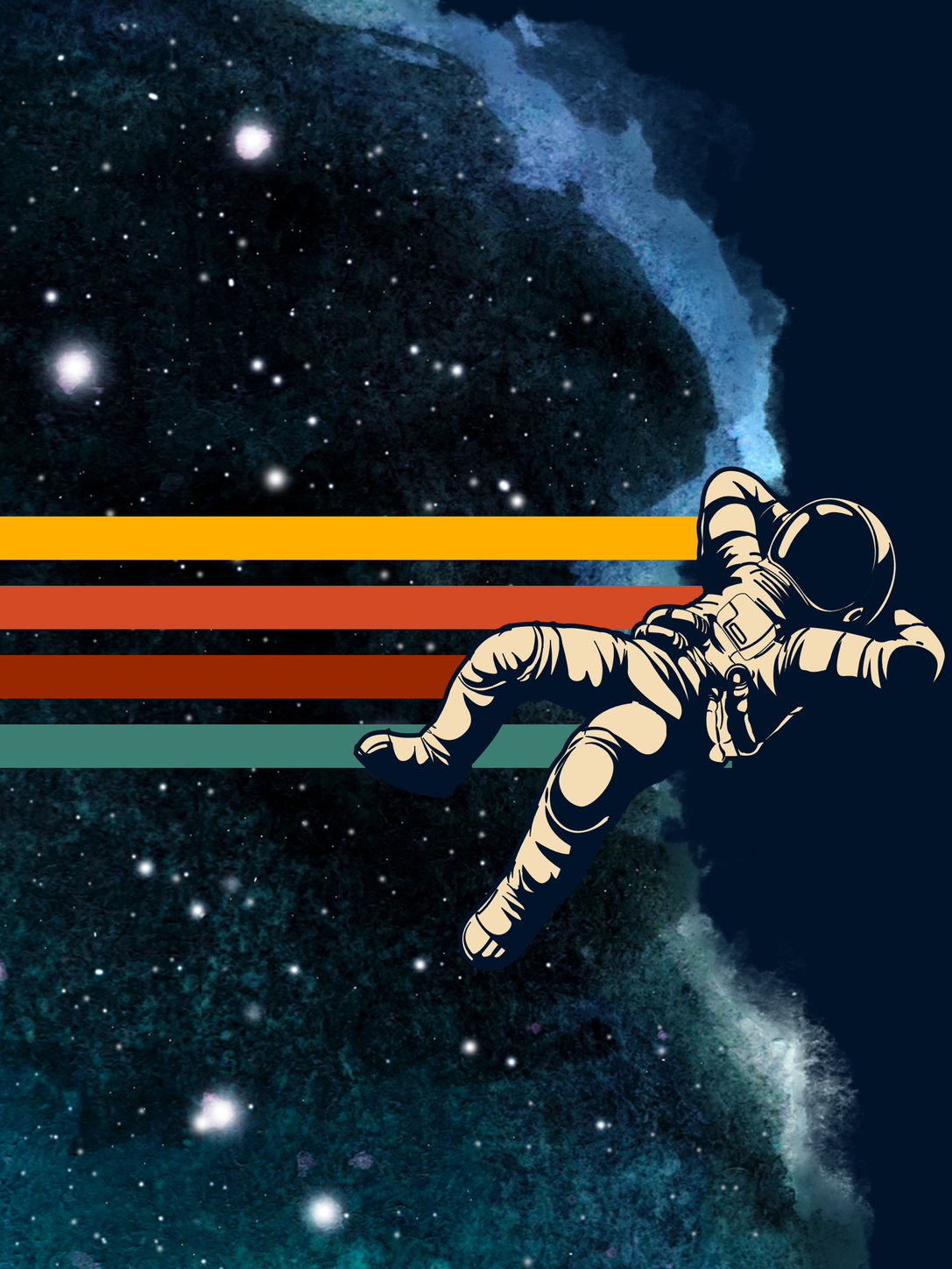
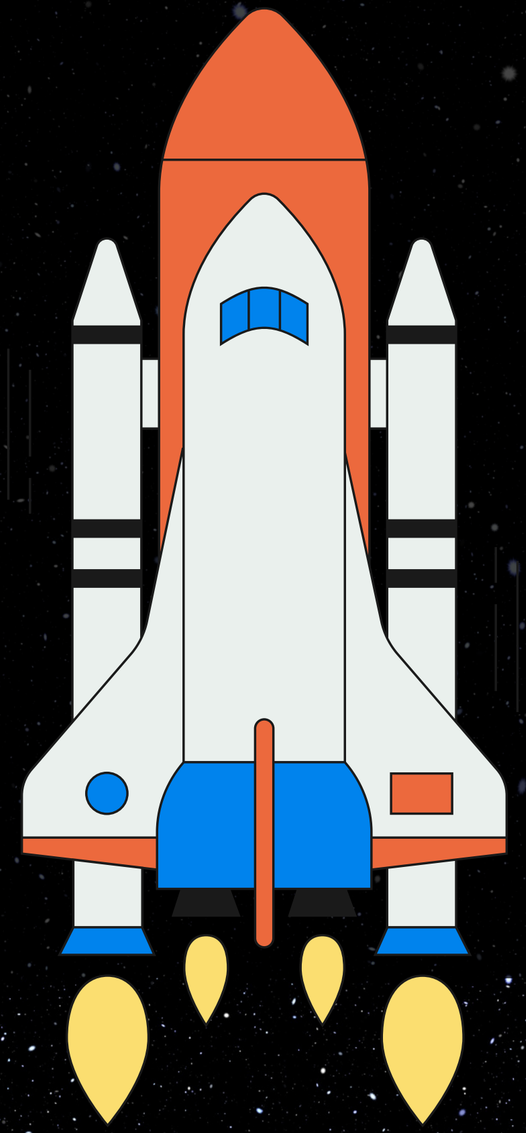


APPLIED DATA SCIENCE CAPSTONE PROJECT

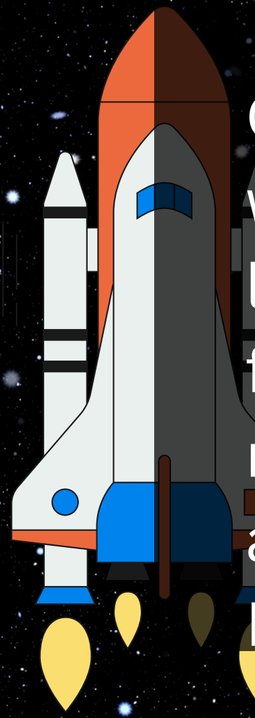
by Nhlanhla Sanele Dludla

Outline

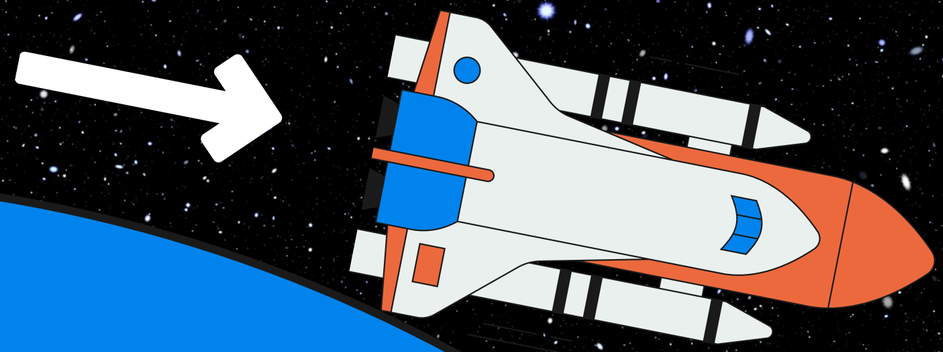
- Executive summary
- Introduction
- Methodology
- Results
- Conclusion



Executive summary



Conventional space exploration is expensive. To cut costs associated with space travel, SpaceX reuses their first stage. In the bid to cut launch costs, in this capstone project we aim to predict whether the first stage of Falcon 9 will land successfully or not on landing pads for reuse. To achieve this, data will be collected cleaned and formatted. In addition exploratory and predictive analyses will be performed. Results will be presented in graph format.

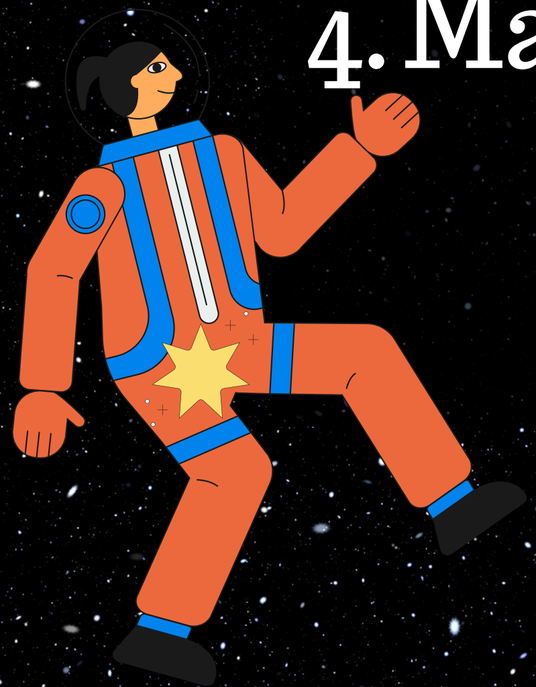


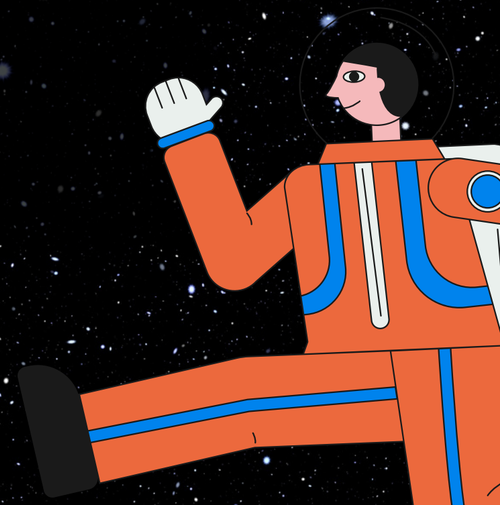
Introduction

Conventional space exploration is expensive. The typical costs to launch space crafts costs up to $165 Millon. To cut costs associated with space travel, SpaceX reuses their first stage. In the bid to cut launch costs, in this capstone project we aim to predict whether the first stage of Falcon 9 will land successfully or not on landing pads for reuse. To achieve this, data will be collected cleaned and formatted. In addition, exploratory and predictive analyses will be performed. Results will be presented in graph format.

Methodology

1. Data collection and processing.
2. Data analyses
3. Data visualization
4. Machine learning and prediction





Data collection: API + Data wrangling

1. “https://api.spacexdata.com/v4/launches/past” was used to access SpaceX data.
2. Collected data was decoded into a JSON file and IDs were assigned to specific columns.
3. The dataframe was then filtered to include only Falcon 9 launches.
4. data_falcon9.isnull().sum() was used to identify missing values.
5. Using data_falcon9['PayloadMass'] = data_falcon9['PayloadMass'].replace(np.nan, payloadmass_mean) missing values were replaced with mean values.

FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins	Reused	Legs	LandingPad	Block	ReusedCount	Serial	Longitude	Latitude
1	2010-06-04	Falcon 9	NaN	LEO	CCSFS SLC 40	None None	1	False	False	False	None	1.0	0	B0003	-80.577366	28.561857
2	2012-05-22	Falcon 9	525.0	LEO	CCSFS SLC 40	None None	1	False	False	False	None	1.0	0	B0005	-80.577366	28.561857
3	2013-03-01	Falcon 9	677.0	ISS	CCSFS SLC 40	None None	1	False	False	False	None	1.0	0	B0007	-80.577366	28.561857
4	2013-09-29	Falcon 9	500.0	PO	VAFB SLC 4E	False Ocean	1	False	False	False	None	1.0	0	B1003	-120.610829	34.632093
5	2013-12-03	Falcon 9	3170.0	GTO	CCSFS SLC 40	None None	1	False	False	False	None	1.0	0	B1004	-80.577366	28.561857

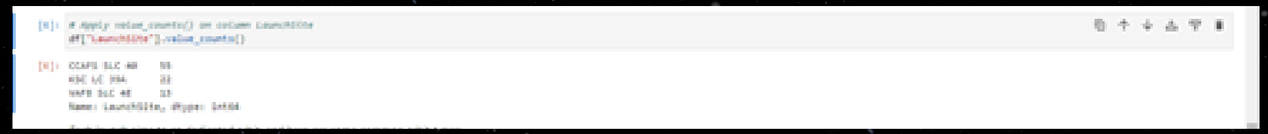
Data collection: Webscrapping

1. Using BeautifulSoup Falcon 9 launch table records were extracted from Wikipedia.
2. HTML tables were parsed and a dataframe was created.

	Flight No.	Launch site	Payload	Payload mass	Orbit	Customer	Launch outcome	Version Booster	Booster landing	Date	Time
0	1	CCAFS	Dragon Spacecraft Qualification Unit	0	LEO	SpaceX	Success\n	F9 v1.0B0003.1	Failure	4 June 2010	18:45
1	2	CCAFS	Dragon	0	LEO	NASA	Success	F9 v1.0B0004.1	Failure	8 December 2010	15:43
2	3	CCAFS	Dragon	525 kg	LEO	NASA	Success	F9 v1.0B0005.1	No attempt\n	22 May 2012	07:44
3	4	CCAFS	SpaceX CRS-1	4,700 kg	LEO	NASA	Success\n	F9 v1.0B0006.1	No attempt	8 October 2012	00:35
4	5	CCAFS	SpaceX CRS-2	4,877 kg	LEO	NASA	Success\n	F9 v1.0B0007.1	No attempt\n	1 March 2013	15:10
...	...	...	...	...	...	...	...	...	...	...	...
116	117	CCSFS	Starlink	15,600 kg	LEO	SpaceX	Success\n	F9 B5B1051.10	Success	9 May 2021	06:42
117	118	KSC	Starlink	~14,000 kg	LEO	SpaceX	Success\n	F9 B5B1058.8	Success	15 May 2021	22:56
118	119	CCSFS	Starlink	15,600 kg	LEO	SpaceX	Success\n	F9 B5B1063.2	Success	26 May 2021	18:59
119	120	KSC	SpaceX CRS-22	3,328 kg	LEO	NASA	Success\n	F9 B5B1067.1	Success	3 June 2021	17:29
120	121	CCSFS	SXM-8	7,000 kg	GTO	Sirius XM	Success\n	F9 B5	Success	6 June 2021	04:26

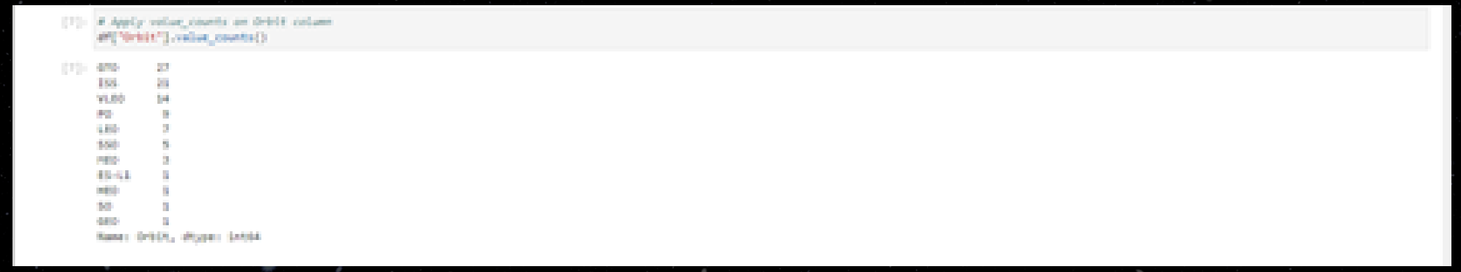
Data analyses: Exploratory

1. Using Pandas and Numpy, A wrangled dataset was loaded and information on the number of launches, orbit occurances, and missions outcomes was collected.



```
[6]: # Apply value_counts() on column LaunchSite
     df['LaunchSite'].value_counts()

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



```
[7]: # Apply value_counts on Orbit column
     df['Orbit'].value_counts()

[7]: 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
```

```
[8]: df.head(5)

[8]:
```

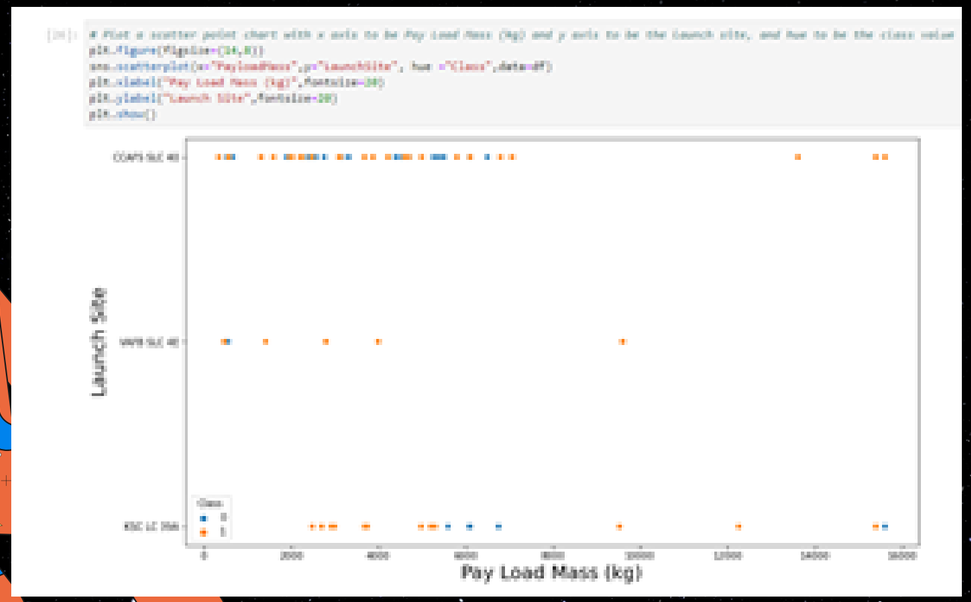
	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	GridFins	Reused	Legs	LandingPad	Block	ReusedCount	Serial	Longitude	Latitude	Class
0	1	2010-06-04	Falcon 9	6104.959412	LEO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B0003	-80.577366	28.561857	0
1	2	2012-05-22	Falcon 9	525.000000	LEO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B0005	-80.577366	28.561857	0
2	3	2013-03-01	Falcon 9	677.000000	ISS	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B0007	-80.577366	28.561857	0
3	4	2013-09-29	Falcon 9	500.000000	PO	VAFB SLC 4E	False Ocean	1	False	False	False	NaN	1.0	0	B1003	-120.610829	34.632093	0
4	5	2013-12-03	Falcon 9	3170.000000	GTO	CCAFS SLC 40	None None	1	False	False	False	NaN	1.0	0	B1004	-80.577366	28.561857	0

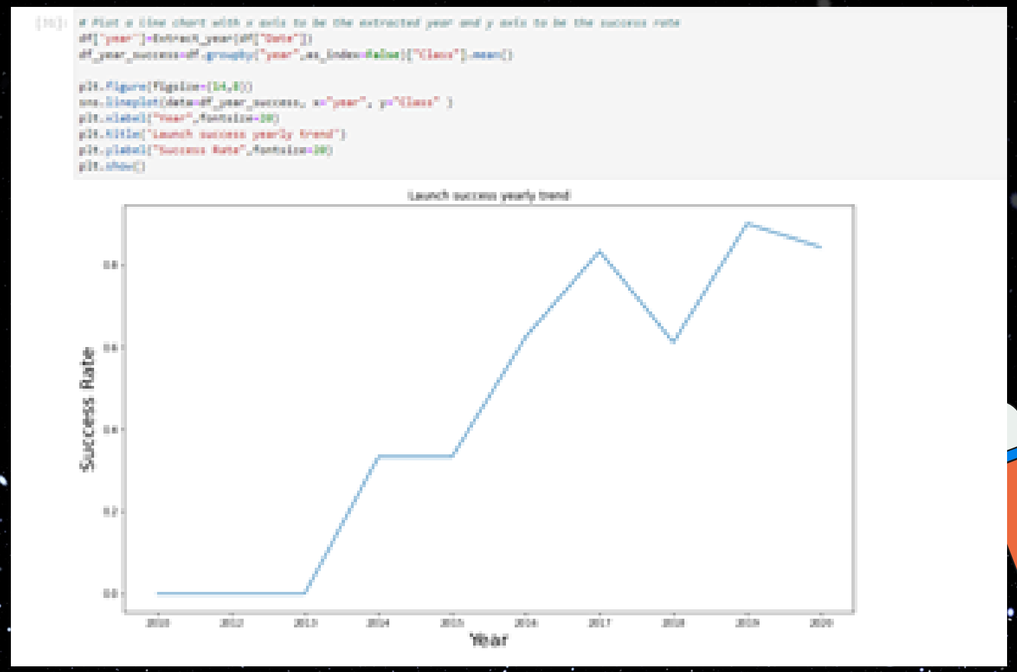
Data analyses: Exploratory

1. Using SQL, previously collected and wrangled data was queried to obtain information: 1) on the unique launch sites 2)launch sites that begin with CCA 3) total payload mass carried by boosters launched by NASA 4) average payload mass carried by booster F9 V1.1. 5) the date of the first successful landing on a ground pad 6) total number of successful and failure missions etc.

Data visualization: Exploratory

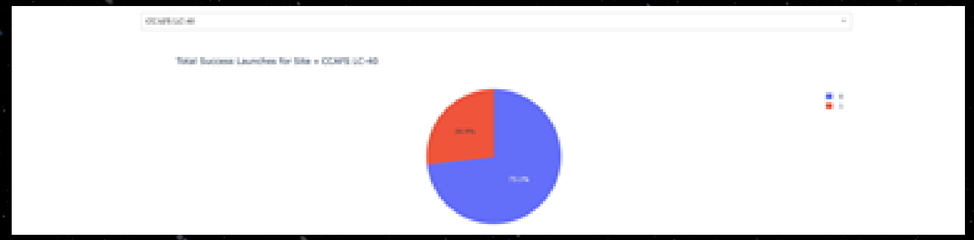
1. To understand relationships between variables of interest, using matplotlib and seaborn, collected and wrangled data was visualized using bar charts, line graphs and scatterplots.
2. To visualize and to mark launch sites where succssful/ unsuccessful launches took place Folium was used to create interactive maps.

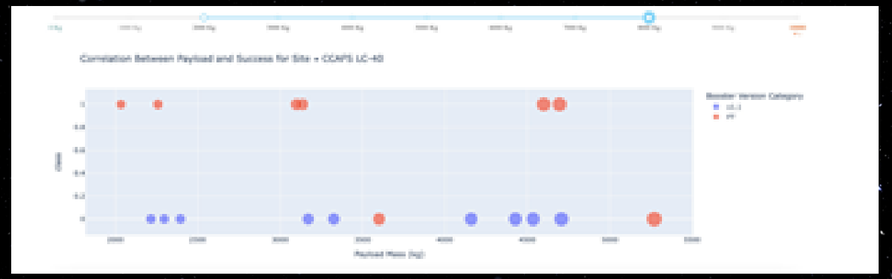




Data visualization: Dash

1. Using dash, an interactive pie and scatterplot chart was created to vizualize the total success launches from each site and the correlation between payloadmass and mission outcome.





Machine learning

1. Using functions from scikit-learn, machine learning models were built. This was achieved by 1) standardizing the data 2) splitting the data into a train and test test 3) fitting built models on the training set 4) evaluating model accuracy based on the confusion matrix.

TASK 3

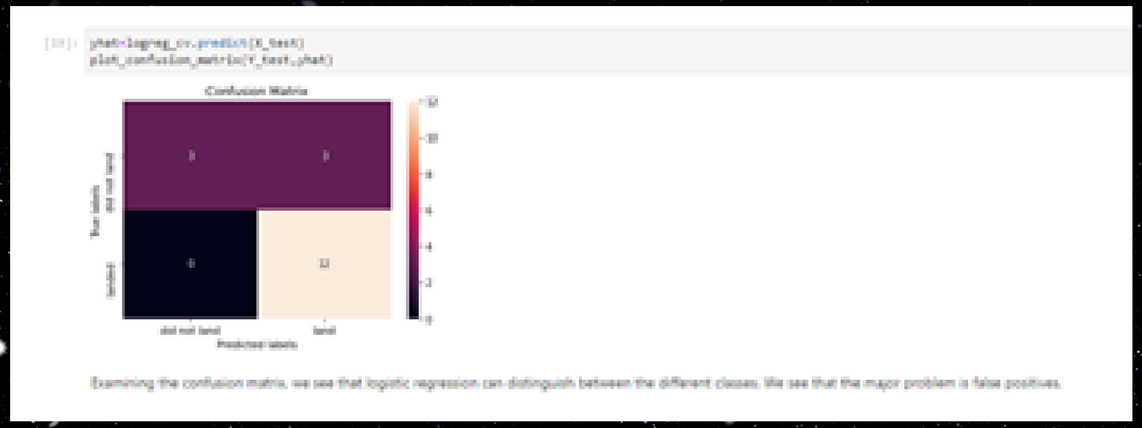
Use the function train_test_split to split the data X and Y into training and test data. Set the parameter test_size to 0.2 and random_state to 2. The training data and test data should be assigned to the following labels.

`X_train, X_test, Y_train, Y_test`

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

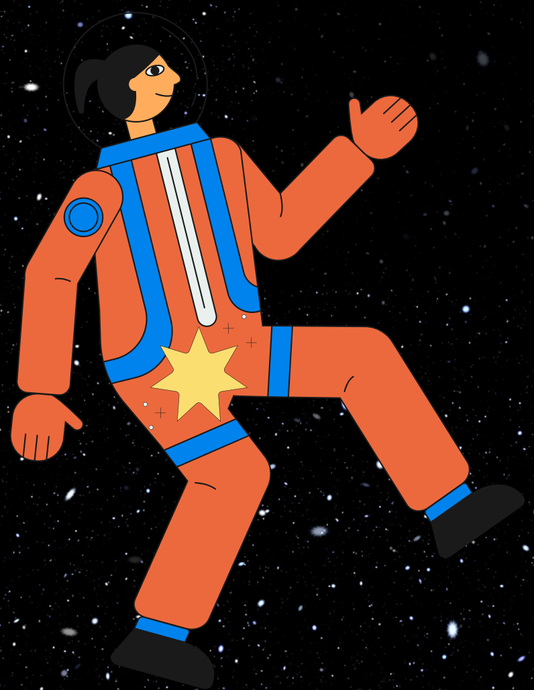
we can see we only have 18 test samples.

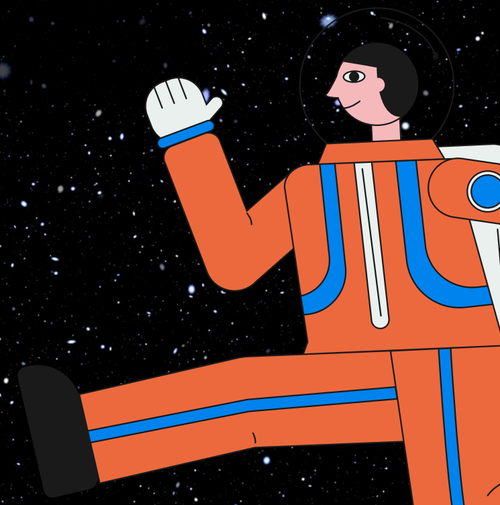
```
[11]: Y_test.shape
[11]: (18,)
```



```
[19]: yhat=logreg_cv.predict(X_test)
      plot_confusion_matrix(Y_test,yhat)
```

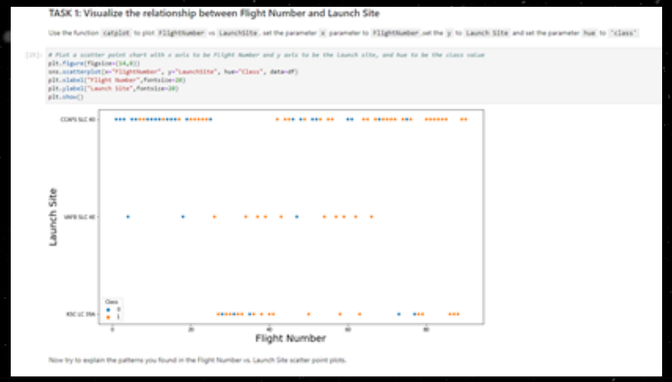

Examining the confusion matrix, we see that logistic regression can distinguish between the different classes. We see that the major problem is false positives.

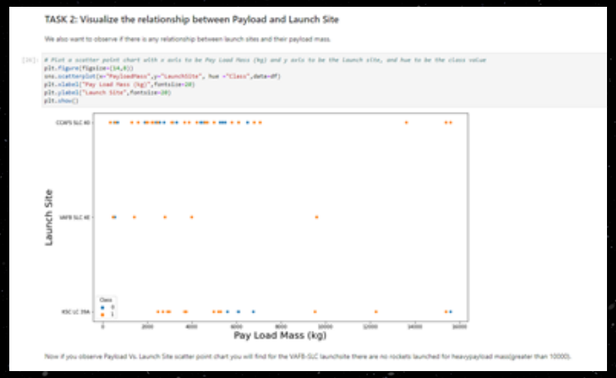


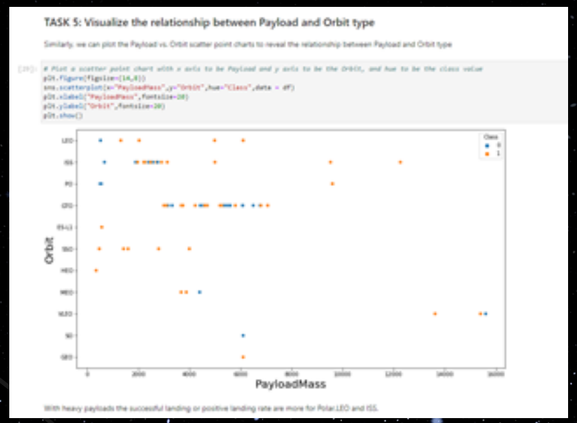


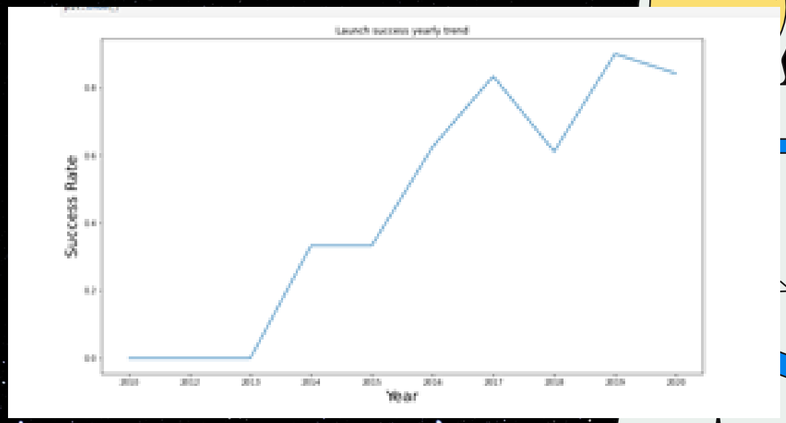
Results

EDA with visualization









Results

EDA with visualization

TASK 7: Create dummy variables to categorical columns

Use the function `get_dummies` and `features` dataframe to apply OneHotEncoder to the column `Orbits`, `LaunchSite`, `LandingPad`, and `Serial`. Assign the value to the variable `features_one_hot`, display the results using the method head. Your result dataframe must include all features including the encoded ones.

```
# HINT: Use get_dummies() function on the categorical columns
features_one_hot = features

features_one_hot = pd.concat([features_one_hot,pd.get_dummies(df['Orbit'])], axis=1)
features_one_hot.drop(['Orbit'], axis = 1,inplace=True)

features_one_hot = pd.concat([features_one_hot,pd.get_dummies(df['LaunchSite'])], axis=1)
features_one_hot.drop(['LaunchSite'], axis = 1,inplace=True)

features_one_hot = pd.concat([features_one_hot,pd.get_dummies(df['LandingPad'])], axis=1)
features_one_hot.drop(['LandingPad'], axis = 1,inplace=True)

features_one_hot = pd.concat([features_one_hot,pd.get_dummies(df['Serial'])], axis=1)
features_one_hot.drop(['Serial'], axis = 1,inplace=True)

features_one_hot.head()
```

	FlightNumber	PayloadMass	Flights	GridFins	Reused	Legs	Block	ReusedCount	ES-L1	GEO	...	B1048	B1049	B1050	B1051	B1054	B1056	B1058	B1059	B1060	B1062
0	1	6104.959412	1	False	False	False	1.0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
1	2	525.000000	1	False	False	False	1.0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
2	3	677.000000	1	False	False	False	1.0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
3	4	500.000000	1	False	False	False	1.0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
4	5	3170.000000	1	False	False	False	1.0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

5 rows × 80 columns

Results

EDA with SQL

Task 1

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

```
%sql SELECT DISTINCT LAUNCH_SITE as "Launch_Sites" FROM SPACEXTBL;
```

```
 * ibm_db_sa://dgy37633:***@0853d391-7217-4624-b827-8baa776ffad1.c3n41cmd0nqnrk39u98g.databases.appdomain.cloud:30875/bludb
Done.
```

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

Task 2

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

```
%sql SELECT * FROM SPACEXTBL WHERE LAUNCH_SITE LIKE 'CCA%' LIMIT 5;
```

```
 * ibm_db_sa://dgy37633:***@0853d391-7217-4624-b827-8baa776ffad1.c3n41cmd0nqnrk39u98g.databases.appdomain.cloud:30875/bludb
Done.
```

DATE	time_utc_	booster_version	launch_site	payload	payload_mass_kg_	orbit	customer	mission_outcome	landing_outcome
2010-06-04	18:45:00	F9 v1.0 B0003	CCAFS LC-40	Dragon Spacecraft Qualification Unit	0	LEO	SpaceX	Success	Failure (parachute)
2010-12-08	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)
2012-05-22	07:44:00	F9 v1.0 B0005	CCAFS LC-40	Dragon demo flight C2	525	LEO (ISS)	NASA (COTS)	Success	No attempt
2012-10-08	00:35:00	F9 v1.0 B0006	CCAFS LC-40	SpaceX CRS-1	500	LEO (ISS)	NASA (CRS)	Success	No attempt
2013-03-01	15:10:00	F9 v1.0 B0007	CCAFS LC-40	SpaceX CRS-2	677	LEO (ISS)	NASA (CRS)	Success	No attempt

Task 3

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

```
%sql SELECT SUM(PAYLOAD_MASS__KG_) AS "Total payload mass by NASA (CRS)" FROM SPACEXTBL WHERE CUSTOMER = 'NASA (CRS)';
```

```
 * ibm_db_sa://dgy37633:***@0853d391-7217-4624-b827-8baa776ffad1.c3n41cmd0nqnrk39u98g.databases.appdomain.cloud:30875/bludb
Done.
```

Total payload mass by NASA (CRS)
45596

Task 4

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

```
%sql SELECT AVG(PAYLOAD_MASS__KG_) AS "Average payload mass by Booster Version F9 v1.1" FROM SPACEXTBL WHERE BOOSTER_VERSION = 'F9 v1.1';
```

```
 * ibm_db_sa://dgy37633:***@0853d391-7217-4624-b827-8baa776ffad1.c3n41cmd0nqnrk39u98g.databases.appdomain.cloud:30875/bludb
Done.
```

Average payload mass by Booster Version F9 v1.1
2928

Task 5

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

Hint: Use min function

```
%sql SELECT MIN(DATE) AS "Date of first successful landing outcome in ground pad" FROM SPACEXTBL WHERE LANDING__OUTCOME = 'Success (ground pad)';
```

```
 * ibm_db_sa://dgy37633:***@0853d391-7217-4624-b827-8baa776ffad1.c3n41cmd0nqnrk39u98g.databases.appdomain.cloud:30875/bludb
Done.
```

Date of first successful landing outcome in ground pad
2015-12-22

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

```
%sql SELECT BOOSTER_VERSION FROM SPACEXTBL WHERE LANDING__OUTCOME = 'Success (drone ship)' AND PAYLOAD_MASS__KG_ BETWEEN 4000 AND 6000;
```

```
 * ibm_db_sa://dgy37633:***@0853d391-7217-4624-b827-8baa776ffad1.c3n41cmd0nqnrk39u98g.databases.appdomain.cloud:30875/bludb
Done.
```

booster_version
F9 FT B1022
F9 FT B1026
F9 FT B1021.2
F9 FT B1031.2

Task 7

List the total number of successful and failure mission outcomes

```
%sql SELECT number_of_success_outcomes, number_of_failure_outcomes FROM (SELECT COUNT(*) AS number_of_success_outcomes FROM SPACEXTBL WHERE MISSION_OUTCOME LIKE 'Success%') success_table, (SELECT COUNT(*)
```

```
 * ibm_db_sa://dgy37633:***@0853d391-7217-4624-b827-8baa776ffad1.c3n41cmd0nqnrk39u98g.databases.appdomain.cloud:30875/bludb
Done.
```

number_of_success_outcomes	number_of_failure_outcomes
100	1

Task 8

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

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

```
 * ibm_db_sa://dgy37633:***@0853d391-7217-4624-b827-8baa776ffad1.c3n41cmd0nqnrk39u98g.databases.appdomain.cloud:30875/bludb
Done.
```

booster_version
F9 B5 B1048.4
F9 B5 B1048.5

Results

EDA with SQL

Task 9

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

```
%sql SELECT DATE, BOOSTER_VERSION, LAUNCH_SITE FROM SPACEXTBL WHERE year(DATE) = '2015' AND LANDING__OUTCOME = 'Failure (drone ship)';
```

Done.

DATE	booster_version	launch_site
2015-01-10	F9 v1.1 B1012	CCAFS LC-40
2015-04-14	F9 v1.1 B1015	CCAFS LC-40

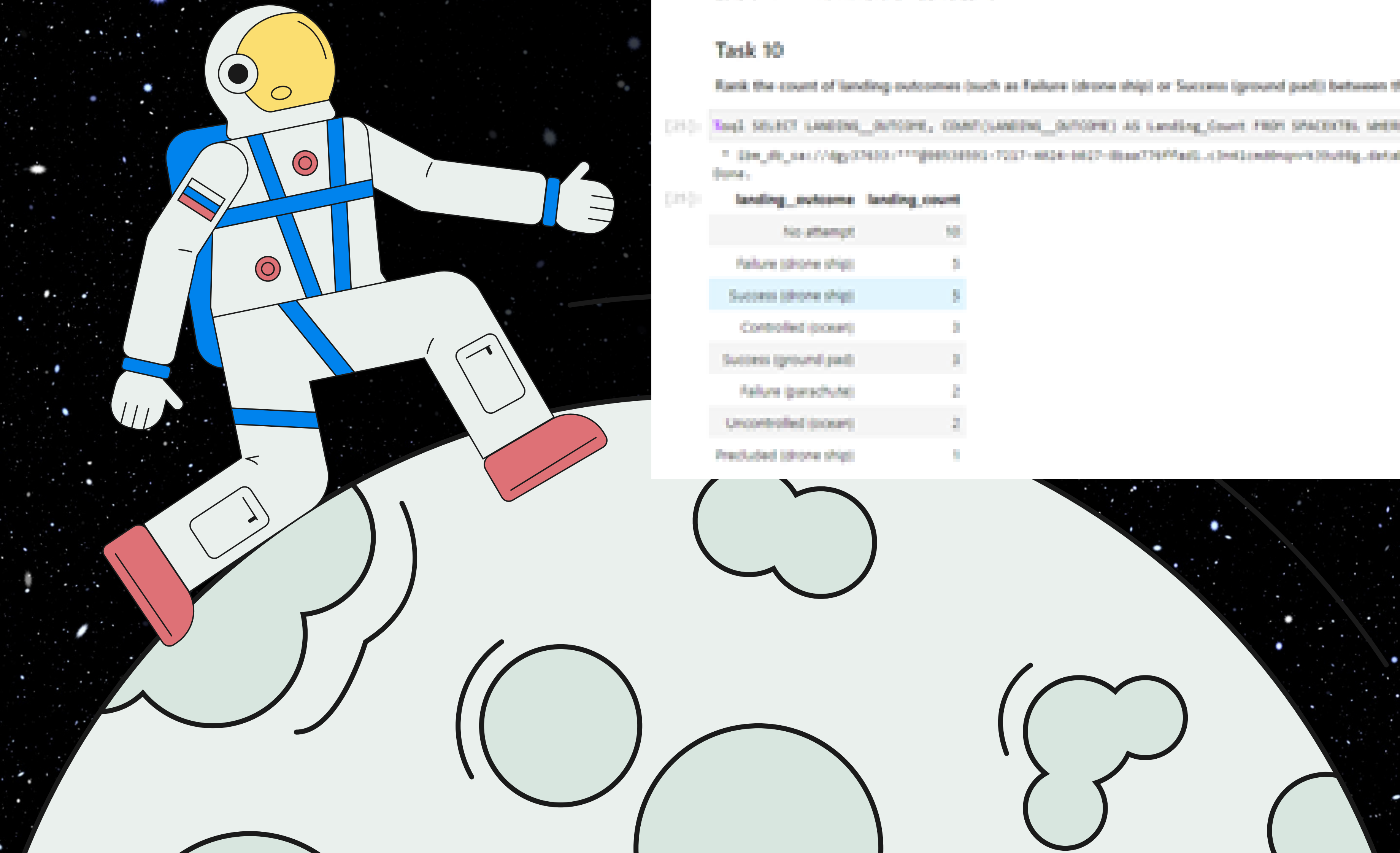
Task 10

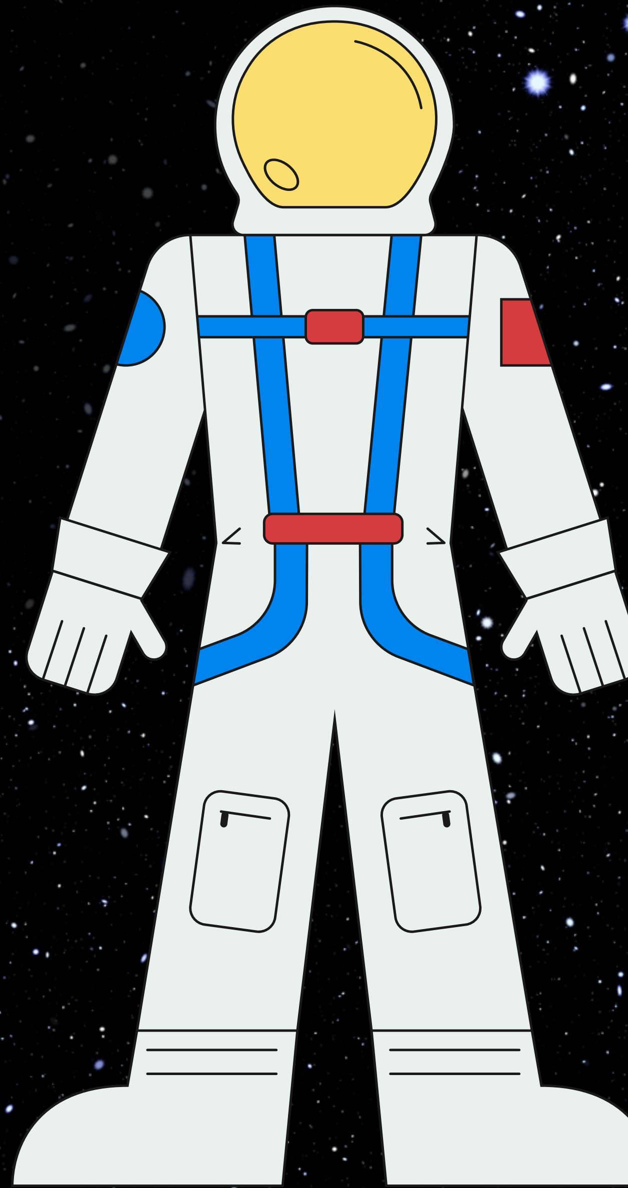
Rank the count of landing outcomes (such as Failure (drone ship) or Success (ground pad)) between the date 2010-06-04 and 2017-03-20, in descending order

```
%sql SELECT LANDING__OUTCOME, COUNT(LANDING__OUTCOME) AS Landing_Count FROM SPACEXTBL WHERE DATE BETWEEN '2010-06-04' AND '2017-03-20' GROUP BY LANDING__OUTCOME ORDER BY COUNT(LANDING__OUTCOME) DESC;
```

Done.

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





Results

Interactive maps with folium

Task 1: Mark all launch sites on a map

First, let's try to add each site's location on a map using site's latitude and longitude coordinates

The following dataset with the name `spacex_launch_geo.csv` is an augmented dataset with latitude and longitude added for each site.

```
[28]: # Download and read the `spacex_launch_geo.csv`
spacex_csv_file = wget.download("https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0321EN-SkillsNetwork/datasets/spacex_launch_geo.csv")
spacex_df=pd.read_csv(spacex_csv_file)
```

Now, you can take a look at what are the coordinates for each site.

```
[29]: # Select relevant sub-columns: `Launch Site`, `Lat(Latitude)`, `Long(Longitude)`, `Class`
spacex_df = spacex_df[['Launch Site', 'Lat', 'Long', 'Class']]
launch_sites_df = spacex_df.groupby(['Launch Site'], as_index=False).first()
launch_sites_df = launch_sites_df[['Launch Site', 'Lat', 'Long']]
launch_sites_df
```

[29]:

	Launch Site	Lat	Long
0	CCAFS LC-40	28.562302	-80.577356
1	CCAFS SLC-40	28.563197	-80.576820
2	KSC LC-39A	28.573255	-80.646895
3	VAFB SLC-4E	34.632834	-120.610746

Results

Interactive maps with folium

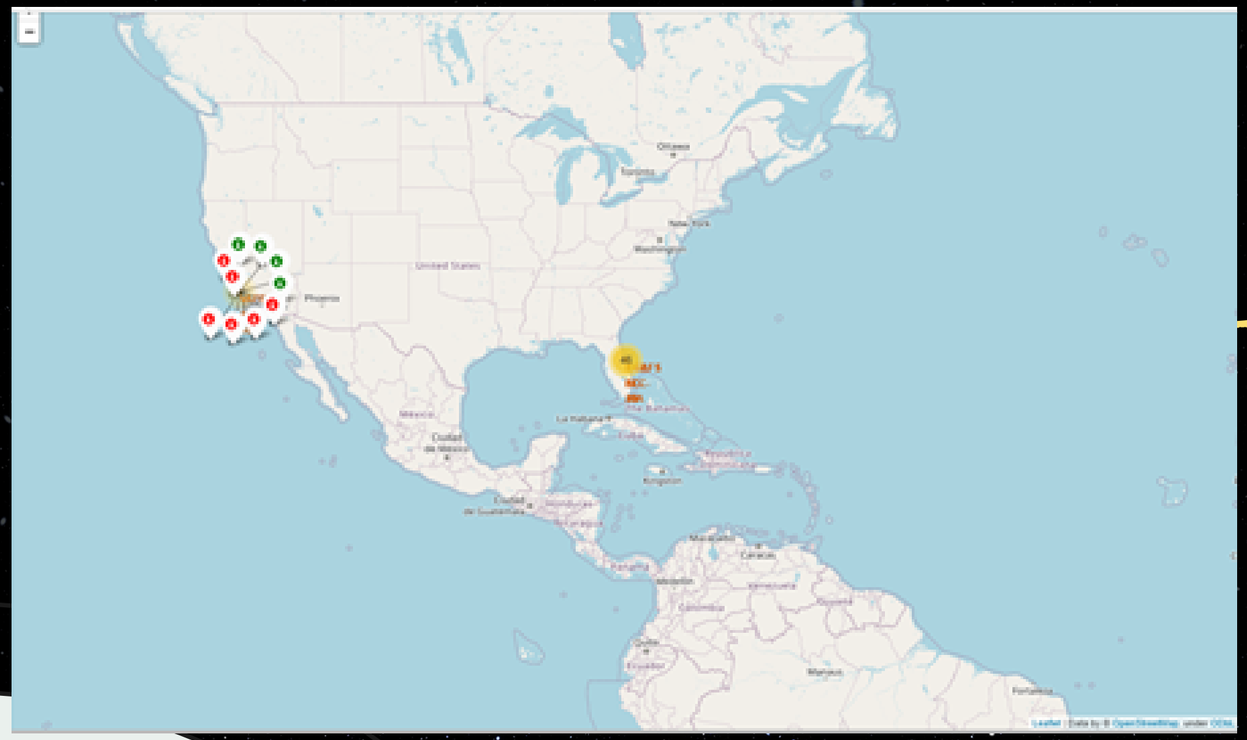

```
    else:
        return "red"

spacex_df["marker_color"] = spacex_df["class"].apply(assign_marker_color)
spacex_df.tail(10)
```

	Launch Site	Lat	Long	class	marker_color
46	KSC LC-39A	28.573255	-80.646895	1	green
47	KSC LC-39A	28.573255	-80.646895	1	green
48	KSC LC-39A	28.573255	-80.646895	1	green
49	CCAFS SLC-40	28.563197	-80.576820	1	green
50	CCAFS SLC-40	28.563197	-80.576820	1	green
51	CCAFS SLC-40	28.563197	-80.576820	0	red
52	CCAFS SLC-40	28.563197	-80.576820	0	red
53	CCAFS SLC-40	28.563197	-80.576820	0	red
54	CCAFS SLC-40	28.563197	-80.576820	1	green
55	CCAFS SLC-40	28.563197	-80.576820	0	red

Results

Interactive maps with folium



Results

Dashboard

Results

Predictive analyses

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`.

```
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, scoring='accuracy', cv=10)
logreg_cv = logreg_cv.fit(X_train, Y_train)
```

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_`.

```
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
```

TASK 5

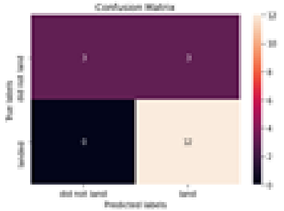
Calculate the accuracy on the test data using the method `score`:

```
lr_accuracy = logreg_cv.score(X_test, Y_test)
lr_accuracy
```

```
0.8333333333333334
```

Lets look at the confusion matrix:

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



Examining the confusion matrix, we see that logistic regression can distinguish between the different classes. We see that the major problem is false positives.

Results

Predictive analyses

TASK 6

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, scoring='accuracy', cv=10)
svm_cv = svm_cv.fit(X_train, Y_train)
```

```
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
```

TASK 7

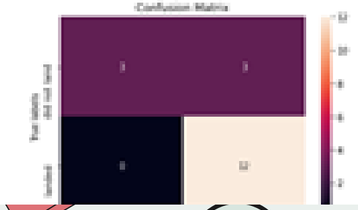
Calculate the accuracy on the test data using the method `score`:

```
svm_accuracy = svm_cv.score(X_test, Y_test)
svm_accuracy
```

```
0.8333333333333334
```

We can plot the confusion matrix

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



Results

Predictive analyses

TASK 8

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`.

```
[28]: 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()
```

```
[29]: tree_cv = GridSearchCV(tree, parameters, scoring='accuracy', cv=10)
      tree_cv = tree_cv.fit(X_train, Y_train)
```

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

```
tuned hpyerparameters :(best parameters)  {'criterion': 'gini', 'max_depth': 4, 'max_features': 'sqrt', 'min_samples_leaf': 1, 'min_samples_split': 2, 'splitter': 'best'}
accuracy : 0.8892857142857142
```

TASK 9

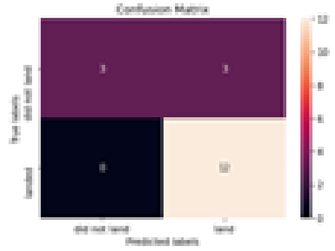
Calculate the accuracy of tree_cv on the test data using the method `score`:

```
[31]: tree_accuracy = tree_cv.score(X_test, Y_test)
      tree_accuracy
```

```
[31]: 0.8333333333333334
```

We can plot the confusion matrix

```
[32]: yhat = svm_cv.predict(X_test)
      plot_confusion_matrix(Y_test,yhat)
```



Results

Predictive analyses

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`.

```
[33]: 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()

[34]: knn_cv = GridSearchCV(KNN, parameters, scoring='accuracy', cv=10)
      knn_cv = knn_cv.fit(X_train, Y_train)


[35]: print("tuned hpyorparameters :(best parameters) ",knn_cv.best_params_)
      print("accuracy :",knn_cv.best_score_)

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

TASK 11

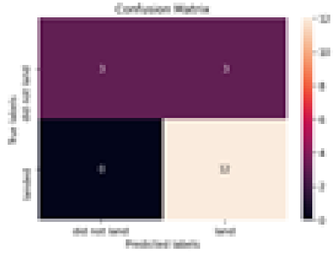
Calculate the accuracy of tree_cv on the test data using the method `score`:

```
[36]: knn_accuracy = knn_cv.score(X_test, Y_test)
      knn_accuracy

[36]: 0.8333333333333334
```

We can plot the confusion matrix

```
[39]: yhat = knn_cv.predict(X_test)
      plot_confusion_matrix(Y_test,yhat)
```



Results

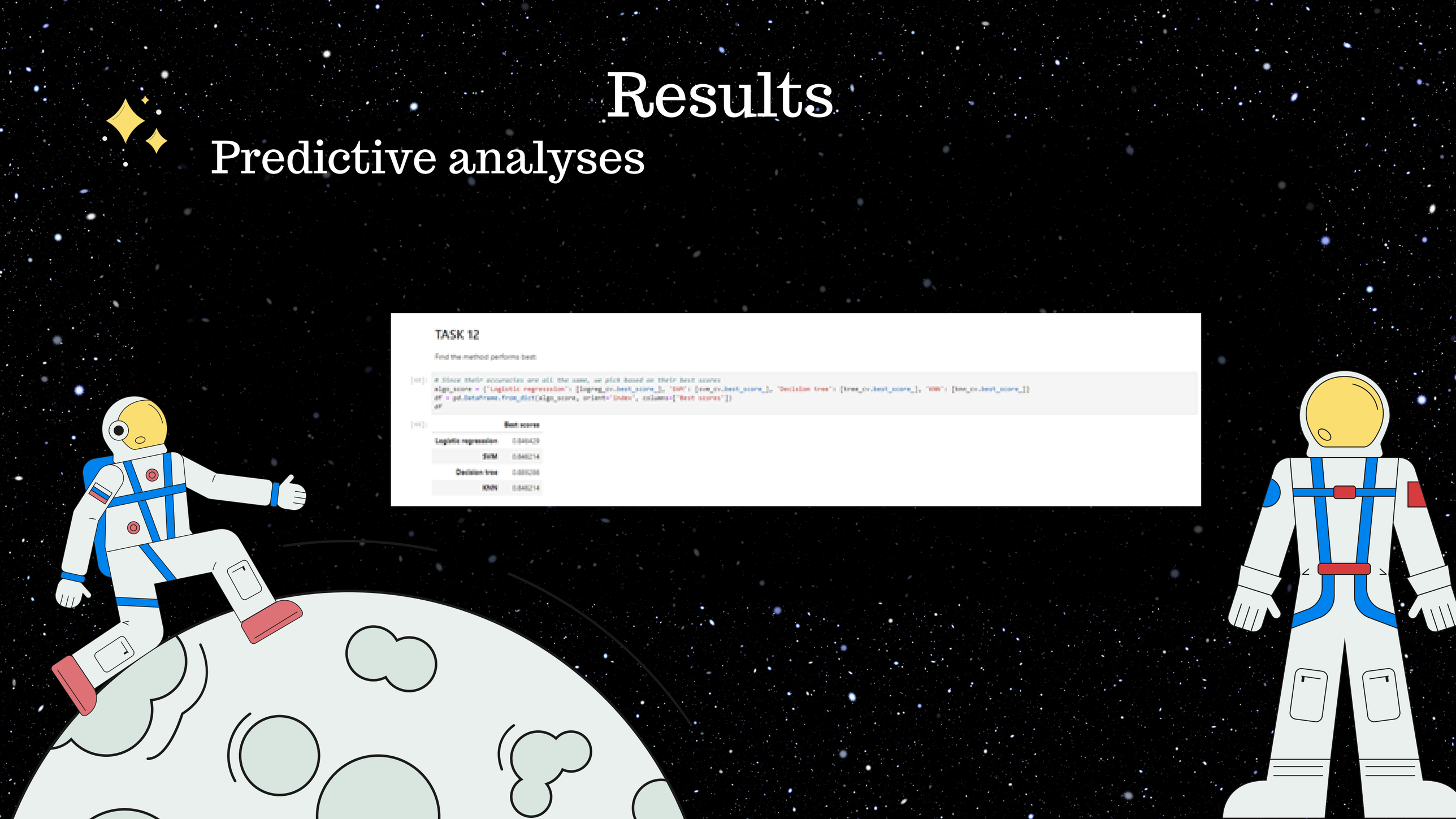
Predictive analyses

TASK 12

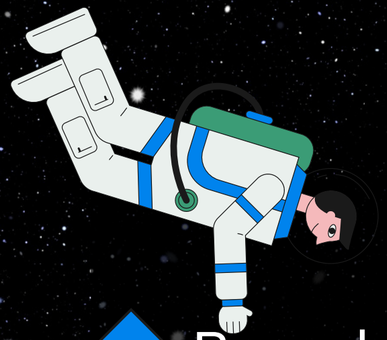
Find the method performs best

```
# Since their accuracies are all the same, we pick based on their best scores
algo_score = {'Logistic regression': [logreg_cv.best_score_], 'SVM': [svm_cv.best_score_], 'Decision tree': [tree_cv.best_score_], 'KNN': [knn_cv.best_score_]}
df = pd.DataFrame.from_dict(algo_score, orient='index', columns=['Best scores'])
df
```

	Best scores
Logistic regression	0.846429
SVM	0.848214
Decision tree	0.885536
KNN	0.848214



Conclusion



Based on collected and wrangled data, we conclude that booster version F9 B5 B1048 - B1060 are the best booster versions for successful landing. Booster version F9 v1.1 B1012 and F9 v1.1 B1015 resulted in landing failures on drone ships. A greater success rate was observed ES-L1, SSo, HEO, and GEO. From the year 2013, significant improvements in success landing rates were observed. However, in 2017, a dip in success rates was observed.