BY ADIL

# ANALYZING SPACEX ROCKET DATA


WE WANT:
PREDICT WILL FIRST STAGE LAND OR NOT DEPENDS ON ROCKET CHARACTERISTICS

# IN THIS PROJECT WE WILL:

Collect data using APIs and Web Scraping

Process data

Apply EDA (Exploratory Data Analysis) using SQL, python and visualization techniques

Create interactive Visual Analitycs and Dashboard

Finally apply machine learning algorithms to predict will rocket land or not

# COLLECTING AND PROCESSING DATA

- SpaceX provides their API (https://api.spacexdata.com) for getting access to data, data about all launches, its outcomes and characteristics of a rocket
- We will observe only one Booster Version, most succesful one, and that is Falcon 9
- After connecting to the SpaceX API and using `pd.json_normalize` function we get dataset which contains unusual data

# HOW DATA LOOKS LIKE

We can see that rocket column contain string, that string is link to table with data about certain rocket

In dataset there is a lot of that type of links

We need to extract data from links and collect them in one dataset

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

| | 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 | [] | [] | [] | [5eb0e4b5b6c |
| **1** | None | NaN | False | 0.0 | 5e9d0d95eda69955f709d1eb | False | [{'time': 301, 'altitude': 289, 'reason': 'harmonic oscillation leading to premature engine shutdown'}] | Successful first stage burn and transition to second stage, maximum altitude 289 km, Premature engine shutdown at T+7 min 30 s, Failed to reach orbit, Failed to recover first stage | [] | [] | [] | [5eb0e4b6b6c |
| **2** | None | NaN | False | 0.0 | 5e9d0d95eda69955f709d1eb | False | [{'time': 140, 'altitude': 35, 'reason': 'residual stage-1 thrust led | Residual stage 1 thrust led to collision between stage 1 and | [] | [] | [] | [5eb0e4b6b6c 5eb0e4b6b6c |

# FILTERING AND CLEANING

```
# Lets take a subset of our dataframe keeping only the features we want and the flight number, and date_utc.
data = data[['rocket', 'payloads', 'launchpad', 'cores', 'flight_number', 'date_utc']]

# We will remove rows with multiple cores because those are
# falcon rockets with 2 extra rocket boosters and rows that have multiple payloads in a single rocket.
data = data[data['cores'].map(len)==1]
data = data[data['payloads'].map(len)==1]

# Since payloads and cores are lists of size 1 we will also extract the single value in the list and replace the feature.
data['cores'] = data['cores'].map(lambda x : x[0])
data['payloads'] = data['payloads'].map(lambda x : x[0])

# We also want to convert the date_utc to a datetime datatype and then extracting the date leaving the time
data['date'] = pd.to_datetime(data['date_utc']).dt.date

# Using the date we will restrict the dates of the launches
data = data[data['date'] <= datetime.date(2020, 11, 13)]
```

- From the `rocket` we would like to learn the booster name
- From the `payload` we would like to learn the mass of the payload and the orbit that it is going to
- From the `launchpad` we would like to know the name of the launch site being used, the longitude, and the latitude.
- From `cores` we would like to learn the outcome of the landing, the type of the landing, number of flights with that core, whether gridfins were used, whether the core is reused, whether legs were used, the landing pad used, the block of the core which is a number used to seperate version of cores, the number of times this specific core has been reused, and the serial of the core.

# FUNCTIONS TO EXTRACT DATA FROM LINKS

Some few functions as example how we extract data from links

From the `rocket` column we would like to learn the booster name.

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

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

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

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

```
#Global variables
BoosterVersion = []
PayloadMass = []
Orbit = []
LaunchSite = []
Outcome = []
Flights = []
GridFins = []
Reused = []
Legs = []
LandingPad = []
Block = []
ReusedCount = []
Serial = []
Longitude = []
Latitude = []
```

# CREATE DATAFRAME WITH WHOLE DATA WE NEEDED

After that we will filter df to get data only about Falcon 9 and replace missing data with mean values

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

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

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

# COLLECTING AND PORCESSING DATA

- Also we can use websraping, and wikipedia (https://en.wikipedia.org/w/index.php?title=List_of_Falcon_9_and_Falcon_Heavy_launches&oldid=1027686922) as data source
- In wikipedia data already prepared and don't need to be precessed
- As always using bs4 for webscraping, same steps for collecting we get usefull dataset
- That is boring lets go forward

- If some one have interest about how everything is done using webscraping, can looks to my github

# EDA AND SQL:

List the total number of successful and failure mission outcomes

```
%%sql
select (select count(*) from spacextable
        where mission_outcome like "%Success%") as successful,
       (select count(*) from spacextable
        where mission_outcome like "%Failure%") as failure,
        (select count(*) from spacextable) as total
```

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

| successful | failure | total |
|---|---|---|
| 100 | 1 | 101 |

```
select distinct(booster_version) from spacextable
where payload_mass__kg_ == (select max(payload_mass__kg_) from spacextable)
```

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

| Booster_Version |
|---|
| F9 B5 B1048.4 |
| F9 B5 B1049.4 |
| F9 B5 B1051.3 |
| F9 B5 B1056.4 |
| F9 B5 B1048.5 |
| F9 B5 B1051.4 |
| F9 B5 B1049.5 |
| F9 B5 B1060.2 |
| F9 B5 B1058.3 |

- SQL is useful thing. It helps us more understand data and answer some questions.
- For example, here we observe total number of successful/failure mission outcomes and which booster version caried maximum payload mass

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

```
%%sql
select sum(payload_mass__kg_) from spacextable
where customer == "NASA (CRS)"
```

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

| sum(payload_mass__kg_) |
|---|
| 45596 |

```
%%sql
select distinct(launch_site) from spacextable;
```

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

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

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

```
%%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__KG_ | Orbit | Customer | Mission_Outcome | Landing_Outcc |
|---|---|---|---|---|---|---|---|---|---|
| 2010-04-06 | 18:45:00 | F9 v1.0 B0003 | CCAFS LC-40 | Dragon Spacecraft Qualification Unit | 0 | LEO | SpaceX | Success | Failure (parach |
| 2010-08-12 | 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 (parach |
| 2012-05-22 | 07:44:00 | F9 v1.0 B0005 | CCAFS LC-40 | Dragon demo flight C2 | 525 | LEO (ISS) | NASA (COTS) | Success | No atte |
| 2012-08-10 | 00:35:00 | F9 v1.0 B0006 | CCAFS LC-40 | SpaceX CRS-1 | 500 | LEO (ISS) | NASA (CRS) | Success | No atte |
| 2013-01-03 | 15:10:00 | F9 v1.0 B0007 | CCAFS LC-40 | SpaceX CRS-2 | 677 | LEO (ISS) | NASA (CRS) | Success | No atte |

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

```
%%sql
select avg(payload_mass__kg_) from spacextable
where booster_version like "F9 v1.1%"
```

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

| avg(payload_mass__kg_) |
| --- |
| 2534.6666666666665 |

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

*Hint:Use min function*

```
%%sql
select "Date" from spacextable
where landing_outcome == "Success"
limit 1
```

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

| Date |
| --- |
| 2018-07-22 |

# Task 6

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

```
%%sql
select distinct(booster_version) from spacextable
where payload_mass__kg_ > 4000 and payload_mass__kg_ < 6000
        and landing_outcome like "%Success%"
        and landing_outcome like "%drone ship%"
```

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

| Booster_Version |
|---|
| F9 FT B1022 |
| F9 FT B1026 |
| F9 FT B1021.2 |
| F9 FT B1031.2 |

List the total number of successful and failure mission outcomes

```
%%sql
select (select count(*) from spacextable
        where mission_outcome like "%Success%") as successful,
       (select count(*) from spacextable
        where mission_outcome like "%Failure%") as failure,
        (select count(*) from spacextable) as total
```

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

| successful | failure | total |
|---|---|---|
| 100 | 1 | 101 |

```sql
%%sql
select substr("Date",6,2) as month , landing_outcome, booster_version, launch_site from spacextable
where landing_outcome like "%Failure%" and landing_outcome like "%drone ship%"
       and substr("Date",0,5)=='2015'
```

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

| month | Landing_Outcome | Booster_Version | Launch_Site |
|---|---|---|---|
| 10 | Failure (drone ship) | F9 v1.1 B1012 | CCAFS LC-40 |
| 04 | Failure (drone ship) | F9 v1.1 B1015 | CCAFS LC-40 |

```sql
%%sql
select distinct(Landing_Outcome), count(*) as "count" from spacextable
where   "Date" < '2017-03-20' and "Date" > '2010-06-04'
group by Landing_Outcome
order by "count" desc
```

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

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

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

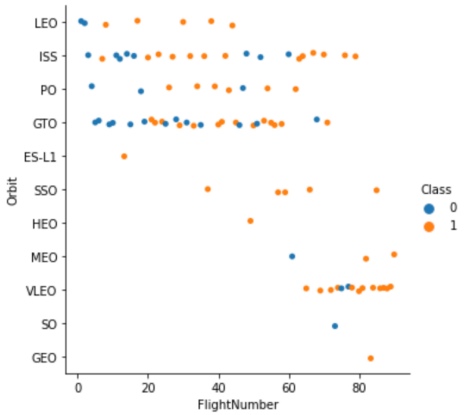
```
%%sql
select distinct(booster_version) from spacextable
where payload_mass__kg_ == (select max(payload_mass__kg_) from spacextable)
```

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

| Booster_Version |
| --- |
| F9 B5 B1048.4 |
| F9 B5 B1049.4 |
| F9 B5 B1051.3 |
| F9 B5 B1056.4 |
| F9 B5 B1048.5 |
| F9 B5 B1051.4 |
| F9 B5 B1049.5 |
| F9 B5 B1060.2 |

# EDA AND VISUALIZATION

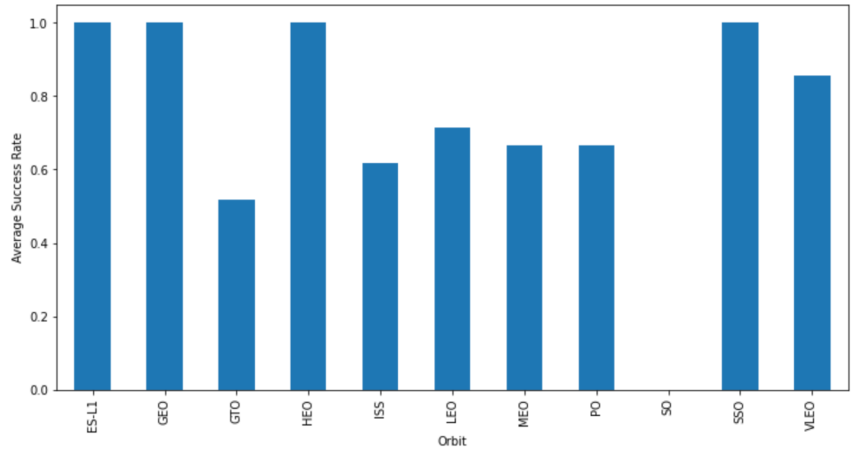
```
# Plot a scatter point chart with x axis to be FlightNumber and y axis to be the Orbit, and hue to be the class value
sns.catplot(data=df, x='FlightNumber', y='Orbit', hue='Class')
plt.show()
```




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.

```
# HINT use groupby method on Orbit column and get the mean of Class column
gr = df.groupby('Orbit')['Class'].mean()

gr.plot(kind='bar', figsize=(12,6))
plt.xlabel('Orbit')
plt.ylabel('Average Success Rate')
plt.show()
```

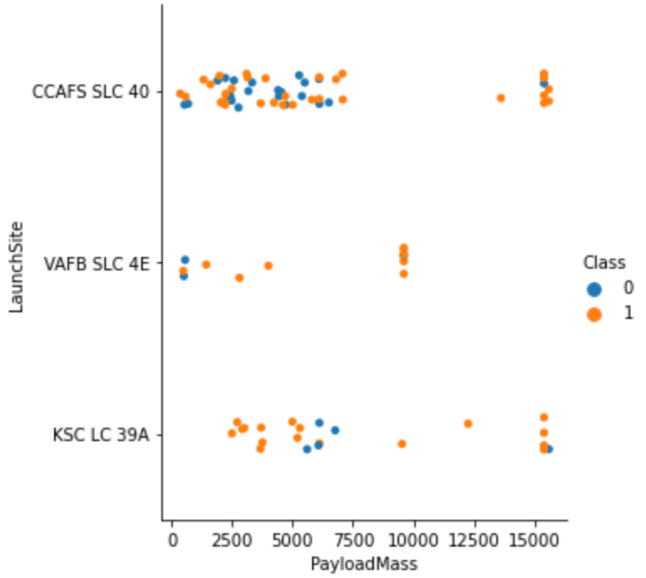



Analyze the ploted bar chart try to find which orbits have high sucess rate.

# USING EDA WE CAN ANSWER:

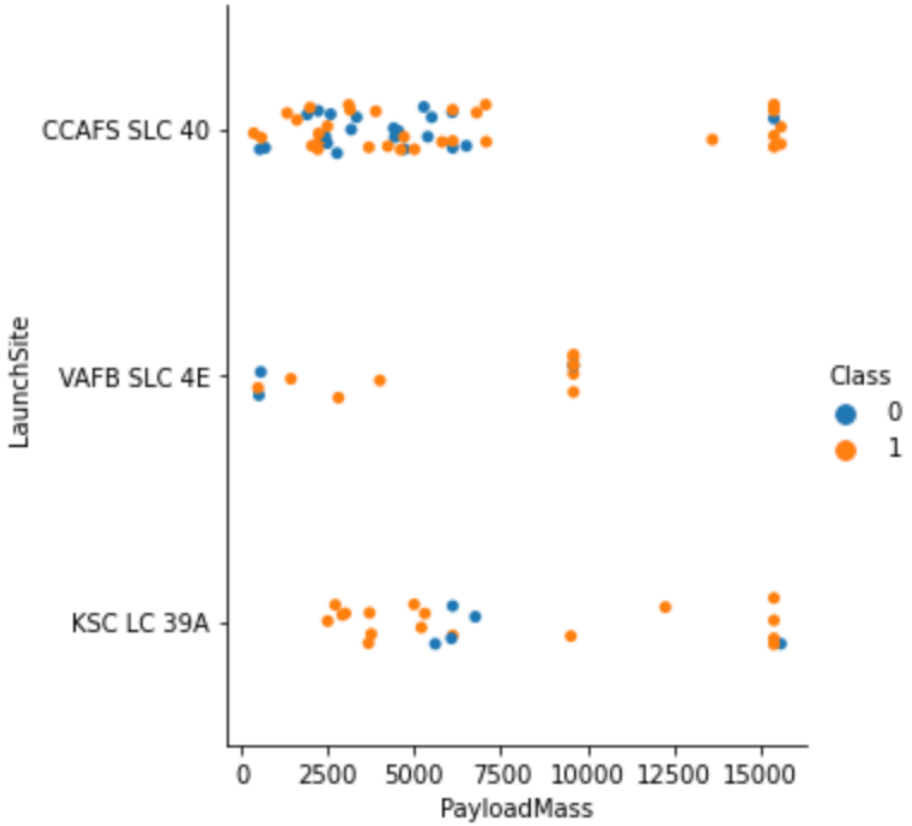
We also want to observe if there is any relationship between launch sites and their payload mass.

```
# 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 class value
sns.catplot(data=df, y='LaunchSite', x='PayloadMass', hue='Class')
plt.show()
```



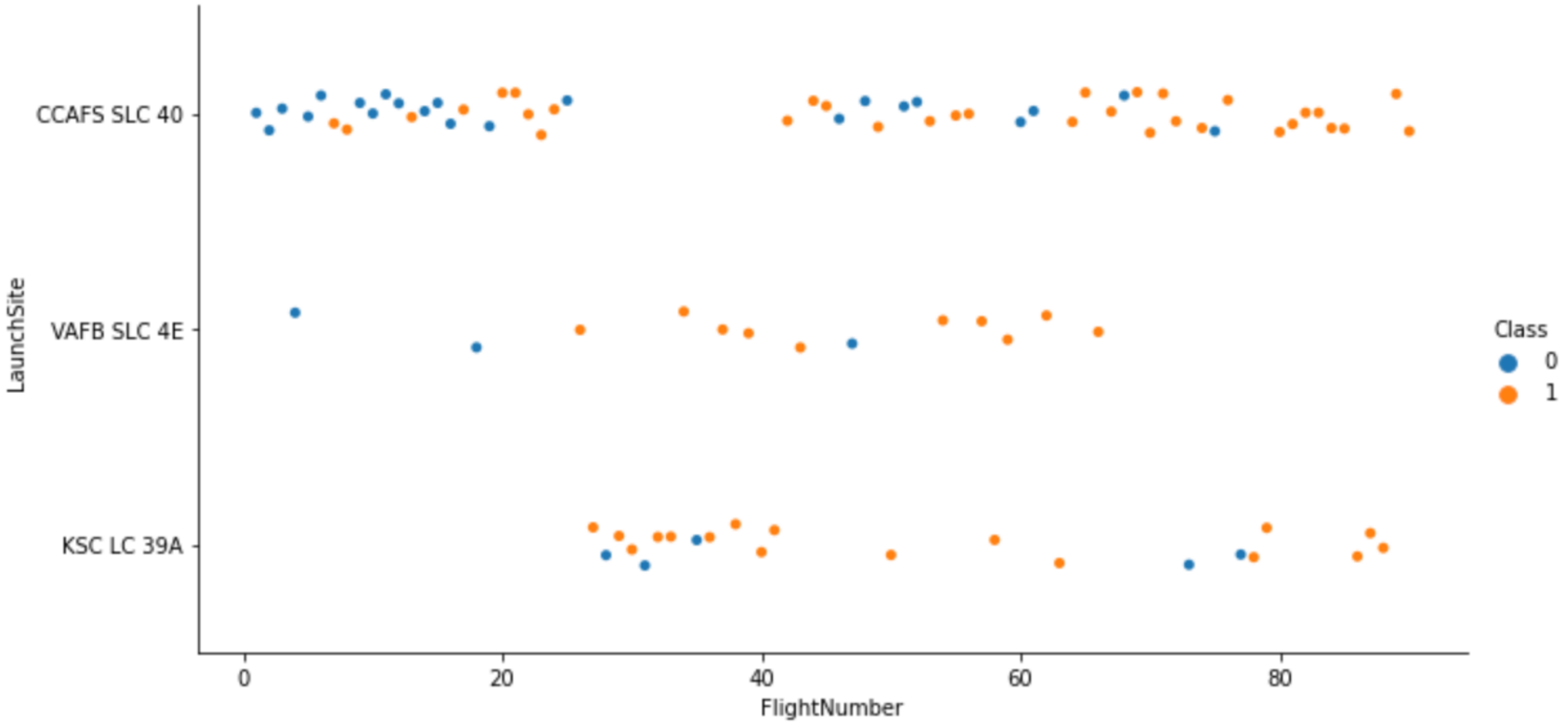

With what weight are the missions success?
(between 9000 and 13000 kg)

```
# 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 class value
sns.catplot(data=df, y='LaunchSite', x='PayloadMass', hue='Class')
plt.show()
```



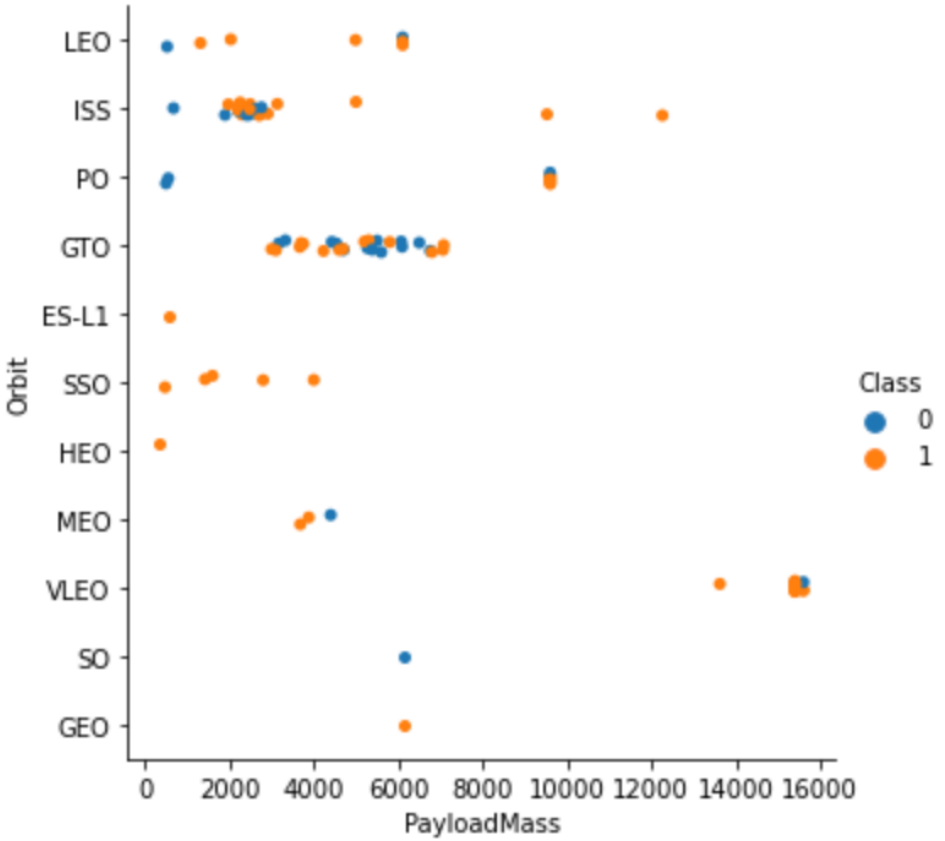

Now if you observe Payload Vs. Launch Site scatter point chart you will find for the VAFB-SLC launchsite there are no rockets launched for heavypaylo
than 10000).

```
# 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
sns.catplot(data=df, x='FlightNumber', y='LaunchSite', hue='Class', aspect = 2)
plt.show()
```




Now try to explain the patterns you found in the Flight Number vs. Launch Site scatter point plots.

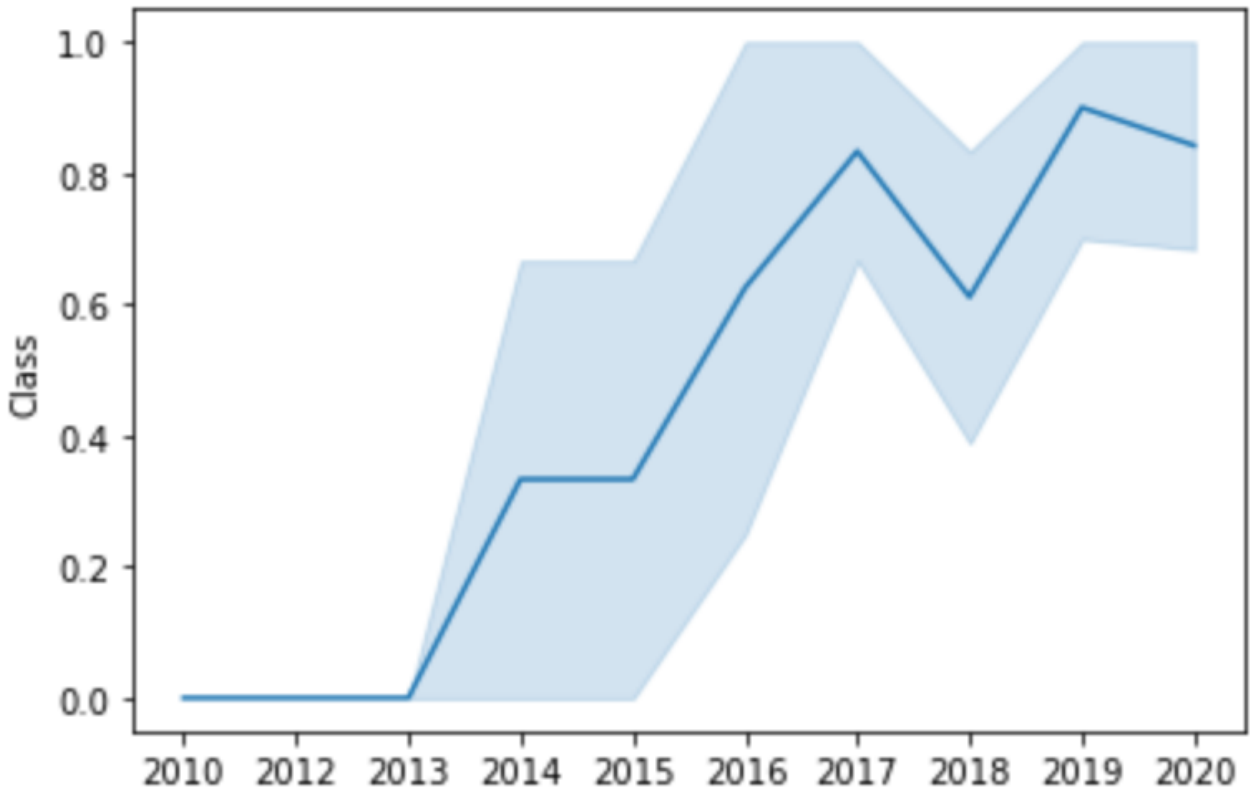
```
# Plot a scatter point chart with x axis to be Payload and y axis to be the Orbit, and hue to be the class value
sns.catplot(data=df, x='PayloadMass', y='Orbit', hue='Class')
plt.show()
```




With heavy payloads the successful landing or positive landing rate are more for Polar,LEO and ISS.

However for GTO we cannot distinguish this well as both positive landing rate and negative landing(unsuccessful mission) are both there here.

```
# Plot a line chart with x axis to be the extracted year and y axis to be the success rat
sns.lineplot(x=Extract_year(12), y=df['Class'])
plt.show()
```

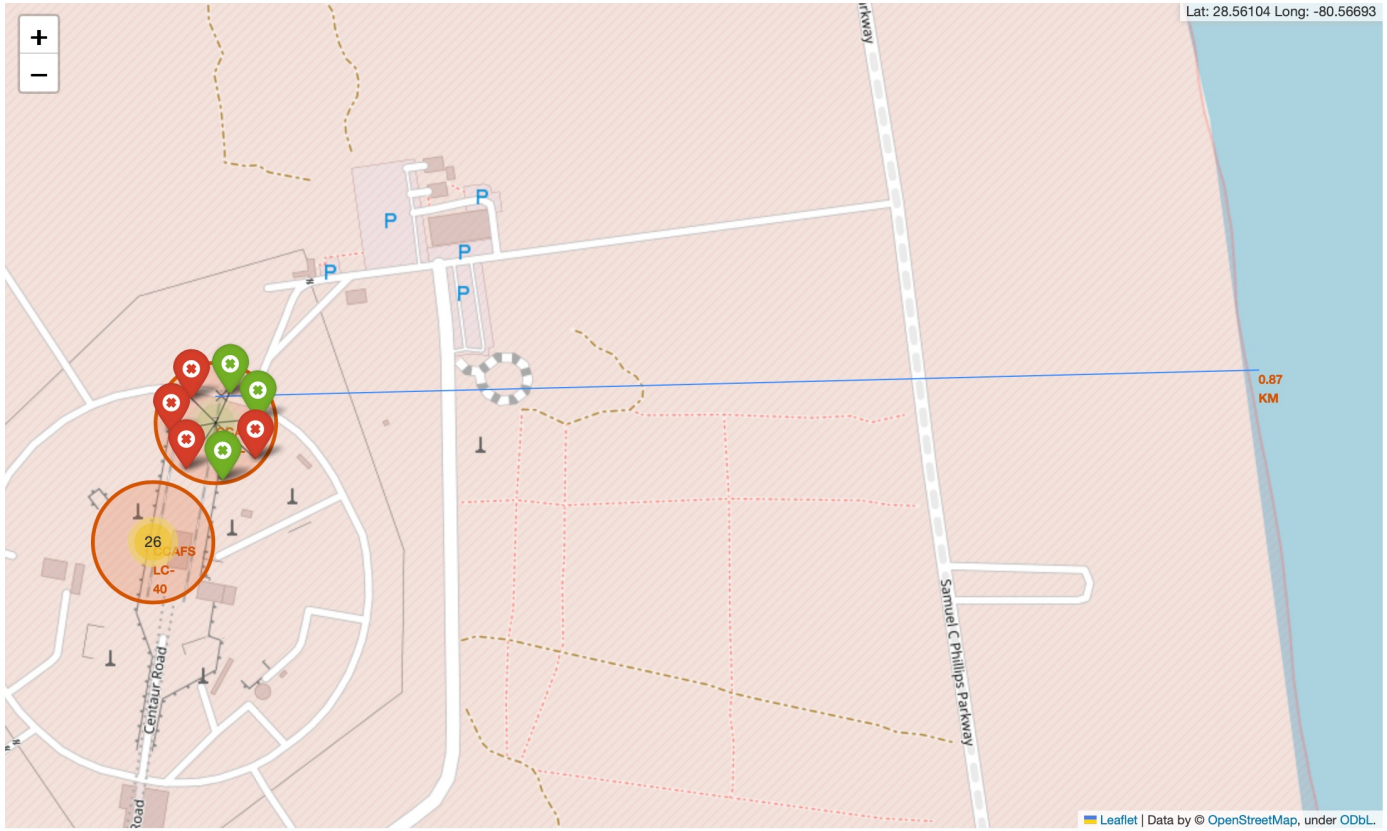



You can observe that the success rate since 2013 kept increasing till 2017 (stable in 2014) and after 2015 it

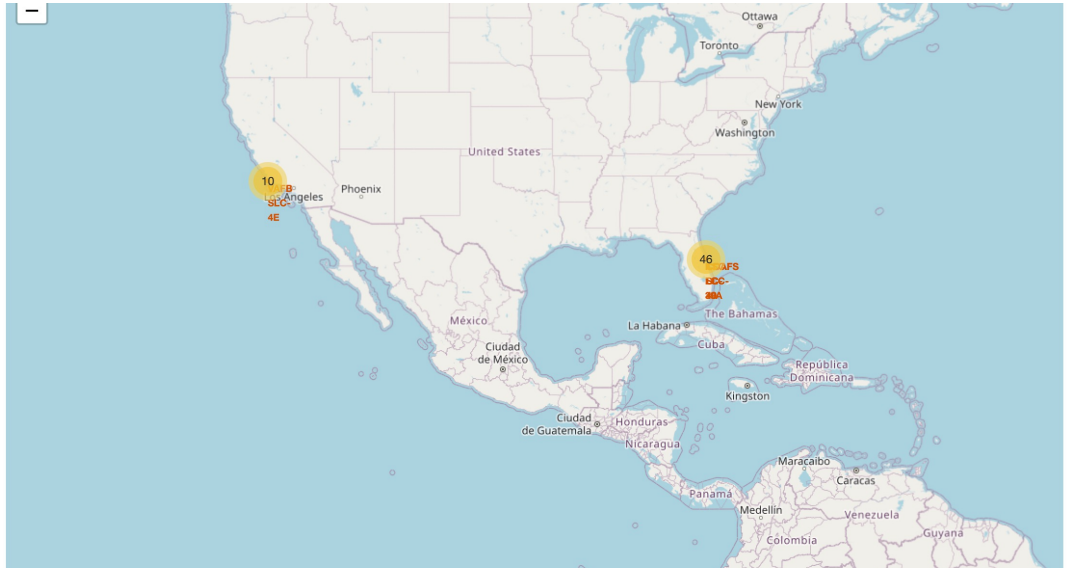
# EDA AND INTERACTIVE VISUAL ANALYTICS

Using Folium library we can display all launches on map and view info about launhces

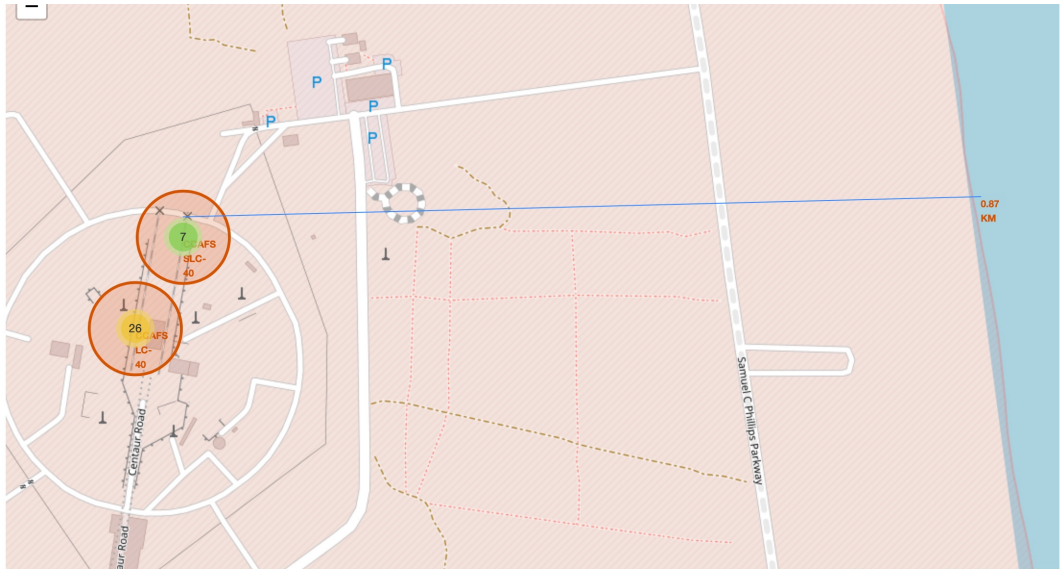
For example later we could use this info and add info from meteorological sensors to answer more difficult questions



- Using Folium we can mark all Launch Sites and provide some info about those Sites



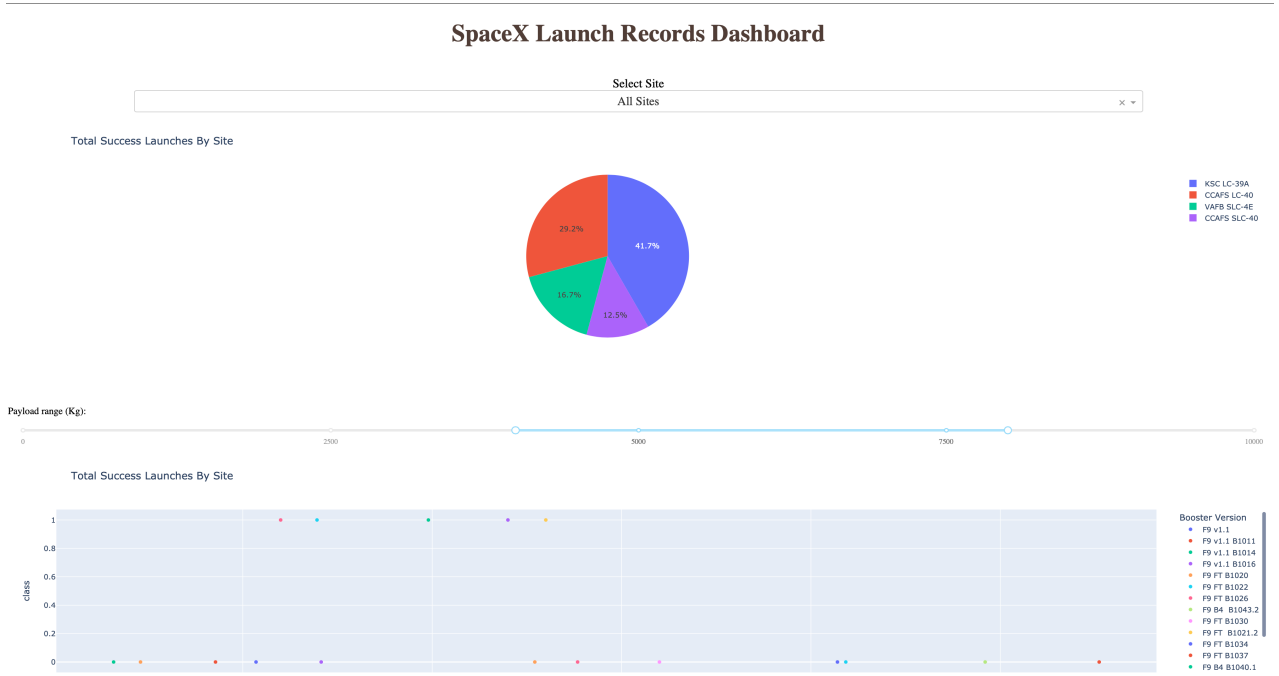
- Using Folium we can display how far from launch is coastline



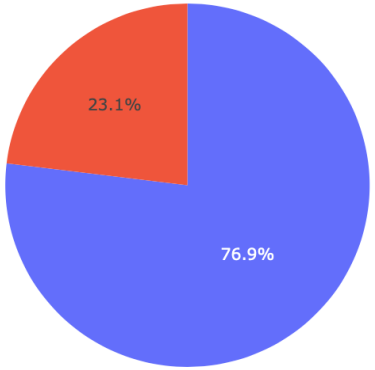
# EDA AND INTERACTIVE VISUAL ANALYTICS

- Using Dash and Plotly we can create app which could automaticly update data, so we get always fresh data and observe it
- Created app can be filtered and people who never do programing can analyse

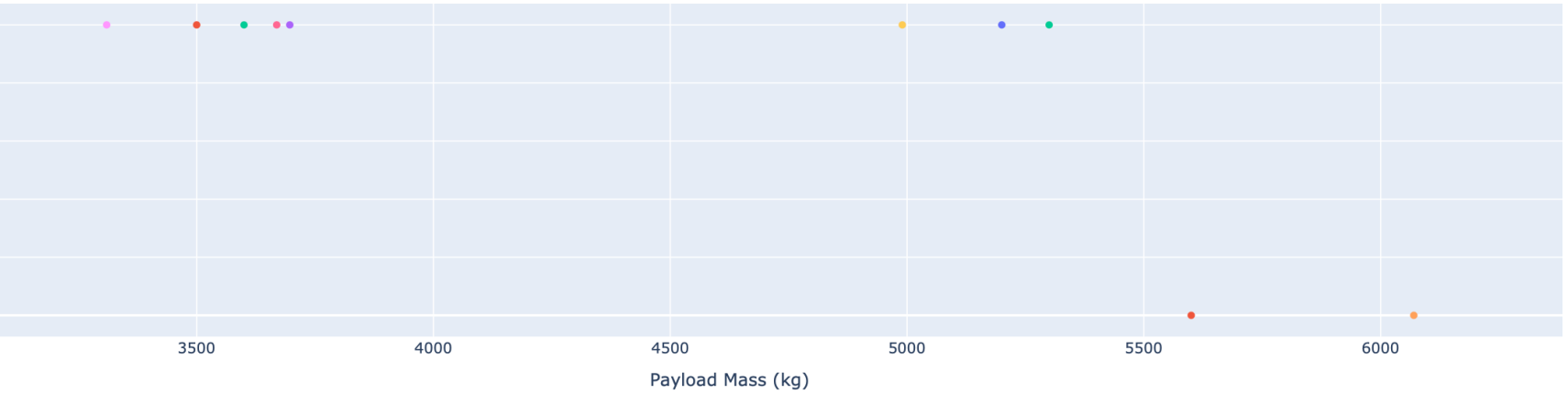
- Here we can see pie chart with Total Success by Site, winner is KSL LC-39A
- On bottom we can see scatter plot with max payload mass caried by rocket



KSC LC-39A








- This pie chart shows us best launch site (KSC LC-39A), where 76.9 percent of success launches
- On Bottom we can see payload mass caried by rocket

# PREDICTIVE ANALYSIS METHODOLOGY:

- As always split dataframe into fetures set (X) and label set (Y)
- As always use train_test_split to split datasets into training sets and test sets
- Will first stage land or not = boolean = Categorical => we will use DecisionTreeClassification, SVC (SVM), LogisticRegression, KNN
- For creating models we will apply almost every hyperparameters and for searching best hyperparameters use GridSearchCV
- We will use .score function, confusion matrix, roc-curve, auc for compating models

# BEST HYPERPARAMETERS

```
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
```

Logistic Regression

```
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', 'probability': True}
accuracy : 0.8482142857142856
```

SVC (SVM)

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

```
tuned hpyerparameters :(best parameters)  {'criterion': 'entropy', 'max_depth': 18, 'max_features': 'sqrt', 'min_samples_leaf': 2, 'min_sampl
es_split': 5, 'splitter': 'random'}
accuracy : 0.875
```

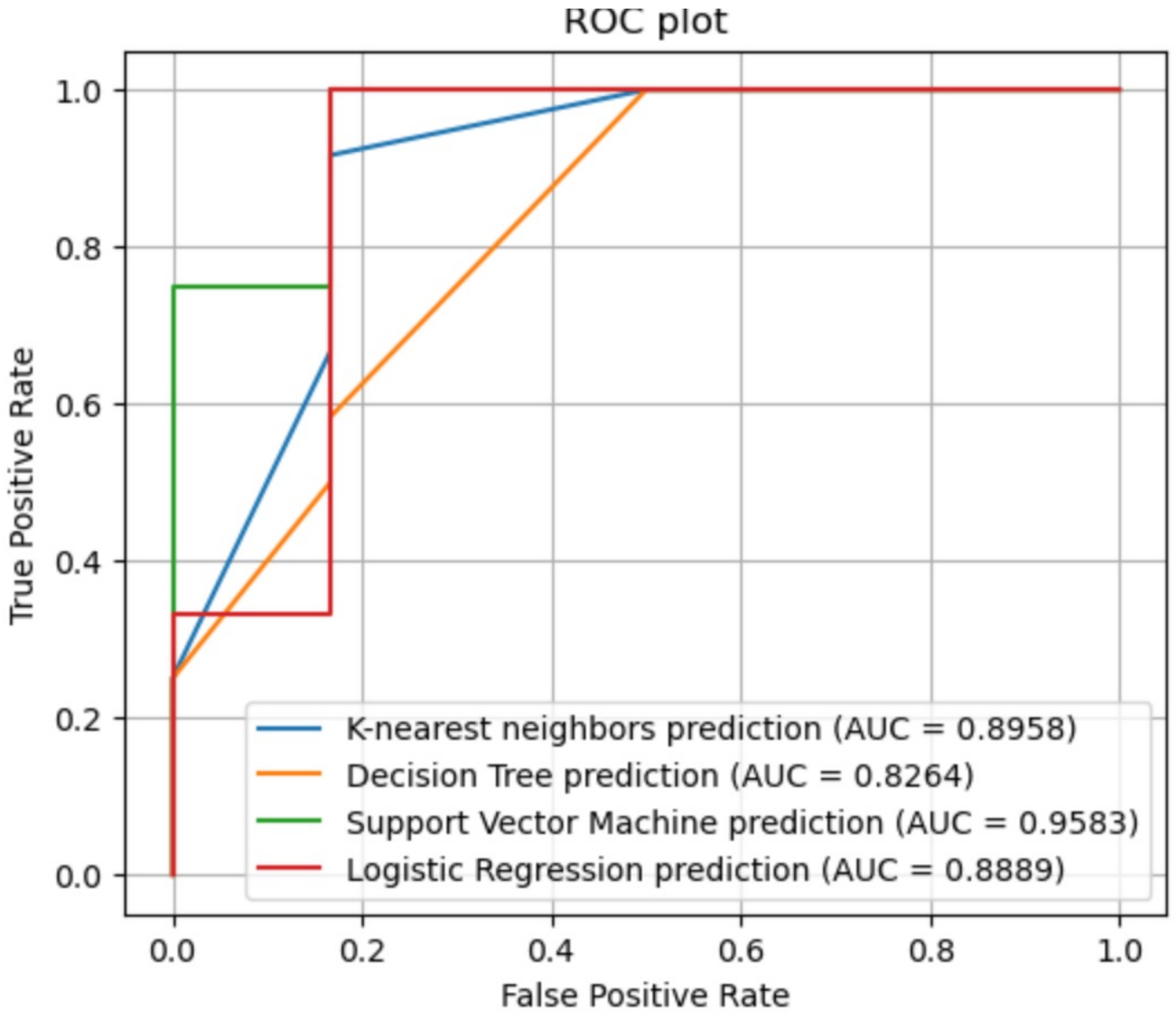
DecisionTreeClassifier

```
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
```

KNN

# ROC-CURVE


ROC plot
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
K-nearest neighbors prediction (AUC = 0.8958)
Decision Tree prediction (AUC = 0.8264)
Support Vector Machine prediction (AUC = 0.9583)
Logistic Regression prediction (AUC = 0.8889)

# FUNNY THINGS:

- Accuracy score on test set = 83.333333….. Every madel!
- Confusion Matrix is same for every model

# CONCLUSIONS

- Now we have working model for predicting will first stage land successfully or not depend on characteristics of rocket