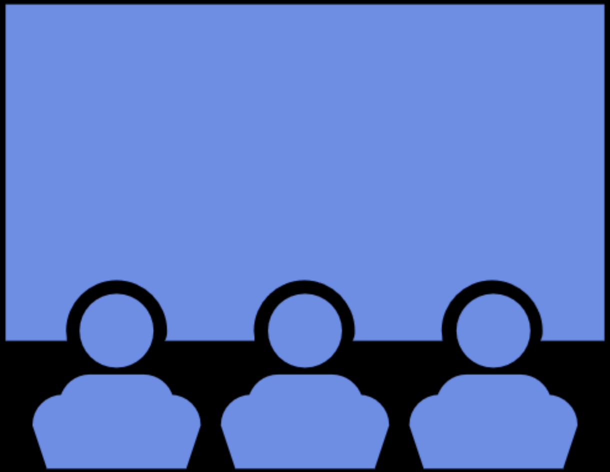
IBM Data Science Capstone Project

Tom Cruz - Data Analyst
August 25th, 2021

IBM

Contents



Executive Summary

With the accelerated advancements in technology within this century alone, the race to commercial space travel has never been more tangible.

SpaceX is one of the companies looking to make space travel attainable by minimizing on the most critical component: **cost**.

The cost of a single rocket launch can amount to over **$165 million**.

In this project, I incorporated the power of machine learning to help predict if a rocket will successfully land after launch for cost reduction through rocket reusability.

Introduction

As one of the leading companies innovating in the evolving space travel industry, SpaceX is heavily testing the capabilities of reusing rockets by attempting to land them after launch.

In this project, historical data of these experiments is used to define a machine learning model that can predict whether a landing will be successful.

The analysis and modeling will attempt to answer the following questions:

- *What key factors impact the outcome a landing?*
- *How can we plan more successful experiments?*
- *Should an experiment be attempted under set conditions?*

According to UBS, the space tourism market is estimated to be worth **$3 billion** by 2030

Today, costs for a rocket launch *alone* are upwards of **$165 million**

With the ability to land a rocket after launch, SpaceX estimates **62%** cost savings with rocket reuse – the first giant step to space travel affordability

Methodology

Data Collection

Multiple methods were used to acquire the datasets required for capturing the information necessary for analysis and modeling:

- SpaceX API calls
- Wikipedia Web Scraping

Data Wrangling

Performed inspection on the dataset of rocket launches to uncover distinct values across the features, calculate significant metrics (averages, success rates)

Exploratory Data Analysis

Utilized multiple data management frameworks to store and analyze the structured data, including metrics and visualizations to find relationships and patterns in the data

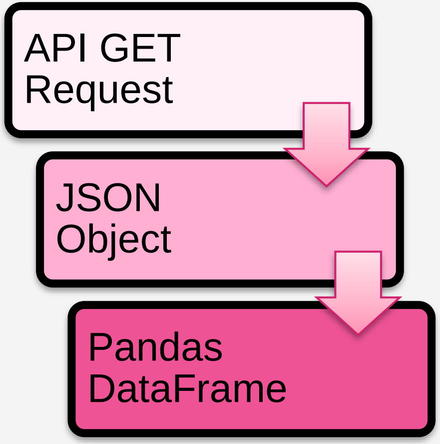
- DB2 on IBM Cloud
- Python Pandas DataFrame

Predictive Analysis & Modeling

Split data into train and test sets, trained multiple classification models, optimized hyperparameters, and validated the accuracy of the models

- Support Vector Machine (SVM)
- K-Nearest Neighbor (KNN)
- Decision Tree
- Logistic Regression

Data Collection: SpaceX API



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'
```

```
response_static = requests.get(static_json_url)
```

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_static.json())
```

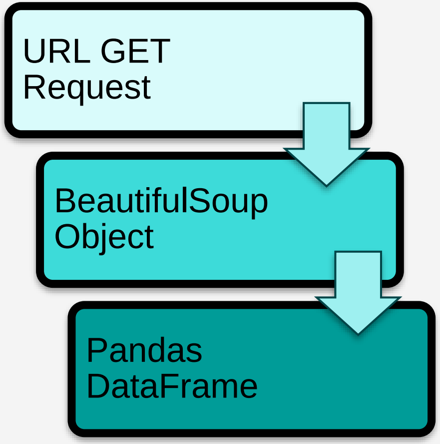
Using the dataframe `data` print the first 5 rows

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

Link to Jupyter Notebook: [SpaceX Falcon9 Landing Prediction](#)
[GitHub Repo: Data Collection with SpaceX API](#)

**Additional code available in Appendix A*

Data Collection:
Wikipedia Web Scrapping




```
static_url = "https://en.wikipedia.org/w/index.php?title=List_of_Falcon_9_and_Falcon_Heavy_launches&oldid=1027686922"
```

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

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

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

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

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

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

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

Link to Jupyter Notebook: [SpaceX Falcon9 Landing Prediction](#)
[GitHub Repo: Data Collection with Wikipedia Web Scrapping](#)

**Additional code available in Appendix A*

Data Wrangling

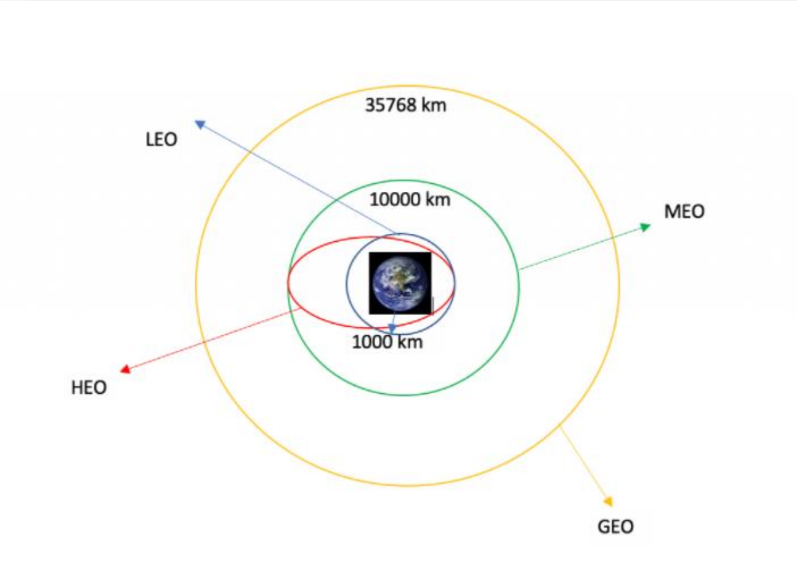
With the collected data organized in a Pandas DataFrame, the feature values were explored.

In order to understand the dataset, *value counts* were taken of different features to discover information on the following:

- Launch Sites
- Orbit Types
- Landing Outcomes

Lastly, the *landing outcomes* data was transformed to a new column *Class,* which provides a binary indication whether a launch was successful or not.

Orbits of Falcon 9 Launches



Link to Jupyter Notebook: [SpaceX Falcon9 Landing Prediction](#)
[GitHub Repo: Data Wrangling](#)

Exploratory Data Analysis

Performed SQL queries to calculated metrics on success, averages, and ranges of data values using a DB2 instance on IBM Cloud

- Total payload of the NASA (CRS) booster
- Success rate of landings

Created data visualizations with seaborn and matplotlib libraries to inspect relationships between features in the dataset

- Scatter Plots, Categorical Plots, Line Plots

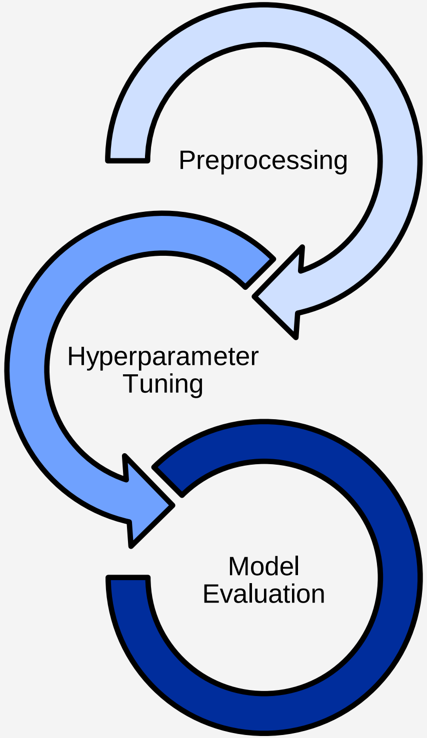
Links to Jupyter Notebooks:

– SpaceX Falcon9 Landing Prediction GitHub Repo: EDA with SQL

– SpaceX Falcon9 Landing Prediction GitHub Repo: EDA with Pandas

Predictive Analysis:
Modeling & Evaluation

- In the preprocessing of data, the feature set was standardized, since the variance across features was large
- The data was then split into training and testing sets for fitting and testing the models
- To optimize the hyperparameters for each model, *GridSearchCV* ran the classifiers with different sets of hyperparameters to identify the best ones
- R^2, accuracy, and the confusion matrix of each classifier were compared to identify the best model for the dataset



Link to Jupyter Notebook: [SpaceX Falcon9 Landing Prediction]()
[GitHub Repo: Classification Models for Rocket Landing]()

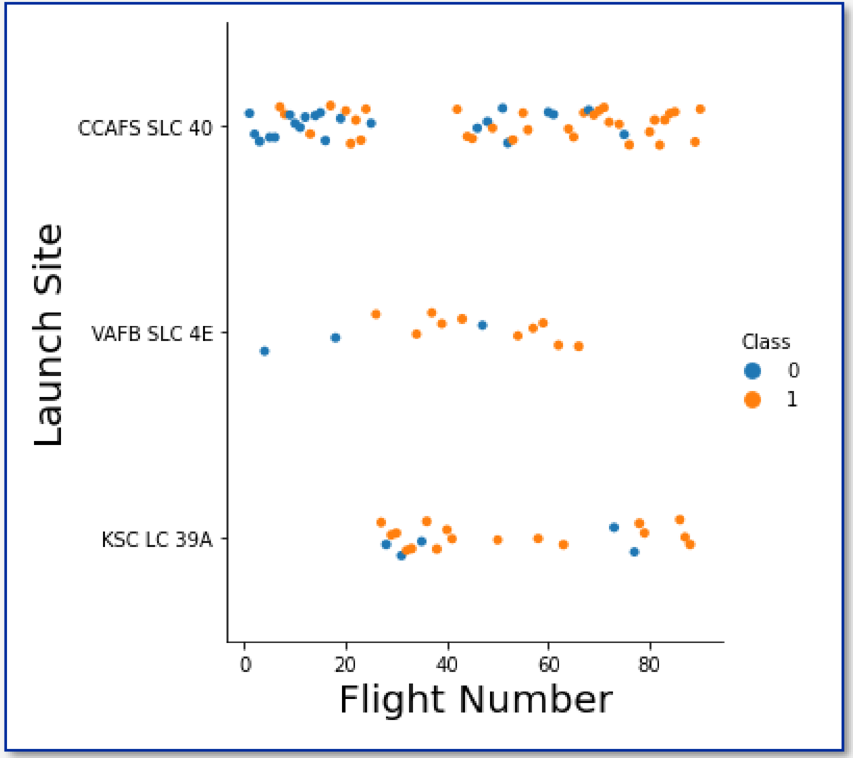
Results



Exploratory Data Analysis with Visualizations

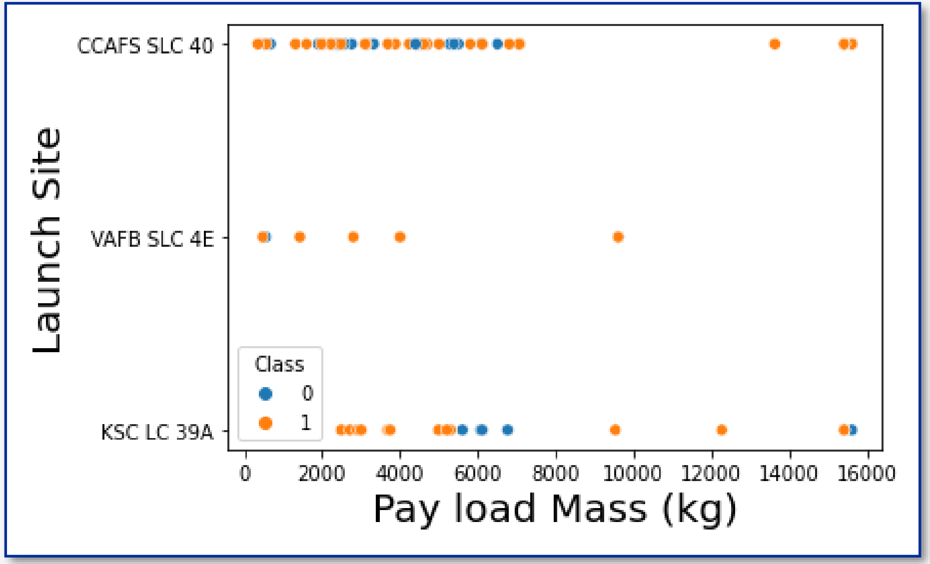
Python Visualizations:
Flight Number vs. Launch Site

- *CCAFS SLC 40* and *VAFB SLC 4E* were the launch sites of about the first 20 missions
  - Naturally, as with any experimentation, most of those first attempts were failures
- While launches at *VAFB SLC 4E* have a high success rate, not many launches occurred there
- Most all the latest launches, across all the sites have been successful
  - There appears to be a new factor impacting the latest streak in successful landings



Python Visualizations:
Payload Mass vs. Launch Site

- Most flights have consisted of payloads between ~100kg to ~8,000kg
- The varying payload masses show varying scenarios across launch sites
  - **CCAFS SLC 40**
    Inconsistent results regardless of payload mass, although higher payloads look promising
  - **VAFB SLC 4E**
    Most consistent results regardless of payload mass
  - **KSC LC 39A**
    All but one of the failures at this site were with flights with a payload mass in the neighborhood of ~6,000kg



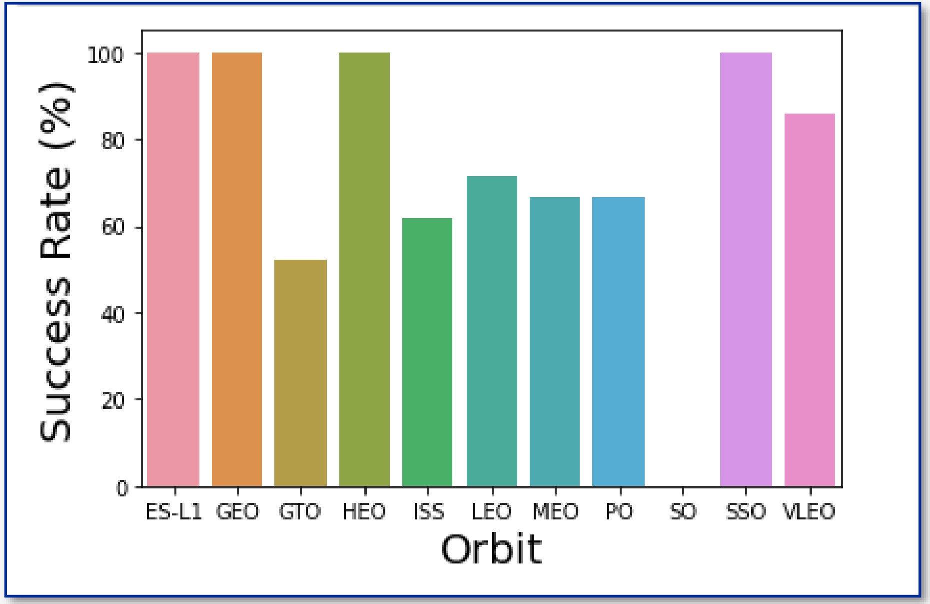
Python Visualizations:
Success Rate vs. Orbit Type

– A variety of different orbits have been selected for the Falcon 9 flights, along with a variance in success

- One set of orbits depict a great success rate of 100%, however, the number of flights for these orbits is needed to determine the significance of that rate

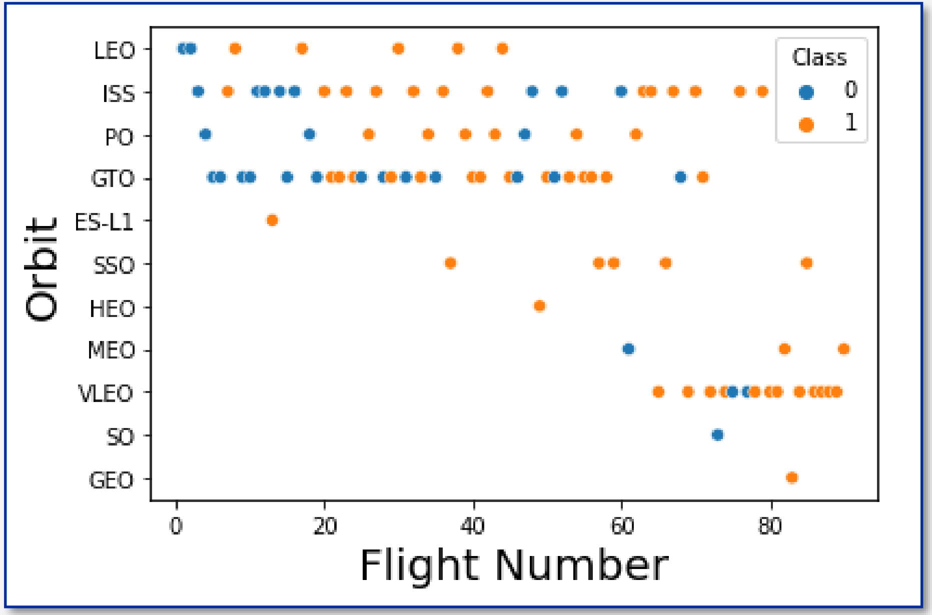
- Another set of orbits average at about a 60% success rate

- SO is the only orbit type that has not yet seen a flight with a successful landing



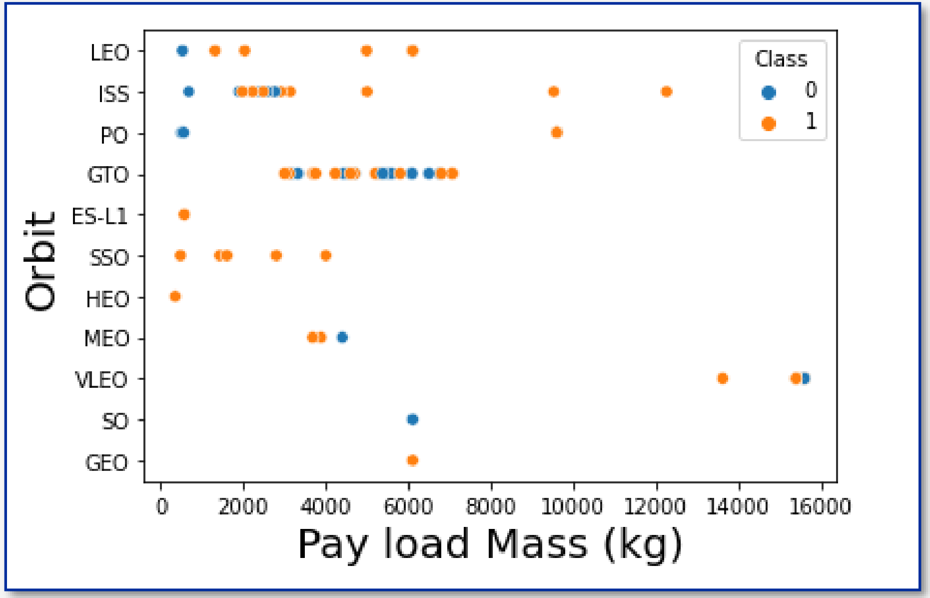
Python Visualizations:
Flight Number vs. Orbit Type

- Many of the first flights took orbit *LEO, ISS, PO,* and *GTO* – this is also where many of the failures occurred
  - There appears to be a correlation between early flight failures and orbit type failures
  - For *LEO, ISS, and PO*, many successful landings occur with later flights, but *GTO* continues to provide inconsistent results
- *VLEO* (Very Low Earth Orbit) appears to give very consistent success
  - This could be correlated with later flights, accounting for new factors in the launches



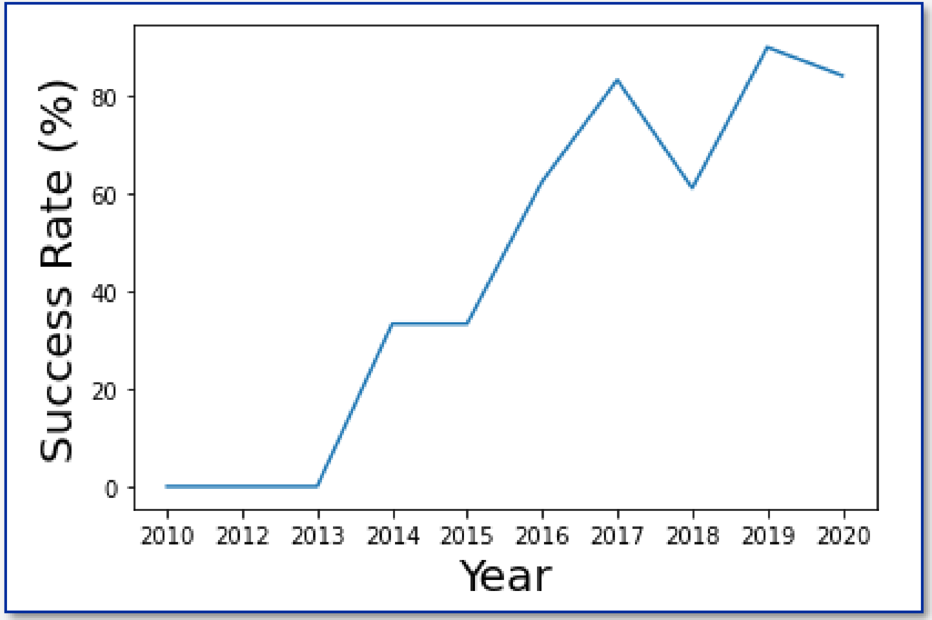
Python Visualizations:
Payload Mass vs. Orbit Type

- As was noted in an earlier observation, flights with heavier payloads resulted in consistent success rates
  - Note that this may be a factor for the success rate with VLEO, as the chart shows that only higher payloads were used in this orbit
- Flights in SSO look to be promising – flights with increasingly heavier payloads continue to result in successful landings



Python Visualizations:
Launch Success Yearly Trend

- Overall, there exists a positive linear relationship between the success rate of landings to the dates of the launches
- A considerable factor was at play in 2018 as the success rate plummeted by ~20%
  - Some factors that may be involved here may include:
    - New orbits for flights
    - Different payloads
    - External conditions at different launch sites



Exploratory Data Analysis with SQL

SQL Queries: Launch Site Names

```
SELECT DISTINCT launch_site
FROM SPACEXTBL
```

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

Four distinct launch sites for the ninety launches

SQL Queries:
5 Records for Launch Site with ‘CCA’

```
SELECT *
FROM SPACEXTBL
WHERE launch_site LIKE 'CCA%’
LIMIT 5
```

DATE	time__utc_	booster_version	launch_site	payload	payload_mass__kg_	orbit	customer	mission_outcome	landing__outcome
2010-06-04	18:45:00	F9 v1.0 B0003	CCAFS LC-40	Dragon Spacecraft Qualification Unit	0	LEO	SpaceX	Success	Failure (parachute)
2010-12-08	15:43:00	F9 v1.0 B0004	CCAFS LC-40	Dragon demo flight C1, two CubeSats, barrel of Brouere cheese	0	LEO (ISS)	NASA (COTS) NRO	Success	Failure (parachute)
2012-05-22	07:44:00	F9 v1.0 B0005	CCAFS LC-40	Dragon demo flight C2	525	LEO (ISS)	NASA (COTS)	Success	No attempt
2012-10-08	00:35:00	F9 v1.0 B0006	CCAFS LC-40	SpaceX CRS-1	500	LEO (ISS)	NASA (CRS)	Success	No attempt
2013-03-01	15:10:00	F9 v1.0 B0007	CCAFS LC-40	SpaceX CRS-2	677	LEO (ISS)	NASA (CRS)	Success	No attempt

First five launches at *CCAFS LC-40* were successful

SQL Queries:
Total Payload Mass of NASA (CRS) Boosters

```
SELECT SUM(payload_mass__kg_) AS
Total_Payload
FROM SPACEXTBL
WHERE customer = 'NASA (CRS)'
```

total_payload
45596

By inspection, payloads of NASA (CRS) boosters appear to be heavier than others

SQL Queries:
Average Payload Mass of F9 Booster (v1.1)

```
SELECT AVG(payload_mass__kg_) AS
Avg_payload
FROM SPACEXTBL
WHERE booster_version = 'F9 v1.1'
```

avg_payload
2928.400000

Knowing the payloads of F9 boosters range from several hindered to thousands of kilograms, this implies that heavier payloads are better

SQL Queries:
Date of First Successful Landing on Ground Pad

```
SELECT MIN(DATE) AS First_Success
FROM SPACEXTBL
WHERE landing__outcome = 'Success (ground pad)'
```

first_success
2015-12-22

Considering flights began in 2010, it took almost six years to achieve a success in landing a rocket

SQL Queries: Successful Landing with Mid-range Payload

```
SELECT payload, payload_mass__kg_,
landing__outcome
FROM SPACEXTBL
WHERE landing__outcome = 'Success (drone ship)’
AND payload_mass__kg_ BETWEEN 4000 AND 6000
```

payload	payload_mass__kg_	landing__outcome
JCSAT-14	4696	Success (drone ship)
JCSAT-16	4600	Success (drone ship)
SES-10	5300	Success (drone ship)
SES-11 / EchoStar 105	5200	Success (drone ship)

With only four successes for this payload range, this supports an earlier observation that this payload range is troublesome

SQL Queries:
Count of Mission Outcomes

```
SELECT mission_outcome, COUNT(*) AS Total
FROM SPACEXTBL
GROUP BY mission_outcome
```

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

While many failures were observed in landings, the missions had other objectives

SQL Queries:
Boosters with Highest Payload

```
SELECT booster_version, payload_mass__kg_
FROM SPACEXTBL
WHERE payload_mass__kg_ =
                    (SELECT MAX(payload_mass__kg_)
                     FROM SPACEXTBL)
```

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

F9 boosters have been the only booster type to carry the heaviest payload

SQL Queries:
Failed Drone Ship Landings in 2015

```
SELECT date, booster_version, launch_site,
landing__outcome
FROM SPACEXTBL
WHERE landing__outcome = 'Failure (drone ship)'
AND YEAR(DATE) = 2015
```

DATE	booster_version	launch_site	landing__outcome
2015-01-10	F9 v1.1 B1012	CCAFS LC-40	Failure (drone ship)
2015-04-14	F9 v1.1 B1015	CCAFS LC-40	Failure (drone ship)

Two attempts with two different models ended with the same result

SQL Queries:
Total Landing Outcomes between 2010 & 2017

```
SELECT landing__outcome,
COUNT(landing__outcome) AS Total
FROM SPACEXTBL
WHERE DATE BETWEEN '2010-06-04' AND '2017-03-20’
GROUP BY landing__outcome
ORDER BY Total DESC
```

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

Many missions did not even attempt a landing – these should be excluded from the input data of the models

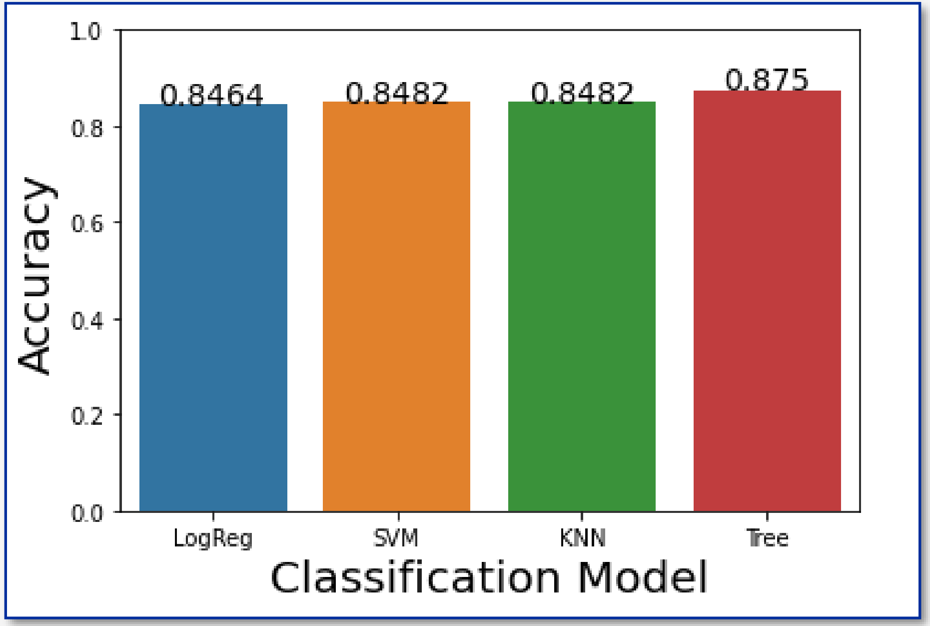
Predictive Analysis (Classification)

Predictive Analysis: Classification Accuracy

Four classification models were designed in the following steps:

- Standardization of the feature dataset
- Hyperparameter tuning with *GridSearchCV*
- Model fitting with training dataset

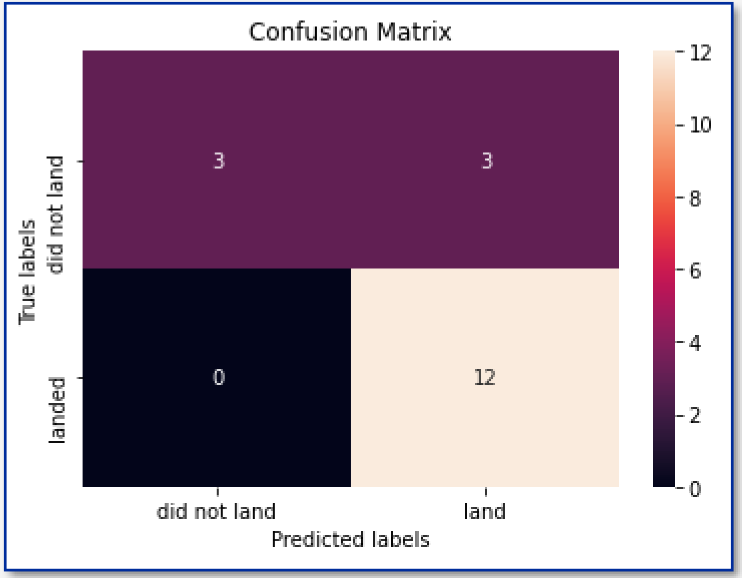
The results of the four models' accuracy are displayed in the graph – with the **Decision Tree classifier** as the most accurate model.



Predictive Analysis: Confusion Matrix

The confusion matrix of the Decision Tree classifier provides the following information:

- **Accuracy:** 0.83
  Rate at which records were accurately predicted
- **Recall (Type I Error):** 1.0
  How well the model detects a landing
- **Precision (Type II Error):** 0.8
  How well the model avoids mistaking a 'did not land' for a true 'land'



Conclusion

- While it took ten years to see the first successful rocket landing, machine learning help accelerate innovation by not waiting time on missions that are predicted for failure
- Additional features, such as weather conditions and other technical components of the rocket and booster, can help to build a more accurate model
- With the rate of success seen in recent years, commercial space travel will soon be ready for launch!

Thank you

Tom Cruz
Data Analyst
—
tom.cruz93@gmail.com
+1-669-225-1425

Appendix

Appendix A: Data Collection
SpaceX API

Filter on Falcon 9 Launches

Finally we will remove the Falcon 1 launches keeping only the Falcon 9 launches. Filter the data dataframe using the `BoosterVersion` column to only keep the Falcon 9 launches. Save the filtered data to a new dataframe called `data_falcon9`.

```
data_falcon9 = df[df['BoosterVersion'] != 'Falcon 1'].copy()
```

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

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

	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridF
4	1	2010-06-04	Falcon 9	NaN	LEO	CCSFS SLC 40	None None	1	Fa
5	2	2012-05-22	Falcon 9	525.0	LEO	CCSFS SLC 40	None None	1	Fa
6	3	2013-03-01	Falcon 9	677.0	ISS	CCSFS SLC 40	None None	1	Fa
7	4	2013-09-29	Falcon 9	500.0	PO	VAFB SLC 4E	False Ocean	1	Fa
8	5	2013-12-03	Falcon 9	3170.0	GTO	CCSFS SLC 40	None None	1	Fa
...	...	...	...	...	...	...	...	...	
89	86	2020-09-03	Falcon 9	15600.0	VLEO	KSC LC 39A	True ASDS	2	T
90	87	2020-10-06	Falcon 9	15600.0	VLEO	KSC LC 39A	True ASDS	3	T
91	88	2020-10-18	Falcon 9	15600.0	VLEO	KSC LC 39A	True ASDS	6	T
92	89	2020-10-24	Falcon 9	15600.0	VLEO	CCSFS SLC 40	True ASDS	3	T
93	90	2020-11-05	Falcon 9	3681.0	MEO	CCSFS SLC 40	True ASDS	1	T

90 rows × 17 columns

Replace Missing Values in DataFrame

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

```
# Calculate the mean value of PayloadMass column
payload_avg = data_falcon9.loc[:,'PayloadMass'].mean()
# Replace the np.nan values with its mean value
data_falcon9.loc[:,'PayloadMass'].fillna(payload_avg,inplace=True)
```

Appendix A: Data Collection
Web Scrapping

Extract Data from HTML Table

```
static_url = "https://en.wikipedia.org/w/index.php?title=List_of_Falcon_9_and_Falcon_Heavy_launches&oldid=1027686922"
```

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

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

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

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

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

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

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

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

```
# 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.findAll("table")
```

Starting from the third table is our target table contains the actual launch records.

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

Parse HTML Table into DataFrame

```
extracted_row = 0
#Extract each table
for table_number,table in enumerate(soup.find_all('table',"wikitable plainrowheaders collapsible")):
   # get table row
    for rows in table.find_all("tr"):
        #check to see if first table heading is as number corresponding to launch a number
        if rows.th:
            if rows.th.string:
                flight_number=rows.th.string.strip()
                flag=flight_number.isdigit()
        else:
            flag=False
        #get table element
        row=rows.find_all('td')
        #if it is number save cells in a dictonary
        if flag:
            extracted_row += 1
            # Flight Number value
            # TODO: Append the flight_number into launch_dict with key `Flight No.`

            launch_dict['Flight No.'].append(flight_number)
            #print(flight_number)
            datatimelist=date_time(row[0])

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

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

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

```
            # 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)

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

            # Customer
            # TODO: Append the customer into launch_dict with key `Customer`
            if row[6].a == None:
                launch_dict['Customer'].append(None)
            else:
                customer = row[6].a.string
                launch_dict['Customer'].append(customer)
            #print(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)

            # 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)
```

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