpaceX API cont.

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

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

```
# Create a data from launch_dict
data2 = pd.DataFrame.from_dict(launch_dict)
```

Show the summary of the dataframe

```
# Show the head of the dataframe
data2.head()
```

	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins	Reused	Legs	LandingPad	Block	ReusedCount	Serial	Longitude	Latitude
0	1	2006-03-24	Falcon 1	20.0	LEO	Kwajalein Atoll	None None	1	False	False	False	None	NaN	0	Merlin1A	167.743129	9.047721
1	2	2007-03-21	Falcon 1	NaN	LEO	Kwajalein Atoll	None None	1	False	False	False	None	NaN	0	Merlin2A	167.743129	9.047721
2	4	2008-09-28	Falcon 1	165.0	LEO	Kwajalein Atoll	None None	1	False	False	False	None	NaN	0	Merlin2C	167.743129	9.047721
3	5	2009-07-13	Falcon 1	200.0	LEO	Kwajalein Atoll	None None	1	False	False	False	None	NaN	0	Merlin3C	167.743129	9.047721
4	6	2010-06-04	Falcon 9	NaN	LEO	CCSFS SLC 40	None None	1	False	False	False	None	1.0	0	B0003	-80.577366	28.561857

- Here we make a dictionary with the collected data

Data Wrangling

TASK 4: Create a landing outcome label from Outcome column

Using the `Outcome`, create a list where the element is zero if the corresponding rc

[5]:
```
# landing_class = 0 if bad_outcome
landing_class = df['Outcome'].apply(lambda x: 0 if x in bad_outcomes
# landing_class = 1 otherwise
else 1)
```

This variable will represent the classification variable that represents the outcome c

[6]:
```
df['Class']=landing_class
df[['Class']].head(8)
```

[6]:

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

- Once we got the data there was several occurrences where the mission failed so we use data wrangling to make a column separating the successes and loses

Web Scraping

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.

[26]:
```
# use requests.get() method with the provided static_url
# assign the response to a object
data = requests.get(static_url).text
```

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

[28]:
```
# Use BeautifulSoup() to create a BeautifulSoup object from a response text content
soup = BeautifulSoup(data, 'html.parser')
```

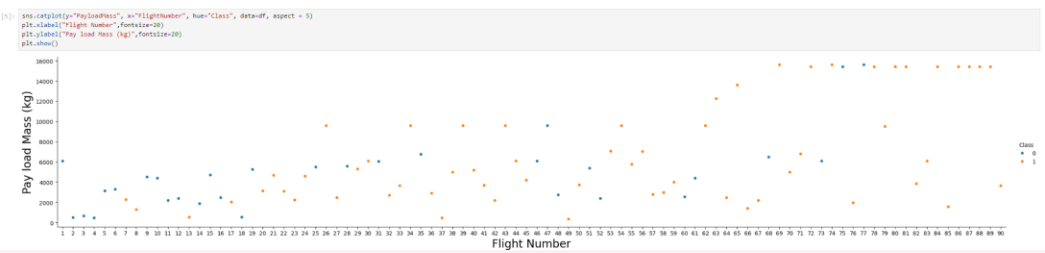
Print the page title to verify if the `BeautifulSoup` object was created properly

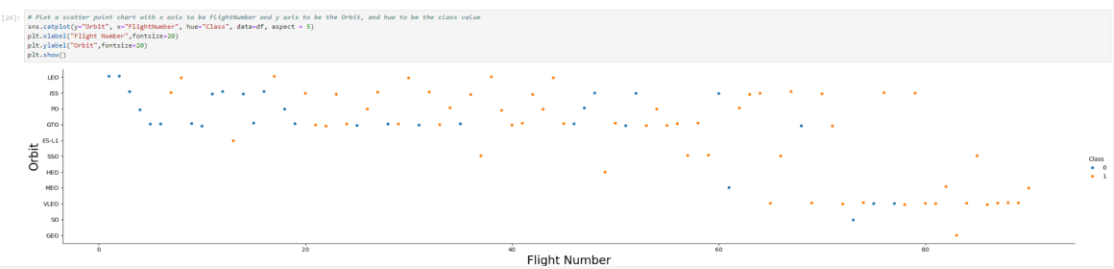
[29]:
```
# Use soup.title attribute
print(soup.title)
```

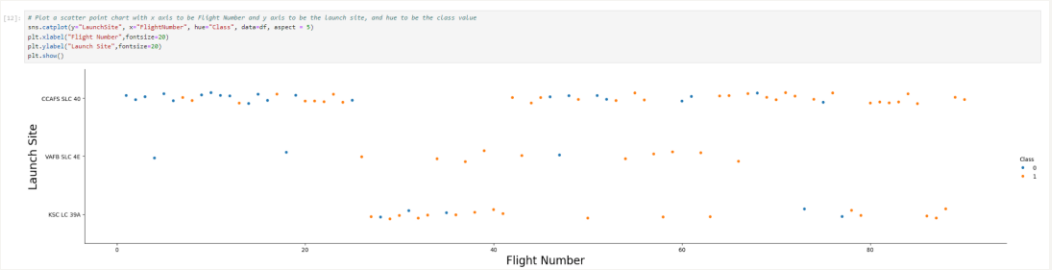
```
<title>List of Falcon 9 and Falcon Heavy launches - Wikipedia</title>
```

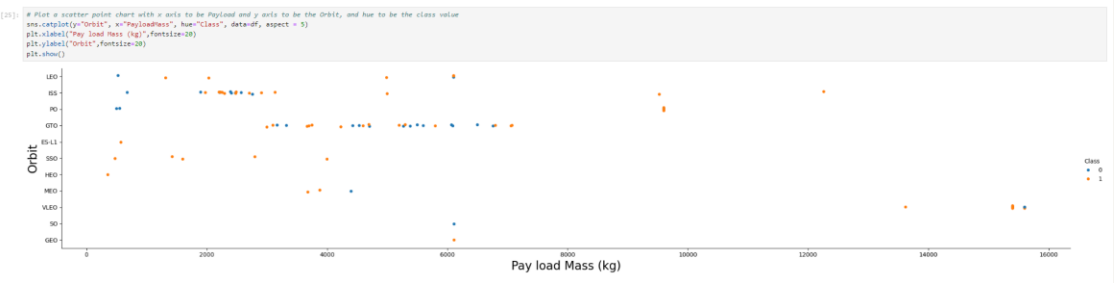
- For Web Scraping we used the html get function and on the static url with the data from their Wikipedia and used Beautiful Soup to make it readable

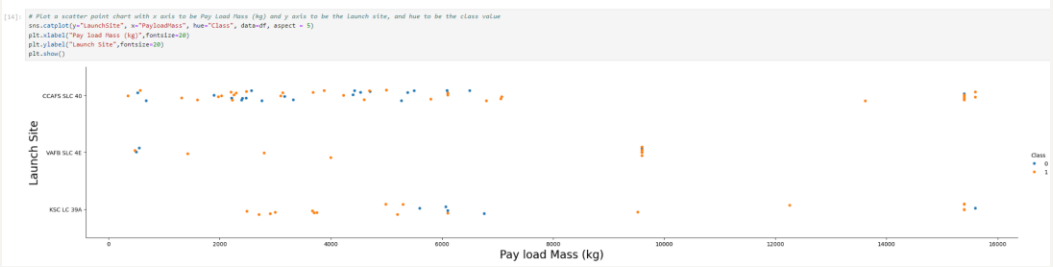
EDA with Data Visualization



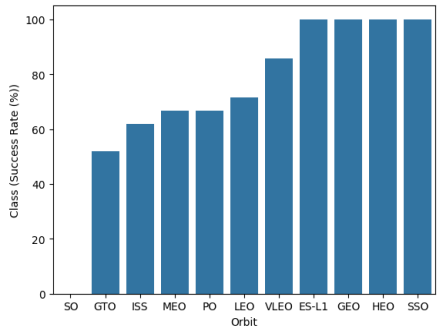




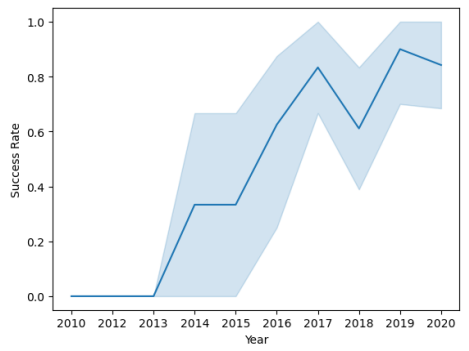





```
# HINT use groupby method on Orbit column and get the mean of Class column
sr_df = df.groupby('Orbit')['Class'].mean().reset_index().sort_values(by='Class')
sr_df['Class'] = sr_df['Class'] * 100
sns.barplot(sr_df, x='Orbit', y='Class')
plt.xlabel('Orbit')
plt.ylabel('Class (Success Rate (%))')
plt.show()
```




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



EDA with Data Visualization cont.

- Scatter Plots were used to visualize the relationship between Flight Number and Pay load Mass, Launch Site and Flight Number , Launch Site and Pay load Mass, Orbit and Flight Number and Orbit and Pay load Mass.
- Bar Chart is used to compare the success rate of the different Orbit types
- Line plot is used to view the yearly trend of the success rate

EDA with SQL

Task 1

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

```
[12]: %sql SELECT DISTINCT LAUNCH_SITE as "Launch_Sites" FROM SPACEXTBL;
```

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

[12]:

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

Task 2

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

```
[13]: %sql SELECT * FROM 'SPACEXTBL' WHERE Launch_Site LIKE 'CCA%' LIMIT 5;
```

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

[13]:

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

Task 3

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

```
[14]: %sql SELECT SUM(PAYLOAD_MASS__KG_) as "Total Payload Mass(Kgs)", Customer FROM 'SPACEXTBL' WHERE Customer = 'NASA (CRS)';
```

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

[14]:

Total Payload Mass(Kgs)	Customer
45596	NASA (CRS)

Task 4

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

```
[15]: %sql SELECT AVG(PAYLOAD_MASS__KG_) as "Average Payload Mass(Kgs)", Customer, Booster_Version FROM 'SPACEXTBL' WHERE Booster_Version LIKE 'F9 v1.1%';
```

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

[15]:

Average Payload Mass(Kgs)	Customer	Booster_Version
2534.6666666666665	MDA	F9 v1.1 B1003

Task 5

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

Hint:Use min function

```
[17]: %sql SELECT MIN(DATE) FROM 'SPACEXTBL' WHERE "Landing_Outcome" = "Success (ground pad)";
```

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

[17]:

MIN(DATE)
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

```
[18]: %sql SELECT DISTINCT Booster_Version, Payload FROM SPACEXTBL WHERE "Landing_Outcome" = "Success (drone ship)" AND PAYLOAD_MASS__KG_ > 4000 AND PAYLOAD_MASS__KG_ < 6000;
```

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

[18]:

Booster_Version	Payload
F9 FT B1022	JCSAT-14
F9 FT B1026	JCSAT-16
F9 FT B1021.2	SES-10
F9 FT B1031.2	SES-11 / EchoStar 105

Task 7

List the total number of successful and failure mission outcomes

```
[19]: %sql SELECT "Mission_Outcome", COUNT("Mission_Outcome") as Total FROM SPACEXTBL GROUP BY "Mission_Outcome";
```

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

[19]:

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

Task 8

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

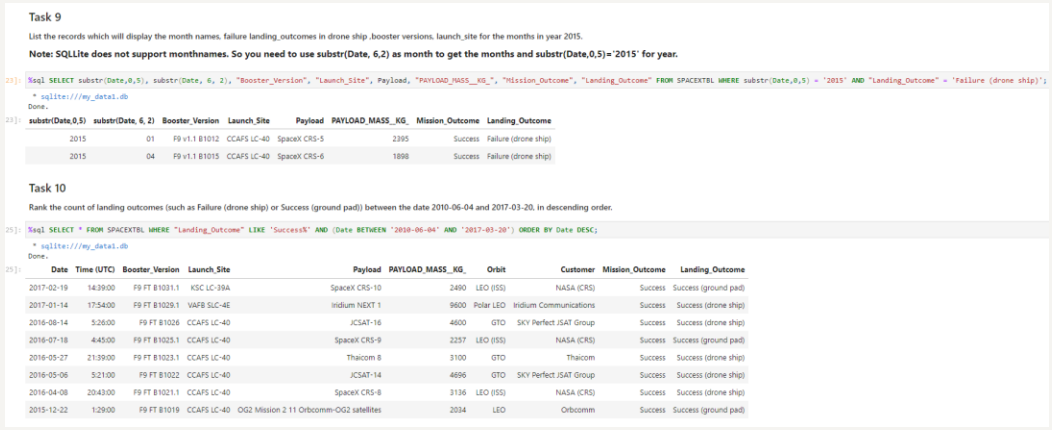
```
[20]: %sql SELECT "Booster_Version", Payload, "PAYLOAD_MASS__KG_" FROM SPACEXTBL WHERE "PAYLOAD_MASS__KG_" = (SELECT MAX("PAYLOAD_MASS__KG_") FROM SPACEXTBL);
```

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

[20]:

Booster_Version	Payload	PAYLOAD_MASS__KG_
F9 B5 B1048.4	Starlink 1 v1.0, SpaceX CRS-19	15600
F9 B5 B1049.4	Starlink 2 v1.0, Crew Dragon in-flight abort test	15600
F9 B5 B1051.3	Starlink 3 v1.0, Starlink 4 v1.0	15600
F9 B5 B1056.4	Starlink 4 v1.0, SpaceX CRS-20	15600
F9 B5 B1048.5	Starlink 5 v1.0, Starlink 6 v1.0	15600
F9 B5 B1051.4	Starlink 6 v1.0, Crew Dragon Demo-2	15600
F9 B5 B1049.5	Starlink 7 v1.0, Starlink 8 v1.0	15600
F9 B5 B1060.2	Starlink 11 v1.0, Starlink 12 v1.0	15600
F9 B5 B1058.3	Starlink 12 v1.0, Starlink 13 v1.0	15600
F9 B5 B1051.6	Starlink 13 v1.0, Starlink 14 v1.0	15600
F9 B5 B1060.3	Starlink 14 v1.0, GPS III-04	15600
F9 B5 B1049.7	Starlink 15 v1.0, SpaceX CRS-21	15600

EDA with SQL cont.



Task 9

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

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

```
%sql SELECT substr(Date,0,5), substr(Date, 6, 2), "Booster_Version", "Launch_Site", Payload, "PAYLOAD_MASS__KG_", "Mission_Outcome", "Landing_Outcome" FROM SPACEXTBL WHERE substr(Date,0,5) = '2015' AND "Landing_Outcome" = 'Failure (drone ship)';
```

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

substr(Date,0,5)	substr(Date, 6, 2)	Booster_Version	Launch_Site	Payload	PAYLOAD_MASS__KG_	Mission_Outcome	Landing_Outcome
2015	01	F9 v1.1 B1012	CCAFS LC-40	SpaceX CRS-5	2395	Success	Failure (drone ship)
2015	04	F9 v1.1 B1015	CCAFS LC-40	SpaceX CRS-6	1898	Success	Failure (drone ship)

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.

```
%sql SELECT * FROM SPACEXTBL WHERE "Landing_Outcome" LIKE 'Success%' AND (Date BETWEEN '2010-06-04' AND '2017-03-20') ORDER BY Date DESC;
```

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

Date	Time (UTC)	Booster_Version	Launch_Site	Payload	PAYLOAD_MASS__KG_	Orbit	Customer	Mission_Outcome	Landing_Outcome
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)
2017-01-14	17:54:00	F9 FT B1029.1	VAFB SLC-4E	Iridium NEXT 1	9600	Polar LEO	Iridium Communications	Success	Success (drone ship)
2016-08-14	5:26:00	F9 FT B1026	CCAFS LC-40	JCSAT-16	4600	GTO	SKY Perfect JSAT Group	Success	Success (drone ship)
2016-07-18	4:45:00	F9 FT B1025.1	CCAFS LC-40	SpaceX CRS-9	2257	LEO (ISS)	NASA (CRS)	Success	Success (ground pad)
2016-05-27	21:39:00	F9 FT B1023.1	CCAFS LC-40	Thaicom 8	3100	GTO	Thaicom	Success	Success (drone ship)
2016-05-06	5:21:00	F9 FT B1022	CCAFS LC-40	JCSAT-14	4696	GTO	SKY Perfect JSAT Group	Success	Success (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)
2015-12-22	1:29:00	F9 FT B1019	CCAFS LC-40	OG2 Mission 2 11 Orbcomm-OG2 satellites	2034	LEO	Orbcomm	Success	Success (ground pad)

We performed a number of sql queries as shown above to compare the data in the data base

Interactive Map with Folium

```
[6]: # 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
```

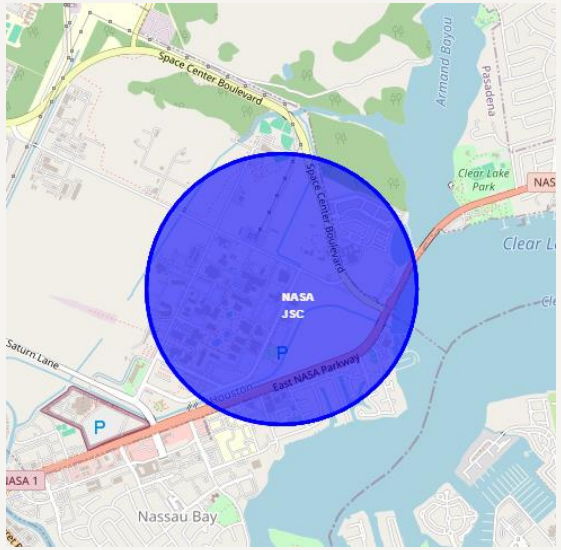
[6]:

	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	-120.610745

```
# 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,

```
# Create a blue circle at NASA Johnson Space Center's coordinate with a popup label showing its name
circle = folium.Circle(nasa_coordinate, radius=1000, color='#0000FF', 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:#FFFFFF;"><b>%s</b></div>' % 'NASA JSC',
        )
    )
site_map.add_child(circle)
site_map.add_child(marker)
```



Launch site map display using Folium
Visit my Github page for the rest of the maps, the code will be shown in the remaining slides
Here we are highlighting NASA Johnson Space Center with a blue circle

Interactive Map with Folium

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

	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

```
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

```
# 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
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)
```

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

```
# Add marker_cluster to current site_map
site_map.add_child(marker_cluster)

# 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 index, record in spacex_df.iterrows():
    # TODO: Create and add a Marker cluster to the site map
    # marker = folium.Marker(...)
    marker_cluster.add_child(marker)

site_map
```

```
# Create a `folium.PolyLine` object using the coastline coordinates and launch site coordinate
# lines=folium.PolyLine(locations=coordinates, weight=1)
launch_site_coordinates = [launch_site_lat, launch_site_lon]
lines=folium.PolyLine(locations=[coast_coordinates, launch_site_coordinates], weight=1)
site_map.add_child(lines)
```

```
# 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)

```
# 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',
    long_first=False,
    num_digits=20,
    prefix='Lat:',
    lat_formatter=formatter,
    long_formatter=formatter,
)

site_map.add_child(mouse_position)
site_map
```

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.

```
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 coastline using MousePosition and calculate the distance between the coastline point and the launch site.

```
# find coordinate of the closet coastline
coastline_lat = 28.56398
coastline_lon = -80.56809
launch_site_lat = 28.56321
launch_site_lon = -80.57683
distance_coastline = calculate_distance(launch_site_lat, launch_site_lon, coastline_lat, coastline_lon)
```

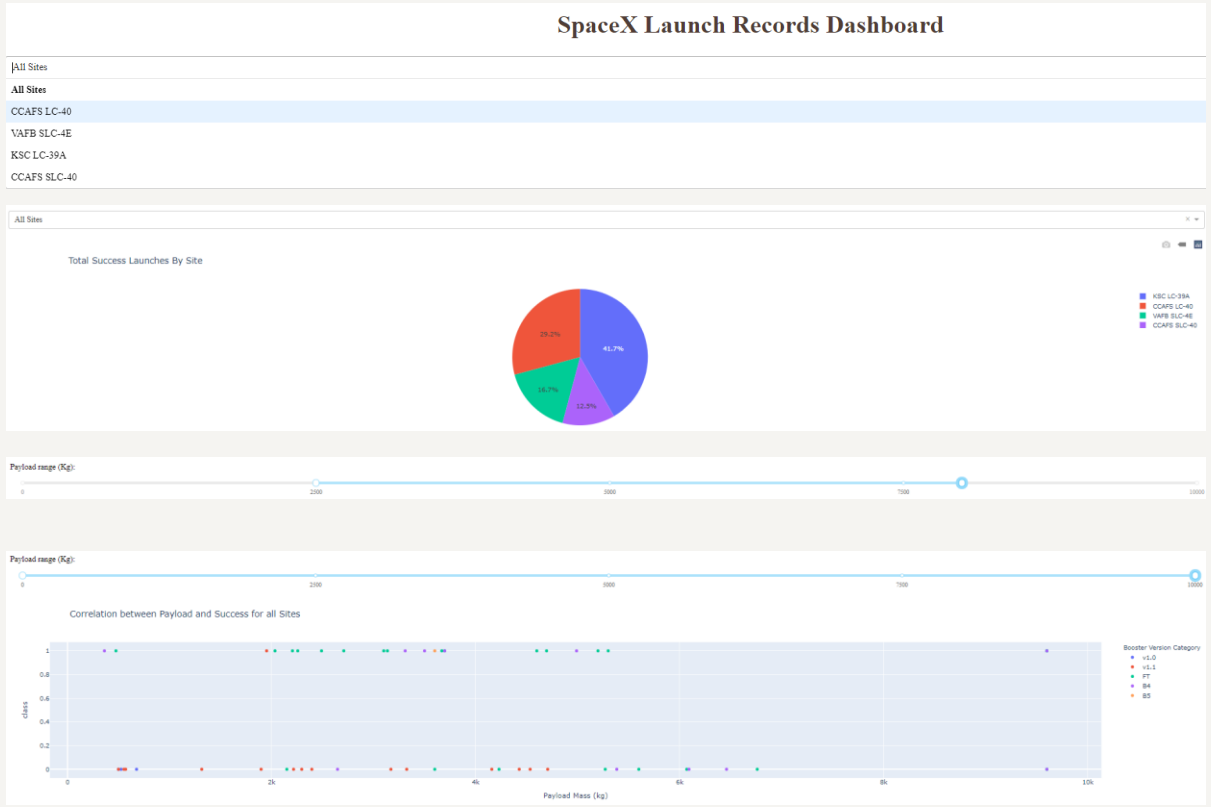
```
# Create and add a folium.Marker on your selected closest coastline point on the map
# Display the distance between coastline point and launch site using the icon property
coast_coordinates = [coastline_lat, coastline_lon]
distance_marker = folium.Marker(
    coast_coordinates,
    icon=DivIcon(
        icon_size=(20,20),
        icon_anchor=(0,0),
        html='%s' % "{:10.2f} KM".format(distance_coastline),)
    )
distance_marker.add_to(site_map)
site_map
```

```
# Create a marker with distance to a closest city, railway, highway, etc.
# Draw a line between the marker to the launch site
city_lat = 28.61208
city_lon = -80.80764
distance_city = calculate_distance(launch_site_lat, launch_site_lon, city_lat, city_lon)
city_coordinates = [city_lat, city_lon]
distance_marker = folium.Marker(city_coordinates,icon=DivIcon(icon_size=(20,20),icon_anchor=(0,0),html='%s' % "{:10.2f} KM".format(distance_city),)
distance_marker.add_to(site_map)
launch_site_coordinates = [launch_site_lat, launch_site_lon]
lines=folium.PolyLine(locations=[city_coordinates, launch_site_coordinates], weight=1)
site_map.add_child(lines)
site_map
```

Plotly Dash dashboard

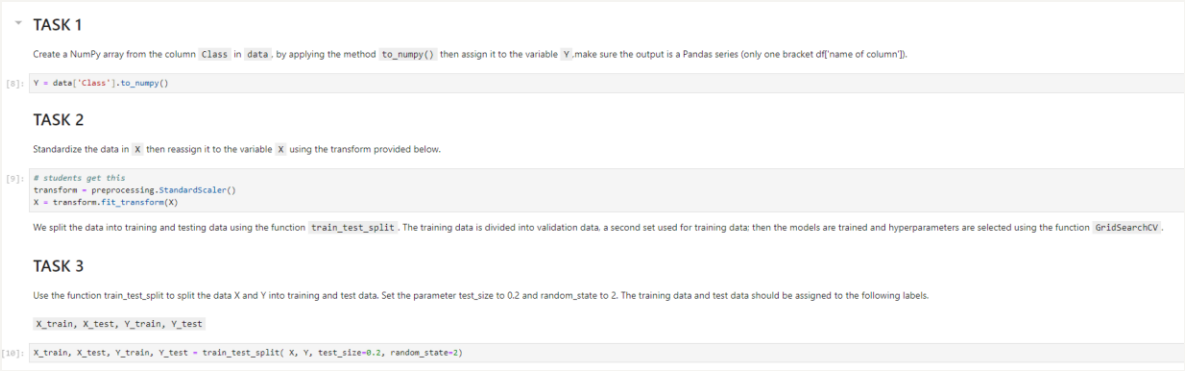
Tasks Included:

- Add a Launch Site Drop-down Input Component
- Add a callback function to render based on selected site dropdown
- Add a Range Slider to Select Payload
- Add a callback function to render the scatter plot



Predictive Analysis (Classification)

- I prepare the data by loading all the data into Panda's Dataframe, then standardize it and after that I split it into train and test sets
- I compare the different Machine learning techniques with the use of the GridsearchCV with the train and test data to find the best score, I also plotted a confusion matrix with that data
- These are the results of said comparison



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']).

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

TASK 2

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

```
[9]: # students get this
transform = preprocessing.StandardScaler()
X = transform.fit_transform(X)
```

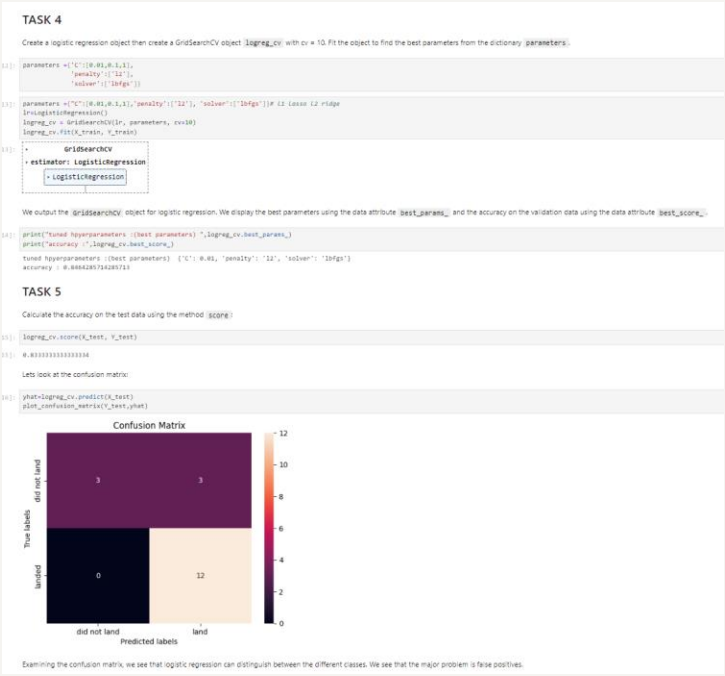
We split the data into training and testing data using the function `train_test_split`. The training data is divided into validation data, a second set used for training data; then the models are trained and hyperparameters are selected using the function `GridSearchCV`.

TASK 3

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

`X_train, X_test, Y_train, Y_test`

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



TASK 4

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

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

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

[13]: GridSearchCV
      estimator: LogisticRegression
        LogisticRegression
```

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_`.

```
[14]: print("tuned hyperparameters :(best parameters) ",logreg_cv.best_params_)
print("accuracy :",logreg_cv.best_score_)

tuned hyperparameters :(best parameters)  {'C': 0.01, 'penalty': 'l2', 'solver': 'lbfgs'}
accuracy : 0.8464285714285715
```

TASK 5

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

```
[15]: logreg_cv.score(X_test, Y_test)

[15]: 0.8333333333333334
```

Lets look at the confusion matrix:

```
[16]: yhat=logreg_cv.predict(X_test)
plot_confusion_matrix(Y_test,yhat)
```


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

```
print("Logistic Regression score = ",logreg_cv.score(X_test, Y_test))
print("Support Vector Machine score = ",svm_cv.score(X_test, Y_test))
print("Decision Tree score = ",tree_cv.score(X_test, Y_test))
print("K-Nearest Neighbor score = ",knn_cv.score(X_test, Y_test))
print("They are all the same.")
```

```
Logistic Regression score =  0.8333333333333334
Support Vector Machine score =  0.8333333333333334
Decision Tree score =  0.8333333333333334
K-Nearest Neighbor score =  0.8333333333333334
They are all the same.
```

OVERALL FINDINGS & IMPLICATIONS

Findings

- Finding 1: Looking at the bar chart in EDA with Data Visualization we can see that not orbits have different rates of success
- Finding 2: Looking at the scatter plots in EDA with Data Visualization for the launch sites tells us the success rates
- Finding 3: Looking at the Line plot in EDA with Data Visualization we can see it is trending upward

Implications

- Implication 1: Orbits like SSO, HEO, GEO an ES-L1 are more successful than other orbits
- Implication 2: CCAFS LC-40 has more failures than e KSC LC-39A and VAFB SLC 4E so those launch sites are more safe
- Implication 3: SpaceX has been on an upward trajectory with regards to success rates since 2013

CONCLUSION

- Certain Orbits are safer than others
- Launch sites KSC LC-39A and VAFB SLC 4E are more consistent than CCAFS LC-40
- SpaceX is getting better every year and is going in the right Direction

THANK YOU