IBM Data Science Capstone Project

SPACE X

Asmerom NOV 2021

OUTLINE

- Exclusive summary
- Introduction
- Methodology
- Result
  - Chart
  - Dashbord
- Discussion
- Conclusion

Exclusive summary

1. Data Collection
2. Web scraping
3. Data wrangling
4. EDA with SQL
5. EDA with data visualisation
6. Location analysis with Folium
7. Dashboard
8. Prediction Classification

Introduction

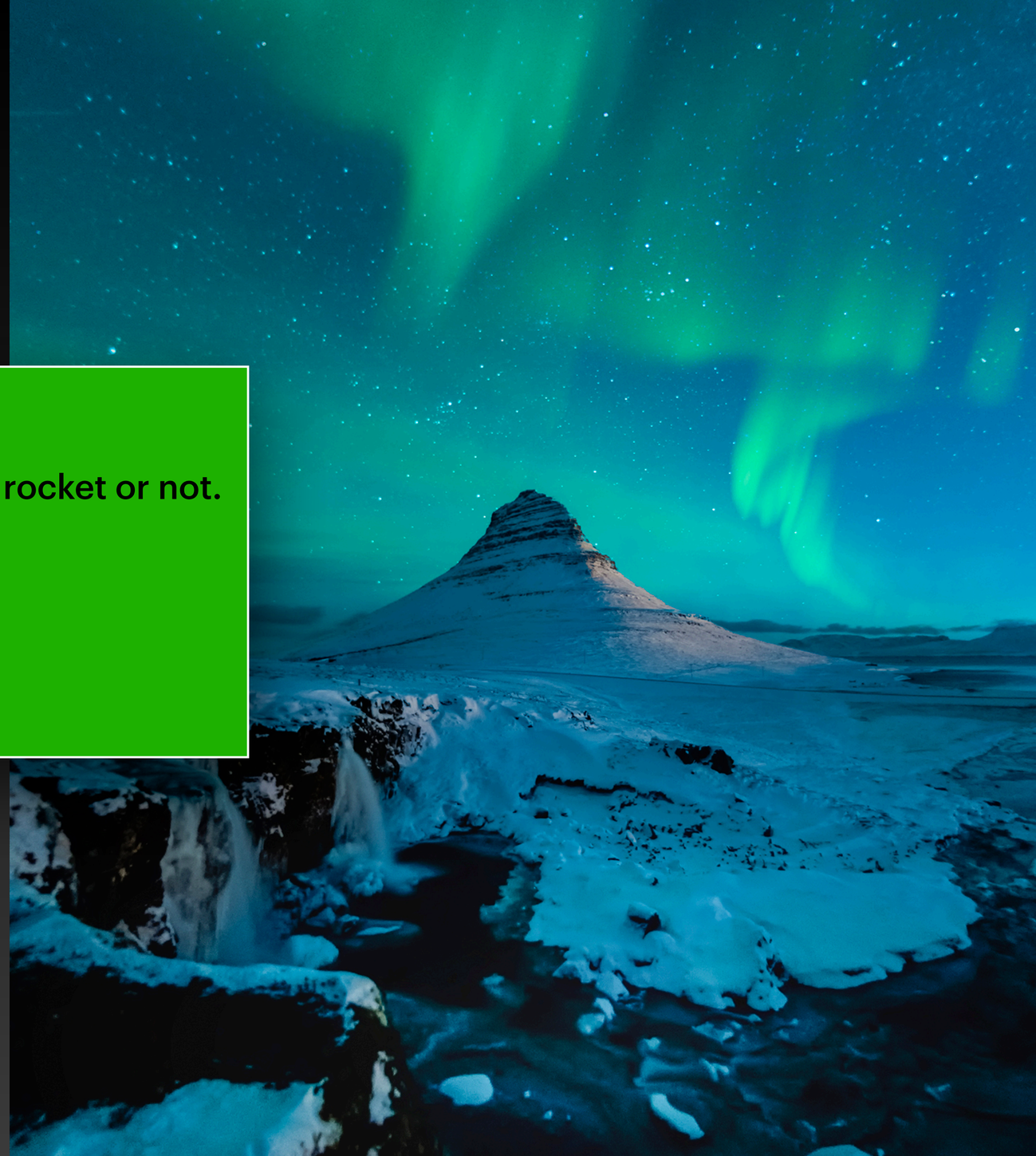
Data Collection and Processing

- The quality of data further improved by data wrangling.
- To explore the dataset in-depth sql used to query so to have insight. Basic
- exploratory statistics applied to analyse the dataset and visualise using python.
- To further drill down deep in to the dataset, the dataset further grouped into different categories.
- Using Folium the proximity to launch site analysed.
- With the help of plot dash board application different interactive plots displayed.
- And later applied models to discover more predictive exciting insights.

Study Goal

Preliminary goal of this study is:

- to predict whether SpaceX will attempt to land a rocket or not.
- To predict, If first stage can land successfully.
- Optimise launch site and proximities.



Exploratory data analysis result

- Success rate increases since 2013
- CCAFS LC-40 found to be 40% success rate
- KSC LC-39A with 77% success rate.
- However the mass above 10,000kg found to have 100% success rate.

Methodology

Data Collection and Wrangling

- Request and parse data - using GET request
- Filter data frame - only Falson 9 launches
- Dealing with missing dataset

```
# 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']:
        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']:
        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.

```
# Takes the dataset and uses the payloads column to call the API and append the data to the lists
def getPayloadData(data):
    for load in data['payloads']:
        response = requests.get("https://api.spacexdata.com/v4/payloads/"+load).json()
        PayloadMass.append(response['mass_kg'])
        Orbit.append(response['orbit'])
```

```
# Takes the dataset and uses the cores column to call the API and append the data to the lists
def getCoreData(data):
    for core in data['cores']:
            if core['core'] != None:
                response = requests.get("https://api.spacexdata.com/v4/cores/"+core['core']).json()
                Block.append(response['block'])
                ReusedCount.append(response['reuse_count'])
                Serial.append(response['serial'])
            else:
                Block.append(None)
                ReusedCount.append(None)
                Serial.append(None)
            Outcome.append(str(core['landing_success'])+' '+str(core['landing_type']))
            Flights.append(core['flight'])
            GridFins.append(core['gridfins'])
            Reused.append(core['reused'])
            Legs.append(core['legs'])
            LandingPad.append(core['landpad'])
```

We can see below that some of the rows are missing values in our dataset.

```
data_falcon9.isnull().sum()
```

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

```
# Calculate the mean value of PayloadMass column
mean_PlayloadMass = data_falcon9["PayloadMass"].mean()
# Replace the np.nan values with its mean value
data_falcon9["PayloadMass"]= data_falcon9["PayloadMass"].fillna(mean_PlayloadMass)
```

Web scraping

- Extract HTML data from web
- Request page from URL
- Extract variable from HTML table header
- Create data frame by parsing

```
def date_time(table_cells):
    """
    This function returns the data and time from the HTML  table cell
    Input: the  element of a table data cell extracts extra row
    """
    return [data_time.strip() for data_time in list(table_cells.strings)][0:2]

def booster_version(table_cells):
    """
    This function returns the booster version from the HTML  table cell
    Input: the  element of a table data cell extracts extra row
    """
    out=''.join([booster_version for i,booster_version in enumerate( table_cells.strings) if i%2==0][0:-1])
    return out

def landing_status(table_cells):
    """
    This function returns the landing status from the HTML table cell
    Input: the  element of a table data cell extracts extra row
    """
    out=[i for i in table_cells.strings][0]
    return out


def get_mass(table_cells):
    mass=unicodedata.normalize("NFKD", table_cells.text).strip()
    if mass:
        mass.find("kg")
        new_mass=mass[0:mass.find("kg")+2]
    else:
        new_mass=0
    return new_mass


def extract_column_from_header(row):
    """
    This function returns the landing status from the HTML table cell
    Input: the  element of a table data cell extracts extra row
    """
    if (row.br):
        row.br.extract()
    if row.a:
        row.a.extract()
    if row.sup:
        row.sup.extract()

    colunm_name = ' '.join(row.contents)

    # Filter the digit and empty names
    if not(colunm_name.strip().isdigit()):
        colunm_name = colunm_name.strip()
        return colunm_name
```

Result

EDA and Interactive visual analytics

To query and analyse dataset:
- IBM Watson studio
- Sql Db2

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

```
%sql select date, landing__outcome from SPACEXTBL where DATE between '2010-06-04' and '2017-03-20' order by DATE desc;
```

```
 * ibm_db_sa://kpz71093:***@8e359033-a1c9-4643-82ef-8ac06f5107eb.bs2io90108kqb1od8lcg.databases.appdomain.cloud:30120/blud
b
Done.
```

DATE	landing__outcome
2017-03-16	No attempt
2017-02-19	Success (ground pad)
2017-01-14	Success (drone ship)
2016-08-14	Success (drone ship)
2016-07-18	Success (ground pad)
2016-06-15	Failure (drone ship)
2016-05-27	Success (drone ship)
2016-05-06	Success (drone ship)
2016-04-08	Success (drone ship)
2016-03-04	Failure (drone ship)
2016-01-17	Failure (drone ship)
2015-12-22	Success (ground pad)
2015-06-28	Precluded (drone ship)
2015-04-27	No attempt
2015-04-14	Failure (drone ship)

List the total number of successful and failure mission outcomes

```
%sql select mission_outcome, count(mission_outcome) as total_missionoutcomes from SPACEXTBL group by mission_outcome;
```

```
 * ibm_db_sa://kpz71093:***@8e359033-a1c9-4643-82ef-8ac06f5107eb.bs2io90108kqb1od8lcg.databases.appdomain.cloud:30120/blud
b
Done.
```

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

```
CREATE VIEW SpaceX AS
SELECT *
FROM SPACEXTBL;

select MONTH(date), mission_outcome, booster_version, launch_site
from SPACEXTBL
where EXTRACT(YEAR from date)= '2015'
order by mission_outcome
limit 1;
```

select MONTH(date), mission_outcome, booste... Run time: **0.012 s**

Result set 1

Find

1	MISSION_OUTCOME	BOOSTER_VERSION	LAUNCH_SITE
6	Failure (in flight)	F9 v1.1 B1018	CCAFS LC-40

Predictive analysis

For prediction and classification:

Logistic regression

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`.

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

```
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']})
```

Support vector machines (SVM)

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`.

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

```
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, 3.162277
66e+01,
       1.00000000e+03]),
                         'gamma': array([1.00000000e-03, 3.16227766e-02, 1.00000000e+00, 3.16
227766e+01,
       1.00000000e+03]),
                         'kernel': ('linear', 'rbf', 'poly', 'rbf', 'sigmoid')})
```

K-Nearest Neighbours (KNN)

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`.

```
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()
```

```
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("parameters",knn_cv.best_params_)
print("accuracy:",knn_cv.best_score_)
```

```
parameters {'algorithm': 'auto', 'n_neighbors': 10, 'p': 1}
accuracy: 0.8482142857142858
```

Decision tree

Create a decision tree classifier object then create a `GridSearchCV` object `tree_cv` with cv = 10. Fit the object to find the best parameters from the dictionary `parameters`.

```
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()
```

```
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']})
```

Create a NumPy array from the column `Class` in `data`, by applying the method `to_numpy()` then assign it to the variable `Y`,make sure the output is a Pandas series (only one bracket df['name of column']).

```
Y = data["Class"].to_numpy()
print(Y)
```

```
[0 0 0 0 0 0 1 1 0 0 0 0 1 0 0 0 1 0 0 1 1 1 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1
 1 1 1 1 1 1 1 0 0 0 1 1 0 0 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 0 1 1 1 1 0 1
 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
```

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. The training data and test data should be assigned to the following labels.

`X_train, X_test, Y_train, Y_test`

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

```
Y_test.shape
```

```
(18,)
```

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

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

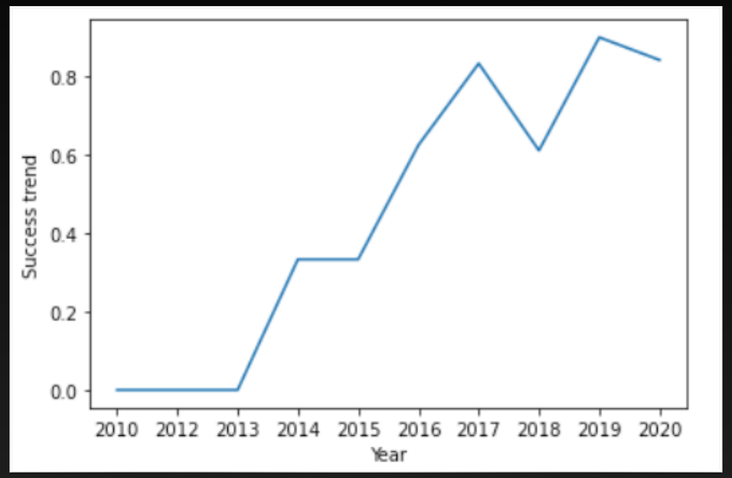
```
X
```

	FlightNumber	PayloadMass	Flights	Block	ReusedCount	Orbit_ES-L1	Orbit_GEO	Orbit_GTO	Orbit_HEO	Orbit_ISS	...
0	1.0	6104.959412	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	...
1	2.0	525.000000	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	...
2	3.0	677.000000	1.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	...
3	4.0	500.000000	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	...
4	5.0	3170.000000	1.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	...

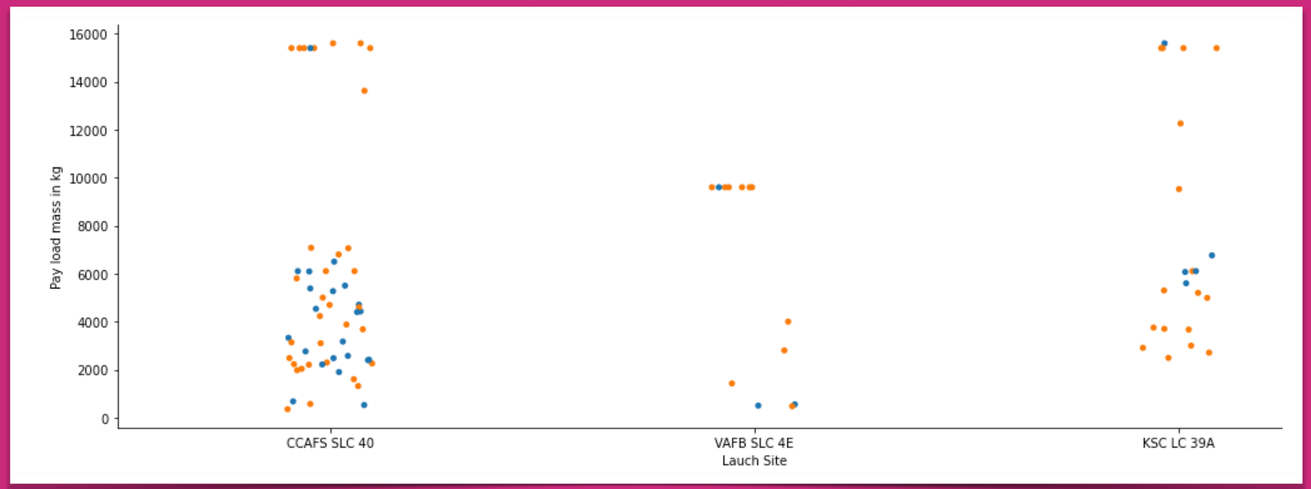
EDA with Visualisation

Seaborn used a python data visualisation library based on matplotlib

```
# Plot a line chart with x axis to be the extracted year and y axis to be the success rate
plt.plot(yearly_average["Year"],yearly_average["Class"])
plt.xlabel("Year")
plt.ylabel("Success trend")
plt.show()
```



Visualise the relationship between payload and launch site



EDA with SQL

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

```
%sql select unique(launch_site) from SPACEXTBL;
```

```
 * ibm_db_sa://kpz71093:***@8e359033-a1c9-4643-82ef-8ac
loud:30120/bludb
Done.
```

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

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

```
%sql select date, booster_version
from SPACEXTBL
where landing__outcome = 'Success (drone ship)' and payload_mass__kg_ between 4000 and 6000;
```

```
 * ibm_db_sa://kpz71093:***@8e359033-a1c9-4643-82ef-8ac06f5107eb.bs2io90l08kqb1od8lcg.databases.appdo
loud:30120/bludb
Done.
```

DATE	booster_version
2016-05-06	F9 FT B1022
2016-08-14	F9 FT B1026
2017-03-30	F9 FT B1021.2
2017-10-11	F9 FT B1031.2

List the failed landing_outcomes in drone ship, their booster versions, and launch site names for in year 2015

```
%sql select MONTH(date), mission_outcome, booster_version, launch_site
from SPACEXTBL where EXTRACT(YEAR from date)= '2015';
```

```
 * ibm_db_sa://kpz71093:***@8e359033-a1c9-4643-82ef-8ac06f5107eb.bs2io90l08kqb1od8lcg.dat
loud:30120/bludb
Done.
```

1	mission_outcome	booster_version	launch_site
1	Success	F9 v1.1 B1012	CCAFS LC-40
2	Success	F9 v1.1 B1013	CCAFS LC-40
3	Success	F9 v1.1 B1014	CCAFS LC-40
4	Success	F9 v1.1 B1015	CCAFS LC-40
4	Success	F9 v1.1 B1016	CCAFS LC-40
6	Failure (in flight)	F9 v1.1 B1018	CCAFS LC-40
12	Success	F9 FT B1019	CCAFS LC-40

List the total number of successful and failure mission outcomes

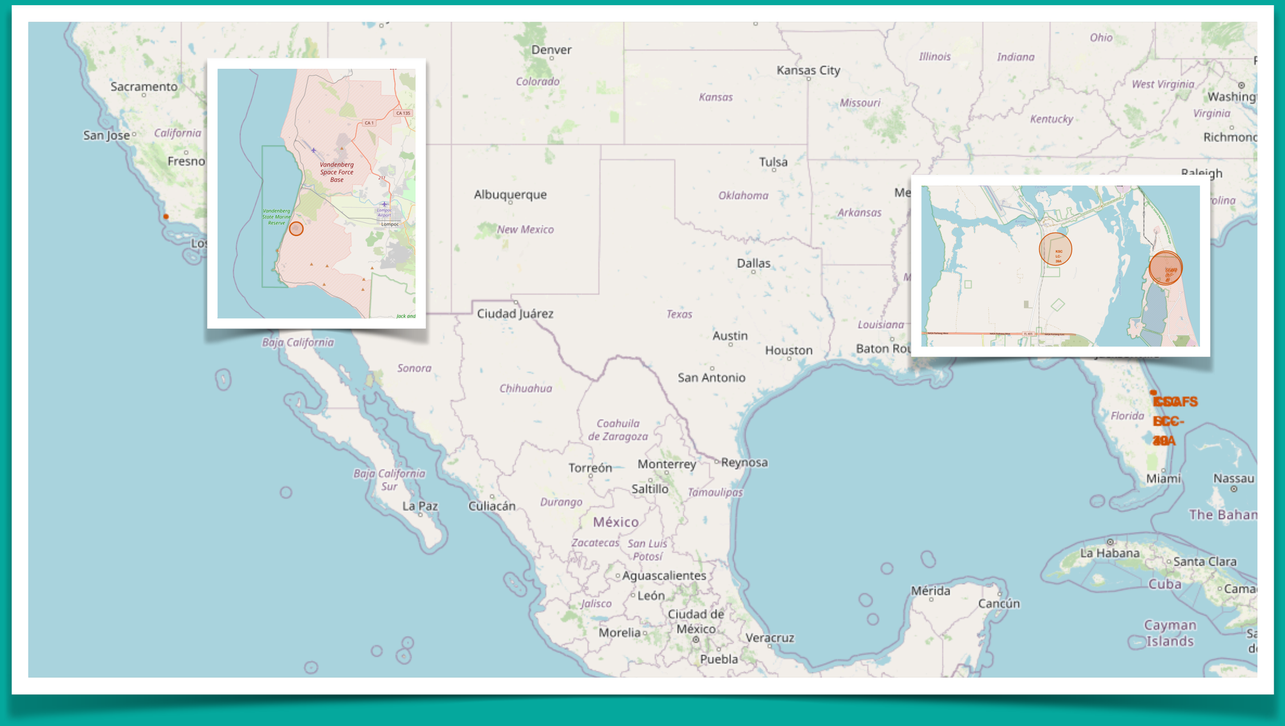
```
%sql select mission_outcome, count(mission_outcome) as total_missionoutcomes
from SPACEXTBL group by mission_outcome;
```

```
 * ibm_db_sa://kpz71093:***@8e359033-a1c9-4643-82ef-8ac06f5107eb.bs2io90l08k
loud:30120/bludb
Done.
```

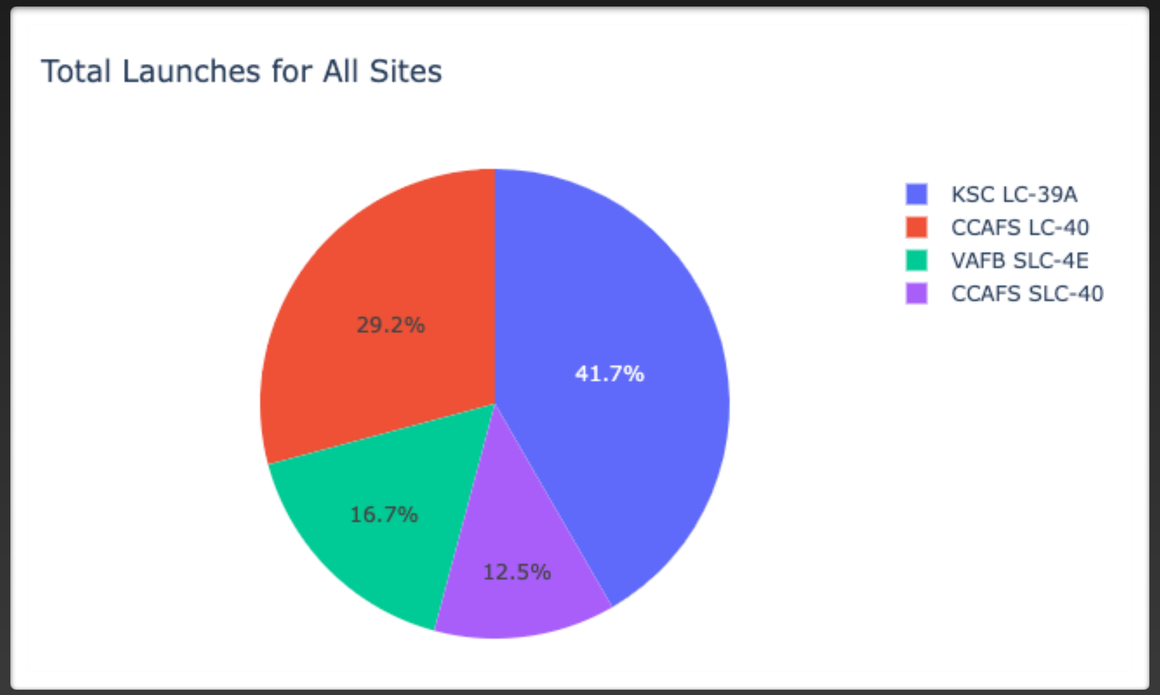
mission_outcome	total_missionoutcomes
Failure (in flight)	1
Success	99
Success (payload status unclear)	1

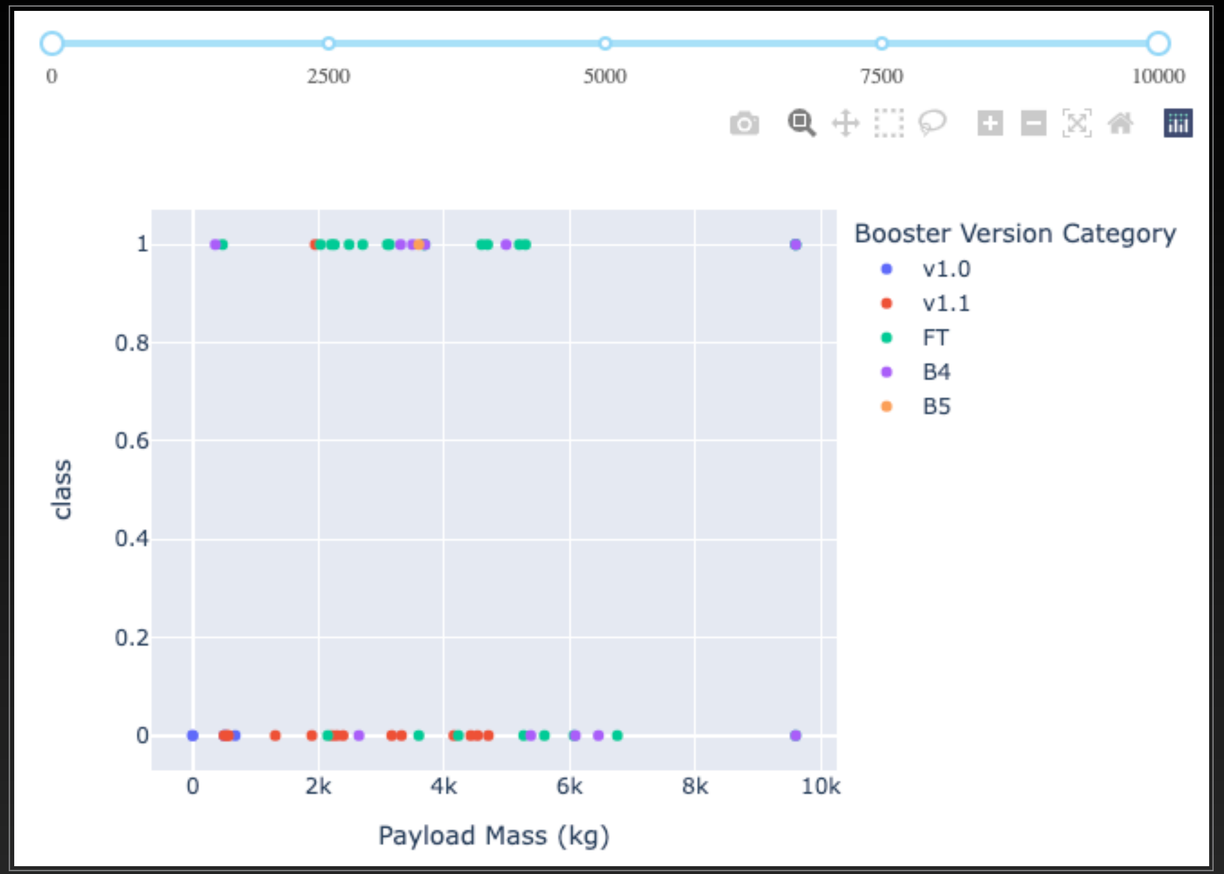
Interactive map with Folium

Mark all launch sites on map
Mark success/failed launches
Calculating launching site to its proximity



Ploty Dash dashboard





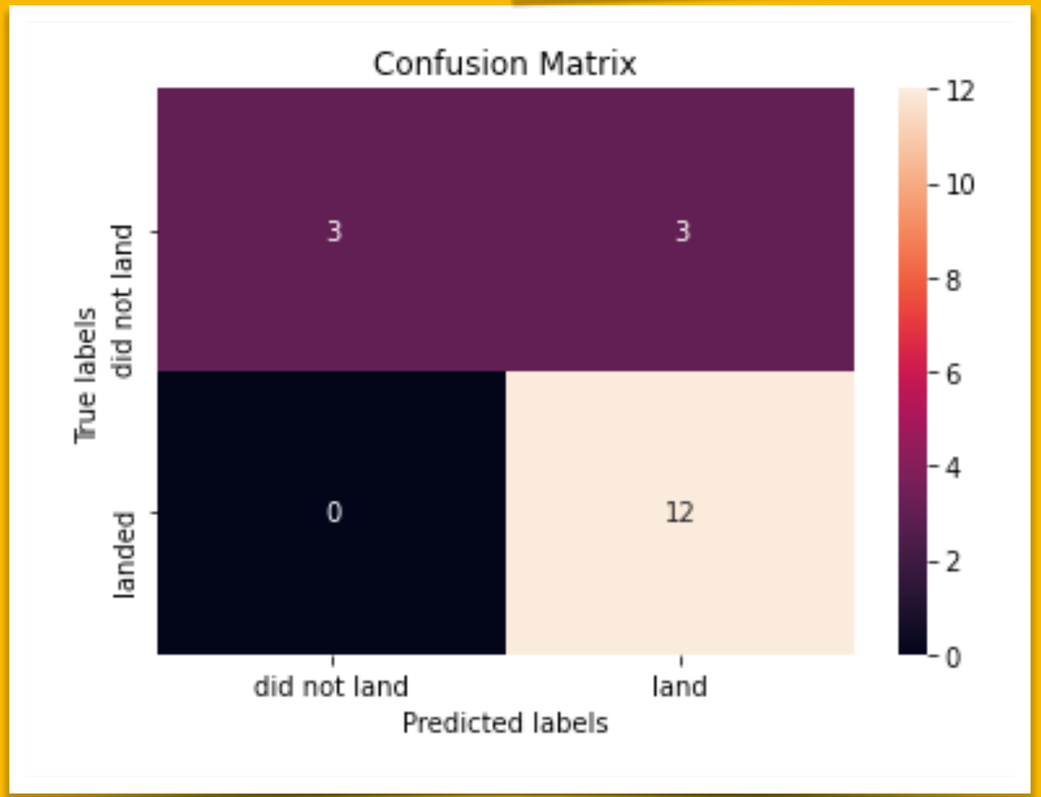
Predictive analysis - Classification

Find the method performs best:

```
scores = [lr_score,svm_score,tree_score,knn_score]
print(scores)
print(scores.index(max(scores)))
```

```
[0.8333333333333334, 0.8333333333333334, 0.6666666666666666, 0.8333333333333334]
```

Confusion matrix find the the method perform best.



Conclusion

Data analysis using exploratory statistics and visualisation technique with the application of machine learning technique enhance the available knowledge about the dataset and this helps to predict proximity of launch site and success of launch and.