Winning Space Race with Data Science

Rafael Paz Silva
05/01/2022
https://github.com/rafapaz/ML_IBM-10

Outline

- Executive Summary
- Introduction
- Methodology
- Results
- Conclusion
- Appendix

Executive Summary

- Summary of methodologies
  - Collecting the data
  - Web scraping
  - Data wrangling
  - EDA for data visualization
  - EDA with SQL
  - Map Analysis with Folium
  - Dashboard with Plotly Dash
  - Machine Learning Prediction

- Summary of all results
  - Exploratory data analysis results
  - Interactive analytics demo in screenshots
  - Predictive analysis results

Introduction

- Project background and context
  - We predicted if the Falcon 9 first stage will land successfully. SpaceX advertises Falcon 9 rocket launches on its website, with a cost of 62 million dollars; other providers cost upward of 165 million dollars each, much of the savings is because SpaceX can reuse the first stage. Therefore, if we can determine if the first stage will land, we can determine the cost of a launch. This information can be used if an alternate company wants to bid against SpaceX for a rocket launch.

- Problems you want to find answers
  - What influences if the rocket will land successfully?
  - The effect each relationship with certain rocket variables will impact in determining the success rate of a successful landing.
  - What conditions does SpaceX have to achieve to get the best results and ensure the best rocket success landing rate.

Section 1

Methodology

Methodology

- Data collection methodology:
  - SpaceX Rest API
  - (Web Scrapping) from Wikipedia
- Performed data wrangling (Transforming data for Machine Learning)
  - One Hot Encoding data fields for Machine Learning and dropping irrelevant columns
- Performed exploratory data analysis (EDA) using visualization and SQL
  - Plotting : Scatter Graphs, Bar Graphs to show relationships between variables to show patterns of data.
- Performed interactive visual analytics using Folium and Plotly Dash
- Performed predictive analysis using classification models
  - How to build, tune, evaluate classification models

Data Collection

- We worked with SpaceX launch data that is gathered from the SpaceX REST API.
- This API will give us data about launches, including information about the rocket used,

payload delivered, launch specifications, landing specifications, and landing outcome.

- Our goal is to use this data to predict whether SpaceX will attempt to land a rocket or not.
- The SpaceX REST API endpoints, or URL, starts with api.spacexdata.com/v4/.
- Another popular data source for obtaining Falcon 9 Launch data is web scraping Wikipedia using BeautifulSoup.

Data Collection – SpaceX API

Request and parse the SpaceX launch data using the GET request

```
In [6]: spacex_url="https://api.spacexdata.com/v4/launches/past"

In [7]: response = requests.get(spacex_url)
```

Check the content of the response

You should see the response contains massive information about SpaceX launches. Next, let's try to discover some more relevant information for this project.

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

To make the requested JSON results more consistent, we will use the following static response object for this project:

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

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

```
In [10]: response.status_code
Out[10]: 200
```

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

```
In [12]: # Use json_normalize meethod to convert the json result into a dataframe
data = pd.json_normalize(response.json())
```

Using the dataframe `data` print the first 5 rows

```
In [13]: # Get the head of the dataframe
data.head()
```

Out[13]:

	static_fire_date_utc	static_fire_date_unix	net	window	rocket	success	failures	details	crew	ships	capsules	
0	2006-03-17T00:00:00.000Z	1.142554e+09	False	0.0	5e9d0d95eda69955f709d1eb	False	[{'time': 33, 'altitude': None, 'reason': 'merlin engine failure'}]	Engine failure at 33 seconds and loss of vehicle	[]	[]	[]	[5eb0e4b5

Successful first stage

Filter the dataframe to only include Falcon 9

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

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
```

Dealing with Missing Values

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


Github URL: https://github.com/rafapaz/ML_IBM-10/blob/master/jupyter-labs-spacex-data-collection-api.ipynb

Data Collection - Scraping

Request the Falcon9 Launch Wiki page from its URL

```
static_url = "https://en.wikipedia.org/w/index.php?title=List_of_Falcon_9_and_Falcor
```

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

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

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

```
# 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.content)
```

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>
```

Extract all column/variable names from the HTML table header

```
# Use the find_all function in the BeautifulSoup obje
# Assign the result to a list called `html_tables`
html_tables = soup.find_all('table')
```

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

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

```
column_names = []

# Apply find_all() function with `th` element on f
# Iterate each th element and apply the provided e
# Append the Non-empty column name (`if name is no
column_th = first_launch_table.find_all('th')
for th in column_th:
    name = extract_column_from_header(th)
    if name is not None and len(name) > 0:
        column_names.append(name)
```

Check the extracted column names

```
print(column_names)
```

```
['Flight No.', 'Date and time ( )', 'Launch site',
```

Create a data frame by parsing the launch HTML tables

```
extracted_row = 0
#Extract each table
for table_number,table in enumerate(soup.find_all('table',"w
   # get table row
    for rows in table.find_all("tr"):
        #check to see if first table heading is as number co
        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_dic
            #print(flight_number)
            launch_dict['Flight No.'].append(flight_number)
            datatimelist=date_time(row[0])

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

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

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

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

Save csv:

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

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


Github URL: https://github.com/rafapaz/ML_IBM-10/blob/master/jupyter-labs-webscraping.ipynb

Data Wrangling

Calculate the number of launches on each site

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

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

Calculate the number and occurrence of each orbit

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

```
GTO      27
ISS      21
VLEO     14
PO        9
LEO       7
SSO       5
MEO       3
ES-L1     1
HEO       1
SO        1
GEO       1
Name: Orbit, dtype: int64
```

Calculate the number and occurence of mission outcome per orbit type

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

```
landing_outcomes
```

```
True ASDS      41
None None      19
True RTLS      14
False ASDS      6
True Ocean      5
False Ocean     2
None ASDS       2
False RTLS      1
Name: Outcome, dtype: int64
```

`True Ocean` means the mission outcome was successfully landed to a specific
unsuccessfully landed to a specific region of the ocean. `True RTLS` means the
means the mission outcome was unsuccessfully landed to a ground pad. `True A`
`False ASDS` means the mission outcome was unsuccessfully landed to a drone

```
for i,outcome in enumerate(landing_outcomes.keys()):
    print(i,outcome)
```

```
0 True ASDS
1 None None
2 True RTLS
3 False ASDS
4 True Ocean
5 False Ocean
6 None ASDS
7 False RTLS
```

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

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

```
{'False ASDS', 'False Ocean', 'False RTLS', 'None ASDS', 'No
```

Create a landing outcome label from Outcome column

```
# landing_class = 0 if bad_outcome
# landing_class = 1 otherwise
landing_class = [0 if outcome in bad_outcomes else 1 for outcome in df['Outcome']]
```

Save file:

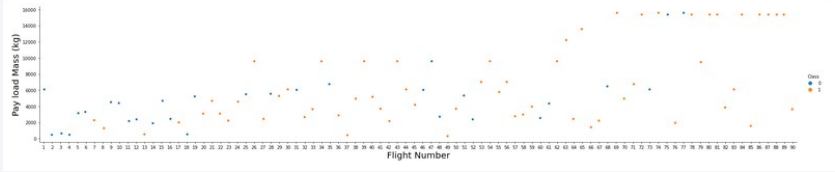
```
df.to_csv("dataset_part_2.csv", index=False)
```


Github URL: https://github.com/rafapaz/ML_IBM-10/blob/master/labs-jupyter-spacex-Data%20wrangling.ipynb

EDA with Data Visualization

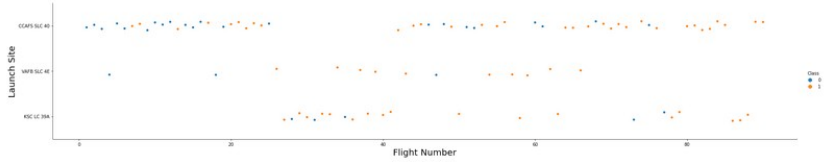
Flight Number x Pay Load Mass

```
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()
```



Flight Number x Launch Site

```
# Plot a scatter point chart with x axis to be Flight Number and y axis to be the launch site, and hue to be the clas
sns.catplot(y="LaunchSite", x="FlightNumber", hue="Class", data=df, aspect = 5)
plt.xlabel("Flight Number",fontsize=20)
plt.ylabel("Launch Site",fontsize=20)
plt.show()
```



Launch Site x Pay Load Mass

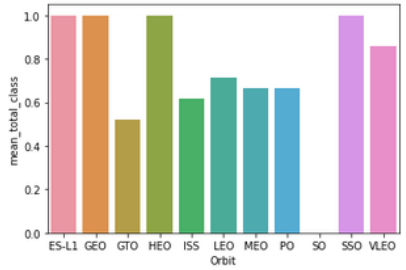
```
# Plot a scatter point chart with x axis to be Pay Load Mass (kg) and y axis to be the launch site, and hue to be the
sns.catplot(y="LaunchSite", x="PayloadMass", hue="Class", data=df, aspect = 5)
plt.xlabel("Pay Load Mass (kg)",fontsize=20)
plt.ylabel("Launch Site",fontsize=20)
plt.show()
```

Relationship between success rate of each orbit type

```
# HINT use groupby method on Orbit column and get the mean of Class column
df_aux = df.groupby(['Orbit']).agg(mean_total_class=("Class", 'mean'))
df_aux = df_aux.reset_index()
```

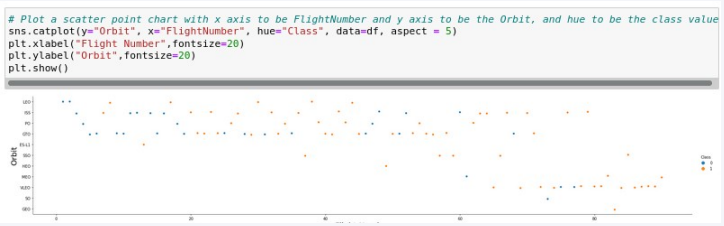
```
# plot barplot
sns.barplot(x="Orbit",
            y="mean_total_class",
            data=df_aux)
```

```
<AxesSubplot:xlabel='Orbit', ylabel='mean_total_class'>
```



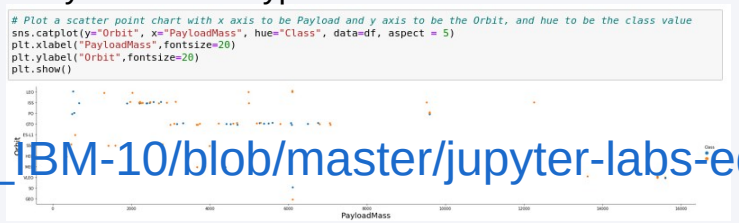
FlightNumber x Orbit type

```
# Plot a scatter point chart with x axis to be FlightNumber and y axis to be the Orbit, and hue to be the class value
sns.catplot(y="Orbit", x="FlightNumber", hue="Class", data=df, aspect = 5)
plt.xlabel("Flight Number",fontsize=20)
plt.ylabel("Orbit",fontsize=20)
plt.show()
```



Payload x Orbit type

```
# Plot a scatter point chart with x axis to be Payload and y axis to be the Orbit, and hue to be the class value
sns.catplot(y="Orbit", x="PayloadMass", hue="Class", data=df, aspect = 5)
plt.xlabel("PayloadMass",fontsize=20)
plt.ylabel("Orbit",fontsize=20)
plt.show()
```



Launch success yearly trend

```
# Plot a line chart with x axis to be the extracted year and y
sns.lineplot(data=df, x=Extract_year(df["Date"]), y="Class")
```

```
<AxesSubplot:ylabel='Class'>
```

Create dummy variables to categorical columns

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

```
# HINT: use astype function
features_one_hot = features_one_hot.astype('float64')
features_one_hot.head()
```

	FlightNumber	PayloadMass	Flights	GridFins	Reused	Legs	Block	ReusedCount	Orbit_ES-L1	Orbit_GEO	...	Serial_B1048	Serial_B1049
0	1.0	6104.959412	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	...	0.0	0.0
1	2.0	525.000000	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	...	0.0	0.0
2	3.0	677.000000	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	...	0.0	0.0
3	4.0	500.000000	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	...	0.0	0.0
4	5.0	3170.000000	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	...	0.0	0.0

5 rows × 80 columns

Github URL: https://github.com/rafapaz/ML_IBM-10/blob/master/jupyter-labs-eda-dataviz.ipynb

EDA with SQL

I imported the csv file into a SQLite database instead of using DB2.

```
%sql SELECT DISTINCT Launch_Site FROM SPACEXTBL;
```

```
 * sqlite:///spacex.db3
Done.
```

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

```
%sql SELECT * FROM SPACEXTBL WHERE Launch_Site like 'CCA%' LIMIT 5;
```

```
 * sqlite:///spacex.db3
Done.
```

Date	Time (UTC)	Booster_Version	Launch_Site	Payload	PAYLOAD_MASS__KG_	Orbit	Customer	Mission_Outcome	Landing_Outcome
04-06-2010	18:45:00	F9 v1.0 B0003	CCAFS LC-40	Dragon Spacecraft Qualification Unit	0	LEO	SpaceX	Success	Failure (parachute)
08-12-2010	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)
22-05-2012	07:44:00	F9 v1.0 B0005	CCAFS LC-40	Dragon demo flight C2	525	LEO (ISS)	NASA (COTS)	Success	No attempt
08-10-2012	00:35:00	F9 v1.0 B0006	CCAFS LC-40	SpaceX CRS-1	500	LEO (ISS)	NASA (CRS)	Success	No attempt
01-03-2013	15:10:00	F9 v1.0 B0007	CCAFS LC-40	SpaceX CRS-2	677	LEO (ISS)	NASA (CRS)	Success	No attempt

```
%sql SELECT SUM(PAYLOAD_MASS__KG_) FROM SPACEXTBL WHERE Customer = 'NASA (CRS)';
```

```
 * sqlite:///spacex.db3
Done.
```

SUM(PAYLOAD_MASS__KG_)
45596

```
%sql SELECT AVG(PAYLOAD_MASS__KG_) FROM SPACEXTBL WHERE Booster_Version = 'F9 v1.1';
```

```
 * sqlite:///spacex.db3
Done.
```

AVG(PAYLOAD_MASS__KG_)
2928.4

```
%sql SELECT MIN(Date) FROM SPACEXTBL WHERE Mission_Outcome like '%Success%';
```

```
 * sqlite:///spacex.db3
Done.
```

MIN(Date)
01-03-2013

```
%sql SELECT Payload FROM SPACEXTBL WHERE Mission_Outcome like '%Success%'
    AND PAYLOAD_MASS__KG_ > 4000 AND PAYLOAD_MASS__KG_ < 6000;
```

```
 * sqlite:///spacex.db3
Done.
```

Payload
Dragon demo flight C2
SpaceX CRS-1
CASSIOPE
AsiaSat 8
AsiaSat 6
DSCOVR
ABS-3A Eutelsat 115 West B
Turkmen 52 / MonacoSAT
Jason-3
SES-9
JCSAT-14
JCSAT-16

```
%sql SELECT Mission_Outcome, COUNT(Mission_Outcome) FROM SPACEXTBL GROUP BY Mission_Outcome;
```

```
 * sqlite:///spacex.db3
Done.
```

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

```
%sql SELECT Booster_Version FROM SPACEXTBL WHERE PAYLOAD_MASS__KG_ = (SELECT MAX(PAYLOAD_MASS__KG_) FROM SPACEXTBL);
```

```
 * sqlite:///spacex.db3
Done.
```

Booster_Version
F9 FT B1029.1
F9 FT B1036.1
F9 B4 B1041.1
F9 FT B1036.2
F9 B4 B1041.2
F9 B5B1048.1
F9 B5 B1049.2

```
%sql SELECT Landing_Outcome, Booster_Version, Launch_Site FROM SPACEXTBL WHERE Mission_Outcome like '%Fail%'
    AND Date like '%2015%';
```

```
 * sqlite:///spacex.db3
Done.
```

Landing_Outcome	Booster_Version	Launch_Site
Precluded (drone ship)	F9 v1.1 B1018	CCAFS LC-40

```
%sql SELECT Landing_Outcome, COUNT(*) as qtd FROM SPACEXTBL
    WHERE substr(Date,7)||substr(Date,4,2)||substr(Date,1,2) between '20100604' and '20170320'
    GROUP BY Landing_Outcome ORDER BY qtd DESC;
```

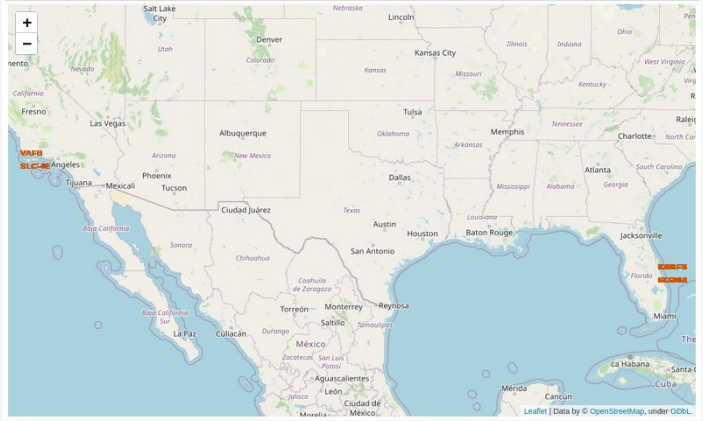
```
 * sqlite:///spacex.db3
Done.
```

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

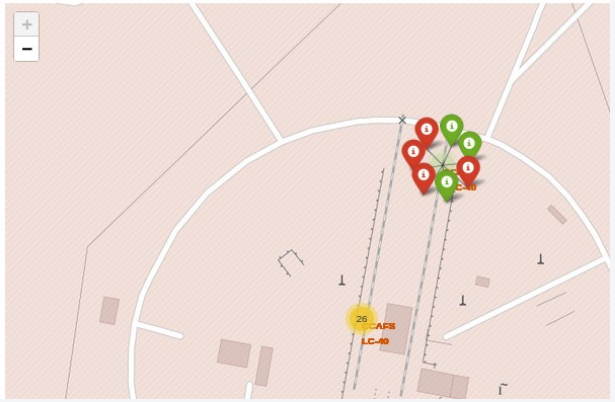
Github URL: https://github.com/rafapaz/ML_IBM-10/blob/master/jupyter-labs-eda-sql-coursera.ipynb

Build an Interactive Map with Folium

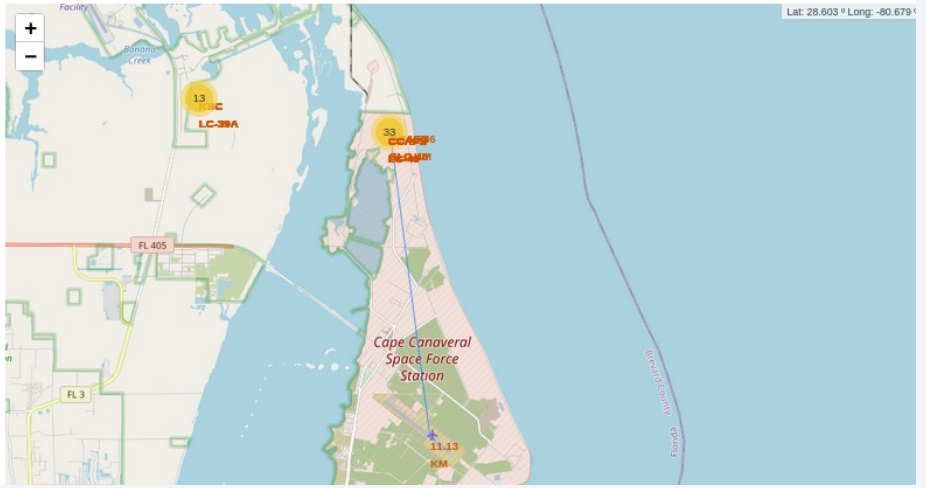
```
# Initial the map
site_map = folium.Map(location=nasa_coordinate, zoom_start=5)
# For each launch site, add a Circle object based on its coordinate (Lat, Long) values. In addition, add Launch site
for index, linha in spacex_df.iterrows():
    coordinate = [linha['Lat'], linha['Long']]
    #circle = folium.Circle(coordinate, radius=1000, color='#d35400', fill=True)
    circle.add_child(folium.Popup(linha['Launch Site']))

    marker = folium.map.Marker(
        coordinate,
        # Create an icon as a text label
        icon=DivIcon(
            icon_size=(2,2),
            icon_anchor=(0,0),
            html='<div style="font-size: 12; color:#d35400;"><b>%s</b></div>' % linha['Launch Site'],
            )
        )
    #site_map.add_child(circle)
    site_map.add_child(marker)
site_map
```



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

# for each row in spacex_df data frame
# create a Marker object with its coordinate
# and customize the Marker's icon property to indicate if this launch was successed or fa
# e.g., icon=folium.Icon(color='white', icon_color=row['marker_color']
for index, record in spacex_df.iterrows():
    # TODO: Create and add a Marker cluster to the site map
    coordinate = [record['Lat'], record['Long']]
    marker = folium.map.Marker(
        coordinate,
        popup='{} / {}'.format(record['Launch Site'], coordinate),
        icon=folium.Icon(color=record['marker_color'])
        )
    marker_cluster.add_child(marker)

site_map
```



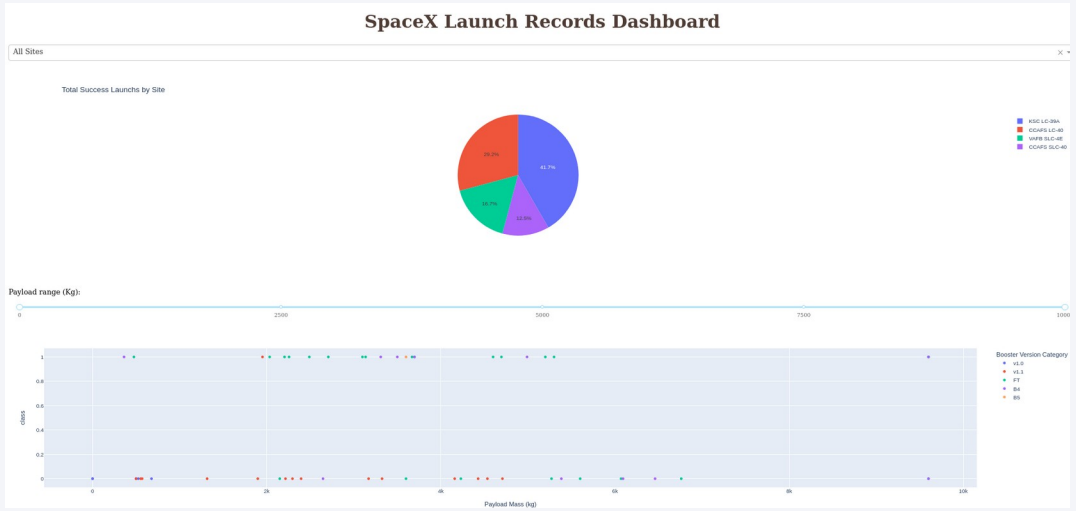
```
distance_airport = calculate_distance(28.563197, -80.576820, 28.464, -80.562)
coordinate = [28.464, -80.562]
distance_marker = folium.Marker(
    coordinate,
    icon=DivIcon(
        icon_size=(20,20),
        icon_anchor=(0,0),
        html='<div style="font-size: 12; color:#d35400;"><b>%s</b></div>' % "{:10.2f} KM".format(distance_airport),
        )
    )
site_map.add_child(distance_marker)

coordinates = [[28.563197, -80.576820], [28.464, -80.562]]
lines=folium.PolyLine(locations=coordinates, weight=1)
site_map.add_child(lines)
site_map
```

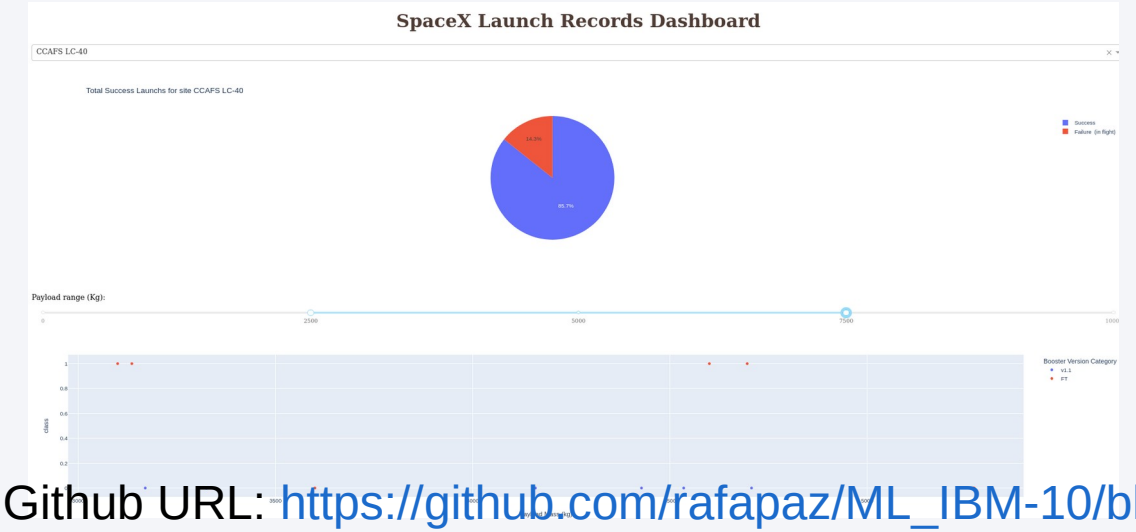



```
# Create a `folium.PolyLine` object using the coastline coordinates and launch site coordinate
# 28.563197, -80.576820, 28.557, -80.567
coordinates = [[28.563197, -80.576820], [28.563, -80.568]]
lines=folium.PolyLine(locations=coordinates, weight=1)
site_map.add_child(lines)
```

Github URL: https://github.com/rafapaz/ML_IBM-10/blob/master/lab_jupyter_launch_site_location.ipynb

Build a Dashboard with Plotly Dash





Which site has the largest successful launches?
CCAFS LC-40

Which site has the highest launch success rate?
KSC LC-39A

Which payload range(s) has the highest launch success rate?
Between 2k and 4k

Which payload range(s) has the lowest launch success rate?
Between 6k and 10k

Which F9 Booster version (v1.0, v1.1, FT, B4, B5, etc.)
has the highest launch success rate?
FT

Github URL: https://github.com/rafapaz/ML_IBM-10/blob/master/spacex_dash_app.py

Predictive Analysis (Classification)

1 - Standardizing the data
2 - Split data in train and test
3 - Apply Logistic Regression
4 - Apply SVM
5 - Apply Decision Tree
6 - Apply K Neighboor

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

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

1

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

we can see we only have 18 test samples.

```
Y_test.shape
```

```
(18,)
```

2

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

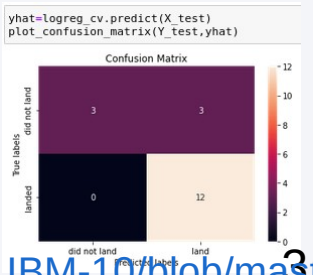
```
GridSearchCV(cv=10, estimator=LogisticRegression(),
             param_grid={'C': [0.01, 0.1, 1], 'penalty': ['l2'],
                         'solver': ['lbfgs']})
```

We output the `GridSearchCV` object for logistic regression. We display the best parameters using the data attribute `bes` the validation data using the data attribute `best_score_`.

```
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
```

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



3

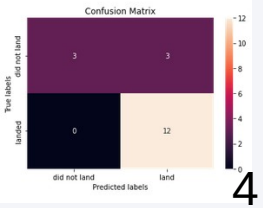
```
svm_cv = GridSearchCV(svm, parameters, cv=10)
svm_cv.fit(X_train, Y_train)
```

```
GridSearchCV(cv=10, estimator=SVC(),
             param_grid={'C': array([1.00000000e-03, 3.16227766e-02, 1.00000000e+00, 3.16227766e+01,
       1.00000000e+03]),
                         'gamma': array([1.00000000e-03, 3.16227766e-02, 1.00000000e+00, 3.16227766e+01,
       1.00000000e+03]),
                         'kernel': ('linear', 'rbf', 'poly', 'rbf', 'sigmoid')})
```

```
print("tuned hpyerparameters :(best parameters) ",svm_cv.best_params_)
print("accuracy :",svm_cv.best_score_)
```

```
tuned hpyerparameters :(best parameters)  {'C': 1.0, 'gamma': 0.03162277660168379, 'kernel': 'sigmoid'}
accuracy : 0.8482142857142856
```

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



4

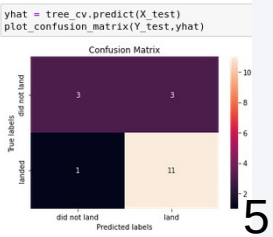
```
tree_cv = GridSearchCV(tree, parameters, cv=10)
tree_cv.fit(X_train, Y_train)
```

```
GridSearchCV(cv=10, estimator=DecisionTreeClassifier(),
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': [2, 4, 6, 8, 10, 12, 14, 16, 18],
                         'max_features': ['auto', 'sqrt'],
                         'min_samples_leaf': [1, 2, 4],
                         'min_samples_split': [2, 5, 10],
                         'splitter': ['best', 'random']})
```

```
print("tuned hpyerparameters :(best parameters) ",tree_cv.best_params_)
print("accuracy :",tree_cv.best_score_)
```

```
tuned hpyerparameters :(best parameters)  {'criterion': 'gini', 'max_depth': 14, 'max_features': 'auto', 'min_samples_leaf': 1, 'min_samples_split': 10, 'splitter': 'best'}
accuracy : 0.8892857142857145
```

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



5

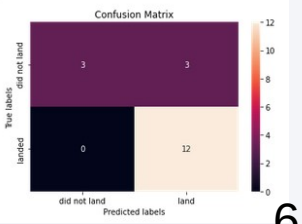
```
knn_cv = GridSearchCV(KNN, parameters, cv=10)
knn_cv.fit(X_train, Y_train)
```

```
GridSearchCV(cv=10, estimator=KNeighborsClassifier(),
             param_grid={'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute'],
                         'n_neighbors': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
                         'p': [1, 2]})
```

```
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
```

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

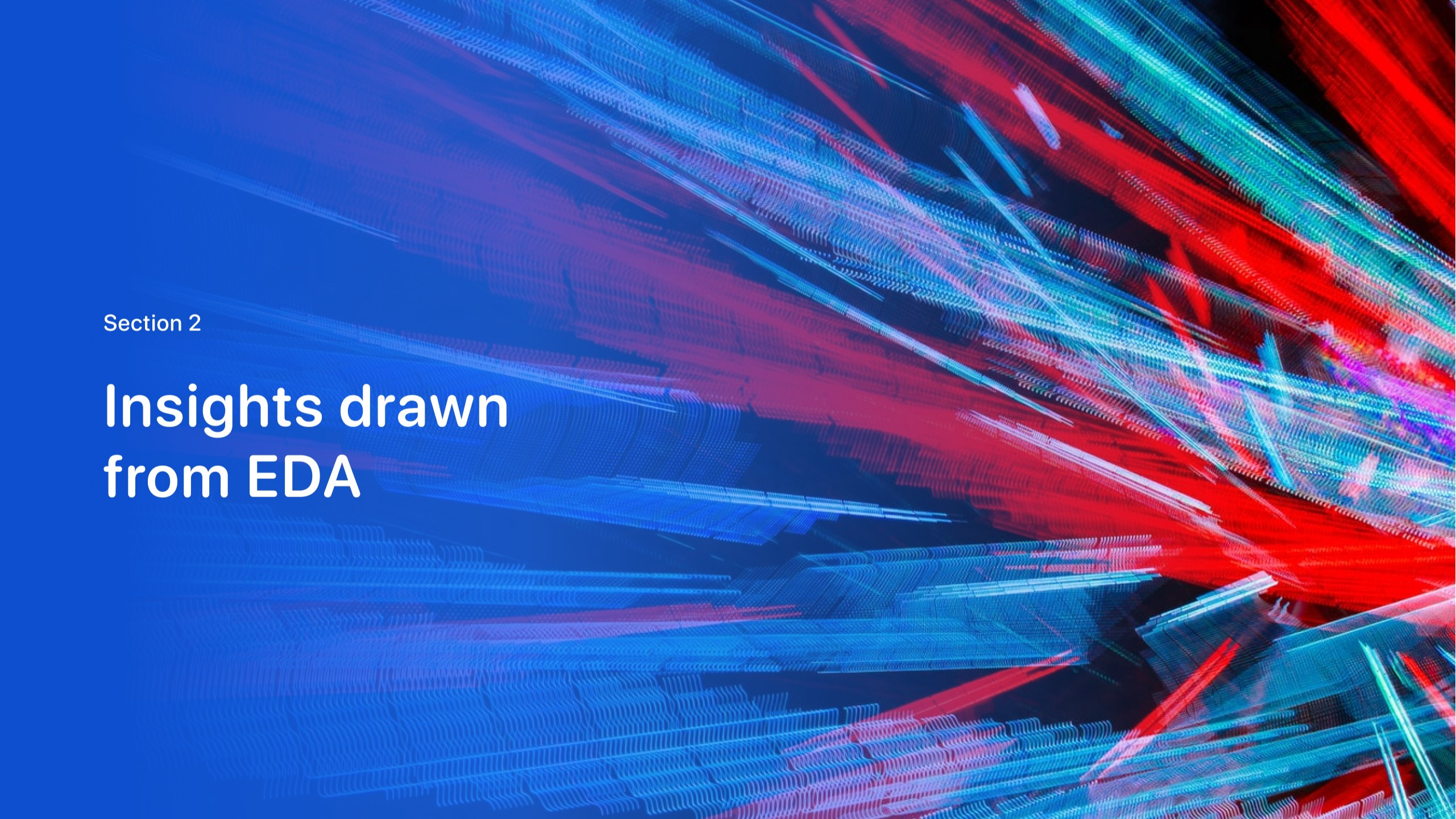


6

Github URL: https://github.com/rafapaz/ML_IBM-10/blob/master/SpaceX_Machine%20Learning%20Prediction_Part_5.ipynb

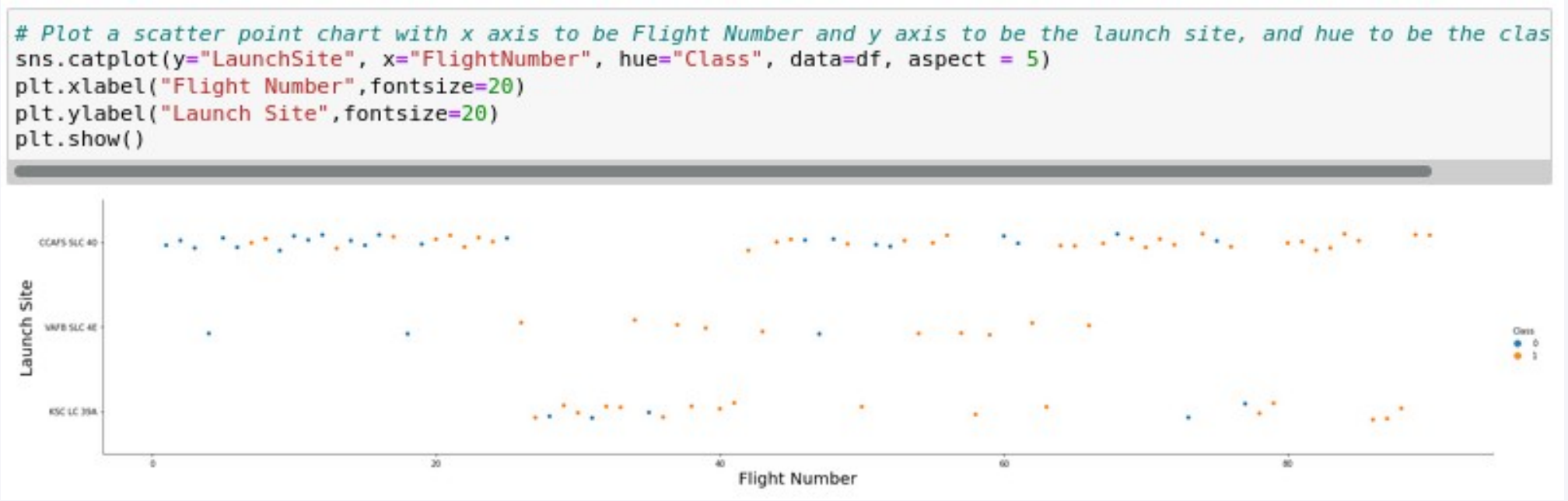
Results

- Exploratory data analysis results
- Interactive analytics demo in screenshots
- Predictive analysis results


Section 2
Insights drawn from EDA

Flight Number vs. Launch Site

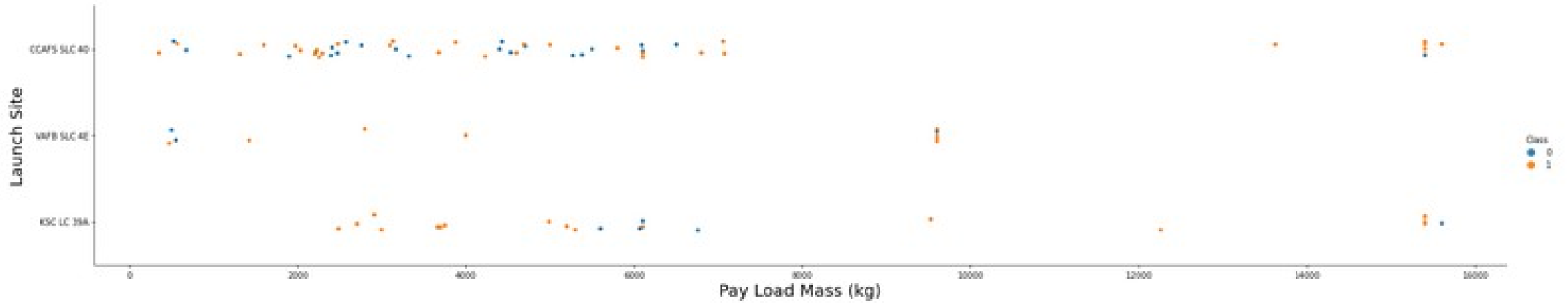
```
# Plot a scatter point chart with x axis to be Flight Number and y axis to be the launch site, and hue to be the clas
sns.catplot(y="LaunchSite", x="FlightNumber", hue="Class", data=df, aspect = 5)
plt.xlabel("Flight Number",fontsize=20)
plt.ylabel("Launch Site",fontsize=20)
plt.show()
```



The more amount of flights at a launch site the greater the success rate at a launch site.

Payload vs. Launch Site

```
# Plot a scatter point chart with x axis to be Pay Load Mass (kg) and y axis to be the launch site, and hue to be the
sns.catplot(y="LaunchSite", x="PayloadMass", hue="Class", data=df, aspect = 5)
plt.xlabel("Pay Load Mass (kg)",fontsize=20)
plt.ylabel("Launch Site",fontsize=20)
plt.show()
```



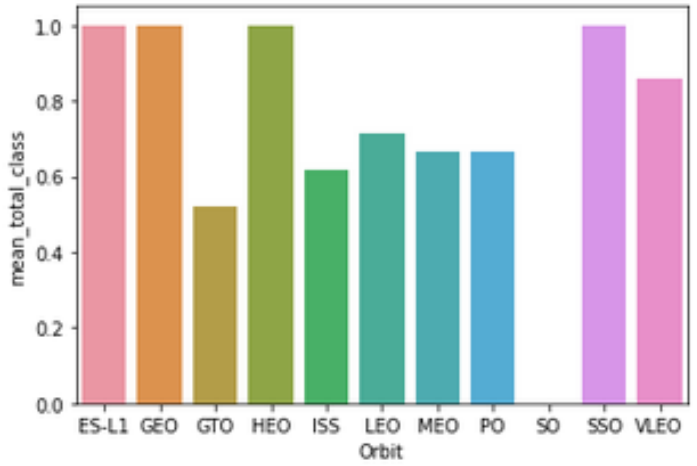
The greater the payload mass for Launch Site CCAFS SLC 40 the higher the success rate for the Rocket. There is not quite a clear pattern to be found using this visualization to make a decision if the Launch Site is dependant on Pay Load Mass for a success launch.

Success Rate vs. Orbit Type

```
# HINT use groupby method on Orbit column and get the mean of Class column
df_aux = df.groupby(['Orbit']).agg(mean_total_class=("Class", 'mean'))
df_aux = df_aux.reset_index()
```

```
# plot barplot
sns.barplot(x="Orbit",
            y="mean_total_class",
            data=df_aux)
```

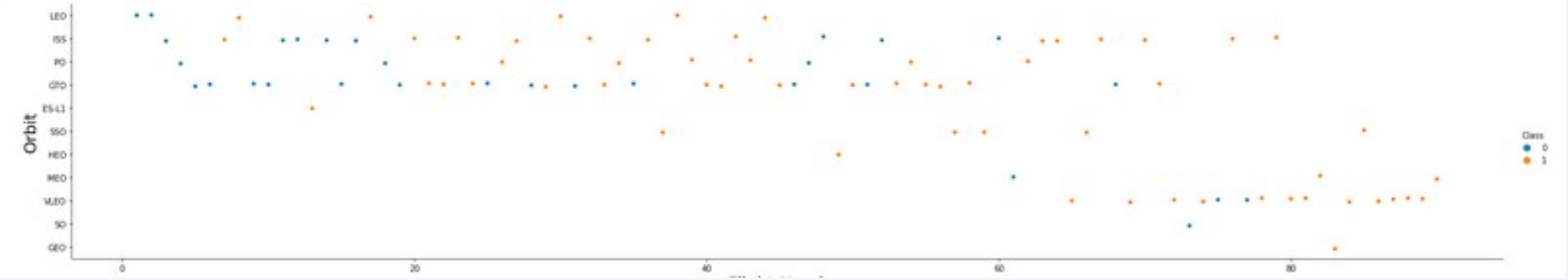
```
<AxesSubplot:xlabel='Orbit', ylabel='mean_total_class'>
```



Orbit GEO,HEO,SSO,ES-L1 has the best Success Rate

Flight Number vs. Orbit Type

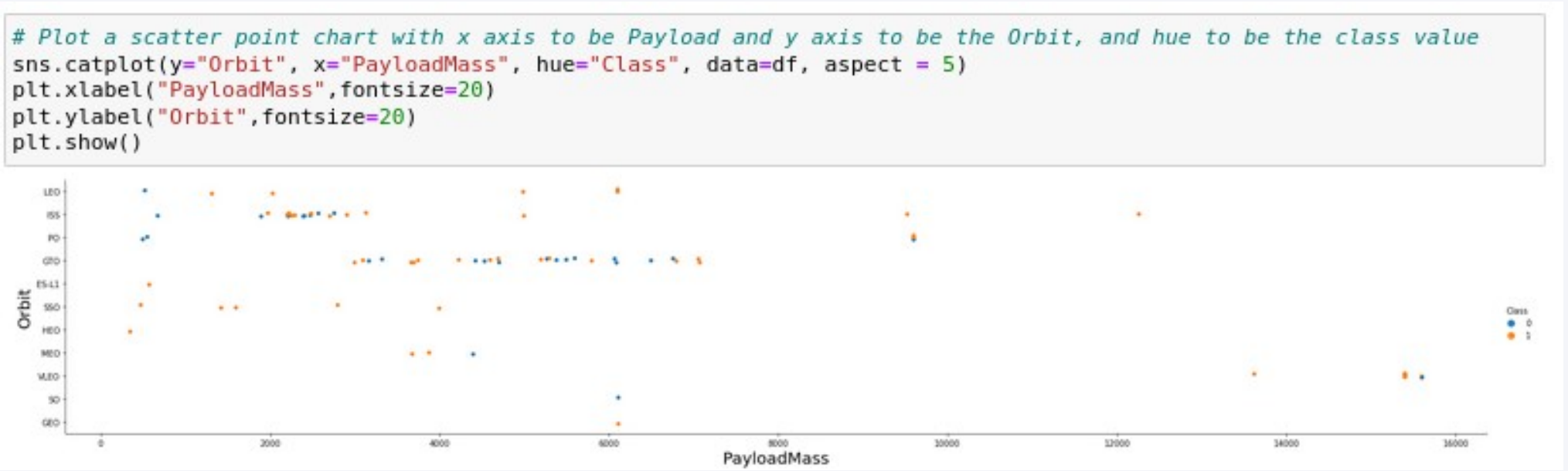
```
# Plot a scatter point chart with x axis to be FlightNumber and y axis to be the Orbit, and hue to be the class value
sns.catplot(y="Orbit", x="FlightNumber", hue="Class", data=df, aspect = 5)
plt.xlabel("Flight Number",fontsize=20)
plt.ylabel("Orbit",fontsize=20)
plt.show()
```



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

Payload vs. Orbit Type

```
# Plot a scatter point chart with x axis to be Payload and y axis to be the Orbit, and hue to be the class value
sns.catplot(y="Orbit", x="PayloadMass", hue="Class", data=df, aspect = 5)
plt.xlabel("PayloadMass",fontsize=20)
plt.ylabel("Orbit",fontsize=20)
plt.show()
```



You should observe that Heavy payloads have a negative influence on GTO orbits and positive on GTO and Polar LEO (ISS) orbits.

Launch Success Yearly Trend

```
# Plot a line chart with x axis to be the extracted year and y
sns.lineplot(data=df, x=Extract_year(df["Date"]), y="Class")
```

```
<AxesSubplot:ylabel='Class'>
```

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

All Launch Site Names

```
%sql SELECT DISTINCT Launch_Site FROM SPACEXTBL;
```

```
 * sqlite:///spacex.db3
Done.
```

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

Using the word DISTINCT in the query means that it will only show Unique values in the Launch_Site column from tblSpaceX

Launch Site Names Begin with 'CCA'

```
%sql SELECT * FROM SPACEXTBL WHERE Launch_Site like 'CCA%' LIMIT 5;
```

```
 * sqlite:///spacex.db3
Done.
```

Date	Time (UTC)	Booster_Version	Launch_Site	Payload	PAYLOAD_MASS__KG_	Orbit	Customer	Mission_Outcome	Landing_Outcome
04-06-2010	18:45:00	F9 v1.0 B0003	CCAFS LC-40	Dragon Spacecraft Qualification Unit	0	LEO	SpaceX	Success	Failure (parachute)
08-12-2010	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)
22-05-2012	07:44:00	F9 v1.0 B0005	CCAFS LC-40	Dragon demo flight C2	525	LEO (ISS)	NASA (COTS)	Success	No attempt
08-10-2012	00:35:00	F9 v1.0 B0006	CCAFS LC-40	SpaceX CRS-1	500	LEO (ISS)	NASA (CRS)	Success	No attempt
01-03-2013	15:10:00	F9 v1.0 B0007	CCAFS LC-40	SpaceX CRS-2	677	LEO (ISS)	NASA (CRS)	Success	No attempt

Using the word LIMIT 5 in the query means that it will only show 5 records from tblSpaceX and LIKE keyword has a wild card with the words ‘CCA%’ the percentage in the end suggests that the Launch_Site name must start with CCA.

Total Payload Mass

```
%sql SELECT SUM(PAYLOAD_MASS__KG_) FROM SPACEXTBL WHERE Customer = 'NASA (CRS)';
```

```
 * sqlite:///spacex.db3
Done.
```

SUM(PAYLOAD_MASS__KG_)
45596

Using the function SUM summates the total in the column PAYLOAD_MASS_KG_
The WHERE clause filters the dataset to only perform calculations on Customer NASA (CRS)

Average Payload Mass by F9 v1.1

```
%sql SELECT AVG(PAYLOAD_MASS__KG_) FROM SPACEXTBL WHERE Booster_Version = 'F9 v1.1';
```

```
 * sqlite:///spacex.db3
Done.
```

AVG(PAYLOAD_MASS__KG_)
2928.4

Using the function AVG works out the average in the column PAYLOAD_MASS_KG_
The WHERE clause filters the dataset to only perform calculations on Booster_version F9 v1.1

First Successful Ground Landing Date

```
%sql SELECT MIN(Date) FROM SPACEXTBL WHERE Mission_Outcome like '%Success%';
```

```
 * sqlite:///spacex.db3
Done.
```

MIN(Date)
01-03-2013

Using the function MIN works out the minimum date in the column Date
The WHERE clause filters the dataset to only perform calculations on Mission_Outcome Success (drone ship)

Successful Drone Ship Landing with Payload between 4000 and 6000

```
%sql SELECT Payload FROM SPACEXTBL WHERE Mission_Outcome like '%Success%'
     AND PAYLOAD_MASS__KG_ > 4000 AND PAYLOAD_MASS__KG_ < 6000;
```

```
 * sqlite:///spacex.db3
Done.
```

Payload
Dragon demo flight C2
SpaceX CRS-1
CASSIOPE
AsiaSat 8
AsiaSat 6
DSCOVR
ABS-3A Eutelsat 115 West B
Turkmen 52 / MonacoSAT
Jason-3
SES-9
JCSAT-14
JCSAT-16

Selecting only Booster_Version
The WHERE clause filters the dataset to Landing_Outcome = Success (drone ship)
The AND clause specifies additional filter conditions Payload_MASS_KG_ > 4000 AND Payload_MASS_KG_ < 6000

Total Number of Successful and Failure Mission Outcomes

```
%sql SELECT Mission_Outcome, COUNT(Mission_Outcome) FROM SPACEXTBL GROUP BY Mission_Outcome;
```

```
 * sqlite:///spacex.db3
Done.
```

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

Here we use GROUP BY clause to group the unique Mission_Outcome and its count with COUNT clause

Boosters Carried Maximum Payload

```
%sql SELECT Booster_Version FROM SPACEXTBL WHERE PAYLOAD_MASS__KG_ = (SELECT MAX(PAYLOAD_MASS__KG_) FROM SPACEXTBL);
```

```
 * sqlite:///spacex.db3
Done.
```

Booster_Version
F9 FT B1029.1
F9 FT B1036.1
F9 B4 B1041.1
F9 FT B1036.2
F9 B4 B1041.2
F9 B5B1048.1
F9 B5 B1049.2

Here we use subquery to get the maximum Payload and the clause WHERE to filter only Booster version that has Payload equal the maximum.

2015 Launch Records

```
%sql SELECT Landing_Outcome, Booster_Version, Launch_Site FROM SPACEXTBL WHERE Mission_Outcome like '%Fail%'
     AND Date like '%2015%';
```

```
 * sqlite:///spacex.db3
Done.
```

Landing_Outcome	Booster_Version	Launch_Site
Precluded (drone ship)	F9 v1.1 B1018	CCAFS LC-40

Here we use the % symbol to filter Mission Outcomes that has failed and happens in 2015 year.

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

```
%sql SELECT Landing_Outcome, COUNT(*) as qtd FROM SPACEXTBL
      WHERE substr(Date,7)||substr(Date,4,2)||substr(Date,1,2) between '20100604' and '20170320'
      GROUP BY Landing_Outcome ORDER BY qtd DESC;
```

```
 * sqlite:///spacex.db3
Done.
```

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

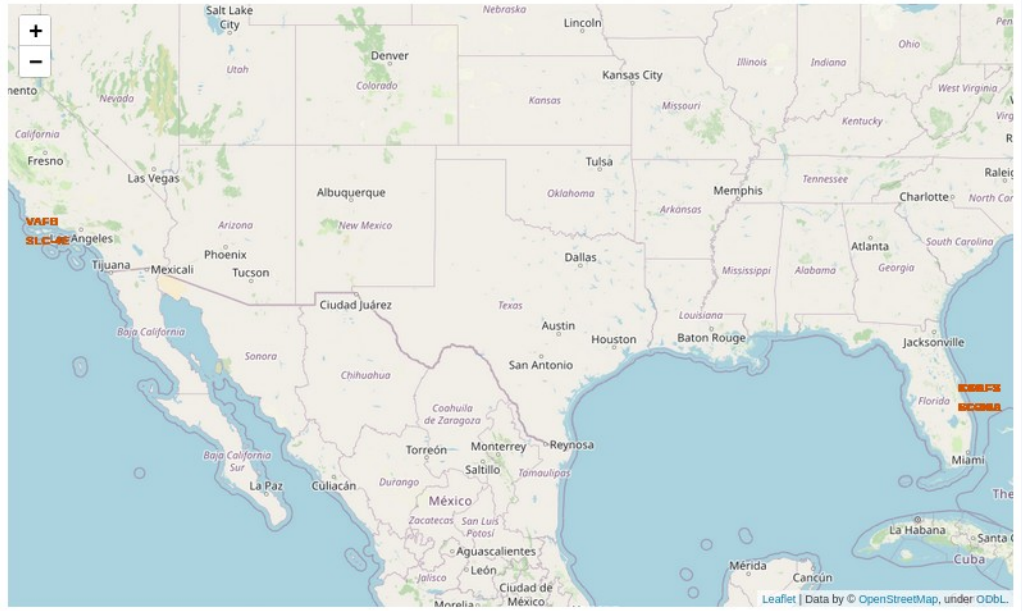
Here we use GROUP BY to group unique Landing_Outcome and COUNT to count its records. We use substr function to facilitate filter date as string.

Section 4

Launch Sites Proximities Analysis

All launch sites global map markers

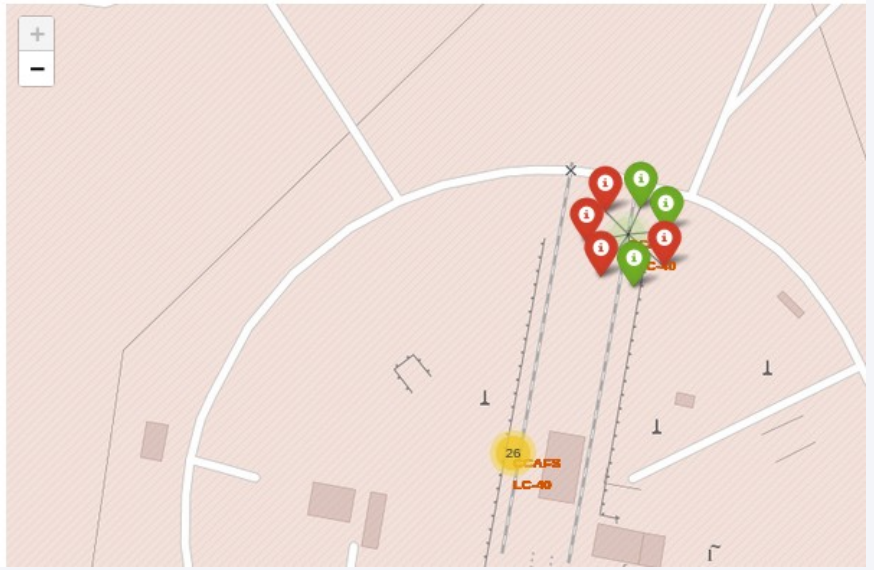
```
# Initial the map
site_map = folium.Map(location=nasa_coordinate, zoom_start=5)
# For each launch site, add a Circle object based on its coordinate (Lat, Long) values. In addition, add Launch site
for index, linha in spacex_df.iterrows():
    coordinate = [linha['Lat'], linha['Long']]
    #circle = folium.Circle(coordinate, radius=1000, color='#d35400', fill=True)
    circle.add_child(folium.Popup(linha['Launch Site']))

    marker = folium.map.Marker(
        coordinate,
        # Create an icon as a text label
        icon=DivIcon(
            icon_size=(2,2),
            icon_anchor=(0,0),
            html='<div style="font-size: 12; color:#d35400;"><b>%s</b></div>' % linha['Launch Site'],
            )
        )
    #site_map.add_child(circle)
    site_map.add_child(marker)
site_map
```



We can see that the SpaceX launch sites are in the United States of America coasts. Florida and California

Colour Labelled Markers

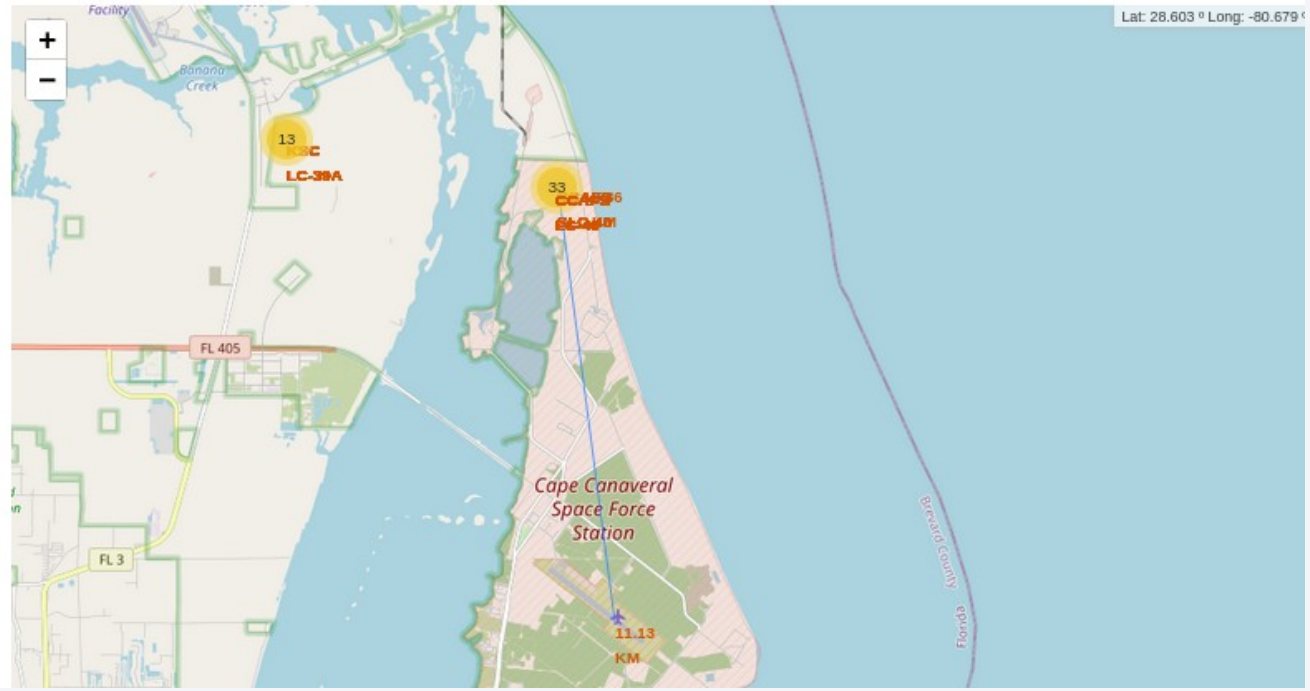
```
# Add marker_cluster to current site_map
site_map.add_child(marker_cluster)

# for each row in spacex_df data frame
# create a Marker object with its coordinate
# and customize the Marker's icon property to indicate if this launch was successed or fa
# e.g., icon=folium.Icon(color='white', icon_color=row['marker_color']
for index, record in spacex_df.iterrows():
    # TODO: Create and add a Marker cluster to the site map
    coordinate = [record['Lat'], record['Long']]
    marker = folium.map.Marker(
        coordinate,
        popup='{} / {}'.format(record['Launch Site'], coordinate),
        icon=folium.Icon(color=record['marker_color'])
        )
    marker_cluster.add_child(marker)

site_map
```

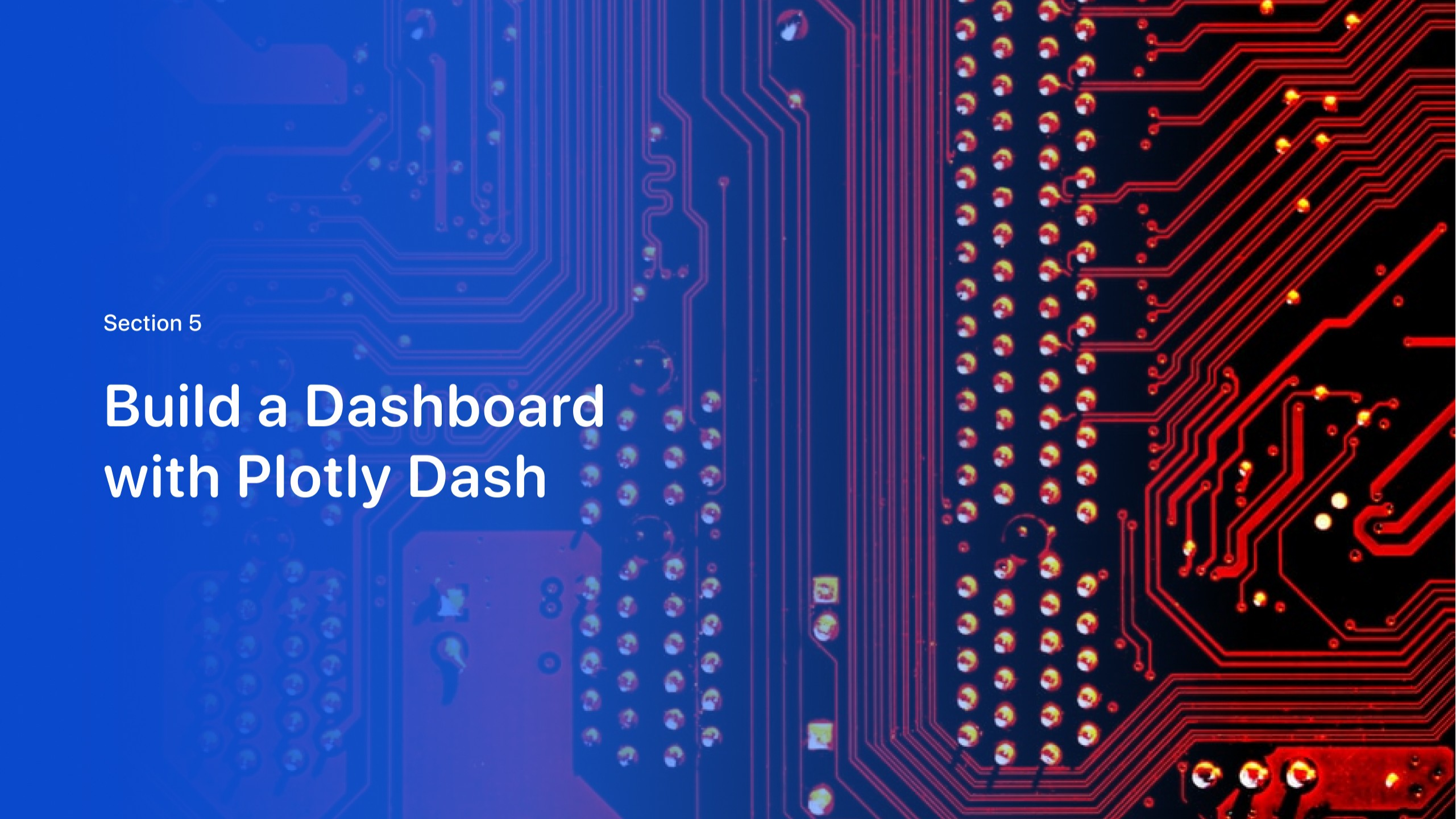


Green Marker shows successful Launches and Red Marker shows Failures

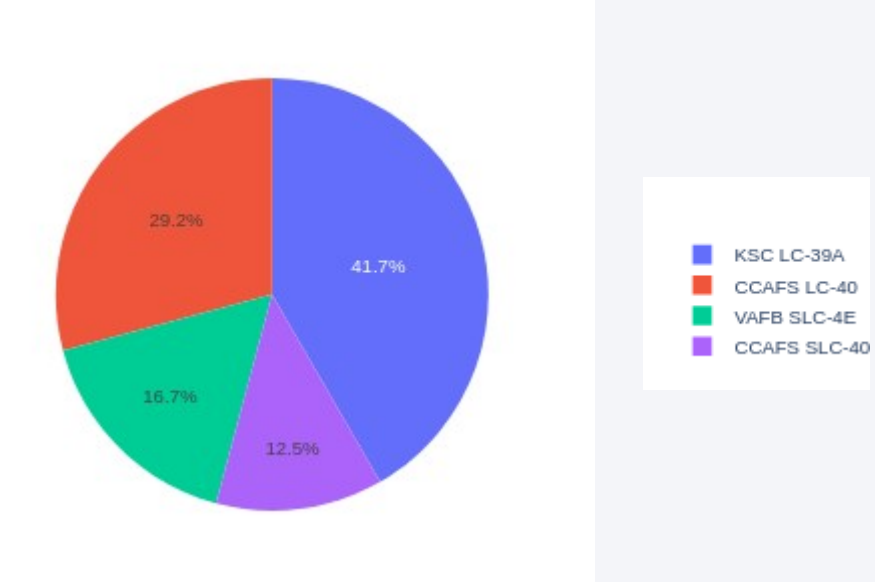
Launch Sites distance to the nearest airport

```
distance_airport = calculate_distance(28.563197, -80.576820, 28.464, -80.562)
coordinate = [28.464, -80.562]
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".format(distance_airport),
        )
    )
site_map.add_child(distance_marker)

coordinates = [[28.563197, -80.576820], [28.464, -80.562]]
lines=folium.PolyLine(locations=coordinates, weight=1)
site_map.add_child(lines)
site_map
```



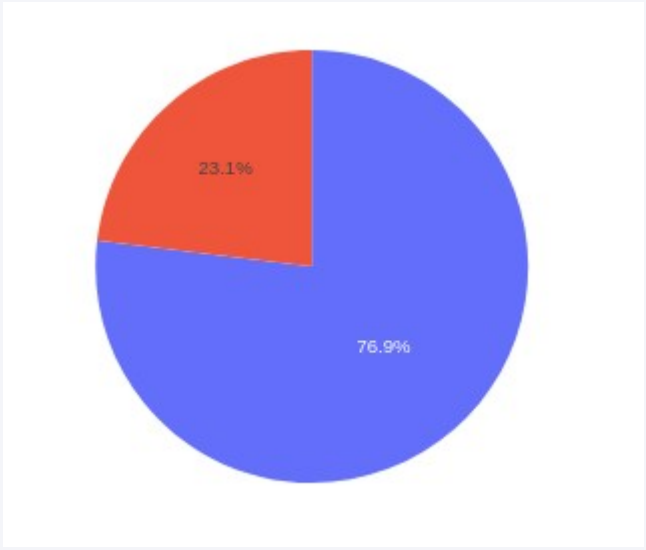

Section 5
Build a Dashboard with Plotly Dash

DASHBOARD – Pie chart showing the success percentage achieved by each launch site



We can see that KSC LC-39A had the most successful launches from all the sites

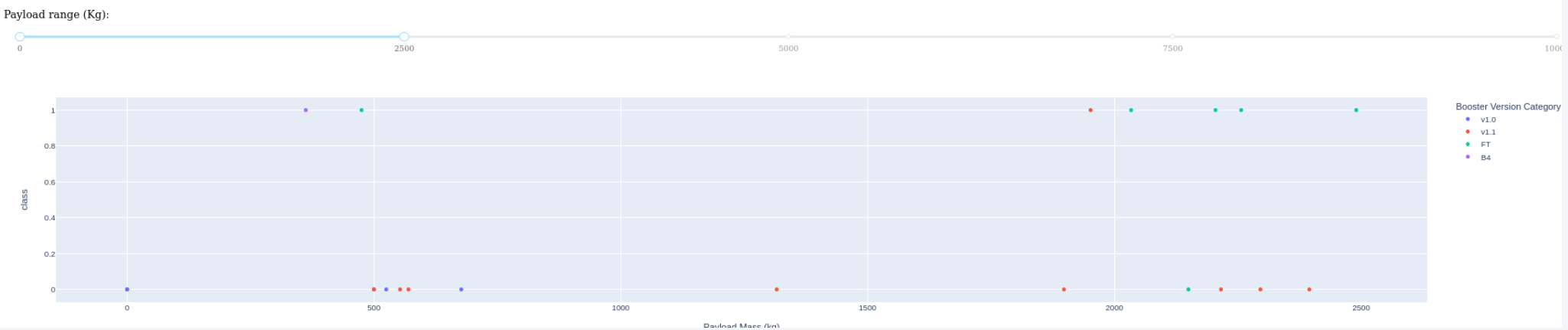
Pie chart for the launch site with highest launch success ratio



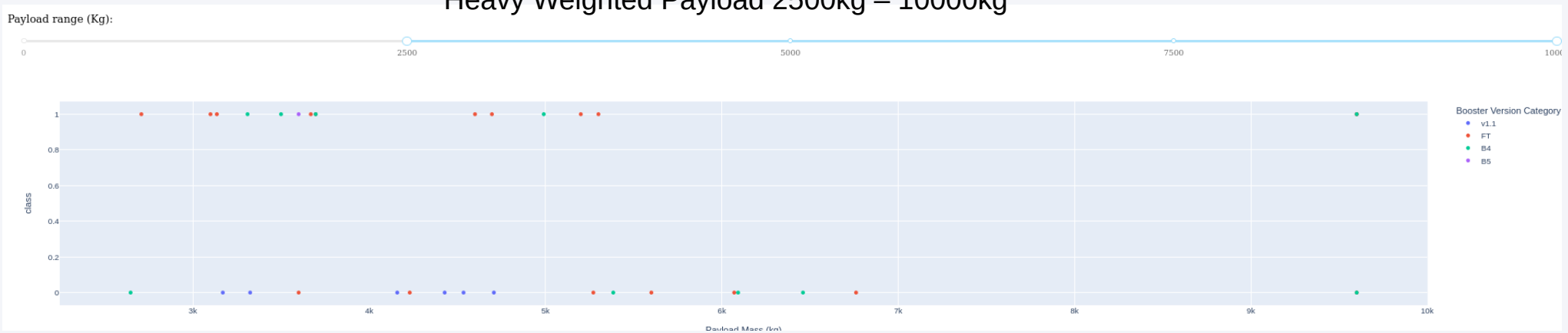
KSC LC-39A achieved a 76.9% success rate while getting a 23.1% failure rate

Payload vs. Launch Outcome scatter plot for all Sites, with different payload selected screenshot in the range slider

Low Weighted Payload 0kg – 2500kg



Heavy Weighted Payload 2500kg – 10000kg



Section 6

Predictive Analysis (Classification)

Classification Accuracy

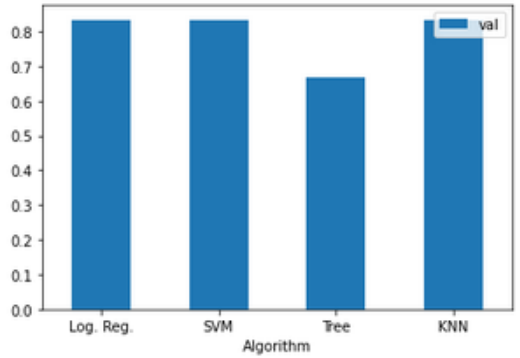
```
scores = [logreg_cv.best_score_, svm_cv.best_score_, tree_cv.best_score_, knn_cv.best_sc
scores_test = [logreg_cv.score(X_test, Y_test), svm_cv.score(X_test, Y_test), tree_cv.sc
```

```
df = pd.DataFrame({'Algorithm':['Log. Reg.', 'SVM', 'Tree', 'KNN'], 'val':scores_test})
df.head()
```

	Algorithm	val
0	Log. Reg.	0.833333
1	SVM	0.833333
2	Tree	0.666667
3	KNN	0.833333

Authors

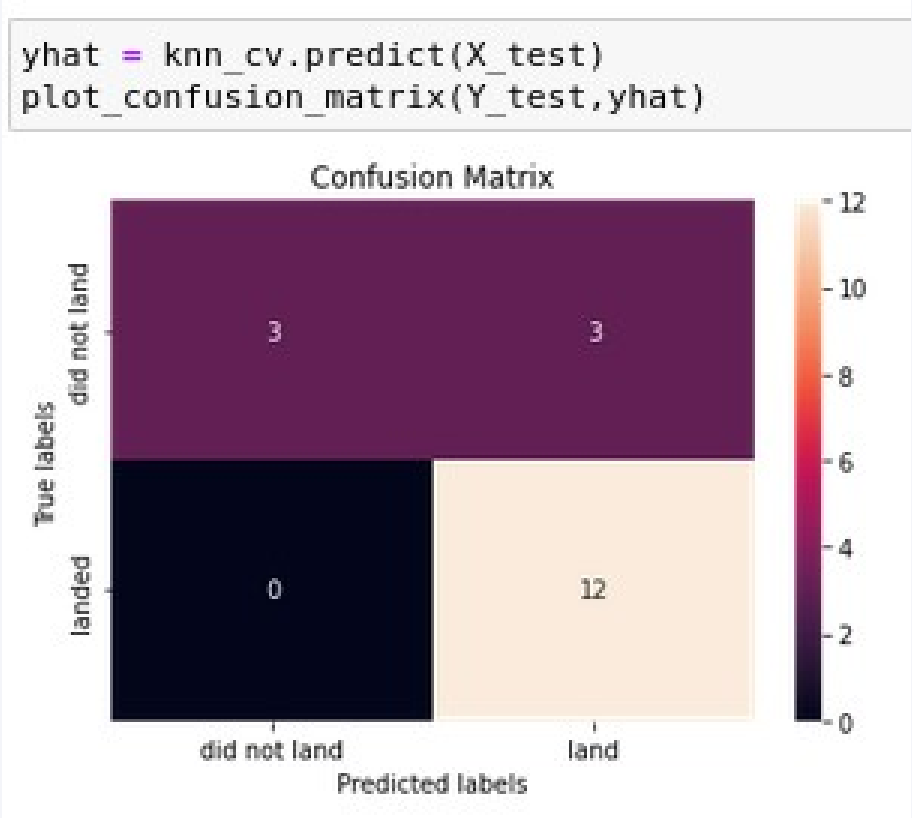
```
ax = df.plot.bar(x='Algorithm', y='val', rot=0)
```



The Logistic Regression, SVM and KNN algorithms had the same value for the score with test data.

Confusion Matrix

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



Examining the confusion matrix, we see that there was few false positives.

Conclusions

- There was a tie between the Logistic Regression, SVM and KNN algorithms.
- Low weighted payloads perform better than the heavier payloads
- The success rates for SpaceX launches is directly proportional time in years they will eventually perfect the launches
- We can see that KSC LC-39A had the most successful launches from all the sites
- Orbit GEO,HEO,SSO,ES-L1 has the best Success Rate

Appendix

I created a SQLite database to do the SQL queries. I imported de csv file to that database.

Thank you!