



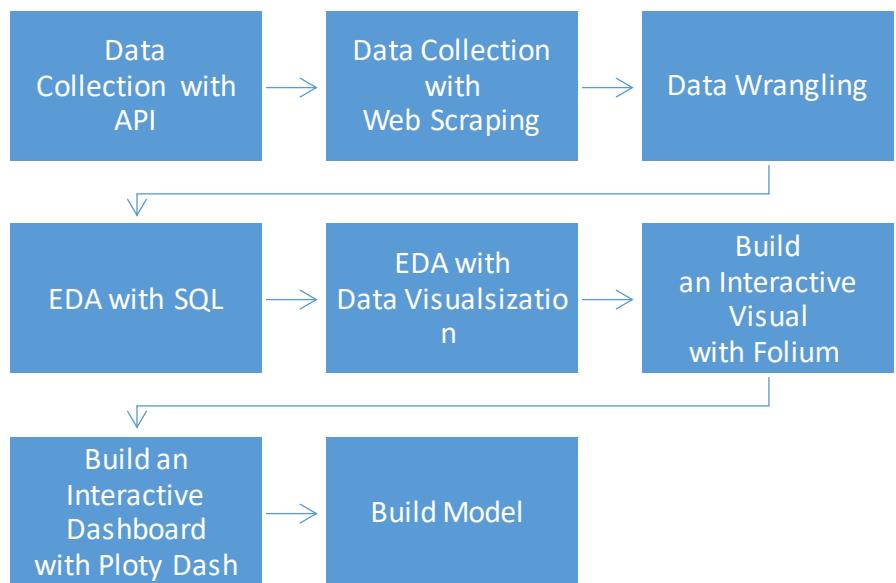
SpaceX
Maryam
19/2/2024

OUTLINE

- Executive Summary
- Introduction
- Methodology
- Results
 - EDA with Visualization Results
 - EDA with SQL Results
 - Interactive Map with Folium Results
 - Plotly Dashboard Resluts
 - Predictive Analysis (Classification) Results
- Discussion
- Conclusion
- Appendix



EXECUTIVE SUMMARY



INTRODUCTION

- **Background :**

The commercial space age is here, companies are making space travel affordable for everyone. SpaceX advertises Falcon 9 rocket launches on its website with a cost of 62 million dollars; other providers cost upwards of 165 million dollars each, much of the savings is because SpaceX can reuse the first stage.

- **Problem to Solve:**

- Determine if the first stage will land, and determine the cost of a launch.
- Determine if SpaceX will reuse the first stage.

METHODOLOGIES



Data Collection (API)

- we were working with SpaceX launch data that is gathered from an API, specifically the SpaceX REST API. This API will give us data about launches, such as rocket used, payload delivered, launch specifications, landing specifications, and landing outcome.
- The link to the notebook :
- <https://github.com/marAlsolami/Coursera-/blob/main/jupyter-labs-spacex-data-collection-api.ipynb>

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

```
]:#Global_variables
BoosterVersion = []
PayloadMass = []
Orbit = []
LaunchSite = []
Outcome = []
Flights = []
Gridfins = []
Reused = []
Legs = []
LandingPad = []
Block = []
ReusedCount = []
Serial = []
Longitude = []
Latitude = []
```

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

```
] : BoosterVersion
]: []
]: 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'])
```

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

```
] : # Call getBoosterVersion
getBoosterVersion(data)
```

the list has now been update

```
] : BoosterVersion[0:4]
]: ['Falcon 1', 'Falcon 1', 'Falcon 1', 'Falcon 1']
```

Data Collection(Web Scraping)

- We Perform web scraping to collect Falcon 9 historical launch records from a Wikipedia page titled List of Falcon 9 and Falcon Heavy launches.
- We do Web scrap to extract Falcon 9 launch records with BeautifulSoup library. Extract a Falcon 9 launch records HTML table from Wikipedia. Parse the table and convert it into a Pandas data frame
- The link to the notebook :
- <https://github.com/marAlsolami/Coursera-/blob/main/jupyter-labs-webscraping.ipynb>

• Refrence

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

1: Request the Falcon9 Launch Wiki page from its URL

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

```
requests.get() method with the provided static_url  
in the response to a object  
se = requests.get(static_url).text
```

a BeautifulSoup object from the HTML response

```
BeautifulSoup() to create a BeautifulSoup object from a response text content  
BeautifulSoup(response, 'html.parser')
```

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

```
soup.title attribute  
= soup.title  
  
title)  
List of Falcon 9 and Falcon Heavy launches - Wikipedia</title>
```

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

We want to collect all relevant column names from the HTML table header

to find all tables on the wiki page first. If you need to refresh your memory about BeautifulSoup, please do of this lab

```
the find_all function in the BeautifulSoup object, with element type 'table'  
in the result to a list called 'html_tables'  
tables = soup.find_all('table')
```

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

```
print the third table and check its content  
launch_table = html_tables[2]  
first_launch_table)
```

Data Wrangling

we were perform some Exploratory Data Analysis (EDA) to find some patterns in the data and determine what would be the label for training supervised models. Such as Calculate the number of launches on each site, Calculate the number and occurrence of each orbit, and Calculate the number and occurrence of mission outcome of the orbits.

- The link to the notebook :

<https://github.com/marAlsolami/Coursera-/blob/main/labs-jupyter-spacex-Data%20wrangling.ipynb>

```
Outcome      0.000000
Flights      0.000000
GridFins     0.000000
Reused       0.000000
Legs         0.000000
LandingPad   28.888889
Block        0.000000
ReusedCount  0.000000
Serial        0.000000
Longitude    0.000000
Latitude     0.000000
dtype: float64
```

Identify which columns are numerical and categorical:

`df.dtypes`

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

TASK 1: Calculate the number of launches on each site

The data contains several Space X launch facilities: [Cape Canaveral Space](#) Launch Complex 40 VAFB SLC 4 Complex 4E (SLC-4E), Kennedy Space Center Launch Complex 39A KSC LC 39A .The location of each Laur

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

Use the method `value_counts()` on the column `LaunchSite` to determine the number of launches or

EDA with SQL

We perform Exploratory Data Analysis and Feature Engineering using SQL queries .

- The link to the notebook :

https://github.com/marAlsolami/Coursera-/blob/main/jupyter-labs-eda-sql-coursera_sqlite.ipynb

Tasks

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

Note: If the column names are in mixed case enclose it in double quotes For Example "Landing_Pad"

Task 1

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

```
[0]: %sql select DISTINCT Launch_Site from SPACEXTABLE;
```

* sqlite:///my_data1.db
Done.

```
[0]: Launch_Site
```

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'

```
[0]: %sql select * from SPACEXTABLE WHERE Launch_Site LIKE '%CCA%'  
limit 5
```

* sqlite:///my_data1.db
Done.

Date	Time (UTC)	Booster_Version	Launch_Site	Payload	PAYLOAD_MASS__K
2010-06-04	18:45:00	F9 v1.0 B0003	CCAFS LC-40	Dragon Spacecraft Qualification Unit	
2010-12-08	15:43:00	F9 v1.0 B0004	CCAFS LC-40	Dragon demo flight C1, two CubeSats, barrel of Brouere cheese	
2012-05-22	7:44:00	F9 v1.0 B0005	CCAFS LC-40	Dragon demo flight C2	

EDA with Data Visualization

- We Perform exploratory Data Analysis and Feature Engineering using 'Pandas' and 'Matplotlib' libraries.
- The link to the notebook :

<https://github.com/marAlsolami/Coursera-/blob/main/jupyter-labs-eda-dataviz.ipynb.jupyterlite.ipynb>

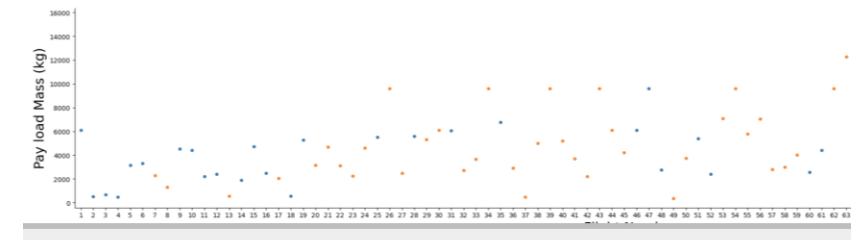


```
dataset_part_2_csv = io.BytesIO((await resp.arrayBuffer()).to_py())
df=pd.read_csv(dataset_part_2_csv)
df.head(5)
```

	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins	Reused	Legs	Landi
0	1	2010-06-04	Falcon 9	6104.959412	LEO	CCAFS SLC 40	None None	1	False	False	False	
1	2	2012-05-22	Falcon 9	525.000000	LEO	CCAFS SLC 40	None None	1	False	False	False	
2	3	2013-03-01	Falcon 9	677.000000	ISS	CCAFS SLC 40	None None	1	False	False	False	
3	4	2013-09-29	Falcon 9	500.000000	PO	VAFB SLC 4E	False Ocean	1	False	False	False	
4	5	2013-12-03	Falcon 9	3170.000000	GTO	CCAFS SLC 40	None None	1	False	False	False	

First, let's try to see how the `FlightNumber` (indicating the continuous launch attempts.) and `Payload` variables would affect the outcome. We can plot out the `FlightNumber` vs. `PayloadMass` and overlay the outcome of the launch. We see that as the flight number increases, the payload mass is also important; it seems the more massive the payload, the less likely the first stage will return.

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

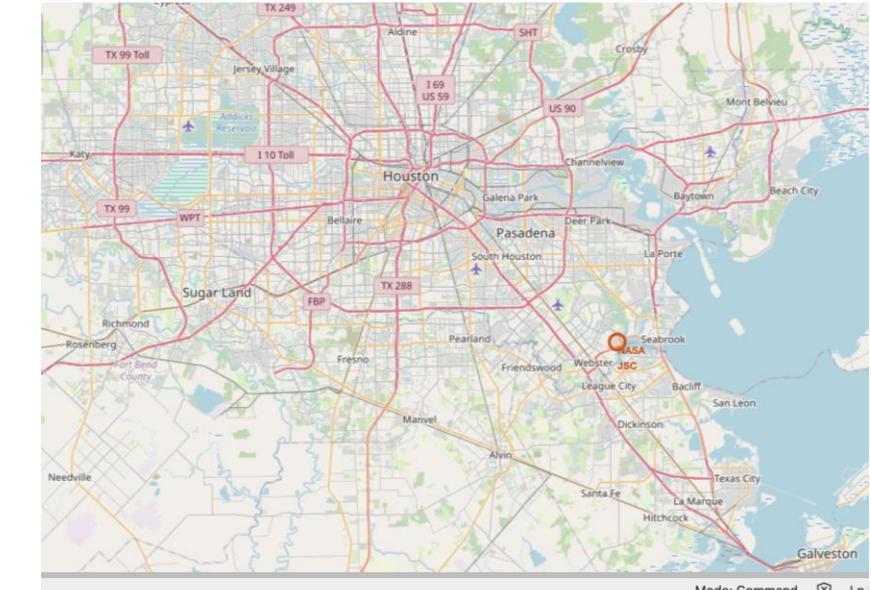


Build an Interactive Visual with Folium

We analyze launch site geo and proximities with Folium. We will first mark the launch site locations and their close proximities on an interactive map. Then, we can explore the map with those markers and try to discover any patterns from them. Finally, we should be able to explain how to choose an optimal launch site.

- The link to the notebook :

https://github.com/marAlsolami/Coursera-/blob/main/lab_jupyter_launch_site_location.jupyterlite.ipynb



The image shows an interactive map of the Houston metropolitan area and surrounding regions. A red circle marks the location of NASA Johnson Space Center (JSC) in Webster, Texas. A blue circle marks the coordinate of NASA Johnson Space Center's coordinate with a marker icon showing its name. The map displays various roads, highways (I-10, I-69, TX 99, TX 288), and local neighborhoods. Labels for cities like Houston, Webster, Seabrook, and Galveston are visible. The map is overlaid on a satellite or terrain background.

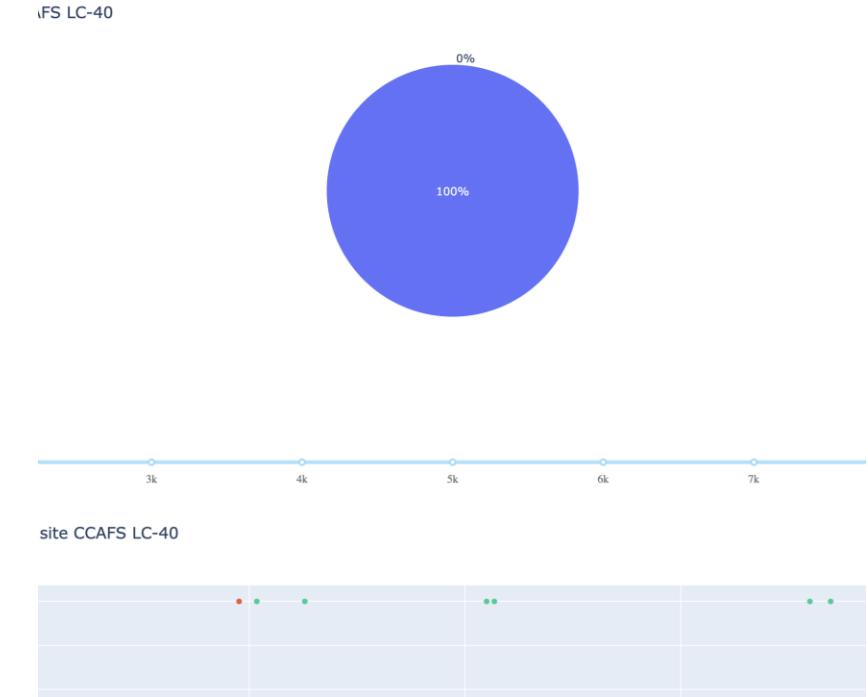
```
set a blue circle at NASA Johnson Space Center's coordinate with a popup label showing its name
create a blue circle at NASA Johnson Space Center's coordinate with a icon showing its name
marker = folium.map.Marker(
    nasa_coordinate,
    # Create an icon as a text label
    icon=DivIcon(
        icon_size=(20,20),
        icon_anchor=(0,0),
        html='<div style="font-size: 12; color:#d35400;"><b>%s</b></div>' % 'NASA JSC',
    )
)
_map.add_child(circle)
_map.add_child(marker)
```

Build an Interactive Dashboard with Ploty Dash

- We build a dashboard application with the Python Plotly Dash package. This dashboard application contains input components such as a dropdown list and a range slider to interact with a pie chart and a scatter point chart. After the dashboard is built, we can use it to find more insights from the SpaceX dataset more easily
- The link to the notebook :

[https://github.com/marAlsolami/Coursera-
/blob/main/Hands-
on%20Lab%3A%20Build%20an%20Interactive%20Dashbo
ard%20with%20Ploty%20Dash](https://github.com/marAlsolami/Coursera/blob/main/Hands-on%20Lab%3A%20Build%20an%20Interactive%20Dashboard%20with%20Ploty%20Dash)

SpaceX Launch Records Dashboard



Build Model

we will build a machine learning pipeline to predict if the first stage of the Falcon 9 lands successfully. We test Logistic Regression, Support Vector machines, Decision Tree Classifier, and K-nearest neighbors.

- The link to the notebook :

https://github.com/marAlsolami/Coursera-/blob/main/SpaceX_Machine_Learning_Prediction_Part_5.jupyterlite.ipynb

ASK 2

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

```
students get this  
ansform = preprocessing.StandardScaler()  
= transform.fit transform(X)
```

› split the data into training and testing data using the function `train_test_split`. The training data is divided into validation data, a series of models are trained and hyperparameters are selected using the function `GridSearchCV`.

ASK 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. Assign the labels to the following variables.

```
_train, X_test, Y_train, Y_test
```

```
train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=2)
```

~~...so we only have 19 test samples~~

Results



Results

- EDA with visualization results
- EDA with SQL Results
- Interactive map with Folium results
- Plotly Dash dashboard results
- Predictive analysis (classification) results



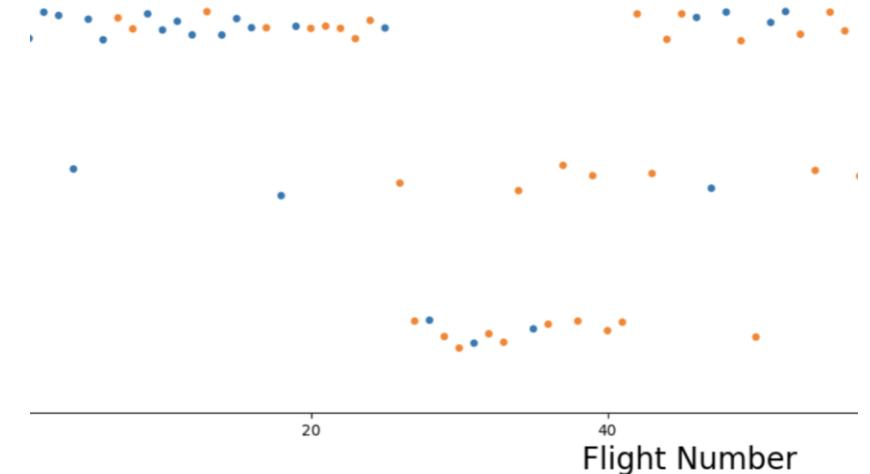
EDA with Visualization Results

Visualize the relationship between Flight Number and Launch Site

The relationship between Flight Number and Launch Site

It is recommended to plot FlightNumber vs LaunchSite, set the parameter x parameter to FlightNum

```
# Create a scatter plot showing the relationship between Flight Number and Launch Site
# Set the x-axis to be Flight Number and y axis to be the launch site, and
# Set the hue to be Class
# Set the aspect ratio to 3
# Set the font size to 20
# Set the title to be "Flight Number vs Launch Site"
# Set the subtitle to be "The relationship between Flight Number and Launch Site"
```



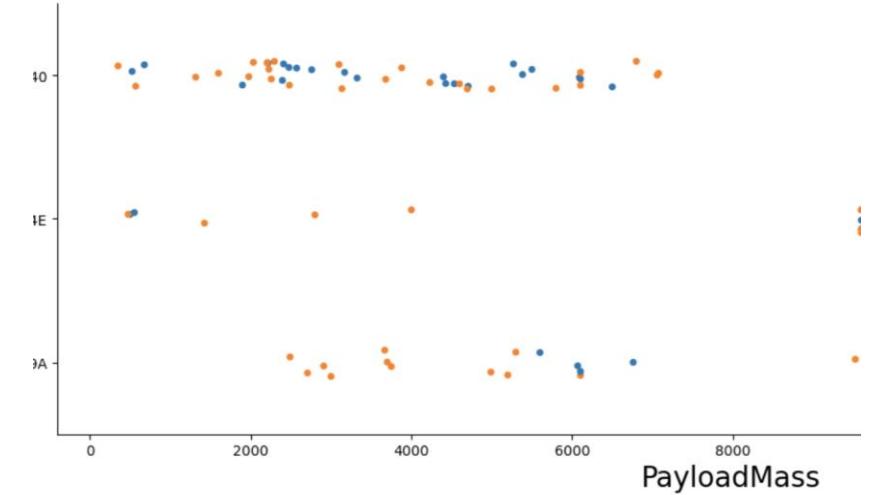
EDA with Visualization Results

Visualize the relationship between Payload and Launch Site

Visualize the relationship between Payload and Launch Site

Observe if there is any relationship between launch sites and their payload mass.

```
scatter point chart with x axis to be Pay Load Mass (kg) and y axis to be the launch site  
x="PayloadMass",y="LaunchSite",hue='Class',data=df, aspect = 3)  
PayloadMass",fontsize=20)  
Launch Site",fontsize=20)
```



serve Payload Vs. Launch Site scatter point chart you will find for the VAFB-SLC launchsite there are no

EDA with Visualization Results

- Visualize the relationship between success rate of each orbit type

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

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

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

```
INT use groupby method on Orbit column and get the mean of Class column
it_x = df.groupby('Orbit').mean()

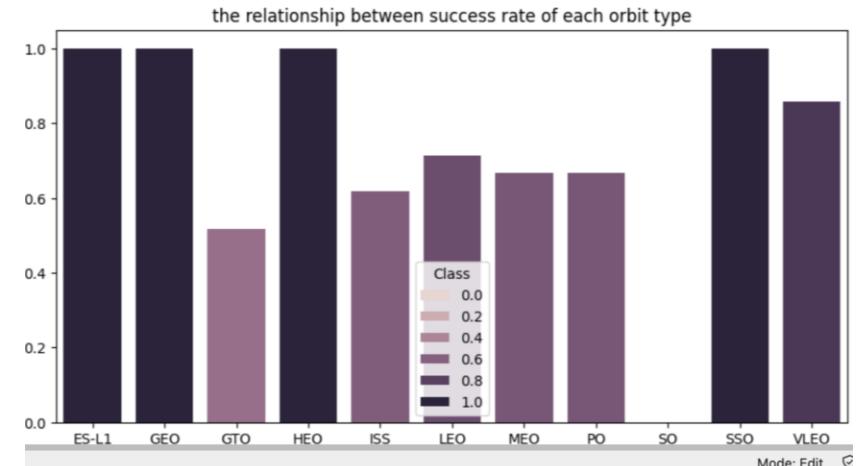
y= python>>> df.groupby('Orbit').mean()
          Orbit      Class
          GEO      1.000000
          HEO      1.000000
          LEO      0.714286
          MEO      0.666667
          PO       0.666667
          SSO      1.000000
          VLEO     0.857143
          ES-L1    1.000000
          GTO      0.523810
          ISS      0.625000
          Mode: Edit
```

```
it_x = df.groupby('Orbit').mean()

# Python 3.7+ syntax
# it_x = df.groupby('Orbit').mean(numeric_only=True)

# Python 3.6 and earlier
# it_x = df.groupby('Orbit').mean(numeric_only=True)
# FutureWarning: The default value of numeric_only in DataFrameGroupBy.mean() is deprecated. It will default to False. Either specify numeric_only or select only columns which should be valid for the function.

# Create a bar chart
plt.figure(figsize=(10, 5))
barplot(x="Orbit", y="Class", data=it_x, hue='Class')
xlabel("Orbit")
ylabel("Class")
title("the relationship between success rate of each orbit type")
show()
```



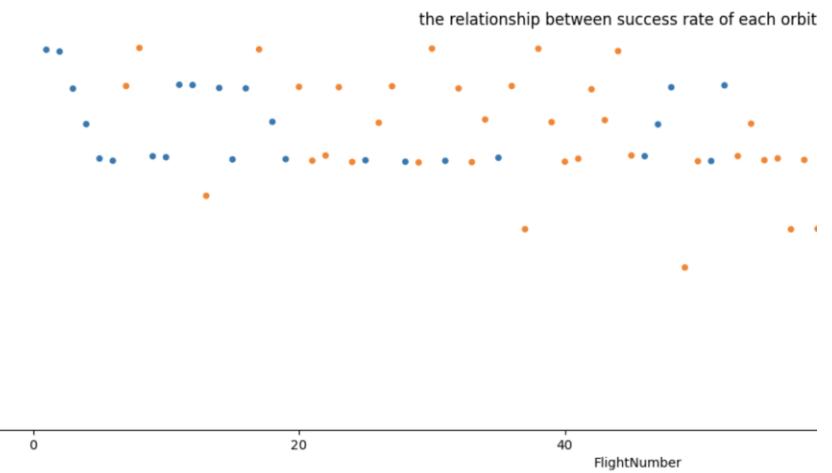
EDA with Visualization Results

Visualize the relationship between FlightNumber and Orbit type

4: Visualize the relationship between FlightNumber and Orbit type

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

```
# scatter point chart with x axis to be FlightNumber and y axis to be the Orbit, and hue to  
plt.figure(figsize = (10, 5))  
plot(x="FlightNumber",y="Orbit",data=df, hue='Class',aspect = 3)  
del("FlightNumber")  
del("Orbit")  
lef("the relationship between success rate of each orbit type")  
n()
```



I'd see that in the LEO orbit the Success appears related to the number of flights; on the other hand, there seem

Mode

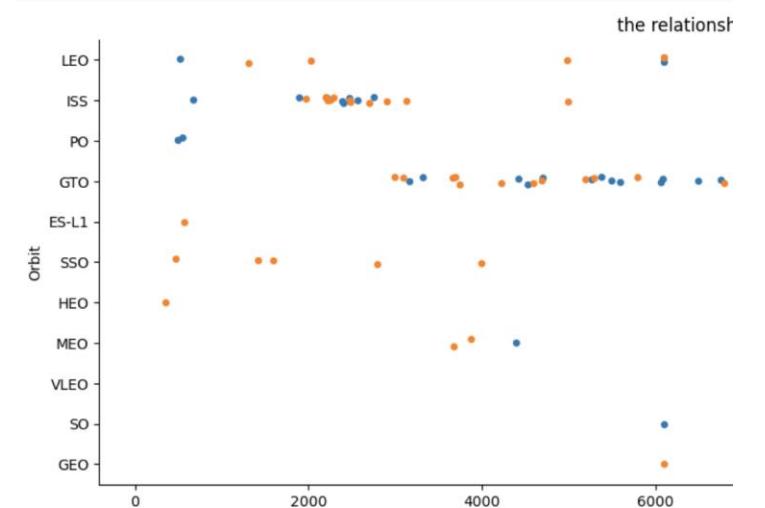
EDA with Visualization Results

Visualize the relationship between Payload and Orbit type

TASK 5: Visualize the relationship between Payload and Orbit

Similarly, we can plot the Payload vs. Orbit scatter point charts to reveal the relations

```
[62]: # Plot a scatter point chart with x axis to be Payload and y axis to be Orbit
sns.catplot(x="PayloadMass",y="Orbit",data=df, hue='Class',aspect = 3)
plt.xlabel("PayloadMass")
plt.ylabel("Orbit")
plt.title("the relationship between Payload and Orbit type")
plt.show()
```

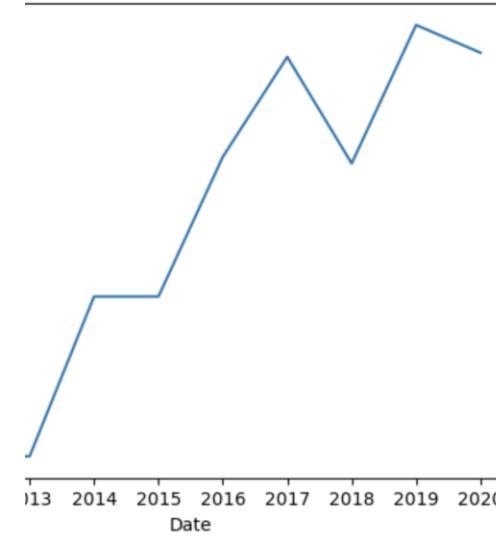


EDA with Visualization Results

Visualize the launch success yearly trend

```
th x axis to be the extracted year and y axis to be the success rate  
roupy(by="Date").mean()  
index(inplace=True)  
ar['Date'], average_by_year['Class'])
```

```
9073ace0>:2: FutureWarning: The default value of numeric_only in DataFrameGroupBy is  
either specify numeric_only or select only columns which should be valid for the  
.groupby(by="Date").mean()
```



Mode: Edit

EDA with Visualization Results

- Create dummy variables to categorical columns

7: Create dummy variables to categorical columns

```
function get_dummies and features dataframe to apply OneHotEncoder to the column  
features_one_hot , display the results using the method head. Your result dataframe must
```

```
Use get_dummies() function on the categorical columns  
s_one_hot = pd.get_dummies(features, columns=['Orbit', 'LaunchSite', 'LandingP  
s_one_hot
```

htNumber	PayloadMass	Flights	GridFins	Reused	Legs	Block	ReusedCount	Orbit_
1	6104.959412	1	False	False	False	1.0	0	
2	525.000000	1	False	False	False	1.0	0	
3	677.000000	1	False	False	False	1.0	0	
4	500.000000	1	False	False	False	1.0	0	
5	3170.000000	1	False	False	False	1.0	0	
...
86	15400.000000	2	True	True	True	5.0	2	
87	15400.000000	3	True	True	True	5.0	2	
88	15400.000000	6	True	True	True	5.0	5	
89	15400.000000	3	True	True	True	5.0	2	
90	3681.000000	1	True	False	True	5.0	0	

: 80 columns

EDA with Visualization Results

Cast all numeric columns to float64

TASK 8: Cast all numeric columns to float64

Now that our `features_one_hot` datafram only contains numbers cas

```
[84]: # HINT: use astype function  
features_one_hot = features_one_hot.astype('float64')  
features_one_hot
```

	FlightNumber	PayloadMass	Flights	GridFins	Reused	Legs	BI
0	1.0	6104.959412	1.0	0.0	0.0	0.0	0.0
1	2.0	525.000000	1.0	0.0	0.0	0.0	0.0
2	3.0	677.000000	1.0	0.0	0.0	0.0	0.0
3	4.0	500.000000	1.0	0.0	0.0	0.0	0.0
4	5.0	3170.000000	1.0	0.0	0.0	0.0	0.0
...
85	86.0	15400.000000	2.0	1.0	1.0	1.0	1.0
86	87.0	15400.000000	3.0	1.0	1.0	1.0	1.0
87	88.0	15400.000000	6.0	1.0	1.0	1.0	1.0
88	89.0	15400.000000	3.0	1.0	1.0	1.0	1.0
89	90.0	3681.000000	1.0	1.0	0.0	1.0	1.0

90 rows × 80 columns

EDA with SQL Results

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

Task 1

Display the names of the unique launch site

```
[0]: %sql select DISTINCT Launch_Site fr
```

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

Done.

```
[0]: Launch_Site
```

CCAFS LC-40

VAFB SLC-4E

KSC LC-39A

CCAFS SLC-40

EDA with SQL Results

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

:h sites begin with the string 'CCA'

```
XTABLE WHERE Launch_Site LIKE '%CCA%'
```

Version	Launch_Site	Payload	PAYLOAD_MAS
0 B0003	CCAFS LC-40	Dragon Spacecraft Qualification Unit	
0 B0004	CCAFS LC-40	Dragon demo flight C1, two CubeSats, barrel of Brouere cheese	
0 B0005	CCAFS LC-40	Dragon demo flight C2	
0 B0006	CCAFS LC-40	SpaceX CRS-1	
0 B0007	CCAFS LC-40	SpaceX CRS-2	

EDA with SQL Results

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

Task 3

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

```
[26]: %sql select SUM(PAYLOAD_MASS__KG_) from SPACEXTABI  
* sqlite:///my_data1.db  
Done.  
[26]: SUM(PAYLOAD_MASS__KG_)  
45596
```

Task 4

Display average payload mass carried by booster version F

```
[27]: %sql select AVG(PAYLOAD_MASS__KG_) from SPACEXTABI  
* sqlite:///my_data1.db  
Done.  
[27]: AVG(PAYLOAD_MASS__KG_)  
2928.4
```

EDA with SQL Results

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

Task 3

Display the total payload mass carried by boosters launched

```
[26]: %sql select SUM(PAYLOAD_MASS__KG_) from SPACEXTABI  
* sqlite:///my_data1.db  
Done.  
[26]: SUM(PAYLOAD_MASS__KG_)  
-----  
45596
```

Task 4

Display average payload mass carried by booster version F

```
[27]: %sql select AVG(PAYLOAD_MASS__KG_) from SPACEXTABI  
* sqlite:///my_data1.db  
Done.  
[27]: AVG(PAYLOAD_MASS__KG_)  
-----  
2928.4
```

EDA with SQL Results

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

Task 5

List the date when the first succesful landing outcome in gro

Hint:Use min function

```
: %sql select MIN (Date) from SPACEXTABLE where Landing_Pad = 1  
* sqlite:///my_data1.db  
Done.
```

MIN (Date)

2015-12-22

```
: #####
```

```
: %sql select * from SPACEXTABLE WHERE Launch_Site LIKE '%Pad 1%'  
* sqlite:///my_data1.db  
Done.
```

Time
1d (additional servers needed) Python Idle Mem: 419.33 / 6144.0

EDA with SQL Results

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

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

```
[65]: %%sql select Booster_Version,PAYLOAD_MASS__KG_,Landing_Outcome  
from SPACEXTABLE  
where Landing_Outcome == 'Success (drone ship)' AND PAYLOAD_MASS__KG_ between '4000' AND '6000'  
* sqlite:///my_data1.db  
Done.  
[65]: 

| Booster_Version | PAYLOAD_MASS__KG_ | Landing_Outcome      |
|-----------------|-------------------|----------------------|
| F9 FT B1022     | 4696              | Success (drone ship) |
| F9 FT B1026     | 4600              | Success (drone ship) |
| F9 FT B1021.2   | 5300              | Success (drone ship) |
| F9 FT B1031.2   | 5200              | Success (drone ship) |


```

Task 7

List the total number of successful and failure mission outcomes

```
[93]: %%sql select count(Mission_Outcome) as success  
from SPACEXTABLE where Mission_Outcome like 'Success%'  
* sqlite:///my_data1.db  
Done.  
[93]: success  
100  
  
[95]: %%sql select count(Mission_Outcome) as fail  
from SPACEXTABLE where Mission_Outcome like 'Failure%'  
* sqlite:///my_data1.db  
Done.  
[95]: fail  
1
```

EDA with SQL Results

List the total number of successful and failure mission outcomes

Task 6

List the names of the boosters which have success in drone ship and have payload mass greater than 4000 kg.

```
[65]: %%sql select Booster_Version, PAYLOAD_MASS__KG_, Landing_Outcome  
      from SPACEXTABLE  
      where Landing_Outcome == 'Success (drone ship)' AND PAYLOAD_MASS__KG_ between '4000'  
      * sqlite:///my_data1.db  
Done.  
[65]: 

| Booster_Version | PAYLOAD_MASS__KG_ | Landing_Outcome      |
|-----------------|-------------------|----------------------|
| F9 FT B1022     | 4696              | Success (drone ship) |
| F9 FT B1026     | 4600              | Success (drone ship) |
| F9 FT B1021.2   | 5300              | Success (drone ship) |
| F9 FT B1031.2   | 5200              | Success (drone ship) |


```

Task 7

List the total number of successful and failure mission outcomes

```
[93]: %%sql select count(Mission_Outcome) as success  
      from SPACEXTABLE where Mission_Outcome like 'Success%'  
      * sqlite:///my_data1.db  
Done.  
[93]: success  
100  
  
[95]: %%sql select count(Mission_Outcome) as fail  
      from SPACEXTABLE where Mission_Outcome like 'Failure%'  
      * sqlite:///my_data1.db  
Done.  
[95]: fail  
1
```

EDA with SQL Results

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

Task 8

List the names of the booster_versions which have carried the maximum payload ma

```
: %sql select Booster_Version  
from SPACEXTABLE  
where PAYLOAD_MASS_KG_ = (select Max(PAYLOAD_MASS_KG_ from SPACEXTAB  
* sqlite:///my_data1.db  
(sqlite3.OperationalError) near "from": syntax error  
[SQL: select Booster_Version  
from SPACEXTABLE  
where PAYLOAD_MASS_KG_ = (select Max(PAYLOAD_MASS_KG_ from SPACEXTAB  
(Background on this error at: http://sqlalche.me/e/e3q8)
```

```
.]: %sql select * from SPACEXTABLE limit 5;  
* sqlite:///my_data1.db  
Done.  
.]:  


| Date       | Time (UTC) | Booster_Version | Launch_Site | Payload                                                      |
|------------|------------|-----------------|-------------|--------------------------------------------------------------|
| 2010-06-04 | 18:45:00   | F9 v1.0 B0003   | CCAFS LC-40 | Dragon Spacecraft Qualificati<br>U                           |
| 2010-12-08 | 15:43:00   | F9 v1.0 B0004   | CCAFS LC-40 | Dragon demo flight C1, t<br>CubeSats, barrel of Brou<br>chee |
| 2012-05-22 | 7:44:00    | F9 v1.0 B0005   | CCAFS LC-40 | Dragon demo flight                                           |
| 2012-10-08 | 0:35:00    | F9 v1.0 B0006   | CCAFS LC-40 | SpaceX CRS                                                   |
| 2013-03-01 | 15:10:00   | F9 v1.0 B0007   | CCAFS LC-40 | SpaceX CRS                                                   |


```

EDA with SQL Results

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.

ask 9

st the records which will display the month names, failure landing_outcomes in drone ship ,booster versions, launch_site for the months in year 2015.
ote: SQLite does not support monthnames. So you need to use substr(Date, 6,2) as month to get the months year.

```
sqlite> select substr(Date, 6,2) as month,substr(Date,0,5)='2015' as year  
      from SPACEXTABLE ;  
: sqlite:///my_data1.db  
month year  
06 0  
12 0  
05 0  
10 0  
03 0  
09 0  
12 0  
01 0  
04 0  
07 0  
08 0  
09 0  
09 0  
01 1  
02 1  
03 1  
04 1  
04 1  
06 1  
12 1  
01 0
```

ditional servers needed) Python | Idle Mem: 365.07 / 6144.00 MB Mode: Command ↵ Ln 1, Col 32 English (United States)

EDA with SQL Results

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.

Task 10

Rank the count of landing outcomes (such as Failure (drone ship) or Success (ground

```
3]: %%sql select * from SPACEXTABLE  
where Date between '2010-06-04'and '2017-03-20'  
group by Landing_Outcome;  
  
* sqlite:///my_data1.db  
Done.
```

	Date	Time (UTC)	Booster_Version	Launch_Site	Payload
1]	2014-04-18	19:25:00	F9 v1.1	CCAFS LC-40	SpaceX CRS-3
2]	2015-01-10	9:47:00	F9 v1.1 B1012	CCAFS LC-40	SpaceX CRS-5
3]	2010-06-04	18:45:00	F9 v1.0 B0003	CCAFS LC-40	Dragon Spacecraft Qualification Unit
4]	2012-05-22	7:44:00	F9 v1.0 B0005	CCAFS LC-40	Dragon demo flight C2
5]	2015-06-28	14:21:00	F9 v1.1 B1018	CCAFS LC-40	SpaceX CRS-7
6]	2016-04-08	20:43:00	F9 FT B1021.1	CCAFS LC-40	SpaceX CRS-8
7]	2015-12-22	1:29:00	F9 FT B1019	CCAFS LC-40	OG2 Mission 2 11 Orbcomm OG2 satellites
8]	2013-09-29	16:00:00	F9 v1.1 B1003	VAFB SLC-4E	CASSIOPE

ized (additional servers needed) Python | Idle Mem: 365.10 / 6144.00 MB Mode: Co

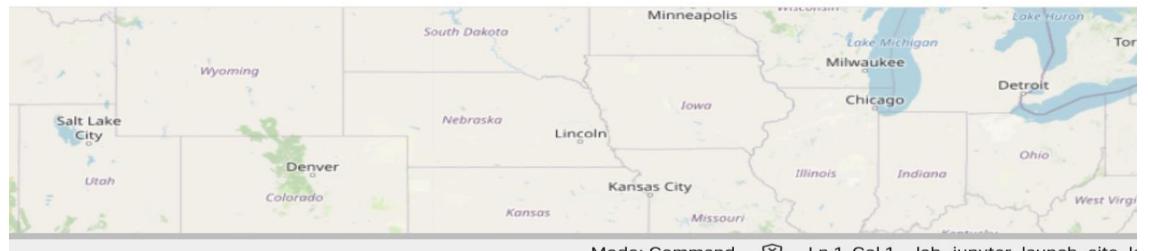
Interactive Map with Folium Results

Mark all launch sites on a map

```
_coordinate, zoom_start=5)
# object based on its coordinate (Lat, Long) values. In addition, add Launch site name as a popup label
son Space Center's coordinate with a popup label showing its name
terrows():
Long']]
s=1000, color="#000000", fill=True).add_child(folium.Popup(row['Launch Site'])).add_to(site_map)
con=DivIcon(icon_size=(20,20),icon_anchor=(0,0),
e="font-size: 12; color:#d35400;"><b>%s</b></div>' % row['Launch Site'], )).add_to(site_map)
```



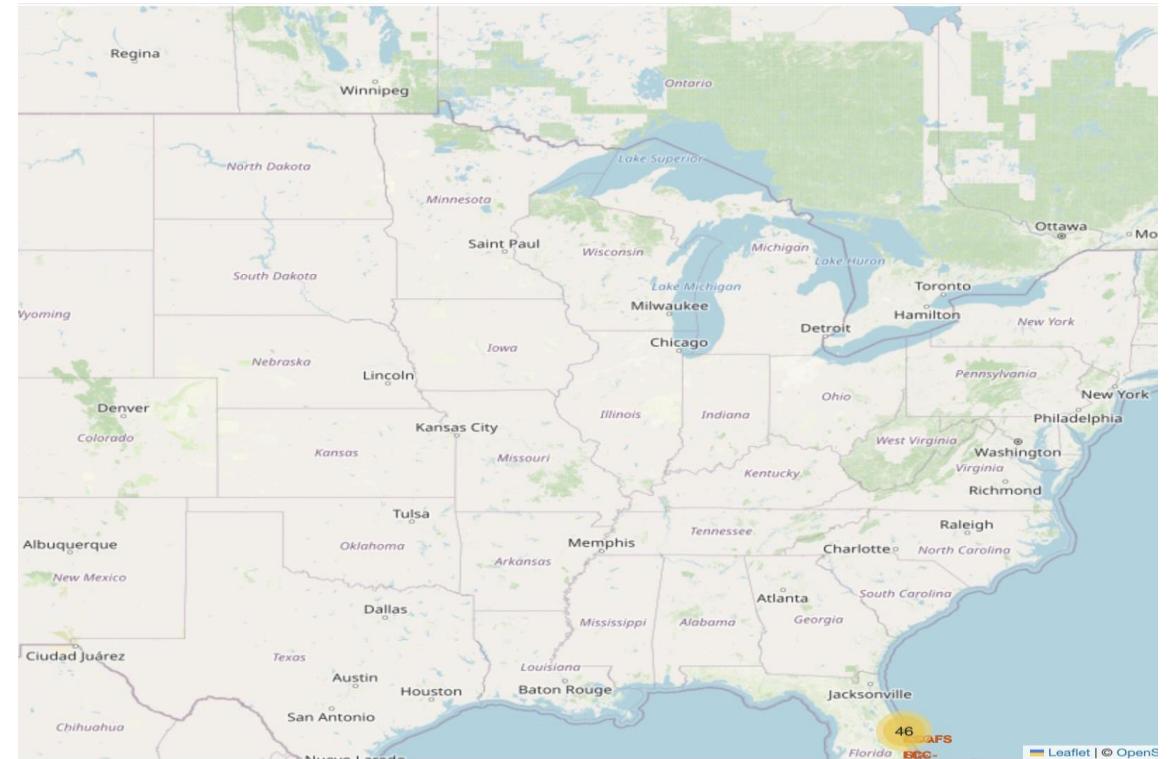
Map should look similar to the following:



Mode: Command Ln 1, Col 1 lab_jupyter_launch_site_k

Interactive Map with Folium Results

Mark the success/failed launches for each site on the map



screenshots:



Interactive Map with Folium Results

Calculate the distances between a launch site to its proximities

```
[19]: # 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
# for example
# distance_marker = folium.Marker(
#     coordinate,
#     icon=DivIcon(
#         icon_size=(20,20),
#         icon_anchor=(0,0),
#         html='<div style="font-size: 12; color:#d35400;"><b>%s</b></div>' % "{:10.2f} KM".
#     )
# Given coordinates
launch_site_lat = 28.56341
launch_site_lon = -80.56744

# Closest coastline point
coastline_lat = 28.56342
coastline_lon = -80.56744

# Calculating distance
distance_coastline = calculate_distance(launch_site_lat, launch_site_lon, coastline_lat, co

# Create a map centered on the launch site
m = folium.Map(location=[launch_site_lat, launch_site_lon], zoom_start=15)

# Create a marker for the launch site
folium.Marker(
    location=[launch_site_lat, launch_site_lon],
    icon=folium.Icon(color='red', icon='rocket', prefix='fa')
).add_to(m)

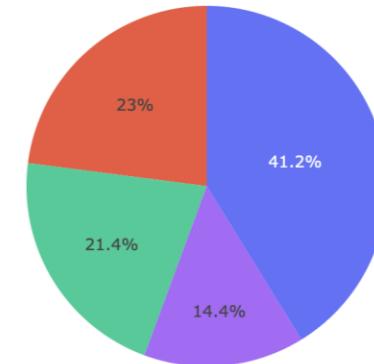
# Create a marker for the coastline point and add the distance
distance_marker = folium.Marker(
    location=[coastline_lat, coastline_lon],
    icon=DivIcon(
        icon_size=(20, 20),
        icon_anchor=(0, 0),
        html='<div style="font-size: 20; color:#d35400;"><b>%s</b></div>' % "{:10.2f} KM".f
    )
)

distance_marker.add_to(m)
```

Plotly Dash dashboard Results

SpaceX Launch Records Dashboard

From Plotly dashboard we find that is the most suitable launch sites.

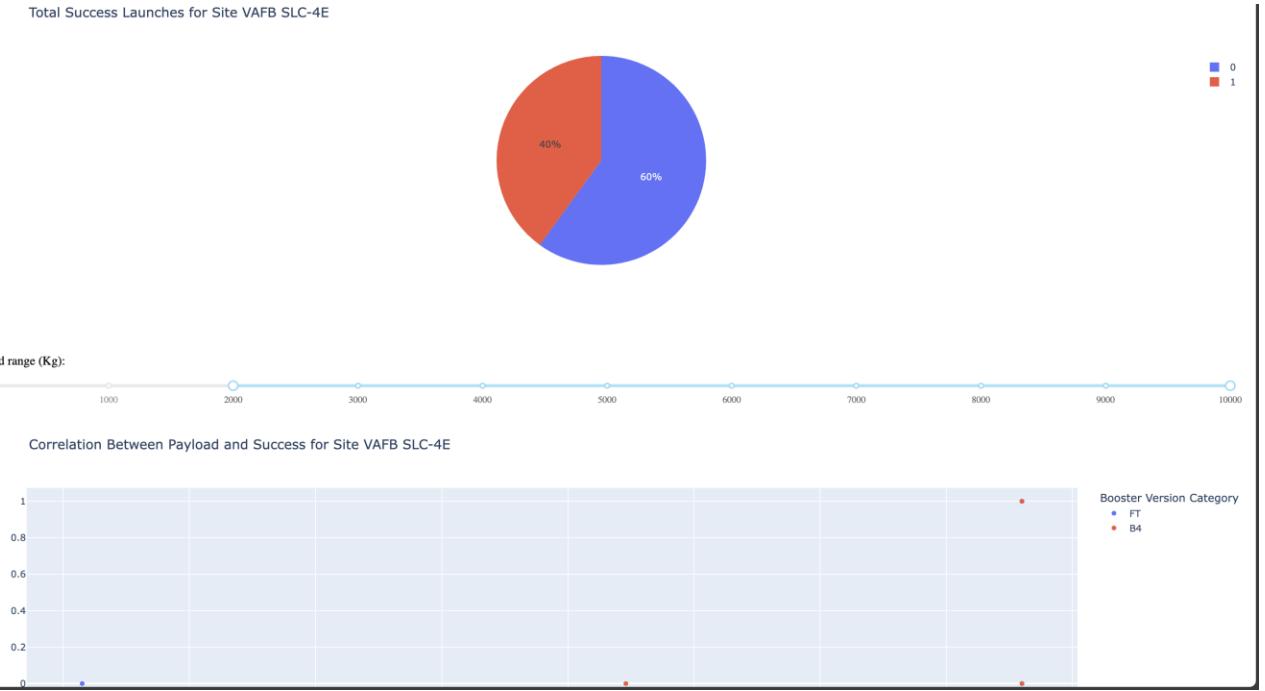


Plotly Dash dashboard Results

- At 1000 kg Payloads we find all Booster versions have successful lunches.
- For CCAFS LC-40 we find when we increase Payload only the booster FT has successful lunches.

Plotly Dash dashboard Results

For VAFB SLC-4E we
find when we increase Payload no one has
successful lunches



Predictive Analysis (classification) Results

We use Grid Search which is a hyperparameter tuning technique used in machine learning to find the best combination of hyperparameters for a given model. Hyperparameters are variables that are not learned by the model, but rather set by the user before training.

Predictive Analysis (classification) Results

First machine learning model is logistic regression. We 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.after that 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_. Then we calcaulate thhe accuracy of logistic Regression model by score function.

```
[19]: (18,)

▼ 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.

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

[21]: parameters =[{"C":0.01, "penalty":'l2', "solver":['lbfgs']}] # l1 lasso l2 ridge
lr=LogisticRegression()

[24]: logreg_cv = GridSearchCV(lr,parameters, cv=10)
logreg_cv.fit(X_train, Y_train)

[24]: > 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_.

[25]: print("tuned hpyerparameters :(best parameters) ",logreg_cv.best_params_)
print("accuracy :",logreg_cv.best_score_)

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

Predictive analysis (classification) Results

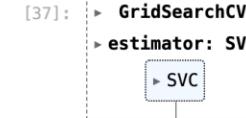
- Second machine learning model is support vector machine. We Create a support vector machine object then create a GridSearchCV object `svm_cv` with `cv = 10`. Fit the object to find the best parameters from the dictionary `parameters`. after that We output the GridSearchCV object for logistic regression. We display the best parameters using the data attribute `best_params_` and the accuracy on the validation data using the data attribute `best_score_`. Then we calculate the accuracy of support vector machine model by `score` function.

TASK 6

Create a support vector machine object then create a `GridSearchCV` object `svm_cv`:

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

```
[37]: svm_cv = GridSearchCV(svm,parameters, cv=10)  
svm_cv.fit(X_train, Y_train)
```



```
[38]: print("tuned hyperparameters :(best parameters) ",svm_cv.best_params_)  
print("accuracy :",svm_cv.best_score_)  
  
tuned hyperparameters :(best parameters) {'C': 1.0, 'gamma': 0.0316227  
accuracy : 0.8482142857142856
```

TASK 7

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

```
[60]: score_svm_cv = svm_cv.score(X_train, Y_train)  
score_svm_cv  
  
[60]: 0.8888888888888888
```

Predictive analysis (classification) Results

Third machine learning model is decision tree machine. We Create a decision tree classifier object then create a GridSearchCV object tree_cv with cv = 10. First find the best parameters from the dictionary parameters. after that We output the GridSearch CV object for decision tree. We display the best parameters using the data attribute best_params_ and the accuracy on the validation data using the data attribute best_score_. Then we calculate the accuracy of decision tree model by score function.

```
[40]: parameters = {'criterion': ['gini', 'entropy'],
                   'splitter': ['best', 'random'],
                   'max_depth': [2*n for n in range(1,10)],
                   'max_features': ['auto', 'sqrt'],
                   'min_samples_leaf': [1, 2, 4],
                   'min_samples_split': [2, 5, 10]}

tree = DecisionTreeClassifier()

[44]: tree_cv = GridSearchCV(tree, parameters, cv=10)
tree_cv.fit(X_train, Y_train)

Below are more details about the failures:
-----
3240 fits failed with the following error:
Traceback (most recent call last):
  File "/lib/python3.11/site-packages/sklearn/model_selection/_validation.py", line 729, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "/lib/python3.11/site-packages/sklearn/base.py", line 1145, in wrapper
    estimator._validate_params()
  File "/lib/python3.11/site-packages/sklearn/base.py", line 638, in _validate_params
    validate_parameter_constraints()
  File "/lib/python3.11/site-packages/sklearn/utils/_param_validation.py", line 95, in validate_parameter_constraints
    raise InvalidParameterError(
sklearn.utils._param_validation.InvalidParameterError: The 'max_features' parameter of DecisionTreeClassifier must be an int in the range [1, inf), a float in the range (0.0, 1.0], a str among {'log2', 'sqrt'} or None. Got 'auto' instead.

    warnings.warn(some_fits_failed_message, FitFailedWarning)
/lib/python3.11/site-packages/sklearn/model_selection/_search.py:979: UserWarning: One or more of the test scores are non-finite: [      nan      nan      nan      nan]

[45]: print("tuned hyperparameters :(best parameters) ",tree_cv.best_params_)
print("accuracy :",tree_cv.best_score_)

tuned hyperparameters :(best parameters)  {'criterion': 'gini', 'max_depth': 8, 'max_features': 'sqrt', 'min_samples_leaf': 1, 'min_samples_split': 2, 'splitter': 'best'}
accuracy : 0.8875
```

▼ **TASK 9**

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

```
[63]: score_tree_cv = tree_cv.score(X_train, Y_train)
score_tree_cv

[63]: 0.9722222222222222
```

yodide) | Unknown Mode: Command ↻ Ln 1, Col 1 SpaceX_Machine_Learning_Prediction_Part_5.ipynb 0

Predictive analysis (classification) Results

- Last machine learning model is k nearest neighbors machine. Create a k nearest neighbors object then create a GridSearchCV object knn_cv with cv = 10. Fit the object to find the best parameters from the dictionary parameters.after that We output the Grid SearchCV object for k nearest neighbors . We display the best parameters using the data attribute best_params_ and the accuracy on the validation data using the data attribute best_score_. Then we calcaulate the accuracy of decision tree model by score function.

TASK 10

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

```
[47]: parameters = {'n_neighbors': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],  
                 'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute'],  
                 'p': [1,2]}
```

```
KNN = KNeighborsClassifier()
```

```
[48]: knn_cv = GridSearchCV(KNN, parameters, cv=10)  
knn_cv.fit(X_train, Y_train)
```

```
/lib/python3.11/site-packages/threadpoolctl.py:1019: RuntimeWarning: libc not found. The ctypes module in Python 3.11 is maybe too old for this OS.  
    warnings.warn!
```

```
[48]: > GridSearchCV  
  estimator: KNeighborsClassifier  
    > KNeighborsClassifier
```

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

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

TASK 11

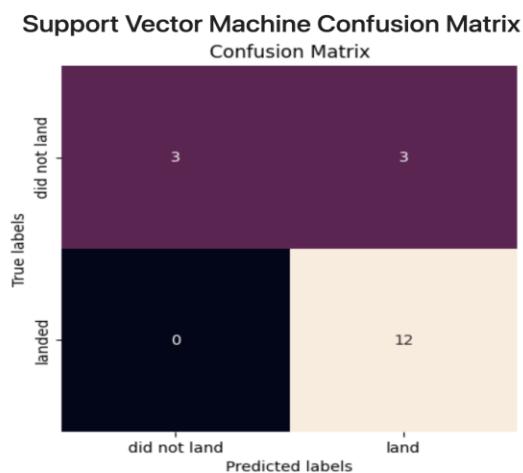
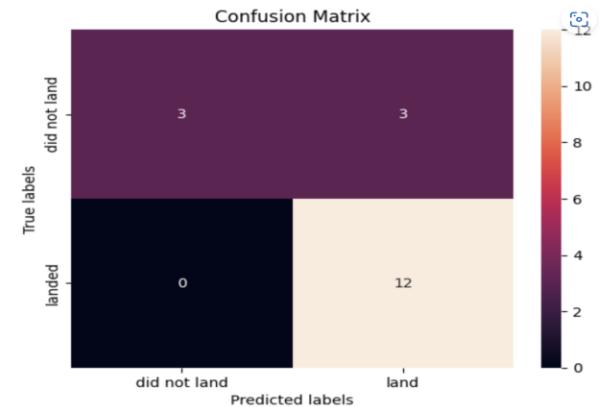
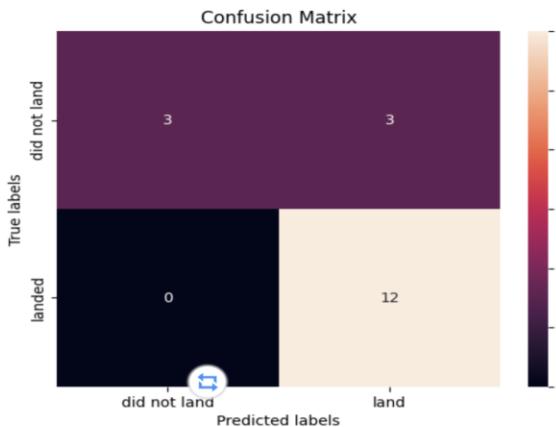
```
Calculate the accuracy of knn_cv on the test data using the method score :
```

```
[56]: score_knn_cv =knn_cv.score(X_train, Y_train)  
score_knn_cv
```

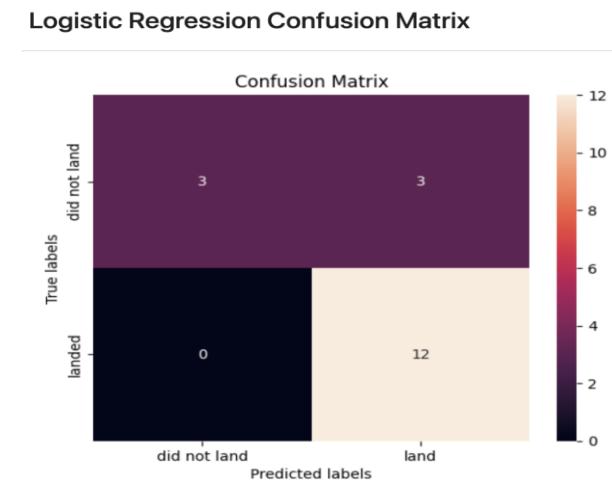
```
[56]: 0.8611111111111112
```

Predictive analysis (classification) Results

Here is the confision matrix for all machine learning models



Confusion Matrix k-Nearest Neighbors



Decison Tree ConfusioN Matrixx

Conclusion



Conclusion

- Lunches sites:
 - CCAFS LC-40 has 26.9% successful lunches
 - VAFB SLC-4E has 40% successful lunches
 - KSC LC-39A has 23.1% successful lunches
 - CCAFS SLC-40 has 42.9% successful lunches
- Booster Version:
 - At CCAFS LC-40 only FT Booster Version has successful lunch
 - At VAFB SLC-4E only B4 Booster Version has successful lunch
 - At KSC LC-39A only FT Booster Version has successful lunch
 - At CCAFS LC-40 only FT Booster Version has successful lunch

Conclusion

- PayloadMass:
- When we increase Payload, FT booster Version has successful launch.
- Model Performance:
- When we calculate the accuracy of models by score function we find that the decision tree has the most accurate model which is equal to 0.97



Thank You