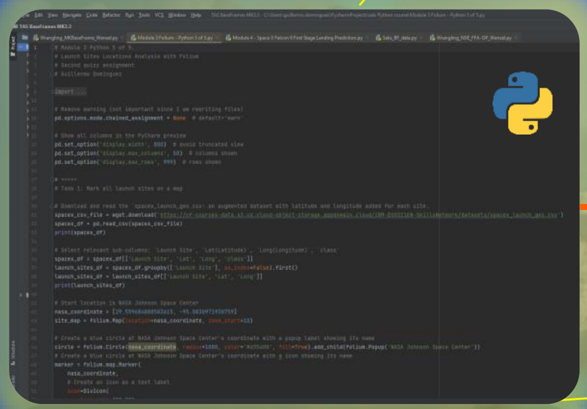
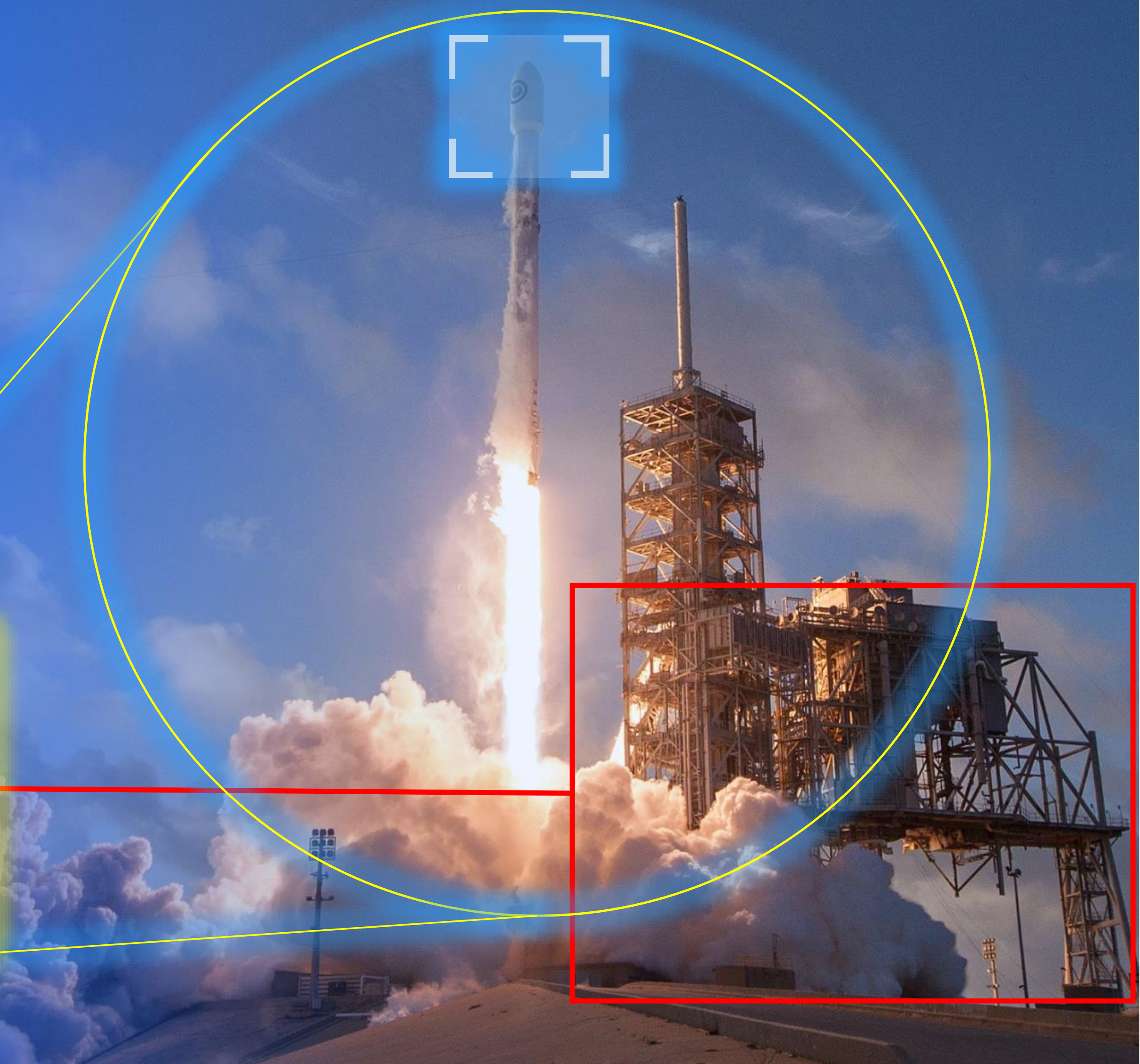
IBM Developer
SKILLS NETWORK

Winning Space Race with Data Science





Guillermo Domínguez Cañizares, PhD.
August 2022

Outline

- Introduction
- Methodology
- Results
- Conclusion
- Appendix

Introduction

- Project background and context
    - Using public data from SpaceX launches, it will be shown several techniques of data collecting, wrangling and visualization.
    - Using the processed data several strategies of inference, data training, extrapolation and predictive analysis will help to understand trends and extract conclusions.
- Problems you want to find answers
    - From the point of view of an investor on the products of SpaceX: what are the risks? How often there is a successful launch? What is the best method to put in different orbits specific amount of weight (cargo’s, satellites, etc.)

Section 1

Methodology

Methodology

Executive Summary

- Data collection methodology
  - Raw data location and retrieval, SpaceX API and Web scraping
- Perform data wrangling
  - Data processing methods
- Perform exploratory data analysis (EDA) using visualization
- Perform interactive visual analytics using Folium
- Perform predictive analysis using classification models
  - How to build, tune, evaluate classification models

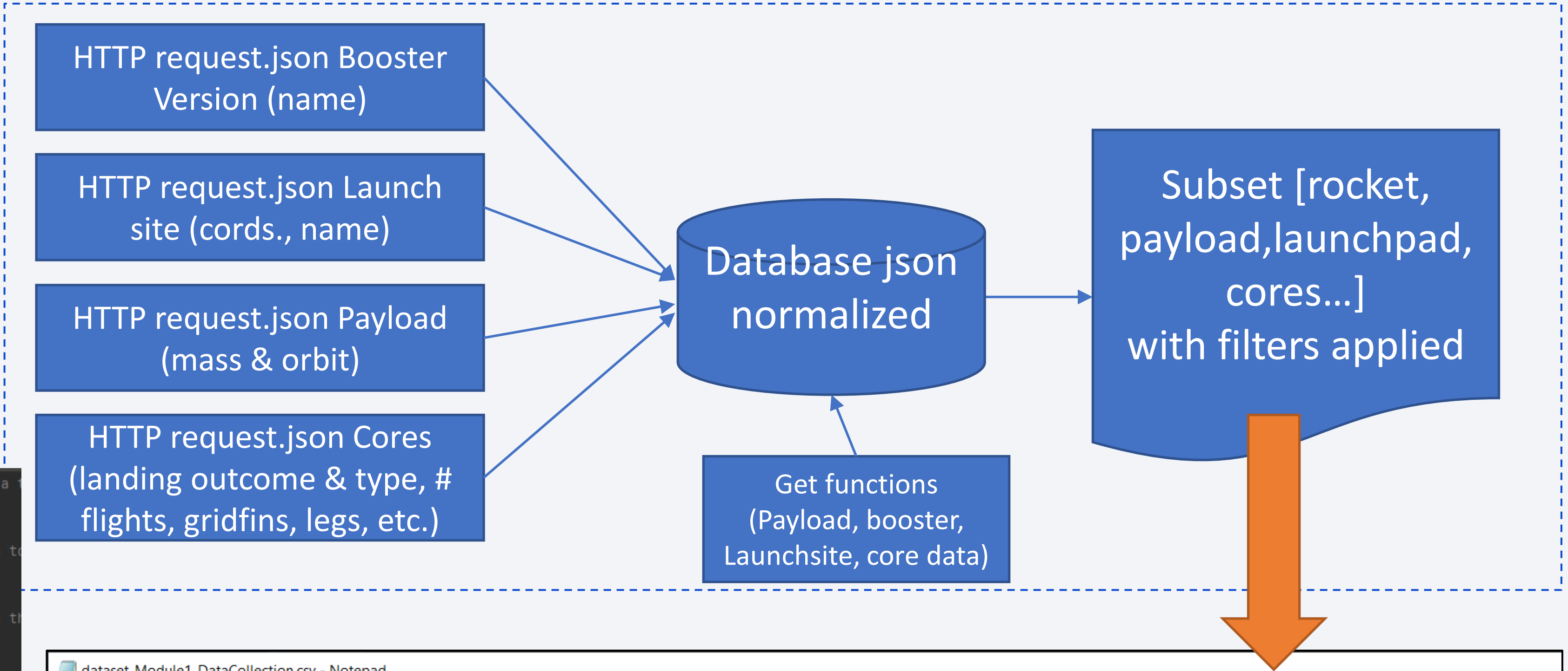
Data Collection Methodology

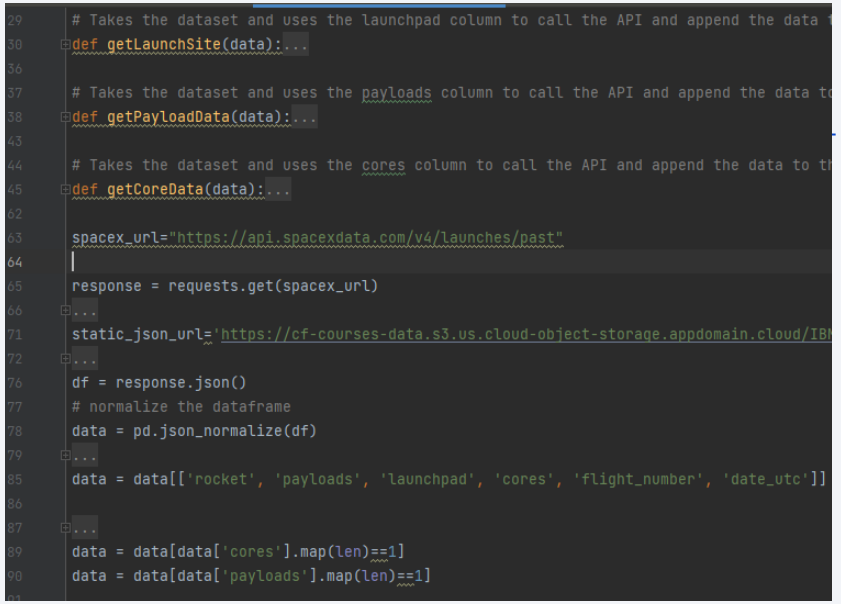
- Data is collected by a HTTP request to the SpaceX API*
- Once data is retrieved, several *get functions* are defined to extract specific data
- Finally data is completed (where NaN can cause problems), cleaned and organized to subsequently be analyzed
- IBM Cloud → SpaceX API (HTTP request) → retrieve data → Wrangling (give format, clean (scraping) and organize)
- Libraries used for Data collection: Python Interpreter v.3.9
  - Requests (HTTP requests to API)
  - Pandas
  - NumPy
  - Datetime
  - sys
  - requests
  - from bs4 import BeautifulSoup
  - re
  - unicodedata

* *An **application programming interface** (**API**) is a way for two or more [computer programs]() to communicate with each other.*

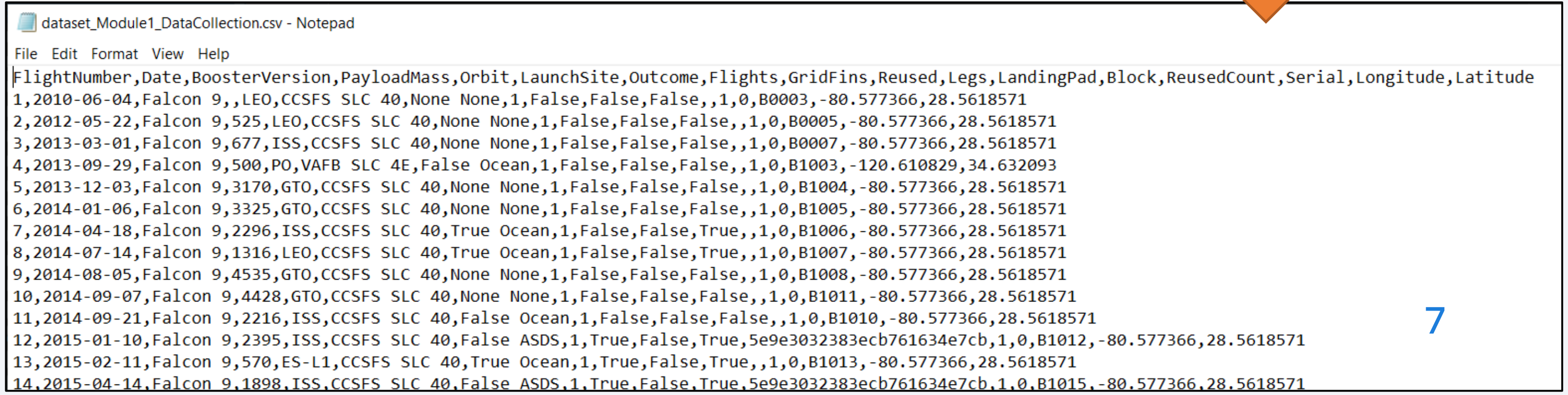
Data Collection Methodology – SpaceX API

- Data collection flowchart
- Screenshot of code example (PyCharm IDE) & final data subset ready for analysis
- GitHub URLs:
  - Python code for API
  - Dataset resulted from API





```python
# Takes the dataset and uses the launchpad column to call the API and append the data t
def getLaunchSite(data):...

# Takes the dataset and uses the payloads column to call the API and append the data to
def getPayloadData(data):...

# Takes the dataset and uses the cores column to call the API and append the data to th
def getCoreData(data):...

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

response = requests.get(spacex_url)
...
static_json_url='https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM
...
df = response.json()
# normalize the dataframe
data = pd.json_normalize(df)
...
data = data[['rocket', 'payloads', 'launchpad', 'cores', 'flight_number', 'date_utc']]

...
data = data[data['cores'].map(len)==1]
data = data[data['payloads'].map(len)==1]
```



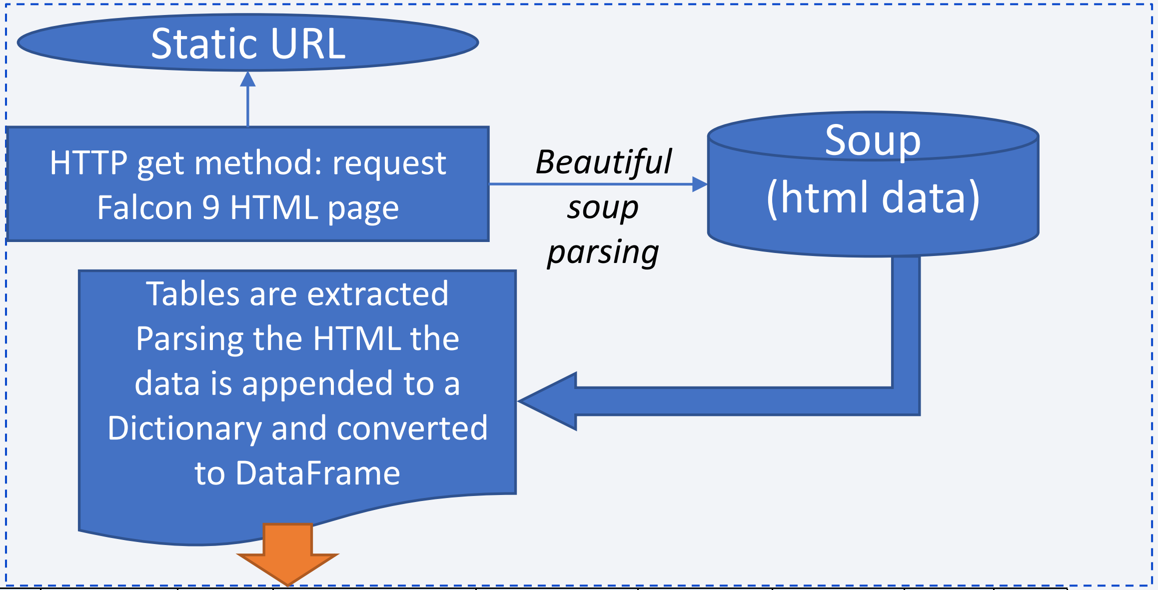
dataset_Module1_DataCollection.csv - Notepad

File Edit Format View Help

```
FlightNumber,Date,BoosterVersion,PayloadMass,Orbit,LaunchSite,Outcome,Flights,GridFins,Reused,Legs,LandingPad,Block,ReusedCount,Serial,Longitude,Latitude
1,2010-06-04,Falcon 9,,LEO,CCSFS SLC 40,None None,1,False,False,False,,1,0,B0003,-80.577366,28.5618571
2,2012-05-22,Falcon 9,525,LEO,CCSFS SLC 40,None None,1,False,False,False,,1,0,B0005,-80.577366,28.5618571
3,2013-03-01,Falcon 9,677,ISS,CCSFS SLC 40,None None,1,False,False,False,,1,0,B0007,-80.577366,28.5618571
4,2013-09-29,Falcon 9,500,PO,VAFB SLC 4E,False Ocean,1,False,False,False,,1,0,B1003,-120.610829,34.632093
5,2013-12-03,Falcon 9,3170,GTO,CCSFS SLC 40,None None,1,False,False,False,,1,0,B1004,-80.577366,28.5618571
6,2014-01-06,Falcon 9,3325,GTO,CCSFS SLC 40,None None,1,False,False,False,,1,0,B1005,-80.577366,28.5618571
7,2014-04-18,Falcon 9,2296,ISS,CCSFS SLC 40,True Ocean,1,False,False,True,,1,0,B1006,-80.577366,28.5618571
8,2014-07-14,Falcon 9,1316,LEO,CCSFS SLC 40,True Ocean,1,False,False,True,,1,0,B1007,-80.577366,28.5618571
9,2014-08-05,Falcon 9,4535,GTO,CCSFS SLC 40,None None,1,False,False,False,,1,0,B1008,-80.577366,28.5618571
10,2014-09-07,Falcon 9,4428,GTO,CCSFS SLC 40,None None,1,False,False,False,,1,0,B1011,-80.577366,28.5618571
11,2014-09-21,Falcon 9,2216,ISS,CCSFS SLC 40,False Ocean,1,False,False,False,,1,0,B1010,-80.577366,28.5618571
12,2015-01-10,Falcon 9,2395,ISS,CCSFS SLC 40,False ASDS,1,True,False,True,5e9e3032383ecb761634e7cb,1,0,B1012,-80.577366,28.5618571
13,2015-02-11,Falcon 9,570,ES-L1,CCSFS SLC 40,True Ocean,1,True,False,True,,1,0,B1013,-80.577366,28.5618571
14,2015-04-14,Falcon 9,1898,ISS,CCSFS SLC 40,False ASDS,1,True,False,True,5e9e3032383ecb761634e7cb,1,0,B1015,-80.577366,28.5618571
```

Data Collection Methodology – Web scraping

- Scraping of the data
  - Install packages: beautifulsoup4, requests (PyCharm IDE)
  - Static URL contains all info: HTML specific page is requested
  - Beautiful soup parses the HTML
  - Loops/functions are created to extract info form the tables in the HTML
    - Date time table
    - Booster table
    - Landing status table
    - Mass table
  - Dictionary is fed with data → DataFrame is created (.csv exported), ready for analysis.



- GitHub links:
  - Python code of the Web Scraping
  - Dataset ouput of the Web Scraping

Flight No.	Launch site	Payload	Payload mass	Orbit	Customer	Launch outcome	Version Booster	Booster landing	Date	Time
1	CCAFS	Dragon Spacecraft Qualification Unit	0	LEO	SpaceX	Success	F9 v1.0B0003.1	Failure	04-Jun-10	18:45
2	CCAFS	Dragon	0	LEO	NASA	Success	F9 v1.0B0004.1	Failure	08-Dec-10	15:43
3	CCAFS	Dragon	525 kg	LEO	NASA	Success	F9 v1.0B0005.1	No attempt	22-May-12	07:44
4	CCAFS	SpaceX CRS-1	4,700 kg	LEO	NASA	Success	F9 v1.0B0006.1	No attempt	08-Oct-12	00:35
5	CCAFS	SpaceX CRS-2	4,877 kg	LEO	NASA	Success	F9 v1.0B0007.1	No attempt	01-Mar-13	15:10
6	VAFB	CASSIOPE	500 kg	Polar orbit	MDA	Success	F9 v1.1B1003	Uncontrolled	29-Sep-13	16:00
7	CCAFS	SES-8	3,170 kg	GTO	SES	Success	F9 v1.1	No attempt	03-Dec-13	22:41
8	CCAFS	Thaicom 6	3,325 kg	GTO	Thaicom	Success	F9 v1.1	No attempt	06-Jan-14	22:06
9	Cape Canaveral	SpaceX CRS-3	2,296 kg	LEO	NASA	Success	F9 v1.1	Controlled	18-Apr-14	19:25
10	Cape Canaveral	Orbcomm-OG2	1,316 kg	LEO	Orbcomm	Success	F9 v1.1	Controlled	14-Jul-14	15:15
11	Cape Canaveral	AsiaSat 8	4,535 kg	GTO	AsiaSat	Success	F9 v1.1	No attempt	05-Aug-14	08:00
12	Cape Canaveral	AsiaSat 6	4,428 kg	GTO	AsiaSat	Success	F9 v1.1	No attempt	07-Sep-14	05:00
13	Cape Canaveral	SpaceX CRS-4	2,216 kg	LEO	NASA	Success	F9 v1.1	Uncontrolled	21-Sep-14	05:52
14	Cape Canaveral	SpaceX CRS-5	2,395 kg	LEO	NASA	Success	F9 v1.1	Failure	10-Jan-15	09:47
15	Cape Canaveral	DSCOVR	570 kg	HEO	USAF	Success	F9 v1.1	Controlled	11-Feb-15	23:03
16	Cape Canaveral	ABS-3A	4,159 kg	GTO	ABS	Success	F9 v1.1	No attempt	02-Mar-15	03:50
17	Cape Canaveral	SpaceX CRS-6	1,898 kg	LEO	NASA	Success	F9 v1.1	Failure	14-Apr-15	20:10
18	Cape Canaveral	TÃ¼rkmenÃ„lem 52Â°E / MonacoSAT	4,707 kg	GTO		Success	F9 v1.1	No attempt	27-Apr-15	23:03
19	Cape Canaveral	SpaceX CRS-7	1,952 kg	LEO	NASA	Failure	F9 v1.1	Precluded	28-Jun-15	14:21
20	Cape Canaveral	Orbcomm-OG2	2,034 kg	LEO	Orbcomm	Success	F9 FT	Success	22-Dec-15	01:29
21	VAFB	Jason-3	553 kg	LEO	NASA	Success	F9 v1.1	Failure	17-Jan-16	18:42
22	Cape Canaveral	SES-9	5,271 kg	GTO	SES	Success	F9 FT	Failure	04-Mar-16	23:35

Data Wrangling

- Using Data collected from Web Scraping (existing dataframe of 90 rows x 17 columns)
- Since data is already ready to analyze, different *pandas queries* are performed to create different tables of data.
- An additional *binary Class* is added to indicate if the landing was successful or not. This will be useful later for plotting results and analyze the data.
- *One-Hot encoding* of relevant parameters
- GitHub URLs:
  - [Python code of Data Wrangling]
  - [Dataset ouput of the Wrangling]

Falcon 9 dataframe (90x17) sorted by date

.value_counts()
.groupby()

Different tables with information.

Class added for Outcome failed or successful

=== Number of launches for each site ===

CCAFS SLC 40	55
KSC LC 39A	22
VAFB SLC 4E	13

=== Number and occurrence of each orbit ===

GTO	27
ISS	21
VLEO	14
PO	9
LEO	7
SSO	5
MEO	3
ES-L1	1
HEO	1
SO	1
GEO	1

=== Number and occurrence of mission outcome per orbit type ===

Orbit	Outcome	
ES-L1	True Ocean	1
GEO	True ASDS	1
GTO	True ASDS	13
	None None	11
	False ASDS	1
	None ASDS	1
	True Ocean	1
HEO	True ASDS	1
ISS	True RTLS	7
	True ASDS	5
	None None	3
	False ASDS	2
	False Ocean	1
	False RTLS	1
	None ASDS	1
	True Ocean	1
LEO	True RTLS	4
	None None	2
	True Ocean	1
MEO	True ASDS	2
	None None	1
PO	True ASDS	5
	False ASDS	1
	False Ocean	1
	None None	1
	True Ocean	1
SO	None None	1
SSO	True RTLS	3
	True ASDS	2
VLEO	True ASDS	12
	False ASDS	2

Task 4: find the failures to land

A much more *pythonic* way to reduced all the Hands-on lab proposed code lines to only one:

```
# Using a fast numpy method:
bad_outcomes = df['Outcome'].str.contains('None', regex=False).sum()
```

Result: 21 failures → success rate is 66%

Commented on Course Discussion forum [here].

Data Wrangling – One-hot encoding

- *One-hot encoding* is a method of pre-processing data, used to deal with categorical data so that the input for the for machine learning models is numerical (binary).
- *One-hot encoding* of relevant parameters is applied before the predictive analysis models are used, in this particular case the table shown below has <u>4 columns one-hot encoded: Orbit, Launchsite, LandingPad, Serial</u>
  - Initial Table: 12 columns, where the 4 columns mentioned, have different values:
    - Orbit types: 11
    - LaunchSites: 3
    - LandingPads: 5
    - Serial: 53
  - **Output:** one-hot encoded table: 80 columns (8 columns not encoded + 72 (11+3+5+53) columns encoded)

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

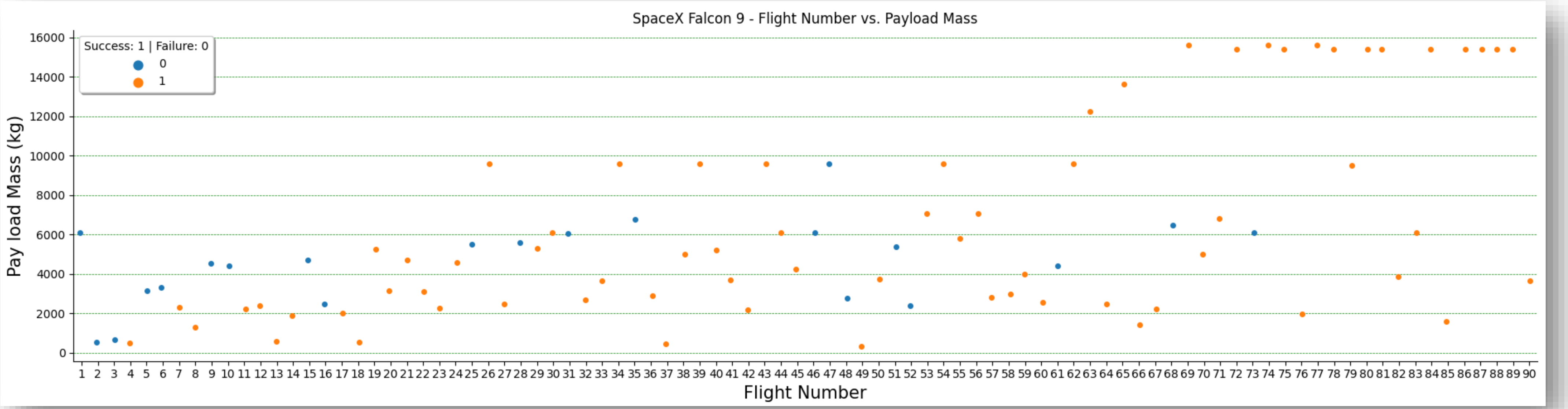
	FlightNumber	PayloadMass	Orbit	LaunchSite	Flights	GridFins	Reused	Legs	LandingPad	Block	ReusedCount	Serial	Orbit_GEO	Orbit_GTO	Orbit_HEO	Orbit_ISS	Orbit_LEO	Orbit_MEO	Orbit_PO	Orbit_SO	Orbit_SSO	Orbit_VLEO	Launch
0	1	6104.959412	LEO	CCAFS SLC 40	1	False	False	False	NaN	1.0	0	B0003	0	0	0	0	1	0	0	0	0	0	0
1	2	525.000000	LEO	CCAFS SLC 40	1	False	False	False	NaN	1.0	0	B0005	0	0	0	0	1	0	0	0	0	0	0
2	3	677.000000	ISS	CCAFS SLC 40	1	False	False	False	NaN	1.0	0	B0007	0	0	0	1	0	0	0	0	0	0	0
3	4	500.000000	PO	VAFB SLC 4E	1	False	False	False	NaN	1.0	0	B1003	0	0	0	0	0	0	1	0	0	0	0
4	5	3170.000000	GTO	CCAFS SLC 40	1	False	False	False	NaN	1.0	0	B1004	0	1	0	0	0	0	0	0	0	0	0

	LaunchSite_KSC LC 39A	LaunchSite_VAFB SLC 4E		LandingPad_5e9e3032383ecb554034e7c9	...	Serial_B1032	Serial_B1034	Serial_B1035	Serial_B1036	Serial_B1037	Serial_B1038	Serial_B1039	Serial_B1040	Serial_B1041	Serial_B1042	Serial_B1
	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	0	0	0
	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	0	0	0	0	0
	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0	0

EDA with Data Visualization

Exploratory Data Analysis (EDA) with Data Visualization

- Helps to quickly identify main relationships between parameters, relevant parmeters and regions of interest within the database space
- Relevant parameters will be later subselected to create models with them
- GitHub link to the different Modules of Data Visualization (Python code and data subsets outputs): https://github.com/GuillermoDC/Python5
- Finally, some key features are convertes to dummies variables (binary) to allow further analysis (only to seen in the Python code in GitHub repository). The subselection of 13 columns binary encoded results in 81 columns dataframe.
- See example below of scatter plot for Payload mass vs. Flight Number:



Payload Mass vs. Flight Number scatter plot:

- Launch sites success rates:
  - CCAFS LC-40: 60% success rate
  - KSC LC-39A: 77% success rate
  - VAFB SLC 4E: 77% success rate

EDA with SQL

- As stated in the discussion thread Module 2 SQL of February 2022, I do not agree that SQL should be used in this IBM Python course to perform EDA since there is a specific IBM course using SQL. Several other course students concur.
    - See discussion thread: link to edx IBM DS0720EN Discussion
- Therefore, the questions proposed in the Hands-on lab have been solved using Python instead.
- See in GitHub the complete code file in this link
- I fully agree that SQL is more efficient for large database (which is not the case for the databased samples used in this course) but the goal of Module 2 is perform an EDA, independent of the language.

Build an Interactive Map with Folium

- Libraries required for this section:
  - Folium
    - Pluging MarkerCluster
    - Pluging MousePosition
    - Pluging DivIcon
  - Wget
  - Pandas

- All Launch sites have been marked on a map using their Latitude & Longitude coordinades from SpaceX public information, popup marks add information:

  - Labels with Launch name

  - Color mark Successful (green) amd Failure (red) launchs

  - Mark Clusters to simplify several launches in same site

  - Straight line from Launch sites to its proximities (railroad, highway, coastline) and their estimated distance is given

- Maps enriched with title, scale and coordinates on mouse position

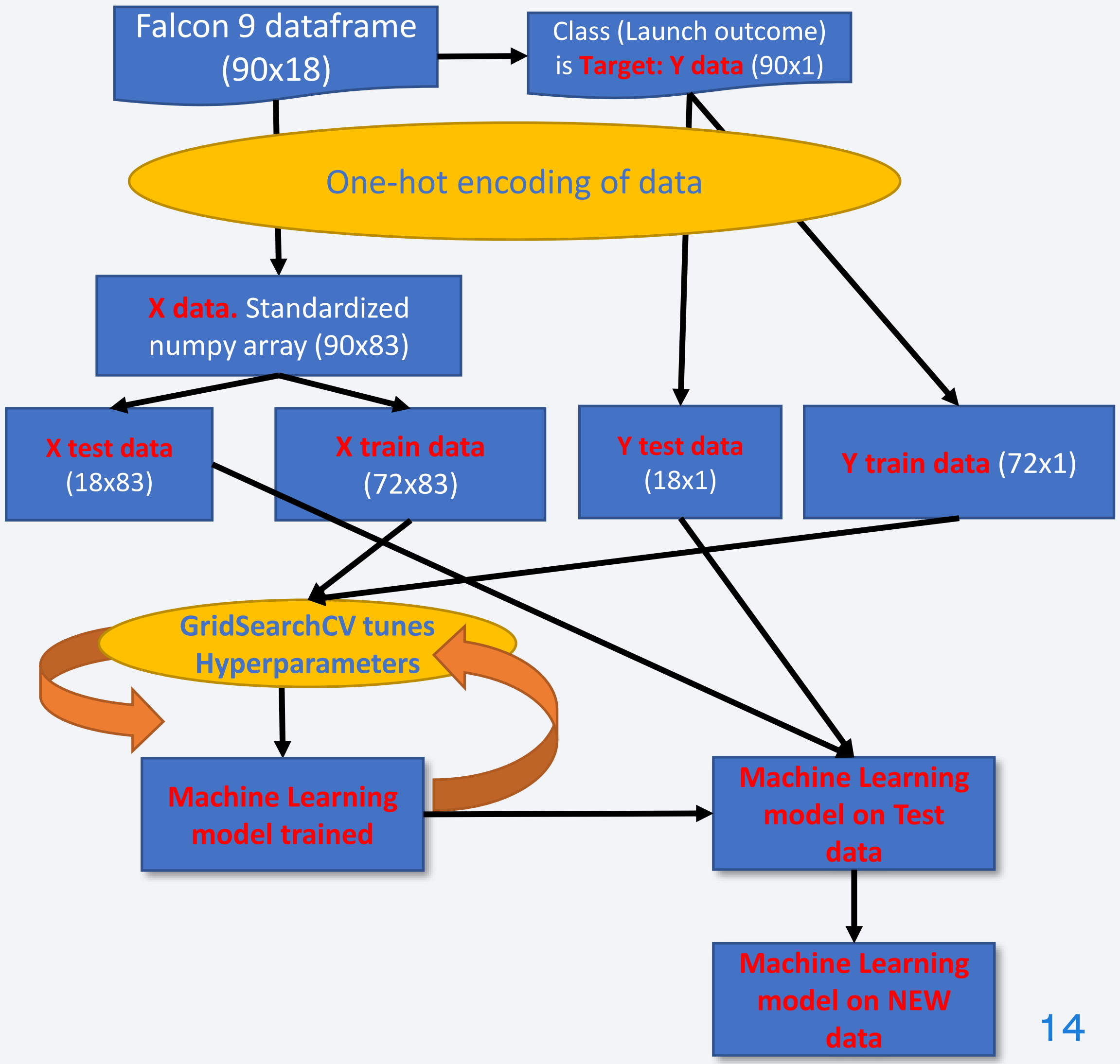
Predictive Analysis (Classification)

Before predictive Analysis data has to be prepared:

- *One-hot encoding* of relevant parameters (features and target)
- Standardize the data (mean removed and scaled by its standard deviation) so all data values lies within [-1, +1]
- Split the data into Train and Test sets: 20% of the data size is assigned to be the test (18 out of 90 sets)

GitHub URL of the completed predictive analysis lab: link

Flowchart of data processing and input into model



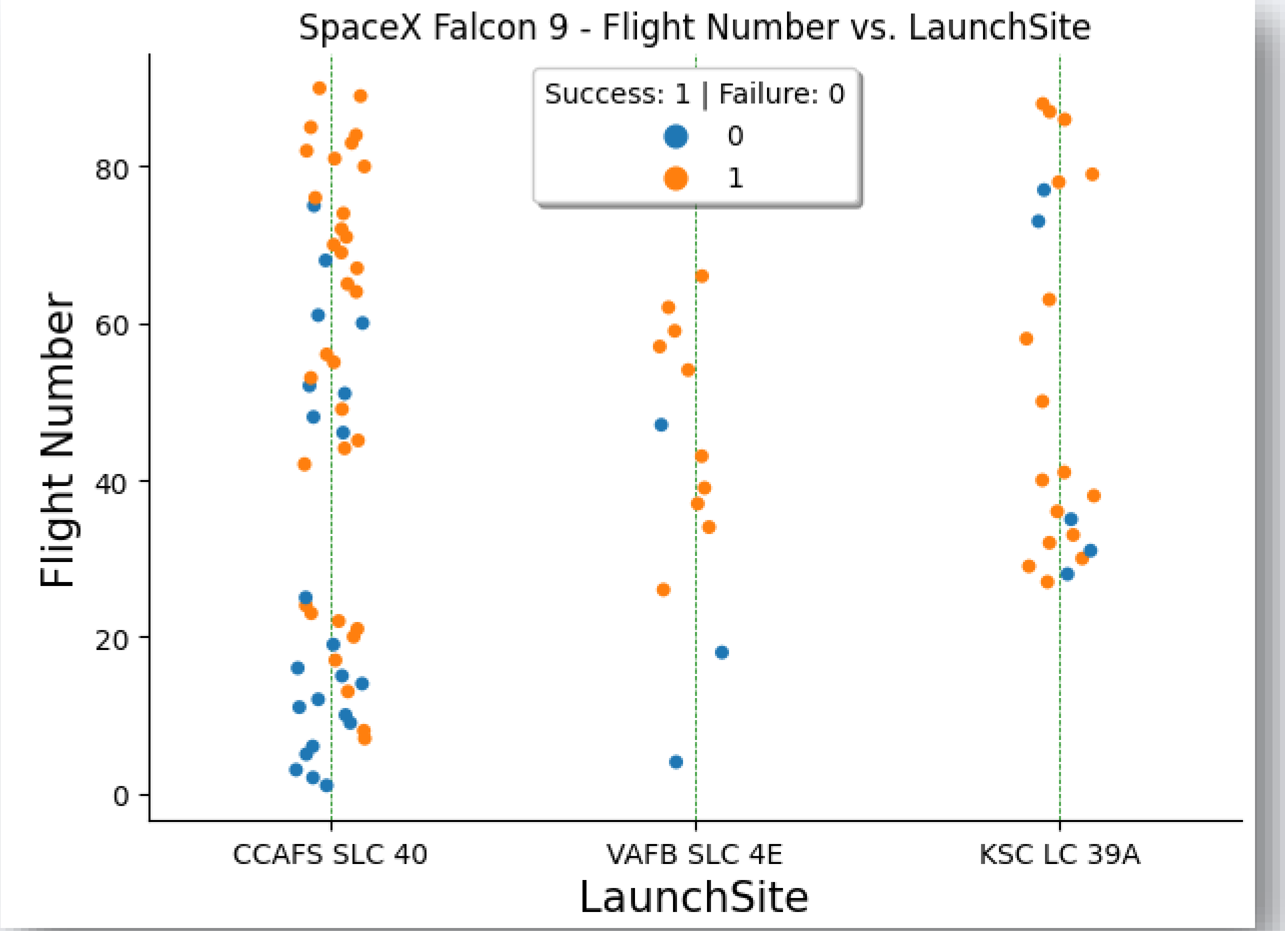
Results

- Exploratory data analysis results
  - Data is processed and ready to be plotted, used by Foil Maps, given as input for classification models, etc.
- Interactive analytics demo in screenshots
  - Foil Map
  - Dashboard has not been included in this final project due to difficulties with the platform. I consider this a minor topic in this course and not relevant. I understand the usability of a dashboard, but I wanted to focus in the Python capabilities and the machine learning
- Predictive analysis results
  - Four classification methods have been used

Section 2

Insights drawn from EDA

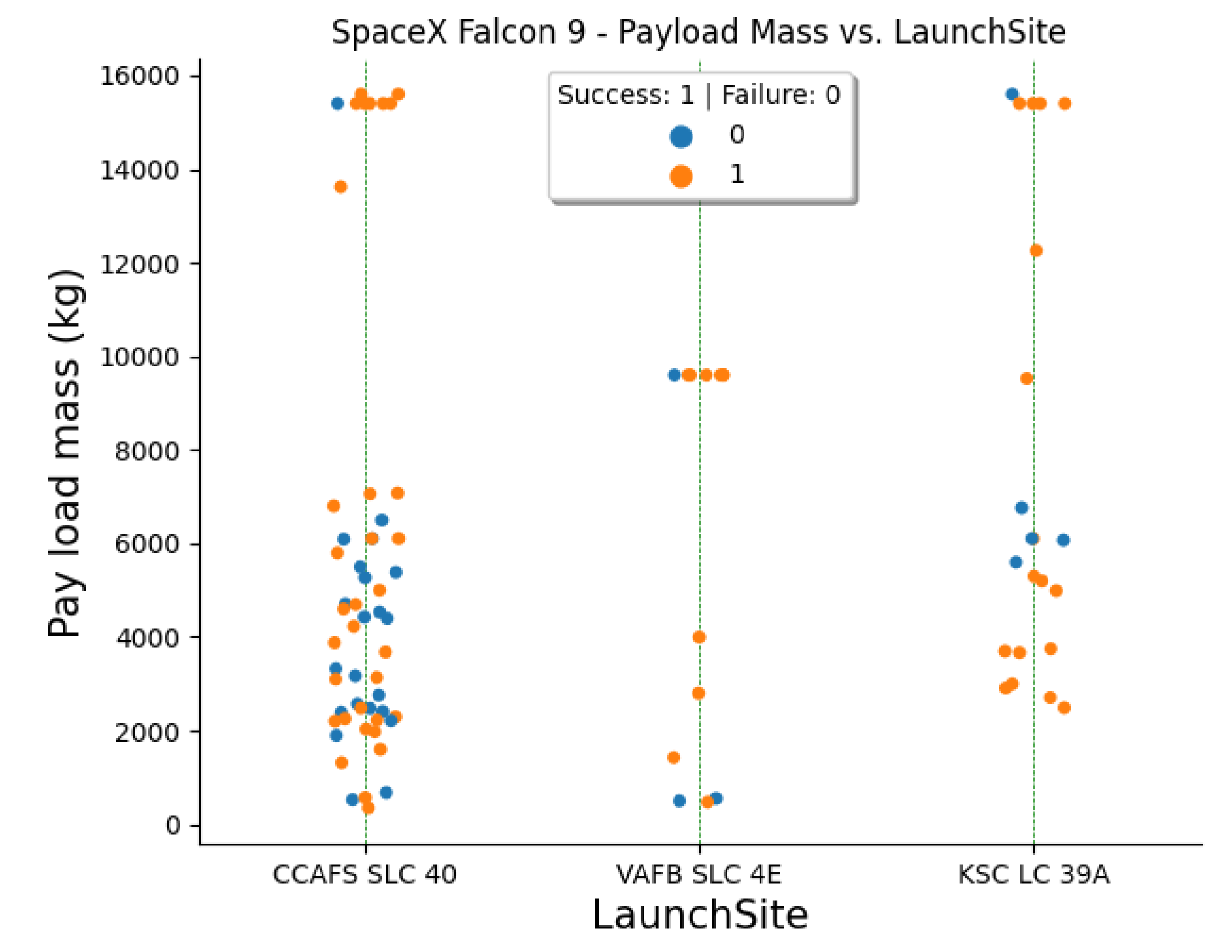
Flight Number vs. Launch Site



Task 1:

- **Flight Number vs. Launch site scatter plot:**
  - CCAFS LC-40: Most of the first and last flights are launched here, accumulates more landing failures
  - KSC LC-39A: most Intermmediate flight numbers
  - VAFB SLC 4E: Mainly intermmediate flight numbers

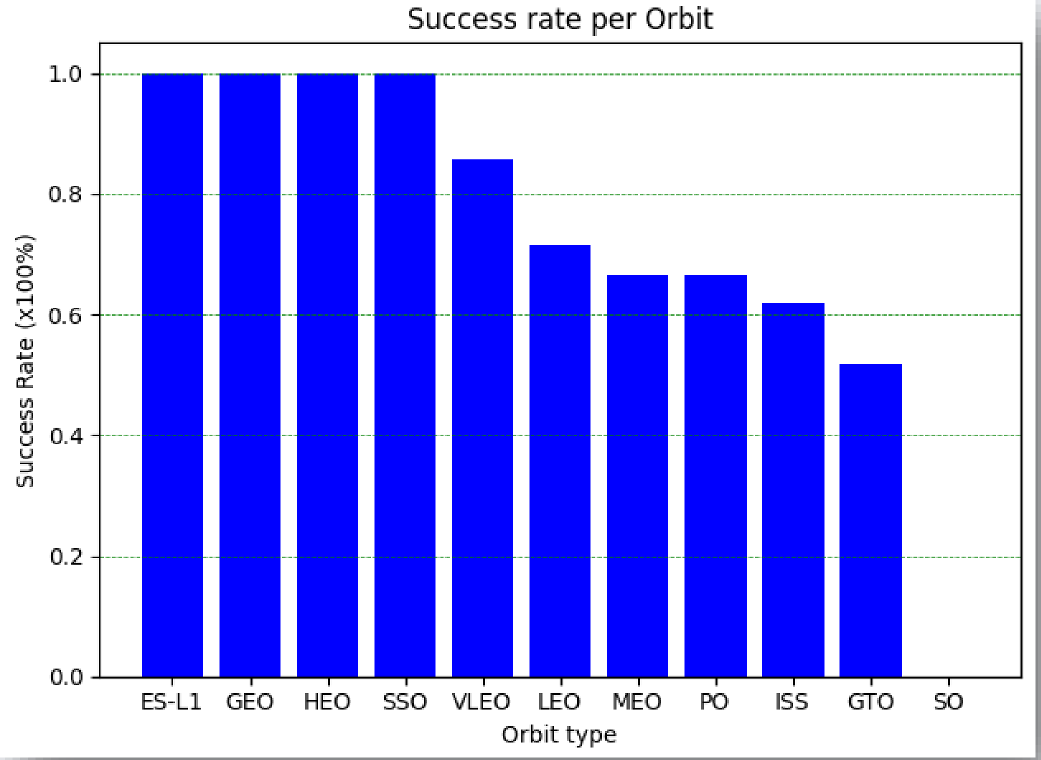
Payload vs. Launch Site



Task 2:

- **Payload mass vs. Launch site scatter plot:**
  - VAFB SLC 4E: limited to 10 Ton payload mass (higher latitude of site implies more energy to place mass in orbit)
  - Cape Canaveral and Kennedy Space center launch site accumulate the lighter payloads (< 8 Ton) and also the heaviest payloads (due to proximity to Equator).

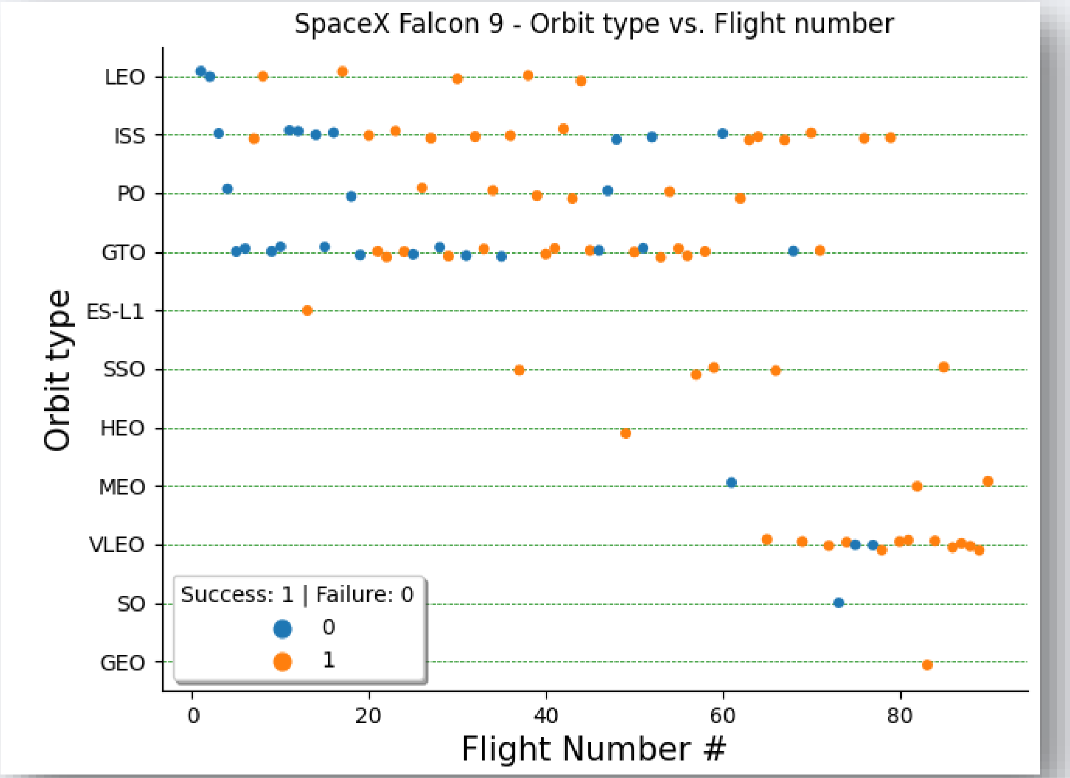
Success Rate vs. Orbit Type



Task 3: Orbit type success rate

- Using this bar plot each orbit type can easily be evaluated w.r.t. success rate.
- Only one SO orbit was done with failer during landing (although SSO “Sun-Synchronous Orbit”, which is the same type, had 100% success)
- GTO “Geosynchronous Orbit” failed to land in more than 40%
- LEO “Low Earth Orbit” failed in more than 20%
- ISS (module sent to the International Space Station) failed to land in almost 20%
- MEO “Middle Earth Orbit” failed in 30% of the times in landing safely back home.

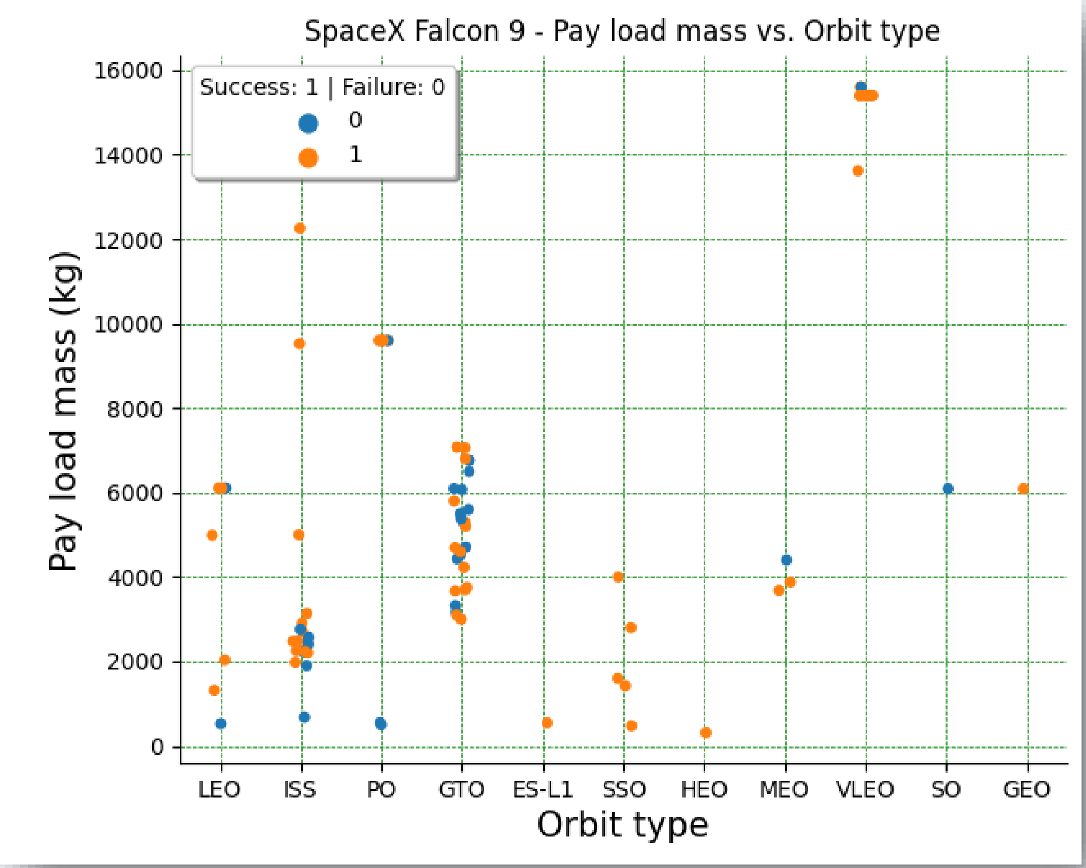
Flight Number vs. Orbit Type



Task 4: relationship between Flightnumber and Orbit Type

- A few first flights were launched to LEO orbits, with higher flightnumber the success is highly correlated
- ISS orbits have been sent continuously (makes sense since the upload of new components/maintenance/provisions are regularly needed at the ISS)
- GTO orbits are also regularly used, with higher incidence in the first 60 flights. However, success is not related to flightnumber, since is occurs regularly as well.
- PO orbits are also regularly used, more spaced in time
- Orbits ES-L1, HEO were barely used
- SO, SSO, MEO and GEO orbits require more energy, these have launched from 40 onwards, were more experience was acquired
- VLEO orbit has high demand from flight number 65 when SpaceX started to launch commercial services for satellites, success is here 100%

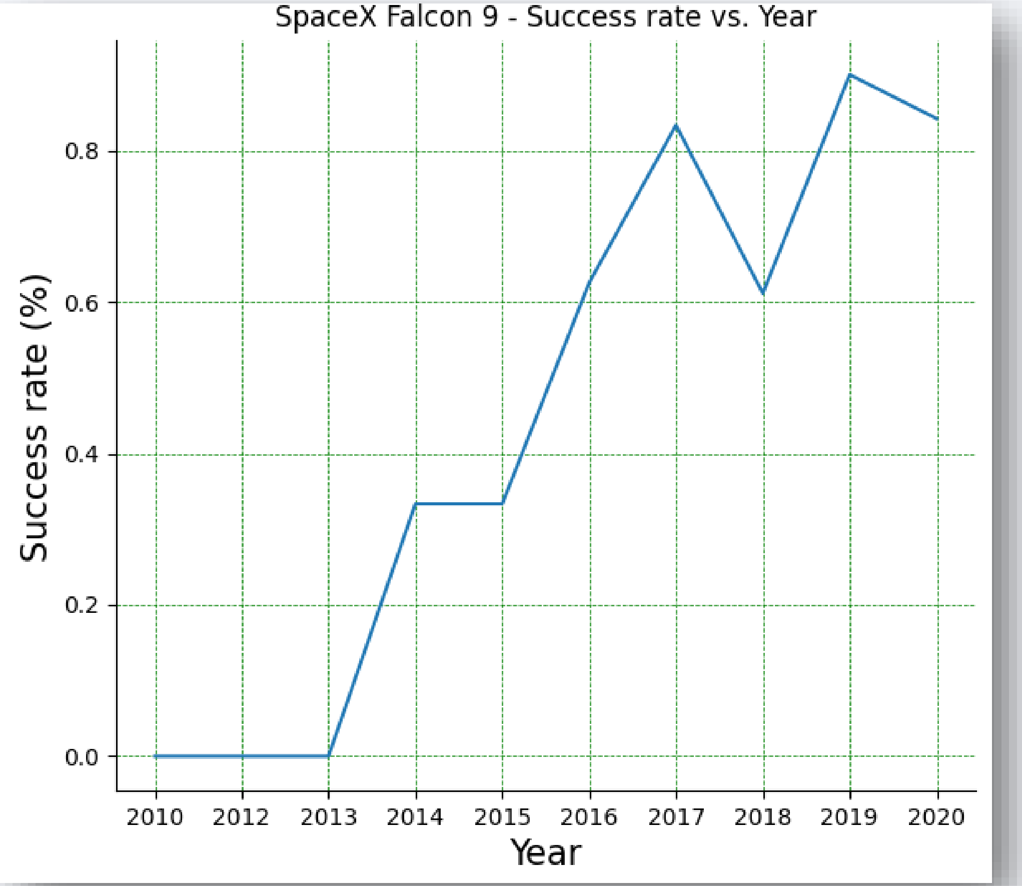
Payload vs. Orbit Type



Task 5: relationship between Payload and Orbit type

- Heavy payloads (> 8 Ton) the success landing rate is mostle guaranteed for Poler, ISS and Very Low Orbits types
- Lighter payload (< 8 Tons) success rate is high for SSO but not for GTO, where failures continue to occur for every mass and flightnumber
  - The Geostationary Transfer Orbit [requires a higher delta-V compared to other orbits] hence recovering the Falcon 9 booster is more complex (is bigger and travelled further wawy). All these launches are done closer to the Equator.

Launch Success Yearly Trend



Task 6: Success rate vs. Year

- For all orbits and Launch pads the success rate keep increasing from 2013, with a 1-year plateau between 2015 and 2016.
- In only 4 years (2013-2017) the success rate increased 50%
- In 2018 occurred more failures during landing which recovered back in 2019. Projection estimation can be discussed in the modeling later, but it is expected that the success rate surpasses 95% from 2021 onwards.

All Launch Site Names

- Query code and result:
  - On database column ‘Launch site’ the names are selected keeping only the uniques ones and sicarding the duplicates, reducing the list therefore to only four.

- **Query (Python):**

```
file_path = 'Module2_EDA_with_SQL_Spacex.csv'
df = pd.read_csv(file_path, sep=',', skiprows=0, encoding=None)
# print(df)
# Task 1: Display the names of the unique launch sites in the space mission
launch_site = df['Launch_Site'].unique()
print("Task 1: Names of the unique launch sites in the space mission")
print(launch_site)
```

- **Query result:**

```
Task 1: Names of the unique launch sites in the space mission
['CCAFS LC-40' 'VAFB SLC-4E' 'KSC LC-39A' 'CCAFS SLC-40’]
```

Launch Site Names Begin with 'KSC'

- Query code and result:
  - On database, rows are kept only if on the column 'Launch site' the name starts with the letters 'KSC'

- **Query (Python):**

```
# Task 2: Display 5 records where launch sites begin with the string 'KSC'
KSC = df[df['Launch_Site'].str.startswith('KSC')]
print("Task 2: Show 5 records where launch sites begin with the string 'KSC'")
print(KSC.head(5))
```

- **Query result:**

```
Task 2: Show 5 records where launch sites begin with the string 'KSC'
         Date Time (UTC) Booster_Version Launch_Site        Payload  PAYLOAD_MASS__KG_      Orbit    Customer Mission_Outcome       Landing _Outcome
29  19-02-2017   14:39:00   F9 FT B1031.1  KSC LC-39A  SpaceX CRS-10               2490  LEO (ISS)  NASA (CRS)         Success  Success (ground pad)
30  16-03-2017   06:00:00     F9 FT B1030  KSC LC-39A    EchoStar 23               5600        GTO    EchoStar         Success            No attempt
31  30-03-2017   22:27:00  F9 FT  B1021.2  KSC LC-39A         SES-10               5300        GTO         SES         Success  Success (drone ship)
32  01-05-2017   11:15:00   F9 FT B1032.1  KSC LC-39A        NROL-76               5300        LEO         NRO         Success  Success (ground pad)
33  15-05-2017   23:21:00     F9 FT B1034  KSC LC-39A  Inmarsat-5 F4               6070        GTO    Inmarsat         Success            No attempt
```

Total Payload Mass

- Query code and result:
  - For rows for which the column ‘Customer’ is ‘NASA (CRS)’ the column value of the Payload mass is summed up. The total sum is given to a variable called *total_mass*

- **Query (Python):**

```
# Task 3: Display the total payload mass carried by boosters launched by NASA (CRS)
total_mass = df[df['Customer'] == 'NASA (CRS)']['PAYLOAD_MASS__KG_'].sum()
print("Task 3: Total mass of NASA (CRS) (kg): ", total_mass)
```

- **Query result:**

```
Task 3: Total mass of NASA (CRS) (kg):  45596
```

Average Payload Mass by F9 v1.1

- Query code and result:
  - For rows where the column ‘Booster_version’ is ‘F9 v1.1’ the column value of the Payload mass considered for an average calculation. The total sum is given to a variable called *avg_mass*

- **Query (Python):**

```
# Task 4: Display average payload mass carried by booster version F9 v1.1
avg_mass = df[df['Booster_Version']=='F9 v1.1']['PAYLOAD_MASS__KG_'].mean()
print("Task 4: Average payload mass of the Booster F9 V1.1 (kg): ", avg_mass)
```

- **Query result:**

```
Task 4: Average payload mass of the Booster F9 V1.1 (kg):  2928.4
```

First Successful Ground Landing Date

- Query code and result:
  - In the database, the row where the column ‘Landing_Outcome’ equals to ‘Success (drone ship)’ is kept, then the value of the column ‘Date’ is taken. In order to give an string object instead of a one-element list, the .iat() function is used to access to the single value.

- **Query (Python):**

```
# Task 5: List the date where the first successful landing outcome in drone ship was achieved.
First_success = df[df['Landing _Outcome'] == 'Success (drone ship)']['Date'].iat[0]
print('Task 5: First Successful landing in drone ship date: ', First_success)
```

- **Query result:**

```
Task 5: First Successful landing in drone ship date:  08-04-2016
```

Successful Drone Ship Landing with Payload between 4000 and 6000

- Query code and result:
  - In the database, the rows where the column 'Landing_Outcome' equals to 'Success (drone ship)' and the column 'Payload mass' values are within 4000 and 6000 kg are kept. On these conditions, the value of the column 'Booster version' is taken, resulting in three Boosters that meet these conditions.

- **Query (Python):**

```
# Task 6: List the names of the boosters which have success in ground pad and have payload mass greater than 4000 but less than 6000
big_success = df[(df['Landing _Outcome']=='Success (ground pad)') & (df['PAYLOAD_MASS__KG_'].between(4000,
6000,inclusive='left'))]['Booster_Version']
print("Task 6: List of boosters which had success in ground pad and have payload mass greater than 4000 but less than 6000")
print(big_success)
```

- **Query result:**

```
Task 6: List of boosters which had success in ground pad and have payload mass greater than 4000 but less than 6000
32    F9 FT B1032.1
40    F9 B4 B1040.1
F9    B4 B1043.1
```

Total Number of Successful and Failure Mission Outcomes

- Query code and result:
  - Rows where the string value of the column ‘Mission_outcome’ contains ‘success’ are given to a variable success, whose dimension (length or number of rows) corresponds to the number of successful missions.
  - Alternatively the be used the count_value() function.
  - Same applies to the landing Failure missions. Giving a total of 100 to 1.

- **Query (Python):**

```
# Task 7: List the total number of successful and failure mission outcomes
success = df[df['Mission_Outcome'].str.contains('Success')]
failure = df[df['Mission_Outcome'].str.contains('Failure')]
print("Task 7: Successful missions:", len(success))
print(success)
print("Task 7: Failure missions:", len(failure))
print(failure)
```

- **Query result:**

```
Task 7: Successful missions: 100
Task 7: Failure missions: 1
```

Boosters Carried Maximum Payload

- Query code and result:
  - Grouping the rows by columns Booster_version and Payload_Mass, the maximum value of the latter is taken. The length (no. of rows) of this selections contains 97 entries. This is due to the large plethora of Boosters versions. A sample of 15 are shown, including the payload mass.

- **Query (Python):**

```
# Task 8: List the names of the booster_versions which have carried the maximum payload mass. Use a subquery
max_boosters = df.groupby(['Booster_Version'])['PAYLOAD_MASS__KG_'].max()
print("Task 8: Booster version names carrying the maximum payload mass. Table length: ", len(max_boosters))
print(max_boosters.head(15))
```

- **Query result:**

```
Task 8: Booster version names carrying the maximum payload mass. Table length:  97
Booster_Version
F9 B4  B1039.2    2647
F9 B4  B1040.2    5384
F9 B4  B1041.2    9600
F9 B4  B1043.2    6460
F9 B4 B1039.1     3310
F9 B4 B1040.1     4990
F9 B4 B1041.1     9600
F9 B4 B1042.1     3500
F9 B4 B1043.1     5000
F9 B4 B1044       6092
F9 B4 B1045.1      362
F9 B4 B1045.2     2697
F9 B5  B1046.1    3600
F9 B5 B1046.2     5800
F9 B5 B1046.3     4000
```

2015 Launch Records

- Query code and result:
  - First the two new columns are created for Year and Month: using *datetime* function on the Date column the standard tuple time is used, and then extracted the month and year into the new columns.
  - For the new dataframe subset, the rows of a successful landing on ground pad and year 2017 are selected, resulting in six landings.

- **Query (Python):**

```
# Task 9: List the records which will display the month names, succesful landing_outcomes in ground pad ,booster versions, launch_site for the months in year 2017
df['Date_clean'] = pd.to_datetime(df['Date']) #convert to date tuple
df['Month'] = df['Date_clean'].dt.strftime('%b') # extract month
df['Year'] = df['Date_clean'].dt.strftime('%Y') # extract year
df_selection = df[(df['Landing _Outcome'] == 'Success (ground pad)') & (df['Year'] == '2017')]
```

- **Query result:**

```
Task 9: List of month names, succesful landing_outcomes in ground pad, booster versions, launch_site for the months in year 2017
    Year Month      Landing _Outcome Booster_Version   Launch_Site
29  2017   Feb  Success (ground pad)   F9 FT B1031.1    KSC LC-39A
32  2017   Jan  Success (ground pad)   F9 FT B1032.1    KSC LC-39A
34  2017   Mar  Success (ground pad)   F9 FT B1035.1    KSC LC-39A
38  2017   Aug  Success (ground pad)   F9 B4 B1039.1    KSC LC-39A
40  2017   Jul  Success (ground pad)   F9 B4 B1040.1    KSC LC-39A
44  2017   Dec  Success (ground pad)  F9 FT  B1035.2  CCAFS SLC-40
```

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

- Query code and result:
    - The rows where column of "Landing Outcome" contains "success" and the values of column "Date" are between the two given and subselected. Additionally, sorted descending (no need for re-indexing). Total is 10 landings within those dates.

- **Query (Python):**

```
# Task 10: Rank the count of successful landing_outcomes between the date 2010-06-04 and 2017-03-20 in descending order.
success_landings = df[(df['Landing _Outcome'].str.contains('Success')) & (df['Date_clean'].between('2010-06-04', '2017-03-20'))].sort_values(by='Date_clean', ascending=False)
print("Task 10: Rank the count of successful landing_outcomes between the date 2010-06-04 and 2017-03-20 in descending order.")
print(success_landings)
```

- **Query result:**

Task 10: Rank the count of successful landing_outcomes between the date 2010-06-04 and 2017-03-20 in descending order.

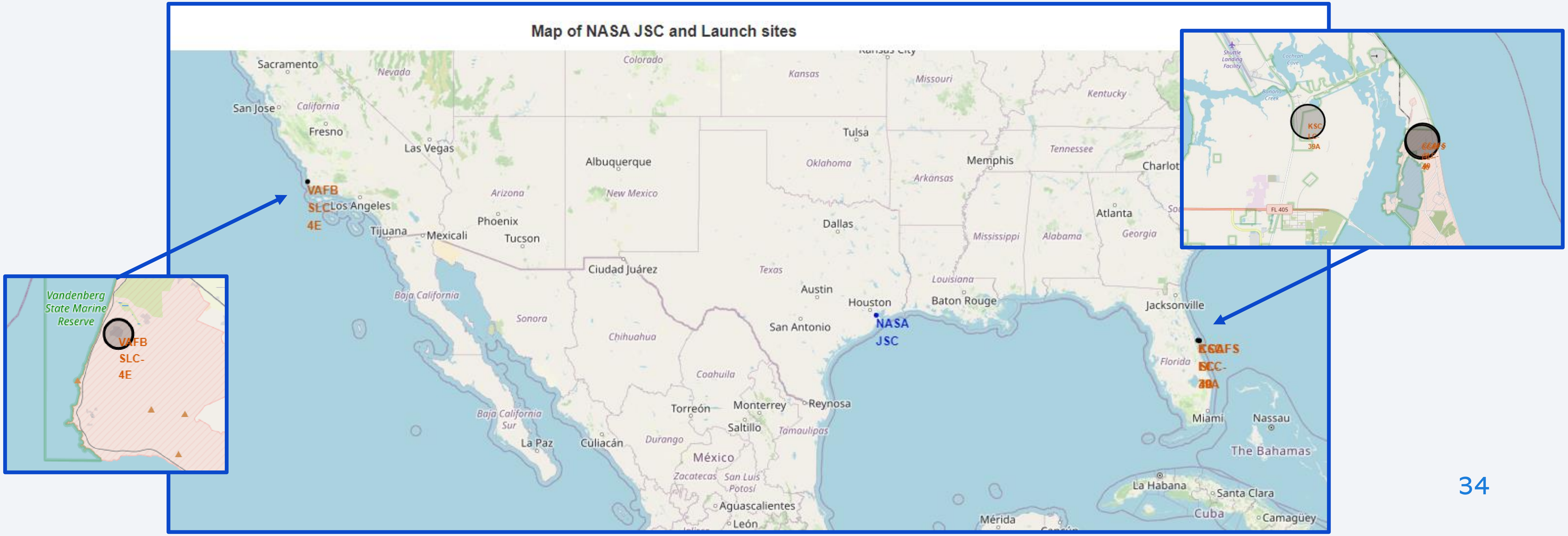
	Date	Time (UTC)	Booster_Version	Launch_Site	Payload	PAYLOAD_MASS__KG_	Orbit	Customer	Mission_Outcome	Landing _Outcome	Date_clean	Month	Year
34	03-06-2017	21:07:00	F9 FT B1035.1	KSC LC-39A	SpaceX CRS-11	2708	LEO (ISS)	NASA (CRS)	Success	Success (ground pad)	2017-03-06	Mar	2017
29	19-02-2017	14:39:00	F9 FT B1031.1	KSC LC-39A	SpaceX CRS-10	2490	LEO (ISS)	NASA (CRS)	Success	Success (ground pad)	2017-02-19	Feb	2017
28	14-01-2017	17:54:00	F9 FT B1029.1	VAFB SLC-4E	Iridium NEXT 1	9600	Polar LEO	Iridium Communications	Success	Success (drone ship)	2017-01-14	Jan	2017
32	01-05-2017	11:15:00	F9 FT B1032.1	KSC LC-39A	NROL-76	5300	LEO	NRO	Success	Success (ground pad)	2017-01-05	Jan	2017
27	14-08-2016	05:26:00	F9 FT B1026	CCAFS LC-40	JCSAT-16	4600	GTO	SKY Perfect JSAT Group	Success	Success (drone ship)	2016-08-14	Aug	2016
22	08-04-2016	20:43:00	F9 FT B1021.1	CCAFS LC-40	SpaceX CRS-8	3136	LEO (ISS)	NASA (CRS)	Success	Success (drone ship)	2016-08-04	Aug	2016
26	18-07-2016	04:45:00	F9 FT B1025.1	CCAFS LC-40	SpaceX CRS-9	2257	LEO (ISS)	NASA (CRS)	Success	Success (ground pad)	2016-07-18	Jul	2016
23	06-05-2016	05:21:00	F9 FT B1022	CCAFS LC-40	JCSAT-14	4696	GTO	SKY Perfect JSAT Group	Success	Success (drone ship)	2016-06-05	Jun	2016
24	27-05-2016	21:39:00	F9 FT B1023.1	CCAFS LC-40	Thaicom 8	3100	GTO	Thaicom	Success	Success (drone ship)	2016-05-27	May	2016
19	22-12-2015	01:29:00	F9 FT B1019	CCAFS LC-40	OG2 Mission 2 11 Orbcomm-OG2 satellites	2034	LEO	Orbcomm	Success	Success (ground pad)	2015-12-22	Dec	2015

Section 3

Launch Sites Proximities Analysis

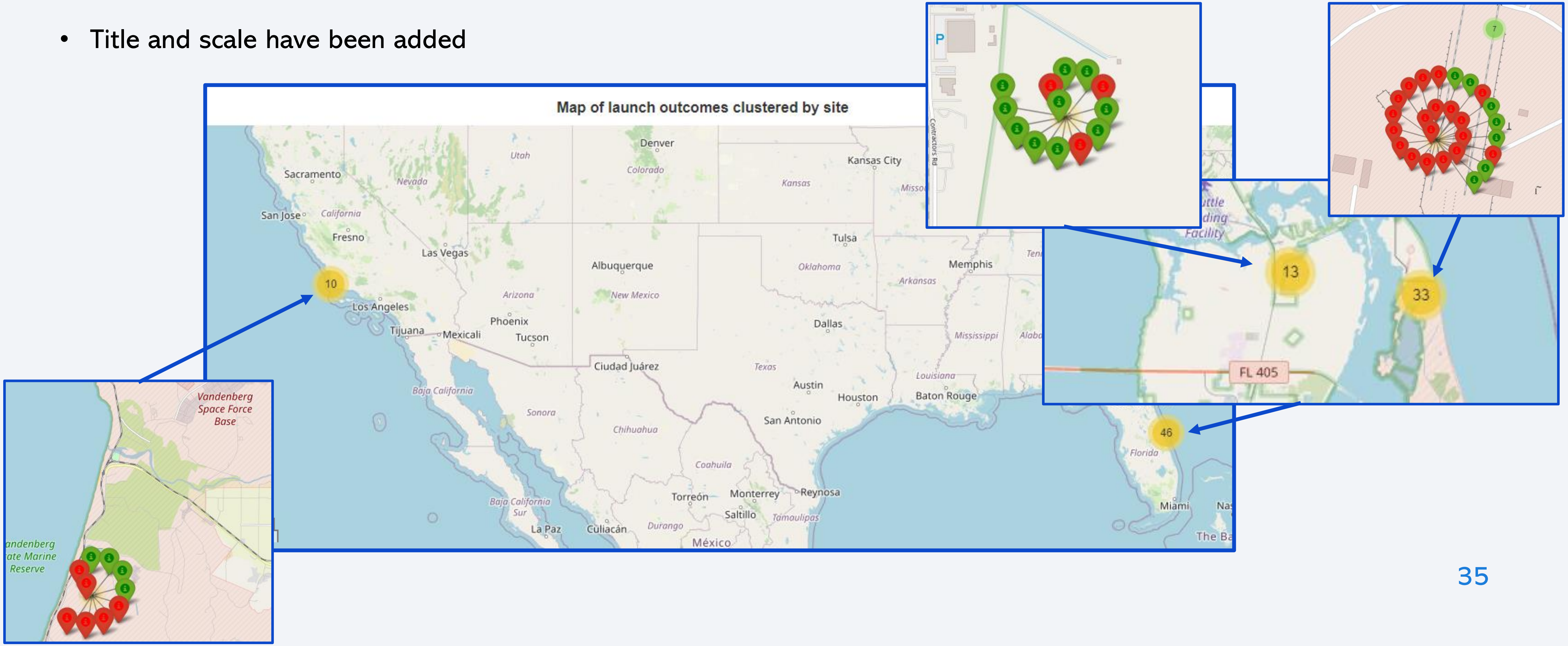
Folium Map for NASA JSC and Launch sites

- Folium map includes the NASA JSC and Launch sites in West and East coast. Zooming in and out is allowed (see small screenshots in boxes) and all geographical items are represented.
- Title has been added



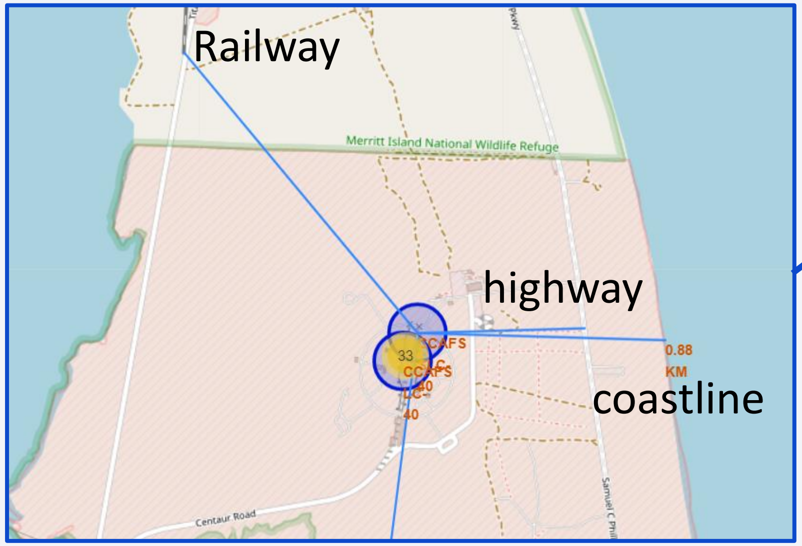
Folium Map of launch outcomes clustered

- Folium map includes the launch outcomes clustered in each launch site. Zooming in allows to see the outcomes per location (4) with an icon marked in outcome color.
- Title and scale have been added



Folium Map with Poly lines to specific coordinates

- The Interactive Folium map is enriched with the Latitud and Longitude coordinates shown where the mouse is located.
- Using these coordinates, markers can be assigned to them and polylines connecting two markers. Examples are coastline, nearest railway, nearest highway and nearest city, see screenshots.
- A function to calculate the absolute distance between two coordinates (geodesic curve) gives the following output:
  - Distance of Launch site CCAFS SLC-40 to coast is: 0.877 km
  - Distance of Launch site CCAFS SLC-40 to closest railway is: 1.291 km (this is terminal railway to transport rockets)
  - Distance of Launch site CCAFS SLC-40 to closest highway is: 0.589 km (within the Cape Canaveral Space Force Station)
  - Distance of Launch site CCAFS SLC-40 to closest city (Cabo Canaveral) is: 19.735 km
- The Python code and Interactive map in .html format are located in the GitHub link https://github.com/GuillermoDC/Python5



Section 5

Predictive Analysis (Classification)

Classification Accuracy

- For each classification model the Gridsearch CV method tunes the hyperparameters to find the optimum selection that enhances the accuracy.
- A manual inspection of the Hyperparameters has been done to have a complete overview (see tables).

Classification model 1: Logistic Regression

- Using optimization solver [limited-memory BFGS]()
- classification accuracy is 0.846 (rnd 2) obtained with two solvers

Classification model 2: Support Vector Machine (SVM)

- Best kernel found: sigmoid
- classification accuracy is 0.84821

Classification model 3: Decision Tree

- Best Hyperparameter: Max depth = 18
- classification Gini criterion accuracy is 0.8875

Classification model 4: K-Nearest Neighbors

- Best algorithm: auto (all of them gave same results), max N neighbors chosen: 10, more than 10 did not improved the accuracy.
- classification accuracy is 0.84821

Logistic Regression method

variable	*variable*	
Solver	**Penalty**	**Accuracy**
lbfgs	none	0.71071
	l2	0.84643
liblinear	l1	0.83393
	l2	0.80536
newton-cg	none	0.69643
	l2	0.84643
saga	elasticnat	*Not converged*
	l1	0.86071
	l2	0.84643
	none	1.75179

Support Vector Machine

variable		
kernel	**gamma**	**Accuracy**
rbf	0.001	0.80714
sigmoid	0.0316	0.84821
linear	0.001	0.82143

Decision Tree

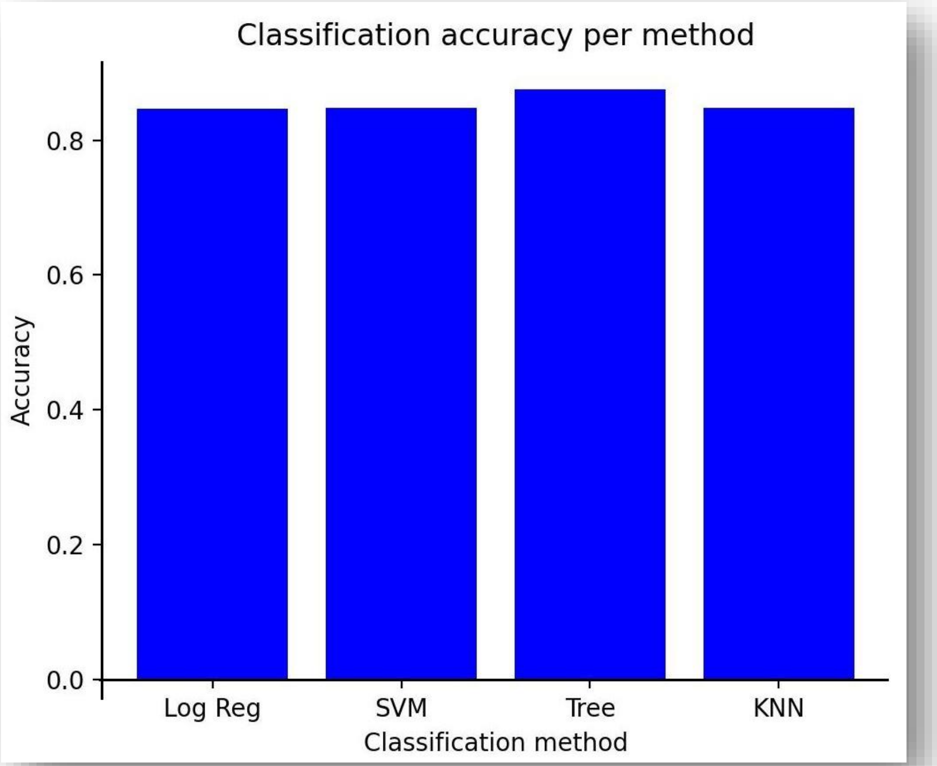
variable			
criterion	**max. depth**	**splitter**	**Accuracy**
log_loss	4	best	0.8893
gini	18	random	0.8875

K-Nearest Neighbor

variable		
algorithm	**N neighbors**	**Accuracy**
auto	10	0.84821
ball_tree	10	0.84821
kd_tree	10	0.84821
brute force search	10	0.84821
auto	10*	0.84821
	** 12 was allowed*	

Classification Accuracy II

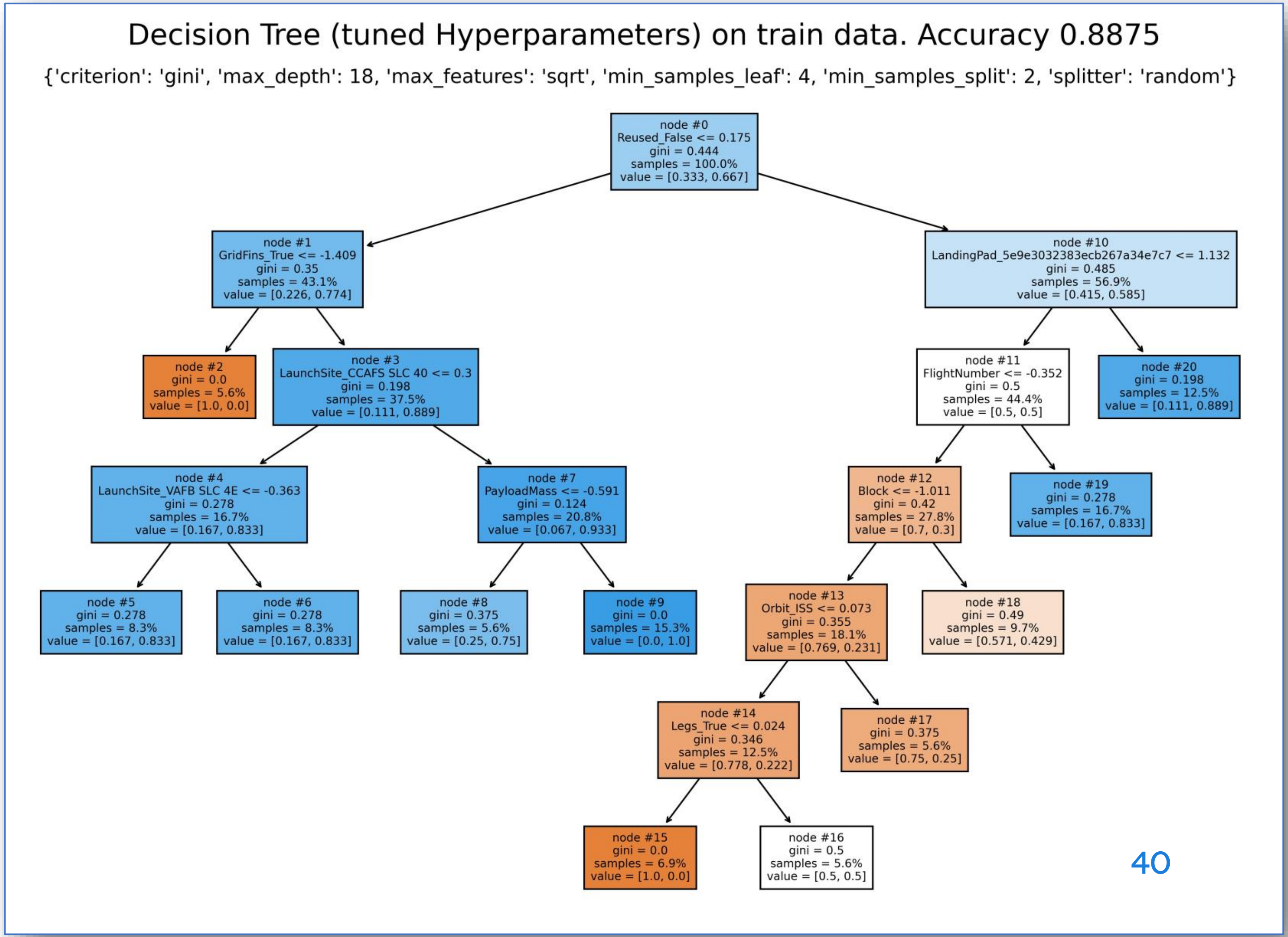
- Classification accuracy of the successful Falcon 9 launches is similar for the 4 evaluated models, being slightly higher for the Decision Tree.
- Reason is the very low datapoints on the used dataset and hgomogeneity of the values



Classification Accuracy – Decision Tree

- Decision tree on trained data has been plotted
  - The darker the color, the more pure (gini closer to 0) that node is until it arrives to a leaf (no chance of misclassification, end of branch)
  - Tree has 20 nodes, 3 leafs (Gini = 0.0)
  - Algorithm tries all all possible boundaries between data and chooses the one that gives the lowest Gini impuriy (when using Gini index as metric)
  - Gini = chance of misclassification
    - Gini Test, where the p_i are the difference samples values in each node

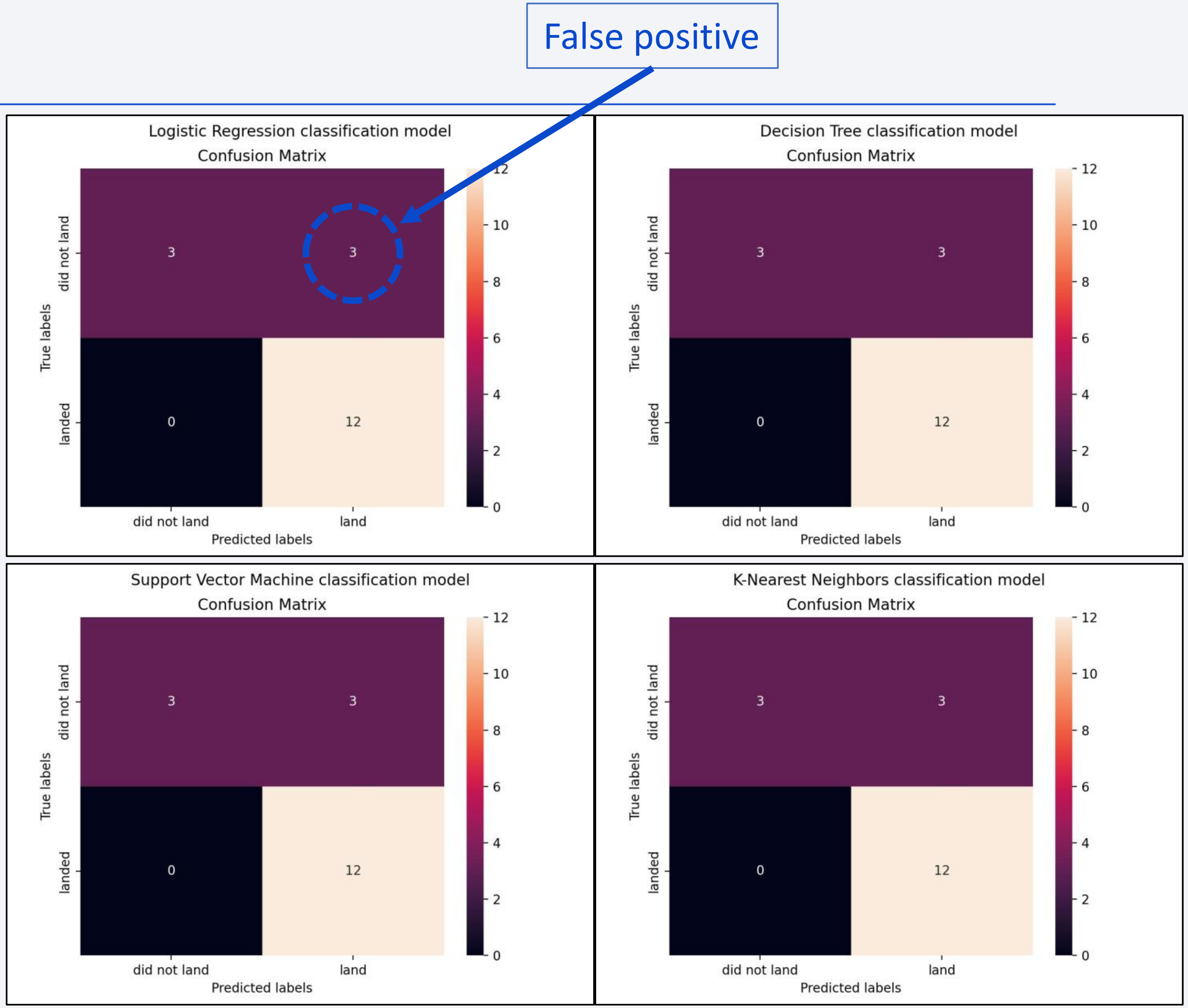
$$I_G(p) = 1 - \sum_{i=1}^{J} p_i^2$$



Confusion Matrix

- Since dataset is not quite large, all confusion matrices are equal.

- Confusion Matrix indicates that the models:
  - Can distinguish between the different classes
  - Gives several false positives: 3 launches have been classified as landed but did not actually land (upper right sector)

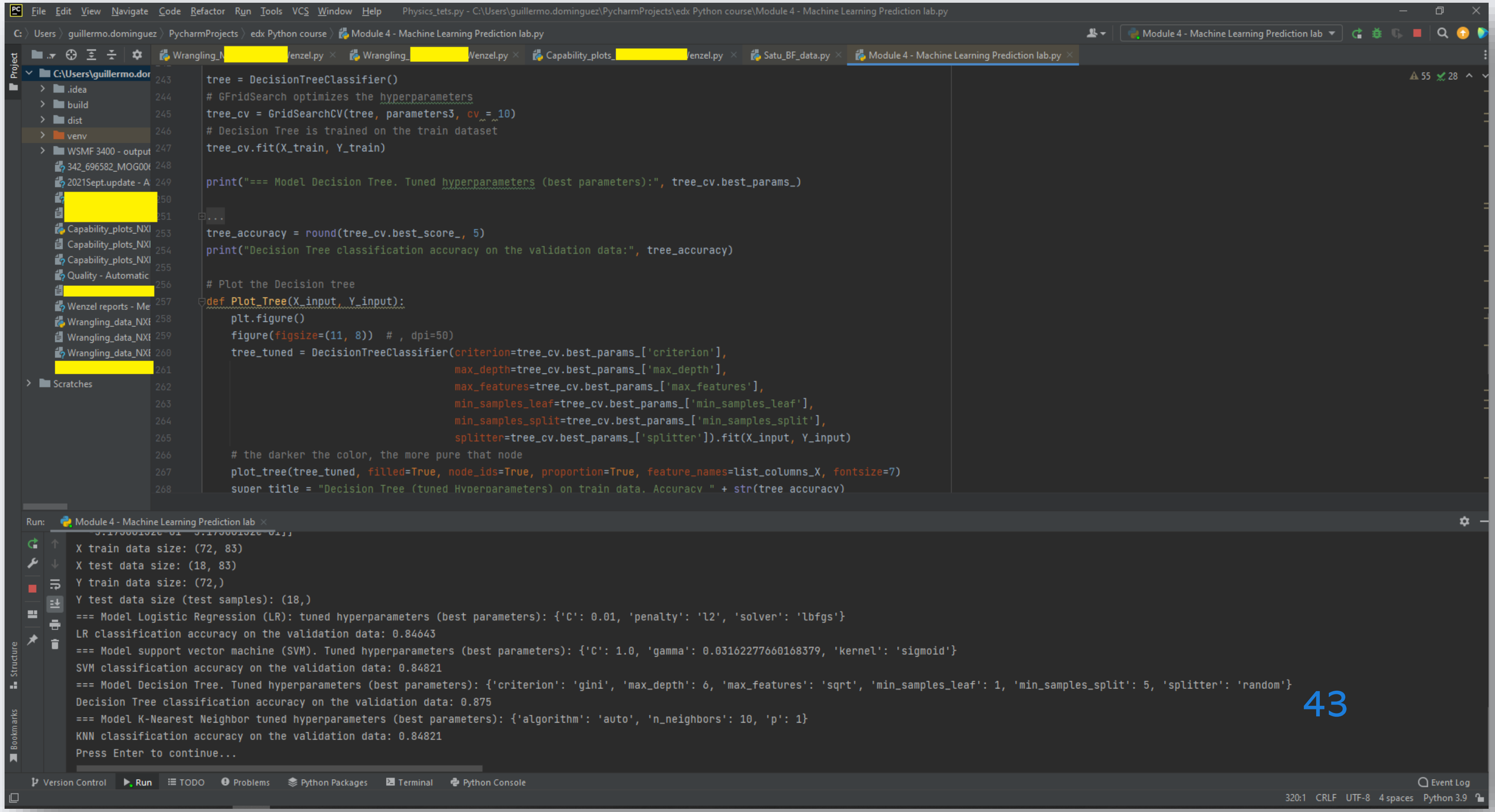


Conclusions

- Data from API SpaceX has been processed to prepara for analysis, useful Pandas and Numpy commands have been refreshed from pervious courses of *IBM Python Data Science* and implemented
- During exploratory analysis the goals were settled and relevant parameters considered
- FoilMap tool is avery useful interactive tool to visualize data
- Classification Models tools provided are able to predict the success rate of the SpaceX Falcon 9 launch. All of them with an accuracy of ~ 0.85, slightly higher for Decision Tree method (accuracy of 0.89)
- *I wil implement the tools learned in this Capstone project on the data analysis and SPC for my professional work: analysis of the Coordinate Measuring Machine data output of extra large high accurate milled aluminium frames within the semiconductor industry. See already a step done at this respect in my personal LinkedIn account in this link.*

Appendix

- For this project, Python v. 3.9 code has been written using IDE PyCharm 2021.3 (Community Edition). *Author subjective opinion is that this IDE is more readable than IBM Watson Studio Jupiter Notebook.*



Thank you!